

5

DIGITAL BASEBAND PROCESSOR

10

15

Inventors: Michael Allen
Joern Soerensen
Palle Birk
Jean-Louis Tardieux
Poul Jensen
Morten Nielsen
Mogens Christiansen
Flemming Baltzersen

20

DIGITAL BASEBAND PROCESSOR

The attached documents describe a digital baseband processor.

The documents are entitled as follows:

- 5 (1) New features invented for the Monaco Timing and Event Processor (2) pages,
- (2) Monaco TEP Specification (46 pages),
- (3) DMA Controller Chaining Feature (2 pages),
- (4) Monaco MCU/DSP-DMA Controller Module Specification (48 pages),
- 10 (5) Monaco Clock Control Short Form Technical Note (4 pages),
- (6) Simplified Dynamic Voltage Control Intellectual Property Disclosure (6 pages),
- (7) High Speed Data Logging and Trace Functions (HSL) (5 pages),
- (8) Monaco Patent: PLL-Clock Before Lock Technical Note (6 pages),
- 15 (9) eGSP Patentable Items (8 pages),
- (10) Programmable Power Supply (5 pages),
- (11) Monaco Improving MCU Performance Technical Note (6 pages),
- (12) Cache Support for Real Time Multithreading IP Disclosure (5 pages, and
- (13) Monaco Digital Baseband Processor Design Specification (110 pages).

20 A number of different inventions are disclosed in the documents. Some of those inventions are listed below, but it should be understood that inventions in addition to those listed may be disclosed. In addition, each listed invention may not be limited in scope to the description provided. Where appropriate, references to the documents are included in the list.

- 25 1. Timing Event Processor (multi-mode, multi-thread, ISA, DMA interface, clock calibration). Reference documents: "New Features Invented for the Monaco Timing and Event Processor" and "Monaco TEP Specification".
2. DMA chaining mode and other extensions. Reference documents:
- 30 "DMA Controller Chaining Feature" and "Monaco MCU/DSP-DMA Controller Module Specification".

3. Thread based power management.
4. Automatic H/W-based power mode management (H/W sets power level based on different subsystem requirements). Reference document:
5 “Monaco Clock Control Short Form Technical Note”.
5. The clock to dynamic power level translation hardware scheme.
Reference document: “Simplified Dynamic Voltage Control Intellectual
10 Property Disclosure”.
6. HSL compression algorithms for Frio ISA. Reference document: “High
Speed Data Logging and Trace Functions (HSL)”.
7. PLL fast power up scheme (clocking before locking). Reference
15 document: “Monaco Patent: PLL- Clock Before Lock Technical Note”.
8. Back bias control mechanism. We are also considering individual RAM
subarray power enables, visible to software. Reference document:
20 “Monaco Digital Baseband Processor Design Specification”, section
11.3.3 entitled “Back Bias Controller”.
9. GSP enhancements. Reference document: “eGSP Patentable Items”.
10. EDGE coprocessor IP.
25
11. Off-chip power management control (dynamic voltage management).
Reference document: presentation slides entitled “Programmable Power
Supply”.
- 30 12. Security features (running ID on each chip, pin disabling onboard ROM,
SW validation in HW).

13. Utilizing the FLASH burst mode to speed up ARM processing without using cache. Reference document: "Monaco Improving MCU Performance Technical Note".
- 5 14. Out of order cache line fill support for multithreaded real time system. Reference document: Cache Support For Real Time Multithreading IP Disclosure".

SECRET

New features invented for the Monaco Timing- and Event Processor

The Timing- and Event Processor (TEP) is intended for the Monaco baseband chip supporting several wireless systems concurrently. It offers a superset of the combined uSM and slow-clocking modules known from the previous series of Hercules based baseband chips.

Conceptually TEP handles all time-driven and time-critical tasks such that the Layer1 SW can be purely event-driven, simplifying its design. A functional block diagram of TEP is shown in [1, fig. 2], background description can be found in [1, sec.2.1] and a general listing of the requirements is given in [1, sec. 2.2].

The conceptual new inventions are,

1. A scalable programmable multi-sequencer architecture, consisting of a set of independent sequencers, cf. [1, fig. 7], each executing a stream of instructions corresponding to threads of the Layer1 SW running on top of the baseband chip, cf. [1, sec. 3.1.1]. The direct mapping of threads to independent sequencers lowers the SW load with respect to sorting streams before mapping to sequencers. The scalability of the architecture allows the number of sequencers to be matched by the maximum number of wireless systems to be supported concurrently.
2. A method for proving virtual bus-master privileges of all bus systems of Monaco to TEP via a channel of a DMA module. This is accomplished by letting it overwrite the DMA channel configuration through a dedicated port of the DMA controller, cf. [1, sec. 3.1.3 and fig. 25] without adding actual bus-master logic and bus system to TEP.
3. A method for solving the problem of running more wireless systems with different time-bases concurrently by removing binding to the independent time-bases of the concurrently supported wireless systems by use of a free-running slow clock driving a counter (absolute counter), providing absolute time-values for events to be scheduled in any of the wireless systems being supported at any time, cf. [1, fig. 4].
 - (a) Expressing the absolute time-values as a value of the free-running counter plus a fraction, expressed in system oscillator/-clock (SysClk) cycles, cf. [1, sec. 2.1 and sec. 3.1.2].
 - (b) Calibrating and adjusting the slow clock, driving the absolute counter with respect to temperature- and time drift by periodically measuring the slow clock against the system oscillator, cf. [1, sec. 3.6.1], and adjusting its frequency – when needed - by means of a fractional-N divider, [1, fig. 12], such that an absolute time-value is given as the value of the absolute counter plus the current phase error of the fractional-N divider. The phase error value will be given directly in SysClk cycles by the circuit, cf. [1, sec. 3.1.2 and sec. 3.6.2].

By letting all timing events be scheduled with respect to the absolute timer and a fraction of SysClk cycles all binding to the different time-bases of the various concurrent wireless systems vanishes. Further, the frequencies of the slow clock, SysClk respectively, can be selected freely.

4. A method for directly determining when to power down system oscillator and -PLL. Whenever neither of the modules (processors, etc.) of Monaco requires clock signals, the system oscillator may be powered down to save power. Since TEP controls all timed events

and since this timing is based on the absolute counter, TEP can exactly determine and perform when to power up/down the oscillator without SW intervention, except from supplying information about settling time for the oscillator once, cf. [1, sec. 3.5; fig. 4 and fig. 9].

Reference:

[1] Monaco TEP specification. August 6, 2001. Attached.

a

Monaco TEP specification

File name: Monaco TEP specification.rev0.9.doc

Status: Draft

Author: Poul Rands Jensen, ADI-DK

Revision: 0.9

When	Who	What
Dec. 6, 2000	PRJ	Document created
Feb. 7, 2001	PRJ	Revision in progress
Feb. 26, 2001	PRJ	Revision complete – more or less – for review and feedbacks.
Apr. 27, 2001	PRJ	Total rewrote the specification. Introduced the use of 32 kHz clocking including calibration and absolute timing specification. Integrated slow-clock and power control features. Added close interface to DSP DMA controller. Revised thread hardware concept to a series of independently executing Sequencers. Revised instruction-set.
May 4, 2001	PRJ	Added spec for bus interface (partly). Made 24 bits counter readable. Added new instruction to Sequencers. Added /N

a

May 18, 2001	PRJ	<p>clock option. Added spec of memory size, interface and the memory access resolver. Added comment on postponement of instructions handling. Added spec. for HSL interface. Various other minor changes.</p> <p>Updated document according to in-house review.</p>
		<p>Increased #Sequencers to $2S+1 = 5$. Added independent time-ticks for each Sequencer. Extended snapshot registers to 24/16 bits.</p>
		<p>Removed DataMoveI instruction and added NOP instruction. Made GPIOB asynch. Again. Changed GPIOA to be synchronous to GPSigA[2] rather than time-tick. Made all GPIO's registers within TEP. Changed execution of Set/Clr GPIOA not to block execution.</p>
June 27, 2001	PRJ	<p>Added more details about the DPBUS interface – now PBUS compatible. Changed TEP bus interface to be only 16 bits (ups). Added interrupt signal used for flagging DMA channel for transferring data to/from TEP without blocking DPBUS by wait-state insertions.</p>
July 12, 2001	PRJ	<p>Made a few drawings update and textural corrections. Removed LongSetup instruction.</p> <p>Changed free-running counter from 24 bits to 26 bits (SW-L1 request). Extended TCCFR integer part to 6 bits.</p>
July 31, 2001	PRJ	<p>Added section 5.3 about the software recommendations.</p> <p>Other minor corrections. Added TCR[28] for forcing PLL powered up. Put out as rev. 0.8</p> <p>Corrected description of OWUR and CPPUR registers. Corrected the warm-up timer of Figure 9 to 10 bits.</p>
		<p>Corrected the expressions for TCCMR and TCCFR and fractional-N clock division</p>
August 6, 2001	PRJ	<p>Added comment about reading of TACR register. A few other textural changes.</p> <p>Corrected description of TPSR register.</p> <p>Updated DspDMACH0Cfg width to 86 bits to match DSP DMA controller specs. Put out as rev. 0.9.</p>

Table of contents

1.	Introduction	5
1.1	Overview	5
1.2	Purpose.....	6
1.3	References.....	6
1.4	Terminology.....	6
2.	Requirements	7
2.1	Background	7
2.2	TEP Feature Requirements	8
2.3	Additional Requirements	9
3.	TEP module specification.....	9
3.1	Functional Description.....	9
3.1.1	Scenario	10
3.1.2	Clock Calibration.....	11
3.1.3	SysDMA Usage	12
3.2	Hercules μ SM Backward-Compatibility.....	12
3.3	TEP high-level Architecture	13
3.4	Absolute Counter	14
3.5	Clock & Power Control	14
3.6	Clock Calibration	16
3.6.1	System Clock Snapshot	16
3.6.2	Frequency adjustment.....	16
3.6.3	'Fail-Safe' /N generation of CClk32k	18
3.7	Sequencer Specification.....	18
3.7.1	Instruction-set	18
3.7.2	Sequencer Architecture.....	20
3.7.3	Clock Pre-Scaler	21
3.7.4	Sequencer Data-Path.....	21
3.7.5	Sequencer Control-Path.....	22
3.7.6	Pipelining and Timing	23
3.7.7	Register File.....	26
3.8	I/O Conflict Resolver.....	27
3.9	Memory Access Resolver and SRAM	27
3.9.1	Memory Structure.....	27
3.9.2	Memory Access Resolver	28
3.10	DPBUS Bus Interface.....	29
3.11	TEP - DSP DMA interface.....	31
3.12	Error Handling.....	33
3.13	Debug Features.....	33
3.14	Clock Domains and Inter-Synchronization techniques	33
4.	HW Interface Description.....	34
4.1	Clock, reset and system interface.....	34
4.2	DPBUS (DPBUS) interface	35
4.3	DSP DMA Controller interface	36
4.4	System DMA Controller interface	36
4.5	MCU and DSP Interrupt Controllers interface	36
4.6	Internal General Purpose Signals interface.....	37
4.7	MMI GPIO Signals	37
4.8	Scan-test interface.....	37

4.9	Monitor Port interface.....	37
4.10	High-Speed Logger interface	38
5.	SW Interface Description	38
5.1	Memory Map.....	38
5.2	Register Description	40
5.2.1	OWUR – Oscillator Warm-Up Register.....	40
5.2.2	CPPUR – Clock Pad Power Up Register.....	40
5.2.3	TCLR – TEP Clock Calibration Length Register.....	40
5.2.4	TCCFR – TEP Clock Calibration Fraction Register	40
5.2.5	TCCMR – TEP Clock Calibration Modulus Register	40
5.2.6	TESR – TEP Exception Status Register	41
5.2.7	iTESR – Interruptable TEP Exception Status Register	41
5.2.8	TEIMR – TEP Exception Interrupt Mask Register	41
5.2.9	TCR – TEP Control Register.....	41
5.2.10	TACR – TEP Absolute Counter Register.....	42
5.2.11	TCRBR – TEP Calibration Read-Back Register	42
5.2.12	TSSR – TEP Single-Step trigger Register	42
5.2.13	Sequencer Register-Files	42
5.3	Software Recommendations and Scenarios	43
5.3.1	Accessing the module	43
5.3.2	Configuration.....	44

1. Introduction

1.1 Overview

The Monaco chip is the scheduled 2.5 G supporting baseband for mobile. It must support GSM, GPRS as well as EDGE. Besides, Monaco will serve as development vehicle for 3G together with later designed external add-ons, e.g. a RAKE receiver. For this reason Monaco must be prepared for this. Particularly this includes a special co-processor interface. Another topic is the support for multi-mode systems by the timing module - known as the *Timing- and Event Processor* (TEP) for Monaco.

Monaco will be the first wireless ADI baseband to incorporate the newly developed FRIO processor core. A simplified block diagram of Monaco is shown of Figure 1.

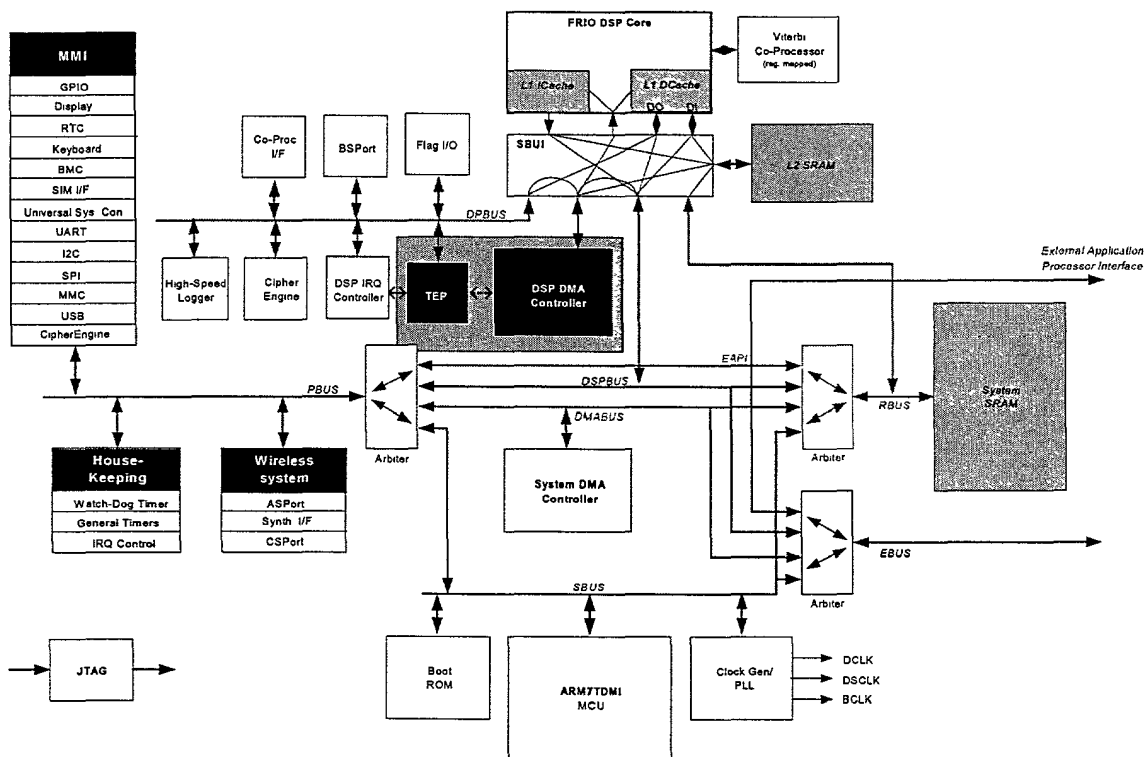


Figure 1: Simplified block diagram of Monaco including the FRIO processor core, cf. [3] for details.

Specifically, all events within the Monaco hardware and Layer 1 software requiring timing precise execution, like generating time-accurate trigger signals for other hardware modules and controlling external pins, e.g. like RxOn for radio unit power up/down or interruption of the processors at specific times for triggering execution of software tasks will be handled by *Timing- and Event Processor*, as this will be the only module knowledgeable about actual time. Another task is transfer of programming data to e.g. RF synthesizer programming and setup of DMA channels for baseband data reception. The latter task will be conducted together with the DSP DMA controller.

1.2 Purpose

This document contains the requirements and module specifications of the TEP module. The TEP is a simple programmable multi-sequencer processor for handling *timing-accurate and -driven events* of the wireless system.

1.3 References

- [1] MicroSM Module Specification. Rev. 1.03. September 29, 1998.
- [2] Monaco Recommendations for Ciphering and Timing modules. Rev 0.4. November 24, 2000.
- [3] Monaco Digital Baseband Processor Design Specification. Rev. 0.0. April 5, 2001.
- [4] MSP430 Interrupt Controller. Rev. 2.3. November 26, 1998.
- [5] Hercules Slow Clocking Block - Design Specification. Rev. 1.0. September 4, 1998.
- [6] HDL for Synthesis Design Methodology. Rev. 1.0. July 8, 1999.
- [7] Hermes DFT modifications. Rev. 1.0. November 16, 2000.
- [8] Coding for predictable implementation. Rev. 1.0. 2001.
- [9] Multi System Mechanism. Rev. 0.7. Feb. 15, 2001.
- [10] AMBA Specification. Rev. 2.0. ARM IHI 0011A. 1999.
- [11] Working notes: Description of synchronous part of WIOS. Version 00.02. April 2001.
- [12] Monaco DSP DMA Controller Specification. [TBD]
- [13] Monaco System DMA Controller Specification. [TBD]
- [14] Monaco DSP Peripheral BUS (DPBUS) Specification. Rev. 0.0. May 3, 2001
- [15] Monaco Peripheral Bus System Specification. Rev. 0.1. May 10, 2001.

1.4 Terminology

MCU	The 32 bits RISC from ARM, ARM7TDMI core.
DSP	The new ADI FRIO DSP core
μSM	Short for micro state machine. The Hercules/Hermes system-timing module, cf. [1].
PBUS	The Peripheral Bus system of Hercules/Monaco [14]
DPBUS	The Monaco DSP peripheral system bus [15]
Asport	Monaco Audio serial port.
Bsport	Monaco Baseband serial port.
Csport	Monaco Control serial port.
Qbit	Referring to a quarter of a bit period in GSM.
TEP	The Timing- and Event Processor. The Monaco system-timing module

2. Requirements

2.1 Background

Solutions/terminals for single-mode wireless systems like GSM, the baseband keeps synchronized to the wireless system clock and may continuously adjust to stay so. Every action on the wireless channel conducted by the terminal are scheduled within the a certain precision, i.e. 1 Qbit for GSM, and to keep track of time and schedule events within such precision a dedicated hardware timer module for generate timed interrupt of processors, initiate other modules, generate control signals for off-chip circuits, like radio unit, is required. An example of such is the μ SM module from the Hercules/Hermes chipsets [1].

For multi-mode solutions, like GSM/EDGE/3G, similar requirements regarding timing precision do exist, but in each wireless system with independent time bases. The baseband may be required to keep track of two or more wireless systems simultaneously and since there is only one reference oscillator, it will not be possible to do lasting AFC based on wireless system inputs.

Rather the Monaco system design approach will be to apply a free-running oscillator and then by software project the time-bases for the various simultaneous supported wireless systems onto this free-running clock. Provided this clock is stable, its frequency is irrelevant and its resolution need not be very high. Hence an independent 32 kHz counter will be used. Once this clock is synchronized to the wireless system by *software means*, any time can be given as an *absolute time-value* of this counter plus a number of a higher frequency clock, which provide a fraction of the 32 kHz clock. *This approach removes bindings to any wireless system. Additional benefits are built-in low power and the freedom in selecting the high frequency system oscillator.*

A standard 32 kHz oscillator plus XTAL do not offer sufficient stability over temperature and time. Hence, measures to calibrate the free-running oscillator must be included in the system as well.

The Layer 1 architecture is based on multi-threading with typically several threads for each wireless system handling various sets of tasks, cf. [11]. The threads are, in the software, mapped down to 'Resource Queues' – one for each set of resources being shared between the wireless systems – each containing the synchronous tasks to be conducted on that set of resources. Wireless systems not sharing resources, like GSM and Bluetooth, with separate radio resources will be mapped onto different resource queues. A resource queue will be mapped into *A/B-buffer-like* streams. These instructions streams must each be mapped independently onto the Timing- and Event Processor for *timing exact* execution. This calls for a *multi-sequencing hardware architecture* of the TEP as well, which - of course - for reasons of flexibility must be fully programmable, executing several streams of instructions concurrently – one for each thread.

A conceptual idea of the TEP and Layer1 architectures [9,11] is that TEP must handle all time-driven and time-critical tasks. This implies the Layer1 can be fully event-driven [11]. The TEP must interface to all hardware modules requiring timing precise actions. This is illustrated of the functional diagram at Figure 2.

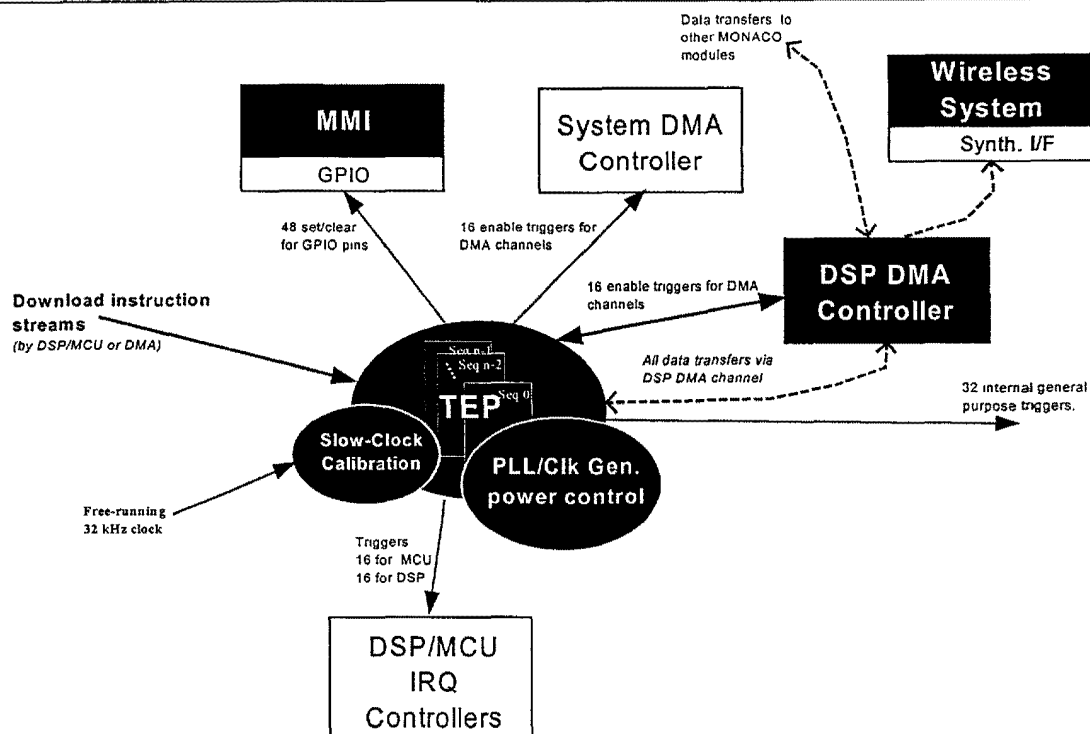


Figure 2: Simplified functional diagram for the Monaco Timing- and Event Processor. The PLL/Clk Gen. Power Control and slow-clock calibration modules are integrated parts of the TEP module.

The interface to the MMI/GPIO are setting pins for controlling external devices, like radio unit power up/down, RxOn, etc. Interfaces to System DMA, DSP/MCU IRQ controllers, are for enabling *time-specific DMA transfers* and generating *timed interrupts* to the processors, respectively. The interface to the DSP DMA controller may also do for enabling channels at specific times. However, this DMA controller is tightly coupled to the TEP and will be used by the TEP to implement various sorts of data transfers, like writing new programming words to the synthesizer I/F or setting up another DMA channel for e.g. burst reception. One channel will be dedicated for this purpose and one additional DMA channel for each Sequencer will supposedly be allocated (statically).

2.2 TEP Feature Requirements

The TEP module must support the following features,

- ❑ The TEP being the only module keeping track of actual time, hence must handle/schedule all events within the Monaco hardware and Layer 1 software requiring timing precise execution, like,
 - Generating time-accurate trigger signals for other hardware modules and
 - Controlling external pins, e.g. like RxOn for radio unit power up/down or
 - Interruption of the processors at specific times for triggering execution of software
- ❑ Another task is transfer of programming data to e.g. RF synthesizer programming and setup of DMA channels for baseband data reception. These tasks must be conducted utilizing the DSP DMA controller.
- ❑ Relaxed software load of processors (DSP and MCU) for servicing the module compared to Hercules and enable these to do event-driven processing. This must be achieved through,

- Introducing the ability to specify events in *absolute time* – using the 32 kHz clock as input for the *absolute timebase*.
 - Providing a dedicated instruction-set allowing dedicated yet flexible programming of the TEP.
 - Providing conditional execution of instructions based on evaluating any bit within the TEP RAM space.
 - Providing extendable hardware architecture with hardware sequencers supporting the instruction-set and numbered to match the number of concurrent software threads.
- Support multi-mode wireless systems *concurrently* with different time-bases, like slotted GSM and 3G modes, where the different systems call for different time-base. Together with Layer 1 handling this will be achieved using the 32 kHz time-base and the SysClk for fine-grain.
 - Provide the necessary timing stability by means of,
 - Continuous calibration of the 32 kHz clock used for absolute time base against the more temperature-stable system clock, SysClk.
 - Fine-grain delays must be measured relative to the 32 kHz timebase using the 13 (or 26) MHz SysClk.
 - Provide a *fail-safe alternative* to the 32 kHz XTAL oscillator based clocking by enable the use of an integer-divided SysClk as slow clock - with seamless switching between the two sources.
 - Provide *automatic SysClk oscillator and system PLL power-control*. Whenever the chip system clock, SysClk, or system PLL are not required they must be powered off. When foreseen the oscillator can be powered off for a long time (compared to the 13 MHz oscillator warm up period), the TEP must power down as well as power up the oscillator (coordinated with other modules like UARTs, etc which may be needing the clock).

2.3 Additional Requirements

- The TEP will be running all the time, including during *wireless idle* mode. This makes the achievable standby current measure quite dependent on the power consumption of the TEP module. Hence, emphasis must be paid to apply *low-power design techniques* for the module.
- The design must comply with the Verilog-based design-flow and -style, cf. [6,8], as well as the DFT requirements, cf. [7,8], applied by ADI for the Monaco project.

3. TEP module specification

This section defines the detailed specifications for TEP module, including interfaces.

3.1 Functional Description

The TEP module encapsulates the features offered by the μ SM and slow clocking modules from Hercules [1]. These features boils down to re/setting I/O pins, generating interrupts to processors and enabling DMA memory-to-memory transfers at specific instances of time plus controlling pins for powering up/down system oscillator PLLs, e.g. slow clocking, and SysClk/32kHz calibration. All of this is handled in similar fashion by TEP.

3.1.1 Scenario

A conceptual new idea behind TEP and Layer1 architectures [9,11] is that TEP must handle all time-driven and time-critical tasks, and hence, allowing the Layer1 to be fully event-driven [11]. In Figure 3 is shown the interface between Layer 1 and TEP, cf. [11]. The system is, in general, multi-mode with *concurrent* handling of modes. The 'Resource A' is a resource request queue for one or more of the wireless systems currently running and sharing resources. The resource requests are translated by the 'Executer' to instruction streams for the TEP hardware and passed to it – all pure event-driven. Whenever, the request (say receiving paging info) has been granted the TEP may generate an interrupt – by means of an instruction – to the host processor for flagging Layer 1.

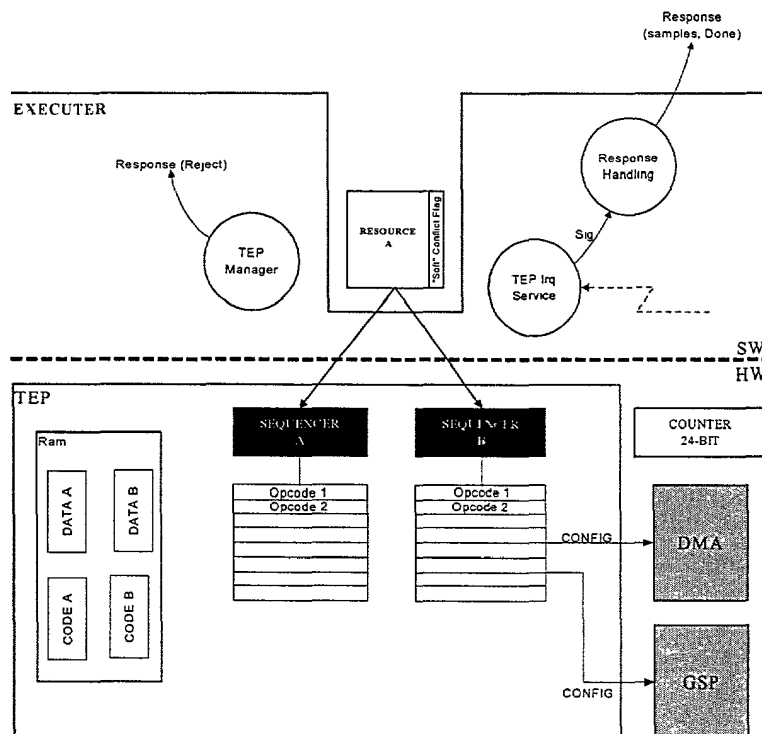


Figure 3: Illustration of the interfacing between the Layer 1 'Executer' block and the TEP module, cf. [11] for details.

For the example with the paging info reception the TEP would need to set up the serial BSPORT as well as a DMA channel for data storing. This is part of the instruction stream passed to the TEP as well as other tasks like programming synthesizers, reception gain, etc. The instruction stream - or program actually – may be run many times with only slight modifications like altering programming data to synthesizer. Hence, there must be separable memory areas for data and code allowing changing data without altering instructions, as well as control for rerunning code.

Execution of instruction streams is carried out in hardware sequencers and there are as many sequencers as threads concurrently supported by the TEP module. The Layer 1 software architecture requires two sequencers per wireless system [11]. The timing-precise features flow from the execution of instruction. TEP uses three clocks for offering these features, the conventional system clock, SysClk, a SysClk divided clock, time-tick, and a slow 32kHz. Due to the multi-mode scenario neither of these clocks can correspond to the (several) wireless system time bases. *Hence, for obvious reasons of power consumption and the nice integration of the slow-clocking feature, the 32 kHz will*

be applied as the overall timebase. This totally removes the time-base/clock switching between the applied system clock and a free-running 32kHz, which reduces software efforts and fits the concept of purely event-driven Layer 1 software execution. In Figure 4 is shown a timing example of the TEP execution.

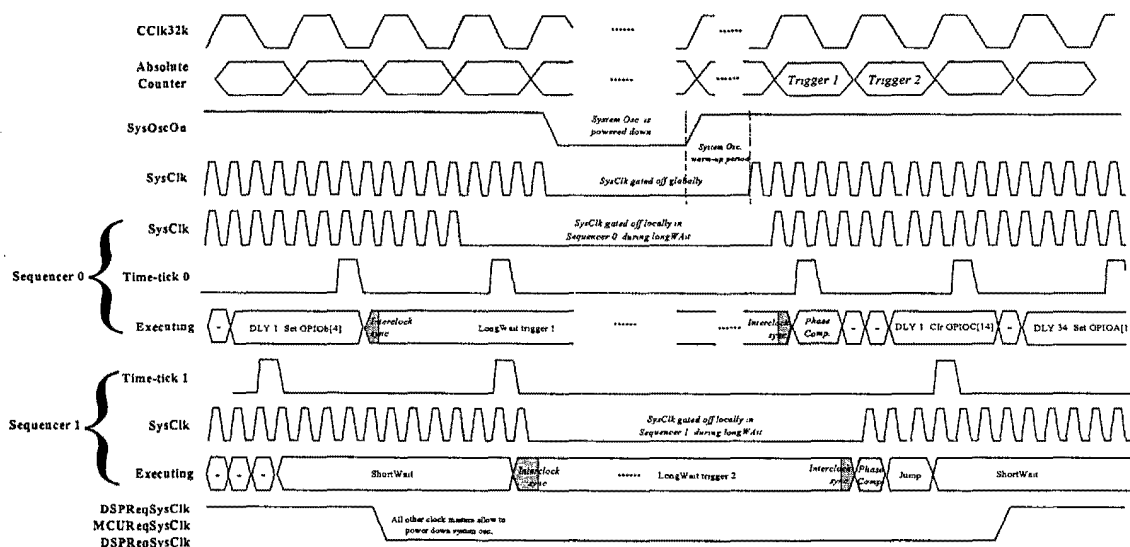


Figure 4: Exemplified timing scenario of the TEP for the case of two Sequencers. Instructions with zero delta-time are just executed sequentially; delta-times are calculated using the local time-tick. Notice the clock control is determined by TEP without software intervention.

3.1.2 Clock Calibration

The 32 kHz clock is free-running, which means software must do a mapping between this free-running counter and times of a wireless system as well as continuously compensate (i.e. synchronize) during operation. This requires frequency stability of the 32 kHz clock better than normally offered by standard oscillator solutions. Especially temperature sensitivity will pose a problem. Therefore some sort of calibration is required. Whenever running the SysClk will be compensated regarding temperature and in general be much more frequency stable than the 32 kHz clock. Hence the 32 kHz clock can be measured during such periods towards the SysClk - simply by counting the number of SysClk cycles in a fixed number of 32 kHz cycles. Redoing this periodically gains information for calibrating the 32 kHz clock. The adjusting technique is illustrated of Figure 5. Adjustment will always be to a lower frequency which is implemented by removing one clock cycle every given period. This adjusts to $(\text{period}-1)/\text{period}$ of the input clock.

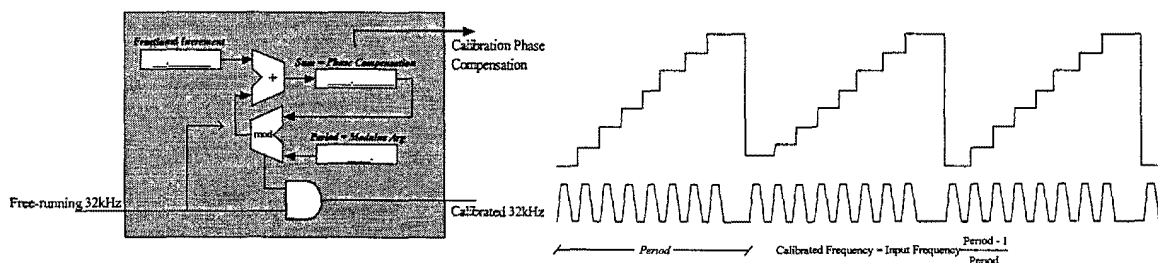


Figure 5: Calibration adjustment of the free-running 32 kHz clock by fractional-N division. The case of down adjusting by $(9-1)/9$ is exemplified to the right.

However, it also introduces phase errors, making it impossible to hit the correct time of each clock cycle unless phase compensated. The phase compensation increases linearly over the period, reaching the length of one clock cycle at the end of the period. This is computed *on-the-fly* with the circuit of Figure 5. The phase compensation will be implemented as an additional delay of SysClk cycles associated with the 'LongWait' instruction. By proper selection of the 'Fractional Increment' and the 'Period' the computed phase compensation will be expressed in SysClk directly.

3.1.3 SysDMA Usage

The TEP must be able to setup DMA channels, program synthesizers, etc. This will be featured by use of one fixed allocated DMA channel of the DSP DMA controller as shown in Figure 6. The DSP DMA controller offers sufficient channels to allow such fixed allocation, and beneficially, use of a DMA channel avoids adding a bus master function to the TEP as well as avoiding adding an additional master port to the arbiters.

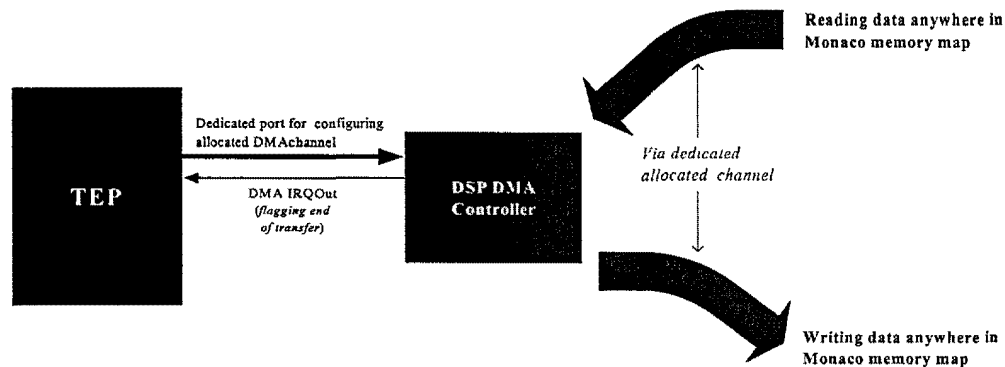


Figure 6: Block diagram showing the use of dedicated DMA channel for data read/write access anywhere in Monaco address space, including setting up other DMA channels.

Depending on the addresses to access the latency time for the access varies, due to variable latency in arbiters, slower clocked buses, etc. However, the *worst-case latency* for each arbiter can be calculated *a priori* and taken into account in writing the TEP code.

3.2 Hercules μ SM Backward-Compatibility

The TEP will not offer code compatibility compared to the μ SM. However, functional wise the TEP may be thought of as a series of hardware Sequencers each similar to the μ SM except for some extensions, like the 32kHz clocked delay, DMA integration and conditional execution. Hence, by not using the TEP extensions and limiting the number of Sequencers to one will offer a μ SM-like behavior. Such scenario does support the same logical set of instructions and interfaces to Slow Clocking, GPIO, etc. Only serious obstacle is the lack of a FIFO feature in the memory and the associated exception flagging. This can be implemented in software – on the host side by software-wise compute a circular address behavior (like a FIFO) – and on the TEP side by use of 'jump' instructions to implement the required circular behavior.

This means conceptually, that existing software design from Hercules/Hermes can be made run-able on Monaco with TEP by means of software porting efforts.

3.3 TEP high-level Architecture

A simplified view of the TEP architecture is shown in Figure 7 as a block diagram. Basically it consists of a series of sequencers, each for executing one thread, a multi-ported memory for instruction streams and data, interfaces to the DSP DPBUS bus [10] and the DSP DMA controller, some common clock and control and various logic for inter-thread conflict resolving.

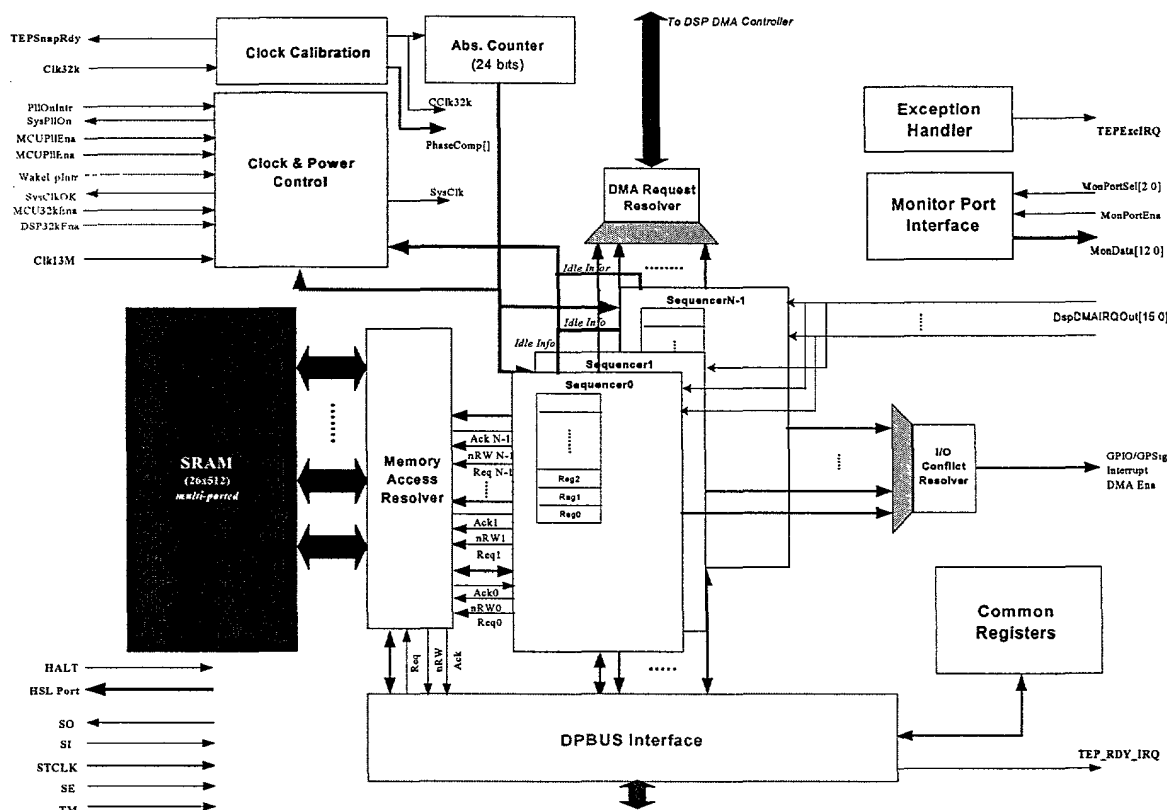


Figure 7: Simplified *block diagram* of the TEP architecture.

The ‘Clock & Power Control’ block includes functionally the oscillator power control of the Hercules Slow-Clocking module. The ‘Clock Calibration’ block implements 32kHz to SysClk calibration feature functional compatible to the Hercules Slow-Clocking module [5].

The architecture is targeted as easily parameterized with respect to the number of Sequencers and the amount of memory ports. Simply, this is trade-off between the number of concurrent threads, which determines the number of simultaneous wireless systems, and gate-count. The table in below summarizes the *selected* values for the TEP implementation.

Parameter Description	Parameter	
	Symbol	Value
<i>Number of simultaneous wireless systems</i>	S	2
<i>Number of hardware Sequencers</i>	$N = 2 \cdot S + 1$	5
<i>Number of memory read-ports</i>	RP (= S)	2
<i>Number of memory write-ports</i>	WP ($\leq$ RP)	1
<i>Memory size (words)</i>	M	512

Better estimates of RP, WP and M may be attained through profiling representative thread streams/scenarios from Layer 1 design, regarding *simultaneous memory access bandwidth* and *stream lengths* – once available. [TBD]

3.4 Absolute Counter

The absolute counter is an ever-running 26 bits counter clocked by the calibrated 32 kHz clock. A 24 bits comparator allows to setup an absolute time trigger. This gives an absolute time period of app. 8½ min. - beyond that must be handled by Layer 1 - e.g. be using the 2 MSBs of the counter to identify *time period swapping*.

For general purpose usage the counter includes two periodic trigger generators. These may be used for processor interrupts or initiation of a snapshot, i.e. measuring the free-running 32 kHz clock towards SysClk.

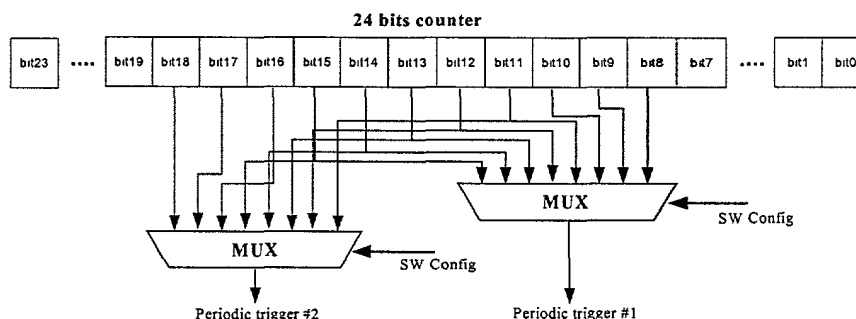


Figure 8: Simplified view of the absolute 26 bit counter with two periodic trigger generators.

The counter must be readable by the host processor via the DPBUS interface. A reading must be guaranteed valid, i.e. synchronized to times safely outside counter updates, e.g. at neg. edge of CClk32k.

3.5 Clock & Power Control

Clocks to the various parts of TEP must be gated off automatically when the part is not in use. Part of these will be done locally, e.g. a Sequencer will gate off itself when idle. However, due to the independent behavior of the Sequencers, the Power control module is required to control gating off clocks at the top-level of TEP.

If one or more Sequencers are executing 'LongWait' instruction of a certain length, the Power Control module must determine if the duration is sufficient to allow power down of the SysClk oscillator or not. If other on-chip modules do not need it either, the module must power off the oscillator.

The general power up/down control of the system oscillator, PLL and clocks is shown of Figure 9. Basically, there exist seven masters regarding clock powering. That is, MCU, DSP, Sys and DSP DMA, TEP Sequencers, TEP clock calibration measure and wake-up interruption. Each of these can re-power the system oscillator and PLL and likewise allow powering off the oscillator and PLL, by means of setting or clearing associated bits of local registers.

The warm up time for the oscillator, settling time for clock pad-buffer, etc. are specifiable via the **OWUR** and **CPPUR** registers, respectively and handled by the 'Power-Up Sequencer' sub-block, consisting of a 10 bit counter, some 10 bits comparators and a FSM. Once triggered the FSM starts the 10 bits counter from zero - and each time a comparator hits the associated signal, e.g. ClkBufOn, is asserted. The signals remain set until the input deasserts.

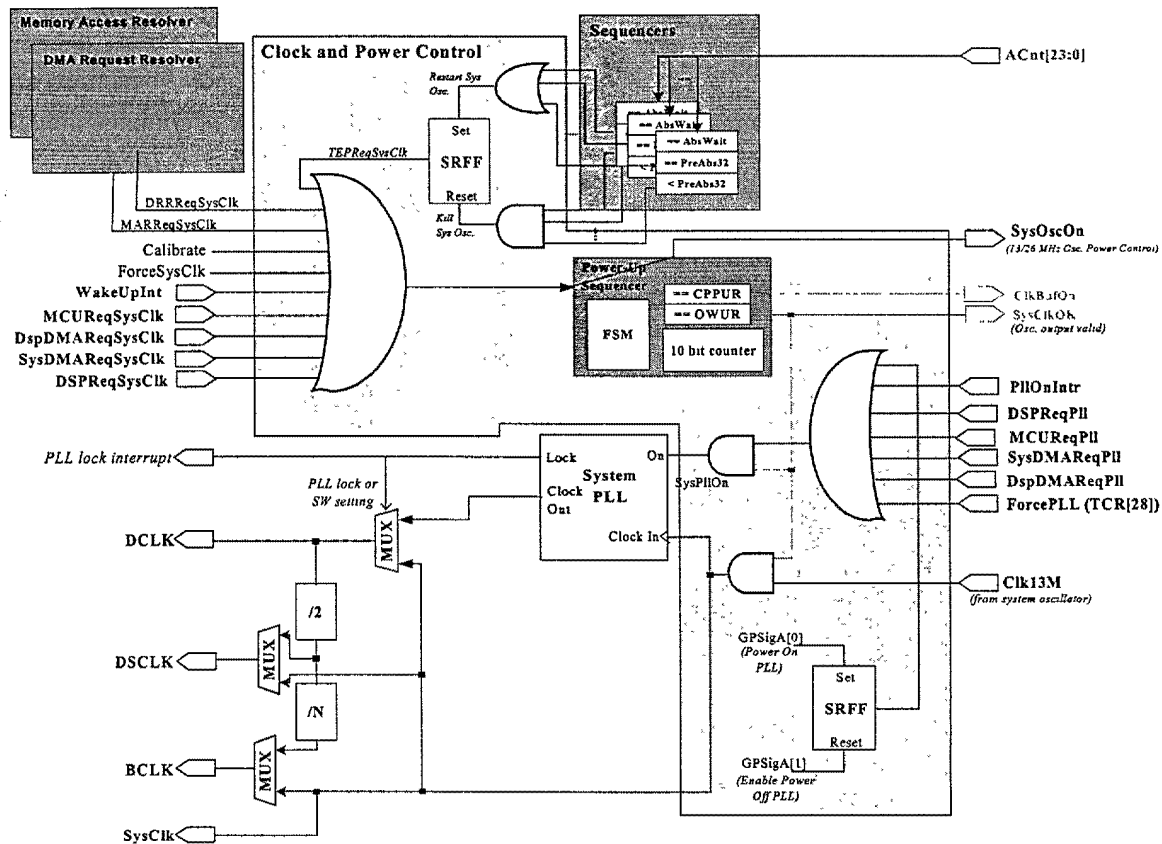


Figure 9: Functional schematic of the Power Control features handled by TEP. The schematic does not show real circuit implementation details – only functionality.

For TEP triggered re-powering the warm-up sequencer period will be triggered by the timing evaluation internal the Sequencers. That is, if *current time* < *PreAbs32*, it is allowed to power off the oscillator. When *current time* = *PreAbs32* the oscillator must be re-powered and when *current time* = *LongWait expire time*, the system clock must have stabilized. Hence, *PreAbs32* must be set equal *LongWait expire time* – *OWUR*.¹

The PLL cannot be powered unless *SysClkOk* has been asserted. For debug the auto-power down feature must be disable through software (via the **TCR** register).

¹ This is done automatically by the ‘LongWait’ instruction in the instruction stream for each Sequencer.

3.6 Clock Calibration

The Clock Calibration block includes the snapshot feature as well as the frequency adjuster and phase compensation calculation.

3.6.1 System Clock Snapshot

The snapshot feature simply counts the number of SysClk cycles over a number of the free-running 32 kHz clock cycles. This is shown of Figure 10. The snapshot can be initiated by setting SeqCtrl[6] of any of the Sequencers or by the two periodic triggers of the absolute timer. Each of these can be dis/enabled by the host processor via the TCR register.

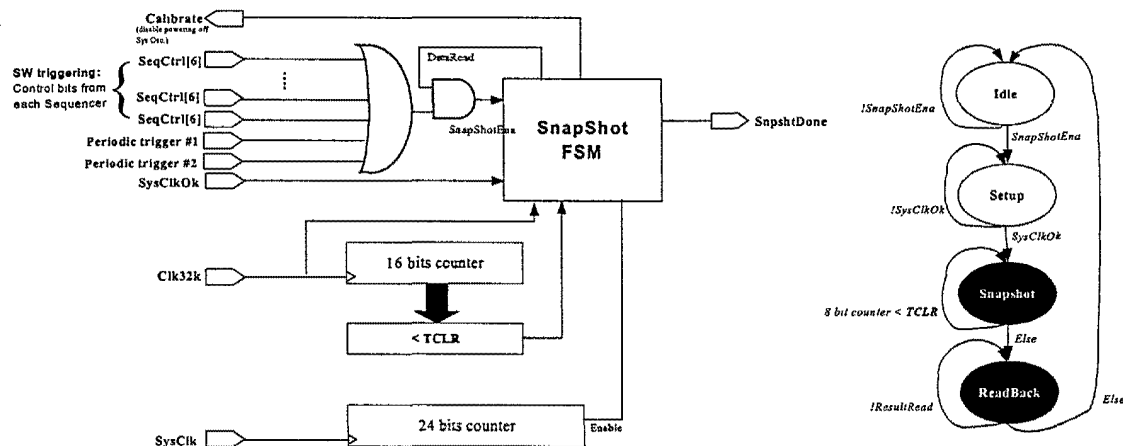


Figure 10: Functional block diagram and state transition diagram for the snapshot feature.

Once enabled the Calibration signal will be asserted, which will disable powering off SysClk oscillator. The FSM enters 'Setup' state, which simply waits for SysClkOk asserts (in case SysClk oscillator had been powered off) after which it enters 'Snapshot' state. The number of SysClk cycles is counted during a number of 32 kHz cycles given by the SW configurable TCLR register, after which an interrupt is generated and the FSM returns to 'ReadBack' state. The 24 bits counter is readable by the host processor via the DPBUS interface while the FSM is in 'ReadBack' state. Once read the FSM returns to 'Idle' state and a new snapshot may start.²

3.6.2 Frequency adjustment

The frequency adjuster circuit known as fractional-N clock divider is detailed in Figure 12. The registers for modulus and fraction values are double-buffered; partly for not slowing down the write access to these registers and partly to ensure updating the values will take place at appropriate time as specified of Figure 11. Data will be clocked into TCCFR and TCCMR registers by SysClk, and hence, must be synchronized to the CClk32k clocked Fraction and Modulus buffers; double edge sampling technique must be used.

The software configurable values must be set so that the calculated phase compensation corresponds to the number of SysClk cycles of additional delay. This is used by the 'LongWait' instruction, which after the absolute time has been reached adds the delay in SysClk cycles, using the delta-timer. This means that the backend of a 'LongWait' instruction always occurs at the exact calibrated time. TCCMR must equal the truncated number of SysClk cycles within one CClk32k cycle,

² Hence, the snapshot result **must** be read always or the FSM remains in 'ReadBack' state.

$$TCCMR = \left\lfloor \frac{SysClk}{Clk32k} \right\rfloor,$$

in order to have PhaseComp[9:0] expressing phase compensation in SysClk cycles directly. Further, given,

$$K = \frac{Clk32k}{CClk32k} > 1,$$

with K being the clock division factor, this relation is expressed in terms of the registers as,

$$K = \frac{TCCMR / TCCFR + 1}{TCCMR / TCCFR}$$

Combining the two, this leads to an expression for the value of TCCFR as,

$$TCCFR = \left(\frac{Clk32k}{CClk32k} - 1 \right) \cdot TCCMR$$

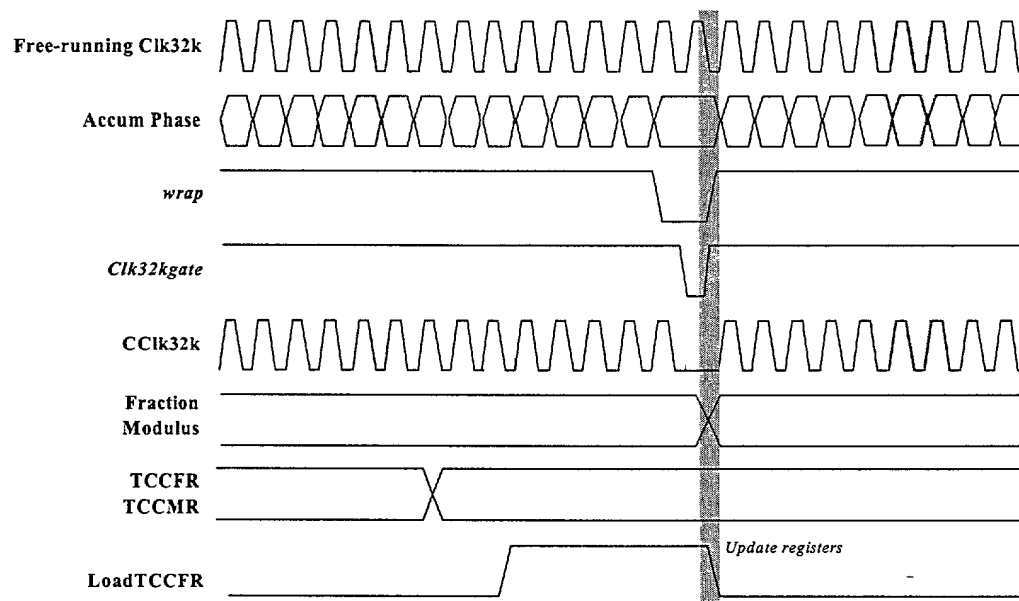


Figure 11: Fractional-N clock divider operation timing. The gray zone indicates the update of modulus/fractional values.

The size of adders and registers are determined from software requirement regarding timing resolution/precision, cf. [11], and the fact that the PhaseComp[9:0] must express the additional delay in SysClk cycles directly.

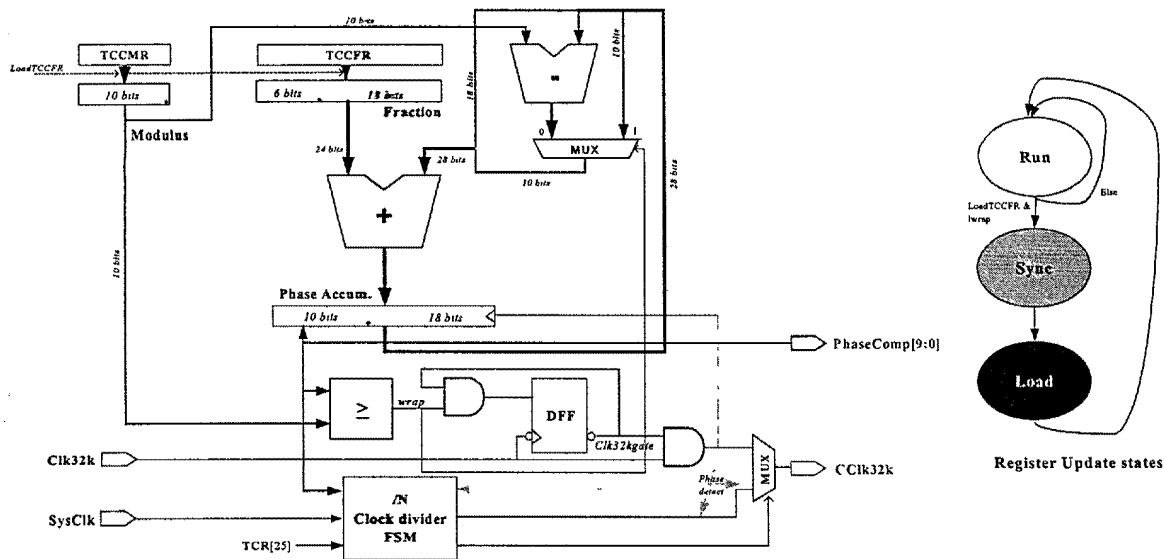


Figure 12: Block diagram for the fractional clock divider and functional state transition diagram for handling update of fractional and modulus values. The FSM is clocked at neg. edge of Clk32k. The /N of the SysClk is contingency in cases the Clk32k cannot be calibrated sufficiently.

3.6.3 'Fail-Safe' /N generation of CClk32k

In case the Clk32k cannot be calibrated to achieve sufficient frequency stability, e.g. due to rapid temperature change or poor XTAL performance, etc. the /N of SysClk can be used. By using the Fractional-N divider to achieve $CClk32k = SysClk/N$ a phase-preserving clock switch can be done in the following manner.

When shifting from CClk32k to SysClk/N, '/N Clock Divider FSM' detect the next rising edge of CClk32k and resets its divider clock and additionally keeps it reset for a number of SysClk cycles equal to the CompPhase[9:0] value before making the rising edge of its clock output. The actual multiplexing when takes place at the following cycle.

When shifting the other way, the CClk32k will in general be out of phase and needs to be phase aligned to SysClk/N not to lose timing. Before doing the switching CClk32k must have been frequency adjusted to SysClk/N. The '/N Clock Divider FSM' measure the phase difference between CClk32k and SysClk/N by counting the number of SysClk cycles from the rising edge of CClk32k to the rising edge of SysClk/N. This value will be biased into the upper part of Phase Accum. register, causing the two clocks to be phase aligned. The actual multiplexing when takes place at the following cycle.

3.7 Sequencer Specification

The Sequencers are all identical. They are simple RISC processors with dedicated instruction-set with single-cycle execution - except for a few cases - implemented on a Von Neumann sort of architecture.

3.7.1 Instruction-set

The instruction-set for the Sequencers is summarized in Table 1. 23 instructions are involved in the set, partly adapted from the Hercules μ SM [1].

Opcode	Instruction (6 bits)	Data Field (4 bits)	Extension Field (8 bits)	Delta-time (8 bits)	Description	
GPIO Instructions						
0x00	Clr GPIOA	Select pin	Not used	Delta-time	Clears selected GPIOA pin, synchronized to GPSigA[2]	
0x01	Set GPIOA				Sets selected GPIOA pin, synchronized to GPSigA[2]	
0x02	Clr GPIOB				Clears selected GPIOB pin,	
0x03	Set GPIOB				Sets selected GPIOB pin	
0x04	Clr GPIOC				Clears selected GPIOC pin.	
0x05	Set GPIOC				Sets selected GPIOC pin.	
Internal Trigger Instructions						
0x06	Toggle GPSigA	Select signal	Not used	Delta-time	Generates pulse at selected GPSigA line.	
0x07	Toggle GPSigB				Generates pulse at selected GPSigB line	
0x08	Toggle ARMInt				Generates pulse at selected MCU interrupt line.	
0x09	Toggle DSPInt				Generates pulse at selected DSP interrupt line.	
0x0A	Toggle DMAInt	Select channel			Generates pulse at selected Sys. DMA enable line which enables associated DMA channel.	
0x0B	Toggle DSPDMAInt				Generates pulse at selected DSP DMA enable line which enables associated DMA channel.	
Control and Condition Instructions						
0x0C	ShortWait	Not used		Wait (12 bits)	Waits for <Wait> time-ticks.	
0x3x	LongWait	Abs. Time (24 bits)			Waits until <Abs. Time> arrives plus 3-4 SysClk cycles additional delay before starting the next instruction ³ The instruction implicitly implements the calibration phase compensation	
0x10	If ... Skip Next	Addres (12 bit)		Bit	Tests the <Bit> at <Address> of internal TEP memory, and if set, the next instruction will be substituted by a NOP with same execution delay	
0x11	ClrBit				Clears the <Bit> at <Address>	
0x12	SetBit				Sets the <Bit> at <Address>	
0x0D	Wait	Trigger select	Not used		Wait until selected trigger GPSig[5:0] or the DSP DMA channel 0 IRQOut has occurred.	
0x14	NOP	Not used			No operation	
0x0E	Jump	Absolute address		Not used	Standard branch, jumps to <Absolute address>.	
0x0F	Die	Not used			Stops execution and puts sequencer hardware into idle.	
Data Move Instructions						

³ The uncertainty reflects the inter-clock domain synchronization between SysClk (e.g. 13 MHz) and Clk32k clocks.

Opcode	Instruction (6 bits)	Data Field (4 bits)	Extension Field (8 bits)	Delta-time (8 bits)	Description
0x18	WriteRegF	Reg. File address	Write date (16 bits)		Writes given data immediately to internal register of sequencer.
0x13	DataMoveE	Not used			Initiate DMA transfer, using DMA setup stored in Reg File
0x15-0x17 0x19-0x1F 0x20-0x2F	*** <i>Reserved</i> ***				

Table 1: Summary of the TEP instruction-set for the hardware sequencers.

The default instruction format for the hardware sequencers is shown of Figure 13.

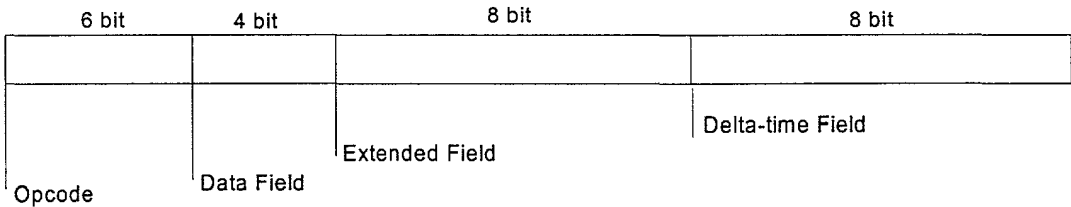


Figure 13: Instruction format for the Timing- and Event Processor.

Exceptions from the format of Figure 13 concerning individual lengths of the various fields exist, as seen from the table.

The ‘If ... Skip Next’ instruction is for handling conditionals based on evaluating any bit in the internal TEP SRAM space. If this instruction evaluates true the following instruction is substituted by a no-operation, the delta-time - if any - is still executed so execution time remains unchanged. The ‘ClrBit’ and ‘SetBit’, respectively associate to the handling of bit-conditional execution as these allow to clear/set any bit in the TEP RAM space.

3.7.2 Sequencer Architecture

The implementation of the instruction-set will be done with the Sequencer architecture shown in Figure 14.

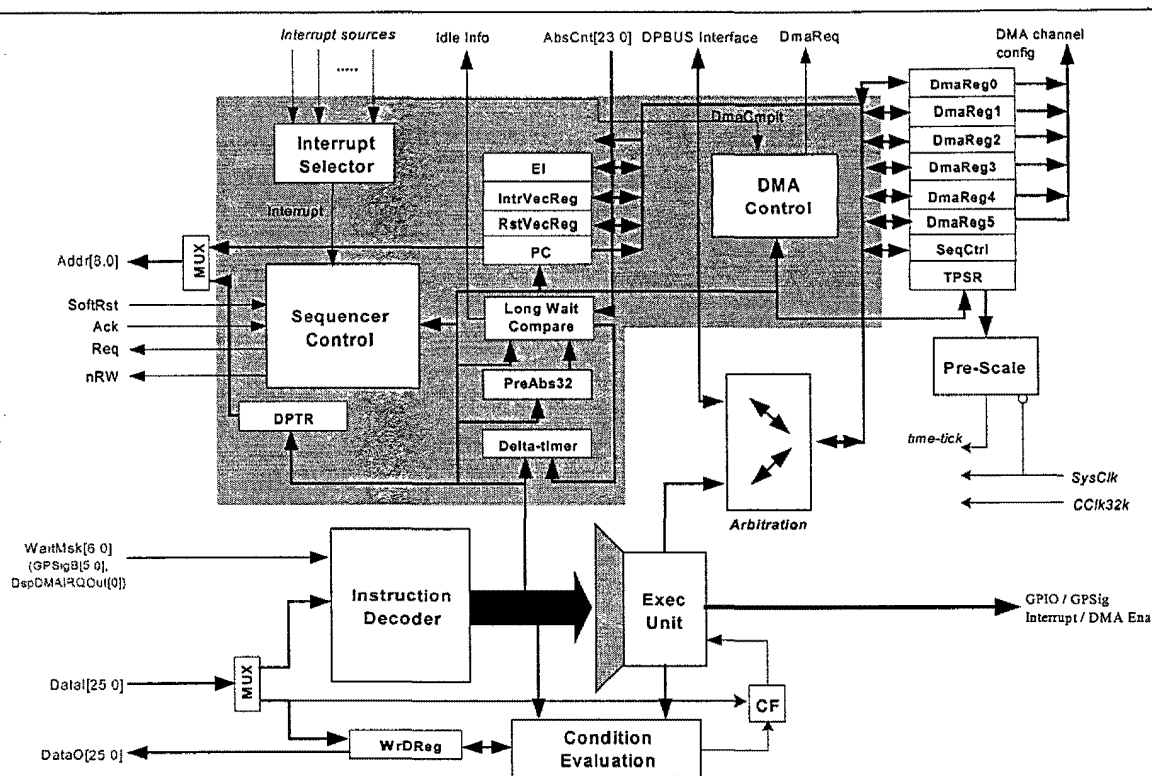


Figure 14: *The Sequencer architecture.*

The parts in the gray box represent control features while the parts outside represent data-paths and processing.

3.7.3 Clock Pre-Scaler

The Clock Pre-Scaler does a clock division of the SysClk to generate an appropriate time-base or time-tick for the Sequencer. Delta-times of instructions are counted with this time-tick. Main purpose is to use as low a frequency as possible while still providing sufficient resolution compared to the actual wireless system. Of course, low frequency is also desirable for power saving reasons.

The pre-scaler must be programmable by the host processor between 2 and 64. This provides a sufficient range to trade-off various wireless system requirements and power savings. Functional, either the host processor or Sequencer can alter the pre-scaling and physically the register is placed within the Sequencer register-file. This way a timing adjustment feature – functional equivalent to the μ SM [1] feature – will be supported.

3.7.4 Sequencer Data-Path

The purpose of the execution unit is mainly to generate appropriate flag signals for GPIO, DMA and interrupt. The number of GPIO groups remains three as for Hercules, GPIOA, GPIOB and GPIOC, each of 16 bits. Setting and clearing of signals from group GIOA will only take place synchronized to a GPSigA[2] pulse. The signals from the GPIOB/C groups will be processed in the next SysClk cycle.

The read/write GPIO registers remain placed in the MMI module as for Hercules but with altered interface from TEP. The interface involves for - each GPIO group - a 16 bits registered value, which must be multiplexed in the MMI module with the read/write register, depending on configuration.

The internal signals involve,

- ☐ Interrupt triggers for processors
- ☐ Channel enables for System and DSP DMA controllers.
- ☐ General-purpose internal triggers.

Each of these will have 16 signals assigned. The interface involves - for each group - 16 trigger lines. A trigger event is signaled as a one SysClk period pulse. The general-purpose internal signals are grouped in two as GPSigA and GPSigB, respectively. These 32 triggers are assigned as stated in the table in below,

Trigger	Assignment Description
GPSigA[15:5]	*** Reserved ***
GPSigA[4]	Trigger to start serialization of GSPe data (synthesizer programming),
GPSigA[3]	*** Reserved ***
GPSigA[2]	Forces update of GPIOA signal group.
GPSigA[1]	Enable/allow system PLL power-off.
GPSigA[0]	Request system PLL power-on.
GPSigB[15:7]	*** Reserved ***
GPSigB[6]	Interrupt signal feeding to all Sequencers.
GPSigB[5:0]	Execution triggers, used by 'Wait' instructions as trigger condition.

The WrDReg and Condition Evaluation module implement data processing and transfer for 'If ... Skip Next', 'ClrBit' and 'SetBit' instructions.

3.7.5 Sequencer Control-Path

The overall control of the Sequencer involves three states as shown of Figure 15. In IDLE state all clock is off and requires a soft-reset or interrupt to enable clocking and exiting the state. The SOFTRESET is an intermediate state and RUNNING is the normal execution state.

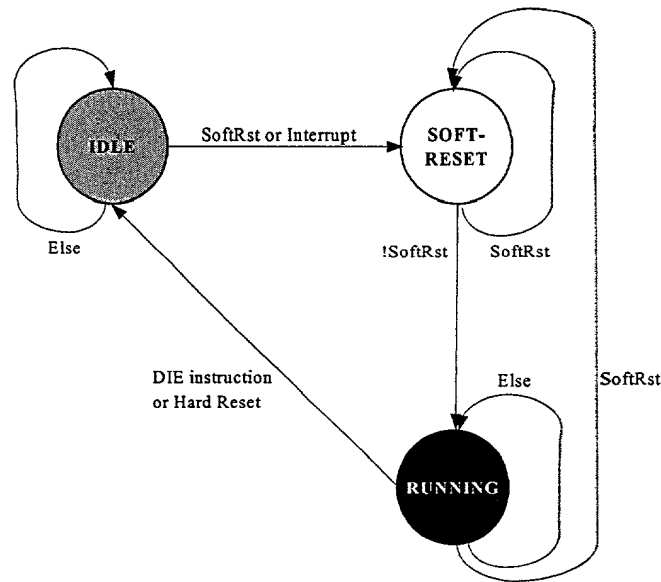


Figure 15: State diagram for the overall control of the Sequencer.

When in IDLE state soft-reset or interrupt (if enabled) works as wake-up events, which triggers the Sequencer to start clocking and shift to SOFT-RESET state. The Sequencer stays in this mode as long as the soft-reset is being asserted. Once deasserted the state shifts to RUNNING and the Sequencer starts executing code until either a 'DIE' instruction has been executed or a hard-reset is being reasserted.

3.7.6 Pipelining and Timing

The execution of instructions is single-cycle on a simple three-stage fetch-decode-execute pipeline. The timing is specified in Figure 16. The execution is always clocked by, and hence, synchronized to the SysClk. However, an instruction remains pending after being fetched until its delta-time value has been reached. The delta-timer will be increased every time-tick and once it matches the delta-time value of the pending instruction this is executed in the next SysClk. For instructions having a non-zero delta-time value, execution can only take place at the time-tick boundary in which the delta-time has been reached. In this case the execution pipeline is blocked until the instruction has been executed. At the same time the delta-timer is reset and the pipeline is re-released.

The 'JUMP' instruction and interrupt are exceptions from the single-cycle execution, each taking two cycles. Their timing being identical is also specified in Figure 16.

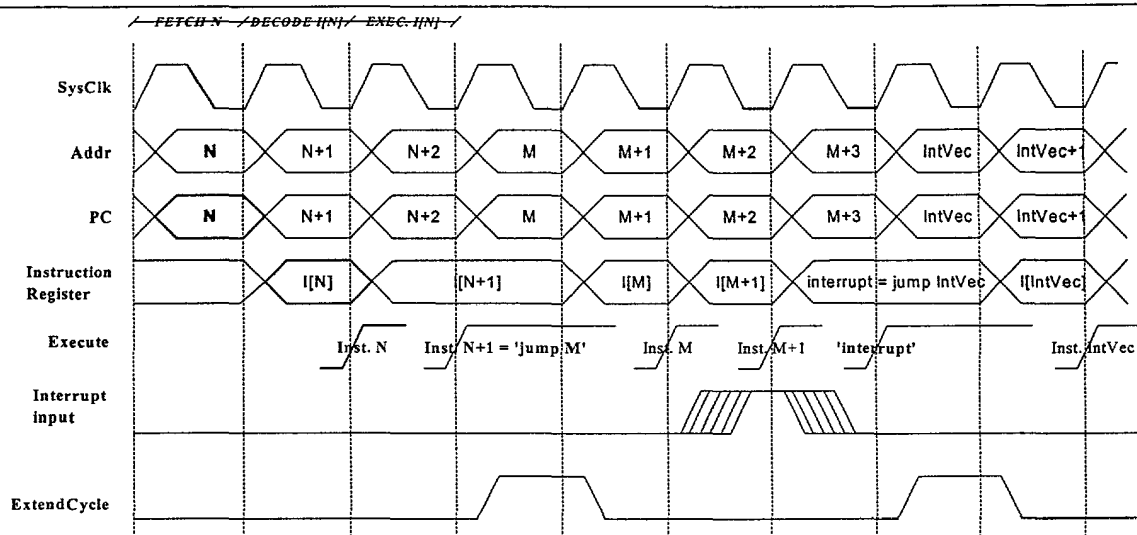


Figure 16: Instruction timing for the standard case as well as jump and interrupt cases. The latter being identical and taking two cycles.

Another exception from the single-cycle instruction is the 'ClrBit' and 'SetBit' instructions, each taking three cycles. The timing for these is shown in Figure 17. The three cycles are used for setting up the address of the word to test, reading the word from RAM and writing back the modified contents. Due to the single bus connection to RAM the instruction fetching and execution must be blocked during all three cycles.

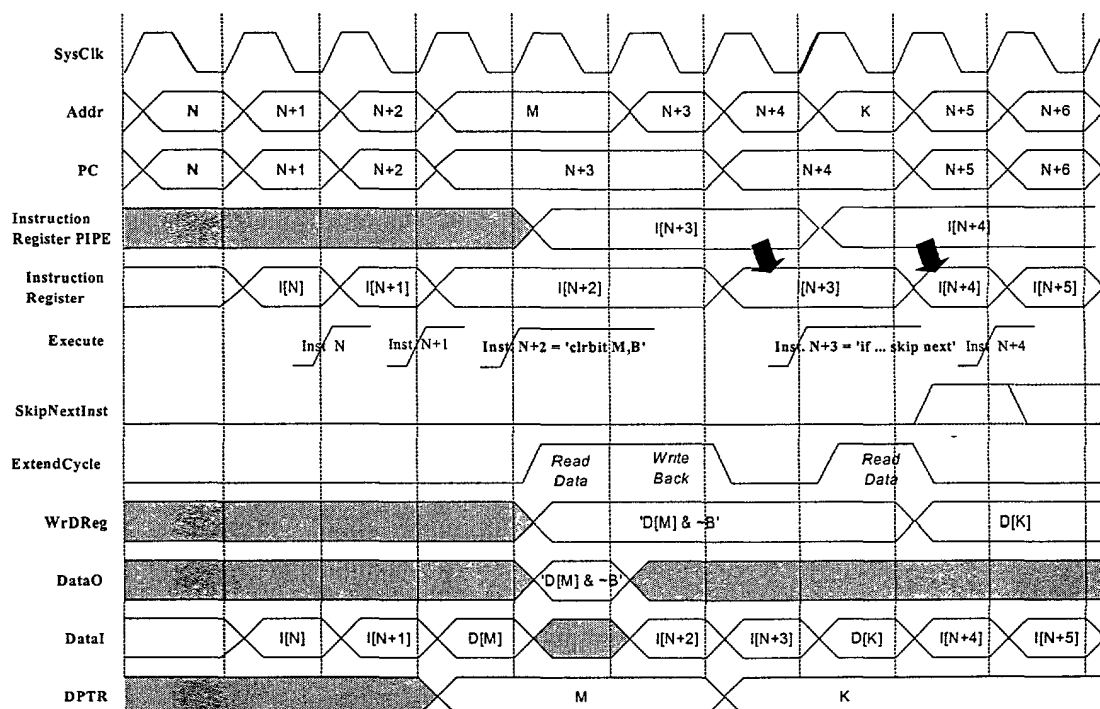


Figure 17: Instruction timing for the three-cycles instructions 'ClrBit' or 'SetBit', and the double-cycle instruction 'If ... Skip Next', respectively.

The 'if ... skip next' instruction is the last exception taking two cycles to execute. It consists of a setup cycle and a data read cycle. If evaluated true the *SkipNextInst* flag will be set, disabling the next

instruction to execute. The flag remains asserted until the delta-timer for the next instruction has expired as shown of Figure 17. The next instruction delta-timer will always count independent of *SkipNextInst*; only the actual operation may be substituted by a NOP.

The 'LongWait' instruction is executed in the Clk32k domain, and hence, requires additional cycles inserted for *inter-clock synchronization* (double edge sampling technique). Depending on clock phases this introduces an unavoidable uncertainty of one SysClk cycle, which may need awareness software wise. The timing is shown of Figure 18.

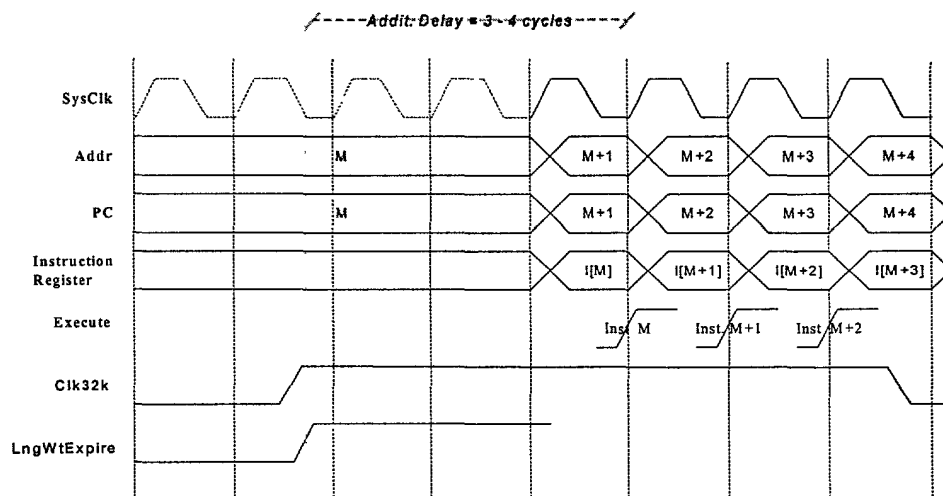


Figure 18: Instruction timing for 'LongWait'. Required inter-clock synchronization insert additional 3-4 SysClk cycles following the absolute time has been reached.

In case a fetch, data read or data write operation cannot be granted by the memory one or more SysClk cycles will be inserted while the Sequencer is halted. If this postponement or too many zero-delta-time instructions in row cause a time-tick crossing, the delta-time value (if none-zero) must be recalculated to avoid the succeeding instructions to be influenced timing wise. This will be achieved by advancing the delta-timer by one as shown of Figure 19.

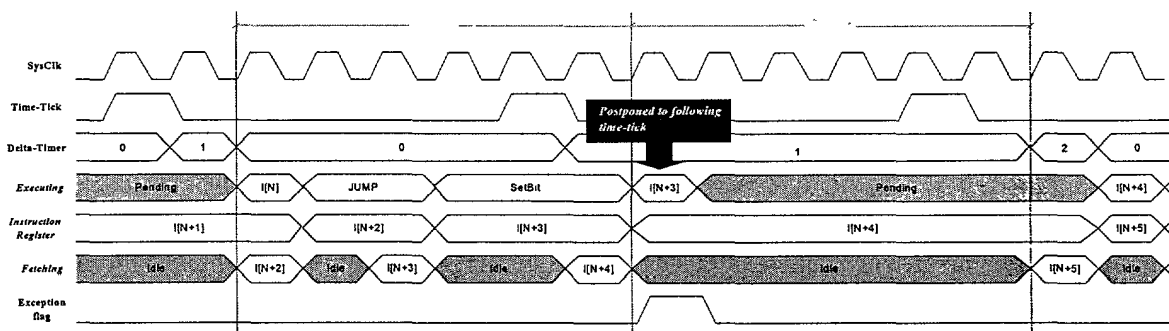


Figure 19: The handling of postponement of instructions to the following time-tick is done by incrementing delta-timer, flagging an exception and nothing else.

The last instruction to be scheduled to execute in any time-tick will always have a none-zero delta-time value and as long as the postponement is less than the delta-time value, the execution will catch up without loosing track – using the specified technique.

An exception must be generated to flag the Layer1 software.

3.7.7 Register File

The Sequencer contains a register file consisting of 15 registers. Most of these can be modified by the Sequencer instruction, 'WrReg'. The registers can be read and written by the Host processor from the DPBUS bus system.

Register Name	Address	SW Access Permission		Description
		Sequencer	DPBUS	
DmaReg0	0x0	Write-only	Read/write	The function of these registers is identical to the 86 bits configuration for the DMA channel 0, cf. [12]
DmaReg1	0x1			
DmaReg2	0x2			
DmaReg3	0x3			
DmaReg4	0x4			
DmaReg5	0x5			
DIRst	0x6	Write-only	Write-only	DMA IRQOut Reset register Init. 0x0000 Bit 15-0: Writing '1' will reset the associated DMA IRQOut signal.
SeqCtrl	0x7	Write-only	Read/write	Sequencer control register. Init. 0x0000 Bit 15-7: Not used. Bit 6: Initiate snapshot feature. Bit 5: Interrupt the Sequencer once. Bit 4-0: Select interrupt source. 0x1F- Reserved. 0x18 0x17- GPSigB[6] for the 0x10 associated Sequencers. 0x0F- Associated DSP 0x00 DMA channel.
EI	0x8	Write-only	Read-only	Sequencer interrupt enable register. Init. 0x0000 Bit 15-1: Not used Bit 0: '1' = Enable. Bit 0 will be reset when jumping to interrupt vector.
RstVecReg	0x9	Write-only	Read/write	Soft reset address vector. Init. 0x0000 Bit 15-9: Not used. Bit 8-0: Reset address, which Sequencer jumps to upon soft reset.
IntVecReg	0xA	Write-only	Read/write	Interrupt address vector. Init. 0x0000 Bit 15-9: 0x0000 Bit 8-0: The interrupt jump vector, which the Sequencer jumps to upon interrupt.

Register Name	Address	SW Access Permission		Description
		Sequencer	DPBUS	
TPSR	0xB	Write-only	Write-only	TEP Pre-Scaler Register. Init. 0x000B Bit 15-6: Not used Bit 5-0: Pre-scale value + 1, that is, time-base = $\text{SysClk}/(\text{TPSR}[5:0]+1)$
PC	0xC	Inaccessible	Read-only	Sequencer program counter.
WrDReg	--	Inaccessible		Data holding register for read/modify write operations.
DPTR	--	Inaccessible		Data pointer register.

Table 2: *The Sequencer Register-File definition.*

Conflicting access to registers by Sequencer and DPBUS cannot occur, as access grants to the register-file are resolved in the arbitration. *The Sequencer has preference over the DPBUS write-access*, which will be postponed until next SysClk cycle. Simultaneous access to different registers does not pose a conflict. Similar, DPBUS read of a register might take place at same time as Sequencer *write access to different register*.

3.8 I/O Conflict Resolver

The purpose of the I/O Conflict Resolver module is to combines the outputs from the Sequencers and in case of conflicts resolves matters as well as registers the GPIO outputs. Conflicts can only occur for the GPIO signals and clock calibration snapshots. For interrupts and DMA channel enables, matching signals, like MCUInt5 from each Sequencer, are just OR'ed together.

For the GPIO groups, each is controlled by a clear and a set signal from each Sequencer. *A conflict occurs if one Sequencer asserts a clear, for e.g. GPIOA5, and another in same cycle asserts a set for the same signal.* In such case clear rules over set, meaning in case of conflict the associated signal will be cleared. To flag software about the incident an exception bit will be set and optionally an interrupt must be generated to the Host processor.

The Clock Calibration block can execute only one snapshot-command at a time. If a new snapshot is being attempted while another is already being executed the latter is ignored.

3.9 Memory Access Resolver and SRAM

The SRAM block is shared between the Sequencers for code and data storage. The Memory Access Resolver provides access to the Sequencers and host processor when granted for, and solves simultaneous access conflicts by postponing one or more access to next SysClk cycle.

3.9.1 Memory Structure

There are N Sequencers and DPBUS, which each may request memory access on a SysClk cycle basis. Offering a M-port memory ($N \geq M$), with ports being equally shared between DPBUS and Sequencers, now and when, access requests may be postponed one or more SysClk cycles. This may cause instructions supposedly executed within a given time-tick to be postponed to the next time-

tick⁴. However practically thinking, it will be very rarely that all Sequencers have instructions to execute within one time-tick. This implies that when this situation occur in one time-tick, it is not likely to happen in next time-tick as well, and even less likely in the following time-tick. So within a few time-ticks the buffered instructions will be executed. *To avoid accumulated timing errors caused by such a case, it is necessary the delta-time of the postponed instruction is decreased with the same amount so that succeeding instructions will not be influenced regarding their timing.*

A multi read-port will be at the expense of increased area and the instructions decoders, etc. must be increased in number to match the number of read-ports as well. A fair combination will be to add a minor amount of read-ports and applying prioritized cycle-shared access among these. The selected memory structure must provide two read-ports and one write-port (full dual-port memory), as shown of Figure 20.

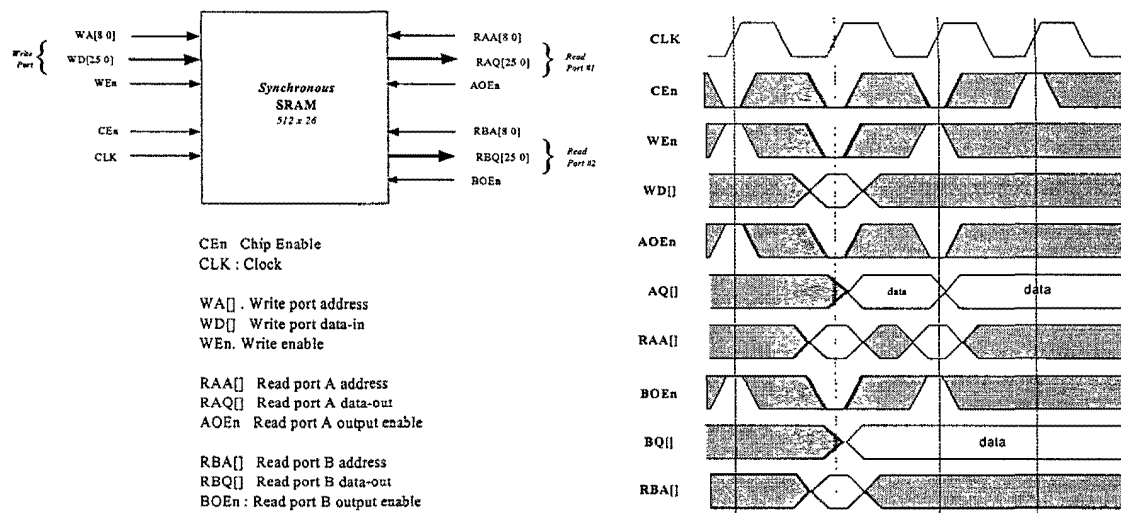


Figure 20: Block diagram of the TEP multi-ported memory, one write- and two read ports with separate address busses and allowing simultaneous and independent access.

3.9.2 Memory Access Resolver

Main purpose of the Memory Access Resolver is, on a cycle-by-cycle basis, to process every incoming access request from the Sequencers and DPBUS interface and granting access to as many as allowed by the number of read-ports of the memory. In case of more requests than read-ports the module must prioritize among the request based on the simple round-robin scheme.

A memory access must be done in 1 clock cycle, and hence, the round-robin priority selection must be combinatorial. The round-robin will be implemented as a shift-back register with as many bits as requesters and one bit set, indicating priority. Priority decoding will be done combinatorial from incoming requests and the shift-register and at each access the shift-register must be updated.

Each memory port must have its own priority selector, which must ensure a granting requester is signaled by asserting its *acknowledge flag* and the request being canceled at the other selectors. A requester not being granted in the current cycle will have to insert wait cycles until granted. *In order*

⁴ Naturally, this can cause exceptions, e.g. with many zero delta-time instructions, instructions requiring data fetch as well and extensive host processor access. This needs some statistics and some exception analysis.

not to block the Sequencer execution the DPBUS access will always have lowest priority at the read ports.

3.10 DPBUS Bus Interface

The DPBUS bus is an *ARM APB inspired bus*. The DPBUS Interface module provides bridging between the SysClk and DPBUS clock domains within TEP and handles the 16/32 bits interfacing, besides the usual memory decoding, signal generation, etc. A block diagram is shown of Figure 21.

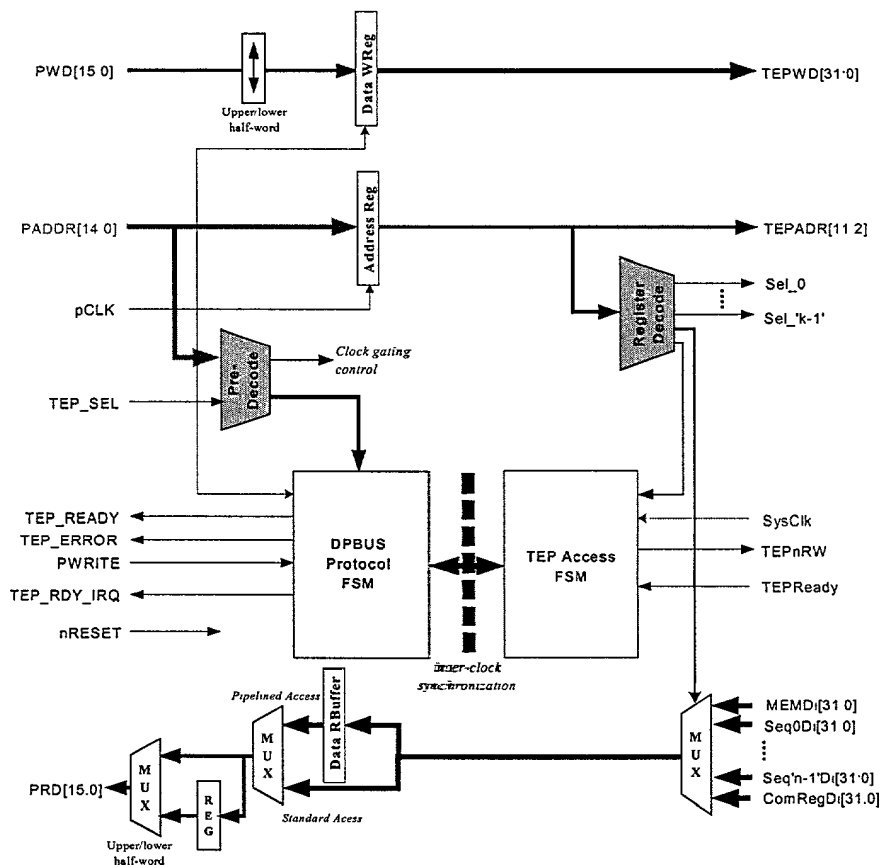


Figure 21: Functional block diagram for the DPBUS bus interface.

The DPBUS [15] is compatible to the PBUS [14]. It is a byte addressable half-word aligned 16 bits width bus. The TEP however is word aligned 32 bits width internally. This must be solved by using two little-endian successive DPBUS accesses for each 32 bits read/write.

16 bits bus / 32 bits access (device)		
Access Addr [1:0]	00	10
TEP Data[31:16]		PWD[15:0]/PRD[15:0]
TEP Data[15:0]	PWD[15:0]/PRD[15:0]	

A DPBUS write access must commence with writing lower half-word to address 00 (will stored temporarily internal TEP) and (immediately) followed by writing upper half-word to address 10 at which point the full word will be written to specified field of TEP. Whereas a DBPUS read access must commence by reading lower half-word from address 00 at which point all 32 bits will be read from TEP and the upper half-word stored temporarily. Hence the upper half-word may be read

immediately afterwards by a DPBUS access to address 10. A detection mechanism must flag the DPBUS error signal in case this access sequence is being violated.

The module performs inter-clock synchronization between DPBUS clock and SysClk. Each clock domain is controlled separately with handshaking signals for inter-synchronization. The 'DPBUS Protocol FSM' handles the interface signals towards the DPBUS. A transition diagram showing its functional states are shown of Figure 22.

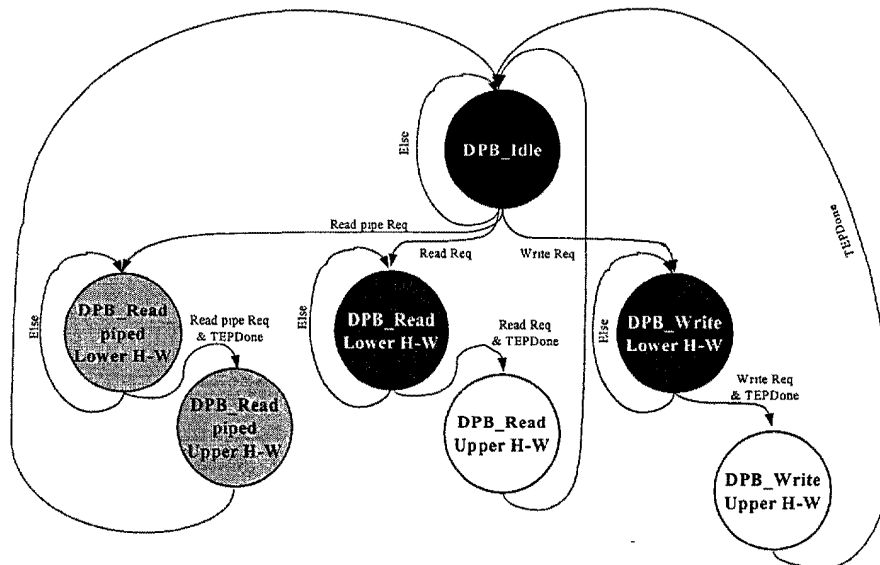


Figure 22: Functional state transition diagram for the DPBUS Protocol.

This FSM triggers the 'TEP Access FSM' to conduct a write or read operation from internal TEP, which flags back via *TEPDone* when complete.

The ‘TEP Access FSM’ is a read/write state machine towards the internal of TEP; its state diagram is shown of Figure 23.

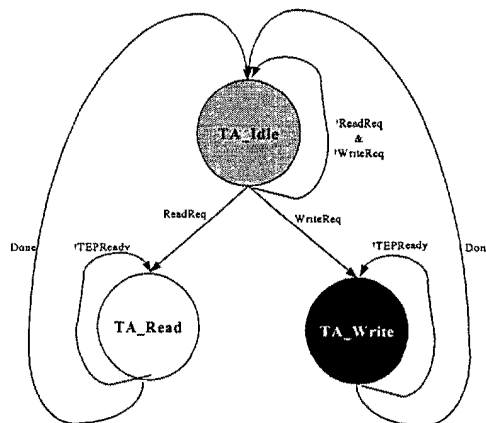


Figure 23: State *transition* diagram for the ‘TEP Access FSM’.

The *ReadReq* and *WriteReq* flags are passed, pos.edge/neg.edge sampled, from the 'DPBUS Protocol FSM' and initiate either a read- or a write operation. Once complete, the FSM flags back the 'DPBUS Protocol FSM' and goes back to TA Idle state.

All accesses from the DPBUS will take at least three DSCLK cycles (1 + 2 waits). In order not to lock up the faster (compared to internal TEP) DPBUS bus, the DPBUS interface feature a single word write operation with zero additional wait-cycles. Another immediate write operation will cause wait-cycles insertion until the previous words has been transferred to the SysClk domain and written by 'TEP Access FSM'. Read access features to functions, *direct* and *pipelined reading*, differentiated by address. A *direct read* access will cause wait-states insertion until data has been transferred from SysClk domain, and hence, lock up the DPBUS. A *pipelined read* access will give zero additional wait-cycles access to the data stored at the address of the previous read operation. At the same time the read will cause 'TEP Access FSM' to initiate fetching data of the current address. A new read access from DPBUS before this fetch has been completed will cause wait-states insertion until so done. By proper software pipelining this can prevent locking up the DPBUS.

To enable DMA handled access pipelined read and write operations may optionally generate an interrupt whenever TEP is ready for new data or pipeline read data has been fetched and ready for reading.

The timing diagram for the DPBUS signals is shown of Figure 24. Write data must be clocked into TEP at falling edge of PCLK and read data must be latched by DPBUS as falling edge also. The wait signal, TEP_nWAIT, will be evaluated at rising edge of PCLK.

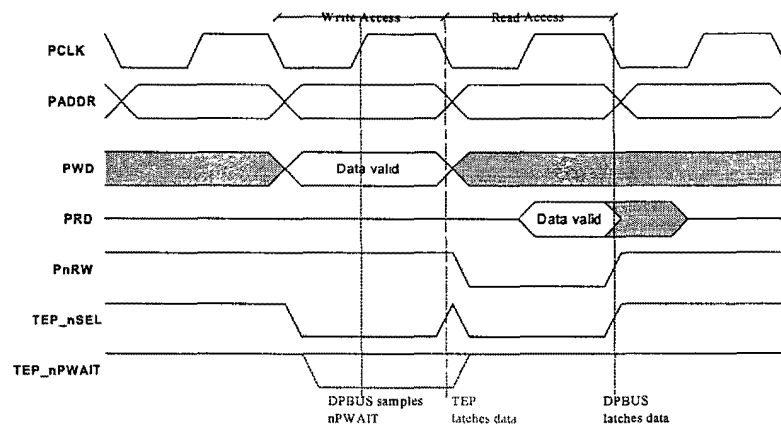


Figure 24: Illustration of the DPBUS interface timing diagram.

3.11 TEP - DSP DMA interface

The TEP employs one DMA channel of the DSP DMA to do **any** read/write accesses external to TEP area. The use of the channel is semi-hidden and done via the 'DataMoveE' from the sequencer instruction-set. The interface/-play is illustrated in Figure 25. The DMA channel is fixed allocated for this usage and cannot be used by other resources; channel #0 will be assumed.

The 'DataMoveE' instruction uses the contents of the DmaReg5-DmaReg0 of the Sequencer RegFile as configuration for the DMA channel. Several Sequencers may acquire the DMA channel simultaneously. This must be handled by the **Request Resolver FSM** module based on a simple *round-robin priority scheme*, identical to the Memory Access Resolver one. Once granted the DmaReg's are copied to the 83 bit register⁵, which internal the DMA controller substitutes the

⁵ The DSP DMA controller specification is not available. However, the configuration probably must be extended to 83 bits compared to Hermes.

normal SRAM area for this channel. Next, the **Request Resolver FSM** module sets the ChanEnable0 for enabling DMA channel 0 and when waits for it to return an interrupt indicating transfer completion as shown of Figure 26. While running the block asserts the DRRReqSysClk flag to ensure the SysClk is not being powered-off.

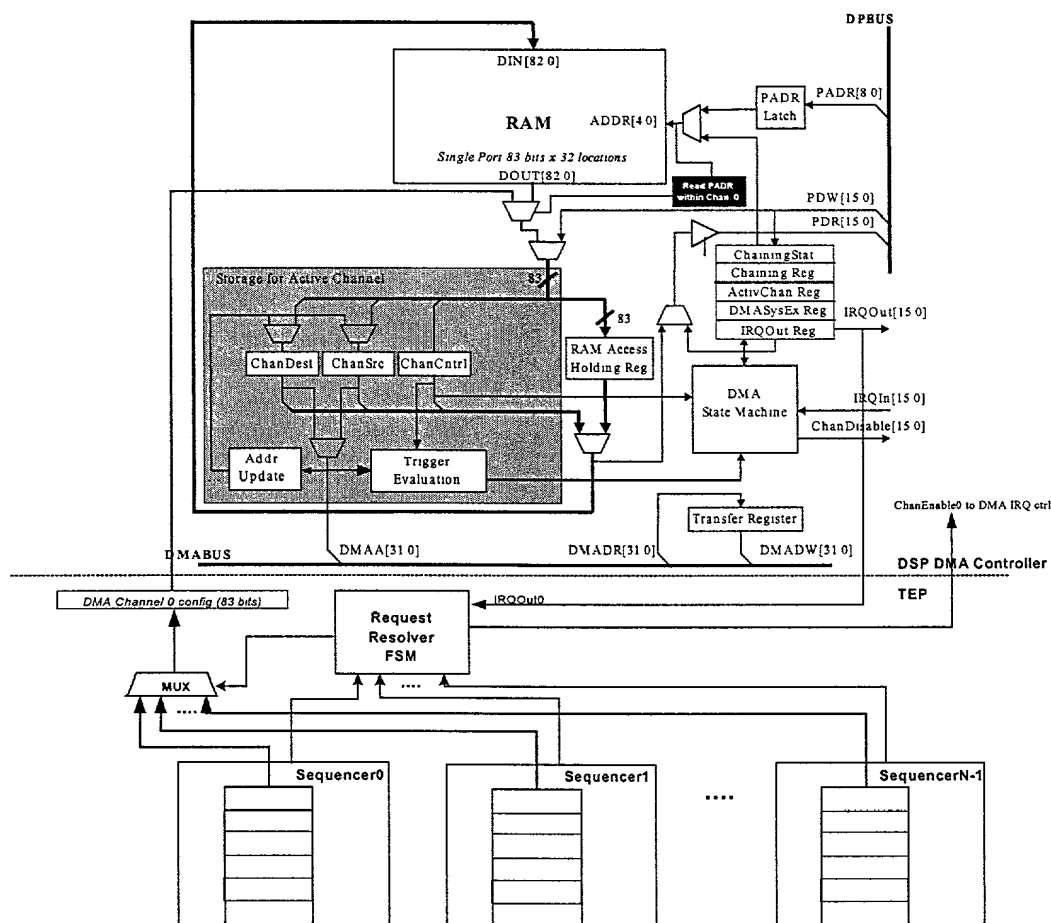


Figure 25: The interface between the TEP and the DSP DMA controller assuming channel 0 is allocated for this purpose.

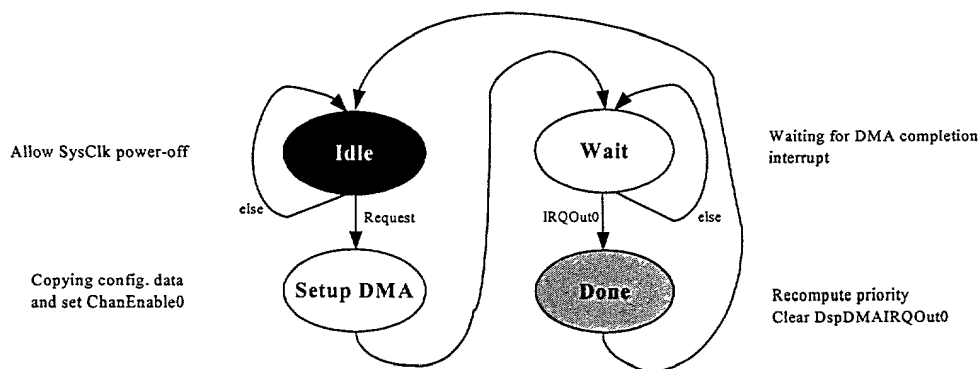


Figure 26: State transition diagram for the DMA Request Resolver FSM.

3.12 Error Handling

In case of exception, proper error flagging must be generated to the host for software to identify and process. This means that errors should be flagged with a dedicated error interrupt, and that a status register should give detailed information on the nature of the error.

The following error conditions have been identified (partly adapted from Hercules),

- ❑ DMA Request Resolver conflict. This means two or more Sequencers are requesting the dedicated DMA channel simultaneously, and hence, some must be postponed. Status bits must show which Sequencer has been postponed.
- ❑ Memory Access Resolver conflict. This means the number of access request during one cycle has exceeded the number of read-ports. Status bits must show which Sequencer has been postponed.
- ❑ Concurrency limit has been reached. This means more instructions were intended executed within one time-tick than available, i.e. *cycles/time-tick-read-ports*.
- ❑ I/O conflict – e.g. two instructions executed simultaneously conflicts by setting/clearing one signal.
- ❑ Snapshot conflict. A new snapshot has been attempted while the Clock Calibration block is already executing another snapshot.
- ❑ Illegal instruction has been executed.

Each exception event must set an associated status bit in the **TESR** register and interrupt the host if enabled by the **TEIMR** register.

3.13 Debug Features

For debugging TEP features,

- ❑ The well-known Monitor port for viewing various internal signals
- ❑ The High-Speed Logger for logging the *executing instructions*, selectable for one of the Sequencers at a time.
- ❑ A halt mode with *software triggered single SysClk execution*. That is, once in halt mode, writing a register from the host processor causes normal execution for one cycle at which the Sequencers stops again, waiting for next triggering.

3.14 Clock Domains and Inter-Synchronization techniques

The TEP module involves three clock domains, the 13/26 MHz System Clock (SysClk), the DPBUS clock (DSCLK) and the free running 32 kHz (Clk32k). Derived (e.g. time-tick derived from SysClk will serve as enable/gate rather than an actual clock) and several gated versions of these clocks will be used. All clock gating must be bypassed by asserting the TM control signal, for reasons of DFT [7].

However, only when crossing the three listed clock domains inter-clock synchronization must be used. Straightforward, double-edge sampling techniques will be used as shown of Figure 27. In order not to increase latency times too much pos.edge/neg.edge sampling will be applied for latency critical crossings.

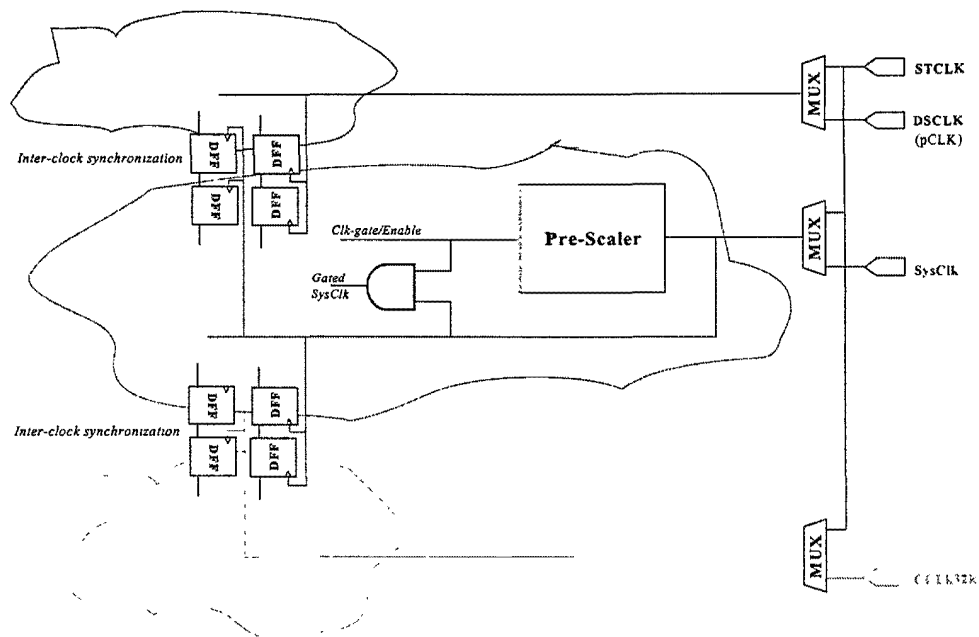


Figure 27: Simplified view of the clock domains for TEP, including MUX's for selecting STCLK for scan-test usage.

4. HW Interface Description

The Timing- and Event Processor interfaces to the DSP DPBUS, the System- and DSP DMA controllers, MCU and DSP interrupt controllers, MMI GPIO pins and system clock oscillators, reset and PLL control and power control.

The timing of the interface signals to GPIO, GPSig, IRQ- and DMA controllers and HSL logger is in general shown of Figure 28.

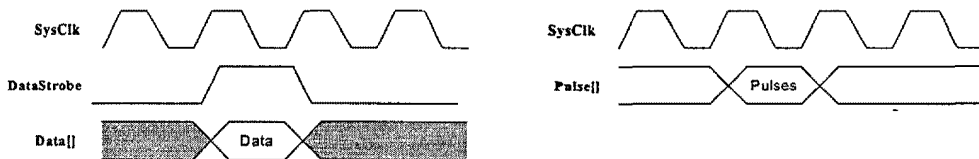


Figure 28: Timing for the TEP interfacing signals to GPIO (to the left), GPSig, IRQ- and DMA controllers (to the right).

The interface either consists of SysClk synchronized data and an associated data-strobe or a set of SysClk synchronized triggers, each one SysClk cycle in duration.

4.1 Clock, reset and system interface

Signal Name	Direction	Polarity	Signal Description
nReset	I	Low	Asynchronous hardware reset

Signal Name	Direction	Polarity	Signal Description
Clk13M	I	--	13/26 MHz system clock input
Clk32k	I	--	Free-running 32 kHz clock input
SysClk	O	--	System clock (gated 13 MHz) for on-chip usage.
CClk32k	O	--	Calibrated 32 kHz clock output
DSCLK	I	--	DPBUS System Clock input.
HALT	I	High	Halt signal input.
PllOnIntr	I	High	Leveled interrupt input causing the PLL to power up/ stay on. Must remain asserted.
DSPReqPll	I	High	DSP requires PLL signal. Must remain asserted.
MCUReqPll	I	High	MCU requires PLL signal. Must remain asserted.
SysDMAReqPll	I	High	System DMA controller requires PLL signal. Must remain asserted.
DspDMAReqPll	I	High	DSP DMA controller requires PLL signal. Must remain asserted.
MCUReqSysClk	I	High	DSP requires system clock signal. Must remain asserted.
DSPReqSysClk	I	High	MCU requires system clock signal. Must remain asserted.
SysDMAReqSysClk	I	High	System DMA controller requires system clock signal. Must remain asserted.
DspDMAReqSysClk	I	High	DSP DMA controller requires system clock signal. Must remain asserted.
WakeUpInt	I	High	Wake-Up interrupt requires system clock to be re-powered. Must remain asserted.
SysOscOn	O	High	Power-on control signal for system oscillator LDO
ClkBufOn	O	High	Power-on signal for clock input pad LDO.
SysPllOn	O	High	Power-on control signal for system PLL

4.2 DPBUS (DPBUS) interface

The TEP interfaces to the ARM APB inspired DPBUS [TBD]

Signal Name	Direction	Polarity	Signal Description
PWD[15:0]	I	--	DPBUS write data bus.
PADDR[13:0]	I	--	DPBUS address bus.
TEP_nSEL	I	Low	TEP chip select.
PnRW	I	--	DPBUS read/write flag.
TEP_nWAIT	O	Low	DPBUS wait flag used for wait-state insertion when low.
TEP_ERROR	O	High	TEP DPBUS error signal.
PRD[15:0]	O	--	DPBUS read data bus.
TEP_RDY_IRQ	O	High	Interrupt indicating that the TEP DPBUS interface is ready for new data.

4.3 DSP DMA Controller interface

The interface to the DSP DMA is similar to the other trigger interfaces allowing multiple settings per cycle, and hence, represents a change compared to Hercules. It includes also the dedicated port for configuration and interrupt-feedback of the fixed allocated channel.

Signal Name	Direction	Polarity	Signal Description
DspDMAChEna[15:0]	O	High	DMA channel enable signals, high for one SysClk cycle.
DspDMACh0Con	O	High	Enable signal indicating DMA controller must load configuration from dedicated port rather than RAM.
DspDMACh0Cfg[85:0]	O	-	Configuration data for Channel 0 passed to DMA controller when initiating transfer.
DspDMAIRQOut[15:0]	I	High	IRQOut from DSP DMA used for interrupting Sequencers.
DspDMAIRQRst[15:0]	O	High	Reset strobe for deasserting the DspDMAIRQOut[15:0]

4.4 System DMA Controller interface

The interface to the System DMA is similar to the other trigger interfaces allowing multiple settings per cycle, and hence, represent a change compared to Hercules.

Signal Name	Direction	Polarity	Signal Description
SysDMAChEna[15:0]	O	High	DMA channel enable signals, high for 1 SysClk cycle.

4.5 MCU and DSP Interrupt Controllers interface

The interfaces the MCU and DSP IRQ controllers each consist of 16 bits parallel strobe field, allowing multiple interrupt triggers being set per cycle.

Signal Name	Direction	Polarity	Signal Description
TEPMcuIRQ[15:0]	O	High	MCU interrupt triggers, high for one SysClk cycle.
TEPExcIRQ	O	High	TEP Exception Interrupt output.
TEPDspIRQ[15:0]	O	High	DSP interrupt triggers, high for one SysClk cycle.
TEPRdyIRQ	O	High	Interrupt indicating write or pipelined read operation has completed.
TEPSnapRdy	O	High	Snapshot operation complete interrupt, asserted synchronized to clk32k.

4.6 Internal General Purpose Signals interface

The internal general purpose signal groups each consist of 16 bits parallel strobe field, allowing multiple interrupt signals being pulsed per cycle.

Signal Name	Direction	Polarity	Signal Description
GPSigA[15:0]	O	High	The group A signal enables/triggers, asserted for one SysClk cycle.
GPSigB[15:0]	O	High	The group B signal enables/triggers, asserted for one SysClk cycle.

4.7 MMI GPIO Signals

The interface to the GPIO signal groups each involves a 16 bits data field being the output pin values. These are to be multiplexed or OR'ed with the associated pin values of the GPIO's in the MMI module to form the eventual pin value.

Signal Name	Direction	Polarity	Signal Description
GPIOAD[15:0]	O	-	Registered data for GPIOA.
GPIOBD[15:0]	O	-	Registered data for GPIOB.
GPI OCD[15:0]	O	-	Registered data for GPIOC.

4.8 Scan-test interface

The set of scan-test signals are except for TM not part of the specifications. Their purpose is to facilitate DFT pf the module and will be assumed handled by test-compiler/DFT process and used by ATPG tool. The test mode signal, TM, will be taken into account in the sense it must, when asserted high, overrule DFT-incompliant design issues, like gated clocks, internally generated resets, etc.

Signal Name	Direction	Polarity	Signal Description
SI	I	-	Scan-test serial data input
SE	I	High	Scan-test enable input
SO	O	-	Scan-test serial data output
STCLK	I	-	Scan-test clock input
STRST	I	High	Scan-test reset input
TM	I	High	Scan-test mode selector input

4.9 Monitor Port interface

The monitor port interface follows previous implementations [1]. And likewise for the actual monitor port function itself.

Signal Name	Direction	Polarity	Signal Description
MonitorPort[12:0]	O	-	13 bits output from internal monitor MUX for debug, etc. Tri-state outputs, only driven when ModuleSelect is high.
MonitorSelect[2:0]	I	-	MUX selector inputs.
ModuleSelect	I	High	Select signal for enabling cipher engine

Signal Name	Direction	Polarity	Signal Description
			Monitor port.

Actual features/signals to be routed to the port are [tbd].

4.10 High-Speed Logger interface

The HSL interface follows previous implementations [1]. Only one of the Sequencers have access to the port, depending on HSLSel[2:0]. The Sequencer having access will write instructions being executed, when done so.

Signal Name	Direction	Polarity	Signal Description
HSLStrobe	O	High	Enable strobe for HSL data, one SysClk cycle high.
HSLData[7:0]	O	-	HSL data output, data valid whenever HSLStrobe is set.
HSLSel[2:0]	I	-	Select which Sequencer to have access to HSL port.

5. SW Interface Description

5.1 Memory Map

This section contains a listing of the registers of the Timebase and Event Processor and a short description of each of the registers. The address map is shown in the table in below,

Every register/RAM is word aligned. To read the entire contents the lower half-word can be read at the address and the upper half-word can be read next at the address + 2, and the reading must be done in that order. For writings the lower half-word must be written to the address and the upper half-word next written the address + 2, and again, in that order – **both** half-words **must** be written always.

Register Name		Address
<i>Standard Read Access Area</i>		
OWUR	Oscillator Warm Up Register	0x000
CPPUR	Clock Pad Power Up Register	0x004
TCLR	TEP Clock Calibration Length Register	0x008
*** Reserved ***		0x00C
TESR	TEP Exception Status Register	0x010
ITESR	Interruptable TEP Exception Status Register – equals TESR & TEIMR	0x014
TEIMR	TEP Exception Interrupt Mask Register	0x018
TCR	TEP Control Register	0x01C
The Sequencer 0 register-file ⁶		0x05F-0x020

⁶ Addressable as word aligned four-bytes-separated entities.

Register Name		Address
The Sequencer 1 register-file ⁶		0x09F-0x060
The Sequencer 2 register-file ⁶		0x0DF-0x0A0
The Sequencer 3 register-file ⁶		0x11F-0x0E0
The Sequencer 4 register-file ⁶		0x15F-0x120
*** Reserved ***		0x21F-0x160
TACR	TEP Absolute Counter Register	0x220
TSST	TEP Single-Step Trigger register	0x224
TCCFR	TEP Clock Calibration Fraction Register	0x228
TCCMR	TEP Clock Calibration Modulus Register	0x22C
TCRBR	TEP Calibration Read-Back Register	0x230
*** Reserved ***		0x3FF-0x230
512 x 26 SRAM area		0xBFF-0x400
Pipelined Read Access Area		
OWUR	Oscillator Warm Up Register	0xC00
CPPUR	Clock Pad Power Up Register	0xC04
TCLR	TEP Clock Calibration Length Register	0xC08
*** Reserved ***		0xC0C
TESR	TEP Exception Status Register	0xC10
ITESR	Interruptable TEP Exception Status Register – equals TESR & TEIMR	0xC14
TEIMR	TEP Exception Interrupt Mask Register	0xC18
TCR	TEP Control Register	0xC1C
The Sequencer 0 register-file ⁶		0xC5F-0xC20
The Sequencer 1 register-file ⁶		0xC9F-0xC60
The Sequencer 2 register-file ⁶		0xCDF-0xCA0
The Sequencer 3 register-file ⁶		0xD1F-0xCE0
The Sequencer 4 register-file ⁶		0xD5F-0xD20
*** Reserved ***		0xE1F-0xD60
TACR	TEP Absolute Counter Register	0xE20
*** Reserved ***		0xE2F-0xE24
TCRBR	TEP Calibration Read-Back Register	0xE30
*** Reserved ***		0xFFF-0xE34
512 x 26 SRAM area		0x17FF-0x1000
*** Reserved ***		-0x1800

Table 3: TEP internal memory map.

All addresses are given relative to the TEP base address given by the DPBUS system memory map.

The meaning of standard versus pipelined read access is that pipelined read can be with zero wait states. However the read data are from the field of the address of the previous read access. For this type of access there must be $1\frac{1}{2} \cdot (DSCLK/SysClk)$ cycles between two succeeding accesses, which is expected solved by means of *software pipelining*, on the benefit of not locking up the DPBUS.

The standard access generates additional wait cycles for at most $1\frac{1}{2} \cdot (DSCLK/SysClk)$ cycles while locking the DPBUS.

5.2 Register Description

More detailed description of each register will be given in below. Notice that 'x' in the descriptions means an unused bit – when read a '0' will be returned in such cases.

5.2.1 OWUR – Oscillator Warm-Up Register

Read/write register

Initial Value: 32'b0000_0000_0000_0000_0000_0011_1111_1111

Bit 31-10: Not used

Bit 9-0: Oscillator warm-up delay from in Clk32k cycles.

5.2.2 CPPUR – Clock Pad Power Up Register

Read/write register

Initial Value: 32'b0000_0000_0000_0000_0000_0000_0000_0001

Bit 31-10: Not used

Bit 9-0: Clock pad power-up delay from oscillator power-up in Clk32k.

5.2.3 TCLR – TEP Clock Calibration Length Register

Write-only register

Initial Value: 32'b0000_0000_0000_0000_0000_0000_1111_1111

Bit 31-16: Not used

Bit 15-0: Number of Clk32k cycles used for counting SysClk cycles during snapshot operation.

Notice, this register can only be changed safely at least two 32 kHz clock cycles prior to a snapshot operation starts.

5.2.4 TCCFR – TEP Clock Calibration Fraction Register

Read/write register

Initial Value: 32'b0000_0000_0000_0000_0000_0000_0000_0000

Bit 31-22: Not used

Bit 21-18: Integer part of fractional increment for clock calibration

Bit 17-0: Fractional part of fractional increment for clock calibration.

5.2.5 TCCMR – TEP Clock Calibration Modulus Register

Read/write register

Initial Value: 32'b0000_0000_0000_0000_0000_0011_1111_1111

Bit 31-10: Not used

Bit 9-0: Modulus argument of clock calibration

Notice, TCCMR will be altered rarely, whereas TCCFR may be altered often. TCCFR can be written any time. In case TCCMR is to be altered also, this register must be updated prior to updating TCCFR and the update will not be registered before TCCFR has been written.

5.2.6 TESR – TEP Exception Status Register

Read-only register – writing ‘1’s will clear the associated bits.

Initial Value: 32'b0000_0000_0000_0000_0000_0000_0000_0000
 Bit 31-20: Not used
 Bit 19-12: Associated Sequencer has been postponed due to DMA Request Resolver conflict.
 Bit 11-9: Not used.
 Bit 8-4: Associated Sequencer has been postponed due to Memory Access Resolver conflict.
 Bit 3: Concurrency limit has been reached
 Bit 2: I/O conflict on GPIO signal groups
 Bit 1: Snapshot conflict
 Bit 0: Illegal instruction has been executed.

5.2.7 iTESR – Interruptable TEP Exception Status Register

Read-only register

Initial Value: 32'b0000_0000_0000_0000_0000_0000_0000_0000
 Bit 31-20: Not used
 Bit 19-0: TESR[19:0] & TEIMR[19:0]

5.2.8 TEIMR – TEP Exception Interrupt Mask Register

Read/write register

Initial Value: 32'b0000_0000_0000_0000_0000_0000_0000_0000
 Bit 31-20: Not used
 Bit 19-0: ‘1’ = Enable interrupt for associated exception.

5.2.9 TCR – TEP Control Register

Read/write register

Initial Value: 32'b0000_0000_0000_0000_0000_0001_0000_0000
 Bit 31-29: Not used
 Bit 28: ForcePLL; disables powering off system PLL when set.
 Bit 27: ‘1’ = Select SysClk/N as the calibrated slow clock (CClk32k)
 Bit 26: ‘1’ = Enable interrupt generation when DPBUS interface is ready for new write data.
 Bit 25: ‘1’ = Enable interrupt when DPBUS interface contains valid pipe-lined read data.
 Bit 24-22: Periodic trigger #2 selection,
 ‘111’ = Absolute 24 bits counter, bit18.

...

	'001' = Absolute 24 bits counter, bit12.
	'000' = Absolute 24 bits counter, bit11.
Bit 21-19:	Periodic trigger #1 selection,
	'111' = Absolute 24 bits counter, bit15.
	...
	'001' = Absolute 24 bits counter, bit9.
	'000' = Absolute 24 bits counter, bit8.
Bit 18-17:	'1' = Enable snapshot trigger of associated periodic trigger.
Bit 16-14:	Not used.
Bit 13-9:	'1' = Enable snapshot trigger of associated Sequencer.
Bit 8:	ForceSysClk, disables powering off SysClk osc. when set.
Bit 7-5:	Not used.
Bit 4-0:	'1' = Soft-reset of associated Sequencer

5.2.10 TACR – TEP Absolute Counter Register

Read-only register

Initial Value:	32'b0000_00xx_xxxx_xxxx_xxxx_xxxx_xxxx
Bit 31-26:	Not used
Bit 25-0:	Value of 26 bits absolute counter

Notice, reading this register requires internal synchronization to the 32 kHz clock. This implies potentially long reading delay (in the range 0 to 30 μ s). As not to block the DPBUS during this time it is recommended only to read TACR in the pipelined address area - cf. section 5.3.1.

5.2.11 TCRBR – TEP Calibration Read-Back Register

Read-only register

Initial Value:	32'b0000_0000_xxxx_xxxx_xxxx_xxxx_xxxx_xxxx
Bit 31-24:	Not used
Bit 23-0:	Number of SysClk cycles +1, counted during last snapshot.

5.2.12 TSSR – TEP Single-Step trigger Register

Write-only register

Initial Value:	--
Bit 31-0:	Not used

Any writing will advance Sequencers execution until next time-tick if the HALT signal is being asserted. Otherwise accessing TSSR is ignored.

5.2.13 Sequencer Register-Files

Please refer to Table 2 for definition.

5.3 Software Recommendations and Scenarios

[Hint: Keep it simple, make it right - and it will work perfectly – unless there are bugs, of course]

5.3.1 Accessing the module

The TEP module is accessible as any other device located in the DPBUS system. All registers as well as RAM within the TEP memory map may be accessed by the host at any time; internal arbitration makes sure conflicts are solved by inserting sufficient additional wait cycles to the host/DSP access. However, it is under full SW responsibility to avoid contents conflicts, e.g. overwriting instruction stream data currently used by a Sequencer, etc.

The TEP module differs from the other DPBUS modules, however, in that it is 32-bits-device-only. Thus, any access to the module must always be done as a pair of 16 bits DPBUS access, which when is mapped into a single 32 bits access at the TEP DPBUS interface.

Notice, every register/RAM is word aligned. To read the entire contents the lower half-word can be read at the address and the upper half-word can be read next at the address + 2, and the reading must be done in that order. For writings the lower half-word must be written to the address and the upper half-word next written the address + 2, and again, in that order – **both** half-words must be written always.

As an example writing TCR register appears like (in pseudo C)

```
Int16 TCR_lower = *((long) (TEPBaseAddr + TCR));
Int16 TCR_upper = *((long) (TEPBaseAddr + TCR + 2));

TCR_lower = <the lower 16 bits of the value to write>;
TCR_upper = <the upper 16 bits of the value to write>;
```

The data will not be transferred to TCR before the second writing has taken place. Once this has happened data are transferred internal TEP to the TCR (in this case) register. This transfer happens at the TEP clock (SysClk) speed and further may needs arbitrated with other internal transfers, e.g. Sequencers. The TEP module does not accept another write (nor a read) from the DPBUS before the internal transfer has completed. An attempted write operation in this case will be stalled by means of additional wait-states insertion. To avoid this situation, which will block the DPBUS for a long time (1½-2 SysClk cycles), either SW must ensure sufficient delay between consecutive writes or use interrupt. The TEP DPBUS interface can optionally generate an interrupt, TEPRdyIRQ, upon completion of the write, signaling the interface is ready for new data, cf. TCR register.

Reading access to TEP must similarly be done as two consecutive 16 bits half-words reads. E.g.

```
Int32 ReadRegister;

ReadRegister = TCR_lower;
ReadRegister |= (TCR_upper<<16);
```

The reading will be initiated by the first reading and hence, will block the DPBUS by inserting additional wait cycles until the data are available at the TEP bus interface. However all 32 bits are read and stored, which means the second reading (of the upper half-word) can take place afterwards without additional wait cycles.

To avoid the blocking of the DPBUS and the processor during a read, which is slow compared to the DPBUS speed ($1\frac{1}{2}$ -2 SysClk cycles), registers can be read via the pipelined address space. E.g. TCR has address 0x01C for normal read and write and address 0xC1C for pipelined read – both are offered for reasons of flexibility, cf. Table 3.

‘Pipelined’ refers to the fact that a read from TEP in this address area returns the contents from a previous read at the same or any other address.

```
Int16 pTCR_lower = *((long) (TEPBaseAddr + pipelined_TCR));
Int16 pTCR_upper = *((long) (TEPBaseAddr + pipelined_TCR + 2));
Int16 pTESR_lower = *((long) (TEPBaseAddr + pipelined_TESR));
Int16 pTESR_upper = *((long) (TEPBaseAddr + pipelined_TESR + 2));

Int32 ReadRegister;

ReadRegister = pTCR_lower;
ReadRegister |= (pTCR_upper<<16);
/* ReadRegister now contains the contents from the previous read */

ReadRegister = pTESR_lower;
ReadRegister |= (pTESR_upper<<16);
/* ReadRegister now contains the contents of TCR */
```

For this type of access there must be $1\frac{1}{2}$ ·(DSCLK/SysClk) cycles between two succeeding accesses, otherwise additional wait cycles will be inserted. The TEP DPBUS interface can optionally generate an interrupt, TEPRdyIRQ, upon completion of the internal read, signaling data are ready to be read, cf. TCR register.

Notice, the two half-word accesses for a full read or write access are unbreakable. I.e. in between the two half-word accesses the TEP DPBUS interface does not allow any other access. This suggests a software interface design featuring one single driver implicitly guaranteeing this behavior may handle all access to the TEP.

5.3.2 Configuration

The Configuration of the TEP basically consists of downloading code and data to its internal RAM for the Sequencers and setup of various registers, common as well as individual for each Sequencer.

5.3.2.1 Downloading

Downloading code is done as any other writing access to the module. Of course, SW must ensure that currently executing code is not being overwritten without proper handling. Reading back the program counters of currently executing Sequencers may assist in this matter.

In the real system the instruction streams are assumed generated/or modified by the L1 SW as they as requested and then downloaded to TEP – fully or partially. However for the test case (system as well as chip) a very simple assembler has been made easing the writing of test code.

A simple *Perl*-based assembler is available for the engine, which follows the conventions of other common assemblers. Each instruction may contain optional label, which must end with a ‘:’, followed by one or more spaces. The label may contain any alphanumerical character and ‘_’. The opcode then follows, also followed by an optional number of spaces, and lastly the operand of the

instruction, if any. The opcodes and operands are stated as in the description above, cf. Table 1. Each instruction line must be terminated with a semi-colon. Any characters on a line after a semi-colon character are ignored, and blank lines are permitted. The assembler supports the instruction set of the engine, as already described. In addition, the assembler supports a few pseudo-operations, as listed below.

ORG *N*;

The assembler's program counter is loaded with the value of *N* (which must lie between 0 and 255 inclusive). The format of *N* accepted is unsigned integers as decimal or hex format. Subsequent instructions and data are then assembled into memory starting from location *N*.

If no ORG statement is present in the beginning of the source file, the default start address is 0x00.

DW *N*[*N*1, *N*2, *N*3, ...];

The current location in the memory contains the literal value *N* and the current location moves on to the next location. The format of *N* is the same as that for the ORG instruction described above plus constants, except it will be treated as 16 bits figure.

CONST *IDENTIFIER* = *N*;

The compile-time identifier is assigned the value *N* and each occurrence of the *IDENTIFIER* in the code, will be substituted by *N* after parsing all source files and before code generation. The format of *N* follows that of ORG and the naming of *IDENTIFIER* follows that of labels.

The Perl source files,

File Name	Description
TEPasm.pl	Top level file
code.pl	Contains functions for object code and output file generation
misc.pl	Contains various small functions used throughout.
rdassem.pl	Contains functions for parsing assembler files
log.pl	Contains functions for opening, closing and writing a log file.

must be placed in one directory. The assembler is invoked according to,

Usage:

```
/<Install directory path>/TEPasm.pl [options] file1 file2 ...
```

Options:

```
-h          display this help
-l logfile  specifies log file. (default: tep.log)
-o outfiles specifies output files (default: tep)
-v          verbose mode
```

Invoking the assembler generates three files,

- ❑ <outfiles>.obj contains the object code in 'Verilog simulation suited' form.
- ❑ <outfiles>.dat contains the same object code data in a form suited for including into a C code compilation.
- ❑ A log file summarizing the information.

5.3.2.2 *Initiating Execution*

For the initialization (e.g. at power up) the TEP RAM is blank and all Sequencers are in idle. For this case the sequence of tasks for the host processor to conduct will be

- ❑ Generate (or fetch from memory) appropriate instruction streams for the appropriate amount of Sequencers and download it to the TEP RAM.
- ❑ Setup Reset and interrupt vectors for each Sequencer, cf. RstVecReg, IntVecReg.
- ❑ Setup any other optional registers needed, e.g.
 - Setup interfacing modules like DSP DMA, GSP's, MMI module, etc.
 - Setup exception interrupt enable, cf. TEIMR
 - Enable interrupt of Sequencers if needed, cf. TCR
 - Selection of periodic calibration, cf. TCR
 - Etc.
- ❑ Generate soft-reset via TCR to each of the Sequencers to wake and initiate their code execution.

----- END ----- END -----

DMA Controller Chaining Feature

Chaining DMA channels provides a possibility for using the transfer-completion interrupt from one channel to initiate another DMA channel. The particular usage of this is for implementing an A/B buffer system, e.g. for servicing a serial communication channel like an UART. The A/B buffer system provides relaxed timing constraints for post processing data and decreased risks for simultaneously working on same set of data. The idea is that a "normal" DMA channel performs a pre-programmed transfer, and once this channel finishes its transfer, the chaining channel takes over automatically and performs its pre-programmed transfer. Up to the point where the "normal" channel ends its transfer, the chaining feature does not have an impact on the DMA transfer. Upon the chaining channels transfer-completion, the transfers have either come to an end or, in case the chaining channel has been re-enabled, another transfer is performed by the chaining channel. Only one chaining channel is available for each "normal" channel, and the chaining channel cannot perform a transfer unless a "normal" channel calls the chaining channel after its own transfer has come to an end. The execution flow is shown in the following figure.

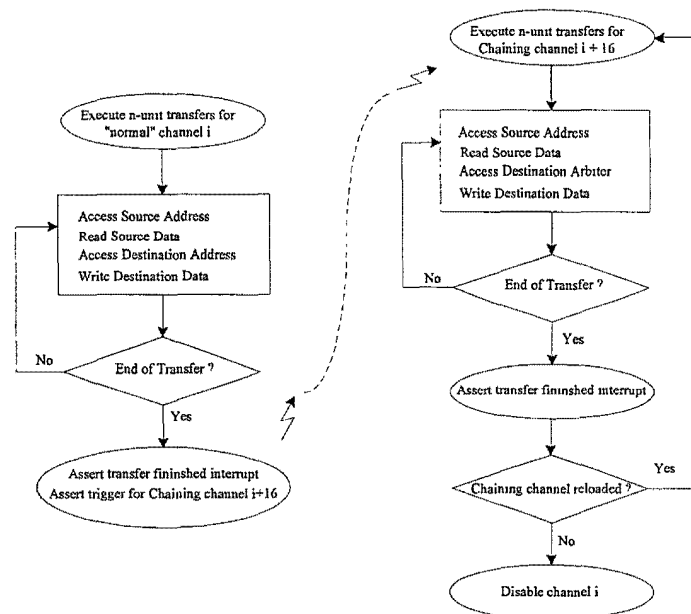


Figure 1: Execution flow for a transfer in chaining mode.

It is possible to re-program the chaining channel (once the chaining transfer has begun) to perform another transfer as soon as the first chaining transfer has ended. The scheme behind this is provided by effectively moving the chaining channel configuration to the "normal" channel configuration, which allows for the chaining channel to be re-programmed even though it is doing transfers. In case of a higher priority channel interrupt, the status of the chaining transfer is stored in the "normal" channel configuration area. This is vital for resuming a transfer after the higher priority interrupt and allows the interface, and especially the interrupt interface to be transparent regarding to the chaining feature. A "normal" channels corresponding chaining channel will therefore always have the same priority level as the "normal" channel has.

A chaining channel performs interrupt driven transfers only and supports all unit transfers (byte, halfword, word, longword) and all the "normal" channel types of transfers as described in sec. 4.3. in *Monaco DMA Controller Module Spec.03.doc*.

SECRET

a

Monaco MCU/DSP - DMA Controller Module Specification

File name: Monaco DMA Controller module spec.0.2.doc

Status: Draft

Author: Flemming Baltzersen, ADI-DK

Revision: 0.2

When	Who	Revision	What
Aug 06, 2001	FBA	Rev 0.1	First 'more or less' complete revision.
Aug 15, 2001	FBA	Rev 0.2	Document ready for review.

a

Table of contents

1.	Introduction	4
1.1	Overview	4
1.2	References	4
1.3	Terminology	4
2.	Background	5
3.	DMA Architecture Overview	6
3.1	DMA Functional View	7
3.2	Interface	9
3.2.1	Interface to the Arbiters	10
3.2.2	Interface to the PBUS/DPBUS	11
3.2.3	Interrupt Controller Interface	11
4.	Functionality	13
4.1	Address Increment and Wrap	13
4.2	Transfer Register	14
4.3	Transfer Modes	15
4.3.1	Circular, Incremental-to-Fixed transfer in Go mode	16
4.3.2	Circular, Incremental-to-Incremental transfer in Go mode	16
4.3.3	Circular, Fixed-to-Incremental transfer in Go mode	17
4.3.4	Linear, Incremental-to-Fixed in Go mode	17
4.3.5	Linear, Incremental-to-Incremental in Go mode	17
4.3.6	Linear, Fixed-to-Incremental in Go mode	18
4.3.7	Linear, Fixed-to-Fixed transfer in Go mode	18
4.3.8	Linear, Incremental-to-Fixed in Stop mode	18
4.3.9	Linear, Incremental-to-Incremental in Stop mode	19
4.3.10	Linear, Fixed-to-Incremental in Stop mode	19
4.4	DMA Interrupt Driven	19
4.5	Number of DMA Channels	20
4.6	Interrupt registers	20
4.7	Chaining Mode	22
4.7.1	Using the Chaining Mode	22
4.8	ImmediateTx Mode	24
4.9	TEP Interface	25
4.10	Channel Programming	26
4.10.1	ChanSrcHi and ChanDestHi	29
4.10.2	ChanSrdLo and ChanDestLo	29
4.10.3	ChanConfig	29
4.10.4	NotifyAddrIO and ChanN	30
4.10.5	NotifyAddrMem	30
4.10.6	IRQOut and IRQOutClr	31
4.10.7	SysEx	31
4.10.8	ActivChan	31
5.	User Guide	31
5.1	Performing a Memory to Memory Block Transfer	33
5.2	Configure an IO Transmit Buffer	35
5.3	Configure an IO Receive Buffer	37
5.4	Configure an Chaining transfer	40
5.5	DMA View of Monaco Memory Map	43
6.	Register description	45

6.1.1	Reset – Reset address accessible from PBUS	45
6.1.2	IRQOut[15:0] – Interrupt output register	45
6.1.3	IRQOutClr[15:0] – Interrupt output Clear register.....	45
6.1.4	SysEx – System Exception register	45
6.1.5	ActivChan[5:0] – Active Channel Number	45
6.1.6	ChainingReg[15:0] – Chaining Mode Enable Register	46
6.1.7	ChaniSrcLo[15:0] – Lower HalfWord of Source Address.....	46
6.1.8	ChaniSrcHi[12:0] – Upper HalfWord of Source Address.....	46
6.1.9	ChaniDestLo[15:0] – Lower HalfWord of Destination Address	46
6.1.10	ChaniDestHi[12:0] – Upper HalfWord of Destination Address	46
6.1.11	ChaniConfig[6:0] – Channel Configuration register	47
6.1.12	ChaniNotifyAddrMem[14:0] – Linear Transfer Notify Address register	47
6.1.13	ChaniNotifyAddrIO[10:0] – Circular Transfer Notify Address register.....	47
6.1.14	ChaniN[3:0] – Circular Buffer Register	47

TCL 1022

1. Introduction

1.1 Overview

This document describes the MCUDMA controller module and the DSPDMA Controller module, which both are modules of the Monaco Baseband processor. It has been chosen to describe both of these modules in the same specification since they are very close to identical. The purpose of this document is to provide the reader with a basic understanding of the functionality of the DMA controller modules and to some extent provide a more technical insight of the design of the DMA modules. The document also provides a users guide for setting up and using the various modes and features of the DMA controllers.

The DMA controller modules for Monaco are based on the DMA controller module featured in the AD6522 (Hercules) Baseband processor and the AD6528 (Hermes) Baseband processor. Rather than providing a lot of references to these Specifications ([2] and [3]) for the common features shared by the DMA controller modules, the descriptions from [2] and [3] has been integrated into this specification, which therefore provides a standalone description of the Monaco DMA controller modules.

1.2 References

- [1] MSP430 DMA Module Requirement and Module Specification. Rev. 1.1. April 23, 2000.
- [2] MSP430 DMA Module Requirement and Module Specification.
- [3] DMA Controller module Specification. Rev. 0.2, August 2001
- [4] Note on DMA features.rev0.4.doc. Rev. 0.4. June 1. 2001
- [5] Hercules Bus Arbitration specification. Rev. 1.1, 06/17/99.
- [6] AD6522 Digital Baseband Processor Specification. Rev. 1.7b, Dec. 22, 2000
- [7] Monaco Digital Baseband Processor Design Specification. Rev. 02, July 23, 2001
- [8] Monaco TEP specification.rev0.9.doc. Rev. 0.9, Aug. 06, 2001

1.3 Terminology

Hercules	AD6522 Baseband Processor from the AD20msp430 chipset
Hermes	AD6528 Baseband Processor. Evolution of the AD6522.
MicroSM™	Module in charge of providing low level system timing. Trademark of TTP Communications Ltd.
Word	A 32 bit data element
HalfWord	A 16 bit data element
Byte	An 8 bit data element

2. Background

The main functionality of the DMA controllers is to relieve the MCU (and DSP) from repetitive operations such as servicing IO devices or performing transfers from one memory block to another. The DMA controllers featured in the Monaco Baseband processor supports all the features and transmission modes provided by the Hercules and Hermes DMA controllers plus some additional functionality.

The DSPDMA Controller is capable of performing the same kind of bus master functionality on the DPBUS, SysL2BUS, DSP Core Ports, and PBUS, EBUS, RBUS (via the DSPBUS) as the MCU and the DSP. Likewise, the MCUDMA Controller is capable of performing the same kind of bus master functionality on the EBUS, RBUS, PBUS, and DSP Sub system (via the RBUS arbiter) as the MCU and the DSP. A simplified view of the Monaco DMA controller system is shown of Figure 1.

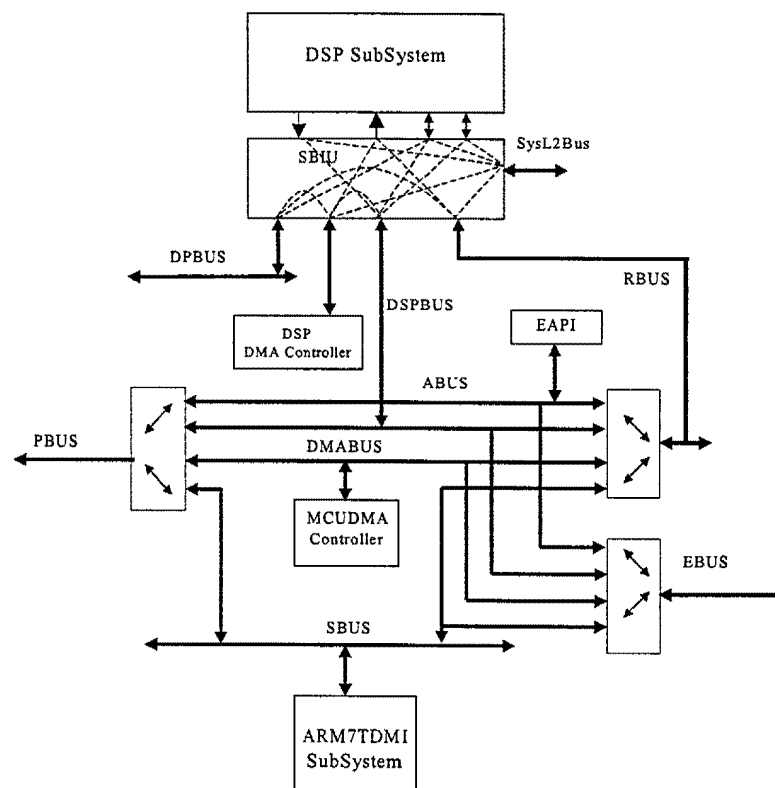


Figure 1: Simplified view of the Monaco DMA controller system.

Both of the Monaco DMA controllers supports the following features

- Interrupt driven transfers only
- 32 channels, 16 normal channels and 16 chaining mode channels
- Byte, Halfword, and Word transfers
- LongWord transfers (for DSPDMA only).
- Linear and Circular transfer
- All kinds of transfer (Mem-to-Mem, Mem-to-IO, IO-to-Mem, IO-to-IO)
- Immediate transfer where data is placed in the Address Source field

- Maximum linear transfer block size of 32 Kbytes
- Maximum circular transfer buffer size of 2 Kbytes
- Chaining mode between channels
- Full 32 bit address space accesses
- Interface, which provides the TEP with the same master functionality as the MCU and the DSP (for DSPDMA only!)
- Possibility of interrupting transfer by higher priority channel transfer
- Early warning for both linear and circular transfers
- "Transfer finished" indicator
- Support for IO devices
- Possibility to program channel at runtime

Thus, the only thing that is different between the MCUDMA and the DSPDMA is the LongWord transfer and the TEP interface. These features are available in the DSPDMA but not in the MCUDMA. The possibility to perform LongWord transfers in the DSPDMA is used when moving a whole 32 bit sequence in one transfer (as opposed to Word transfers where the transfer is performed HalfWord-HalfWord wise. In a Word transfer it thus takes twice as long to move a 32bit sequence as in LongWord mode). This also means that the data bus interface in the DSPDMA is 32 bit wide but only 16 bit wide in the MCUDMA. It should be mentioned that while access to the internal DMA registers (and thereby also the channel configuration) is performed via the PBUS for the MCUDMA, the access to the DSPDMA is done via the DPBUS. Also note that while the MCUDMA is placed in the BClk domain, the DSPDMA is placed in the faster DSClk domain (see [7]).

3. DMA Architecture Overview

The DSP/MCU DMA controllers consists of four major blocks

1. • For the MCUDMA: A module for interfacing to the PBUS
 • For the DSPDMA: A module for interfacing to the DPBUS
2. • For the MCUDMA: A module for interfacing and synchronizing to the MCU Interrupt controller
 • For the DSPDMA: A module for interfacing and synchronizing to the DSP interrupt Controller
3. • For the MCUDMA: A module for generation -and control of the signals for the DMABUS
 • For the DSPDMA: A module for generation -and control of the signals for the DABUS
4. • For the MCUDMA: A module for synchronizing the PBUS signals/data with the DMABUS signals/data
 • For the DSPDMA: A module for synchronizing the DPBUS signals/data with the DABUS signals/data

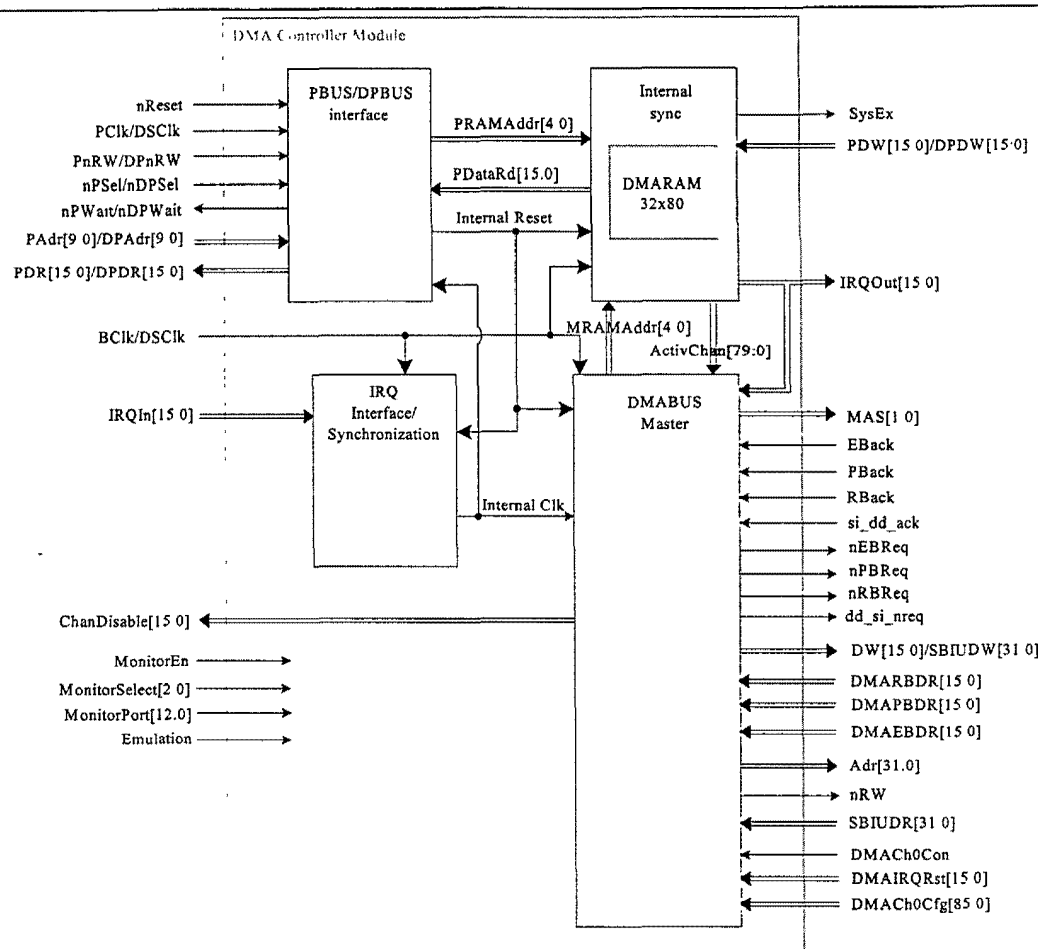


Figure 2: The internal submodules of the MCUDMA/DSPDMA controllers and the interfacing signals. The naming convention used is: MCUDMA/DSPDMA. The TEP interface signals *DMACH0Con*, *DMAIRQRst[15:0]*, and *DMACH0Cfg[85:0]* are valid for the DSPDMA only. The same goes for the SBIU request and acknowledge signals (*dd_si_nreq* and *si_dd_ack*) and the data read from SBIU (*SBIUDR[31:0]*). The data read from the PBUS, EBUS, and RBUS (*DMARBDR[15:0]*, *DMAEBDR[15:0]*, *DMARBDR[15:0]*) are valid for the MCUDMA only.

3.1 DMA Functional View

There are two possibilities for the DMA Channel Register storage: a local RAM inside the DMA Module or System RAM external to the DMA. Using the System RAM to store the channel configuration will lead to reading 6 x 16 bit values and storing 6 x 16 bit values whenever the DMA changes to processing a different channel. With a worst case access time to the System RAM of 320 ns, a DMA cycle in this scenario will be more than 3.8 μ s. Both the DMA controllers therefore provide a single-port 86 bit wide RAM with 32 locations (where channel 0-15 is normal channels and channel 16-31 is chaining channels) local to the DMA, which will be used to store the DMA channel configurations.

The overall functional architecture of the DMA controllers is shown in Figure 3. Each channel requires 86 bit of storage, which is stored in the 86 bit single port RAM. When a channel becomes active, on receiving its corresponding *IRQIn*, the 86 bit channel word is read from the RAM into the

active channel register. During the transfer, both *ChanSrc* and *ChanDest* are modified locally to the active channel according to the type of transfer.

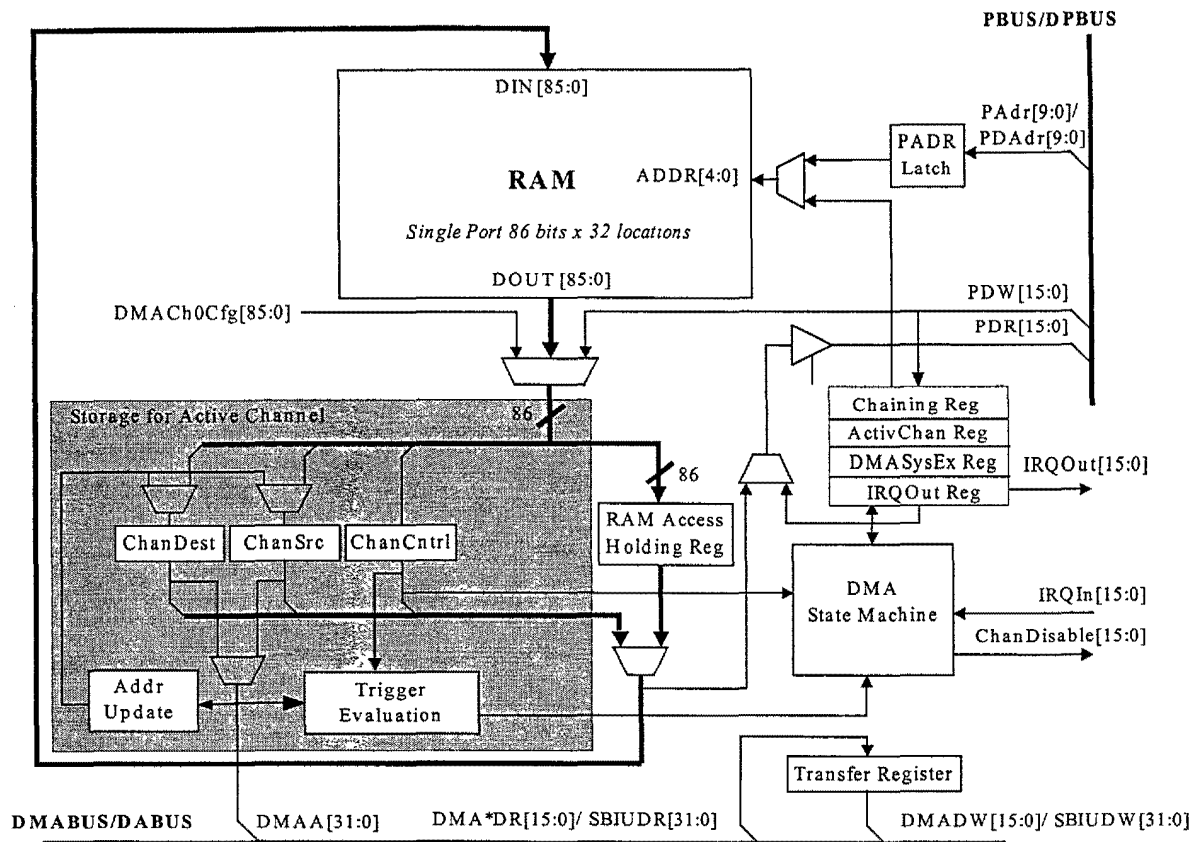


Figure 3: Simplified architecture for the MCUDMA function. Note that the MCUDMA and the DSPDMA does differ in the width of some interfacing signals. The naming convention used is: MCUDMA/DSPDMA in these cases. Also note that the muxed signal *DMACh0Cfg[85:0]* is available in the DSPDMA only.

A write back to the RAM, to update *ChanSrc* and *ChanDest*, only occurs when another channel interrupts with a higher priority. This minimizes RAM access and thus saves power. A DMA read cycle results in data being held in the *transfer register* before being written by the DMA write cycle. During normal DMA operation, a read or a write operation to the RAM to configure a channel can occur freely; even the currently active channel can be modified. In this case the active channel registers are read or written directly with zero additional PBUS wait states (after any arbitration wait states). A PBUS (or DPBUS) read access to a currently inactive channel would stall the DMA for one access if the DMA also needs to access the RAM in that cycle, but will otherwise result in zero PBUS (or DPBUS) wait states.

A PBUS (or DPBUS) write access to a currently active channel will result in a write to the appropriate bits of the *RAM access holding register*. In the following cycle, any PBUS (or DPBUS) access will see a wait state; this is because the DMA will write back the *RAM access holding register* into the RAM to complete the cycle.

3.2 Interface

The DSPDMA controller interfaces to the DPBUS, the DSPDMA interrupt controller, the TEP, and the SBIU. The MCUDMA controller interfaces to the SysDMA Interrupt controller module, the PBUS, and to the EBUS, RBUS, and PBUS arbiters through the DMABUS. The interfacing signals and their description are listed in the following table.

Signal Name	Direction	Polarity	Signal Description
PBUS/DPBUS Interface signals			
NReset	I	Low	Asynchronous hardware reset
PClk**/DPClk*	I	--	PBUS/PDBUS clock input
PnRW**/DPnRW*	I	--	PBUS/DPBUS read/write signal (read is active low)
nPSEL**/nDPSEL*	I	Low	DMA controller select signal
nPWait**/nDPWait*	O	Low	Wait signal for PBUS/DPBUS arbiter
PAdr[9:0]**/DPAAdr[9:0]*	I	--	PBUS/DPBUS address signals for internal mapping
PDW[15:0]**/DPDW[15:0]*	I	--	Data input from PBUS/DPBUS
PDR[15:0]**/DPDR[15:0]*	O	--	Data output to PBUS/DPBUS
Interrupt control interface signals			
IRQIn[15:0]	I	High	Enable channel signal from interrupt ctrl.
ChanDisable[15:0]	O	High	Disable channel signal to interrupt ctrl.
IRQOut[15:0]	O	High	Transfer finished signal to interrupt ctrl.
SysEx	O	High	System exception for Interrupt ctrl.
PBUS, EBUS, RBUS, SBIU Arbiter interface signals			
BClk**/DSClk*	I	--	System clock in MCU subsystem domain
MAS[1:0]	O	--	Size type of transfer (2'b00: Byte, 2'b01: HalfWord and Word, 2'b10: LongWord).
EBAck**	I	High	EBUS acknowledge from EBUS arbiter
PBAck**	I	High	PBUS acknowledge from PBUS arbiter
RBAck**	I	High	RBUS acknowledge from RBUS arbiter
si_ddack*	I	High	Acknowledge from SBIU
nEBReq**	O	Low	EBUS access request to EBUS arbiter
nPBReq**	O	Low	PBUS access request to PBUS arbiter
nRBReq**	O	Low	RBUS access request to RBUS arbiter
dd_si_nreq*	O	Low	SBIU access request
DW[15:0]**/SBIUDW[31:0]*	O	--	Data word to destination address
DMAEBDR[15:0]**	I	--	Data word from EBUS source address.
DMA PBDR[15:0]**	I	--	Data word from PBUS source address.
DMARBDR[15:0]**	I	--	Data word from RBUS source address.
SBIUDR[31:0]*	I	--	Data word from SBIU source address.
Adr[31:0]	O	--	Source or destination address
NRW	O	Low	Read/write signal for EBUS, RBUS, PBUS or SBIU arbiter (read is active low)
TEP Interface			
DMACH0Con*	I	High	Configuration signal from the TEP

Signal Name	Direction	Polarity	Signal Description
DMAIRQrst[15:0]*	I	--	Bit wise clear of the IRQOut register from TEP
DMACH0Cfg[85:0]*	I	--	Channel configuration data from TEP
<i>Test signals</i>			
MonitorEn	I	High	Enable signal for monitor mode
MonitorSelect[2:0]	I	--	Select of the various monitor modes
MonitorPort[12:0]	O	--	Output for monitored signals
Emulation	I	High	Enable signal for emulation mode

Table 1: The interface signals to/from the DMA controllers. The naming convention used is: MCUDMA/DSPDMA in case of mismatch between the controller names. A * marks a signal that is available on the DSPDMA only, and ** marks a signal that is available on the MCUDMA only.

3.2.1 Interface to the Arbiters

The DMAs mimics the MCU and DSPs memory interface on the DMABUS and DABUS. The DMABUS has a 32 bit address bus (Adr[31:0], and provides access by Byte, HalfWord, or Word memory quantities), and a 16 bit databus with separate data read and data write. The data read is done via either DMAEBRD[15:0] (read from EBUS), DMAPBRD[15:0] (read from PBUS), or DMARBRD[15:0] (read from RBUS) which is multiplexed internally. The data write is done via DW[15:0]. The following list shows the constraints when performing data transfers on a DMA channel in the MCUDMA.

- DMA transfers must have the same source and destination data size, i.e. Byte to Byte, Halfword to Halfword and Word to Word.
- A word transfer will be converted into two sequential Halfword transfers.
- The DMA is capable of reading bytes from DMA*BDR[15:8] on odd addresses and DMA*BDR[7:0] on even addresses. When writing a byte, the data will be placed on both DW[15:8] and DW[7:0].
- In chaining mode, each "normal" channel (i.e. channel zero to channel 15) has a chaining channel counterpart so that e.g. channel zero's chaining channel is channel 16 and the chaining channel for channel 15 is channel 31.
- PBUS transfers cannot use 32bit.

The DABUS has a 32 bit address bus (Adr[31:0], and provides access by Byte, HalfWord, Word or LongWord memory quantities), and a 32 bit databus with separate data read and data write. The data read is done via SBIUDR[31:0] and data write is done via SBIUDW[31:0]. The following list shows the constraints when performing data transfers on a DMA channel in the DSPDMA.

- DSPDMA transfers must have the same source and destination data size, i.e. Byte to Byte, Halfword to Halfword, Word to Word or LongWord to LongWord.
- A Word transfer will be converted into two sequential HalfWord transfers.
- A LongWord transfer is performed in a full 32 bit transfer.
- The DSPDMA is capable of reading bytes from SBIUDR[15:8], or SBIUDR[7:0] depending on the current address. When writing a byte, the data will be copied onto

SBIUDW[31:24] and SBIUDW[23:16] and SBIUDR[15:8] and SBIUDR[7:0] (see Figure 5).

- In chaining mode, each “normal” channel (i.e. channel zero to channel 15) has a chaining channel counterpart so that e.g. channel zero’s chaining channel is channel 16 and the chaining channel for channel 15 is channel 31.
- DPBUS transfers cannot use 32bit.

3.2.2 Interface to the PBUS/DPBUS

The DMAs setup are done by loading the control registers and the entries in the internal DMA RAM. This is performed from the PBUS/DPBUS side which allows for both the MCU and the DSP to setup the DMA controllers.

The DMAs conform to the PBUS/DPBUS system specification and uses a gated BClk/DSClk to clock internal circuitry. Since the PClk/DPClk and the BClk/DPClk has nominally zero skew, synchronous interaction is possible with caution. PBUS/DPBUS accesses are zero wait state for all reads and non back-to-back writes. Back-to-back writes requires 1 wait state.

The MCU and DSP can read and write the channel configuration registers including the currently active channel without having to disable it first. However, a write to the currently active channel should only modify channel control registers and not the transfer source and destination addresses. For example, *NotifyAddr* (see Table 5) might be changed dynamically on IO Tx channels after the MCU has added data to the Tx buffer area or on IO Rx channels after the MCU has removed data from the Rx buffer area.

If the MCU or DSP wishes to halt the DMA channel during a transfer, it must disable the channel interrupt by writing to the *ChanEnable* register in the MCU Interrupt controller (for MCUDMA transfers) or in the DSP interrupt controller (for DSPDMA transfers). If the MCU or DSP then wishes to use the halted channel for a new transfer and restore the halted transfer later, it must then poll the corresponding DMAs active channel register (*ActiveChan*) to determine when any out-going transfer unit of the halted channel has completed. When *ActiveChan* is no longer the channel just disabled, the MCU or DSP can then read out the halted channels configuration so that it can be restored later.

3.2.3 Interrupt Controller Interface

The MCUDMA IRQ Controller and DSPDMA interrupt controller each generates 16 unique ‘triggers’ that will initiate a transfer on the corresponding DMA controllers channel. The Interrupt controllers trigger the DMA controllers on a cycle-by-cycle basis and every transfer requires an interrupt. For Memory-to-Memory transfers, the DMA controller is only triggered once but for IO-to-Memory or Memory-to-IO, the DMA controller is triggered for every single unit transfer (i.e. Byte, Halfword, Word or LongWord). In the latter case (IO channel transfers), this will be sourced from a peripheral Tx or Rx interrupt (which includes DSPFlag and GPIO_IRQLevel interrupts), once the channel is enabled upon a TEP DMASWI trigger, or manually by setting the IRQ Controller Channel Enable register. In case of Memory-to-Memory channels, the trigger would be defined by selector input 0, which is hardcoded to always active, and enabled either upon a TEP DMASWI or manually by setting the IRQ Controller Channel Enable register.

The DMA controller monitors its input interrupt bus ($IRQIn[15:0]$) for interrupts. If there is any DMA interrupts asserted, the DMA controller will perform the transfer of the channel with the highest priority indicated by $IRQIn$. The DMA controllers interface to the Interrupt controller is shown in Figure 4.

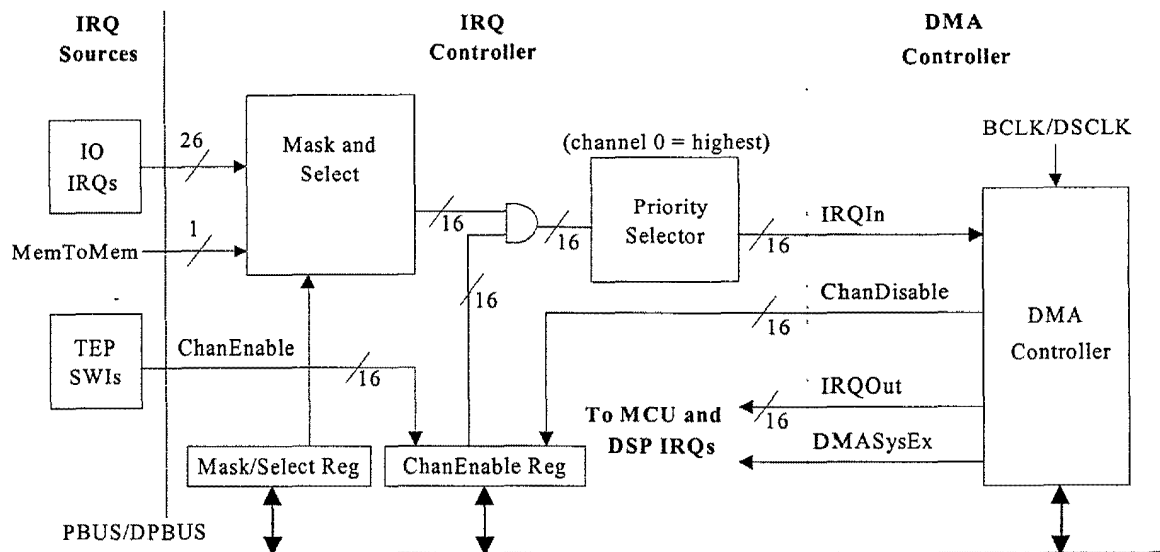


Figure 4 : DMA IRQ Interface

The 16 DMA channels have a hard-coded priority, channel 0 having the highest priority and channel 15 having the lowest. Since all DMA controller transfers are prioritised using a single priority encoder inside the Interrupt Controller, a MemToMem move with a high priority cannot be interrupted by an IO service with a lower priority. Therefore, MemToMem transfers would normally be performed using lower priority channels, with IO channels assigned to channel numbers based on maximum latency requirements.

When the DMA services the interrupt, the associated channel will become the active DMA channel. The DMA controller uses its output interrupt bus ($IRQOut[15:0]$) to indicate transfer completion to the MCU or DSP. The 16 DMA IRQ outputs are routed to both the MCU and DSP IRQ Controllers, such that the DMA channels in the MCUDMA can be used directly by the MCU and the DMA channels in the DSPDMA can be used directly by the DSP. It is possible to setup any DMA channel in the MCUDMA or the DSPDMA by both the DSP and the MCU. However, the MCU can only receive interrupt from the lowest priority channel in the DSPDMA, and the DSP cannot receive interrupts from any of the MCUDMA channels. Even though the possibility exists, the MCU should not perform DSPDMA transfers without extreme SW precautions. The reason for this is that since the DSPDMA channel[15] configuration can be performed by the MCU, the possibility exists that this will conflict with the DSPs configuration. If for example the DSP at any given time has configured the channel, and the MCU changes the same channel data to support its own needs, then the transfer setup no longer match what is expected by the DSP.

The *ChanEnable* register in the IRQ controller controls whether a channel is enabled or disabled and can be accessed via the PBUS/DPBUS. Additionally, the TEP interrupts (TEP SWIs) can enable a

channel and the DMA has the ability to disable a channel via this register, which it will do in any of the following scenarios:

- A Linear or Circular transfer in Stop mode has reached its *NotifyAddr* and the transfer therefore has finished.
- A Linear or Circular transfer in Go mode has completed, i.e. all data in buffer has been transmitted.
- Transfer one unit then stop (*TranOneStop*) is configured (synthesizer programming)
- Circular IO Rx/Tx Overrun, i.e. the DMA goes to set the *IRQOut* register bit, but it has already been set.
- A Chaining channel has finished its transfer. If chaining mode is enabled, the “normal” channel will not clear the *ChanEnable* register but give control to the chaining channel instead which then, upon its transfer completed, clears the *ChanEnable*.

Additionally, if Circular IO Tx Overrun occurs, the DMA controller will assert the DMA System Exception (*SysEx*). For circular IO Rx, this does not occur and by the action of the DMA disabling the channel, the UART will generate an Rx buffer overflow error. There is one DMA Exception interrupt from each DMA controller. The *SysEx* from the DSPDMA is attributable to the DSP only, and the *SysEx* from the MCUDMA is attributable to the MCU only.

The *ChanEnable* register values are ANDed with the IRQ source before entering the Priority Selector. The IO IRQ selector can be set to a logic ‘1’, i.e. permanently asserted, this is effectively a dummy interrupt source used for Memory to Memory block transfers only.

4. Functionality

The DMAs runs through the same transfer cycle for every single unit of data (Byte, HalfWord, Word, LongWord)

- Arbitration of the source bus system
- Read source value to internal transfer register
- Release source bus system
- Arbitration of the destination bus system
- Write data from internal transfer register to destination address
- Release the destination bus system

This transfer cycle is also independent of the transfer mode chosen (except the ImmediateTx mode). The available transfer modes are described in section 4.3.

4.1 Address Increment and Wrap

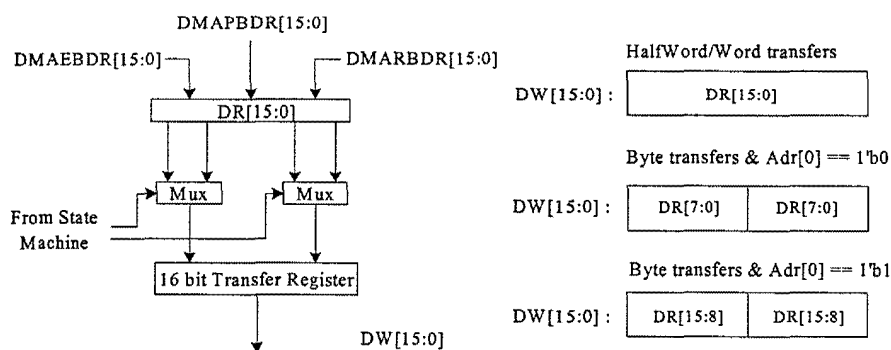
If a DMA channel is configured as either IO Tx or IO Rx (circular transfer), then the channel will model a FIFO of size $L=2^N$, where N is specified in the channel configuration register, see Figure 11. A restriction exist since the FIFO data must start at an address such that the N least significant bits of the address are zero. For the case of IO Rx, the *ChanDest* address will specify the write pointer in the FIFO and will auto-wrap when the N least significant bits of the *ChanDest* are all one, such that these bits become all zero. The same applies to *ChanSrc* for the case of IO Tx. The value of N is therefore decoded and used to modify the address incrementing operation by effectively breaking the carry-

chain at the appropriate point. In a similar way, only the N least significant bits of the Notify address will be used in the comparison with *ChanDest* for IO Rx or *ChanSrc* for IO Tx.

4.2 Transfer Register

The transfer register is a 16 bit wide register in the MCUDMA, and 32 bit wide for the DSPDMA. This register is used for temporary storage of the data between a DMA read and a DMA write cycle. In accordance with the MCU specification, the MCUDMA transfer register is able to place data from the EBUS, PBUS, or RBUS on both the upper and lower 8 bits of the databus when *TranSize* is defined as Byte. In case of a byte transfer on the DSPDMA, data will be copied onto SBIUDW[31:24], SBIUDW[23:16], SBIUDR[15:8], and SBIUDR[7:0]. The data transfer register for the MCUDMA and the DSPDMA is shown in Figure 5 in the upper and lower part respectively.

MCUDMA



DSPDMA

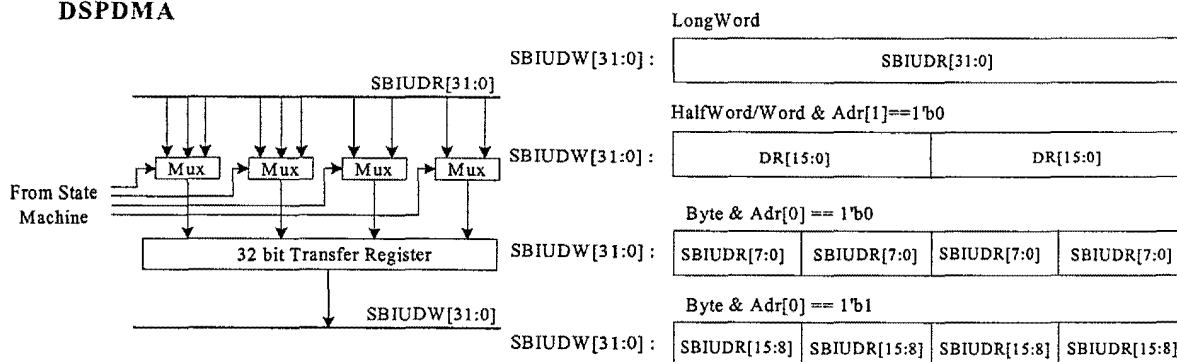


Figure 5: The data transfer register.

4.3 Transfer Modes

The Monaco DMA controllers provide more flexibility regarding transfer modes than the modes available in the Hercules and Hermes DMA controllers. The Monaco DMA controller support the following modes

- Any combination between fixed address mode (typically an I/O device) and incremental address modes (memory).
- “All” of these combinations supports Stop and Go modes.
- The incremental addressing modes supports Circular incremental mode (ring buffers and FIFO implementations) and Linear incremental addressing mode (“conventional” buffer addressing).

The combinations between fixed/incremental/circular/Linear transfer modes are shown in the following table. In this table *Incremental* refers to the source or destination being a memory area, *Fixed* refers to the source or destination being a fixed address, *Circular* refers to the source or destination being a buffer (of a size defined by the *ChanN* register value) that wraps around when the bottom address is reached, *Linear* refers to the source or destination being a memory area (of a size defined by the *NotifyMem* register value) that is accessed linearly, *Go mode* refers to the fact that the transmission continues even though the Notify address is reached, and *Stop mode* refers to the fact that the transfer stops as soon as the Notify address is met.

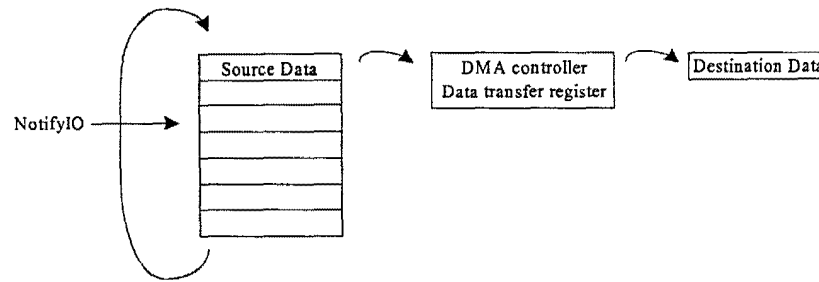
Where: '√' = valid, And: '÷' = invalid.		Incremental to Fixed address mode	Incremental to Incremental address mode	Fixed to Incremental address mode	Fixed to Fixed address mode
Circular	GO	√	√	√	÷
	STOP	÷	÷	÷	÷
Linear	GO	√	√	√	√
	STOP	√	√	√	÷

Table 2: Transfer modes supported by the DMA.

As can be seen from this table, some of the modes are marked as not supported. The reason for this is the following:

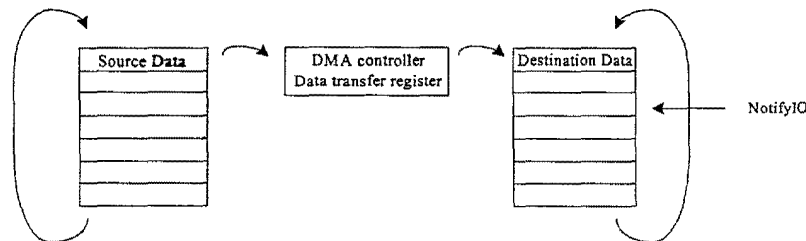
- In case of Incremental-to-Fixed Circular address mode there is no need for circular combined with stop mode since this is just Incremental-to-fixed in stop mode.
- In case of the Incremental-to-Incremental address mode, a Circular Stop mode is the same as a Linear Stop mode.
- In case of the Fixed-to-Incremental address mode, a Circular Stop mode is functionally the same as a Linear Stop mode.
- In case of the Fixed-to-Fixed address mode, a Circular Stop mode, a Linear Go mode or a Linear Stop mode will make no sense.

4.3.1 Circular, Incremental-to-Fixed transfer in Go mode



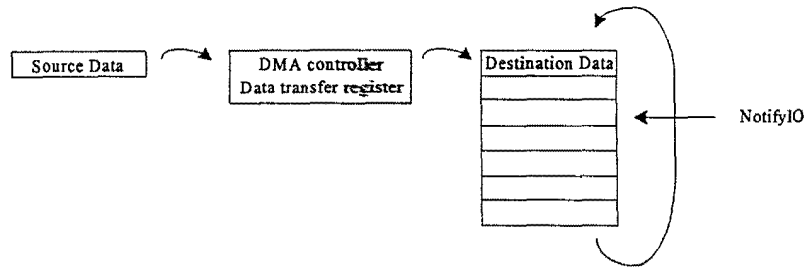
The Incremental to Fixed Circular transfer in Go mode can move a buffer of maximum 2kBytes to a fixed address. The transfer requires a single trigger for each unit transfer. The Notify address is sensitive to the Source address and is given by the *NotifyIO* address register. *IRQOut* is asserted when *NotifyIO* is met, but *ChanDisable* is not asserted.

4.3.2 Circular, Incremental-to-Incremental transfer in Go mode



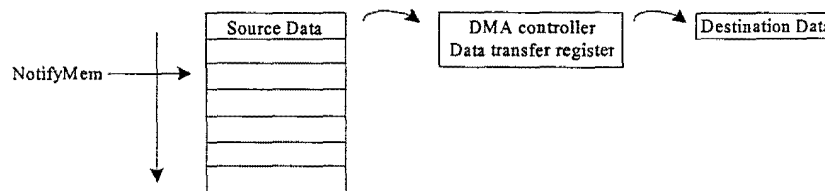
The Incremental-to-Incremental Circular transfer in Go mode can move a buffer of maximum 2kBytes to another buffer of the same size. The transfer requires a single trigger, which must be held during transfer and to finish transmission, the *TranOneStop* bit in the configuration register must be asserted. The Notify address is sensitive to the Destination address and is given by the *NotifyIO* address register. *IRQOut* is asserted when *NotifyIO* is met, but *ChanDisable* is not asserted.

4.3.3 Circular, Fixed-to-Incremental transfer in Go mode



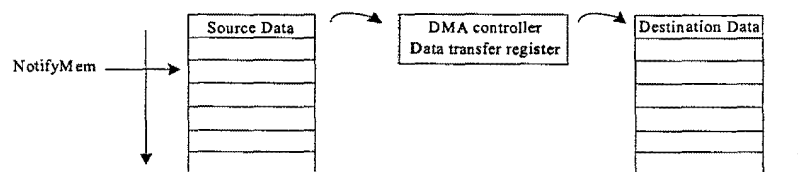
The Fixed to Incremental Circular transfer in Go mode moves data from a fixed address to a buffer of maximum 2kBytes. The transfer requires a single trigger for each unit transfer. The Notify address is sensitive to the Destination address and is given by the *NotifyIO* address register. *IRQOut* is asserted when *NotifyIO* is met, but *ChanDisable* is not asserted.

4.3.4 Linear, Incremental-to-Fixed in Go mode



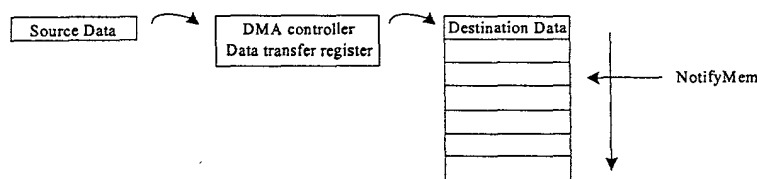
The Incremental to Fixed Linear transfer in Go mode can move a block of data of maximum 32kBytes to a fixed address. The transfer requires a single trigger and to finish transmission, the *TranOneStop* bit in the configuration register must be asserted. The Notify address is sensitive to the Source address and is given by the *NotifyMem* address register. *IRQOut* is asserted when *NotifyMem* is met and the *ChanDisable* is asserted

4.3.5 Linear, Incremental-to-Incremental in Go mode



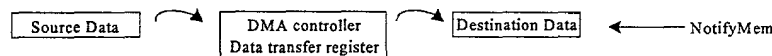
The Incremental to Incremental Linear transfer in Go mode can move a block of data of maximum 32kBytes to another memory block of the same size. The transfer requires a single trigger, which must be held during transfer and to finish transmission, the *TranOneStop* bit in the configuration register must be asserted. The Notify address is sensitive to the Destination address and is given by the *NotifyMem* address register. *IRQOut* is asserted when *NotifyMem* is met, but *ChanDisable* is not asserted.

4.3.6 Linear, Fixed-to- Incremental in Go mode



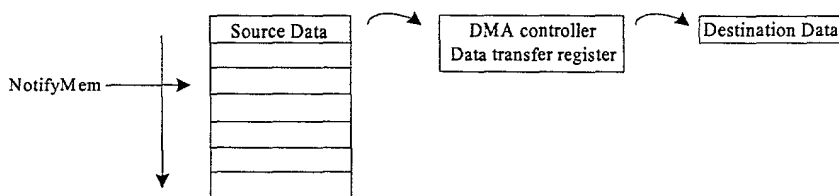
The Fixed to Incremental Linear transfer in Go mode can move data from a fixed address to a block of data of maximum 32kBytes. The transfer requires a single trigger for each unit transfer. The Notify address is sensitive to the Destination address and is given by the *NotifyMem* address register. *IRQOut* is asserted when *NotifyMem* is met, but *ChanDisable* is not asserted.

4.3.7 Linear, Fixed-to-Fixed transfer in Go mode



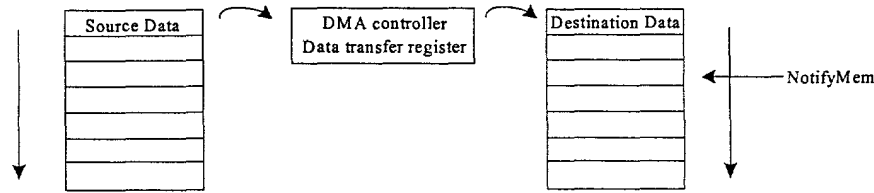
The Fixed to Fixed Linear transfer in Go mode moves data from a fixed address to another fixed address. The transfer requires a single trigger for each unit transfer. Even though the address is fixed, the *NotifyMem* address register needs to define the Notify address, which in this mode always must be the Destination address. *IRQOut* is asserted when *NotifyIO* is met, but *ChanDisable* is not asserted.

4.3.8 Linear, Incremental-to-Fixed in Stop mode



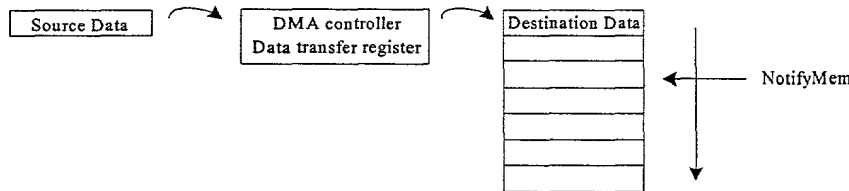
The Incremental to Fixed Linear transfer in Stop mode can move data from a block of memory of maximum 32kBytes to a fixed address. The transfer requires a single trigger for each unit transfer. The Notify address is sensitive to the Source address and is given by the *NotifyMem* address register. *IRQOut* is asserted when *NotifyMem* is met and the *ChanDisable* is asserted

4.3.9 Linear, Incremental-to-Incremental in Stop mode



The Incremental to Incremental Linear transfer in Stop mode can move data from a block of memory to another block of memory, both of maximum 32kBytes. The transfer requires a single trigger, which must be held during transfer and transfer is stopped when the Notify address is met. The Notify address is sensitive to the Destination address and is given by the *NotifyMem* address register. *IRQOut* is asserted when *NotifyMem* is met and the *ChanDisable* is asserted

4.3.10 Linear, Fixed-to-Incremental in Stop mode



The Fixed to Incremental Linear transfer in Stop mode can move data from a fixed address to a block of memory of maximum 32kBytes. The transfer requires a single trigger for each unit transfer and the transfer is terminated when the Notify address is met. The Notify address is sensitive to the Destination address and is given by the *NotifyMem* address register. *IRQOut* is asserted when *NotifyMem* is met and the *ChanDisable* is asserted

4.4 DMA Interrupt Driven

The DMAs are totally interrupt driven, such that each unit of transfer on all channels must be initiated by an interrupt. This applies to all kinds of transfers and the Memory-to-Memory transfers are therefore self-sustaining. This gives the IRQ controller total control over the scheduling of DMA operations. The priority associated with a channel will have a direct influence on the latency seen by that channel and the DMA controllers therefore have a hard-coded transfer priority.

Due to this hard-coded channel priority, the software has the responsibility of allocating the appropriate channel according to response time requirements for that function. The software must decide into which priority category the function belongs. The following list shows one method for categorizing function priorities:

High priority:	Absolute Response Time	(ART)
	As Fast As Possible	(AFAP)
Low Priority:	Average Response Time	(AVRT)

Additionally, it is likely that memory to memory moves would be placed onto low priority channels such that these operations do not consume all of the available DMA bandwidth.

4.5 Number of DMA Channels

Based on the utilization shown in Table 3, with a small overhead for design margin, the total MCUDMA channel requirement is 16 channels. The extra channels will allow for some flexibility in the software framework, which should avoid the situation where the MCU is forced to perform blocks moves itself. Note that the actual number of channels is 32 since the chaining mode provides an additional 16 channels (see section 4.7).

Channel(s)	Channel Function	Function Priority Class
8	4 Full-Duplex GSP UARTs	Absolute Response Time (ART)
2	SIM	Absolute Response Time (ART)
1	Memory copy for MCU	As Fast as Possible (AFAP)
1	Memory Mapped Display	Average Response Time (AVRT)

Table 3: Typical DMA Channel Utilization

The DSPDMA also provides a total of 32 channels. Given this amount of channels, there should be little reason to why the DSP should use any of the MCUDMA channels and likewise for the MCU to use any of the DSPDMA channels. This however, requires that the DMAs have access to all possible memory areas. This is indeed the case (cf. [7]).

4.6 Interrupt registers

The following table shows the interrupt controller registers that have to do with the DMAs interrupt generation in the DSP interrupt controller and the MCU interrupt controller.

Address	Register	R/W	Description
0x8000 0000*	IRQConfig	R/W	Write to this to assign a priority level onto a DMA output interrupt during transfer. Read it to determine which interrupt group has occurred during ISR.
0xF000 0000**			
0x8000 0100*	IRQActiveInterruptReg	R/W	Write to this to clear an IRQOut output interrupt. Part of ISR.
0xF000 0100**			
0x8000 0180*	IRQCurrentInterruptReg	R/W	Read this via the priority encode to determine which interrupt priority has occurred. Part of ISR.
0xF000 0180**			
0x8000 0200*	IRQHWMask	R/W	Mask register base for IRQOut interrupts in the interrupt controller. Use this to unmask DMA IRQOut interrupts.
0xF000 0200**			
0x8000 0280*	IRQHWStatus	R/W	Read this to determine which interrupt has occurred. Part of ISR.
0xF000 0280**			
0x8000 0700*	IRQChan0Config[4:0]	R/W	These 16 registers (1 per DMA channel) select the input interrupt source for the respective DMA channel. A value of 0 selects MemToMem. See Interrupt Controller spec. for more details.
0xF000 0700**			

4.7 Chaining Mode

Chaining DMA channels implies that the transfer-completion interrupt from one channel initiates another DMA channel. The particular usage of this is for implementing an A/B buffer system, e.g. for servicing a serial communication channel like an UART. The execution flow is shown in Figure 7.

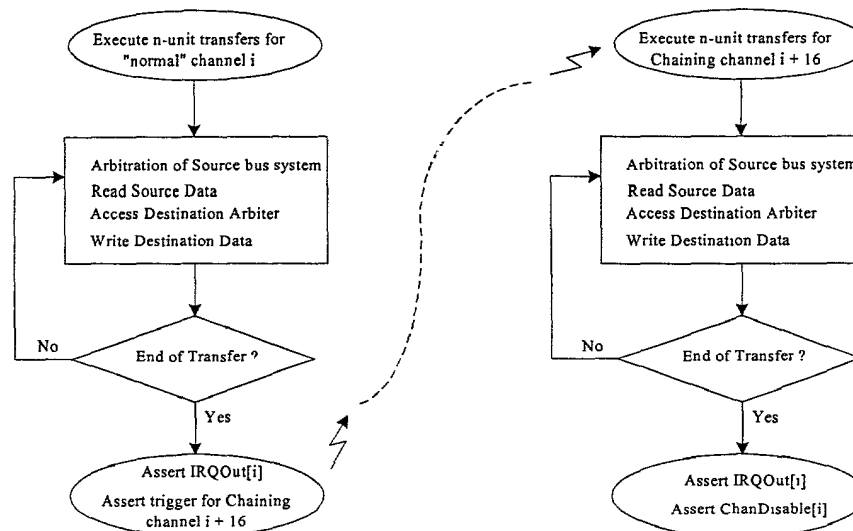


Figure 7: Execution flow for a transfer in chaining mode.

As an example, say DMA channel 4 is setup to serve a UART reception stream. Once the setup number of data has been received the DMA channel will generate an interrupt to the MCU/DSP system via *IRQOut*. At the same time another DMA channel (the chaining channel) takes over the service of the UART stream until its setup number of data has been received. After this, the task can either be halted or by issuing an interrupt, the DMA channel 4 can take over again. By proper setup of the memory areas, the two DMA channels writings form the A- and B-buffer system. This provides relaxed timing constraints for post processing data and decreased risks for simultaneously working on same set of data.

4.7.1 Using the Chaining Mode

The basic idea behind the chaining mode is that the “normal” 16 channels are unaffected, and an additional 16 channels is added for chaining purposes only. Thus, for every channel it is possible to enable (or disable) the chaining mode by use of a control bit in a *ChainingReg* register. When enabled, the chaining channel is available, so that when for example channel 7s chaining mode is enabled, the equivalent chaining channel for channel 7 is available (channel 23). It should be noted that it is possible to set up the chaining channels even when they are not enabled, but it will not be possible to use these channels as normal DMA channels. In order to keep track of the DMA chaining status, reading of the *ChainingReg* register returns the status for which of the chaining channels are currently active.

When the chaining mode is used, the first thing to ensure is that the corresponding bit in the *ChainingReg* is set. Once this is done, and the channel (plus the corresponding chaining channel) has been programmed, the setup and transfer runs in the same way as with a normal DMA channel. It is only when the normal channel transmission finishes that the DMA starts to behave differently.

An example: if the DMA channel 7 has been configured for its chaining mode, the bit7 in the *ChainingReg* will be checked by the DMA controller once the normal channel transfer finishes. The *IRQOut* is set (*ChanDisable* is not!) and the chaining channel takes over. In this way, the master who initiated the transfer is notified by an interrupt, but the *ChanDisable* bit that corresponds to the current channel is not asserted. The purpose of this operation is to put the normal channel x (seven in this example) on hold while letting the chaining channel (x + 16) take over serving the interrupt source. At the time the chaining channel takes over, the *ChainingReg* bit is cleared automatically. Seen from the outside, this switch to the chaining channel is transparent and it will look like the normal channel is just performing a new transfer. When the chaining transfer has been completed, the corresponding bit in both *IRQOut* and *ChanDisable* will be set. The DMA controller will perform the next task in the priority list.

After the *IRQOut* is spotted from the normal channel (i.e. the first *IRQOut* for that channel), a possible usage could be for the master to reprogram the chaining channel and set the *ChainingReg* bit again before the first chaining transmission ends. If this is done, the DMA controller will regard this second chaining transfer in the same way as the first chaining transfer, and once the first chaining transfer ends, the *IRQOut* will be set but the *ChanDisable* will not. This could go on and on and the *ChanDisable* will not be set before the end of the very last transfer.

Care should be taken when using the chaining mode since the setup for a Memory-to-Memory transfer in the Interrupt controller requires that the MemToMem bit (cf. [7], sec. 7.5.8) be selected in the ChanEvent Source Selection Table. Selecting this bit is equal to hardwiring the *IRQIn* for that channel, which is quite valid for a single channels transmission. In chaining mode however, a problem would arise if the transfer performed by the normal channel were a memory-to-memory transfer and the transfer performed by the chaining channel were anything but a memory-to-memory transfer. The reason is that since the end of the normal channel transfer does not set the *ChanDisable*, the *IRQIn* is not updated and the chaining channel will start transferring with a hardwired *IRQIn*. Therefore the following applies when using chaining transfers:

- *If the normal transfer is a Memory-to-Memory transfer, the chaining transfer must also be a Memory-to-Memory transfer.*

It is possible to reprogram the chaining channel even though it is currently executing because once the chaining channel transfer has begun, the transfer will be treated internally in the DMA controller as if it was the normal channel performing the transfer. This means that if a chaining transfer is being conducted and it is interrupted by a transfer that has to be done on a higher priority channel, the current status of the chaining transfer is stored in the internal DMA memory that belongs to the corresponding normal channel. Therefore, when the chaining transfer resumes after the higher priority interrupt, the transfer is continued from the *IRQIn* asserted bit, which will correspond to the normal channel.

In this way, the chaining channels will use the same control signals as their normal counterpart channel. Thus, in case of e.g. the interrupt signals, the chaining channel and the normal channel will use the same signals, which also means that the normal channel and its chaining channel *always* will have the same level of priority.

4.8 ImmediateTx Mode

The ImmediateTx mode is available in both the MCUDMA controller and the DSPDMA controller. This mode provides the user with the possibility pass an immediate source data value to the DMA instead of a source address, which defines the address of the source data.

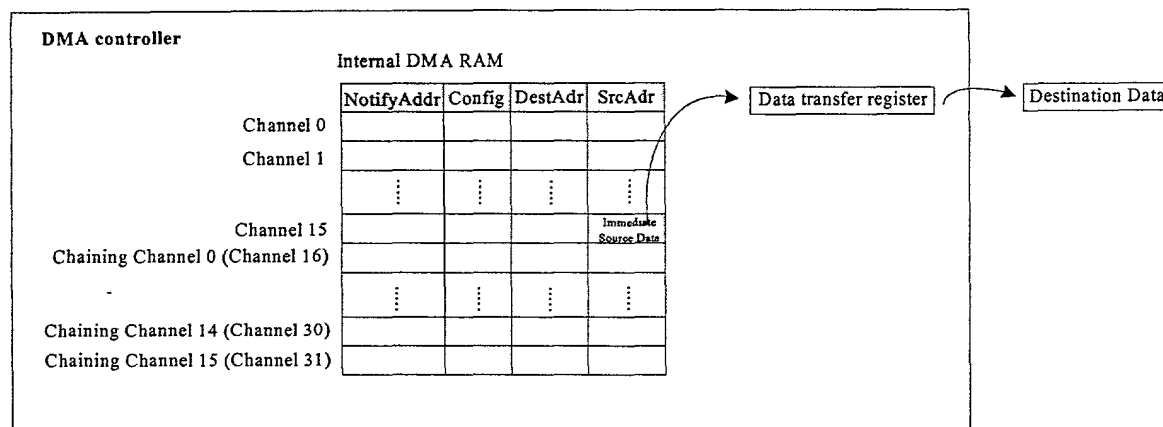


Figure 8: The ImmediateTx mode takes the source data from the internal DMA Memorys Source Address field instead of from an address given by the Source Address field. The destination address is used as the Notify Address.

An ImmediateTx mode transfer behaves almost the same as an ordinary transfer. However, the access to the data source is skipped which saves arbitration time, and the data from the source address field in the internal DMA RAM is instead transferred directly to the destination address. The destination address is used as the Notify Address in the ImmediateTx mode, and the *NotifyAddressMem* field in the internal DMA RAM should therefore hold the destination address. However, since the ImmediateTx mode is inherently a single data type transfer, this is not necessary, since the transfer ends as soon as the first immediate data has been transferred to the destination address. The completion of the ImmediateTx transfer is signaled by the appropriate *IRQOut* bit, and the assertion of the appropriate *ChanDisable* bit.

This feature is especially useful when the TEP module wants to set up the synthesizer since this consists of single word transfers. Figure 9 shows the data put on the data transfer register in the ImmediateTx mode.

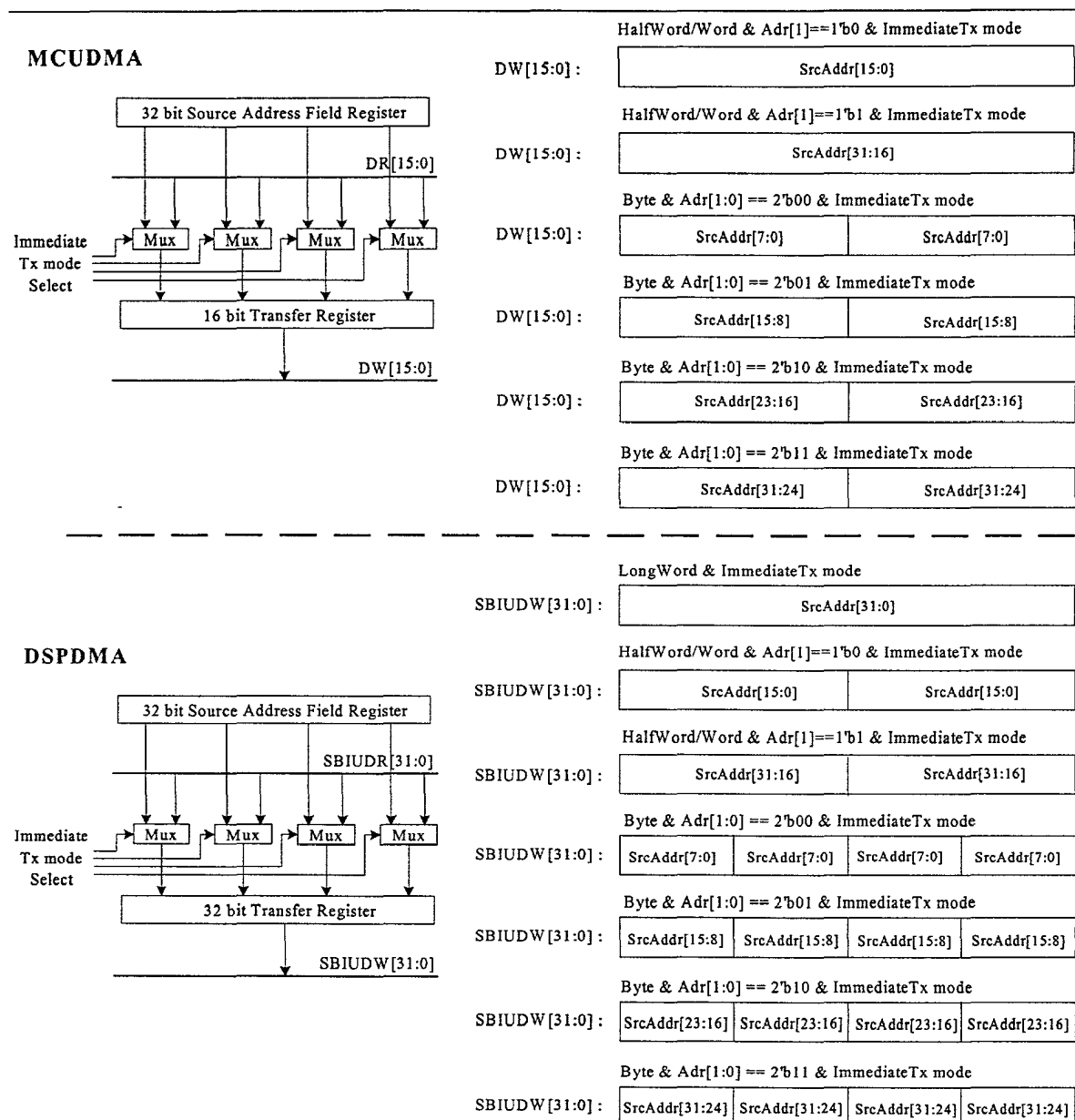


Figure 9: The data transfer register in ImmediateTx mode.

4.9 TEP Interface

The DSPDMA provides a TEP interface besides the interfaces found in the MCUDMA. This interface consists of a control signal (*DMACH0Con*), a data input line (*DMACH0Cfg[85:0]*) and a clear line for the *IRQOut* register (*DMAIRQRst[15:0]*).

The main functionality of the TEP interface is to provide the TEP module with the possibility to program an entire channel in one access. The programmed channel is *always* channel zero, and SW must therefore ensure that this channel is not expected to perform a transfer when the TEP module takes control. The TEP module always has priority over the normal channel zero usage.

If the TEP module wants to take usage of the DSPDMA channel zero, it must perform the same kind of setup of the interrupt controller, as a master would normally do to set up a DMA transfer. When the DSPDMA sees the highest priority interrupt (i.e. channel zero) on its *IRQIn*, it will either load the channel configuration data from the internal DMA RAM or from the TEP interface, depending on the state of the control signal from the TEP interface (*DMACH0Con*). The channel configuration data from the TEP is brought in, in parallel with the configuration data from the internal DMA RAM (see Figure 3). For further information see [8].

Another functionality of the TEP interface is to provide the TEP module with the possibility of performing bit wise clear of the *IRQOut* register. The TEP module does this by asserting the wanted bit in *DMAIRQrst[15:0]*, which causes a clear of the corresponding bit in *IRQOut[15:0]*. Thus, the bit wise clear of the *IRQOut* performed by TEP interface is done in a similar way as when the bit wise clear is done via the DPBUS interface. However, the TEP clear will always have priority over the DPBUS clear.

4.10 Channel Programming

The programming of the channels are done in a similar way no matter if the channel in question is a normal channel (i.e. from channel zero to 15) or a chaining channel (i.e. from channel 16 to 31). However, a write to the currently active channel should only modify the channel control registers and not transfer source and destination address since this would result in serious transfer errors.

The programming of a channel is done via the PBUS for the MCUDMA and the DPBUS for the DSPDMA, and involves writing the channel configuration data to the registers that specifies the channel in question. The MCUDMA controller is placed in the Monaco peripheral memory map at location 0x801C 0000 to 0x801F FFFF and the DSPDMA is placed in the DPBUS memory map at 0xF00 0000 to [TBD]. In this address range the lower 10 bit is given by the PBUS/SBIU arbiter to the DMA controllers where they are used internally to differentiate between the various registers, channels and the channels control registers. The address PAdr[9:0]/DPAAdr[9:0] is shown in Figure 10.

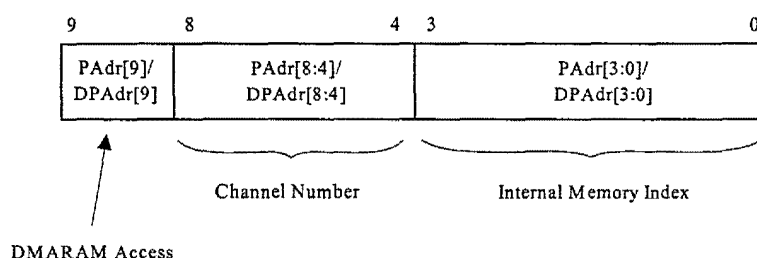


Figure 10: The lower 10 address bit (PAAdr[9:0]/DPAAdr[9:0]) to the DMA controllers from the PBUS/DPBUS arbiter.

The DMARAM access bit (PAAdr[9]/DPAAdr[9]) differentiates between making an access to the internal DMA data registers and the channel specific registers. If PAdr[9] or DPAAdr[9] = 1'b1, the internal DMA data control registers are chosen. The Channel number (PAAdr[8:4]/DPAAdr[8:4]) yields the Row index to the internal DMARAM i.e. the channel number, and the Internal Memory Index (PAAdr[3:0]/DPAAdr[3:0]) yields the Column index of the channel i.e. the various setup/control registers of the channel. The following table shows the address map internal to the DMA controllers.

Address	Register	R/W	Description
---------	----------	-----	-------------

0x801C 0200*	Reset	W	Any data value resets the DMA state machine.
0xF000 0000**			
0x801C 0202*	IRQOut[15:0]	R/W	Each bit corresponds to a channel output interrupt when set high. A DMA channel generates an interrupt on completion of a block transfer.
0xF000 0000**			
0x801C 0204*	IRQOutClr[15:0]	W	Allows bit-wise clearing of <i>IRQOut</i> bits. If a bit is set high the matching <i>IRQOut</i> bit is cleared.
0xF000 0000**			
0x801C 0206*	SysEx[0]	R/W	When asserted it indicates a DMA system exception.
0xF000 0000**			
0x801C 0208*	ActivChan[5:0]	R	A read returns {MasterIdle, ActivChan}. MasterIdle is high when no channel transfer is made. The ActivChan yields the currently active channel code (0-15). When no channel is active, the value 16 is read.
0xF000 0000**			
0x801C 020A*	ChainingReg[15:0]	R/W	The corresponding <i>ChainingReg</i> bit must be set to enable chainingmode for the channel. Bit 0 corresponds to channel 0 and bit 15 corresponds to channel 15.
0xF000 0000**			
0x801C 0000*	Chan0SrcLo[15:0]	R/W	Channel 0 source address lower halfword.
0xF000 0000**			
0x801C 0002*	Chan0SrcHi[12:0]	R/W	Channel 0 source address upper halfword
0xF000 0000**			
0x801C 0004*	Chan0DestLo[15:0]	R/W	Channel 0 destination address lower HalfWord
0xF000 0000**			
0x801C 0006*	Chan0DestHi[12:0]	R/W	Channel 0 destination address upper HalfWord
0xF000 0000**			
0x801C 0008*	Chan0Config[6:0]	R/W	Channel 0 configuration
0xF000 0000**			
0x801C 000A*	Chan0NotifyAddrMem[14:0]	R/W	Channel 0 Notify address for Linear transfers
0xF000 0000**			
0x801C 000C*	Chan0NotifyAddrIO[10:0]	R/W	Channel 0 Notify address for Circular transfers
0xF000 0000**			
0x801C 000E*	Chan0N[3:0]	R/W	Channel 0 Circular buffer size (2 ^N bytes)
0xF000 0000**			
0x801C 0010*	Chan1SrcLo[15:0]	R/W	Channel 1 source address lower HalfWord.
0xF000 0000**			
0x801C 0012*	Chan1SrcHi[12:0]	R/W	Channel 1 source address upper HalfWord
0xF000 0000**			
0x801C 0014*	Chan1DestLo[15:0]	R/W	Channel 1 destination address lower HalfWord
0xF000 0000**			
0x801C 0016*	Chan1DestHi[12:0]	R/W	Channel 1 destination address upper HalfWord
0xF000 0000**			
0x801C 0018*	Chan1Config[6:0]	R/W	Channel 1 configuration
0xF000 0000**			
0x801C 001A*	Chan1NotifyAddrMem[14:0]	R/W	Channel 1 Notify address for Linear
0xF000 0000**			

0x801C 001C*	Chan1NotifyAddrIO[10:0]	R/W	Channel 1 Notify address for Circular
0xF000 0000**			
0x801C 001E*	Chan1N[3:0]	R/W	Channel 1 Circular buffer size (2 ^N bytes)
0xF000 0000**			
the registers for channel 2 to 30 continues as for channel 0 and 1.			
0x801C 01F0*	Chan31SrcLo[15:0]	R/W	Channel 31 source address lower HalfWord.
0xF000 0000**			
0x801C 0122*	Chan31SrcHi[12:0]	R/W	Channel 31 source address upper HalfWord
0xF000 0000**			
0x801C 0124*	Chan31DestLo[15:0]	R/W	Channel 31 destination address lower HalfWord
0xF000 0000**			
0x801C 0126*	Chan31DestHi[12:0]	R/W	Channel 31 destination address upper HalfWord
0xF000 0000**			
0x801C 0128*	Chan31Config[6:0]	R/W	Channel 31 configuration
0xF000 0000**			
0x801C 012A*	Chan31NotifyAddrMem[14:0]	R/W	Channel 31 Notify address for Linear
0xF000 0000**			
0x801C 012C*	Chan31NotifyAddrIO[10:0]	R/W	Channel 31 Notify address for Circular
0xF000 0000**			
0x801C 012E*	Chan31N[3:0]	R/W	Channel 31 Circular buffer size (2 ^N bytes)
0xF000 0000**			

Table 5: The internal address map of the MCUDMA and DSPDMA channels. The addresses marked with a single * is the MCUDMA addresses, and the addresses marked with two ** is the DSPDMA addresses.

The DMA channel registers can be modified at any time, including that of the currently active channel, which will take immediate effect. It is important that only the *NotifyAddr* (or the *TranOneStop*) register is written to in this case, to avoid data corruption. The active channel register data is written back to the internal DMARAM when a higher priority channel becomes active.

The channel data is stored in the internal 32x86bit DMARAM memory, where each row entry looks like shown in Figure 11.

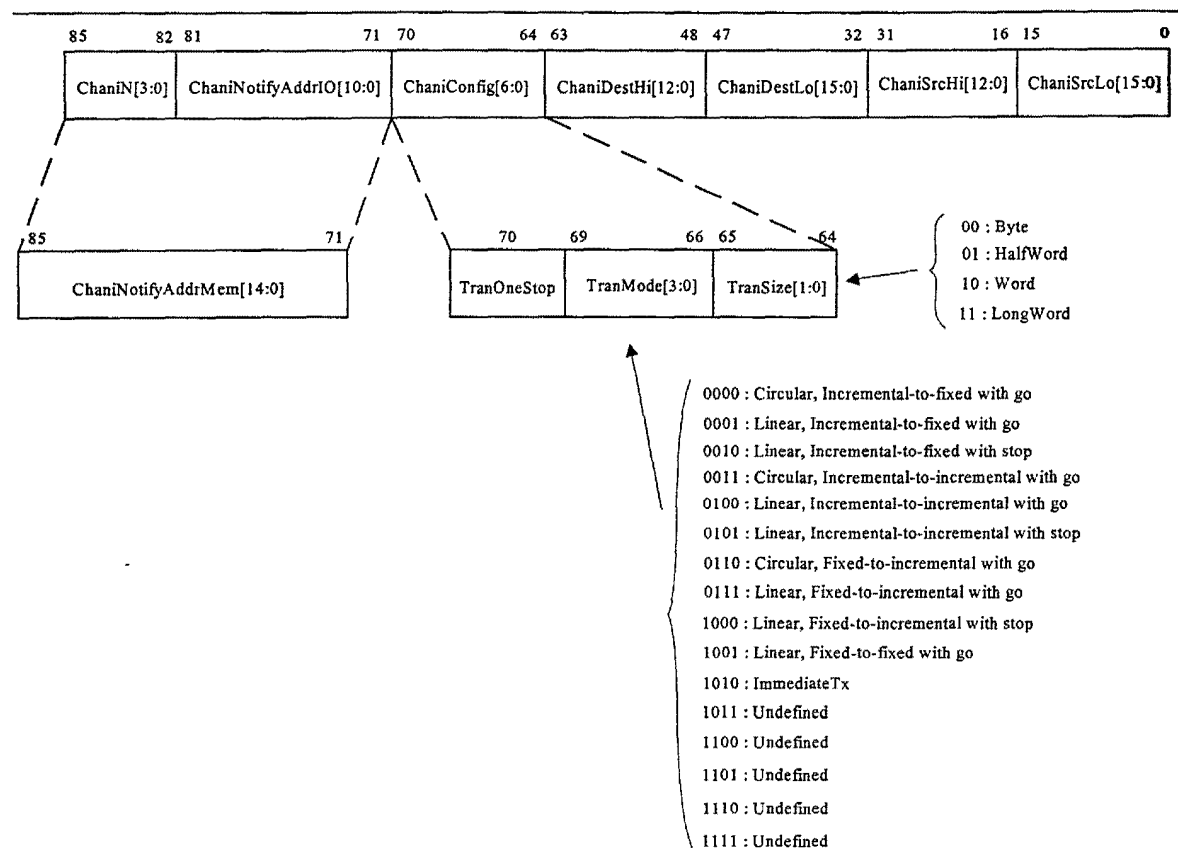


Figure 11: Row entry for channel *i* in the internal DMA memory.

4.10.1 ChanSrcHi and ChanDestHi

The two registers store the upper HalfWord of the DMA source and destination addresses respectively and are repeated for each DMA channel (0 to 31). Note that in ImmediateTx mode, the ChanSrcHi field contains the upper halfword of the transfer data instead of the upper half address at which the source data is placed.

4.10.2 ChanSrdLo and ChanDestLo

The two registers store the lower HalfWord of the DMA source and destination addresses respectively and are repeated for each DMA channel (0 to 31). Note that in ImmediateTx mode, the ChanSrcLo field contains the lower HalfWord of the transfer data instead of the lower half address at which the source data is placed.

4.10.3 ChanConfig

This register defines the functionality of the DMA channel and an instance of it exists for each DMA channel (0 to 31). The *TranSize* field defines whether the channel will transfer Bytes, HalfWords (16bit), Words (32bit in two halfword transfers) or LongWords (32bit in one word transfer). The *TranMode* field defines the channels transfer modes (see sec. 4.3).

An example for the usage of IO Tx in Go mode is to provide a Tx buffer nearly empty early warning. On receiving the early warning interrupt, the processor then has the option of setting *NotifyAddrIO* to the end of the Tx buffer and setting IO Tx Stop mode, or, having placed more data in the Tx buffer, a new early warning *NotifyAddr* could be defined, and the channel could remain in IO Tx Go mode. Caution: IO Tx Go mode should be used in conjunction with careful software design.

If the *TranOneStop* bit in the configuration register is set, the channels transfer will be completed after one unit transfer. This can be used by the TEP module when performing synthesizer programming.

4.10.4 NotifyAddrIO and ChanN

If the *TranMode* field of the *ChanConfig* register is configured to a circular buffer transfer, then the *NotifyAddrIO* and *ChanN* registers are applicable. They are however not applicable for memory to memory moves (linear transfers). All IO servicing transfers require a circular buffer area of memory to be reserved for either the Tx or the Rx data. The *ChanN* register defines the size of the circular buffer to be 2^N bytes. Note that the buffer size is in bytes regardless of whether the transfer is Byte, HalfWord, Word or LongWord. Note also that the first location of the buffer must always have the lower N bits of the buffer equal to zero. Additionally, the DMA will wrap the circular buffer pointer when the lower N bits of the buffer address are all one (e.g. if N=4 then the first address in the circular buffer will be $A_{31}, A_{30}, \dots, A_6, A_5, A_4, 1, 1, 1, 1$).

The *NotifyIO* address is compared against the lower 11 bit of the source address for IO Tx transfers or the lower 11 bit of the destination address for IO Rx transfers. The notify condition is generated when the *NotifyAddrIO* register equals the respective address bits. For IO Tx Stop mode transfers, the DMA will stop after doing this final transfer. For IO Tx Go mode and IO Rx Go mode, the DMA will continue to transfer data.

When the notify condition is met, regardless of the *TranMode*, the DMA will then set the corresponding bit in the *IRQOut* register for that channel. If the interrupt controller has been configured accordingly, this will then generate an interrupt to either the MCU or the DSP.

4.10.5 NotifyAddrMem

If the *TranMode* field of the *ChanConfig* register has been defined as memory to memory block move (Linear transfer), then this register is applicable in both Stop and Go mode. The *NotifyAddrMem* register is compared against the lower 15 bit of either the source or destination address, depending on which transfer mode has been chosen. The notify condition is generated when the *NotifyAddrMem* register equals the lower 15 address bits. The DMA will then stop after doing the final transfer if Stop mode is chosen or continue if Go mode is chosen, and set the corresponding *IRQOut* for that channel. Therefore, this register defines that the size of the block move is maximum 32kBytes (or 16k HalfWords, which is the more likely scenario). Note that in some situations, the *NotifyAddrMem* register may be a smaller number than the lower 15bit of the source or destination address. This is however a perfectly valid way to perform the Linear transfers.

4.10.6 IRQOut and IRQOutClr

The DMA set a bit in the *IRQOut* when the notify condition is met for that channel. This will cause an interrupt to either the MCU or DSP if the interrupt controller has been configured to make this happen (i.e. the corresponding DMA interrupt has been unmasked and the interrupt group, which the DMA interrupt is in, has been assigned to a priority level by programming the Interrupt Controller cross bar switch accordingly). The MCU or DSP on seeing a DMA interrupt must clear the interrupt source by clearing the corresponding bit in *IRQOut*. To clear a bit in *IRQOut*, the *IRQOutClr* register is written to with a data HalfWord, which has a one in each bit position, which is to be cleared.

4.10.7 SysEx

The DMAs will generate a system exception if, when in IO Tx Go mode, the notify condition is met for that channel but the corresponding bit in the *IRQOut* is already found to be set. This functionality is designed to trap a run-away IO Tx GO mode condition, where the DMAs would continue to repeatedly send the data in the IO Tx circular buffer indefinitely.

4.10.8 ActivChan

The *ActivChan* register can be read to discover which, if any, DMA channel is currently active. This register can be used to stop a Memory to Memory transfer cleanly part way through, thus, allowing the channel register to be read out and potentially restored at a later time. The procedure for this is as follows. Firstly, the channel is disabled by clearing the corresponding bit in the *ChanEnable* register which resides in the Interrupt controller. Secondly, the *ActivChan* register is repeatedly read until it no longer is equal to the channel, which is being disabled. This could be immediately, or take a few attempts. At this point it is safe to read out the channel registers for the halted channel. These register values can then be restored later and the channel re-enabled.

If no channel is currently active then the value decimal 16 will be read. If a channel is active, then the channel number 0 to 15 will be read (even though a chaining channel is currently active, the channel number of the chaining channels corresponding "normal" channel is read).

5. User Guide

This section describes how to use the DMA from a programmers perspective. The section includes an overview of the DMA controller state machine, which shows the states through which the DMA flows while performing a transfer. Following this are four DMA worked examples showing how to use the DMA for: Block Move, IO transmit, IO receive buffers, and chaining mode transfer. Finally the DMA view of the Monaco system address map is given.

Figure 12 shows the DMA state machine. Note that the *IRQIn* bus is used to determine the requesting DMA channel. Note also that the DMA gives priority to PBUS/DPBUS access to internal registers over its own accesses. The PBUS/DPBUS will effectively steal a DMA RAM access cycle from the DMA as required. Care should be taken when using Go mode and especially when using Go mode together with chaining mode. The transfer in Go mode must in either case be stopped by reprogramming of the channels config register.

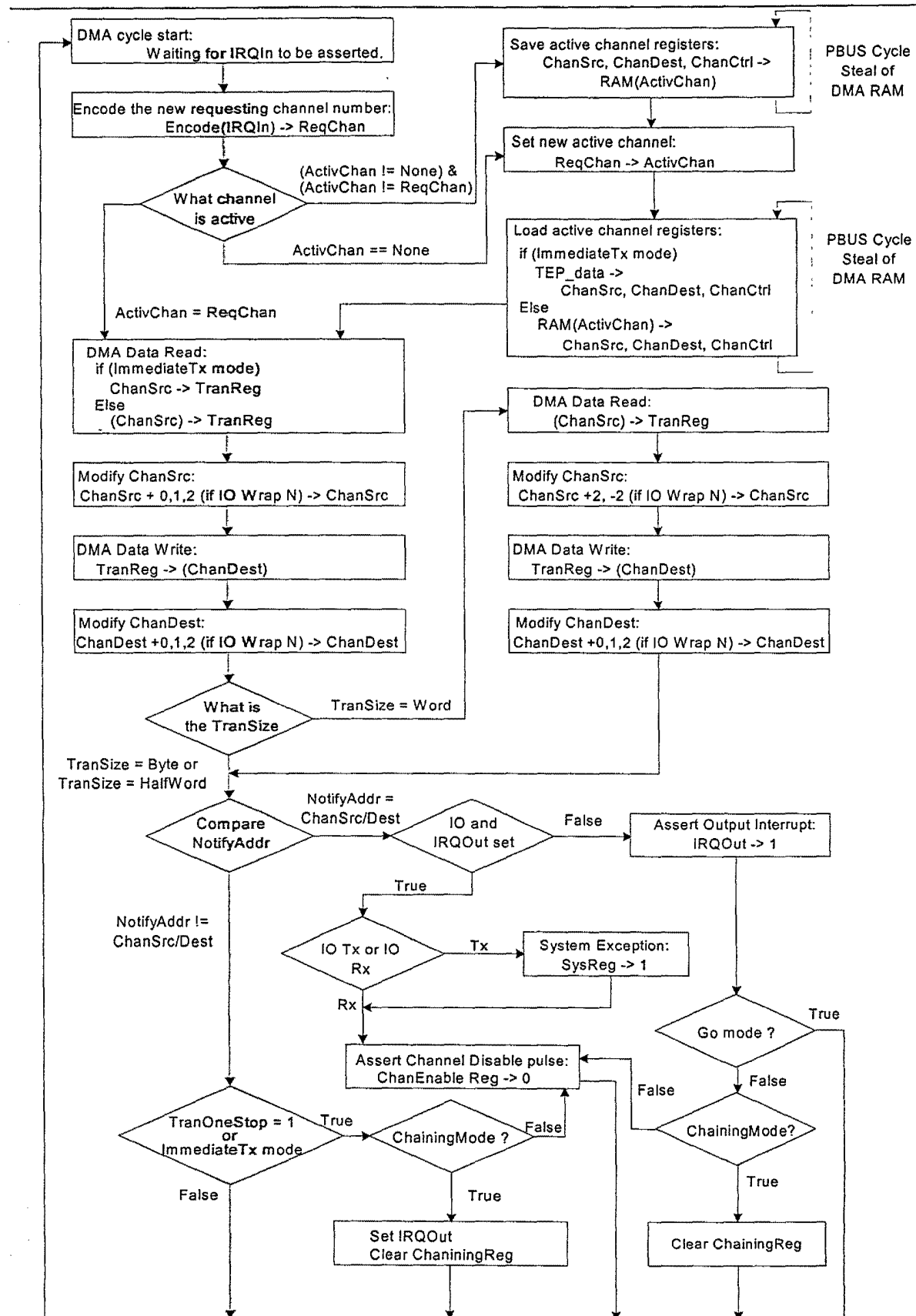


Figure 12: DMA State Machine

5.1 Performing a Memory to Memory Block Transfer

Figure 13 shows how the DMA internal channel registers are used as address pointers during a memory to memory block move transfer. Note that the source and destination memories could be the same of different physical memories.

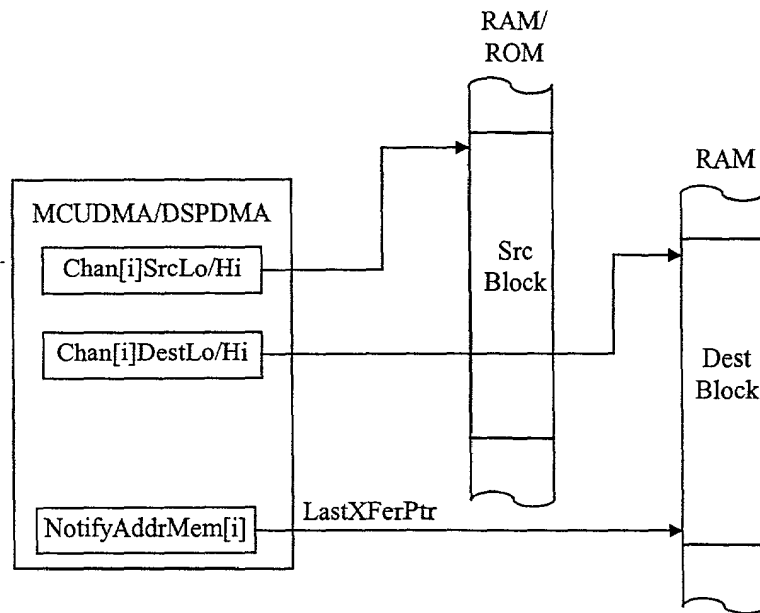


Figure 13: Memory to Memory Block Move Register Usage

This section describes how to configure a DSPDMA memory to memory block transfer in Linear Stop mode with a worked example. The example described is a low-priority block move of 256 HalfWords from external SRAM into the DSPs L2 memory:

Data source	= 0x0800 0000 to 0x0800 01FE
Data destination	= 0xC03A 4000 to 0xC03A 41FE
Data Block Size	= 256 HalfWords

Since the block move is low-priority, a low-priority DSPDMA channel, channel 15, is chosen for this transfer. The sequence of events is as follows:

1. Disable the DSPDMA channel (it should already be disabled). We need to write to address ChanEnable (+ the RAUClrBits offset) to clear bit 15. Therefore, we write 0x8000 to ChanEnable + 0x40.
2. Configure the DSPDMA section of the Interrupt Controller to be block move (Linear, MemToMem in Stop mode) for DMA channel 15. Referring to the Interrupt Controller Spec. we see that for the DSPDMA, MemToMem moves are defined as Interrupt source 0 (a constant logic 1). Therefore, we write 0x0000 to IRQChan15Config.
3. Unmask the MemToMem interrupt. Even though the MemToMem interrupt source is defined as a constant logic 1, we still need to unmask it. From the Interrupt Controller Spec. we see that the

MemToMem mask bit is bit 0 of IRQDMAMaskLo. To set this bit we write to IRQDMAMaskLo (+ RAUSetBits offset) to set bit 0. Therefore, we write 0x0001 to IRQDMAMaskLo + 0x20.

4. Unmask the DSPDMA output interrupt (IRQOut[15]) so that the DSPDMA will notify the DSP on completion of the block move. From the Interrupt Controller Spec. we see that IRQOut[15] (DMAI[15]) mask bit is bit 9 of interrupt group 7. Therefore, to set bit 9 of group 7 we write 0x0200 to IRQHWMask + 0x20 + (2 x 7).
5. Assign interrupt group 7 to a priority level. This could be any priority, but priority level 7 is the logical choice. Therefore, we write 0x0007 to IRQConfig + (2 x 7). The DSP will now receive an interrupt for any unmasked interrupt from group 7, including IRQOut[15].
6. Configure DSPDMA channel 15 for the block move. Configure the source address: write 0x0000 to Chan15SrcLo and 0x0800 to Chan15SrcHi. Configure the destination address: write 0x4000 to Chan15DestLo and 0xC03A to Chan15DestHi. Configure Chan15Config for MemToMem move of HalfWords: write 0x0051 to Chan15Config. Configure the last transfer address to be the lower 15 bits of the last location that we write to in DSP L2. Therefore, write 0x41FE to Chan15NotifyAddrMem. The DSPDMA and Interrupt Controller are now configured.
7. Enable the DSPDMA channel to start the block transfer. Write 0x8000 to ChanEnable + 0x20. This will cause bit 15 of the ChanEnable register to be set and the DSPDMA will see an interrupt on IRQIn[15]. The DSPDMA will start the transfer.
8. On completion of the DSPDMA block move, the DSPDMA will set IRQOut[15] and stop the transfer by clearing IRQIn[15] (the DSPDMA has a hard-wired bit-clearing mechanism for the ChanEnable register). The setting of IRQOut[15] (DSPDMAI[15]) will generate an DSP interrupt.
9. On seeing the interrupt the DSP will perform the normal interrupt service routine, which is discussed in the Interrupt Controller Spec. In general, the DSP will firstly read address IRQCurrentInterruptReg + (RAUPriEnc = 0x20) (a priority encoding read) which will indicate priority level 7. Since the DSP knows that priority level 7 maps to interrupt group 7, the DSP can directly read IRQHWStatus + 0x20 + (2x7) to determine which interrupt in group 7 has occurred. This will indicate 9, which is IRQOut[15] (DSPDMAI[15]).
10. The DSP must clear the interrupt source by writing 0x8000 to IRQOutClr. This will clear IRQOut[15]. Finally, the DSPDMA must clear bit 7 of the Active Interrupt register. This is done by writing 0x0080 to ActiveInterruptReg + 0x40. The DSPDMA transfer is now complete.

5.2 Configure an IO Transmit Buffer

Figure 14 shows how the DMA internal channel registers are used as address pointers to create a circular Tx FIFO buffer in a linear block of memory.

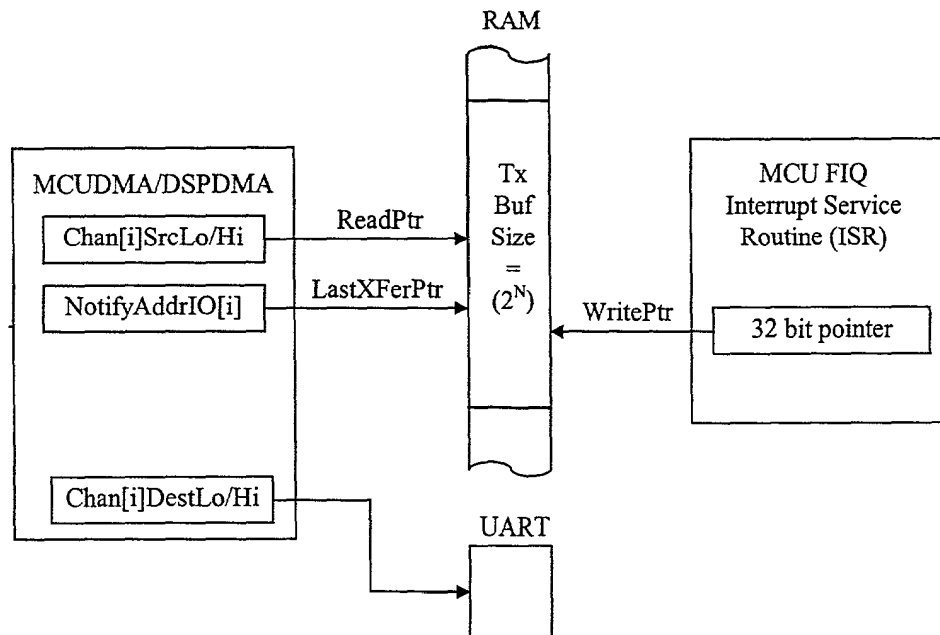


Figure 14: IO Transmit Buffer Register Usage

In the following example we will configure a 32 byte circular UART Tx buffer in internal system RAM and place 5 bytes of data into the buffer to be transmitted byte wise in incremental-to-fixed circular Go mode. The transfer is performed on the MCUDMA.

Buffer source = 0x40000100 to 0x4000011F
 UART Tx destination = 0x803C0000
 Buffer size = 32 Bytes

Since a UART Tx operation is reasonably high-priority we will assign it to MCUDMA channel 5.

1. Disable the MCUDMA channel (it should already be disabled). We need to write to address ChanEnable (+ the RAUClrBits offset) to clear bit 5. Therefore, we write 0x0020 to ChanEnable + 0x40.
2. Configure the MCUDMA section of the Interrupt Controller to be UART Tx (GSPaTx in this example) on MCUDMA channel 5. Referring to the Interrupt Controller Spec. we see that GSPaTx is Interrupt source 25. Therefore, we write 0x0019 to IRQChan5Config.
3. Unmask the GSPaTx interrupt. From the Interrupt Controller Spec. we see that the GSPaTx mask bit is bit 9 of IRQDMAMaskHi. To set this bit we write to IRQDMAMaskHi (+ RAUSetBits offset) to set bit 0. Therefore, we write 0x0200 to IRQDMAMaskHi + 0x20.
4. Unmask the MCUDMA output interrupt (IRQOut[5]) so that the MCUDMA will notify the MCU when the Tx buffer is empty. From the Interrupt Controller Spec. we see that IRQOut[5]

(MCUDMAI[5]) mask bit is bit 7 of interrupt group 3. Therefore, to set bit 7 of group 3 we write 0x0080 to IRQHWMask + 0x20 + (2 x 3).

5. Assign interrupt group 3 to a priority level. This could be any priority, but priority level 3 is the logical choice. Therefore, we write 0x0003 to IRQConfig + (2 x 3). The MCU will now receive a FIQ for any unmasked interrupt from group 3, including IRQOut[5].
6. Configure MCUDMA channel 5. Configure the source address: write 0x0100 to Chan5SrcLo and 0x4000 to Chan5SrcHi. Configure the destination address: write 0x0000 to Chan5DestLo and 0x803C to Chan5DestHi. Configure Chan5Config for circular Tx Go-mode transfer of Bytes: write 0x000C to Chan5Config. Configure the circular buffer size to be 32 bytes: write 0x0005 to Chan5N, since $2^5 = 32$. The MCUDMA and Interrupt Controller are now configured.
7. Having set-up the Tx buffer, we now wait for some data to transmit. Since we have 5 bytes to send we can configure the last transfer address to be the lower 15 bits of the last location that we read from in the buffer. Copy the 5 bytes of Tx data into 0x40000100 to 0x40000104 and write 0x0104 to Chan5NotifyAddrIO.
8. Enable the MCUDMA channel to start the transfer. Write 0x0020 to ChanEnable + 0x20. This will cause bit 5 of the ChanEnable register to be set and the MCUDMA will see an interrupt on IRQIn[5]. The MCUDMA will start to transmit the data to GSPa.
9. On reaching the notify address of the MCUDMA transfer, the MCUDMA will set IRQOut[5] and will stop the transfer by clearing IRQIn[5] (the MCUDMA has a hard-wired bit-clearing mechanism for the ChanEnable register in IO modes). The setting of IRQOut[5] (MCUDMAI[5]) will generate an ARM FIQ.
10. On seeing the FIQ the MCU will perform the normal FIQ interrupt service routine, which is discussed in the Interrupt Controller Spec. In general, the MCU will firstly read address IRQCurrentInterruptReg + (RAUPriEnc = 0x20) (a priority encoding read) which will indicate priority level 3. Since the MCU knows that priority level 3 maps to interrupt group 3, the MCU can directly read IRQHWStatus + 0x20 + (2x3) to determine which interrupt in group 3 has occurred. This will indicate 7, which is IRQOut[5] (MCUDMAI[5]).
11. The MCU must clear the interrupt source by writing 0x0020 to IRQOutClr. This will clear IRQOut[5]. Finally, the MCUDMA must clear bit 3 of the Active Interrupt register. This is done by writing 0x0008 to ActiveInterruptReg + 0x40. The MCUDMA transfer is now complete, but since the Go mode was chosen, the transmission will continue from the notify address 0x40000104 when IRQIn[5] is asserted. Thus, the MCU needs to reconfigure the notify address but does not have to re-enable the channel. Note that the NotifyAddrIO can be re-written to the MCUDMA channel even when it is actively being used by the MCUDMA at the time.

Figure 15 shows how the DMA internal channel registers are used as address pointers to create a circular Rx FIFO buffer in a linear block of memory.

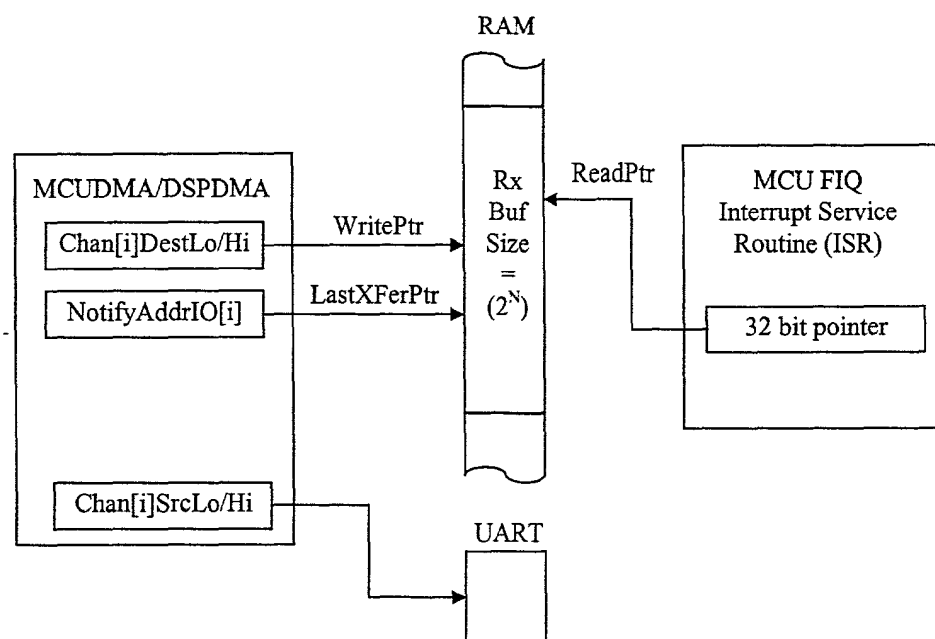


Figure 15: IO Receive Buffer Register Usage

In this example we will configure a 32 byte circular UART Rx buffer in internal system RAM and wait for 11 bytes to be received before alerting the MCU. The transfer type is therefore byte wise Circular Fixed-to-incremental in Go mode on the MCUDMA.

```

Buffer source      = 0x803C0000
UART Tx destination = 0x40000120 to 0x4000013F
Buffer size       = 32 Bytes

```

Since a UART Tx operation is reasonably high-priority we will assign it to MCUDMA channel 4.

1. Disable the MCUDMA channel (it should already be disabled). We need to write to address ChanEnable (+ the RAUClrBits offset) to clear bit 4. Therefore, we write 0x0010 to ChanEnable + 0x40.
2. Configure the MCUDMA section of the Interrupt Controller to be UART Rx (GSPaRx in this example) on MCUDMA channel 4. Referring to the Interrupt Controller Spec. we see that GSPaTx is Interrupt source 26. Therefore, we write 0x001A to IRQChan4Config.
3. Unmask the GSPaRx interrupt. From the Interrupt Controller Spec. we see that the GSPaRx mask bit is bit 10 of IRQDMAMaskHi. To set this bit we write to IRQDMAMaskHi (+ RAUSetBits offset) to set bit 0. Therefore, we write 0x0400 to IRODMAMaskHi + 0x20.

4. Unmask the MCUDMA output interrupt (IRQOut[4]) so that the MCUDMA will notify the MCU when the Rx buffer has reached a specified level. From the Interrupt Controller Spec. we see that IRQOut[4] (MCUDMAI[4]) mask bit is bit 6 of interrupt group 3. Therefore, to set bit 6 of group 3 we write 0x0040 to IRQHWMask + 0x20 + (2 x 3).
5. Assign interrupt group 3 to a priority level. This could be any priority, but priority level 3 is the logical choice. Therefore, we write 0x0003 to IRQConfig + (2 x 3). The MCU will now receive a FIQ for any unmasked interrupt from group 3, including IRQOut[4].
6. Configure MCUDMA channel 4. Configure the source address: write 0x0000 to Chan4SrcLo and 0x803C to Chan4SrcHi. Configure the destination address: write 0x0120 to Chan4DestLo and 0x4000 to Chan4DestHi. Configure Chan4Config for IO Rx go-mode transfer of Bytes: write 0x0018 to Chan4Config. Configure the circular buffer size to be 32 bytes: write 0x0005 to Chan5N, since $2^5 = 32$. Configure the MCU to be alerted when 11 bytes have been received: write 0x012A to Chan5NotifyAddrIO. The MCUDMA and Interrupt Controller are now configured.
7. Enable the MCUDMA channel to receive UART data. Write 0x0010 to ChanEnable + 0x20. This will cause bit 4 of the ChanEnable register to be set and the MCUDMA will see an interrupt on IRQIn[4]. The MCUDMA will start to receive the data from GSPa.
8. Having set-up the Rx buffer, we now wait for some data to be received, eventually, when 11 bytes have been received, the MCUDMA will set IRQOut[4]. The setting of IRQOut[4] (MCUDMAI[4]) will generate an ARM FIQ. The MCUDMA will continue to receive data on this channel.
9. On seeing the FIQ the MCU will perform the normal FIQ interrupt service routine, which is discussed in the Interrupt Controller Spec. In general, the MCU will firstly read address IRQCurrentInterruptReg + (RAUPriEnc = 0x20) (a priority encoding read) which will indicate priority level 3. Since the MCU knows that priority level 3 maps to interrupt group 3, the MCU can directly read IRQHWStatus + 0x20 + (2x3) to determine which interrupt in group 3 has occurred. This will indicate 6, which is IRQOut[4] (MCUDMAI[4]).
10. The MCU must clear the interrupt source by writing 0x0010 to IRQOutClr. This will clear IRQOut[4]. Finally, the MCUDMA must clear bit 3 of the Active Interrupt register. This is done by writing 0x0008 to ActiveInterruptReg + 0x40.
11. Still during the FIQ ISR, the MCU then reads the 11 bytes of data out of the Rx buffer and must then set a new value for NotifyAddrIO. However, during the ISR it is possible that more data has been received, the MCUDMA can optionally check this by reading the value of Chan4DestLo. The MCUDMA can then read this extra data and then set a suitable new value for NotifyAddrIO. Note that NotifyAddrIO can be re-written to the MCUDMA channel even when it is actively being used by the MCUDMA at the time.

[illegible]

5.4 Configure an Chaining transfer

Figure 16 shows how the DMA internal channel registers are used as address pointers to perform a memory-to-memory block move transfer in chaining mode. Note that the source and destination memories could be the same of different physical memories.

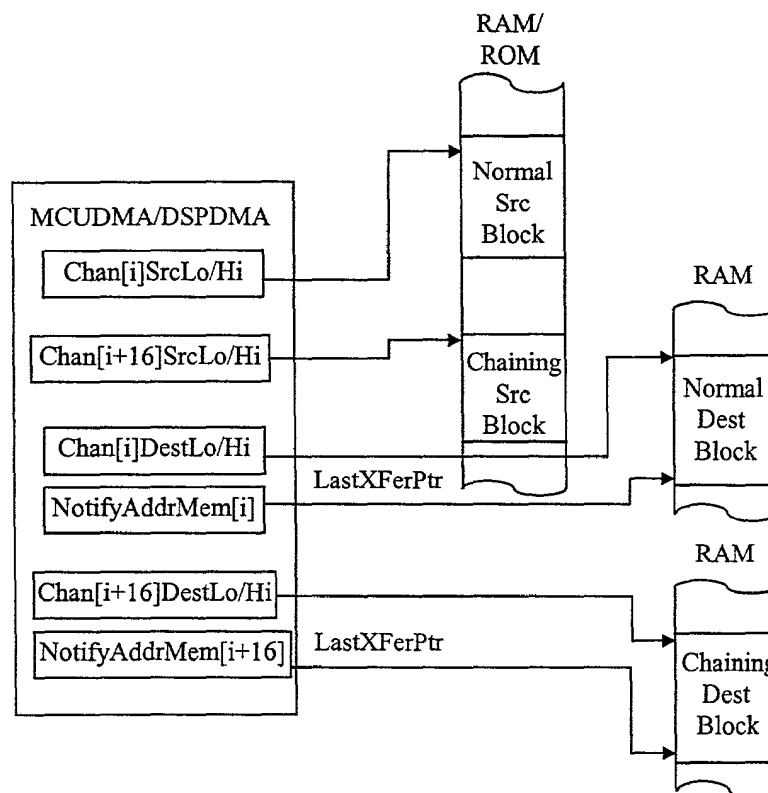


Figure 16: Memory to Memory block transfer in chaining mode

The following example shows how to configure a DSPDMA Linear Incremental-to-Incremental transfer in Stop mode. The example described is a low-priority block move of 128 Words from internal SRAM to DSP L2 memory and the corresponding chaining transfer moves 64 HalfWords from external SRAM to internal system RAM.

Data source for normal channel transfer	= 0x40000000 to 0x400001FE
Data destination for normal channel transfer	= 0xC03A4008 to 0xC03A4206
Data Block Size for normal transfer	= 128 Words
Data source for chaining channel transfer	= 0x09000000 to 0x0900007E
Data destination for normal channel transfer	= 0x40A0400C to 0x40A0408A
Data Block Size for normal transfer	= 64 HalfWords

Since the block move is low-priority, a low priority DSPDMA channel, channel 14, is chosen for this transfer. The sequence of events is as follows.

1. Disable the DSPDMA channel (it should already be disabled). We need to write to address ChanEnable (+ the RAUClrBits offset) to clear bit 14. Therefore, we write 0x4000 to ChanEnable + 0x40.
2. Configure the DSPDMA section of the Interrupt Controller to be block move (Linear, MemToMem in Stop mode) for DSPDMA channel 14. Referring to the Interrupt Controller Spec. we see that for the DSPDMA, MemToMem moves are defined as Interrupt source 0 (a constant logic 1). Therefore, we write 0x0000 to IRQChan14Config.
3. Unmask the MemToMem interrupt. Even though the MemToMem interrupt source is defined as a constant logic 1, we still need to unmask it. From the Interrupt Controller Spec. we see that the MemToMem mask bit is bit 0 of IRQDMAMaskLo. To set this bit we write to IRQDMAMaskLo (+ RAUSetBits offset) to set bit 0. Therefore, we write 0x0001 to IRQDMAMaskLo + 0x20.
4. Unmask the DSPDMA output interrupt (IRQOut[14]) so that the DSPDMA will notify the DSP on completion of the block move. From the Interrupt Controller Spec. we see that IRQOut[14] (DSPDMAI[14]) mask bit is bit 8 of interrupt group 7. Therefore, to set bit 8 of group 7 we write 0x0100 to IRQHWMask + 0x20 + (2 x 7).
5. Assign interrupt group 7 to a priority level. This could be any priority, but priority level 7 is the logical choice. Therefore, we write 0x0007 to IRQConfig + (2 x 7). The DSP will now receive an interrupt for any unmasked interrupt from group 7, including IRQOut[14].
6. Configure DSPDMA channel 14 and the chaining channel (channel 30) for the block move. Configure the source address in the normal channel: write 0x0000 to Chan14SrcLo and 0x4000 to Chan14SrcHi. Configure the destination address: write 0x4008 to Chan14DestLo and 0xC03A to Chan14DestHi. Configure Chan14Config for Linear MemToMem move in stop mode of Words: write 0x0016 to Chan14Config. Configure the last transfer address to be the lower 15 bits of the last location that we write to in DSP L2. Therefore, write 0x4206 to Chan14NotifyAddrMem. To configure the chaining channel simply repeat this procedure with writes to channel 30 instead. Configure the source address in the chaining channel: write 0x0000 to Chan30SrcLo and 0x0900 to Chan30SrcHi. Configure the destination address: write 0x400C to Chan30DestLo and 0x40A0 to Chan30DestHi. Configure Chan30Config for Linear MemToMem move in stop mode of HalfWords: write 0x0015 to Chan30Config. Configure the last transfer address to be the lower 15 bits of the last location that we write to in internal system RAM. Therefore, write 0x408A to Chan30NotifyAddrMem. The DSPDMA and Interrupt Controller are now configured.
7. Enable the DSPDMA channel to start the block transfer. Write 0x4000 to ChanEnable + 0x20. This will cause bit 14 of the ChanEnable register to be set and the DSPDMA will see an interrupt on IRQIn[14]. The DSPDMA will start the transfer.
8. On completion of the DSPDMA block move, the DSPDMA will set IRQOut[14] and indicate the end of the normal transfer by clearing IRQIn[14]. The setting of IRQOut[14] (DSPDMAI[14]) will generate an DSP interrupt. Note that the ChanEnable register is not cleared and the IRQIn will therefore still be valid as an input to the DSPDMA controller. The chaining channel thus takes over.
9. On seeing the interrupt, the DSP will perform the normal interrupt service routine, which is discussed in the Interrupt Controller Spec. In general, the DSP will firstly read address IRQCurrentInterruptReg + (RAUPriEnc = 0x20) (a priority encoding read) which will indicate priority level 7. Since the DSP knows that priority level 7 maps to interrupt group 7, the DSP can

directly read $\text{IRQHWStatus} + 0x20 + (2 \times 7)$ to determine which interrupt in group 7 has occurred. This will indicate 8, which is $\text{IRQOut}[14]$ ($\text{DSPDMAI}[14]$).

10. The DSP must clear the interrupt source by writing 0x4000 to IRQOutClr . This will clear $\text{IRQOut}[14]$ and must therefore be performed before the chaining channel terminates its transfer.
11. On completion of the chaining DSPDMA block move, the DSPDMA will set $\text{IRQOut}[14]$ and indicate the end of the chaining transfer by clearing $\text{IRQIn}[14]$ (the DSPDMA has a hard-wired bit-clearing mechanism for the ChanEnable register). The setting of $\text{IRQOut}[14]$ ($\text{DSPDMAI}[14]$) will generate a DSP interrupt.
12. On seeing the interrupt, the DSP will again perform the normal interrupt service routine, which is discussed in the Interrupt Controller Spec. In general, the DSP will firstly read address $\text{IRQCurrentInterruptReg} + (\text{RAUPriEnc} = 0x20)$ (a priority encoding read) which will indicate priority level 7. Since the DSP knows that priority level 7 maps to interrupt group 7, the DSP can directly read $\text{IRQHWStatus} + 0x20 + (2 \times 7)$ to determine which interrupt in group 7 has occurred. This will indicate 8, which is $\text{IRQOut}[14]$ ($\text{DSPDMAI}[14]$).
13. The DSP must clear the interrupt source again by writing 0x4000 to IRQOutClr . Finally, the DSPDMA must clear bit 7 of the Active Interrupt register. This is done by writing 0x0080 to $\text{ActiveInterruptReg} + 0x40$. The chaining DSPDMA transfer is now complete.

5.5 DMA View of Monaco Memory Map

Figure 17 shows the address map as seen from the perspective of the DMA when it drives the 32 bit address SBUS_A[31:0] onto its address bus.

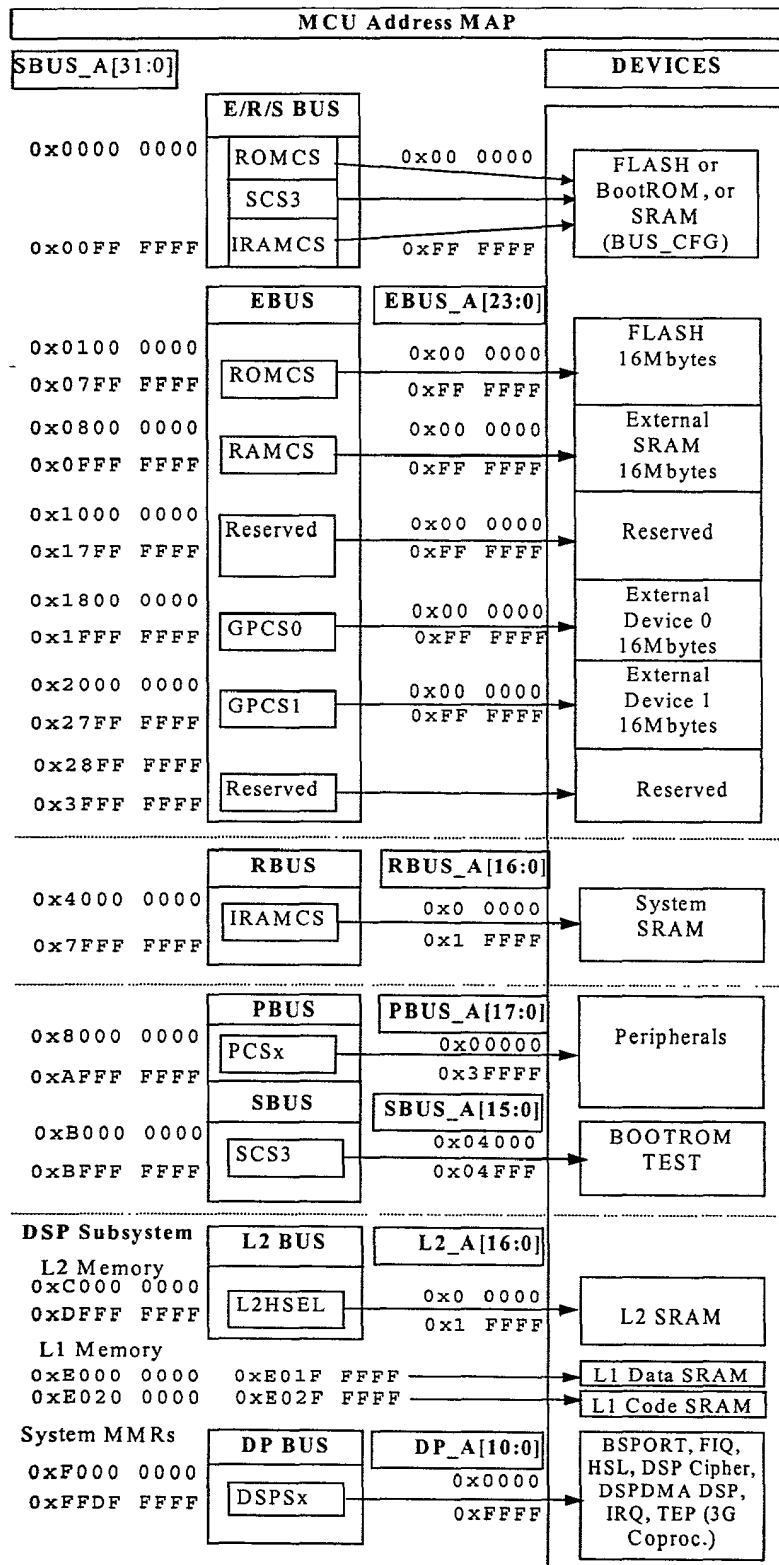


Figure 17: MCUDMA Address Map

Figure 18 shows the address map as seen from the perspective of the DSP when it drives the 32 bit address DSPBUS_A[31:0] onto its address bus.

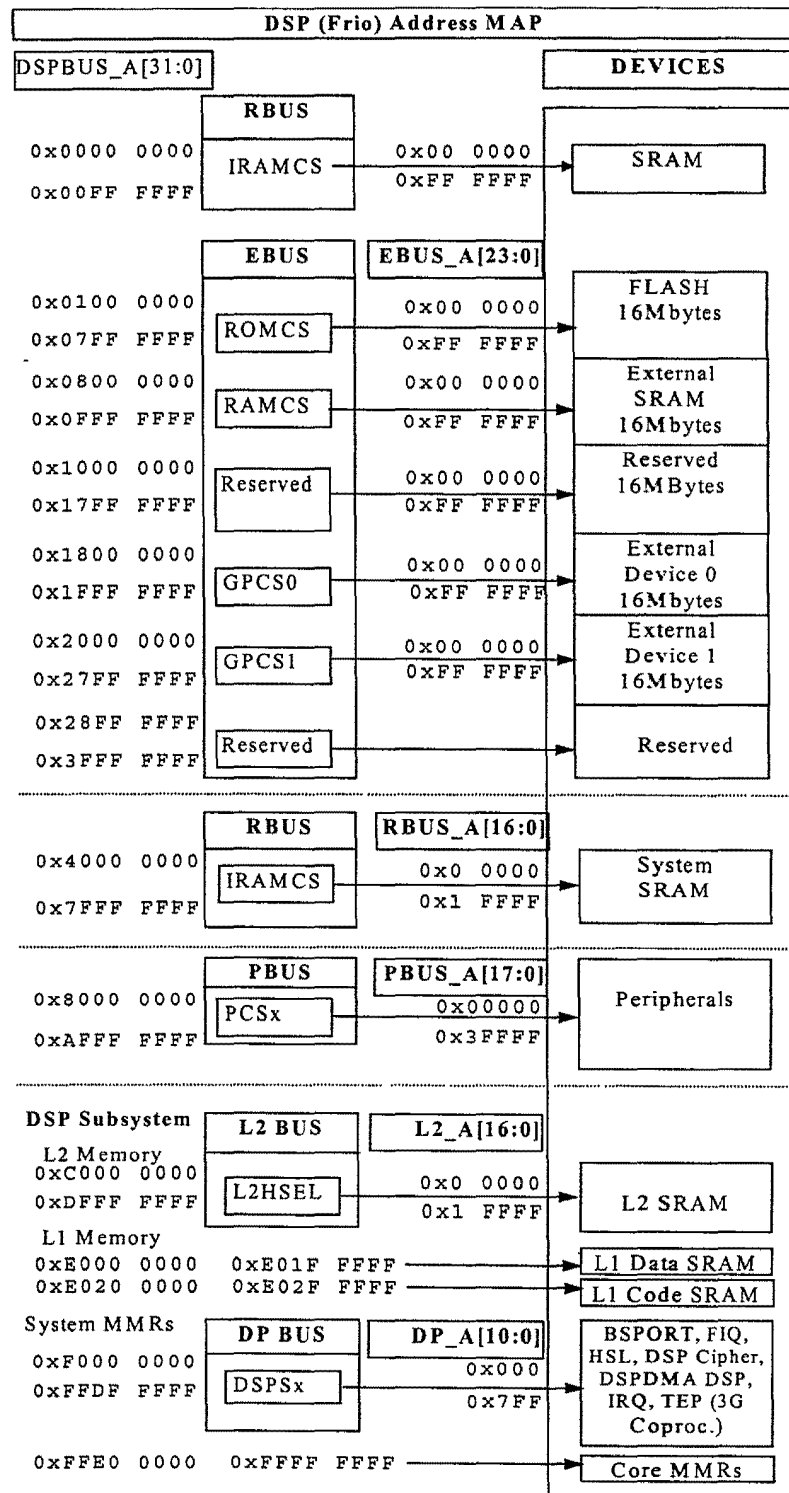


Figure 18: DSPDMA Address Map

6. Register description

Notice that 'x' in the descriptions means an unused bit – when read a '0' will be returned in such cases.

6.1.1 Reset – Reset address accessible from PBUS

Write-only register

Bit 0: No matter the data value, a write to this register will cause a reset of the DMA controller.

6.1.2 IRQOut[15:0] – Interrupt output register

Read/write register

Bit 15:0: Each bit refers to a channel where bit zero refers to channel zero and bit 15 refers to channel 15. An asserted bit (high) indicates that the channel transfer has completed.

6.1.3 IRQOutClr[15:0] – Interrupt output Clear register

Write-only register

Bit 15:0: Each bit refers to a channel where bit zero refers to channel zero and bit 15 refers to channel 15. An asserted bit (high) indicates a clear of the corresponding bit in the IRQOut[15:0] register.

6.1.4 SysEx – System Exception register

Read/Write register

Bit 0: If a system exception has happened i.e. Rx/Tx overrun is detected, the bit will be asserted (high).

6.1.5 ActivChan[5:0] – Active Channel Number

Read only

Bit 5 : Indicates the MasterIdle signal. This bit is asserted when no channel transfer is made.

Bit 4:0: Defines the currently active channel number
 5'b00000 : Channel zero is currently active
 5'b00001 : Channel 1 is currently active
 :
 5'b01111 : Channel 15 is currently active
 5'b10000 : No Channel is currently active

6.1.6 ChainingReg[15:0] – Chaining Mode Enable Register

Read/write register

Bit 15:0: Each bit refers to a channel where bit zero refers to channel zero and bit 15 refers to channel 15. An asserted bit (high) indicates that the corresponding channel is in chaining mode.

6.1.7 ChaniSrcLo[15:0] – Lower HalfWord of Source Address

Read/Write register

Bit 15:0: Lower HalfWord of Source address for channel i (0 ... i ... 31).

6.1.8 ChaniSrcHi[12:0] – Upper HalfWord of Source Address

Read/write register

Bit 12:0: Upper HalfWord of Source address for channel i (0 ... i ... 31).

6.1.9 ChaniDestLo[15:0] – Lower HalfWord of Destination Address

Read/Write register

Bit 15:0: Lower HalfWord of Destination address for channel i (0 ... i ... 31).

6.1.10 ChaniDestHi[12:0] – Upper HalfWord of Destination Address

Read/write register

Bit 12:0: Upper HalfWord of Destination address for channel i (0 ... i ... 31).

6.1.11 ChaniConfig[6:0] – Channel Configuration register

Read/write register

- Bit 6: Active high control bit for transfer one unit then stop (TranOneStop) for channel i (0 ... i ... 31).
- Bit 5:2: Transfer mode selection for channel i (0 ... i ... 31).
 4'b0000: Circular, Incremental-to-Fixed in Go mode
 4'b0001: Linear, Incremental-to-Fixed in Go mode
 4'b0010: Linear, Incremental-to-Fixed in Stop mode
 4'b0011: Circular, Incremental-to-Incremental in Go mode
 4'b0100: Linear, Incremental-to-Incremental in Go mode
 4'b0101: Linear, Incremental-to-Incremental in Stop mode
 4'b0110: Circular, Fixed-to-Incremental in Go mode
 4'b0111: Linear, Fixed-to-Incremental in Go mode
 4'b1000: Linear, Fixed-to-Incremental in Stop mode
 4'b1001: Linear, Fixed-to-Fixed in Go mode
 4'b1010: ImmediateTx mode
 4'b1011: Undefined
 4'b1100: Undefined
 4'b1101: Undefined
 4'b1110: Undefined
 4'b1111: Undefined
- Bit 1:0: Transfer Size selection for channel i (0 ... i ... 31).
 2'b00 : Byte mode
 2'b01 : HalfWord mode
 2'b10 : Word mode
 2'b11 : LongWord

6.1.12 ChaniNotifyAddrMem[14:0] – Linear Transfer Notify Address register

Read/write register

- Bit 14:0: Notify address in linear transfer mode for channel i (0 ... i ... 31)..

6.1.13 ChaniNotifyAddrIO[10:0] – Circular Transfer Notify Address register

Read/write register

- Bit 10:0: Notify address in circular transfer mode for channel i (0 ... i ... 31)..

6.1.14 ChaniN[3:0] – Circular Buffer Register

Read/write register

- Bit 3:0: Size of Buffer in Circular transfer mode for channel i (0 ... i ... 31)..

[illegible]

a

Monaco
Clock Control Short form
Technical note

File name: Monaco Clock Control - short.r01.doc

Status: Draft

Author: Joern Soerensen, ADI-DK

Revision: rev 0.1

When	Who	What
Aug 27 th 2001	Joern Soerensen	Initial document

[illegible]

1. Introduction

Power consumption is of outmost importance in battery-operated equipment. This invention relates mainly to how to reduce power consumption by controlling clocks very tight, such that power is only consumed when absolutely necessary.

2. Prior Art/Background

In our previous generations of chip sets clock control has mostly been under SW control, with minimal HW to sequence power-up and down of the external clock source and the internal processor clocks.

The decision of when a clock could be switched off was in SW, while the power-down/up sequencing was under HW control.

In a system with multiple masters, it is necessary that all masters agree that a given clock is not needed. SW design methodology also has a major impact on how easy it is to determine if a clock is needed or not.

As a result of this it was necessary to switch clocks on a few frames before the frame in which the clock really was needed for receiving the wanted data. It was also necessary to keep the clock running for some additional frames after the last data frame to ensure that all processing was complete.

These additional frames needed turns easily out to carry an overhead of close to 100%, thus being a very significant factor wrt power consumption in a cell phone, when in idle mode (Waiting for an incoming call).

3. Description of the invention

The invention deals with how to tightly control when clocks can be switched on/off, in order to minimize the overhead in clock on time.

The idea is that each master in the system has a register set identifying its minimum clock requirements. These registers are all routed to one clock control block, which then makes the decision of when a given branch of a clock tree can be switched off or needs to be switched on.

All the switching is under HW control, such that as soon as a clock is identified to not being needed any longer it will automatically be switched off.

This module does also work together with the TEP module to determine when the external system oscillator can be switched off. It does take a significant amount of time to switch the external oscillator back on; it is necessary to know when the next scheduled event for the SW will happen. TEP calculates this time, compares this to the time it takes to wake the system oscillator up and generates a signal indicating that it is ok to switch the clock off.

The HW in the clock control module will then switch the System Clock off and will sequence the power-up sequence when it is determined that the clock is needed again.

The state machine, ultimately making decisions on when clocks can be enabled/disabled is shown in the MonacoClockControl document in figure 2.

Figure 4 shows how gating functions, which are derived from each other, generates the various clocks.

The table just before chapter 6 shows how the setting in the ARM control register will influence the PLL setting for those possible modes.

----- END ----- END -----

a

Simplified Dynamic Voltage Control Intellectual Property Disclosure

File name: Dynamic Frequency and Voltage Control.doc

Status: Draft

Author: Michael Allen, ADI-AUS

Revision: revision 0.0

When	Who	What
August 26, 2001	Michael Allen	Initial document

[illegible]

1.	Introduction	4
2.	Overview.....	4
3.	References	4
4.	Prior Art.....	4
5.	Invention Description	4

1. Introduction

This document defines the software interface of the dynamic voltage control function used on Monaco. The purpose of this document is to seed the patent application process for this feature.

2. Overview

In most computer systems today, power consumption is relatively constant: if the device is powered on, it is clocked at a fixed frequency (typically the maximum clock frequency the device is rated to accept, for maximum performance). However, products which have lower power consumption have distinct competitive advantages in mobile, battery powered applications, such as cell phones, laptop computers, or personal digital assistants (PDAs). For the CMOS components in systems such as these, dynamic power consumption is described by the following formula:

$$P_D = cV^2F, \text{ where:}$$

- c is a constant determined by the circuits operating at that voltage and frequency
- V is the voltage difference between Vdd and Gnd
- F is the clock frequency

Since the dynamic power varies with the squared value of the applied voltage, one of the most effective ways of minimize dynamic power consumption is to reduce the applied voltage to the minimum value required for the CMOS design to perform the required function(s) at any point in time. The performance of the design is most often determined by the frequency the device is clocked at. Therefore, if the required clock frequency is known a priori to performing a function, it is possible to set the voltage (dynamically) to the minimum value required to allow the device to operate at that clock frequency, achieving minimal dynamic power consumption.

This invention provides a relatively simple mechanism for software to specify a the minimum required frequency, such that hardware then sets the clock to that frequency, and automatically adjusts the voltage to the minimum value required to support that clock frequency.

3. References

- [1] Monaco Digital Baseband Processor Design Specification, Rev 0.2

4. Prior Art

5. Invention Description

For any of the active power states of a mobile device, there are likely to be periods of time during which maximum frequency of the system in that mode is not required. Whenever this scenario exists, power consumption can be reduced, perhaps significantly, by reducing internal Vdd to the minimum level required to support the minimum clock frequencies required to support the needed performance level. This invention provides dedicated hardware support for explicit software management of clock requirements, which implicitly manages the internal Vdd, when appropriately configured. This hardware provides a simple programming model which minimizes the impact to software, thereby maximizing the potential benefit of this power management tool.

The minimum power required to support any phone state is defined by the clock frequency requirements in that state. Software manages the clock frequency explicitly, in this example [1], by

setting the PLL multiplier ratio, via the MSEL[5:0] control field. The dynamic voltage control block includes a lookup table (RAM), indexed by the PLL MSEL value, used to determine the appropriate internal Vdd level. This function also includes a finite state machine, which sequences the power/PLL appropriately. The following block diagram describes the system:

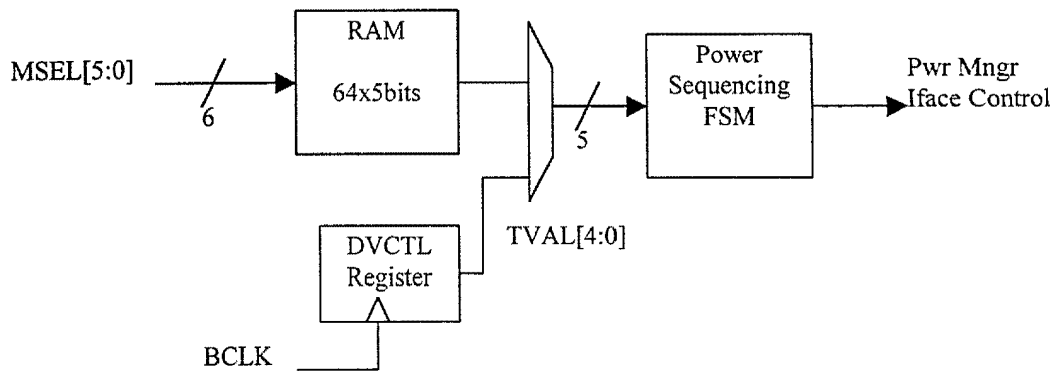


Figure 1 Dynamic Voltage Control Block Diagram

The DVCTL register includes the TLVAL[4:0], the default table lookup value used until the RAM is set up and enabled. Software loads the table lookup values into the RAM, as part of the reset initialization sequence. The table lookup values are determined by characterization of the silicon. The 6-bit MSEL field provides an index of up to 64 unique 5-bit values. The 5-bit lookup values provide a granularity of 32 voltage steps. 32 steps across the variable internal Vdd range of 800mV (assume max of $1.5V - 0.7V = 0.8V$) provides a granularity of 25mV.

The clock frequencies and voltages vary dynamically. The following diagram provides an example of how the various clock frequencies might vary over time. DSCLK is omitted from the diagram for clarity. Note that the BCLK divider ratio can change, to keep BCLK near a desired frequency target.

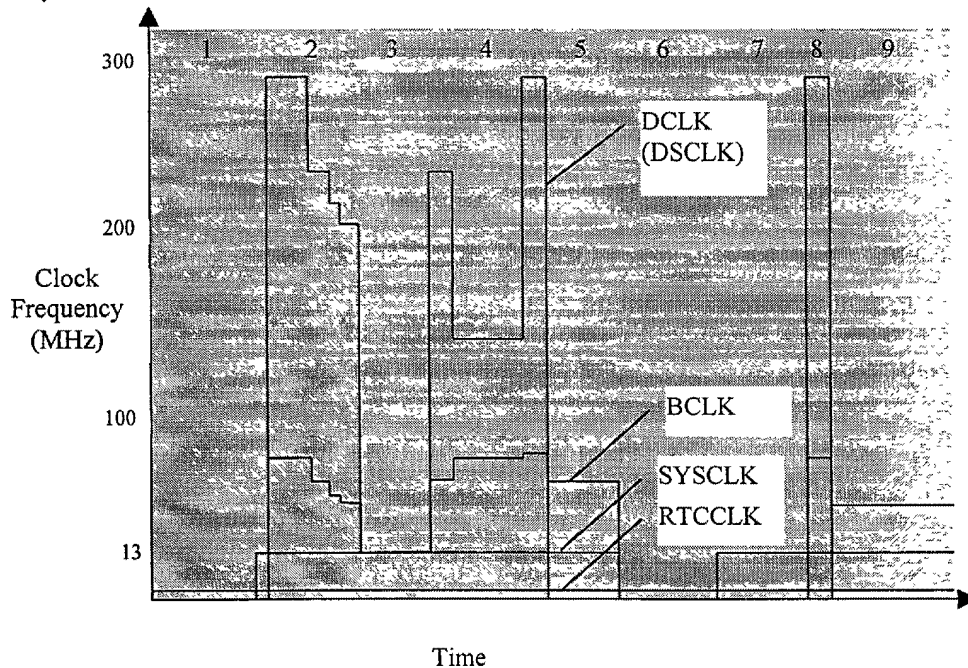


Figure 2: Example of Clock Frequency Variation Over Time

The above diagram describes several different operational states, denoted by the different color bars, and numbered 1-9. A brief description of each of these states, from a system clocking perspective, follows:

1. The baseband is the lowest energy state (slow clocking mode). The only enabled clock in this mode is the 32KHz RTCCLK. This state is exited by the TEP control module, which enables CLKIN generation, then enables the CLKIN receiver, then, after a fixed settling time, enables SYSCLK. Finally, the TEP programs the PLL multiplier ratio, and enables the PLL, disabling the PLL bypass.
2. This state ramps to the highest possible performance/energy level, with both the DSP (DCLK) and MCU (BCLK) cores running at maximum clock rate possible. Note that the cores wakeup and begin interrupt processing *before* the system PLL has acquired phase lock (a separate invention disclosure). Over time, less performance is required; therefore, the dynamic voltage control mechanism steps the clock rate down, allowing the supply voltage to drop.
3. In this state, the PLL has been bypassed, such that DSCLK = BCLK = SYSCLK frequency. The DSP is in IDLE state, and DCLK is gated off.
4. Equivalent to state 2, but at lower energy levels. Note that the BCLK divider changes from /2 to /1 at the first drop in DCLK. This is possible because the PLL clock dropped to a level low enough to allow BCLK to run at DSCLK frequency, and not exceed the maximum supported BCLK frequency.
5. The PLL is enabled; however, the DSP is idled, and DCLK is gated off.
6. Identical to state 1.
7. PLL bypassed state, identical to state 3. Note that here, the phone could be in standby mode. TEP, B SPort, DSP DMA, and L2 memory would be the only enabled modules besides RTC in this mode.
8. Equivalent to state 2, however, clock frequency is not stepped down here.
9. Identical to state 5.

Note that the scope of this invention is not limited to these particular RAM array sizes, or voltage increments, or clock control specifiers. A number of different hardware mechanisms can be used to realize the invention's potential. For example, the lookup table could be implemented in read-only memory (ROM), or FLASH memory, such that the table would need to be initialized at reset.

The novelty of this invention is that the software need only specify a clock frequency requirement. The hardware then provides the minimum voltage (and therefore power consumption) required to support that clock frequency. This approach minimizes the software overhead required to manage dynamic power consumption.

----- END -----

High-Speed Data Logging and Trace Functions (HSL)

The High Speed Logger (HSL) interface facilitates (relatively) non-invasive, real-time, external data logging of key events internal to Monaco. Using this feature, an external data logger (e.g., a PC) can capture internal state such as a highly compressed DSP address trace, then post process the log to aid in debug of software running on the Monaco baseband processor.

The HSL can be programmed to transmit three types of data streams, with the following priorities:

- TEP instructions, showing interrupts to the DSP, DSP-DMA or MCU, and GPO/GPIO port signaling.
- HSL software log messages from both the DSP and the MCU.
- DSP address trace.

The external interface consists of up to 8 data pins, and a clock. This port can be selected to come out over the USC/GPIO pins. In the extended bond out package, the HSL port is also available on dedicated pins, which enables simultaneous HSL logging and serial port operation. DSP address trace uses all 8 data pins, whereas the TEP instructions and software log messages keep the 5-bit encoding scheme from Hercules. Hence, existing data logger devices may be reused without modifications, but cannot provide address trace.

Note that the off-chip, external data logger is responsible for time stamp generation. Thus, timing information is removed from timed TEP instructions.

The Frio DSP supports a dedicated PC Dump interface to the HSL module. This interface provides the entire 31-bit PC, along with a valid bit indicating when the instruction at the PC reaches the writeback stage of the instruction pipeline. Additionally, the DSP provides a signal indicating discontinuities in the PC (i.e., interrupts, branches, call/returns, zero overhead hardware loops) and a signal indicating immediate target addressing (the branch can be resolved statically and the data logger on the PC has access to the source code, so the static branches can be determined without transmitting the target).

To conserve power consumption, the dedicated DSP/HSL interface is disabled until enabled by setting the Enable HSL interface bit in the core MMR. TEP instructions, software log messages and DSP address trace can be switched on independently. The HSL module is also accessible to both processors on the DPBUS. Furthermore, the HSL module can be configured via the JTAG interface. It is possible to trigger address tracing by certain events, such as TEP instructions. DMA cannot be used to transfer data to the HSL.

Additionally, the Frio core has trace capture capability, including an internal 16-element trace buffer with optional nested loop compression feature. However, the trace buffer can only be dumped off core by either reading a PC register via the low-bandwidth JTAG interface, or by taking a trace buffer full exception and reading out the contents of the trace buffer on the ISR. This trace capture facility may be too intrusive for MIPs-intensive real-time applications.

HSL operation and DSP address trace compression

The HSLCLK is free running, derived from the DSP clock with a fixed /4 ratio. Data is transferred in cycles of up to 5 words, giving packets of 8, 16, 24, 32 and 40 bits. If there is no data ready, the interface outputs zeroes. This also provides the mechanism to synchronize the receiver hardware after reset if four consecutive null packets are encountered (in Hercules, the maximum packet word count was 4, so the proposed encoding ensures that a 5 word packet cannot be misinterpreted as a sync). The data is transmitted codeword first in order to allow the receiver to auto-detect the word length.

Serialization of TEP instructions takes priority over DSP message logging, which in turn takes priority over address trace. TEP instructions are buffered if they cannot be output immediately, while the DSP will be held in WAIT states if a previous software log cannot be output. DSP address trace packets containing PC delta values are stored in an output FIFO. If the FIFO overflows due to higher priority TEP instructions, software log messages or a sudden increase in program discontinuities, intermediate PC traces are lost and an absolute PC is sent for resynchronization.

Due to the high clock frequency of the Frio core, several compression levels are required to be able to output all PC activity via the HSL port so that the DSP address trace can be reconstructed:

1) Delta compression: Whenever a change-of-flow occurs in the instruction stream, the linear distance in-between discontinuities and the displacement yield two PC delta values that are encoded independently. The target address is stored and two delta packets are sent.

2) Loop compression: Whenever a stored *from/to* address pair is encountered, the two deltas are not sent again and the target address (from where the next linear portion is calculated from) is not updated. All repeated discontinuities are compressed that way, not just zero overhead hardware loops. Note that in rare cases, if there are conditional branches within a loop, the linear distance might even be negative, so the encoding has to take care about it. The jump/loop compression nests one level, so for any inner and outer loop, repetitions do not cause any more packets to be sent.

3) Displacements that can be statically reconstructed from the program image are not sent. The decoding system must have access to the program image to know exactly which instructions are executed. If a branch, jump or call target is immediately addressed, the decoding system has to supply the displacement.

Logging Data Protocol

A single protocol to cover the TEP instructions, software log messages and DSP address trace data is used so that they can all be routed via the same hardware link to keep the decoding hardware common. The coding scheme utilizes variable length packets and is summarized in following table.

Type	Length	Word1	Word2	Word3	Word4	Word5
synchronization & idle	1	00000000				
linear positive increment 0..63	1	01pppppp				
signed change-of-flow displacement -32..31	1	01sdddd				
linear increment/displacement -4096..4095	2	001sdddd	ddddddd			
linear increment/displacement $-2^{19}..2^{19}-1$	3	0001sddd	ddddddd	ddddddd		
signed change-of-flow displacement $-2^{25}..2^{25}-1$	4	000011sd	ddddddd	ddddddd	ddddddd	
absolute PC (31 bits, linear/displacement)	5	00001010	aaaaaaaa	aaaaaaaa	aaaaaaaa	aaaaaaaa
absolute PC (re-sync after FIFO overflow)	5	00001011	aaaaaaaa	aaaaaaaa	aaaaaaaa	aaaaaaaa
inner loop count (accumulated)	3	00001000	ccccccc	ccccccc		
single/outer loop count	2	00001001	ccccccc			
DSP log last data	4	1000n000	nnnnneee	nnnnneee	nnnnneee	
DSP log data	4	1001n000	nnnnneee	nnnnneee	nnnnneee	
DSP log header	4	1010n000	nnnnneee	nnnnneee	nnnnneee	
DSP log single	4	1011n000	nnnnneee	nnnnneee	nnnnneee	
MCU log single	4	1011n001	nnnnneee	nnnnneee	nnnnneee	
TEP instruction	2	11ttteee	ttttteee			

Symbol	Width	Meaning
p	6	positive linear delta
s	1	linear/displacement sign
d	5	linear/displacement delta
d	12	linear/displacement delta
d	19	linear/displacement delta
d	25	displacement delta
a	31	absolute PC
n	16	message data
e	9	message data (extended)
t	8	TEP instruction
e	6	TEP instruction (extended)

To be backward compatible with existing HSL capture devices and tools, the TEP instructions and DSP software log messages just use 5 of the 8 HSL data pins. Hence, their first five bits can identify all packets. To get the address

trace, all 8 HSL pins have to be connected. Additional information (extended data) may be supplied in the TEP and log messages on future versions. Actual implementation just outputs '0' here.

HSL Configuration Registers

The HSL port is configured as in Hercules, there is a master enable bit and individual enable bits for TEP instructions, software log messages and DSP address traces. In addition, address traces can be switched on conditionally, either triggered by the PC or a TEP instruction.

Register	Address	15:11	10	9	8
HSL_CFG	+0x00	unused	HSL_LOOP2_OFF	HSL_LOOP_OFF	HSL_IMM_OFF
		7	6	5	4
		HSL_TRACE_TEP_TRIG	HSL_TRACE_PC_TRIG	HSL_TRACE_EN	HSL_MCU_LOG_EN
				3	2
				HSL_DSP_LOG_EN	HSL_TEP_EN
					1
					HSL_DIVR
					0
					HSL_PORT_EN

Register	Address	15:0
HSL_Tpch	+0x01	PC[31:16]
HSL_Tpcl	+0x02	PC[15:0]
HSL_Ttrig	+0x03	TEP[7:0]
HSL_Tcnt	+0x04	COUNT[15:0]

Trace trigger PC high word
Trace trigger PC low word
Trace TEP trigger instruction
Trace trigger instruction count (1 = 256k cycles)

- The HSL_PORT_EN flag is a global Enable for the HSL Port. When reset, the output clock HSLCLK will be off (low), and the module will be held in reset, clearing all pending outputs. It must be set to enable logging, but should be cleared to save power when not required.
- The HSL_DIVR bit defines the HSL output clock frequency, derived from the DSPCLK, where

DIVR	Division Ratio
0	8 (75 MHz)
1	16 (37.5 MHz)

- The HSL_TEP_EN flag enables TEP instructions to be output via the HSL port.
- The HSL_DSP_LOG_EN, when clear, dis ables all DSP messages from being output. When set, short and long DSP messages will be filtered according to the MASK register settings.
- The HSL_MCU_LOG_EN enables short MCU messages to be sent. When set, short MCU messages will be filtered according to the MASK register settings.
- The HSL_TRACE_EN enables the DSP address trace to be output via the HSL port. If the instruction cycle counter HSL_Tcnt is non-zero, its value determines how many 256k instruction cycles will be compressed and output before the address trace stream is shut off.
- If HSL_TRACE_PC_TRIG is set, DSP address trace starts as soon as PC and the address trace startpoint HSL_Tpch /HSL_Tpcl register match. HSL_Tcnt is considered, if set.
- If HSL_TRACE_TEP_TRIG is set, DSP address trace starts as soon as a TEP instruction equals the HSL_TEP_TRIGGER register. Again, HSL_Tcnt is considered, if set.

Three additional configuration register flags allow to control the granularity of address trace compression.

- By setting HSL_IMM_OFF, immediate target optimization is turned off. Every discontinuous displacement is sent and the external logger does not need a program image to uncompress the address trace information.
- If HSL_LOOP_OFF is set, loop compression is disabled, both for single level and two -level nested loops.
- If HSL_LOOP2_OFF is set, nested loop compression is disabled. Single level loops are still compressed.

These HSL configuration registers are also JTAG accessible, see JTAG instruction and scan chain details.

HSL TEP Data Format

No change from Hercules MicroSM instruction encoding. The delta time is stripped, leaving 8 instruction bits. Only 5 bits of the HSL port are used, the remaining 3 bits per packet word are set to zero.

HSL DSP and MCU Log Data Format

No change from Hercules besides that there is an additional short MCU message packet and an additional MCU short packet register for that aim. However, since both DSP and MCU have access to these registers, DSP software could trigger an MCU short message, and vice versa. Short DSP and MCU log messages cannot be differentiated if only 5 data bits of the HSL port are being used, see packet encoding table. For the packet payload, just 5 bits of the HSL port are used and the remaining 3 bits are set to zero.

The HSL DSP and MCU software log registers are Write-only.

Register	Address	15:0
LAST_DATA	+0x05	DATA[15:0]
DATA	+0x06	DATA[15:0]
HEADER	+0x07	DATA[15:0]
DSP_SINGLE	+0x08	DATA[15:0]
MCU_SINGLE	+0x09	DATA[15:0]

DSP Address Trace Data Format

On discontinuities in the DSP program flow, the encoding yields a linear and a displacement delta. Since the linear portion is usually an unsigned value, short trace packets prefixed by '01' may contain either signed (displacement) or unsigned (linear) delta data. The linear portion and the displacement are always sent as a pair (with immediately addressed discontinuities being the exception), so the receiver knows if a packet contains signed or unsigned delta data. All other delta packets contain signed data, hence do not differentiate linear and displacement values. If either value exceeds $\pm 2^{25}$, an absolute PC is sent.

A 32-entry FIFO stores outgoing address trace packets. The FIFO may overflow on executing code with frequently occurring discontinuities that have significant displacements. On the other hand, certain code may allow enough headroom for TEP and software log messages to be interleaved by trace packets without losing intermediate address traces. Once the FIFO overflows, any pending multi-word trace packet will be completely sent before the HSL flushes the FIFO. A trace packet containing the actual PC will be sent for resynchronization. Such re-sync PC packet can be distinguished from an absolute PC sent due to a delta linear or displacement overflow since they have different headers.

The HSL compresses loops up to one nesting level. Loop discontinuities are only sent once. If the loop completes, the loop count is sent. Nested inner loops are accumulated. If a nested loop completes, the outer loop count and accumulated inner loop count are sent.

Address trace packet details:

Words	Header	Description
1	01pppppp	linear PC delta, $0 \leq \Delta PC \leq 63$
1	01sddddd	displacement PC delta, $-32 \leq \Delta PC < 32$
2	001sddddd	PC delta, $-2^{12} \leq \Delta PC < 2^{12}$
3	0001sddddd	PC delta, $-2^{19} \leq \Delta PC < 2^{19}$
4	00001sdd	PC delta, $-2^{25} \leq \Delta PC < 2^{25}$
5	00001010	absolute PC if $\Delta PC < -2^{25}$ or $\Delta PC \geq 2^{25}$
5	00001011	absolute PC if address trace FIFO overflows
3	00001000	accumulated inner loop count, $2..2^{16}-1$ (saturated)
2	00001001	single/outer loop count, $2..255$ (saturated)

HSL Message Filtering Support

Same register set as on Hercules, four individual mask settings can be configured:

Register	Address	15:14	13:0
MASK 00	+0x0a	reserved	MASK[13:0]
MASK 01	+0x0b	reserved	MASK[13:0]
MASK 10	+0x0c	reserved	MASK[13:0]
MASK 11	+0x0d	reserved	MASK[13:0]
STACK	+0x0e	FILTERSTACK[15:0]	

The STACK register is used for debugging purposes only, and is read-only. The MASK registers are R/W.

DSP Address Trace Interfacing

- 31-bit PC bus (all instructions are 16-bit aligned, hence skip LSB)
- completion-of-instruction (indicates that data on PC bus is valid, write-back pipe stage)
- change-of-flow (PC discontinuity due to jump/branch/call/return/loop/interrupt/exception)
- immediate-target (displacement encoded in op-code [upper bits 0001.../0010.../1110001...])

The HSL monitors the instruction flow by listening to the PC bus, capturing its current value if the completion-of-instruction signal is enabled. A change-of-flow signal is essential to distinguish compound 32/64-bit instructions from short jumps or branches +2/+4. An additional immediate-target signal indicates that the change-of-flow target is immediately addressed as in conditional branches, calls and certain jumps, where the displacement is part of the op-code. These instructions start with instruction bits [MSB...MSB-6] being 0001XXX (conditional branches), 0010XXX (short jumps) and 1110001 (calls or long jumps).

a

Monaco
Patent: PLL - Clock before Lock
Technical note

File name: Document2

Status: Draft

Author: Joern Soerensen, ADI-DK

Revision: rev 0.2~~rev 0.1~~

When	Who	What
Aug 24 th 2001	<u>Joern</u> <u>Soerensen</u> <u>Soerensen</u> <u>Soerensen</u> <u>Soerensen</u>	Initial document

[illegible]

Table of contents

1. Introduction 1
1.1 Overview 1
1.2 References 1

TCL 1022

1. Introduction

This document gives an overview of how the wake-up time of a PLL circuit can be improved.

1.1 Overview

PLL circuits are being used in many cases to generate a faster clock frequency inside chips than the clock being supplied from external.

The reference clock in a GSM environment is fixed at 13 MHz. For noise and cost reasons this clock is being fed to all individual chips in the system. The chips that require a higher frequency clock will internally multiply the clock by using either a digital or an analog PLL (Phase Locked Loop).

The time it takes to acquire phase lock (Lock time) is critical as this is seen as latency before an interrupt can be serviced. In many cases it has not been possible to sleep the PLL simply because the lock time cannot be absorbed in the system.

The lock time and the fact that the PLL may not be stopped in certain cases can be seen as additional power consumption of no value and will directly affect the standby and operational time of battery operated equipment.

1.2 References

2. Prior art

Many methods have been used to reduce the lock time of the PLL circuit to a minimum. These include making faster loop filters, which in return makes the PLL more sensitive to noise jitter on the input clock.

Another method is store a value representing the voltage across the capacitors in the loop filter, just prior to powering the PLL down, and then restore this value when restarting the PLL. If this value is stored as an analog value the leakage and noise will make this value unreliable over time.

If the value is stored digitally this may also become unreliable overtime, not because the value is wrong, but the operating conditions for the PLL may have changed (Like temperature and supply voltage). Storing these values digitally will also require an ADC and a DAC to store and restore the values, which means area and complexity impacts.

In many cases the PLL multiplier ratio has to be changed, which involves stopping the output of the PLL, changing the multiplier setting, wait until the PLL is locked and then re-enable the clock output. In this case any stored values makes no sense and has to be discarded.

Yet another problem is that it is not easy ~~(not to say impossible — someone may have solved this)~~ to detect when a PLL is locked. Result is that one always has to consider the worst-case number in all

cases across all operating conditions. As a result the PLL is often locked and stable in a fraction of the worst-case time indicated by a lock timer.

3. Description of the invention

By looking at the system design it turns out that the clock driving the processors is not needed to be phase accurate (Locked) with respect to the input clock. What is needed is just a clock that can advance the processor to its next instruction cycle, without violating the processor and system clock input requirements, but at a sufficiently high clock rate that useful work can be accomplished.

The idea is not to improve PLL lock speed, on the contrary. The idea is to output a clock, which can be used to clock the processors as fast as possible, during the phase lock acquisition phase.

By talking to the PLL designer it turns out that the PLL can be designed such that it will output a clock that is not locked, but does not contain any frequency components that are higher than the final target frequency in a fraction of the time it takes to acquire phase lock.

It also turns out that the PLL in addition can be designed such that it will not output any spurious clock spikes or pulses during its warm-up, but will soon after powering-up output a clock, which is nicely squared and duty cycled but at a lower frequency than the final target frequency.

A final advantage of this PLL design is that changing from one clock frequency to another clock frequency is a first order response function, which ensures that no spurious clocks outside of the desired range will be generated.

So the proposal is simply to use the clock from the PLL as is, without any waiting period to assure that the PLL is locked.

With the above proposal it turns out that the following problems have been solved:

- 1) A clock is being generated with very low latency
- 2) No fixed timer to assure lock. On the other hand this also gives a non deterministic time from the wake-up signal is supplied to a clock is being output from this circuit.
- 3) It is not necessary to switch the clock off during a change of PLL multiplier ratio.

As yet another benefit the bandwidth of the loop filter in the PLL can be reduced (Increasing the lock time), which will improve the noise sensitivity.

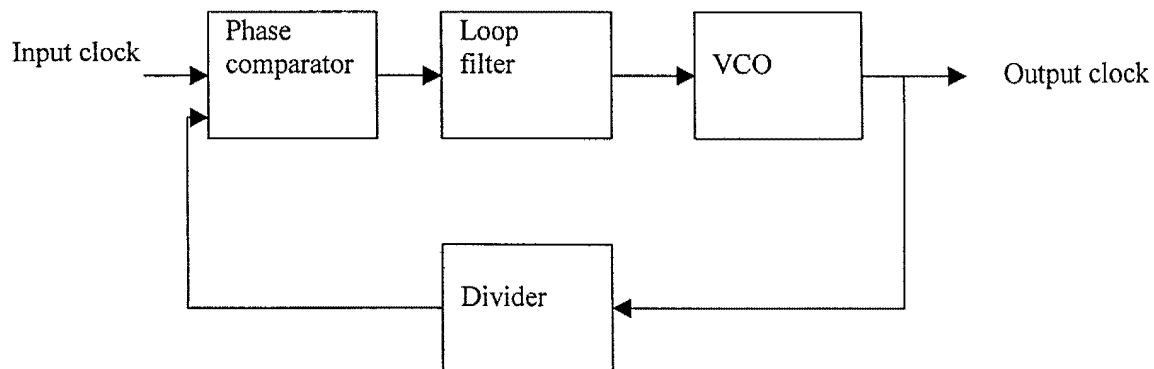


Figure 1: PLL block diagram.

The phase comparator compares the phase relation between the input clock and a divided output clock. The output of the phase comparator is filtered through the loop filter and the output drives VCO (Voltage Controlled Oscillator). The VCO is running at the output frequency, which is divided in the divider. The output of the divider should ideally be the same frequency and in phase lock with the input clock.

The PLL multiplication ratio is defined by the divider, such that if the divider is set to say 10 the output clock will be running at 10 times the input clock.

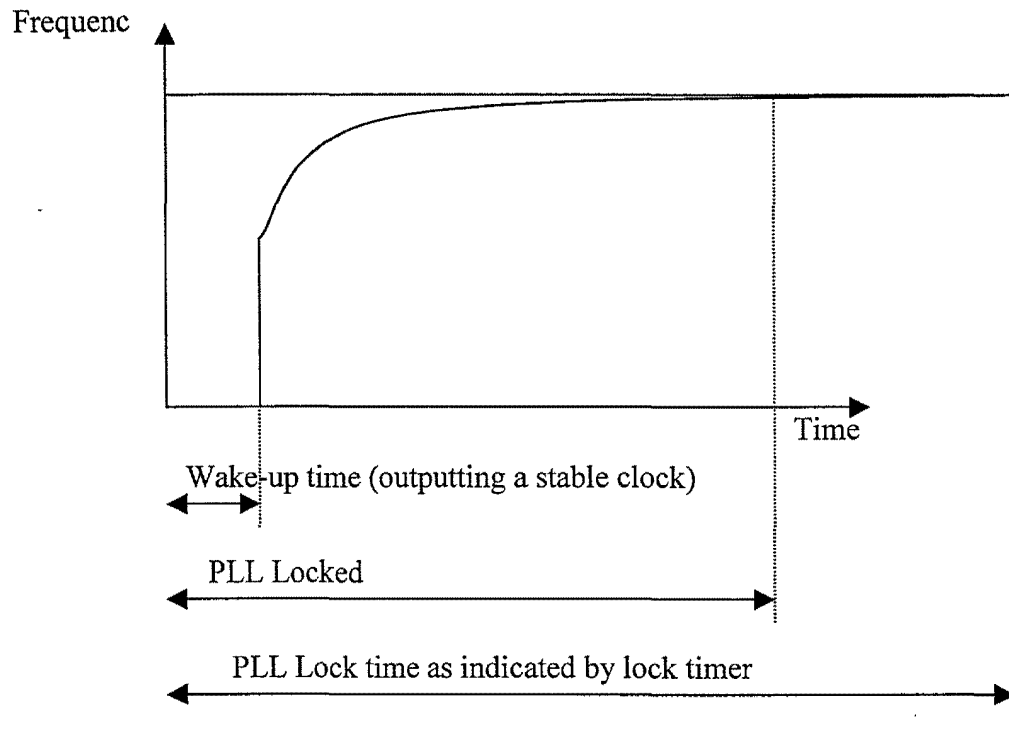


Figure 2: Proposed improvements

In Figure 2, the three indicated times are 1) The time to a stable but not locked can be output, 2) The time it really takes for the PLL to lock and 3) The time that the lock timer must indicate to ensure that the PLL is guaranteed to be locked.

----- END -----

eGSP Patent-able items ?

I) eGSP overall Architecture & I.S.

Serial Communication Interfaces are the main peripheral requirement (in number and variety) in today's Digital Signal Processing systems (DSP). An optimum system would adapt itself to any customer specific requirement. That can only be achieved through a SW configurable interface definition. The eGSP is a small Byte microController targetted at the Serial Communication interfaces emulation. Its Instruction Set (I.S.) and architecture enable a dynamic configuration of its Hardware to behave according to most of the existing Serial protocol standards.

I.a The eGSP macro-architecture : "Symetrical design"

It's a bi-core to enable FD protocol emulation at high speed.

Each core has dedicated HW resources to avoid conflicts between them for autonomous instruction execution. Two stages pipeline control flow (Decode, Execute) to improve the maximum control frequency and enable a dynamic clock network control. A common set of HW resources are defined to enable HD/FD protocol emulation. That mostly deals with a Rx Hardware, a Tx Hardware, a Generic IO logic and a clock network. A highly configurable clock generator that allows different clock domains/frequencies for the instruction executions, shifting hardware and the Serial clock lines. Two exceptions (one per core), to allow the hardware to interrupt the microcode execution when detecting a User configured context calling for a priority processing (such as collision detection).

I.b The I.S. :

20 Byte and Half-word instructions.

Parallel Pair execution supported to improve the control flow latencies and thus help supporting higher bandwidth interfaces.

Configuration related instructions (LD, DBLD, ANDOR, MAP)

Operator control related instructions (TRIGGER, ANDOR)

Control flow related instructions (JPSR, JUMPA, CALLA, LOOP, SWRESET, RET, NULL)

Time control related instructions (DLY, WAIT, IDLY)

Clock control related instruction (CLK)

Instruction Pair Conditioning (CONDEXE)

II) Highly configurable Clock Generator :

Master clock Muxing/ Clock Mode.

Primary clock divider (0 phase delay)

Two Pairs of Secondary clock Generators (0 phase delay, Burst/Continuous, off-state, start-state, stall-mode)

Configurable clock network (Serial / Shift clocks)

Clock line modulation

Low Power clock gating

III) Rx/Tx Hardware.

Macro architecture of an Rx/Tx HW operator.

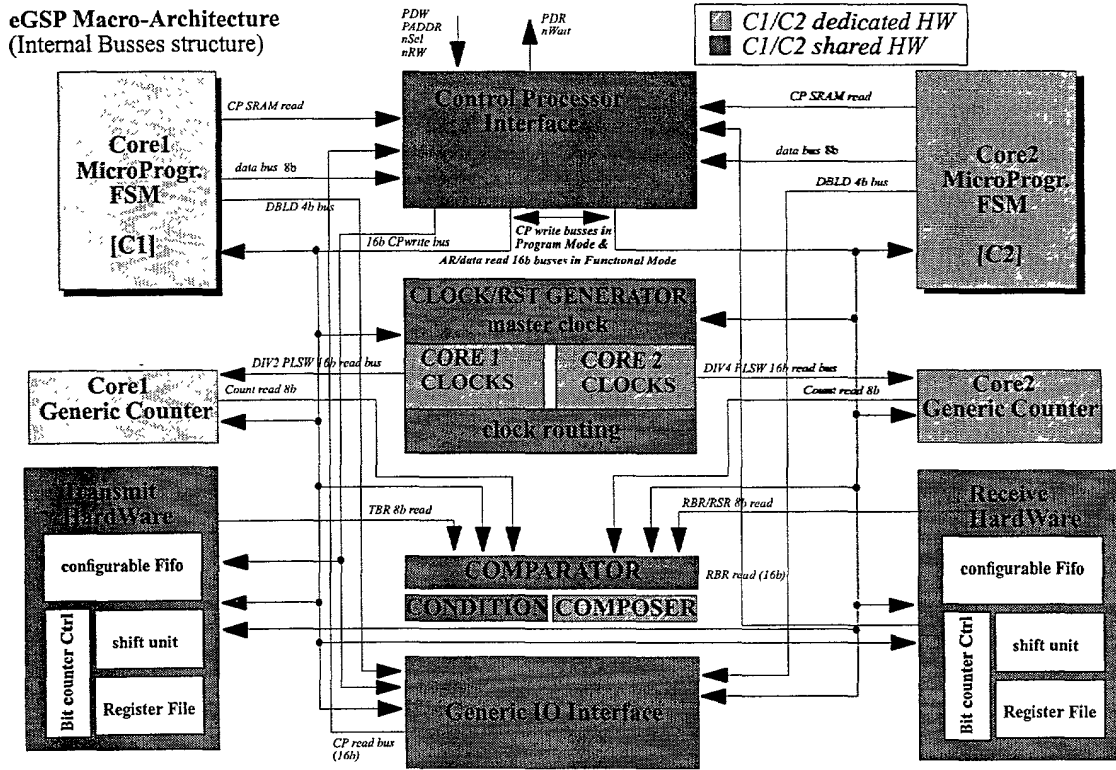
Fifo Configurable buffering

Bit counter control based sub-unit (dataSize, parity, overflow, underflow)

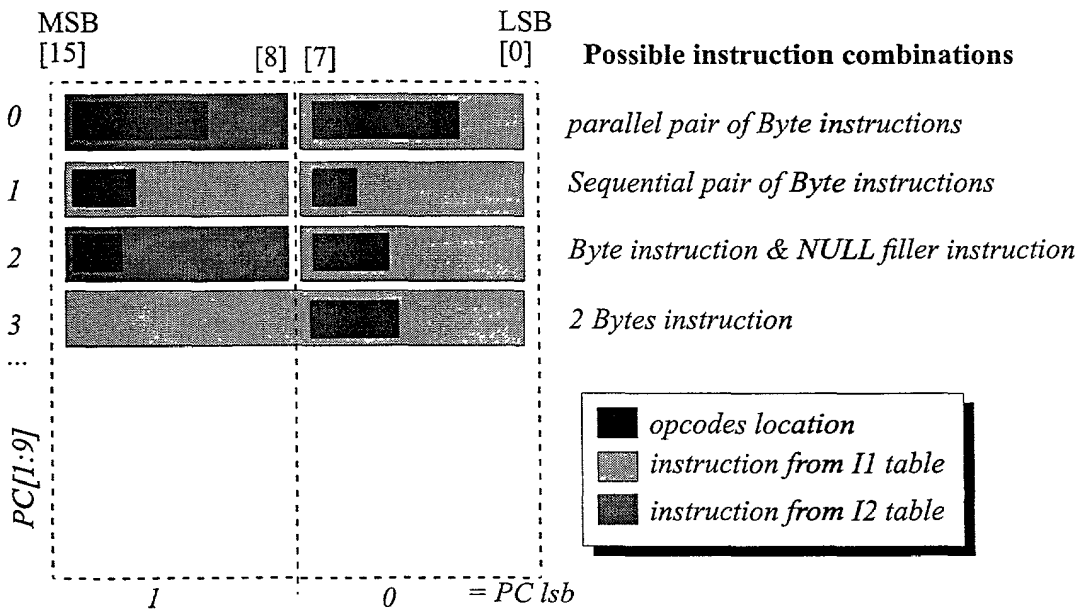
Shift/Load Register.

clock = control (default is shift)

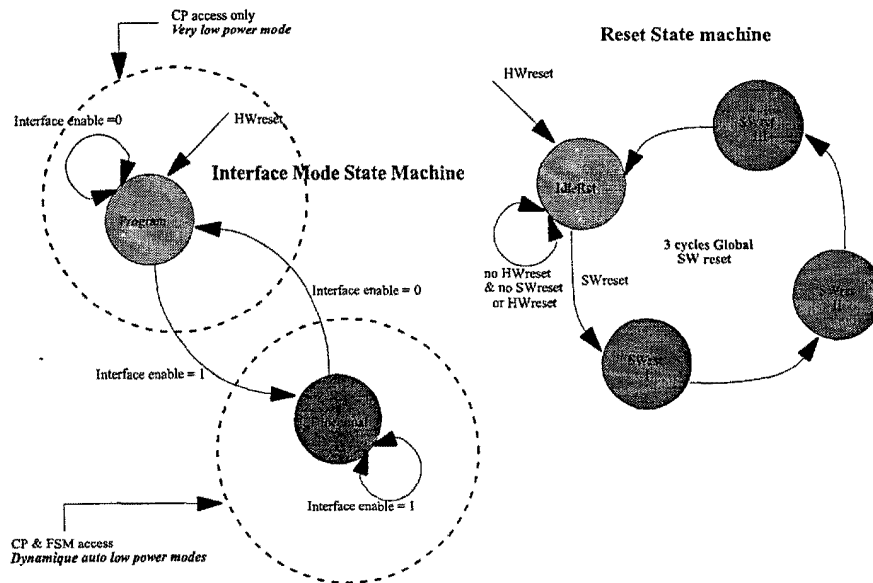
Dual Core eGSP Macro-architecture: Symmetrical design - Clock controlled operators



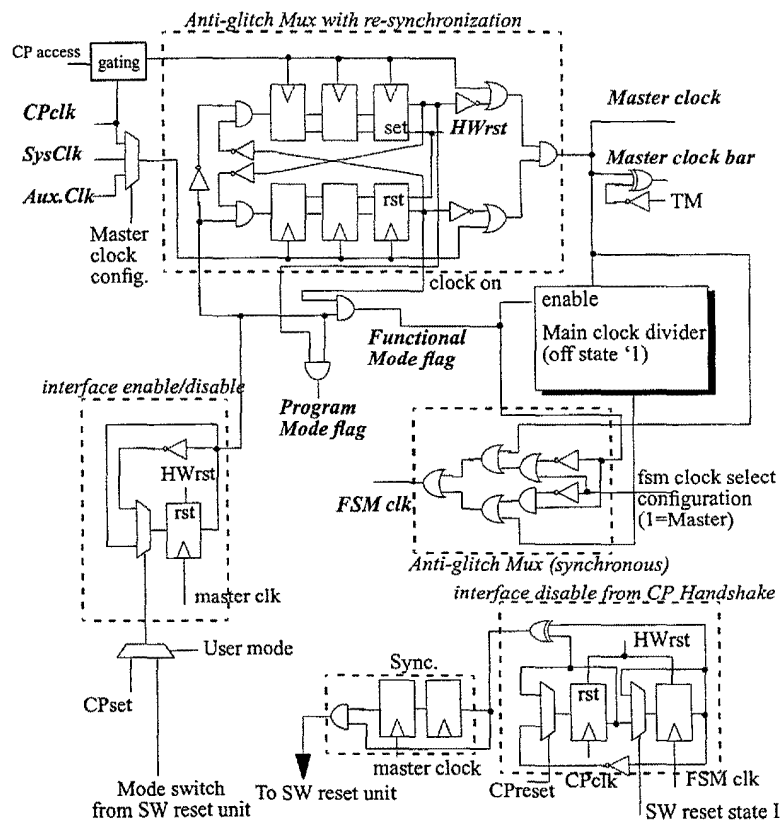
Core Memories: Instructions alignment

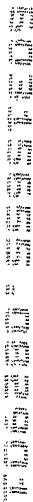


Mode Control: Highly configurability from CP in Prg Mode => ucode reduction.
Very low off state power.



Highly Controllable clock network. Mode control.

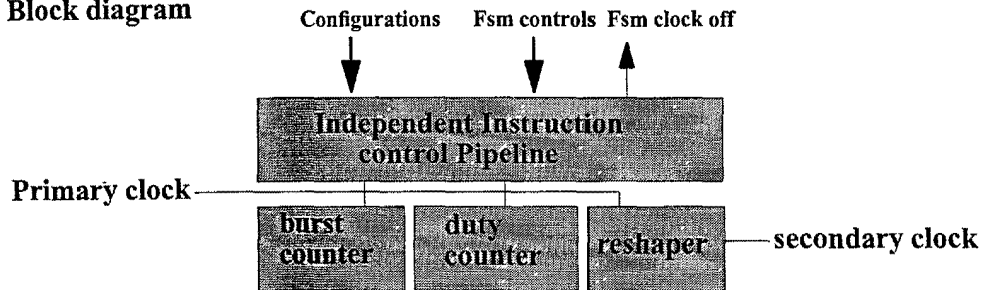


[illegible]

Highly controllable clock dividers (Instruction controlled).

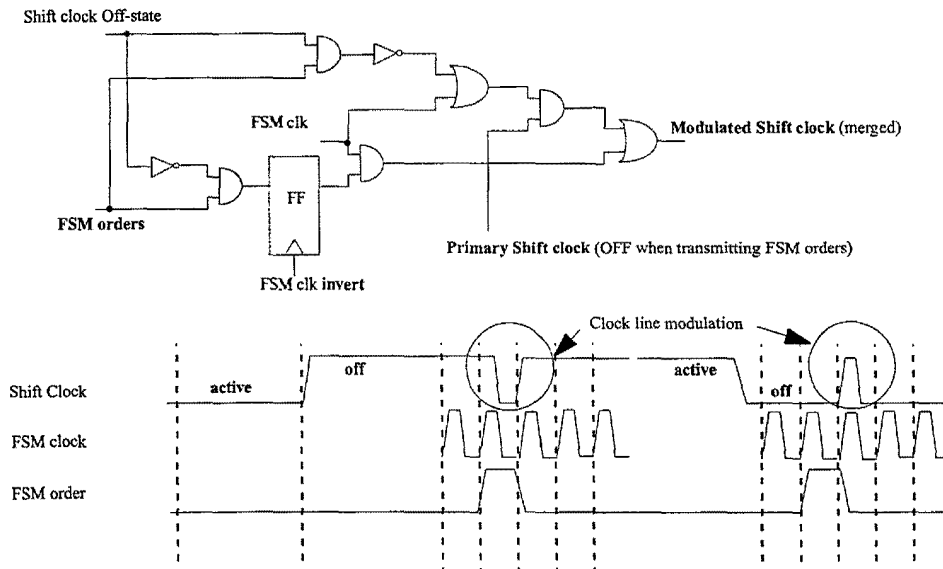
- Zero delay secondary clock vs. Primary clock (Re-shaper)
- Burst/Continuous Modes
- Configurable off-state/state state
- Low power Burst Mode (stalls the FSM clock)
- Configurable stall mode (full burst/Burst - 1)
- Independent divide ratios for low/high states.

Block diagram



Shift Clocks FSM order transmission: Clock modulation

Shifting Clock line modulation to transmit FSM orders (Merge):



Instruction Set:

- ☐ Byte/Halfword instructions.
- ☐ 2 stages pipeline.
- ☐ single cycle execution.
- ☐ in-order execution.
- ☐ Minimum branch/interrupt latency.
- ☐ Parallel Byte instructions execution.
- ☐ Conditional executions.

Table 1: I1 - Primary I1 instruction set encoding

Instruction \ Format	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DLY									0	0	0	d	d	d	d	d
JPSR									0	0	1	o	o	o	o	o
TRIGGER									0	1	0	i	v	bi	bi	bi
DBLD									0	1	1	i	v	v	bi	bi
CMPRD	d	d	d	d	d	d	d	d	1	0	0	0	cc	cc	il	il
A=A [&,,] mask	m	m	m	m	m	m	m	m	1	0	0	1	s	i	i	i
LD	d	d	d	d	d	d	d	d	1	0	1	0	i	i	i	i
WAIT	c2	c2	c2	c2	c1	c1	c1	c1	1	0	1	1	m	m	v2	v1
DBLD_IO45									1	1	0	0	v	v	bi	bi
if (cond) execute //									1	1	0	1	v	i	i	i
CMPRR	cc	cc	i	i	i	i	i	-	1	1	1	0	0	0	-	-
CLK	de	d	d	d	d	d	d	d	1	1	1	0	0	1	cd	cd
JUMPA	a	a	a	a	a	a	a	a	1	1	1	0	1	0	a	a
CALLA	a	a	a	a	a	a	a	a	1	1	1	0	1	1	a	a
extend //									1	1	1	1	0	0	i	i
IDLY									1	1	1	1	0	1	i	i
LOOP	o1	o1	o1	o2	o2	o2	o2	o2	1	1	1	1	1	0	0	o1
SWRESET									1	1	1	1	1	0	1	0
RET									1	1	1	1	1	0	1	1
reserved									1	1	1	1	1	1	-	-

Table 2: Parallel I2 Instruction set encoding

Instruction \ Format	15	14	13	12	11	10	9	8
JPSR	0	0	o	o	o	o	o	o
reserved	0	1	0	0	-	-	-	-
IDLY	0	1	0	1	0	0	i	i
reserved	0	1	0	0	0	1	-	-
RET	0	1	0	1	1	0	0	0
SWRESET	0	1	0	1	1	0	0	1
reserved	0	1	0	0	1	0	-	-
reserved	0	1	0	0	-	-	-	-
DLY	0	1	1	d	d	d	d	d
TRIGGER	1	0	0	i	v	bi	bi	bi

Table 2: Parallel I2 Instruction set encoding

Instruction \ Format	15	14	13	12	11	10	9	8
DBLD	1	0	1	i	v	v	bi	bi
reserved					-	-	-	-
MAP	1	1	0	i	i	i	i	i
DBLD_IO45	1	1	1	0	-	-	-	-
NULL	1	1	1	1	1	1	-	-
reserved							-	-
reserved							-	-

A parallel execution is enabled by the opcode decoding of both Byte instructions. The following pairs are allowed:

I1

TRIGGER
DBLD
DBLD_IO45
any I1 Byte
extend //
if (cond) execute //

I2

TRIGGER,DBLD, DBLD_IO45, MAP,NULL
TRIGGER,DBLD, DBLD_IO45, MAP,NULL
TRIGGER,DBLD, DBLD_IO45, MAP,NULL
NULL
any I2 with effect only on TRIGGER,DBLD
any I2

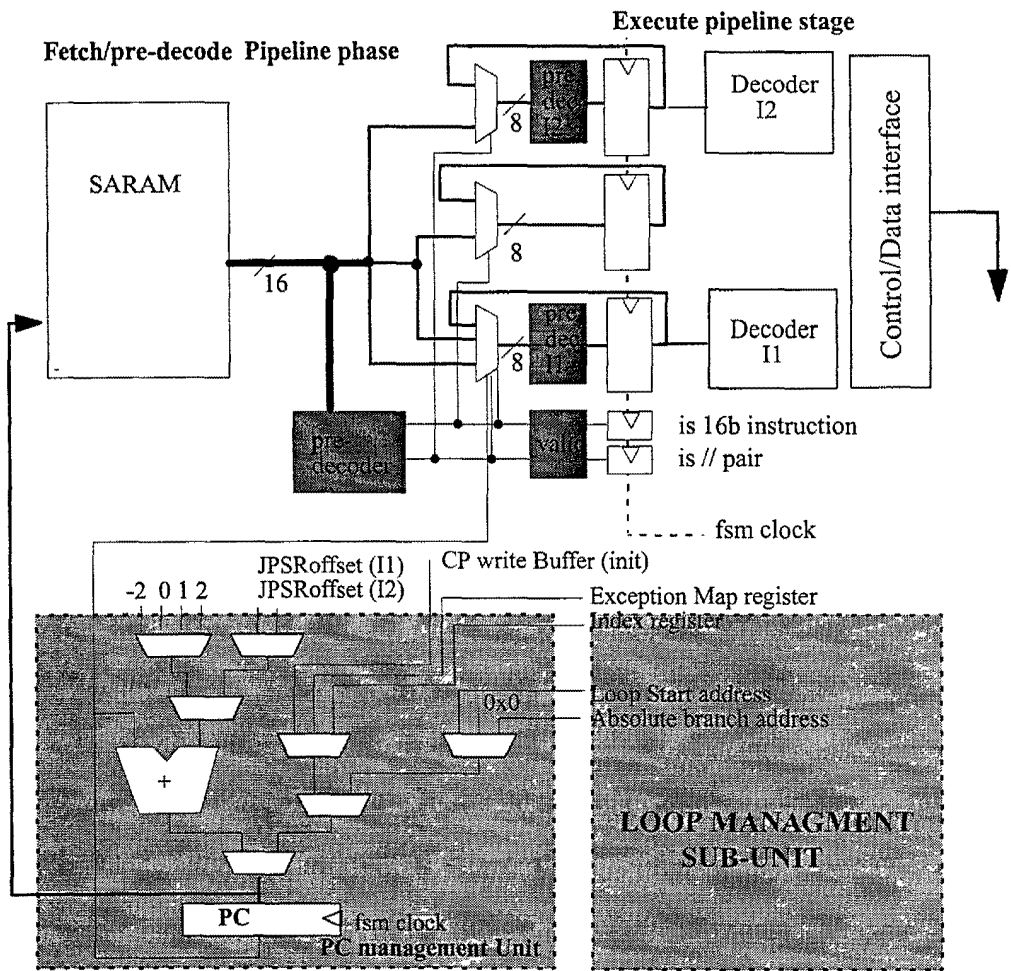
eGSP pipeline organization:

:

Stage\Time	T	T+1	T+2	T+3	T+4
FETCH-DECODE	OUI	CFI	CFI	OUI	
EXECUTE		OUI	CFI	CFI	OUI

fetch-decode
execution
whatever

Core Sequencer micro-architecture:



State	State Field
PROGRAM	000
A	001
B	010
C	011
D	100
E	101
F	110
G	111

Company Confidential

4-bit Voltage Field

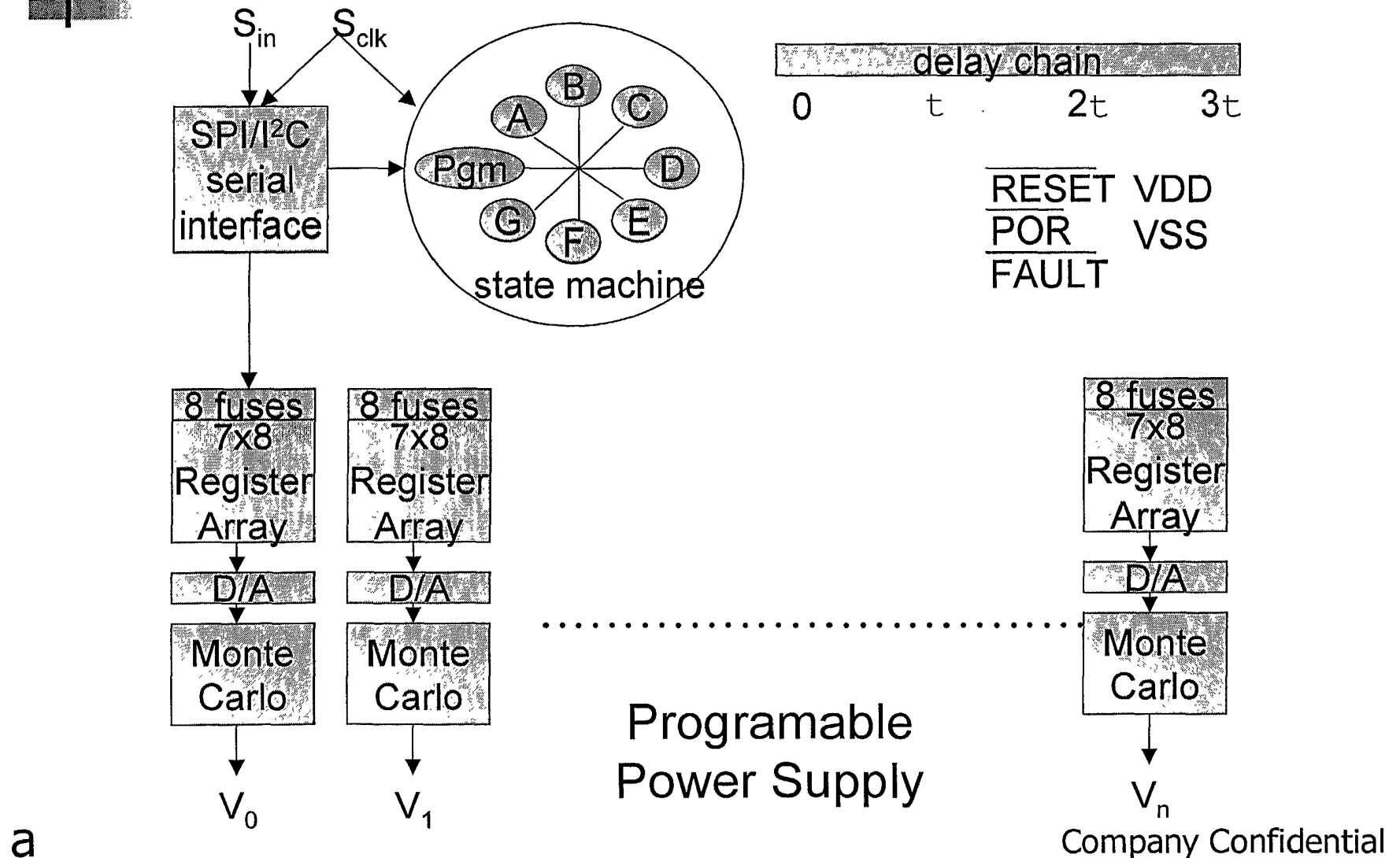
2-bit Delay Field

● ● ● ● ● ● ●

Voltage (V)	Voltage Field
3.3	1111
2.5	1110
1.8	1101
1.6	1100
1.5	1011
1.4	1010
1.3	1001
1.2	1000
1.1	0111
1.0	0110
0.9	0101
0.8	0100
0.7	0011
0.0	0010
-0.5	0001
shut down	0000

Company Confidential

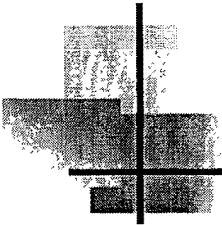
Block Diagram



a

Features

- Any number of “Monte Carlo” power supplies
 - multiple price points
- SPI or I²C serial interface
 - choose which interface by fuse or bonding option
- Register Arrays
 - stores user programmable power and sequencing settings
- State Register
 - Choose PROGRAM or one of 7 user defined states
 - e.g. RUN, SLOW, NAP, SLEEP, etc.



Features

- Fuses
 - set application specific initial state on power up
 - limit maximum voltage, avoid damage due to program errors
- Delay Chain
 - timing for programmable power change sequencing
- POR_ (power on reset open drain output pin)
 - low from entry into state PROGRAM until last power-up sequence completes
- FAULT_ (battery fault open drain output pin)
 - error flag for processor interrupt
- RESET_ (input pin)
 - go to PROGRAM state for wake-up

a

Monaco
Improving MCU performance
Technical note

File name: MCU improvements.doc

Status: Draft

Author: Joern Soerensen, ADI-DK

Revision: rev 1.0

When	Who	What
Aug 28 th 2001	Joern Soerensen	Initial document

[illegible]

Table of contents

1. Introduction	3
1.1 Overview.....	Error! Bookmark not defined.
1.2 References.....	Error! Bookmark not defined.

1. Introduction

This document gives a description of how speed can be improved by utilizing the burst mode capabilities of modern FLASH type memories.

A burst mode flash memory is a memory type in which accessing the first word of burst will take about the same time as for a traditional flash memory (70-120 nS typical), while accessing the following words will be much faster (20 – 25 nS).

Some of these burst mode memories does have limitations on the size of the burst and may also have some address boundary restrictions (Page mode devices). Yet some of these memories does have certain address wrap capabilities, which are optimized for cache functionality.

The FLASH burst mode memories of interest for this invention are the types that supports continues burst, where there is no restriction on address boundary for the first access and either supports unlimited burst length or the burst length is so long that it does not affect the usage in the context of the invention.

2. Background

The processing speed of the MCU system in Hercules is largely determined by the associated memory systems, in particular of the external FLASH memory system.

One way of improving the processing speed in Hercules is to copy code from the external memory system to the internal memory SysRam, which runs at processor speed has a larger bus width, two circumstances that allows for much higher operational speeds.

Main issue with this solution is the size of the code base, which easily can get into the area of 1 Mbyte of code space, whereas the internal memory system is only 128 Kbyte. Another issue is that the internal memory system is intended for data storage and not as a copy of external code.

An obvious option to improve MCU speed is to add cache to the MCU system. The code structure for a typical communication protocol stack does unfortunately prevent much improvement unless the cache system is very large. The reason for this is that the protocol code is written in such a way that most of the code is being touched at each pass of the code.

The code structure is largely a lot of function calls. Each function contains large state machines, which are constructed of switch-case statements. Inside each case statement conditional code builds the functionality of the state machine. The result of this is that very large portion of the program flow

is discontinuities, resulting in pretty poor cache performance, unless the code is resident for the next pass, which is unlikely due to the large code base and limitations in cache size.

3. Description of the invention

None of the above thoughts really changes with the introduction of burst mode FLASH memories. Cache memories can be filled quite a bit faster, improving over all processing speed, but lacks all the same issues described in chapter 2.

An obvious method would be to introduce burst buffers, which should be used to read a head of the MCU processor, and thus ensure that the burst mode capability of the flash memory is being utilized rather than just the single access speed.

Example: Reading a burst of 4 words will result in a possible total access time of $70+20+20+20 \text{ nS} = 130 \text{ nS}$, where as reading these as single words would result in $4*70 \text{ nS} = 280 \text{ nS}$.

Problem in the above case is however that the ARM processor may only need a few of these instructions due to the program structure, thus as a result words being read from the flash memory may be trashed due to a program flow discontinuity. So say only two words are being used results in a 130 nS access time to get two words (4 words get fetched while only two is being used). This does only represent a minimal improvement (130 nS versus $2*70 \text{ nS}$).

Utilizing a burst buffer can be seen as a cache, which is far too small, and may not improve performance to the degree you'd expect.

3.1 The invention

It also turns out that the MCU in Monaco runs at clock speeds fast enough that the currently available burst mode memories can not output data faster than the MCU can consume these. As a result the burst buffer does only become a temporary storage for the instruction stream.

The ARM7TDMI micro controller unit used in Hercules and also in Monaco, does output a signal that indicates if the next access is in sequence with the current or not (next address = current address + 1). This can be used as an indicator to the bus logic to continue reading in burst mode or to cancel the read.

When using a continues burst mode FLASH, with no limitation of the burst size and no address boundary limitations, the above feature can be used to its full extent, such that the first access will take the initial access time of say 70 nS, while all following accesses will take the incremental access time of 20 nS.

Examples of possible improvements:

In the following example an analysis has been performed, where the size of the instruction buffer is set to 4 words, and 6 or 12 instructions are executed in a row as examples. In both cases the instructions are spread across 3 blocks fitting in the instruction buffer. It is assumed that address boundary restrictions exist in the instruction buffer case.

The first two column represents two different instruction sequences, one where a block of 12 instructions in a row are executed, one where a block of 6 instructions are executed in a row.

Third column represents the same two scenarios when using a burst buffer with 4 words in the buffer. In this case there will be no difference between a worst case and best case as, the interface will always read a full block (In this case 4 words).

Fort and fifth row represents "a continues burst with variable length" (also a worst case of 12 instructions and a best case of 6 instructions).

Normal 70 nS		Burst Buffer	Continues burst with variable length	
4*70	1*70	70+3*20	1*70 + 3*20	1*70
4*70	4*70	70+3*20	4*20	4*20
4*70	1*70	70+3*20	4*20	1*20
840	420	390	290	170

As can be seen from the above table significant improvements can be obtained using the principle of continues burst with variable length.

If the block size for the burst buffer is increased it turns out that in some circumstances a burst buffer may even decrease performance:

The following table represents the same number of instructions executed as above, but with a burst buffer size of 16 words. It is assumed that the block of instructions (6 or 12 words) is contained within the same block in the burst buffer.

Normal 70 nS		Burst Buffer	Continues burst with variable length	
12*70	6*70	70+15*20	1*70 + 11*20	1*70 + 5*20
840	420	370	290	170

The last table represents the same scenario, but the block of instructions is split across two blocks:

Normal 70 nS		Burst Buffer	Continues burst with variable length	
6*70	3*70	70+15*20	1*70 + 5*20	1*70 + 2*20
6*70	3*70	70+15*20	6*20	3*20
840	420	740	290	170

In the above case it turns out that the burst buffer may actually perform worse than a standard FLASH, making it very difficult to judge if a burst buffer should be utilized without knowing any statistics about the code being executed.

The principle of continues burst does however always represent a significant improvement over standard flash memory independent of the code statistics. The only case where the new principle would run at the same level of performance is if there is a discontinuity at every instruction. This case is not possible, as this would represent a series of jumps, which would really generate no useful code.

----- END ----- END -----

a

Cache Support for Real-time Multithreading IP Disclosure

File name: Document3

Status: Draft

Author: Michael Allen, ADI-AUS

Revision: revision 0.0

When	Who	What
August 27, 2001	Michael Allen	Initial document

[illegible]

1. Introduction	4
1.1 Overview.....	4
1.2 References.....	5

1. Introduction	4
1.1 Overview.....	4
1.2 References.....	5

1. Introduction

This document discloses a mechanism for improving cache performance in a low-cost, embedded system that employs multiple software threads, or tasks, which can be scheduled opportunistically by the real-time operating system scheduler.

2. Overview

Low cost, embedded processor systems face unique performance challenges. One of these is the constraint to use low-cost, slow memory, while maintaining high throughput.

As an example, in a system such as a cell phone, a digital signal processor is often employed for the compute-intensive tasks. This is a real-time system, where synchronization with the air interface must be maintained. In this system, low cost FLASH memory forms the bulk storage capacity of the system. However, the FLASH memory access time is much longer than the minimum cycle time of the DSP. To achieve high performance on the DSP, it must execute from local memory which is much faster than the (off-chip) FLASH memory. By employing code and/or data caches as local memory, the memory model is still simple, as opposed to having discreet fast and slow memory regions exposed to the programming model.

Whenever the DSP misses the cache, the cache hardware must fill a cache line from the slower memory systems underneath it. This fill-on-demand aspect of the cache often means that the DSP is stalled while all or part of the cache line is filled. However, if the software model employs multiple threads, or tasks, which are scheduled by the RTOS, there may be an opportunity to start a new thread during the cache line fill period.

The requirements of a system definition employing this invention are, then:

1. A processor with much faster cycle time than the memory sub-systems it accesses
2. Code and/or data caches between the processor and the slower memory sub-systems
3. Software modularity such that multiple threads, or tasks, are available to run on the processor at any time.
4. Special hardware to signal to the operating system that a cache miss has occurred, allowing the operating system to start up the next pending software task/thread.
5. As an optional enhancement in embedded systems with multiple memory subsystems, with different access latencies (e.g., off chip FLASH vs. on chip SRAM), the caches can employ multiple line-fill and copyback buffers to further enhance overall throughput.

3. Prior Art

While out-of-order cache line fill-hardware, and even dedicated multi-thread support, concepts have been employed for years in large compute systems (e.g., IBM Power architectures), we are not aware of any prior art in which these concepts have been employed in low cost embedded systems.

4. Out Of Order Line Fill Support

The following excerpt is from the Monaco Memory architecture document, and is specific to the Frio DSP-based Monaco digital baseband:

Adding hardware to support two cache line fills, and allowing the data to return out of order, is required (on both code and data L1) to support two threads, or contexts, running on the Frio core.

One thread belongs to the DSP context (the GSM modem code, for example). The code for this thread likely will not fit entirely in the L1 memory, but will fit in the L2 (or L3 system RAM). The other thread belongs to the microcontroller context, which likely runs from the FLASH. When the microcontroller thread misses the L1 cache, many hundreds of cycles would be lost while filling a cache line from the off-chip FLASH. By switching context (via an interrupt) to the DSP thread, overall throughput can be increased.

Note that the benefit for this out-of-order line fill capability has been offset by the inclusion of Burst FLASH capability in the L3 memory. The original motivation for this feature was due to the potentially huge latency of performing a 32 byte cache line fill from 16-bit wide, 100ns access time FLASH. That memory would require $100\text{ns} \times 16\text{ accesses} = 1.6\mu\text{s}$ to provide 32 bytes of data. That was equivalent to over 500 Frio cycles (@ 333MHz), which is a lot of wasted bandwidth. Even with a context switch overhead of 200 cycles, a lot of additional processing could be accomplished within the line fill time.

4.1 References

----- END -----



Monaco Digital Baseband Processor Design Specification

File name: Monaco Preliminary Specification

Status: Draft

Author: Michael Allen, ADI-AUS

Revision: Rev 0.3.1

The information contained in this document is commercially Proprietary and Confidential Information and must not be disclosed to third parties without prior consent.

CONFIDENTIAL

REVISION HISTORY

When	Who	What
Feb 11 th , 2001	Michael Allen	Initial document
Apr 5 th , 2001	M. Allen	Updates from workshop
May 26 th , 2001	M. Allen	Rev0.0 draft release
June 19, 2001	M. Allen	Edits from r0.0 review. Enabled change bars. Replaced the thread-based power management model w/ a common clock/voltage control mechanism. Lots of cleanup. Added GSP chapter.
July 23, 2001	M. Allen	Updates for latest TEP release, lots of low-level edits. Released for review as Rev0.1
July 23, 2001	M. Allen	Change bars reset. Released as Rev0.2
August 27, 2001	M. Allen	Scrubbed for Provisional IP filing. Added EBUS discussion. Changed DPBUS memory maps. Released as rev 0.3.1. Reset change bars.

CONFIDENTIAL

Table of Contents

1.	Introduction	13
1.1	Overview	13
1.2	Purpose	15
1.3	Scope	15
1.4	Terminology	15
1.5	References	15
1.6	Acknowledgments	16
2.	General Description	16
2.1	Overview	16
2.2	External Interfaces	19
2.3	Internal Architecture	20
2.4	Internal Modules	21
2.5	System Clocking Overview	21
2.5.1	Clock domains	22
3.	ARM7TDMI CORE CONFIGURATION	25
3.1	Changes from Hercules	25
3.2	Component Parts	25
4.	DSP Sub-system	26
4.1	Frio DSP	26
4.1.1	L1 Memory	27
4.1.2	Endianess	28
4.1.3	Frio Interrupt Support	28
4.1.4	DSP Interrupt Latency	29
4.2	DSP Memory	29
4.2.1	DSP Memory Organization	29
4.2.2	DSP Memory Performance	31
4.3	System Bus Interface Unit (SBIU)	31
4.3.1	SysL1 Bus	33
4.3.2	SysL2 Bus	33
4.3.3	DP Bus	34
4.3.4	DA Bus	34
4.3.5	DSP Bus	34

4.3.6	R Bus.....	35
4.3.7	SBIU Arbitration Control.....	35
4.3.8	SBIU Performance.....	35
4.3.9	System Synchronization (SSYNC) from FRIO Core.....	40
4.4	L2 Memory	40
4.5	DSP Coprocessors.....	40
4.5.1	Viterbi Coprocessor	40
4.5.2	Cipher Coprocessor	41
4.5.3	External Coprocessor Interface	41
4.6	DSP DMA	42
4.7	DSP Peripherals	42
4.7.1	Serial Port.....	42
4.7.2	TEP.....	43
4.7.3	High-Speed Data Logging and Trace Functions (HSL).....	43
4.7.4	Programmable Flag IO.....	44
4.8	Timing and Event Processor (TEP).....	45
4.8.1	Background.....	45
4.9	TEP Features.....	46
4.10	Functional Description	47
4.10.1	Event-Driven Layer 1 Scenario	47
4.10.2	Clock Calibration.....	49
4.10.3	DSP DMA Control	50
4.11	Hercules μ SM Backward Compatibility	50
4.12	TEP high-level Architecture.....	50
5.	External Applications Processor Interface (EAPI).....	51
5.1	EAPI Overview.....	51
5.2	EAPI Memory Protection.....	52
6.	Enhanced Generic Serial Port.....	53
6.1	Introduction.....	53
6.2	eGSP Architecture.....	53
7.	Peripheral Bus Specification.....	54
7.1	Peripherals and Interfaces.....	54
7.2	MMI Functions	56
7.2.1	MMI Module.....	56
7.2.2	Tone Generator and Sigma Delta Tone Interfaces	57

7.2.3	RTC.....	57
7.2.4	UART / IrDA (GSP A).....	57
7.2.5	Debug UART (GSP B).....	59
7.2.6	Serial Display Interface (GSP C)	59
7.2.7	Battery Management / Accessory Interface (GSP DA / GSP DB)	59
7.2.8	SIM Interface	59
7.2.9	USB Interface.....	60
7.2.10	MMC Interface.....	60
7.2.11	Cipher Coprocessor	60
7.2.12	USC Interface.....	61
7.3	Clock Generation Peripherals.....	61
7.3.1	Peripheral PLL	61
7.4	House-Keeping Functions	61
7.4.1	Watchdog Timer (WDT).....	61
7.4.2	General Timers.....	61
7.4.3	System Configuration Block.....	61
7.5	Interrupt Controller.....	62
7.5.1	Exception Sources	62
7.5.2	Interrupt Sources	62
7.5.3	Processor Interrupt Resources.....	65
7.5.4	Interrupt Controller Operation.....	65
7.5.5	Interrupt Control & Prioritization.....	70
7.5.6	DMA Interrupt Controller.....	72
7.5.7	Interrupt Controller Group Assignments.....	73
7.5.8	System and DSP DMA Interrupt Controllers.....	75
7.6	System DMA	75
7.6.1	Chaining DMA channels.....	75
7.6.2	General operation mode	80
7.7	Communication System Functions.....	81
8.	External Bus Specification.....	81
8.1	Burst mode FLASH interface.....	81
8.1.1	Performance gain when using burst mode.....	84
8.1.2	Internal Wrapping Burst Transfer Support.....	84
9.	RAM Bus Specification.....	86
9.1	RBUS	86

9.2	System RAM.....	86
9.2.1	MONACO memory maps	86
10.	SBUS Specification.....	88
10.1	MCU Clock Generator	89
10.2	Arbitration Registers	89
10.3	Bus Select Decoders.....	90
10.4	Security Features & Boot ROM	90
10.5	DBGEN Control	90
11.	System Clock Control and Power Management.....	91
11.1	System Clock Control.....	91
11.1.1	Monaco Clock States	91
11.1.2	System Clock Control.....	91
11.1.3	System Clock Control Registers	92
11.1.4	DSP Control Register	95
11.1.5	PLL Multiplier Ratios.....	96
11.1.6	System PLL Control Register.....	98
11.1.7	Peripheral PLL	98
11.2	Software Reset.....	99
11.3	Power Management.....	100
11.3.1	Background	100
11.3.2	Dynamic Voltage Control.....	101
11.3.3	Back-Bias Controller	102
12.	Emulation	102
13.	Debug and Type Approval	102
14.	System Overview.....	102
14.1	Memory Maps.....	102
15.	Oscillator Specification	108
16.	Vertical Function Descriptions	108
17.	Pin Out.....	108
18.	JTAG and Boundary Scan.....	108
19.	Manufacturing Test.....	108
20.	Bootcode ROM Listing.....	108
21.	Packaging	108
21.1	Production Pinout.....	108
21.2	Extended Bondout Package.....	108

21.3	160 Pin MINI-BGA Outline Dimensions.....	109
22.	DC and AC Characteristics	110

CONFIDENTIAL

CONFIDENTIAL

Table of Figures

Figure 1: Monaco Shared Memory Map.....	17
Figure 2: PRELIMINARY Maximum Operating Frequencies.....	18
Figure 3: External Interfaces.....	19
Figure 4 Monaco Block Diagram.....	20
Figure 5 Monaco System Clock.....	22
Figure 6: Example of Clock Frequency Variation Over Time.....	23
Figure 7 DSP Subsystem Block Diagram.....	26
Figure 8 Interrupt Control Block Diagram.....	28
Figure 9 DSP Memory Hierarchy.....	29
Figure 10: SBIU Routing Topology.....	32
Figure 11 External Coprocessor Interface.....	41
Figure 12: Simplified functional diagram for the Monaco Timebase and Event Processor.....	46
Figure 13: Layer 1 'Executer' block and the TEP module interface.....	48
Figure 14: Exemplified timing scenario of the TEP for the case of two Sequencers.....	49
Figure 15: Calibration adjustment of the free-running 32 kHz clock by fractional-N division.....	49
Figure 16: Block diagram showing the use of dedicated DMA channel for data read/write access anywhere in Monaco address space, including setting up other DMA channels.....	50
Figure 17: Simplified block diagram of the TEP architecture.....	51
Figure 18 External Applications Processor Interface.....	51
Figure 19 eGSP Block Diagram.....	54
Figure 20: Overview of the various IrDa data rates supported by the Monaco device.....	58
Figure 21: Block Diagram.....	66
Figure 22: Interrupt prioritizer.....	70
Figure 23 Priority Encoder and Bus Interface.....	71
Figure 24: DMA Interrupt Controller.....	72
Figure 25: The lower 10 address bit (PAdr[9:0]) to the DMA controller from the PBUS arbiter.....	77
Figure 26: Vertical entry for channel i in the internal DMA memory.....	79
Figure 27: Synchronous burst read operation Timing, four half word burst length.....	83
Figure 28: Asynchronous burst read operation Timing, four half word burst length.....	84
Figure 29: Wait signal timing during continuous burst read operation.....	84
Figure 30: Alias of the SRAM.....	88
Figure 31: Alias of the Boot ROM on the SBUS.....	90
Figure 32 Monaco System Clock.....	93
Figure 33: Block diagram of the System PLL.....	96

Figure 34: SYSDSPCR Control register for the DSP PLL.....	98
Figure 35 Peripheral Clock Block Diagram.....	99
Figure 36: Illustration of how the software reset is implemented.	99
Figure 37 Dynamic Voltage Control Block Diagram.....	101
Figure 40 MCU Memory Map.....	103
Figure 41 P Bus Slave Memory Map	104
Figure 42 DSP Memory Map.....	105
Figure 43 DP Bus Slave Memory Map.....	106
Figure 44 System DMA and DSP DMA Memory Map.....	107

CONFIDENTIAL

Table of Tables

Table 1: PRELIMINARY Operating Conditions	17
Table 2 SBIU Internal Routing Priorities.....	32
Table 3 SysL2 Arbitration Priority.....	33
Table 4 DPBUS Arbitration Priority	34
Table 5 DSPBUS Arbitration Priority.....	34
Table 6 SBIU Arbitration Control Register	35
Table 7 DSP DPBUS Read Access Latency	36
Table 8 DSP DPBUS Write Access Latency	36
Table 9 DSP L2 Access Time: Single Read.....	36
Table 10 DSP L2 Access Time: Burst Read.....	36
Table 11 DSP L2 Access Time: Single Write.....	37
Table 12 DSP L2 Access Time: Burst Write.....	37
Table 13 DSP System Memory Read Access Time.....	37
Table 14 DSP System Memory Write Access.....	38
Table 15 DSP DMA L1 Read Access.....	38
Table 16 DSP DMA L1 Write Access.....	38
Table 17 DSP DMA-L2 Read Access.....	38
Table 18 DSP DMA-L2 Write Access.....	39
Table 19 System-DPBUS Read Access.....	39
Table 20 System-DPBUS Write Access.....	39
Table 21 System-L2 Read Access.....	40
Table 22 System-L2 Write Access.....	40
Table 23: Programmable Flag IO Function Assignments	44
Table 24 PBUS Peripheral Access Constraints.....	56
Table 25: GSP A-I Instance Summary	58
Table 26: Minimum, nominal, and maximal pulse duration specs. for the 2,4kb/s to 115,2kb/s data rates.	59
Table 27: Minimum, nominal, and maximal pulse duration specs. for the 0,576 Mb/s and 1,152 Mb/s data rates.	59
Table 28 : System Exceptions.....	62
Table 29: System Interrupts.....	65
Table 30: Interrupt Group Definitions.....	67
Table 31 : Interrupt Group Assignments for the MCU and DSP.	74

Table 32: Interrupt sources for the DMA	75
Table 33 DMA Internal Address Map	78
Table 34: Transfer modes supported by the DMA	80
Table 35: Burst FLASH Interface Signals	81
Table 36: Supported Internal Address-wrapping Burst Transfers for Burst FLASH	84
Table 37: Burst FLASH Pinout Multiplexing	85
Table 38: Cycle accesses to the SRAM places on RBUS (with a 75MHz MCU)	86
Table 39: SBUS Peripheral Set and Registers	89
Table 40 Clock States	91
Table 41: System Clock Control Register	93
Table 42: DSP Control Register	95
Table 43: Multiplication ratios for the System PLL (MSEL[5]=0)	97

CONFIDENTIAL

1. Introduction

1.1 Overview

This document specifies the features and functions of the Monaco baseband processor. This preliminary specification is targeted to the chip design and design verification team, as well as system architects and software engineers using and programming the device.

The Monaco product is a derivative of the Hercules/Hermes baseband processors, with the 218x-based DSP replaced with the more powerful FRIO DSP. In addition to this, interfaces have been changed/added to target the market needs of the product release. In addition to the baseline 2G GSM modem application, Monaco is targeted to support GPRS and EDGE class12-capable phones, and to provide a development platform for 3G phones, through the addition of off-chip acceleration hardware.

This document describes mainly the feature deltas relative to the Hercules AD6522 digital baseband, and the impact these features have on the system. Therefore some existing documents are used as references; for a thorough description of the system the reader is referred to [3].

The list of features includes:

Complete Single Chip Programmable Digital Baseband Processor divided into three main subsystems:

Control Processor Subsystem including:

- 32-bit MCU ARM7TDMI® Control Processor
- 75 MHz operation at 1.2V
- 4-Mbit On-chip System SRAM

DSP Subsystem including:

- New Frio Fixed Point DSP/ Microcontroller Processor
- 300 MHz (600 MMAC) at 1.2V
- Separate L1 code and data memories, each with dedicated SRAM and cache
- 1 Mbit L2 SRAM
- EDGE-capable Viterbi and Ciphering co-processors

Peripheral Subsystem including Shared Peripheral Bus and Interface Peripherals

Peripheral Functions:

- Flexible memory interface supporting most standard memory types, including
 - Normal, Page-Mode, and Burst-Mode FLASH
 - SRAM
 - Stacked memory (FLASH and SRAM in same package)
- Serial Display Interface
- Keypad Interface
- 64 kbps SIM Interface
- Universal System Connector Interface
- Multiple serial port standards through Enhanced GSP (Generic Serial Port)

- Baseband Converter Interface
- Data Services Interface (e.g. UART)
- Synthesizer Interface
- Battery Interface (e.g. Dallas)
- SPI
- I2C
- I2S
- Low and Medium Speed IrDA
- USB Interface
- MMC/SD Card Interface
- Real Time Clock with Alarm
- Three independent programmable backlight outputs
- uSM Timebase and Event Processor
- Watchdog and General-Purpose Timers
- Chip ID Register

Enables off-chip ADI Bluetooth Baseband / Radio

Control of Phone Radio Subsystem

Highly configurable interrupt controller architecture

Programmable Bus arbitration for optimizing system performance

DSP and System DMA controllers

Programmable Power Management and Clock Management

Slow Clocking Scheme for Low Idle Mode Current

Power Down modes

Dynamic Voltage Scaling

External Application Processor Interface

External 3G Coprocessor Interface (Extended Bondout Option)

On-chip support for GSM Data Services up to 14.4Kbits/sec, GPRS, HSCSD, and EDGE

JTAG Interface for Test and In-Circuit Emulation of both the MCU and DSP

1.2V Typical Operating Voltage

Core Operating Voltage Range 0.7 - 1.4V

Independent I/O Voltage

Highly flexible pin assignment

APPLICATIONS

GSM 900 / DCS1800 / PCS1900 Wireless Terminals

GSM Phase 2+ Compliant

GPRS class12

EDGE class 12

TDSCDMA

3G (w/ external ADI coprocessor)

ARM® and ARM7TDMI® are registered trademarks of ARM Limited

1.2 Purpose

This document details the overall architecture, the module definitions, and system interfaces, for the Monaco Baseband Processor. This document is the master reference document for all hardware design issues, and user guide implementation. The aim of the document is to give the reader a full understanding of the hardware components, their interactions and programming features. It also provides details of the external component support, and methods for system optimization.

This document contains company Proprietary information, and must not be distributed or copied without the explicit authorization of Analog Devices Inc.

1.3 Scope

This document lists the functional requirements of each part of the design (module and system functions, and external interfaces) of the first TBD-family digital baseband device, herein defined as “Monaco”, and defines the requirements to support software development of the TBD chipset family. It includes full register descriptions for each module, though it does not cover all aspects of the programmers model for use of the device.

This document is structured very similar to that of the AD6522 Digital Baseband Processor Design Specification. Deltas from that document are described in this specification. Wherever the statement “No change to this section” appears in this document, the reader should refer to the Hercules document for content [3].

This document version supersedes all information previously published on Monaco.

1.4 Terminology

ABB	Audio BaseBand device
DBB	Digital BaseBand device
EDGE	Enhanced Data Rate GSM Evolution
Double-word	A 64-bit data element
Word	A 32-bit data element
Half-word	A 16-bit data element
Byte	An 8-bit data element
USB	Universal Serial Bus
MMR	Memory Mapped Register
MMC	MultiMedia Card
MMI	Man-Machine Interface
USC	Universal System Connector
TEP	Timing and Event Processor. The Monaco system-timing module.
QBit	Referring to a quarter of a bit period in GSM.

1.5 References

- [1] Universal System Connector, July10, 1998

- [2] Universal Serial Bus Specification, Rev. 1.1
- [3] AD6522 Digital Baseband Processor Design Specification, Rev. 1.7a. May 2, 2000
- [4] Hermes DSP Memory Mapping, Rev. 0.0
- [5] Hercules External Bus System Specification. Rev. 1.2. Oct. 6, 1999
- [6] Hercules RBUS Specification. Rev. 1.1. Oct. 11, 1999
- [7] 1.8V Intel Wireless Flash Memory (W18), Preliminary datasheet, Oct. 2000
- [8] Hermes Preliminary Specification. Rev. 0.6. Jan. 12, 2001
- [9] Edinburgh WPRD Rev. 1.2. May 10, 2001
- [10] Frio Core Architecture Specification (EAS), Rev. 1.2
- [11] Frio Core Micro-architecture Specification (MAS), Rev. 0.2
- [12] Monaco TEP Specification, Rev. 0.8, July 12, 2001
- [13] Typhoon Product Definition, Rev 1.0, Oct. 2, 2000
- [14] Multi System Mechanism. Rev. 0.7. Feb. 15, 2001.

1.6 Acknowledgments

The ARM7TDMI core is a product licensed from ARM Ltd.

The Frio DSP core is a product designed by Analog Devices Inc.

2. General Description

2.1 Overview

The digital baseband architecture is designed to support two processing cores: the ARM7TDMI microcontroller from ARM Ltd. (hereafter referred to as the MCU), and the Analog Devices Inc. Frio DSP processor (referred to as the DSP). While each has its own control functions (e.g. coprocessing units), they both have shared access to all other peripherals and memory sub-systems.

Unlike conventional DSP architectures, the Frio DSP architecture supports a flexible microcontroller instruction set, including a byte-addressed memory map. This feature, along with a common memory map between the DSP and the MCU (excluding control registers local to each processor), and a flexible interrupt controller which allows any system interrupt to be mapped to either processor, greatly simplifies the programming model for a heterogeneous multiprocessor system. Software tasks can easily migrate to either processor, to simplify load balancing as the software evolves over time.

All system resources in Monaco are memory mapped. The flat, 32-bit byte-address memory map of Monaco is comprised of four main subsystems, as shown in the following diagram:

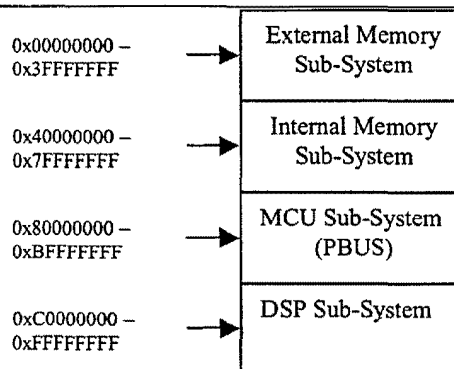


Figure 1: Monaco Shared Memory Map

The two processor systems have their own dedicated DMA controllers, interrupt controllers, JTAG controllers, and local control/status registers.

Monaco is manufactured on a 0.13 μ m process.

Monaco has the following physical specifications:

0.13 μ m Process	Min	Max
Internal Device Supply Voltage (V_{DD})	0.7v	1.5v
System PLL Supply Voltage (V_{DDPLL})	TBD	TBD
Peripheral PLL Supply Voltage (V_{DDPLL2})	TBD	TBD
Real Time Clock Supply Voltage (V_{DDRTC})	2.25v	3.30v
SIM Supply Voltage (V_{SIM})	TBD	TBD
Memory Supply Voltage (V_{MEM})	TBD	TBD
IO Supply Voltage (V_{DDIO})	2.25v	3.30v
MMC Supply Voltage (V_{DDMMC})	1.8v	3.30v
AD6521 Interface Supply Voltage (V_{PEG})	2.25v	3.30v
Core Supply Bias Voltage	TBD	TBD
Core Substrate Bias Voltage	TBD	TBD
Ambient Operating Temperature	-40°C min	+85°C max
MonacoACA Package (Production)	TBD-Ball mini-BGA	
Monaco-PGA Package (Development)	TBD, but target is a 256-pin PGA	

Table 1: PRELIMINARY Operating Conditions

The internal logic, analog PLL, RTC, and the periphery I/O can each be driven off separate power supplies, allowing for simple I/O interfacing while minimizing core power consumption. Level-shifting logic at each of these interfaces ensures valid communication between each power domain, and prevents excess current flow between the domains. The level-shifting logic also minimizes current flow between the domains when voltage to one of the power domains is disabled.

As the internal logic voltage is increased, the control software is able to increase the input clock frequencies to the MCU and DSP as follows:

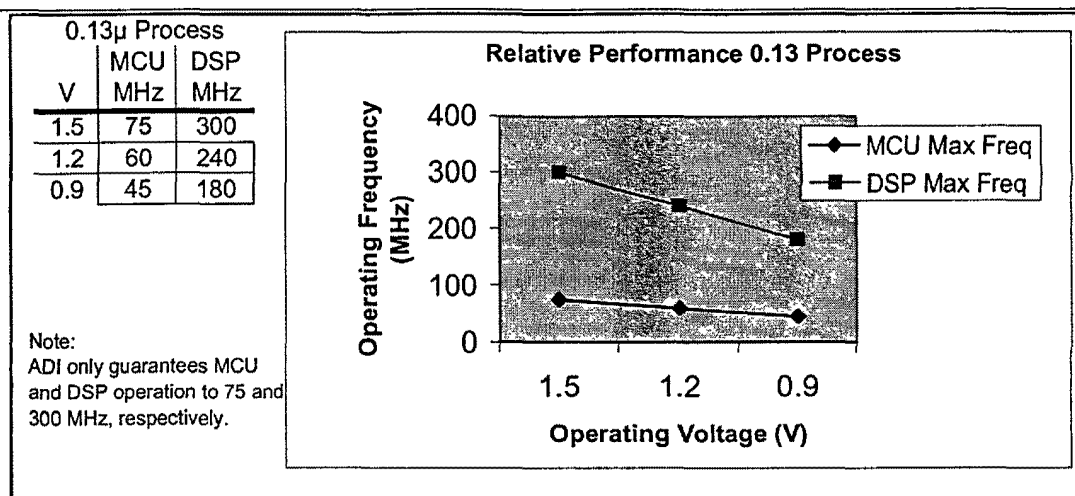


Figure 2: PRELIMINARY Maximum Operating Frequencies

These frequencies are produced by a single analog PLL, which multiplies the input clock frequency to the specified DSP clock frequency, and other internal synchronous clock frequencies.

The Monaco device interfaces to a mixed signal Audio Baseband Codec (ABB), that performs the ADC/DAC and filtering for the speech channel (microphone, speaker), as well as the modulation and demodulation for the up- and downlinks of the communications channel. An auxiliary interface supports additional ADC for functions such as battery and temperature monitoring, and the power management control communication.

The Monaco is optimized to interface to the ADI Typhoon (multi-mode) ABB device [13]. For the other system components, additional flexibility is provided in the signaling to support a wider range of components, including large memory systems and accessory devices.

The production bond-out package for Monaco is a TBD-pin BGA package. Monaco will likely preserve most of the Hercules/Hermes pinout; however, the voltage management chip interface, USB, MMC, and the different operational voltages dictate that Monaco cannot be pin compatible w/ the previous generation baseband products (at least in one or more configurations).

A development bond-out package will be available to support more extensive debug and emulation facilities, but significant support is maintained in the production package via the JTAG port and the Universal System Connector. This includes hardware signal monitoring, high throughput data and instruction trace logging, and serial port (Voice data, I/Q data, Messaging) access, as well as emulation of the MCU and DSP processing cores. The development bond-out package also includes interfaces for off-chip accelerators, and for external mastering of the internal baseband resources.

The JTAG port available in the production bond-out package is fully IEEE compatible, and can be used to support full PCB testing (boundary scan). Note that the boundary scan function can be disabled, to prevent unauthorized reprogramming of the FLASH memory via this interface. This port also supports debug and emulation control of both the MCU and DSP cores.

Software partitioning is achieved by running all of the Protocol Stack Layer 1 code on the DSP. The Protocol Stack layers 2/3, and the Application layer, run on the MCU (although some/all of this code will migrate to the DSP eventually). The Application layer is supported by a real-time kernel. Hardware resources are shared to aid inter-processor communications and optimum functional partitioning. Interrupts can be assigned to either processor, to facilitate code migration from the MCU, to the DSP.

2.2 External Interfaces

The following diagram provides an overview of the production bond-out external interfaces of Monaco, as well as an overview of the modules comprising a Monaco-based terminal device.

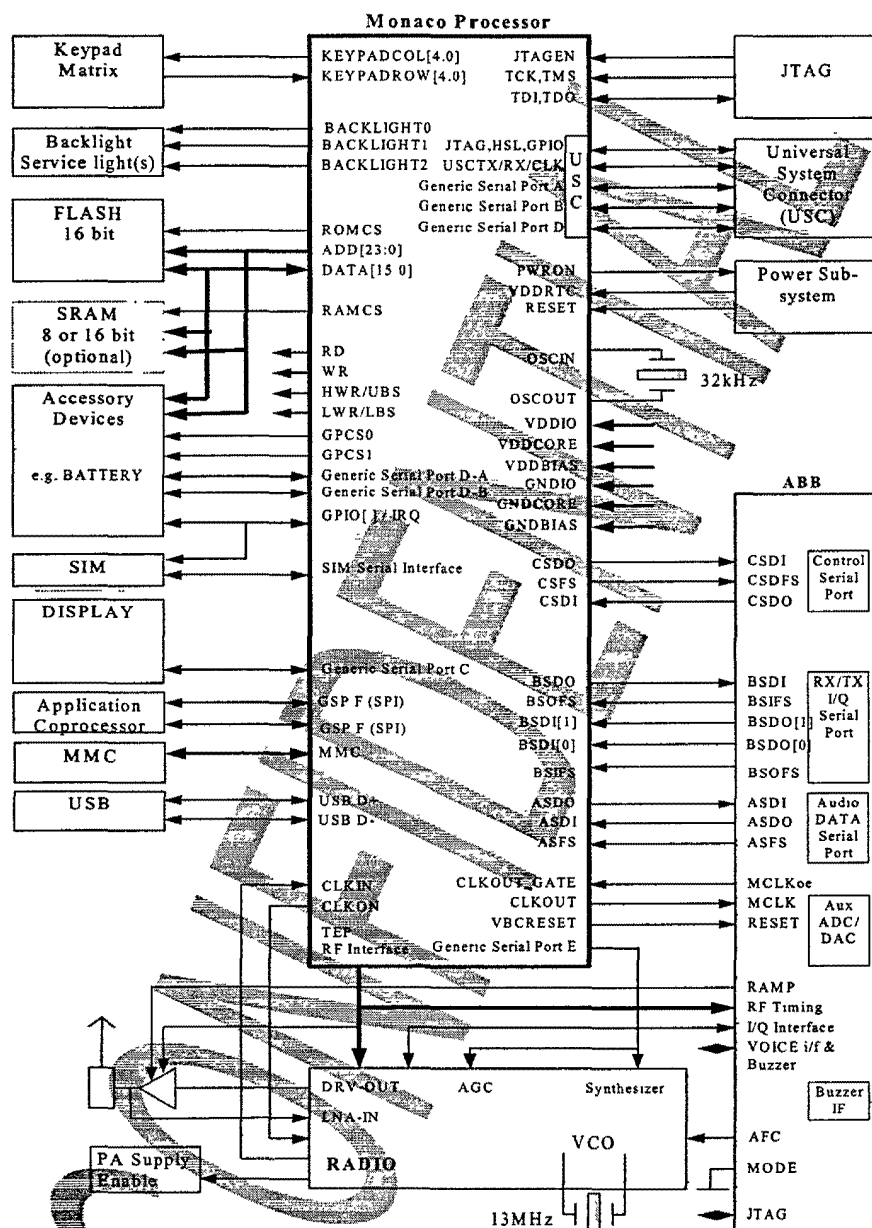


Figure 3: External Interfaces

Opens:

- This diagram shows the dual-SPI application coprocessor interface (EAPI). A parallel port would require more pins, special production package overhead
- Do we need to increase the keyboard matrix size?? We currently support up to 25 keys.
- Communication interface to Typhoon+ pwr mgmnt controller is TBD

Each of the device interfaces is specified in detail in later chapters. A number of pin functions are provided as 'either/or', and must be s/w selected before use. Default pin states are given in chapter TBD.

2.3 Internal Architecture

The following block diagram shows the proposed Monaco system organization. This diagram also shows the obvious lineage from the Hercules baseband architecture.

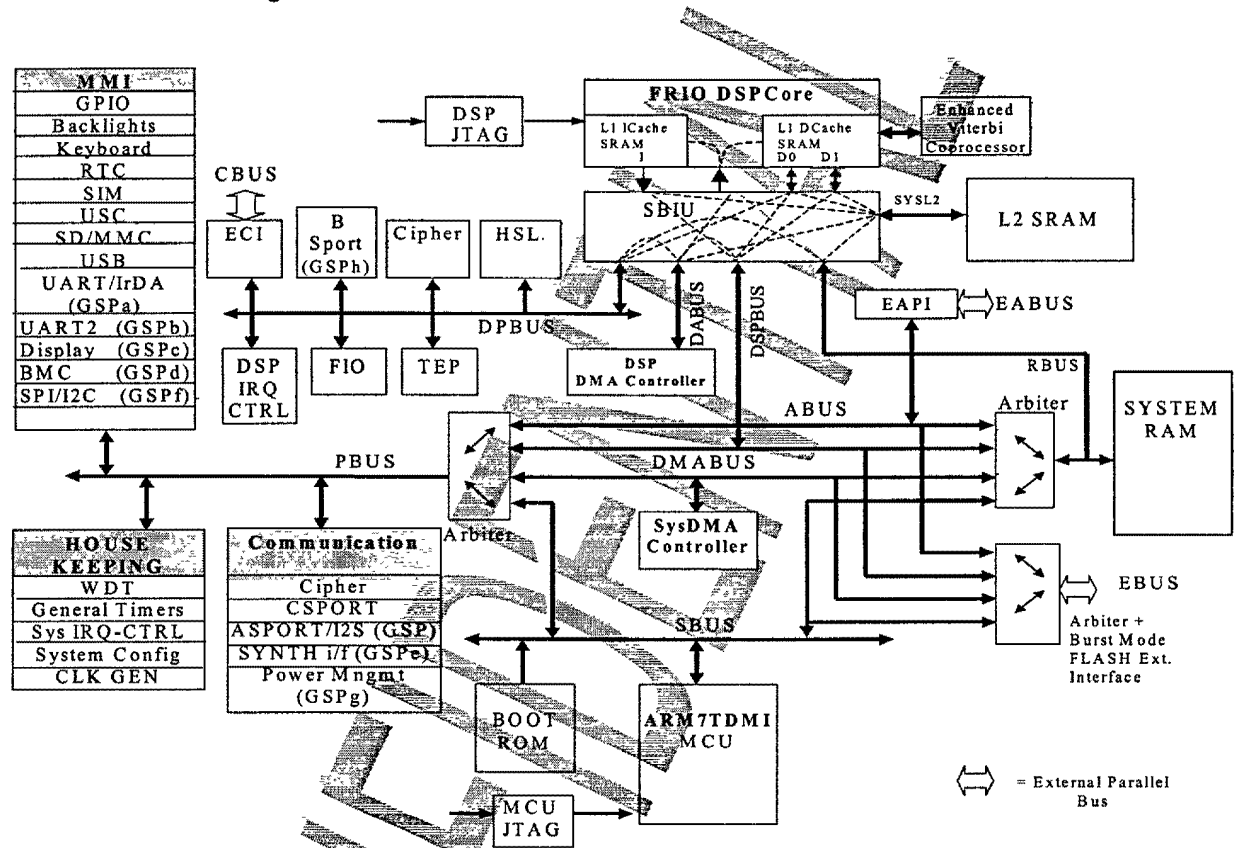


Figure 4 Monaco Block Diagram

The system is served by 12¹ main bus systems:

- SYSL2 The interface to the L2 SRAM, shared between the MCU, the System DMA, the DSP DMA, and the DSP
- DPBUS The DSP peripheral bus, shared between the MCU, the System DMA, the DSP DMA, and the DSP
- DSPBUS The DSP interface to the PBUS, System RAM, and EBUS resources
- DABUS The DSP DMA controller interface to the SBIU
- DMABUS The System DMA controller interface to PBUS, RBUS, and EBUS resources

¹ The Frio DSP also has three dedicated high-speed busses to support DSP access to the system, and one dedicated interface for system access to L1 memory

- RBUS The interface to on-chip System RAM, shared between the MCU, the System DMA, the DSP DMA, and the DSP; this interface also provides MCU and System DMA access to the DSP subsystem
- EBUS The parallel interface to off-chip FLASH, SRAM memory
- SBUS The main MCU System Bus
- ABUS The interface from the external applications processor bridge, to internal resources
- EABUS The interface to the off-chip External Applications Processor
- CBUS The parallel interface to an optional off-chip External Coprocessor
- PBUS The peripheral bus, shared between the MCU, the System DMA, the DSP DMA, and the DSP

This bussing organization provides a high degree of concurrency, to support the real-time requirements of this multi-master system. For example DSP, MCU, System DMA, DSP DMA, and external application processor transfers can often occur in parallel, with no conflicts or collisions.

2.4 Internal Modules

The bus systems connect together four main components of the Monaco architecture:

- DSP Subsystem Includes the Frio DSP, SBIU, L1 and L2 Memories, DSP peripherals, DSP DMA controller, and the Cipher, Viterbi, and ECPI Co-Processing Units
- MCU Subsystem Includes the ARM MCU, System DMA, BootCode ROM, and the System Peripherals
- System Memory Contains on-chip fast static RAM; this memory can store code and data for either processor
- External Devices FLASH, and optional external SRAM and/or other devices.

The subsystems comprising this block diagram are discussed in the following chapters.

2.5 System Clocking Overview

The Monaco system has one primary clock input (CLKIN), from which all internal timing is derived, except for the internal logic which operates on asynchronous clock domains (e.g., RTC, USB, serial ports). This clock can be in the 13-26MHz range.

The internal clock organization of Monaco is shown in the following diagram.

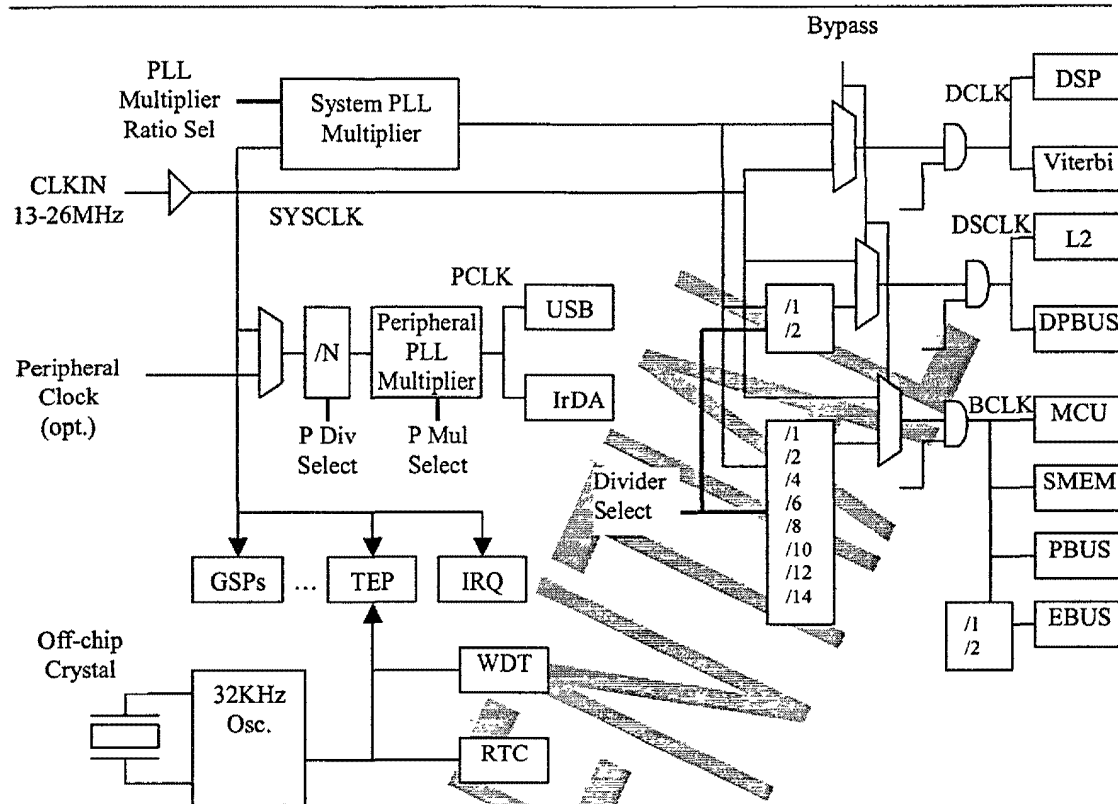


Figure 5 Monaco System Clock

The CLKIN input is a low-voltage swing, sine wave clock input, to conserve power. The CLKIN receiver generates a full-amplitude square wave from the input clock. This signal is routed to the analog PLL, as well as the PLL bypass mux, the timebase and event processor (TEP), and the interrupt controller (IRQ).

The on-chip analog PLL, when enabled, multiplies the primary clock input frequency by the ratio specified on the MSEL multiplier ratio. The PLL can also be bypassed, by setting the appropriate control register bit. When the Bypass bit is asserted, the divide functions are bypassed, thus $DCLK = DSCLK = BCLK = SYSCLK = CLKIN$ frequency.

The PLL provides a clock out frequency range of 30MHz – 300MHz. The PLL provides the minimum clock out frequency required by the DSP to operate at the minimum internal logic supply voltage (e.g., 0.9v).

2.5.1 Clock domains

The Monaco system has the following clock domains, as seen in Figure 5 Monaco System Clock:

- DCLK: DSP Clock (DSP and Viterbi Coprocessor). Divide by 1 from PLL/CLKIN. Range: CLKIN to 300MHz.
- DSCLK: DSP System Clock (SBIU, DPBUS, L2 Memory, DSP DMA). Divide by 1 or 2 from PLL/CLKIN. Range: CLKIN/2 to 150MHz (/1 cannot be used if DCLK > 150 MHz).
- BCLK: MCU Clock (MCU, PBUS, RBUS, EBUS, SBUS, System RAM, System DMA). Divide by 1, 2, 3, 4, 5, 6, or 7 from DSCLK (edge aligned to DCLK). Range CLKIN/7 to 75MHz.

- **SYSCLK:** CLKIN Clock (ASPORT, BSPORT, CSPORT, TEP, IRQ – all logic which needs a fixed clock frequency reference > 32KHz). CLKIN frequency (13-26MHz)
- **Slow Clock:** (RTC, TEP modules). 32KHz.
- **PCLK:** Peripheral Clock (USB, IrDA modules). Programmable divider: SYSCLK /N, multiply resulting 1-2MHz clock to required IrDA or USB peripheral frequency, depending on which peripheral is enabled.
- **ECLK External Bus Clock (EBUS).** Divided by 1 or 2 from BCLK. Range CLKIN/14 to 75MHz.

The clock trees for each of the major clock domains can be disabled in the different low-power states, to conserve power otherwise lost in charging/discharging the clock tree even though the modules driven by that clock tree are idled. Additional clock gating at the module level further provides a finer granularity power conservation feature.

The DCLK, DSCLK, and BCLK frequencies can be changed dynamically. This will often be done in conjunction w/ dynamically varying the internal logic voltage levels, to optimize power consumption (see Dynamic Voltage Control). The following diagram provides an example of how the various clock frequencies might vary over time. DSCLK is omitted from the diagram for clarity. Note that the BCLK divider ratio can change, to keep BCLK near a desired frequency target.

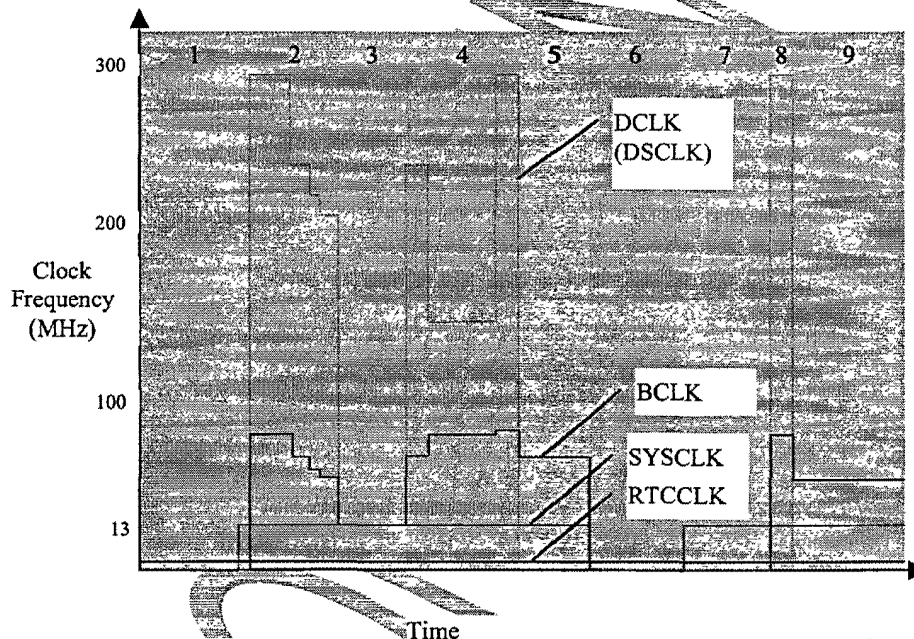


Figure 6: Example of Clock Frequency Variation Over Time

The above diagram describes several different operational states, denoted by the different color bars, and numbered 1-9. A brief description of each of these states, from a system clocking perspective, follows:

1. The baseband is the lowest energy state (slow clocking mode). The only enabled clock in this mode is the 32KHz RTCCLK. This state is exited by the TEP control module, which enables CLKIN generation, then enables the CLKIN receiver, then, after a fixed settling time, enables SYSCLK. Finally, the TEP programs the PLL multiplier ratio, and enables the PLL, disabling the PLL bypass.

2. This state ramps to the highest possible performance/energy level, with both the DSP (DCLK) and MCU (BCLK) cores running at maximum clock rate possible. Note that the cores wakeup and begin interrupt processing *before* the system PLL has acquired phase lock. Over time, less performance is required; therefore, the dynamic voltage control mechanism steps the clock rate down, allowing the supply voltage to drop.
3. In this state, the PLL has been bypassed, such that DSCLK = BCLK = SYSCLK frequency. The DSP is in IDLE state, and DCLK is gated off.
4. Equivalent to state 2, but at lower energy levels. Note that the BCLK divider changes from /2 to /1 at the first drop in DCLK. This is possible because the PLL clock dropped to a level low enough to allow BCLK to run at DSCLK frequency, and not exceed the maximum supported BCLK frequency.
5. The PLL is enabled; however, the DSP is idled, and DCLK is gated off.
6. Identical to state 1.
7. PLL bypassed state, identical to state 3. Note that here, the phone could be in standby mode. TEP, B SPort, DSP DMA, and L2 memory would be the only enabled modules besides RTC in this mode.
8. Equivalent to state 2, however, clock frequency is not stepped down here.
9. Identical to state 5.

See 11.1 System Clock Control for more information on System Clocking, PLL, and control.

3. ARM7TDMI CORE CONFIGURATION

3.1 Changes from Hercules

The only changes from the MCU subsystem of Hercules are:

1. Clock frequency extended to $\geq 75\text{MHz}$, 0.13 micron process.
2. Tracking emulation capability has been replaced by the ETM Embedded Trace Macrocell feature.
3. The SBUS CLK GEN and ARBITER control registers have been moved to the PBUS (MEMCTR, DBGEN, and BootROM Test Mode Access registers remain on the SBUS). The MCU Clock control and System Clock and DSP PLL control registers change definition per the Monaco single system PLL clock architecture. By moving these registers to the PBUS, the DSP can also access these control parameters. These registers, plus new ones to control/configure Frio, are in the Housekeeping peripheral group, described as the System Configuration Block. This block includes registers to control the following:
 - Clock Control
 - Reset vector input to Frio
 - SRAM_INIT, RESET inputs to Frio
 - L1 config, L1 enable inputs to Frio
 - SRAM_ADDRESS input to Frio
 - Arbiter controls
 - Power Management

3.2 Component Parts

The MCU core used in Monaco is the ARM7TDMI device, from Advanced RISC Machines LTD, which includes:

- Thumb Code Execution capability
- Debug Interface
- Multiplier (32x8)
- ICE via Embedded ICE macrocell
- ETM – Embedded Trace Macrocell, in the Medium Configuration

The ARM7 core is based on a RISC 32bit architecture, with a three stage pipeline (fetch, decode, execute) instruction cycle. It interfaces via the SBUS (also 32bit) to the Monaco system, where the arbitration units perform any 16 to 32 bit conversions.

The ARM7TDMI bus interface does not include the AMBA bus system, instead all signaling is defined directly by the ARM7 core signals.

The ARM7TDMI does not include the ARM TIC (Test Interface Controller), instead all manufacturing and functional testing must be done via JTAG and software.

The core includes the EmbeddedICE macrocell (ICEBREAKER), which provides 2 hardware breakpoints (set on address, data and/or control status), that can be augmented by external (e.g. DSP initiated) debug requests via the DBGRQ and BREAKPOINT inputs.

The core includes the ETM debug and trace facility. This module provides a dedicated 13-pin debug port in the extended bondout package.

4. DSP Sub-system

The DSP subsystem, shown in Figure 7 DSP Subsystem Block Diagram, consists of the Analog Device's Frio DSP core, enhanced with a system interrupt controller, a Viterbi coprocessor, a Ciphering unit, and a cache/RAM memory hierarchy. System timing accuracy and sequencing are provided by the Timing Event Processor (TEP). This subsystem also includes a dedicated DMA controller, serial port controller, high-speed data logging unit (HSL), Programmable Flag IO, and an interface to an external coprocessor. This external coprocessor interface (ECI) can support, for example, off-chip RAKE receiver acceleration for 3G/WCDMA applications.

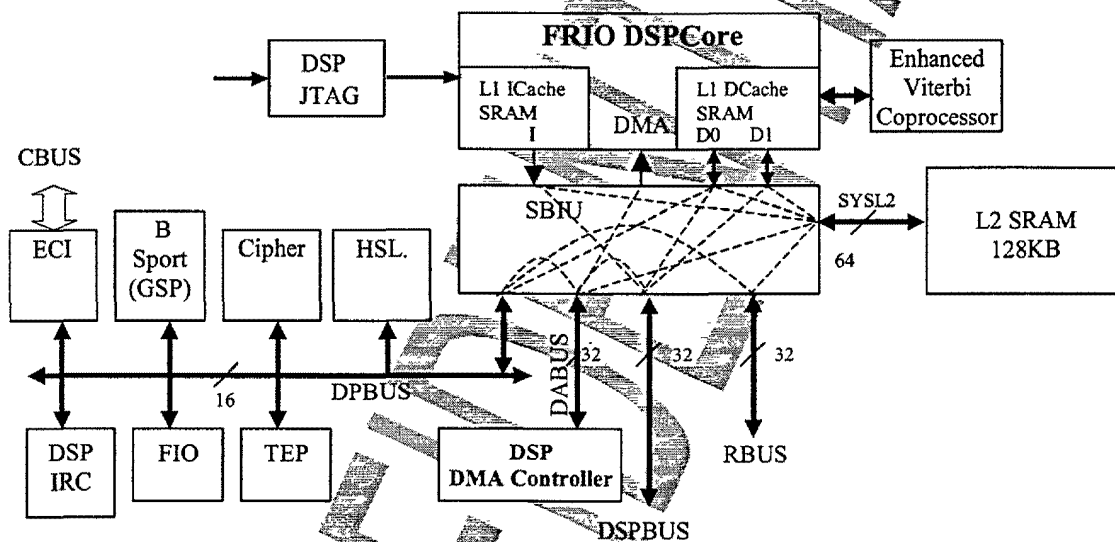


Figure 7 DSP Subsystem Block Diagram

4.1 Frio DSP

The Frio is a digital signal processor (DSP) with added functionality to realize many of the programming advancements normally associated with microcontrollers.

The Frio core is a dual datapath, modified Harvard architecture processor that is optimized for both DSP and micro-controller functions. The DSP features of the architecture include one instruction and two separate data ports to a unified 4GB memory space, two 16-bit single-cycle throughput MACs, two 40-bit split ALUs, four auxiliary 8-bit ALUs, two DAGs with circular and bit-reversed addressing, two loop counters that allow nested zero overhead looping, and hardware support for on-the-fly saturation and clipping. The micro-controller features include arbitrary bit manipulation, mixed 16 and 32 bit instruction encoding for code density, memory protection, stack pointers and scratch SRAM for context switching, flexible power management, and a nested and prioritized interrupt controller for real-time control. Execution time predictability is achieved with lockable caches that can optionally be configured as SRAM, and data-independent instruction execution.[10]

The main components of the Frio processor are

- **Data Arithmetic Unit (DAU):** The Data Arithmetic Unit is the computational heart of the Frio DSP processor and consists of two Data Execution Units and a Data Register File. Data Execution Unit 0 has a 16-bit x 16-bit multiplier, a 40-bit split ALU, two auxiliary 8-bit ALUs, a 40-bit split accumulator, and a 40-bit shifter. Data Execution Unit 1 has no shifter but is otherwise the same. The ALU and accumulators are split. The two Data Execution Units can accumulate two 40-bit numbers or four 16-bit numbers. The Data Execution Units perform all data manipulation instructions including arithmetic, logic, comparison, shifting and DSP instructions. The operands for the Data Execution Units are stored in an 8-entry 32-bit Data Register file.
- **Address Arithmetic Unit (AAU):** This unit consists of two Data Address Generators (DAGs), which are the logic required to generate and update all addressing for the core. In addition, there are two register files: the Pointer Register File, and the DAG Register File.
- The dual DAGs are complex load/store units which are designed to generate addresses to support sophisticated DSP filtering operations. In particular, bit-reversed addressing and circular buffering are supported for DSP addressing, while auto increment, auto decrement and base plus immediate offset addressing modes are supported for the micro-controller-style loads and stores. The DAGs are capable of generating two new addresses and two address register file updates every clock cycle, and use the DAG Register Rename Unit to allow precise interrupts to occur.
- The Pointer Register File contains pointers (addresses) used by the Frio processor to fetch operands as well as manage stack and subroutine call/return operations. All addresses are 32-bits wide. There are 6 registers in the Pointer Register File (P0-P5), a Frame Pointer FP for subroutine calls, a user Stack Pointer USP, and a Kernel Stack Pointer KSP (both user and kernel stack pointers are referenced as SP).
- The DAG Register File contains addresses and offsets used to generate addresses for DSP operands. All addresses and offsets are 32-bits wide. There are 4 sets of DSP registers: I0-I3 (the index registers), M0-M3 (the modifier registers), B0-B3 (the base registers) and L1-L3 (the length registers). Although a given base and length always correspond to a given index register (i.e. B2 and L2 always correspond to I2), different modifier registers can be used with different index registers.
- **Control Unit:** The control unit consists of the Instruction Alignment and Decoding Unit and the Sequencer and Branch logic. Instructions fetched from the Instruction cache are aligned and decoded in the Instruction Decoder. The decoder allows alignment on 2-byte boundaries in memory of 16-bit instructions, 32-bit instructions and 64-bit instruction lines. The sequencer manages the control of the datapaths in the Data Arithmetic Unit and the Address Arithmetic Unit. It also controls stalling of the instruction flow through the pipeline. The sequencer insures that the pipeline is fully interlocked and that all data hazards are hidden from the programmer. The Branch Logic includes the branch offset adder, the sequential address adder, and the zero-overhead loop logic.

4.1.1 L1 Memory

The Frio L1 memory system consists of separate instruction and data memories, where each of these memory sub-systems has both cache and fixed SRAM. The Level 1 memory supports up to three core accesses each core clock cycle. This memory is integrated into the DSP core, due to the number of ports required to service the core, as well as the high frequencies of this memory system. Only the DSP can access the L1 cache memory. The L1 SRAM can be accessed by both the DSP and the DSP DMA controller. See the DSP Memory Organization section, below, for more information.

4.1.2 Endianess

Frio is a little-Endian architecture.

4.1.3 Frio Interrupt Support

The Frio DSP has an internal interrupt controller. This controller works with the DSP Interrupt Controller of the Monaco system to provide the integrated DSP interrupt processing system. The following diagram provides an overview of this system:

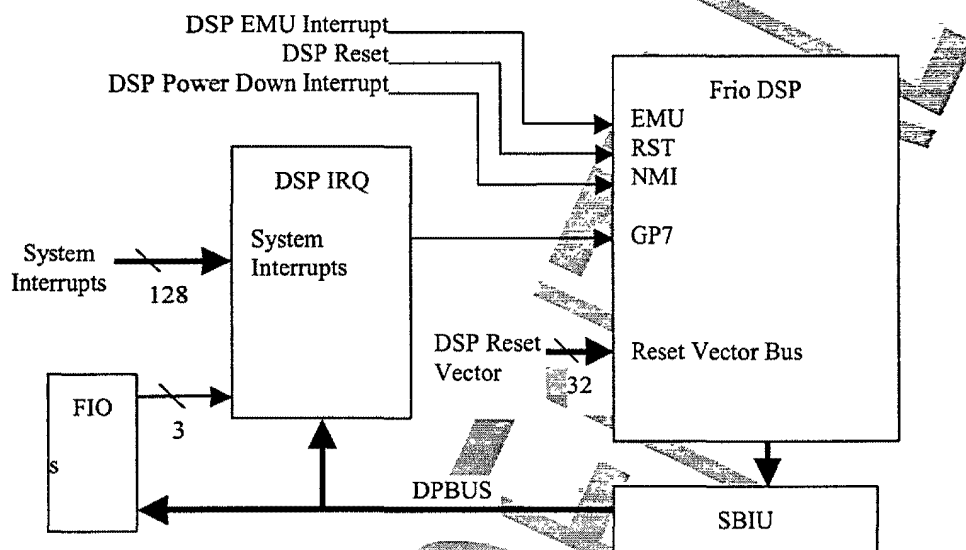


Figure 8 Interrupt Control Block Diagram

The Monaco DSP Interrupt controller prioritizes and groups system level interrupts, including the software interrupts (SWIs) generated by the TEP, or by appropriate register writes from either the DSP or the MCU. Once asserted, all interrupt sources, except for the TEP SWIs, are held asserted until the source of the interrupt has been serviced by the supporting processor's interrupt service routine (i.e., all system interrupts are level, as opposed to edge, assertions). More information on the system interrupt controller and programming model can be found in Ch. 7.5, Interrupt Controller, on page 62.

As can be seen by the above figure, the single DSP interrupt output of the Monaco DSP interrupt controller maps to the highest-priority general purpose interrupt (GP7) on the Frio DSP. The general purpose interrupt inputs of the Frio core are edge-sensitive inputs; therefore, the system-level DSP interrupt controller must ensure that an edge is generated for any pending interrupt event. This is accomplished by ensuring a single cycle low pulse on the interrupt output whenever the CI register of the system-level DSP interrupt controller is read, via the DPBUS.

In addition to the interrupt interface to the system-level DSP interrupt controller, the Frio core also accepts the DSP emulator (EMU), reset, and NMI interrupt inputs directly. The emulator interrupt input is the highest priority interrupt input to the Frio core. Assertion of this input will cause the Frio core to enter emulation state.

Reset is treated as an interrupt request to the Frio DSP. Reset is lower priority than the emulator interrupt. Assertion of this input, assuming the emulator interrupt is not pending, will cause the DSP to sample the reset interrupt vector presented on the DSP Reset Vector input bus. For Monaco, this bus is driven to starting address of either the L1 code memory space, the L2 memory space, the System Memory space, or the external FLASH memory space. The vector is determined by the MCU at boot

time, and set by writing the DSP reset source field of the DSP Reset Control register in the House Keeping register group on the PBUS.

The non-maskable interrupt (NMI) can only be asserted by the power management controller. This interrupt is used to force the DSP into the IDLE, or low power state. See Chapter 11, System Clock Control and Power Management for more information.

4.1.4 DSP Interrupt Latency

The Frio DSP has the following latencies for external interrupts (latency to the highest priority interrupt), in core clock cycles:

1. 1 cycle to capture (flop) the interrupt at the core boundary
2. 1 cycle for edge detection
3. 1 cycle to capture the event into ILAT register
4. 1 cycle to propagate ILAT to IPEND (now a second interrupt event on the same input can be detected w/o overflow)
5. Fetch of the first instruction of ISR from instruction memory

4.2 DSP Memory

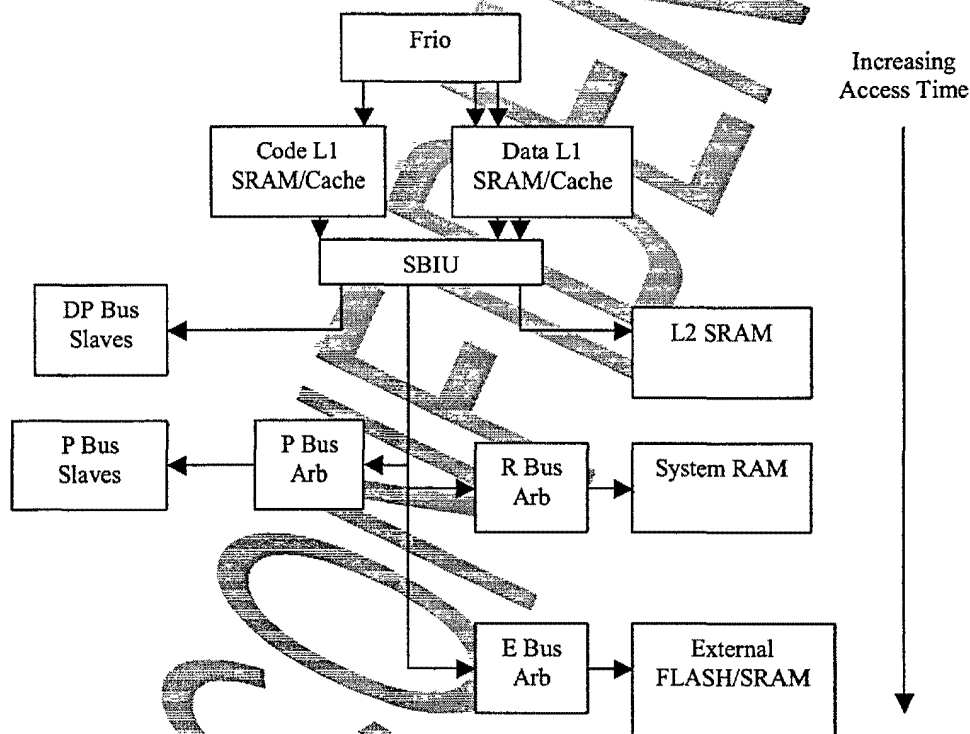


Figure 9 DSP Memory Hierarchy

4.2.1 DSP Memory Organization

4.2.1.1 L1 Memory Organization

The total L1 memory capacity and configuration is [9]:

- 48K bytes of Instruction SRAM
- 16K bytes of Instruction Cache, lockable by way
- 48K bytes of Data cache/SRAM
- 4K bytes of Scratchpad SRAM

Organized and located as :

Data Memory A

- Starting at 0xE000 0000 : Two blocks of 8KBcache / SRAM
- Starting at 0x : 16KB SRAM

Data Memory B

- Starting at 0xE010 0000 : Two blocks of 8KBcache / SRAM
- Starting at 0x : 16KB SRAM

Instruction Memory

- Starting at 0xE020 0000 : 16KB Cache, 4-way set associative
- Starting at 0x : 64KB SRAM

Scratchpad Memory

- Starting at 0xE030 0000 : 4KB SRAM

Each SRAM Bank is made of 4KB mini-banks, which can be accessed by the processor or the DMA engine.

The SRAM base address assignments are determined by by metal programming through manufacturing, verses software programmability by the user.

4.2.1.2 L2 Memory Organization:

- 128KB of simple, fixed SRAM
- Unified code and data storage
- Provides burst access time of 8-2-2-2 core clock cycles at 2:1 DCLK/DSCLK ratio
- Clocked at half the DSP clock frequency if DSP clock > 150MHz. Below 150MHz, L2 can clock at 1:1 ratio.
- No cache control capability
- DMA/MCU access to L2 through same port as DSP (cycle steals from DSP)

4.2.1.3 System Memory Organization:

- Same System RAM as the Hermes project [8]
 - Up to 5M bits of fixed SRAM, depending on customer requirements
 - Current proposal is 4M bits (plus the 1 Mbit of RAM in the L2)
- Unpipelined, clocked at MCU frequency (# 75MHz)

- Single-cycle access by the MCU, System DMA
- #1 priority is density
- Primarily used for code storage for the ARM core

4.2.1.4 Off-chip Memory:

- Burst/Page mode FLASH supported (wrap mode not supported)
- Access times programmable
- External SRAM supported
- Two additional external device chip selects available
- 16-bit data path

4.2.2 DSP Memory Performance

See the SBIU discussion, below.

4.3 System Bus Interface Unit (SBIU)

The SBIU performs the bus bridging functions for the DSP Subsystem. The SBIU functions as an asymmetric crossbar switch, routing requests from the Frio DSP core, the DSP DMA controller, and the microcontroller subsystem (microcontroller and system DMA) and EAPI, to the appropriate slaves (DPBUS peripherals, L1 RAM on the Frio core, L2 RAM, or system resources via the DSP bus. The SBIU provides parallel and concurrent data transfer capability between the various busses.

Figure 10: SBIU Routing Topology, below, diagrams the supported bus-to-bus routes listed below:

1. Frio Core Instruction Memory requests to L2 Memory
2. Frio Core Instruction Memory requests to DSPBUS
3. Frio Core Data Memory Port0 requests to L2 Memory
4. Frio Core Data Memory Port0 requests to DSPBUS
5. Frio Core Data Memory Port0 requests to DPBUS
6. Frio Core Data Memory Port1 requests to L2 Memory
7. Frio Core Data Memory Port1 requests to DSPBUS
8. DSP-DMA requests to Frio L1 Memories
9. DSP-DMA requests to L2 Memory
10. DSP-DMA requests to DPBUS
11. DSP-DMA requests to DSPBUS
12. RBUS requests to L1 Memory
13. RBUS requests to L2 Memory
14. RBUS requests to DPBUS

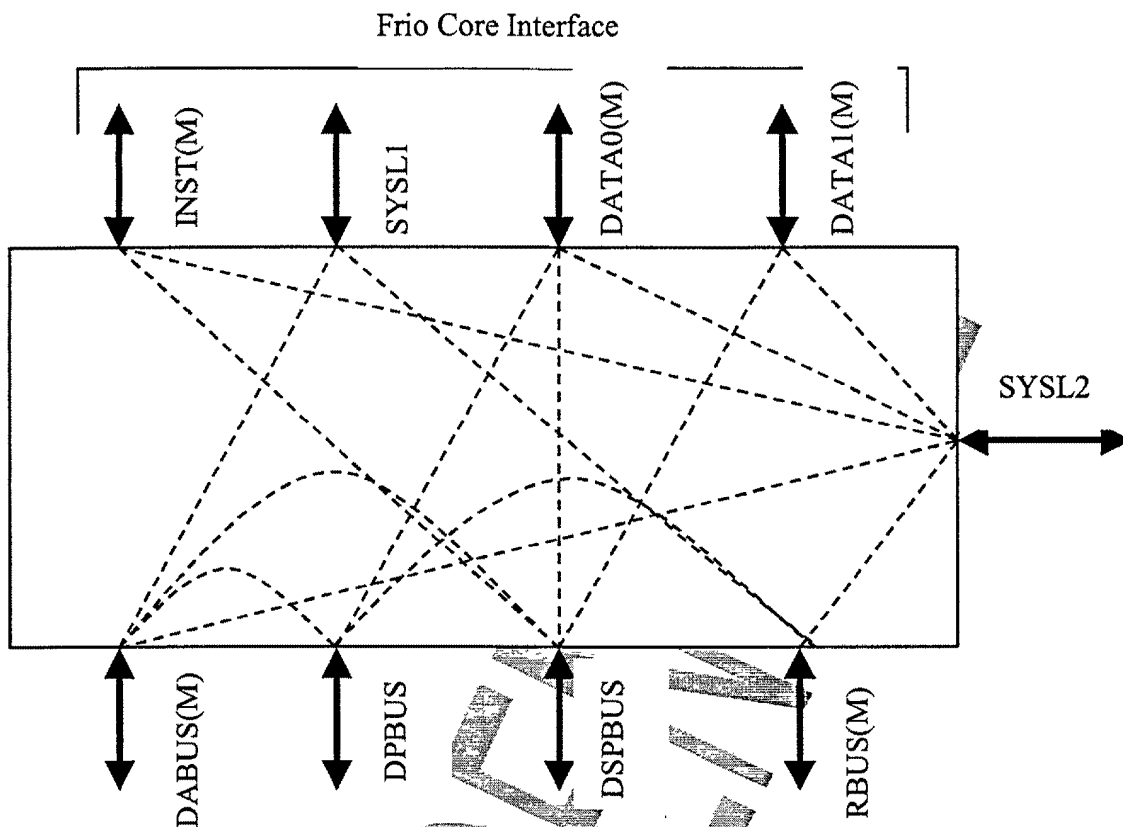


Figure 10: SBIU Routing Topology

In the above diagram, (M) denotes the buses which master operations into the SBIU. The other buses are targets, or slaves, to requests from the SBIU masters. The following table describes the relative priority of each requesting bus:

Table 2 SBIU Internal Routing Priorities

Requestor	Target Bus			
	SysL1	SysL2	DPBUS	DSPBUS
Core D0		1, or 3	1, or 3	1, or 2
Core D1		2, or 4		2, or 3
Core I		3, or 5		3, or 4
DABUS	2	5, or 2	3, or 2	4, or 1
RBUS	1	4, or 1	2, or 1	

The above table indicates a dynamic arbitration scheme for RBUS and DABUS requests into the SBIU. The request generation logic from these buses employ timers, such that they assert high priority requests after a programmable timeout period. See the respective bus sections below, for more details.

The SBIU bridges buses of different data bus widths, as well as protocols (and clocking domains).

- All of the Frio core interfaces to the SBIU employ 64-bit data bus widths, and employ the IBM CoreConnect bus protocols. The instruction fetch port (CoreI) generates 64-bit single and burst

read requests only. The two data ports can generate 8-, 16-, and 32-bit reads and writes, as well as 64-bit wide, 32 byte cache line fill (reads) and copybacks (writes).

- The DABUS, DSPBUS, and RBUS support 8-, 16-, and 32-bit transfers on the 32-bit data bus widths, and employ the ARM7-ish bus protocols.
- The DPBUS supports 16-bit transfers only; it employs the ARM7-ish bus protocols.

The SBIU is responsible for correct data alignment support when bridging between buses. For example, the SBIU must mux the 32-bit RBUS data to the appropriate L2 word lanes. The SBIU also decodes the DSPBUS targets; if accessing the 16-bit wide PBUS or EBUS, the SBIU muxes the data to the lo 16-bit data lane. The SBIU does not perform this data muxing function for accesses to the 32-bit RBUS. However, since the System RAM, which is accessed by the DSP via the RBUS, can be cached, such that the DSP generates 64-bit wide burst line-fill/copyback transfers, the SBIU must correctly mux the 64-bit accesses to the 32-bit RBUS (and further, down to the 16-bit EBUS, which can also be accessed by the L1 cache hardware).

4.3.1 SysL1 Bus

This bus provides system access to the Frio L1 memories. Only L1 code and data SRAM can be accessed through this port – scratchpad SRAM, and L1 memory configured as cache cannot be accessed by the system.

This bus is mastered by the DABUS (DSP DMA controller) and the RBUS (MCU). However, the memories accessed from this bus are shared with the Frio core. In the original Frio core, the SysL1 bus is always higher priority than a pending Frio access; therefore, the access is deterministic (latency is currently 4 core clock cycles for a read). FrioLP core behavior is not known at this time.

The System DMA controller and the EAPI cannot access this bus, nor the L1 memory regions.

4.3.2 SysL2 Bus

This bus provides access to the L2 memory. This bus is mastered by three DSP core ports, the DABUS (DSP DMA), and the RBUS (MCU and System DMA).

The arbitration for this bus has dynamic prioritization, according to the following table:

Table 3 SysL2 Arbitration Priority

Requester	Arbitration Priority	Comments
Hi priority DABUS	1	Hi priority status achieved after timeout
Hi priority RBUS	2	Hi priority status achieved after timeout
Core D0	3	
Core D1	4	
Core I	5	
Lo priority DABUS	6	Initial access priority
Lo priority RBUS	7	Initial access priority

The DSP requesters are highest priority, unless the programmable arbitration latency timers for the DABUS/RBUS timeout, at which point they become the highest priority requester. See the SBIU Arbitration Control for more information.

4.3.3 DP Bus

This bus provides access to the DSP peripherals. All System MMRs are mapped to this bus, except for those of the Viterbi coprocessor. The SBIU is the only master of this bus. The slaves are shown in the Block Diagram, above.

Table 4 DPBUS Arbitration Priority

Requester	Arbitration Priority	Comments
Hi priority DABUS	1	Hi priority status achieved after timeout
Hi priority RBUS	2	Hi priority status achieved after timeout
Core D0	3	
Lo priority DABUS	4	Initial access priority
Lo priority RBUS	5	Initial access priority

The DSP requester is highest priority, unless the programmable arbitration latency timers for the DABUS/RBUS timeout, at which point they become the highest priority requester. See the SBIU Arbitration Control for more information.

4.3.4 DA Bus

This bus provides DSP DMA access to the rest of the system, via the SBIU. The DSP DMA is the only master of this bus; the SBIU is the only slave. Note that the DSP DMA control and status registers are accessed through the DPBUS.

4.3.5 DSP Bus

This bus provides DSP and DSP DMA access to the MCU system and the E-, P-, and R-BUS systems. This bus is mastered by three DSP core ports, and the DABUS (DSP DMA)

The arbitration for this bus has dynamic prioritization, according to the following table:

Table 5 DSPBUS Arbitration Priority

Requester	Arbitration Priority	Comments
Hi priority DABUS	1	Hi priority status achieved after timeout
Core D0	3	
Core D1	4	
Core I	5	
Lo priority DABUS	6	Initial access priority

The DSP requesters are highest priority, unless the programmable arbitration latency timer for the DABUS timeouts, at which point the DABUS becomes the highest priority requester. See the SBIU Arbitration Control for more information.

4.3.6 R Bus

In addition to providing access to the on-chip system memory, the RBUS provides MCU access to the DSP memory and peripheral resources.

Arbitration control of RBUS is identical to that of Hercules, except that the RBSCR0 register has moved to the PBUS.

4.3.7 SBIU Arbitration Control

The SBIU Arbitration Control register determines the arbitration priority of the RBUS and DABUS requesters, relative to each other and the pertinent requesters from the DSP (see the SIBU bus matrix, above). The RBUS and DABUS each have dedicated counters, which determine when the request from the respective source should change from a low priority status, to a high priority status. The register has the following definition:

Register	Add	[7]	[6]	[5]	[4]	[3]	[2]	[1]	[0]	R/W
SBACR	0xTBD	RSRVD				DABUS CNT				r/w
(Reset)		0000				0000				
Register	Add	[15]	[14]	[13]	[12]	[11]	[10]	[9]	[8]	R/W
SBACR	0xTBD	RSRVD				RBUSCNT				r/w
(Reset)		0000				0000				

Table 6 SBIU Arbitration Control Register

Each counter can be programmed to count 0-15 cycles. Programming the counter to zero forces the high-priority state always, for that requester.

4.3.8 SBIU Performance

DSP and DSP DMA access times for the L2 system RAM, off-chip memories, as well as the peripheral busses, follows. Notes on access times:

- All core latencies based on Frio-1 core – needs to be updated w/ FrioLP numbers)
- On all off-core load accesses, the Frio core stalls for at least 1 DCLK cycle. This cycle is consumed in moving the off-core transaction from the L1 memory, to the core boundary.
- For all off-core store accesses, one DCLK cycle is consumed in moving the store from the store buffer to the core boundary (via the write buffer). This cycle does not stall the core, unless the write buffer is full.

4.3.8.1 DSP DPBUS Access Times

The table below assumes 0 wait states in the SBIU arbitration logic and on DPBUS

Frio Core Stall	1 DCLK Cycle
Fast to Slow clk synchronization and Arbitration	1 DCLK to 1 DSCLK (depending on when the core request is asserted relative to the rising edge of DSCLK)
Acknowledge Generation	1 DCLK Cycle
DPBUS access latency	1 DSCLK Cycle

Frio Core to DPBUS Read Latency	(3DCLKs+1DSCLK) to (2DCLKs+2DSCLKs)
--	--

Table 7 DSP DPBUS Read Access Latency

The table below assumes 0 wait states in the SBIU arbitration logic and on DPBUS

Frio Core Stall	0 DCLK Cycle
Fast to Slow clk synchronization and Arbitration	1 DCLK to 1 DSCLK (depending on when the core request is asserted relative to the rising edge of DSCLK)
Acknowledge Generation	1 DCLK Cycle
DPBUS access latency	0 (Hidden)
Frio Core to DPBUS Read Latency	(2DCLKs) to (1DCLKs + 1DSCLK)

Table 8 DSP DPBUS Write Access Latency**4.3.8.2 DSP L2 Access Times**

The table below assumes 0 wait states in the SBIU arbitration logic.

Frio Core Stall	1 DCLK Cycle
Fast to Slow clk synchronization and Arbitration	1 to 2 DCLK Cycle(s) (depending on when the core request is asserted relative to the rising edge of DSCLK)
Acknowledge Generation	1 DCLK Cycle
L2 access latency	4 DCLK Cycles
Frio Core to L2 Read Latency	7 to 8 DCLK Cycles

Table 9 DSP L2 Access Time: Single Read

The table below assumes 0 wait states in the SBIU arbitration logic.

Frio Core Stall	1 DCLK Cycle
Fast to Slow clk synchronization and Arbitration	1 to 2 DCLK Cycle(s) (depending on when the core request is asserted relative to the rising edge of DSCLK)
Acknowledge Generation	1 DCLK Cycle
L2 access latency	4-2-2 DCLK Cycles
Frio Core to L2 Burst Read Latency	7/8-2-2 DCLK Cycles

Table 10 DSP L2 Access Time: Burst Read

The table below assumes 0 wait states in the SBIU arbitration logic.

Frio Core Stall	0 DCLK Cycle
Fast to Slow clk synchronization and Arbitration	1 to 2 DCLK Cycle(s) (depending on when the core request is asserted relative to the rising edge of DSCLK)

Acknowledge Generation	1 DCLK Cycle
L2 access latency	0 (Hidden)
Frio Core to L2 Single Write Latency	2 to 3 DCLK Cycles

Table 11 DSP L2 Access Time: Single Write

The table below assumes 0 wait states in the SBIU arbitration logic.

Frio Core Stall	0 DCLK Cycle
Fast to Slow clk synchronization and Arbitration	1 to 2 nd DCLK Cycle(s) (depending on when the core request is asserted relative to the positive edge of DSCLK)
Acknowledge Generation	1 DCLK Cycle
L2 access latency	0 (Hidden)
Frio Core to L2 Burst Write Latency	2/3-2-2-2 DCLK Cycles

Table 12 DSP L2 Access Time: Burst Write

4.3.8.3 DSP DSPBUS Access Times

All DSP and DSP DMA accesses to system memory (via the RBUS), system peripherals (via the PBUS), and external memory (via the EBUS) use the same DSPBUS port from the SBIU. The SBIU includes logic to decode which bus will be accessed. The following tables attempt to describe the access times all the way to the memory or peripheral target bus.

The table below assumes 0 wait states in the SBIU arbitration logic.

Frio Core Stall	1 DCLK Cycle
Fast to Slow clk synchronization and Arbitration	1 DCLK to 1 BCLK (depending on when the core request is asserted relative to the rising edge of BCLK)
Acknowledge Generation	1 DCLK Cycle
DSPBUS/SysMem access latency	1.5 BCLK
Result Mux, Synchronization to fast clk domain	0.5 BCLK
Frio Core to SysMem Single Read Latency	(3DCLKs + 2BCLKs) to (2DCLKs + 3BCLKs)
Frio Core to SysMem Burst Read Latency	(3DCLKs + 2BCLKs) + 7*2BCLKs = (3DCLKs + 16 BCLKs) to (2DCLKs + 3BCLKs) + 7*2 BCLKs = (2DCLKs + 17BCLKs)

Table 13 DSP System Memory Read Access Time

The table below assumes 0 wait states in the SBIU arbitration logic and RBS arbitration logic.

Also, the DCLK to BCLK ratio is assumed to be 4.

Frio Core Stall	0 DCLK Cycle
-----------------	--------------

Fast to Slow clk synchronization and Arbitration	1 DCLK to 1 BCLK (depending on when the core request is asserted relative to the rising edge of BCLK)
Acknowledge Generation	1 DCLK Cycle
DSPBUS/SysMem access latency	0 (hidden)
Frio Core to SysMem Single Write Latency	(2DCLKs) to (1BCLK)
Frio Core to SysMem Burst Write Latency	(2DCLKs) + 7*2BCLKs = (2DCLKs + 14 BCLKs) to (1BCLK) + 7*2 BCLKs = (15BCLKs)

Table 14 DSP System Memory Write Access

4.3.8.4 DSP PBUS Access Times**4.3.8.5 DSP External Memory Access Times****4.3.8.6 DSP DMA-DPBUS Access Times****4.3.8.7 DSP DMA L1 Access Times**

The table below assumes 0 wait states in the SBIU arbitration logic

SBIU Arbitration	1 DSCLK
L1 memory access latency via DMA port	3 DSCLKs (includes ack generation)
DSP-DMA to L1 Memory Read Latency	4 DSCLKs

Table 15 DSP DMA L1 Read Access

The table below assumes 0 wait states in the SBIU arbitration logic

SBIU Arbitration	1 DSCLK
L1 memory access latency via DMA port	1 DSCLK (includes ack generation)
DSP-DMA to L1 Memory Write Latency	2 DSCLKs

Table 16 DSP DMA L1 Write Access

4.3.8.8 DSP DMA L2 Access Times

The table below assumes 0 wait states in the SBIU arbitration logic

SBIU Arbitration	1 DSCLK
L2 Memory access latency	2 DSCLKs
Acknowledge Generation	1 DSCLK Cycle
DSP-DMA to L2 Memory Read Latency	4 DSCLKs

Table 17 DSP DMA-L2 Read Access

The table below assumes 0 wait states in the SBIU arbitration logic

SBIU Arbitration	1 DSCLK
L2 Memory access latency	0 (Hidden)
Acknowledge Generation	1 DSCLK Cycle
DSP-DMA to L2 Memory Write Latency	2 DSCLKs

Table 18 DSP DMA-L2 Write Access

4.3.8.9 System-DPBUS Access Times

SBIU Arbitration	1 DSCLK
DPBUS access latency	1 DSCLK
Acknowledge Generation	1 DSCLK to 1 BCLK (depending on the clk ratio)
RBUS to L2 Memory Read Latency	
BCLK:DSCLK = 1:1	3 DSCLKs or
BCLK:DSCLK = 2:1	(2 DSCLKs + 1 BCLK) or
BCLK:DSCLK ≥ 3:1	1 BCLK**

Table 19 System-DPBUS Read Access

SBIU Arbitration	1 DSCLK
DPBUS access latency	0 (Hidden)
Acknowledge Generation	1 DSCLK to 1 BCLK (depending on the clk ratio)
RBUS to L2 Memory Read Latency	
BCLK:DSCLK = 1:1	2 DSCLKs or
BCLK:DSCLK ≥ 2:1	1 BCLK**

Table 20 System-DPBUS Write Access

4.3.8.10 System-L1 Access Times

4.3.8.11 System-L2 Access Times

SBIU Arbitration	1 DSCLK
L2 memory access latency	2 DSCLKs

Acknowledge Generation	1 DSCLK
RBUS to L2 Memory Read Latency	4 DSCLKs or 2 BCLKs ²

Table 21 System-L2 Read Access

SBIU Arbitration	1 DSCLK
L2 memory access latency	0 (Hidden)
Acknowledge Generation	1 DSCLK
RBUS to L2 Memory Write Latency	2 DSCLKs or 1 BCLK ³

Table 22 System-L2 Write Access

4.3.9 System Synchronization (SSYNC) from FRIO Core

On the assertion of the `co_ssyclf` pin by the FRIO Core, the SBIU waits for all the pending FRIO Core requests to be completed before asserting the `ci_ssyclacklf` signal.

4.4 L2 Memory

The L2 Memory provides dedicated SRAM primarily for the DSP, and DSP DMA. However, it can also be accessed by the MCU, the System DMA, and the EAPI. The L2 Memory can be accessed any time the DSCLK tree is driven (even if the Frio core is not being clocked). As such, it is the logical target for much of the data coming in from the DSP peripherals.

The L2 memory offers marginally faster access time than the System RAM; however, it is fully pipelined, and has significantly higher throughput than the System RAM. The organization and access times have already been discussed, above.

4.5 DSP Coprocessors

4.5.1 Viterbi Coprocessor

The Viterbi coprocessor reduces the MIPS required for the DSP to perform the Channel Equalization and Channel Decoding tasks. The Viterbi coprocessor is attached to the DSP as a local peripheral. The communication between the DSP and the coprocessor is performed through a set of registers, which are mapped into the System MMR space, which otherwise include registers accessed via the DPBUS. Two interrupts, for input data and output data, can be used to schedule data feed and data read operations.

The coprocessor is the only component besides the DSP, and SBIU, clocked at the full DCLK frequency. For this reason, the Viterbi coprocessor resides on a dedicated interface of the SBIU.

Only the DSP can access the Viterbi coprocessor.

² Note: Latency of 4 DSCLKs is for the 1:1 DSCLK:BCLK ratio. For all the other clock ratios, the latency is 2 BCLKs

³ Note: Latency of 2 DSCLKs is for the 1:1 DSCLK:BCLK ratio. For all the other clock ratios, the latency is 1 BCLK.

This coprocessor has significant enhancements from that of the Hercules system, including:

- EDGE channel equalization support
- Higher clock rate

Opens:

- *Need to define the specific enhancements to the Viterbi coprocessor to support EDGE*
- *CoreMMR access to Viterbi? This would reduce write latency, bypassing the write buffer; therefore, the coprocessor would be more tightly coupled to the DSP instruction pipeline. This would require change to the DSP core, however.*

4.5.2 Cipher Coprocessor

Very similar to the Cipher module on Hermes, although it runs at higher clock rate. Additionally, the coprocessor must support the GEA/GEA2 encryption algorithms required for EDGE.

Note also that there is a 2nd Cipher unit, on the PBUS, to support the MCU.

4.5.3 External Coprocessor Interface

The external coprocessor interface provides a high-bandwidth port to a dedicated accelerator function, such as a RAKE receiver. This interface is only available in the extended bondout package.

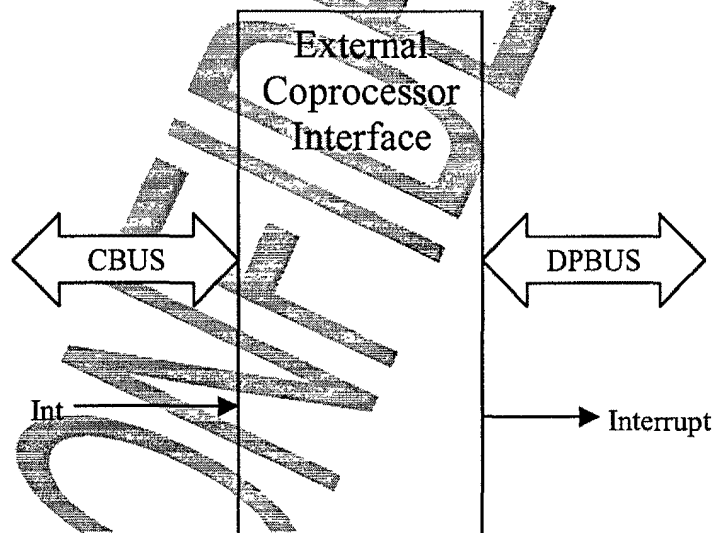


Figure 11 External Coprocessor Interface

Details of the CBUS interface and protocols are TBD. However, at a minimum, the CBUS provides:

- Single master, multiple slave support
- 16-bit parallel, bidirectional data bus
- Address range of at least $2^8 = 256$ byte locations
- Read- and Write-valid strobes
- Transfer completion response from the slave

The ECI provides CBUS master capability, and at least one slave chip select output.

Opens:

- *Should the ECI also provide slave capability?*
- *Should the ECI be able to master the ABUS?*
- *Example timing diagrams, closure on bus protocols*

4.6 DSP DMA

In addition to the system DMA controller, the DSP subsystem of Monaco also has a dedicated DMA controller. This controller is similar to the system DMA controller (i.e., same number of channels, same amount of RAM), with enhancements for better throughput. DMA channel allocation is programmable, however, the following list provides an example of channel allocation:

- TEP support (minimum 1 dedicated, up to 5 additional need to be available for multi-threaded operation of TEP)
- B SPORT receive B SPORT transmit
- Cipher coprocessor receive/transmit
- External coprocessor interface support (development bond-out package only)
- Memory block copy (e.g., FLASH to code L1 transfer)

The DSP DMA controller masters the DAB bus. Through the SBIU, the DSP DMA controller can access all of Monaco's memory-mapped resources. The DSP DMA memory map is a superset of the system DMA memory map, since the DSP DMA can access DSP resources not visible to the system DMA (e.g., DPBUS slaves and L1 memory). The DSP DMA can transfer up to 32 bits per access.

The DSP DMA controller can be configured by the DSP, the TEP, the MCU, or the EAP (in the extended bondout package). Typically, however, only the DSP or the TEP would control and configure the DSP DMA.

See the System DMA section, on pg 75, for more information.

4.7 DSP Peripherals

4.7.1 Serial Port

For Monaco, only one serial port (B SPORT) resides on the DPBUS. The A SPORT (referred to as the V SPORT in Hercules) resides in the GSM System MCU peripherals. This change allows the MCU to better support audio playback (e.g., MP3), independent of DSP state.

4.7.1.1 B Serial Port

The Baseband SPORT is used to transmit uplink burst data, and receive I/Q samples. Transfers are always initiated by the ABB.

The DSP DMA can be programmed to support auto-buffering for the B SPORT. Input/output buffer space can be allocated in either the DSP L1 memory (configured as SRAM), or the L2 memory. However, the L1 memory cannot be accessed when the DCLK is disabled. The DABUS dynamic prioritization arbiter in the SBIU can be programmed to ensure that the L2 memory latency seen by the DSP DMA is low enough to avoid data loss during autobuffering.

The B SPORT interrupt is routed to the DSP IRQ. The DSP flag interrupts of Hercules are not required to link prioritization of the B SPORT with system interrupts.

Monaco employs the Enhanced Generic Serial Ports (EGSP), for the B SPORT function. See the Generic Serial Port chapter for more information on this module.

4.7.2 TEP

- See Timing and Event Processor (TEP) chapter, page 45.

4.7.3 High-Speed Data Logging and Trace Functions (HSL)

The High Speed Logger (HSL) interface facilitates (relatively) non-invasive, real-time, external data logging of key events internal to Monaco. Using this feature, an external data logger (e.g., a PC), can capture internal state, such as a highly-compressed DSP address trace, then post process the log to aid in debug of software running on the Monaco baseband processor.

The HSL can be programmed to transmit three types of data streams, with the following priorities:

1. TEP instructions, showing interrupts to the DSP, DSP DMA, or MCU, and GPO/GPIO port signaling.
2. HSL software log messages from the DSP.
3. DSP address trace.

The external interface consists of up to 8 data pins, and a clock. This port can be selected to come out over the USC/GPIO pins. In the extended bond out package, the HSL port is also available on dedicated pins, which enables simultaneous HSL logging and serial port operation.

Note that the off-chip, external data logger is responsible for time stamp generation. This function is not provided on Monaco.

The Frio DSP supports a dedicated PC Dump interface to the HSL module. This interface provides the entire 31-bit PC, along with a valid bit indicating when the instruction at the PC reaches the writeback stage of the instruction pipeline. Additionally, the DSP provides a signal indicating discontinuities in the PC (i.e., interrupts, branches, call/returns).

To conserve power consumption, the dedicated DSP/HSL interface is disabled until enabled (by setting the Enable HSL interface bit in the core MMR TBD). The HSL module is also accessible to both processors on the DPBUS. DMA cannot be used to transfer data to the HSL.

Additionally, the Frio core has trace capture capability, including an internal 15-element trace buffer, with optional nested loop compression feature. However, the trace buffer can only be dumped off core by either reading a PC register via the low-bandwidth JTAG interface, or by taking a trace buffer full exception, and reading out the contents of the trace buffer on the ISR. This trace capture facility may be too intrusive for MIPS-intensive real-time applications.

The Monaco HSL module includes the following changes from that of the Hercules HSL:

1. Modification of the DSP Address Trace Format (14-bit PC -> 31-bit target address).
2. Timing modifications to allow the HSL to sample at full DSP frequency.
3. The PM_OVLAY and DDL registers are not present in the Frio-based system, and are not supported in the HSL.
4. Changes to interface with the new TEP module, rather than the microSM.
5. The hardware breakpoint feature of the HSL is removed for Monaco, since Frio has extensive internal breakpoint capabilities.

Opens:

1. *HSL control word encodings*
2. *Another possible h/w compression technique would be to only transmit jump targets for the indirect branches, and interrupts. If the branch can be resolved statically, and the data logger on the PC has access to the source code, then the static branches can be determined w/o transmitting the target.*
3. *Should the HSL support loop compression? How many levels?*

4.7.4 Programmable Flag IO

The FIO block provides a flexible communication mechanism between the DSP and the system interrupt controller, HSL, and DAI (Digital Audio Interface, via the USC or JTAG ports).

This block replaces the Programmable IO and Flags modules of the Hercules 2181 DSP core. The following table shows the mapping between the equivalent programmable flag IO functions of Hercules and Monaco:

Monaco FIO Assignment	Hercules Function	Input Connection	Output Connection	Description
FIO[9]	PF[4]	HSL Msg. Filter	N/A	Input from HSL, defines if msg. Is filtered
FIO[8:6]	FL[2:0]	N/A	DSP IRQ CTRL	Output used as interrupt to DSP IRQ CTRL
FIO[5]	PF[2]	DAIRESET & USC	N/A	GPI Input
FIO[4:2]	PF[7:5]	GPIO	GPIO	Alternative Pin Function on Synth i/f pins
FIO[1]	PF[1]	N/A	DAI[1] (i/o) and USC (o)	General Purpose pin, on USC as o/p only
FIO[0]	PF[0]	N/A	DAI[0] (i/o) and USC (o)	General Purpose pin, on USC as o/p only

Table 23: Programmable Flag IO Function Assignments

Each Flag bit has separate Flag (status or control, depending on direction), direction, edge/level sensitivity, both edge sensitive, level polarity, and two interrupt mask control bits, grouped by function into seven registers. The FLAG, MASKA, and MASKB registers have separate set and clear register address decodes, to minimize the software overhead required to change these registers.

The two maskable interrupt outputs of the FIO block provide additional interrupt functions which are connected directly to the DSP General Purpose Interrupt inputs GP[9:8]. Since they bypass the system interrupt controller, these interrupts are not managed (or prioritized) by the system interrupt controller. They must be explicitly unmasked in the Frio core interrupt controller IMASK register. They are the lowest priority interrupts in the DSP sub-system.

Opens:

- *Should the FIO interrupts be mapped through the System Interrupt Controller, instead of bypassing it? If so, which assignments should they take? Note that the Frio core already has s/w interrupt capability internal to the core.*

- *Should we increase the number of FIO, and map more to pins under alternative configurations? Perhaps selectable between GPIO and FIO, to improve DSP access time to them?*

4.8 Timing and Event Processor (TEP)

This module replaces the MicroSM, Slow Clock Module, and Frequency Detection modules of Hermes. It provides significant new capability, as well as programming model simplifications vs. the same function in MicroSM. The interested reader should obtain [12], for additional information.

4.8.1 Background

For solutions/terminals for single-mode wireless systems like GSM, the baseband keeps synchronized to the wireless system clock and may continuously adjust to stay so. Every action on the wireless channel conducted by the terminal is scheduled within a certain precision, i.e. 1 Qbit for GSM. To keep track of time and schedule events within such precision, a dedicated hardware timer module is required to generate timed interrupt of processors, initiate other modules, and generate control signals for off-chip circuits, like the radio unit. An example of such is the μ SM module from the Hercules/Hermes chipsets [3][8].

For multi-mode solutions, like GSM/EDGE/3G, similar requirements regarding timing precision do exist, but in each wireless system with independent time bases. The baseband may be required to keep track of two or more wireless systems simultaneously and since there is only one reference oscillator, it will not be possible to do lasting AFC based on wireless system inputs.

Rather the Monaco system design approach will be to apply a free-running oscillator, and then by software, project the time-bases for the various simultaneous supported wireless systems onto this free-running clock. Provided this clock is stable, its frequency is irrelevant and its resolution need not be very high. Hence an independent 32 kHz counter will be used. Once this clock is synchronized to the wireless system by *software means*, any time can be given as an *absolute time-value* of this counter plus a number of a higher frequency clock, which provide a fraction of the 32 kHz clock. *This approach removes bindings to any wireless system. Additional benefits are built-in low power and the freedom in selecting the high frequency system oscillator.*

A standard 32 kHz oscillator plus XTAL do not offer sufficient stability over temperature and time. Hence, measures to calibrate the free-running oscillator must be included in the system as well.

The Layer 1 architecture is based on multi-threading with typically several threads for each wireless system handling various sets of tasks. The threads are, in the software, mapped down to 'Resource Queues' – one for each set of resources being shared between the wireless systems – each containing the synchronous tasks to be conducted on that set of resources. Wireless systems not sharing resources, like GSM and Bluetooth, with separate radio resources will be mapped onto different resource queues. A resource queue will be mapped into *A/B-buffer-like* streams. These instructions streams must each be mapped independently onto the Timing- and Event Processor for *timing exact* execution. This calls for a *multi-sequencing hardware architecture* of the TEP as well, which - of course - for reasons of flexibility must be fully programmable, executing several streams of instructions concurrently – one for each thread.

A conceptual idea of the TEP and Layer1 architectures is that TEP must handle all time-driven and time-critical tasks. This implies the Layer1 can be fully event-driven [11]. The TEP must interface to all hardware modules requiring timing precise actions. This is illustrated of the functional diagram at Figure 12.

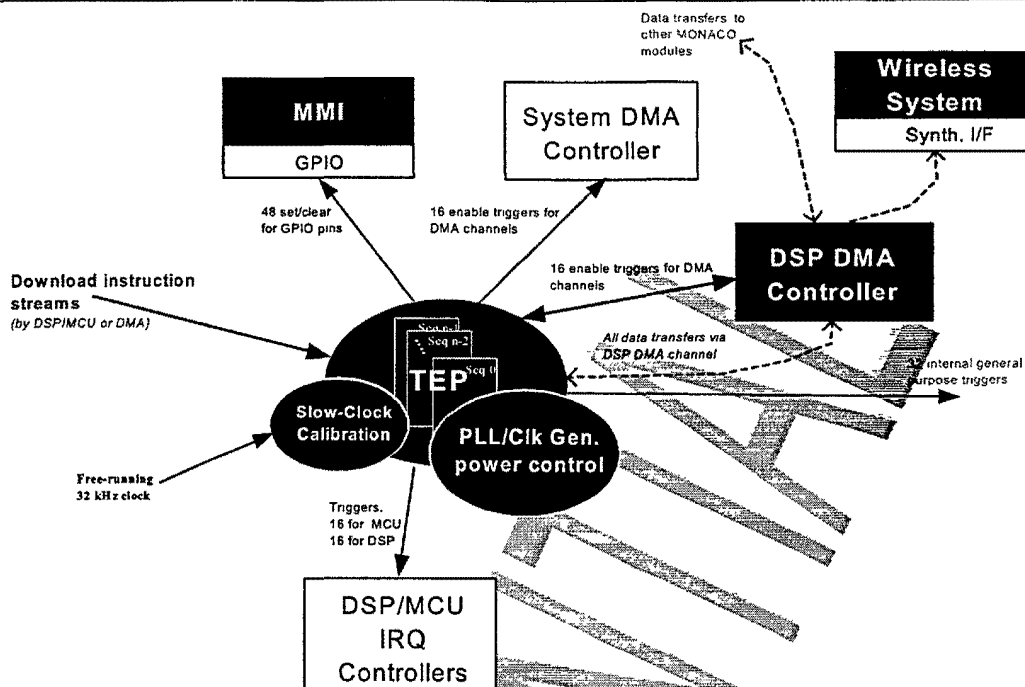


Figure 12: Simplified functional diagram for the Monaco Timebase and Event Processor.

The interface to the MMI/GPIO are setting pins for controlling external devices, like radio unit power up/down, RxOn, etc. Interfaces to System DMA, DSP/MCU IRQ controllers, are for enabling *time-specific DMA transfers* and generating *timed interrupts* to the processors, respectively. The interface to the DSP DMA controller may also do for enabling channels at specific times. However, this DMA controller is tightly coupled to the TEP and will be used by the TEP to implement various sorts of data transfers, like writing new programming words to the synthesizer I/F or setting up another DMA channel for e.g. burst reception. One channel will be dedicated for this purpose and one additional DMA channel for each Sequencer will supposedly be allocated (statically).

4.9 TEP Features

The TEP module supports the following features:

- The TEP is the only module in the system keeping track of actual time, hence it handles/schedules all events within the Monaco hardware and Layer 1 software requiring timing precise execution, like:
 - Generating time-accurate trigger signals for other hardware modules and
 - Controlling external pins, e.g. like RxOn for radio unit power up/down or
 - Interruption of the processors at specific times for triggering execution of software
- Other tasks, including transfer of programming data to the RF synthesizer, and programming and setup of DMA channels for baseband data reception. These tasks are conducted utilizing the DSP DMA controller.
- Relaxed software load of processors (DSP and MCU) for servicing the module compared to Hercules and enable these to do event-driven processing. This is achieved through:

- Introducing the ability to specify events in *absolute time* – using the 32 kHz clock as input for the *absolute timebase*.
- Providing a dedicated instruction-set allowing dedicated yet flexible programming of the TEP.
- Providing conditional execution of instructions based on evaluating any bit within the TEP RAM space.
- Providing extendable hardware architecture with hardware sequencers supporting the instruction-set and numbered to match the number of concurrent software threads.
- Support multi-mode wireless systems *concurrently* with different time-bases, like slotted GSM and 3G modes, where the different systems call for different time-base. Together with Layer 1 handling this will be achieved using the 32 kHz time-base and the SysClk for fine-grain.
- Provide the necessary timing stability by means of,
 - Continuous calibration of the 32 kHz clock used for absolute time base against the more temperature-stable system clock, SysClk.
 - Fine-grain delays must be measured relative to the 32 kHz timebase using the 13 (or 26) MHz SysClk.
 - Provide a *fail-safe alternative* to the 32 kHz XTAL oscillator-based clocking by enabling the use of an integer-divided SysClk as slow clock - with seamless switching between the two sources.

Provide *automatic SysClk oscillator and system PLL power-control*. Whenever the chip system clock, SysClk, or system PLL are not required they must be powered off. When the oscillator can be powered off for a long time (compared to the 13 MHz oscillator warm up period), the TEP must power down as well as power up the oscillator (coordinated with other modules like UARTs, etc which may be needing the clock).

4.10 Functional Description

The TEP module encapsulates the features offered by the μ SM and slow clocking modules from Hercules [1]. These features boils down to re/setting I/O pins, generating interrupts to processors and enabling DMA memory-to-memory transfers at specific instances of time plus controlling pins for powering up/down system oscillator PLLs, e.g. slow clocking, and SysClk/32kHz calibration. All of this is handled in similar fashion by TEP.

4.10.1 Event-Driven Layer 1 Scenario

A conceptual new idea behind TEP and Layer 1 architectures [14] is that TEP must handle all time-driven and time-critical tasks, and hence, allowing the Layer 1 to be fully event-driven. In Figure 13 is shown the interface between Layer 1 and TEP. The system is, in general, multi-mode with *concurrent* handling of modes. The 'Resource A' is a resource request queue for one or more of the wireless systems currently running and sharing resources. The resource requests are translated by the 'Executer' to instruction streams for the TEP hardware and passed to it – all pure event-driven. Whenever, the request (say receiving paging info) has been granted the TEP may generate an interrupt – by means of an instruction – to the host processor for flagging Layer 1.

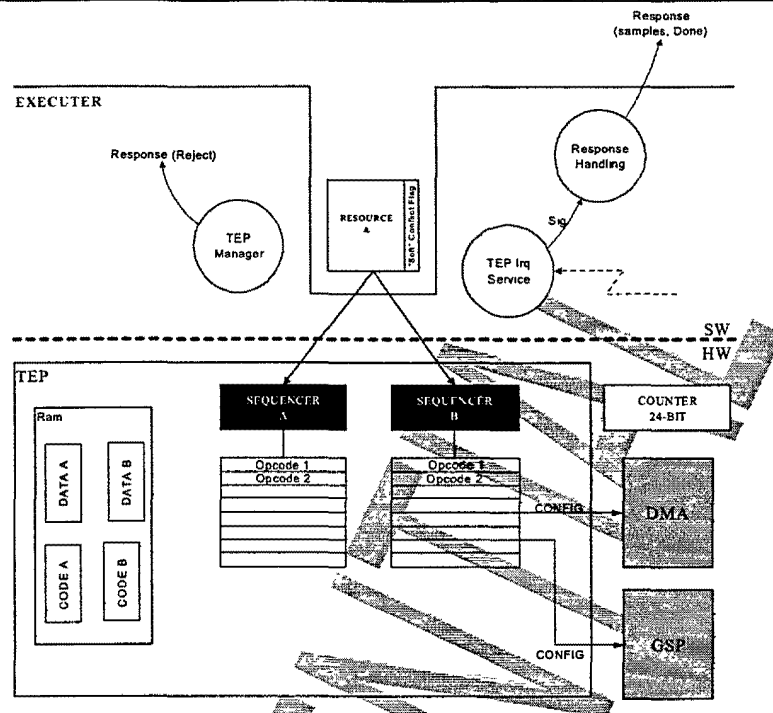


Figure 13: Layer 1 'Executer' block and the TEP module interface

For the example with the paging info reception the TEP would need to set up the serial BSPORT as well as a DMA channel for data storing. This is part of the instruction stream passed to the TEP as well as other tasks like programming synthesizers, reception gain, etc. The instruction stream - or program actually - may be run many times with only slight modifications like altering programming data to synthesizer. Hence, there must be separable memory areas for data and code allowing changing data without altering instructions, as well as control for rerunning code.

Execution of instruction streams is carried out in hardware sequencers and there are as many sequencers as threads concurrently supported by the TEP module. The Layer 1 software architecture requires two sequencers per wireless system. The timing-precise features flow from the execution of instruction. TEP uses three clocks for offering these features, the conventional system clock, SysClk, a SysClk divided clock, time-tick, and a slow 32kHz. Due to the multi-mode scenario neither of these clocks can correspond to the (several) wireless system time bases. Hence, for obvious reasons of power consumption and the nice integration of the slow-clocking feature, the 32 kHz will be applied as the overall timebase. This totally removes the time-base/clock switching between the applied system clock and a free-running 32kHz, which reduces software efforts and fits the concept of purely event-driven Layer 1 software execution. In Figure 14 is shown a timing example of the TEP execution. In this example, instructions with zero delta-time are just executed sequentially; delta-times are calculated using the local time-tick. Notice the clock control is determined by TEP without software intervention.

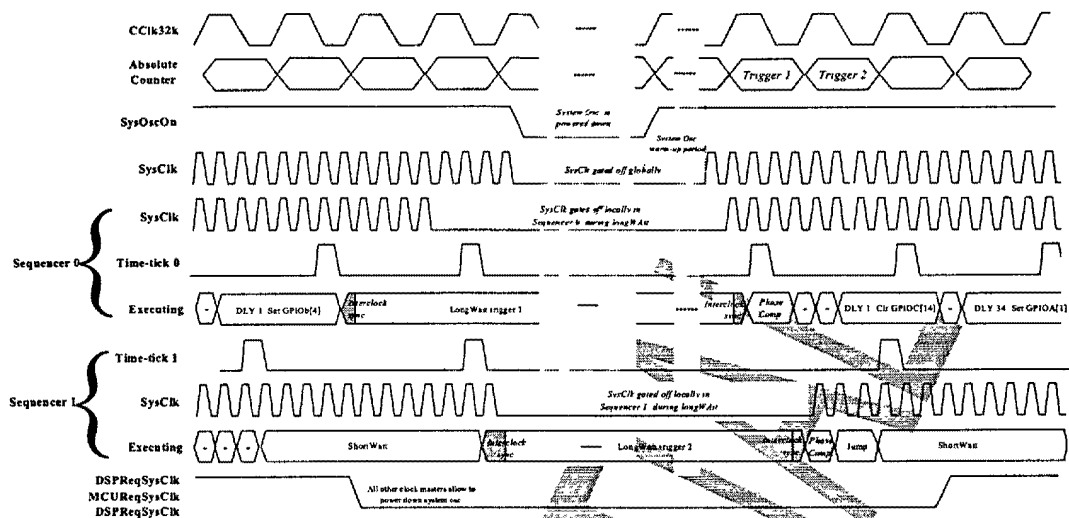


Figure 14: Exemplified timing scenario of the TEP for the case of two Sequencers

4.10.2 Clock Calibration

The 32 kHz clock is free-running, which means software must do a mapping between this free-running counter and times of a wireless system as well as continuously compensate (i.e. synchronize) during operation. This requires frequency stability of the 32 kHz clock better than normally offered by standard oscillator solutions. Especially temperature sensitivity will pose a problem. Therefore some sort of calibration is required. Whenever running the SysClk will be compensated regarding temperature and in general be much more frequency stable than the 32 kHz clock. Hence the 32 kHz clock can be measured during such periods towards the SysClk - simply by counting the number of SysClk cycles in a fixed number of 32 kHz cycles. Redoing this periodically gains information for calibrating the 32 kHz clock. The adjusting technique is illustrated of Figure 15. The case of down adjusting by (9-1)/9 is exemplified to the right. Adjustment will always be to a lower frequency which is implemented by removing one clock cycle every given period. This adjusts to (period-1)/period of the input clock.

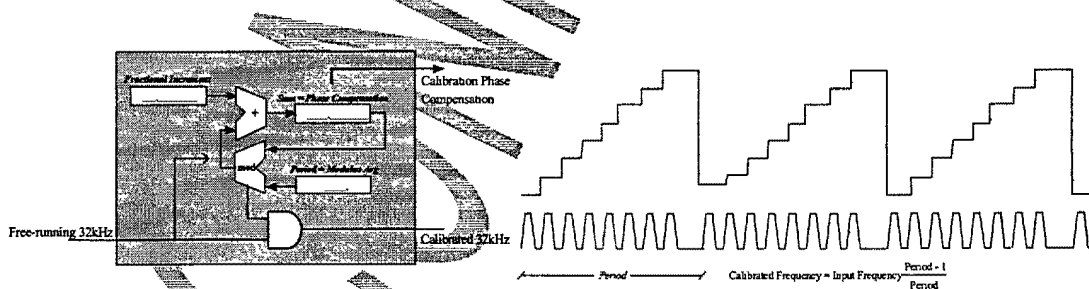


Figure 15: Calibration adjustment of the free-running 32 kHz clock by fractional-N division

However, it also introduces phase errors, making it impossible to hit the correct time of each clock cycle unless phase compensated. The phase compensation increases linearly over the period, reaching the length of one clock cycle at the end of the period. This is computed *on-the-fly* with the circuit of Figure 15. The phase compensation will be implemented as an additional delay of SysClk cycles

associated with the 'LongWait' instruction. By proper selection of the 'Fractional Increment' and the 'Period' the computed phase compensation will be expressed in SysClk directly.

4.10.3 DSP DMA Control

The TEP must be able to setup DMA channels, program synthesizers, etc. This will be featured by use of one fixed allocated DMA channel of the DSP DMA controller as shown of Figure 16. The DSP DMA controller offers sufficient channels to allow such fixed allocation, and beneficially, use of a DMA channel avoids adding a bus master function to the TEP as well as avoiding adding an additional master port to the arbiters.

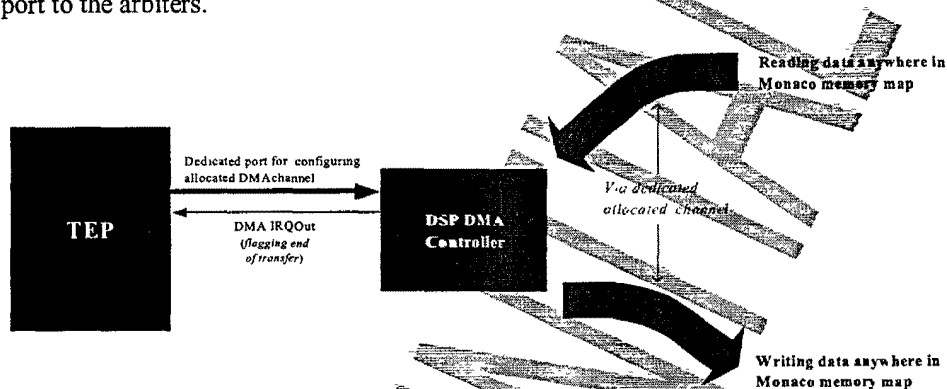


Figure 16: Block diagram showing the use of dedicated DMA channel for data read/write access anywhere in Monaco address space, including setting up other DMA channels.

Depending on the addresses to access the latency time for the access varies, due to variable latency in arbiters, slower clocked buses, etc. However, the *worst-case latency* for each arbiter can be calculated *a priori* and taking into account in writing the TEP code.

4.11 Hercules μ SM Backward-Compatibility

The TEP will not offer code compatibility compared to the μ SM. However, functional wise the TEP may be thought of as a series of hardware Sequencers each similar to the μ SM except for some extensions, like the 32kHz clocked delay, DMA integration and conditional execution. Hence, by not using the TEP extensions and limiting the number of Sequencers to one will offer a μ SM-like behavior. Such scenario does support the same logical set of instructions and interfaces to Slow Clocking, GPIO, etc. Only serious obstacle is the lack of a FIFO feature in the memory and the associated exception flagging. This can be implemented in software – on the host side by software-wise compute a circular address behavior (like a FIFO) – and on the TEP side by use of 'jump' instructions to implement the required circular behavior.

This means conceptually, that existing software design from Hercules/Hermes can be made run-able on Monaco with TEP by means of software porting efforts.

4.12 TEP high-level Architecture

A simplified view of the TEP architecture is shown in Figure 17 as a block diagram. Basically it consists of a series of sequencers, each for executing one thread, a multi-ported memory for instruction streams and data, interfaces to the DSP DPBUS bus and the DSP DMA controller, some common clock and control and various logic for inter-thread conflict resolving.

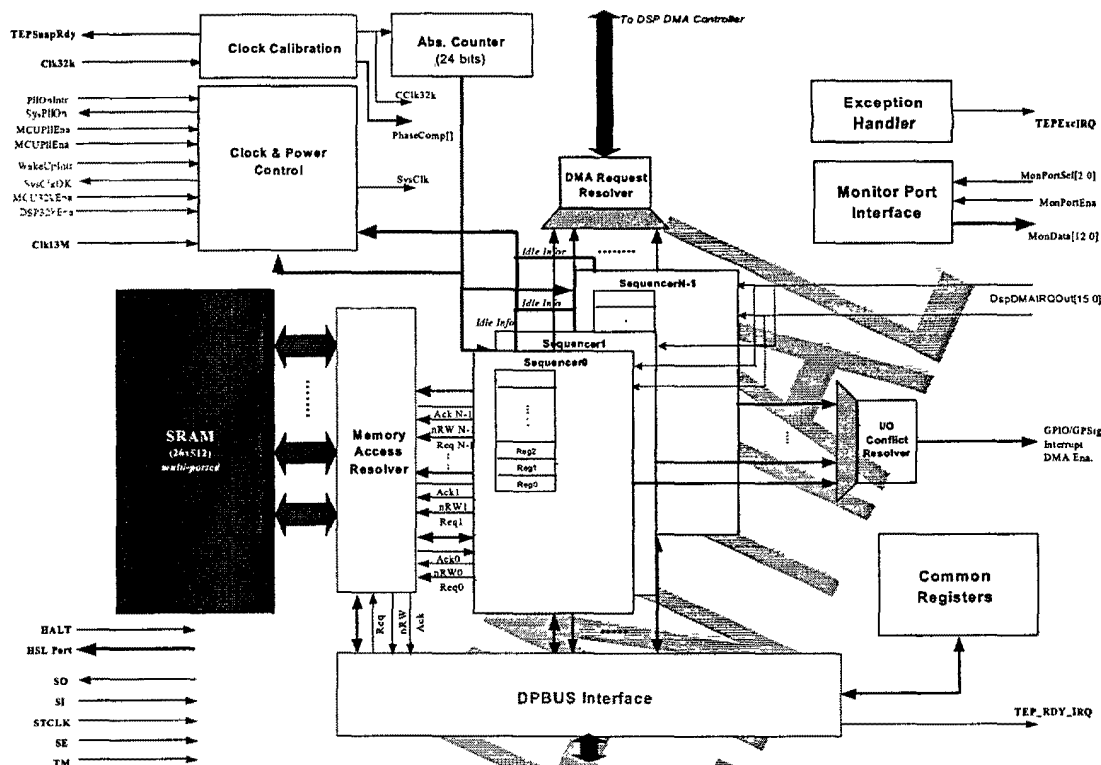


Figure 17: Simplified block diagram of the TEP architecture.

The 'Clock & Power Control' block includes functionally the oscillator power control of the Hercules Slow-Clocking module. The 'Clock Calibration' block implements 32kHz to SysClk calibration feature functional compatible to the Hercules Slow-Clocking module.

5. External Applications Processor Interface (EAPI)

5.1 EAPI Overview

The external applications processor interface provides direct external access to internal Monaco resources, via the EABUS. This interface is only available in the extended bondout package.

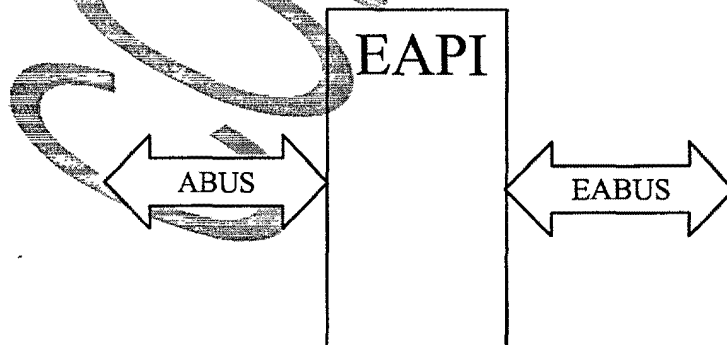


Figure 18 External Applications Processor Interface

EAPI provides a host interface into the Monaco chip. The EAPI module synchronizes the external processor to the Monaco BCLK domain. The EAPI allows Monaco to map into the host system as a simple RAM interface with wait state capability. In addition to the wait state, the EAPI external interface includes an interrupt output (Int), which is asserted on unsupported accesses to the EAPI. Unsupported accesses include access to protected internal resources, or misaligned address accesses. In either case, the EAPI external bus operation is completed by the EAPI; however, write data NOPs, and data provided on a read is not reliable. A base address register is provided in the EAPI, such that all of the internal memory and peripherals of Monaco can be accessed, by appropriate programming of the base address register (see EAPI Memory Protection, below).

Details of the EABUS interface and protocols are TBD. However, at a minimum, the EABUS provides:

- Single slave support, selected by a chip select
- 16-bit parallel, bidirectional data bus
- Address range of at least $2^8 = 256$ byte locations.
- Read- and Write-valid strobes
- Transfer complete/wait slave output

5.2 EAPI Memory Protection

A simple programmable protection scheme is provided, which allows system software internal to Monaco to control which internal resources can be accessed from the EAPI. These include:

- PBUS access protected on a module by module basis
- RBUS allows a programmable memory region to be accessed
 - DPBUS access protected on a module by module basis
 - L2 SRAM has four separate protection regions
 - SysRAM has four separate protection regions
- EBUS allows a programmable memory region to be accessed
 - Individual write and read enables

Protection can be completely disabled if not needed or provided in the application processor.

The protection description registers are in the EAPI module, and are mapped into the PBUS House Keeping address region.

Opens:

- *Should the EABUS address bus width be increased to avoid a segmented (base + offset) address model?*
- *Should the ECI be able to master the EABUS?*
- *Should the EABUS be a simple serial interface (e.g., SPI), rather than a parallel interface?*
- *Example timing diagrams, closure on bus protocols*

6. Enhanced Generic Serial Port

6.1 Introduction

Inter IC communications at minimum pin count, without standardization, generated a huge number of norm proposals (each vendor pushing for its own!). However, going through the specifications of those synchronous/asynchronous protocols, a common structure can be realized around a programmable state machine and a set of highly configurable operators such as shifting units, event counters and bidirectional IO's.

The eGSP (Enhanced Generic Serial Port) is such a microcontroller targeted at handling various Serial Interfacing functions. Among those protocols the primary targets were UART (FD/HD), HC11, I²C, IrDA, I²S, Synchronous serial protocols using FS or enables such as Serial Display, Synthesizer, power management, and ABB interfaces. Of course since the eGSP is a programmable machine, this list is non exhaustive. The GSP (Generic Serial Port) concept isn't new in ADI and was first implemented in the MSP430 (Hercules) Digital Baseband DSP system. This first design successfully emulated a UART (Full and half duplex), some Serial Display, Power Management Chip and Synthesizer Interfaces. However, the development of higher rate serial links and even more, slave interfaces were hardly possible. The eGSP tries to work around those difficulties by pushing forward the microcontroller aspect and an highly configurable/controllable operator set. The amount of modifications in the GSP architecture and its instruction set make the eGSP a completely new design, so whatever was done in the past would have to be re-coded and validated with this new implementation. The main modifications are:

- New Instruction Set, with a different memory organization and the capability of single cycle parallel instructions pair execution
- 2 stages pipeline microcontroller
- New or improved operators (Comparator, condition composer, Rx/Tx fifo, generic counters, integrated bit counters within the shift units, and lots of configuration options)
- Symmetrical design (Full duplex/Half duplex)
- Highly configurable and dynamically controllable clock generator (5 sub-dividers!)

6.2 eGSP Architecture

The eGSP is available in two Hardware versions, mainly to minimize the gate count where a full duplex eGSP is not needed. This versioning is transparent to the user, other than all the protocols won't be supported in the "limited" version. In Half duplex, only one Control flow Unit is needed and as a consequence, only one Generic Counter and one Clock generator will be available. However, to insure the SW compatibility for a same protocol in between the two versions of the eGSP, the clock routing part of the Rx way is maintained using the primary clocks of the Tx clock generator as its own primary clock. The diagram below roughly illustrates the eGSP top level architecture.

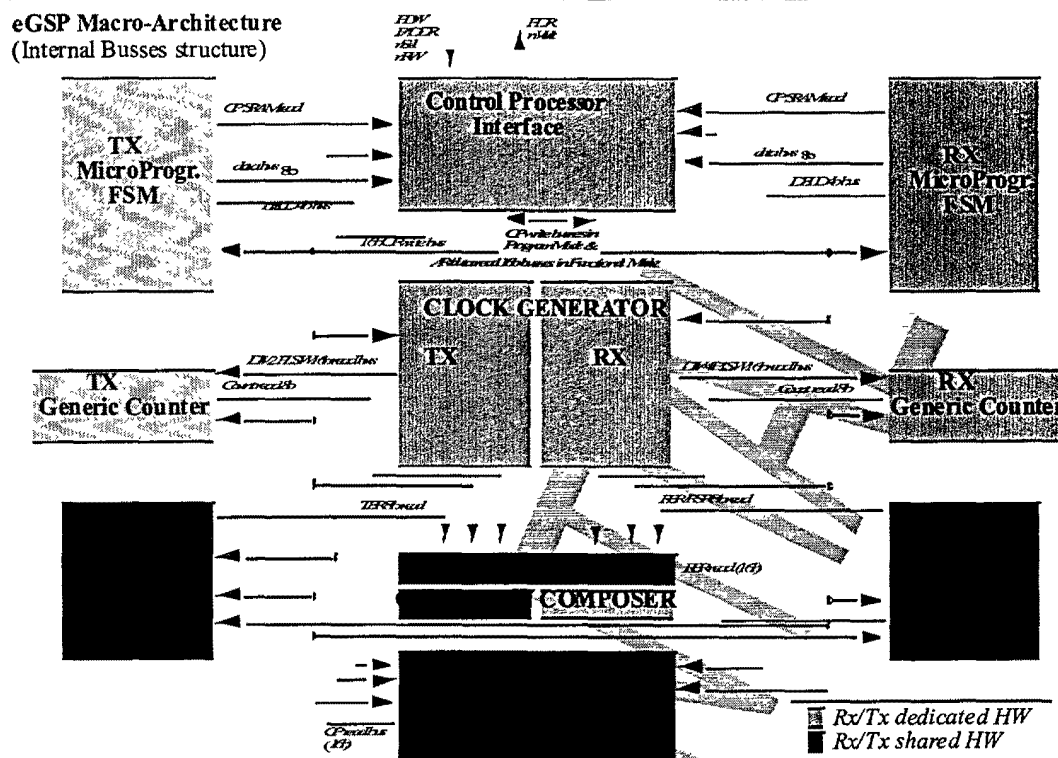


Figure 19 eGSP Block Diagram

Editor's note: will add much more discussion on eGSP in next release...

7. Peripheral Bus Specification

7.1 Peripherals and Interfaces

The PBUS interface is controlled via the PBUS arbiter. This module determines the access priorities between the MCU, DSP, EAP, and DMA requests. The PBUS arbiter, and bus, operate at the BCLK frequency. The PBUS arbiter for Monaco is identical to that of the RBUS arbiter. This change was made to provide the additional arbitration port for the EAP.

The PBUS is identical to that of Hercules, except for the changes in the peripheral configuration, and the higher maximum operating frequency.

As with the Hercules definition, the PBUS supports half-word aligned 16-bit accesses. Additionally, byte-aligned 8-bit reads are supported⁴.

The PBCR definition and location are the same as in Hercules, except that the InsertWaitState field need be set only if the BCLK frequency exceeds 75MHz.

When a peripheral generates asynchronous READ data, it will assert a WAIT state to the PBUS arbiter. This forces the arbiter to perform a second read, and compare the two data values. If they are the same, then the read cycle is completed. If not, the PBUS arbiter will perform a 3rd READ cycle before completing the access. The access will therefore require either 2 or 3 BCLK cycles.

⁴ 8-bit reads may be byte aligned, but 8-bit writes will be interpreted as 16-bit accesses and will write unknown data to either the upper or lower unused byte location.

A peripheral access may require multiple WAIT states. When a peripheral WRITE cycle needs to be retimed to a slower clock, a second WRITE cycle will incur additional WAIT status until the first access is completed. The first access requires no WAIT states.

CONFIDENTIAL

The table below shows the access capabilities for each peripheral.

No.	Peripheral	RD Access	WR Access	Comment
0	SYS IRQ CTRL	0 WS	0 WS	MCU. DSP IRQ on DPBUS
1	MMI: GPIO, Keypad	0 to 2 WS	0 WS	2 WS for asynch i/p pads
2	DISPLAY GSPc	0 to 2 WS	0 WS	Asynch i/p data read on IRQ
3	BMC GSPd	0 to 2 WS	0 WS	Asynch i/p data read on IRQ
4	SIM	WS possible	WS possible	WS on back to back accesses
5	WDT	0 WS	0 WS	
6	RTC	2 WS	WS possible	Specific reg write order
7	SYSTEM DMA	1 WS	WS possible	1 WS on WR if back to back
8a	General Timers	0 WS	0 WS	WS on back to back write
9	SYNTH GSPe	0 to 2 WS	0 WS	Asynch i/p data read on IRQ
10	A SPORT GSPh	2 WS	0 WS	WS on back to back write
11	C SPORT	2 WS	0 WS	
13:12	Reserved	N/A	N/A	Access not allowed
14	USC	2 WS	0 WS	Only RD WS on SPSSR
15	UART GSPa	0 to 2 WS	0 WS	Asynch i/p data read on IRQ
16	DEBUG GSPb	0 to 2 WS	0 WS	Asynch i/p data read on IRQ
17	MMC	WS possible	WS possible	WS on back to back accesses
18	SPI GSPf	0 to 2 WS	0 WS	Asynch i/p data read on IRQ
19	USB	0 to 2 WS	0 WS	Asynch i/p data read on IRQ
20	Reserved	N/A	N/A	Access not allowed
21	System Cipher	0 WS	0 WS	
22	Pwr Mngmt GSPg	0 to 2 WS	0 WS	Asynch i/p data read on IRQ
23	Clock Gen, Arb Ctl, Bus Select	0 WS	0 WS	SBUS peripherals from Hercules

Table 24 PBUS Peripheral Access Constraints

The peripherals served by the PBUS interface are grouped into three categories, according to their system functions:

1. MMI Functions General System Interfaces
2. House Keeping Functions Data Transfer, Timers, and Interrupt Controllers
3. Communication Functions System Timebase and Dedicated RF interfaces

Each peripheral is assigned a 256K byte memory space, although only a few locations may be required. The unused addresses are not decoded, and may return unknown data. WRITE cycles to these locations are not allowed, and may produce unwanted results.

7.2 MMI Functions

The MMI function set is defined by those modules that cover the hardware interfaces of the baseband. These cover hardware configurations and user configurable functions.

The MMI is comprised of the MMI module itself, and peripherals such as the RTC, SIM, UART, USB, MMC, USC, and the baseband pin interfaces.

7.2.1 MMI Module

This section is identical to that of the Hercules definition, with the exception of:

- The GPIO / TEP interface changes per the description in the TEP chapter.
- The emulation Control module changes per the new emulation interface of the Frio DSP. The Frio DSP emulator interface is through a dedicated JTAG port.
- Pin function select: The Monaco pinout differs from that of Hercules, therefore the alternate pinouts controlled by the pinout function select control registers are different, and are TBD.
- CLKOUT Config Register: MicroSM replaced by TEP. CLK13M can now run up to 26MHz. MCU_PLL and DSP_PLL outputs are replaced by BCLK and DCLK, respectively.
- DSP PWD Register: Setting the PWD bit generates the NMI input to the DSP. The NMI interrupt service routine is responsible for putting the DSP in the low-power IDLE state.
- In addition to the Version Control Register, Monaco sports a Serial ID register, which provides a unique ID code for each Monaco chip. This register is read only.

7.2.2 Tone Generator and Sigma Delta Tone Interfaces

The buzzer interface(s) are provided by the ABB in a Monaco-based system; therefore, this module is not supported in Monaco.

7.2.3 RTC

No change to this section.

7.2.4 UART / IrDA (GSP A)

7.2.4.1 Introduction

The Monaco device contains 9 instances of the Enhanced Generic Serial Port (EGSP) module, 8 of which are memory mapped onto the PBUS (the 9th EGSP is memory mapped onto the DPBUS, as the BSPORT). Each instance is capable of supporting a number of serial port protocols, from single wire to full asynchronous modem and synchronous interfaces. The table shows the primary differences between each instance, and a typical example function for which each instance could be used.

Instance	Typical Use	Shift Reg. Size	RAM Size	Base Address	I/O Ports	Comms Type	Modem Support?
GSP A	UART / IrDA	16 bits	2*64 bytes	0x803C 0000	5	FD ⁵	Y
GSP B	UART	16 bits	2*64 bytes	0x8040 0000	4	FD	Y
GSP C	DISPLAY	16 bits	2*64 bytes	0x8008 0000	4	FD	N
GSP D ⁶	BMC ⁷	2 * 8 bits	2*64 bytes	0x800C 0000	4	HD * 2	N
GSP E	SYNTH	32 bits	128 bytes	0x8024 0000	6	HD	N
GSP F	SPI / DC	16 bits	128 bytes	0x8024 0000	6	HD	N
GSP G	Pwr Mngmnt	16 bits	128 bytes	0x8024 0000	4	HD	N
GSP H	A SPORT	16 bits	128 bytes	0x8024 0000	4	HD	N
GSP I	B SPORT	32 bits	128 bytes	0x8024 0000	6	HD	N

⁵ FD = Full Duplex, HD = Half Duplex

⁶ This instance comprises two separate half duplex GSP units, that can be used independently or combined into a single full duplex UART.

⁷ BMC = Battery Management Control, typically also used for Accessory interfacing

Table 25: GSP A-I Instance Summary

See the chapter Enhanced Generic Serial Port for more information on the EGSP.

Changes to this section for Medium/Fast IrDA support. Monaco will *not* support Fast IrDA (4Mbps – requires dedicated hardware rather than/in addition to the GSP) unless it is determined to be a customer requirement.

Excerpts from the Hermes specification [8], applicable to Monaco:

The Hermes device supports the Infrared Data Association (IrDa) standard for all the rates mentioned in the IrDa Serial Infrared Physical Layer Specification except the 4.0Mb/s rate. As can be seen from Figure 20, the frames used within the data rates up to 115,2 kb/s are the usual UART type of frame with a start bit, eight data bit, and a stop bit.

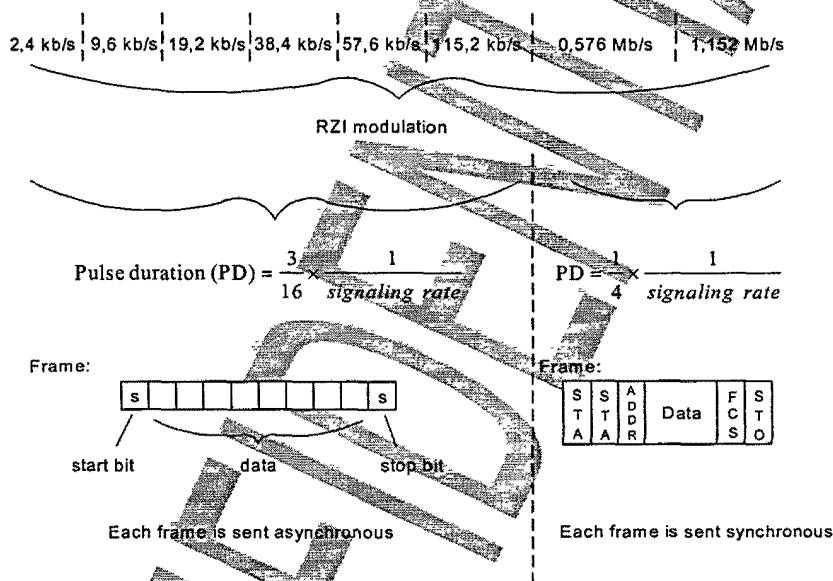


Figure 20: Overview of the various IrDa data rates supported by the Monaco device.

These rates can supported in the Monaco device in the same way as in the Hercules device, i.e. via the GSPs peripherals. The following table shows the minimum, nominal, and maximum pulse duration specifications for the various signalling rates up to 115,2 kb/s.

Signaling Rate	Rate Tolerance % of rate	Minimum Pulse duration	Nominal Pulse duration	Maximum Pulse Duration
2,4 kb/s	+/- 0,87	1,41 us	78,13 us	88,55 us
9,6 kb/s	+/- 0,87	1,41 us	19,53 us	22,13 us
19,2 kb/s	+/- 0,87	1,41 us	9,77 us	11,07 us
38,4 kb/s	+/- 0,87	1,41 us	4,88 us	5,96 us

57,6 kb/s	+/- 0,87	1,41 us	3,26 us	4,34 us
115,2 kb/s	+/- 0,87	1,41 us	1,63 us	2,23 us

Table 26: Minimum, nominal, and maximal pulse duration specs. for the 2,4kb/s to 115,2kb/s data rates.

For the signaling rates 0,576 Mb/s and 1,152Mb/s the frames is transmitted synchronous and introduces bit stuffing after five consecutive binary 1's. The bit stuffing can be supported by the GSPs, as they are in the Hercules device. The following table shows the minimum, nominal, and maximum pulse duration specifications for the signaling rates 0,567 Mb/s 1,152 Mb/s.

Signaling Rate	Rate Tolerance % of rate	Minimum Pulse duration	Nominal Pulse duration	Maximum Pulse Duration
0,576 Mb/s	+/- 0,1	295,2 ns	434,0 ns	520,8 ns
1,152 Mb/s	+/- 0,1	147,6 ns	217,0 ns	260,4 ns

Table 27: Minimum, nominal, and maximal pulse duration specs. for the 0,576 Mb/s and 1,152 Mb/s data rates.

The clock rates for the IrDa protocol are supplied directly by the system clock, by a combination of selectable dividers and the same Peripheral PLL that is used to supply the clock rates for the USB peripheral, or by a dedicated off-chip oscillator.

See the 11.1.7 Peripheral PLL, page 98 for more information on the Peripheral PLL.

If the SysCLK is switching at either the 13MHz or the 26MHz rate, then the rates 2,4 kb/s, 9,6 kb/s, 19,2 kb/s, 38,4 kb/s, 57,6 kb/s and 115,2 kb/s are all supported in the same way as in the Hercules device (from the system clock). The 0,576 Mb/s and 1,152 Mb/s data rates are supported by the peripheral PLL in a multiplication ratio of 4.25 with the following accuracy:

$$1.152\text{Mbits/second: } ((13/3)*4.25)/16 = 1.15104 \text{ (tolerance = 0.083\%)}$$

$$0.576\text{Mbits/second: } ((13/3)*4.25)/32 = 0.57552 \text{ (tolerance = 0.083\%)}$$

If the SysCLK is switching at another frequency (e.g., 19.2MHz for wideband CDMA), then the required tolerances may not be achievable using the on-chip peripheral clock sources. In this case, an off-chip clock oscillator is required.

7.2.5 Debug UART (GSP-B)

No change to this section.

7.2.6 Serial Display Interface (GSP-C)

No change to this section.

7.2.7 Battery Management / Accessory Interface (GSP DA / GSP DB)

No change to this section.

7.2.8 SIM Interface

Highlight that this module already supports Fast SIM. No other change to this section.

7.2.9 USB Interface

Monaco supports a Universal Serial Bus (USB) 1.1 slave interface. This interface is integrated on the PBUS. It is possible to multiplex the USB interface either to predefined pins or to the system connector through the Universal System Connector Interface (USC), in the same manner as all the other supported transmission protocols.

The USB slave interface offers two transfer rates to support full speed devices (12Mbits/s) and low speed devices (1.5Mbits/s). To support those data rates, the USB interface is supplied by a 48MHz clock. Jitter/duty cycle requirements for the 48MHz clock are quite relaxed.

It is important to note that the same Peripheral PLL is used for both the USB slave interface and the IrDa interface; therefore, depending on the modes, the USB and (Medium/Fast) IrDA cannot be active at the same time.

7.2.9.1 USB Endpoints

The USB peripheral supports up to 1 Control, 2 Bulk IN, and 2 Bulk Out endpoints. Isochronous endpoints are not supported. Because the USB hub can request service from any endpoint at any time, each of the endpoints requires its own FIFO. This prevents servicing the USB peripheral with a single DMA channel.

For Monaco, the Control endpoint should be managed directly by the MCU. The System DMA controller can support the Bulk endpoint traffic; however, this may require up to four channels in a full configuration.

7.2.9.2 USB Booting

The Monaco device provides an alternative possibility of boot from the USB instead of the GSPb. This requires detection of whether the synch. byte originates from the GSP or USB which will be done by bit rate detection as part of the boot sequence. No system clock frequency detection will be performed. The host should modify its baud rate to match the baud rate of the target (Monaco).

7.2.9.3 USB Transceivers

The Monaco device integrates the differential transceivers required for the USB interface on chip. Note that the USB electrical specification requires 5V tolerance on these inputs. Also note that the minimum Voh (output voltage high) specification may only be met when the power supply voltage for the USB IO pads is driven to 3.3V.

7.2.9.4 Bus Powered Support

Monaco supports the Bus Powered feature of the USB 1.1. The intent is to support battery charging from the USB cable, at a low maximum current rate (charging current <100mA).

7.2.10 MMC Interface

New section, TBD.

7.2.11 Cipher Coprocessor

New section - Cipher added to this bus to support GPRS packet decoding by MCU system.

In order to ease the MCU processing requirements for GEA and GEA2 ciphering algorithms for GPRS, the Monaco device includes a second cipher module, in the PBUS Man Machine interface (MMI) module. The GPRS ciphering is conducted at the Protocol Stack Layer level.

7.2.12 USC Interface

Changes to this section to route USB, MMC to USC.

7.3 Clock Generation Peripherals

7.3.1 Peripheral PLL

See the 11.1.7 Peripheral PLL, page 98.

7.4 House-Keeping Functions

7.4.1 Watchdog Timer (WDT)

Changes to this module include:

- Counter width is extended, under mode control, to better support non-wireless applications which need longer count values. In wireless mode, the timeout is 8 seconds (18 bits counter width). In non-wireless mode, the timeout is extended up to 512 seconds (24 bits counter width). The selection for the next timeout value should be decided when resetting/refreshing the WDT.
- Early warning is at 3/4 of the full WDT timeout period (6 seconds in wireless mode, or 384 seconds in non-wireless mode)
- The WDT will be inactive:
 - After reset (Therefore, also when the boot ROM is enabled), if the FLASH Boot input (pin) select is cleared. If the FLASH boot input select is set, the WDT comes out of reset enabled (in non-wireless mode; that is, 24-bit count enabled).
 - During emulation of either core, on exit from emulation the WDT counter chain should reset automatically
- Once enabled, the WDT cannot be disabled again unless either processor enters emulation mode. When leaving emulation, the WDT counter is reset, while the rest of the WDT setting is kept.
-

On timeout, the WDT takes Monaco into the hardware reset state.

7.4.2 General Timers

The Monaco device provides the possibility of reading [TBD] the actual values of the general timers for timing test purposes. Otherwise, this module is identical to the Hercules module.

7.4.3 System Configuration Block

This is the control registers that were on the SBUS of Hercules, with changes for the Monaco system.

See SBUS Specification.

7.5 Interrupt Controller

Changes to this section to include new system interrupt inputs, new DSP DMA outputs, new DSP interface (described in DSP sub-system).

7.5.1 Exception Sources

Exceptions are events which are not generally expected to occur. They indicate an error or problem of some type and may require remedial action to be performed or simply provide the opportunity to log the issue for later reference. Exceptions are by their nature most likely to require immediate attention, but little processing time, and as such, are usually set to the highest priority. Although they may be set to any priority level via the configuration registers, they are grouped together into one interrupt group to reflect this.

Table 28 : System Exceptions

Exception	Source of Exception	Comment
System DMA	System DMA Module	Indicates DMA malfunction such as overrun
DSP DMA	DSP DMA Module	Indicates DMA malfunction such as overrun
WDT	Watchdog Timer	Indicates early warning from the WDT
TEP	TEP Module	Read TEP registers to identify cause
SIM	SIM Module	Read SIM registers to identify cause
SWI Overflow Exception	Interrupt Controller	SWI Overflow
SWI Underrun Exception	Interrupt Controller	SWI Underflow – DSP Only
GPIO_EDGE	MMI GPIO Edge Sensitivity	Allows an external device to trigger an exception rather than a lower priority IRQ
GPIO_LEVEL	MMI GPIO Level Sensitivity	
USCSwitch	USC Synch. Serial port	Indicates a RX overrun condition
DSP Double Fault	DSP module	Allows MCU to detect DSP double-fault condition

The DSP has its own dedicated interrupt controller, internal to the core. This interrupt controller directly handles a number of exception cases which can be generated by the DSP (e.g., mis-aligned memory references, bus errors, CPLB miss).

7.5.2 Interrupt Sources

Interrupt sources include the system timing and scheduling events from the TEP, those from the internal processing blocks and on board peripheral devices such as the SIM interface and GSP modules and the external interrupt pins on the Monaco device via GPIO. Although generally all interrupt sources are

available to both cores, in some instances a given source is only routed to one core. The table below summarizes all of the interrupt sources in the Monaco system.

CONFIDENTIAL

Interrupt Source	No of lines	Cause of Interrupt	Comment
SYSDMA[15:0]	16	System DMA Channel no 0 - 15 complete	Indicates completion of System DMA transfer on a given channel
DSPDMA[15:0]	16	DSP DMA Channel no 0 - 15 complete	Indicates completion of DSP DMA transfer on a given channel
GPIO	8	Programmable General Purpose IO event	Allows external sources to trigger IRQ on either edge or level sensitive inputs. See MMI chapter.
MCU Cipher Done	1	MCU Cipher Coprocessor	Indicates cipher completion
DSP Cipher Done	1	DSP Cipher Coprocessor	Indicates cipher completion
Viterbi	2	DSP Viterbi Coprocessor	Input FIFO (IINT) and Output FIFO (OINT)
GSP-a Rx, Tx, CP	3	Rx Full, Tx Empty or CP event	UART 1 - Primarily for Data Services
GSP-b Rx, Tx, CP	3	Rx Full, Tx Empty or CP event	UART 2 - Primarily for Debug
GSP-c Rx, Tx, CP	3	Rx Full, Tx Empty or CP event	Primarily for Display interface
GSP-e Rx, Tx, CP	3	Rx Full, Tx Empty or CP event	Primarily for Synthesizer interface
GSP-da, db Rx, Tx, CP	6	Rx Full, Tx Empty or CP event (x 2)	Primarily for Battery Management interface. Can be configured as two ½ duplex UARTS
GSP-f Rx, Tx, CP	3	Rx Full, Tx Empty or CP event	UART 1 - Primarily for Data Services
GSP-g Rx, Tx, CP	3	Rx Full, Tx Empty or CP event	UART 2 - Primarily for Debug
GSP-h Rx, Tx, CP	3	Rx Full, Tx Empty or CP event	Primarily for Display interface
GSP-i Rx, Tx, CP	3	Rx Full, Tx Empty or CP event	Primarily for Synthesizer interface
USB	2	Rx Full, Tx Empty	
DSP SWI[15 : 0]	16	Software Interrupt [15:0]	Asserted by program code in TEP or either core
MCU SWI [15 : 0]	16	Software Interrupt [31:16]	Asserted by program code in TEP or either core
DMA SWI [15 : 0]	16	Software Interrupt [47:32]	Asserted by program code in TEP or either core to trigger DMA channels 0 to 15.
SIM	1	Rx Full or Tx Empty	
Keyboard event	1	Indicates key press event	Only activated when all KEYPADCOL outputs are all low
RTC	1	Real Time Clock alarm trigger	Provides an alarm function
System PLL Clock Locked	1	System PLL and clock circuit	Indicates the system clock is stable and phase locked
Peripheral PLL Clock Locked	1	Peripheral PLL and clock circuit	Indicates the peripheral clock is stable and phase locked
SPI	1	SPI Service Required	
DSPASpt Rx, Tx	2	ASport Rx full, Tx empty	Dedicated VBC control port - DSP IRQ's
FLAGS[2:0]	3	DSP signaling flags	Asserted by the DSP
UscSwitch Rx, Tx	2	USC Synchron. Serial Port	RxFull and TxEmpty on internal serial port
MMC	1		Multi-Media Card support
Timer	1	TEP Generic Timer	One of the generic timers has expired
A-Sport Rx, Tx	2	Rx Full or Tx Empty	Audio data service
B-Sport Rx, Tx	2	Rx Full or Tx Empty	Radio data samples
C-Sport Rx, Tx	2	Rx Full or Tx Empty	ABB Control interface
EAPI	1	EAPI write to interrupt bit	External Applications Interface data service
ECI	1	ECI interrupt response	External Coprocessor Interface exception

Table 29: System Interrupts

The conditions under which each module interrupt is generated are described within the corresponding module chapters.

7.5.3 Processor Interrupt Resources

7.5.3.1 MCU

The MCU interrupts are routed to the core via FIQ only, such that a central interrupt service routine (ISR) is given access to the private FIQ register set of the MCU which reduces latency.

7.5.3.2 DSP

Emulation, Reset, NMI, and GP14, GP15 directly connected on the DSP. All system interrupts from DSP IRQ routed to GP7.

7.5.3.3 DMA

The DMA interrupts are passed on to the DMA controller, which prioritizes them according to channel number. There must only be one interrupt source for each DMA channel.

7.5.4 Interrupt Controller Operation

7.5.4.1 Block Diagram

The Interrupt Controller contains three interrupt control units, each of which is responsible for masking and routing all the incoming events to either the DSP, MCU and DMA controller. The operation of each unit is designed to be as near identical as possible, though the DSP and MCU units are also responsible for prioritization of all interrupt sources. However, the interrupt handling performed by each processor differs so the input paths to each controller are unique.

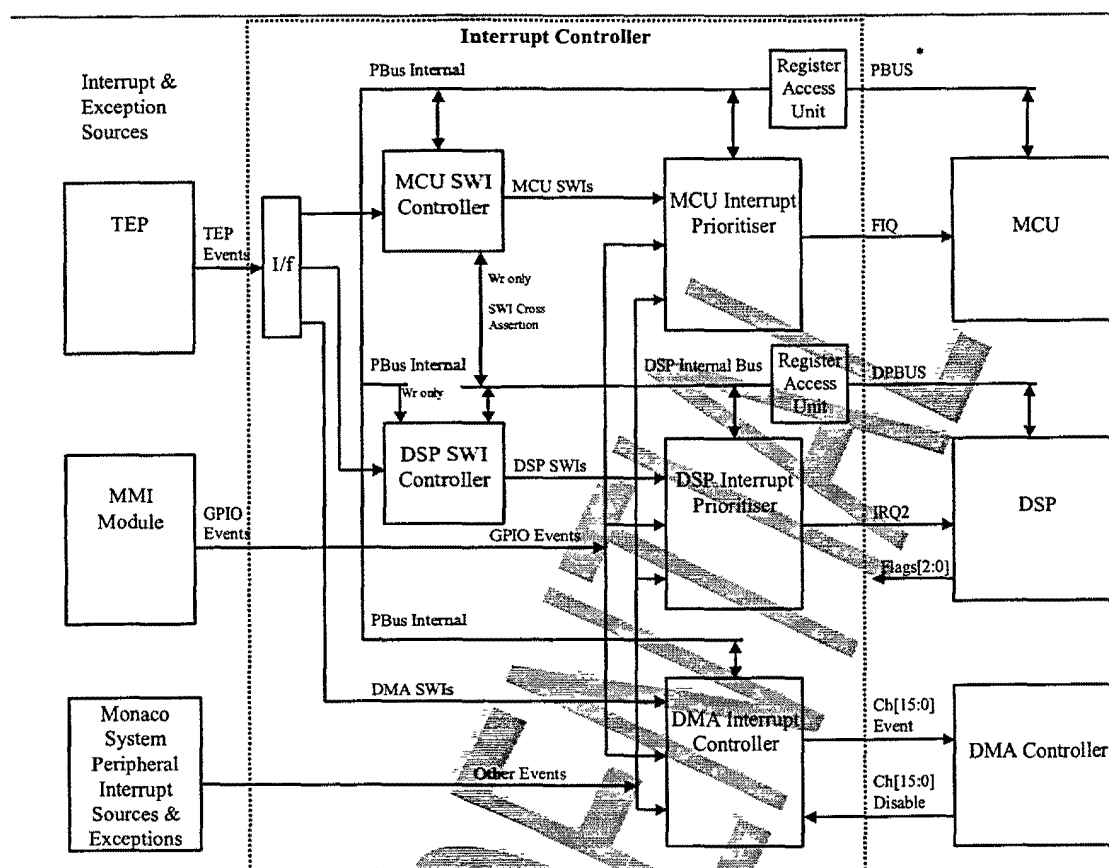


Figure 21: Block Diagram

Each of the major blocks are described below. In contrast with the system peripherals which contain their own local interrupt generation and latching and status logic, the SWI controllers contain all the necessary functionality to support this interrupt source.

Interrupts are routed through each of the three controllers, where they are masked and grouped. Grouping provides a trade off by reducing the complexity and simplifying operation of the interrupt controller. The design allows most interrupt sources to be routed to either MCU, DSP, DMA, or multiple destinations.

The SWI controllers utilize a clock (see later) and so when clocks are disabled SWIs will not be propagated, but in this case, there will be no processors active and able to generate any SWIs. The rest of the system is asynchronous and un-clocked so as to allow interrupts other than SWIs to be fully processed even when the Monaco device is in low power sleep mode (when all clocks are disabled). The interrupt subsystem will wake the target processor when an appropriate event occurs with a correctly prioritized and identified interrupt.

All interrupt sources, including GPIOs, appear as asynchronous interrupts to the IRQ controller. The MCU and DSP IRQ outputs are synchronized before being used by the cores.

The Register Access Units (RAU) - one for each processor core - provide a high degree of flexibility when reading or writing to and from registers. By allowing individual register bits to be read / written and converting between bitwise data and priority encoded decimal, they allow the Interrupt Service Routines to be optimized for speed and latency. The system diagram shows how each RAU interfaces between the internal register data I/O busses and each of the two external busses.

There are three sets of registers - one for each processing core. While the MCU and DMA register sets are all resident in PBUS address space, the DSP register set is accessed via the DPBUS only. This provides the lowest possible latency for the DSP when accessing its interrupt controller logic by avoiding the PBUS access latency and slower PBUS clock speeds.

The two bus domains PBUS and DPBUS are separate except for one common link point which is the 'SWI cross assertion' register where one processor core writes the register as for any other, but this causes an event to be latched in the other processor's SWI status registers.

7.5.4.2 Interrupt Sources and Groups

Each controller is built around a crossbar or selector switch and priority encoder which maps interrupt sources to predefined priority id numbers and allows through only the highest priority active source from all the unique priority assignable sources. The currently active highest priority interrupt is presented to the respective core via the Current Interrupt Source Register (CIR) together with assertion of the appropriate IRQ line. The inputs to the crossbar switches are formed from groups of related single interrupt lines and are referred to as interrupt groups. Each grouped set is treated as one unique priority assignable source by the priority encoder and competes for service with the other groups. Within a group, the priority is equal and since the Interrupt Service Routine (ISR) will need to read and mask the status register for that group - dealing with each set bit in turn - it is up to the software to prioritize within a group. However, if a group status register is read via the RAU priority encoder then a fixed priority scheme will be imposed where the least significant bit ordered member of the group is the highest priority; i.e. the highest in the lists in section 7.5.7.

The groups are as follows:

Group	Name	Typical use / members
0	Exceptions	Groups all system errors - Typically highest priority
1	General A	DMA / GPIO & GSPa if this not via DMA
2	MMI / Co-Pro	MCU: SIM, Timer, Buzzer, etc. DSP: Viterbi or Cipher
3	General B	DMA / GPIO & GSPe if this not via DMA
4	GSP	All GSP interrupts, typically for handling GSP CP interrupts
5	General C	DMA / GPIO & GSPb if this not via DMA
6	MMI / Co-Pro	As 2 above, but in reverse priority order
7	General D	DMA / GPIO, GSPc and USC Switch - Typically miscellaneous low priority sources
8 - 15	SWIA - H	Software Interrupts grouped in pairs

Table 30: Interrupt Group Definitions

- Group 0 is generally expected to be assigned the highest priority and thus contains all the system exceptions.
- Groups 1, 3, 5 and 7 equally share the non-dedicated or general purpose interrupts for DMA channels, GPIO IRQs and CBA functions. This allows for relative priorities to be assigned for

example between the DMA channel interrupts. If not all DMA channels are in use simultaneously, the priorities for these groups can also be set with respect to the GSPs, flags or other sources that are covered by these groups.

- Groups 2 and 6 are identical but reversed to facilitate partitioning of interrupts between these two groups, for example; SIM and TIMER in group 2 and Buzzer and Keyboard in group 6. By reversing the order in group 6, the interrupts can be arranged at lower or higher priorities with respect to the other group members by choosing to enable the interrupt in one group rather than the other.

7.5.4.3 Interrupt Source Signaling

All the interrupt sources are filtered by associated mask registers that ensure each core only receives interrupts intended for it. The status of each group is readable by either processor via PBUS or DSP IO BUS to provide indication of the actual source of an interrupt within a group.

7.5.4.4 Prioritization

Both MCU and DSP controllers are identical internally and provide 16 levels of priority for the 16 possible interrupt groups. Since only one interrupt output is generated, the service routine must be re-entrant as all interrupts will be handled via one ISR which must 'farm' out the task to one of a number of sub-ISRs. Common to the MCU and DSP are the current interrupt group registers which uniquely identify which of the 16 interrupt groups have been asserted. When read with the appropriate RAU mode the register will return successive interrupt priority numbers in order with the highest priority being 'delivered' first. In some cases the group may consist of only one unique interrupt and so reading the group status is not necessary.

7.5.4.5 GPIO Interrupts

The GPIO interrupts are generated within the GPIO interface section of the MMI module. 4 GPIOs are able to trigger on edge triggered events (register selectable between positive or negative edge-triggered, or dual edge) and 4 are level sensitive (register selectable between active high or active low).

The GPIO interrupts are grouped into GPIOA and GPIOB and in common with other interrupts only synchronized within the appropriate core. Level sensitive inputs are routed directly to the core whereas edge sensitive inputs are latched in the MMI module first and then routed to the level sensitive processor inputs. This means that for edge-sensitive GPIO interrupts the ISR will have to clear the MMI GPIRQ_ST interrupt status register. If external GPIRQ level sensitive sources generate glitches, then the MMI module should be configured to sample the GPIRQ input first.

In addition to the external GPIO derived interrupts, the interrupt system also features 4 'CBA' interrupts. These are internal general purpose active high level sensitive interrupt lines which are intended for use with any on chip Cell Based Array (CBA) area such as might be added by a specific application or customer.

7.5.4.6 Interrupt Overruns

Interrupt overruns occur when a second interrupt occurs before the first interrupt has been successfully serviced. Such a situation is a system malfunction and will result in lost data, unacceptable latency or unstable operation, e.g. re-entrancy to non re-entrant code.

The following table summarizes how overruns are dealt within the system:

Exceptions	No protection - ISR must be short and low latency.
DMA	No protection - DMA channel completion for a given channel cannot be re-generated until software has re-configured that channel.
GPIO & CBA	No protection - must be dealt with by user, with exception signaling via a GPIO interrupt.
Cipher / Viterbi	No protection - to be handled within CoProcessor.
GSPS / UARTS	No protection - All UARTs / GSPs including those nominally configured as dallas, display or synth provide data overrun monitoring where a second interrupt is not generated on Rx as the status bit is still set signifying that the data has not yet been read. The exception is signaled via the CP interrupt when data overruns.
SWIs	Handled by the SWI counters and the status latches. The status latch will generate an exception if a set pulse occurs while it is already set.
DSP Flags	No protection - These are used under software control for re-prioritizing Sport interrupts and so may be trapped in software.
Keyboard	No protection - First keypress generates IRQ which is held until cleared in software, though keypad can be scanned at any time.
RTC	No protection - Alarm function must be re-configured before a second interrupt is possible.

CONFIDENTIAL

7.5.5 Interrupt Control & Prioritization

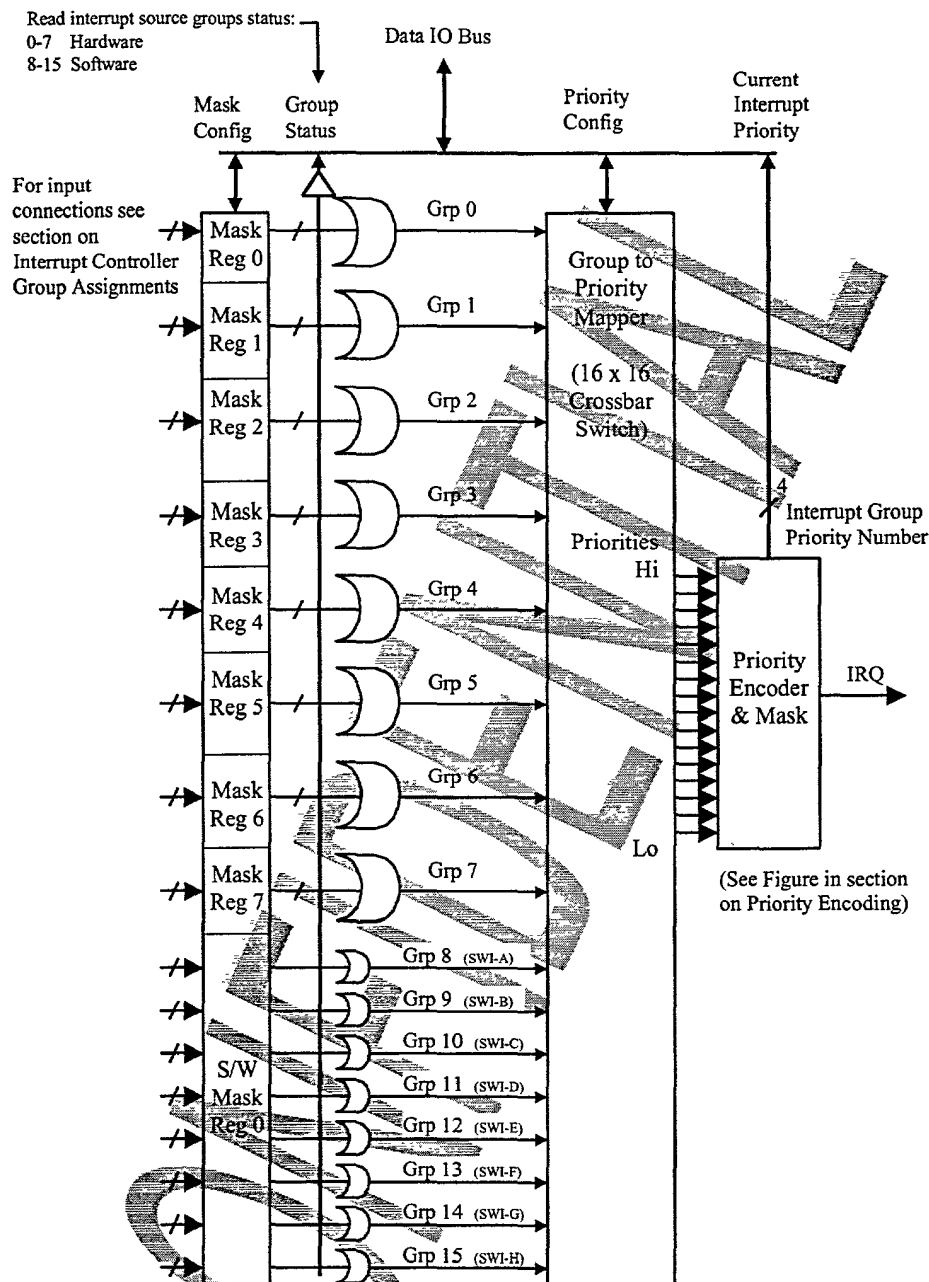


Figure 22: Interrupt prioritizer

At the heart of the system are two interrupt controllers - one for the MCU and one for the DSP - which assign priorities to interrupt sources and present them to the processors in order. Each differ only in the connections at their inputs. The DMA interrupt controller is separate and is described later.

With reference to Figure 22, the individual interrupts are maskable via the mask registers and then grouped by the OR gates. The crossbar switch routes each of these 16 interrupt groups to one of 16 output priority levels according to a set of configuration registers. The crossbar outputs are readable directly by the processor bus bitwise, or via the RAU priority encoder which is how the system is designed to operate. In this case, the crossbar output priorities are hardwired into the priority encoder

Prior to exit from the ISR, the processor must write the same CI Register value to the AI Register, but this time using the RAU write mode 3 function whereby the top bit of the word - 0 in this case - is written into the bit position specified by the 4 lsb's - the interrupt group priority number. This clears that bit in the AI Register and the priority mask updates allowing the appropriate priority levels through.

All interrupts are level sensitive and are simply masked in or out by the AI register and priority mask logic. When the CI Register is read the priority mask will ensure that the IRQ line is disabled until a higher priority interrupt occurs.

7.5.6 DMA Interrupt Controller

The DMA interrupt controller uses a normal selector to route the outputs uniquely from one of the many DMA event input sources to one of the 16 DMA channels. This is sufficient due to the dedicated nature of DMA and lack of any 'intelligence' i.e. software in the DMA controller.

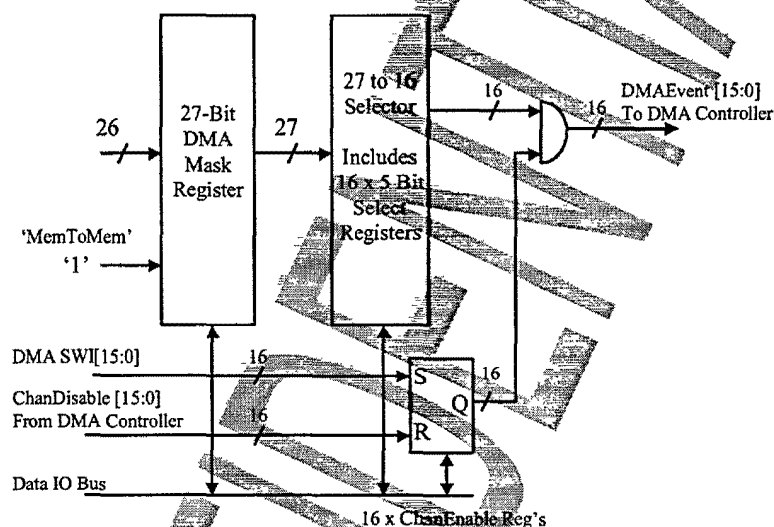


Figure 24: DMA Interrupt Controller

The DMA controller responds to an event on one of the 16 event lines by carrying out a DMA transfer as previously configured. When complete, it asserts the appropriate 'Disable' line to halt that channel while awaiting indication to proceed with the next transfer. The interrupt controller will pass all active events to the DMA controller which is responsible for carrying out its own prioritization and responding appropriately.

Although DMA events can clearly be initiated by more than one source (in the same way as for the MCU and DSP) the DMA controller has no way of knowing which source caused which DMA channel event. It is therefore required that only one source is mapped onto a particular channel at a time. SWIs are used as enables, but can also be used to start a transfer by selecting the '1' input to the selector, this is designed to support inter-memory transfers and is therefore termed the 'MemToMem' bit.

7.5.7 Interrupt Controller Group Assignments

The additional features provided by the Monaco device requires some additional interrupts to be handled by the Interrupt Controller. The additional interrupt sources to be controlled by the respective masters are as follows.

- USB: Requires two interrupts (Rx/Tx)
- SPI: Requires one interrupt for the slave mode.
- MMC
- TEP: Snapshot Ready, Databus Ready
- ECI
- EAPI
- DSP DMA: 16 channels
- 2nd Ciphering Unit: Requires one interrupt for indication of invalid instruction and halt indication.

These interrupt are placed in the interrupt groups as follows.

MCU Interrupt Controller

Priority	Groups							
	0	1	2	3	4	5	6	7
HI: 0	Null Condition							
1	GpioEdgeE	GspARxI	SimI	GspERxI	GspARxI	GspBRxI	GspDbRxI	GspCRxI
2	GpioLevelE	GspATxI	MCUCSportTxI	GspETxI	GspATxI	GspBTxI	GspDbTxI	GspCTxI
3	TEPE	GspACpI	MCUCSportRxI	GspECpI	GspBRxI	GspBCpI	GspDbCpI	GspCCpI
4	McuSwiOverE	GpioI (0)	TimerI	GpioI (1)	GspBTxI	GpioI (2)	GspDaRxI	GpioI (3)
5	WdtE	GpioI (4)	--	GpioI (5)	GspCRxI	GpioI (6)	GspDaTxI	GpioI (7)
6	SysDmaE	SysDmaI (0)	KeyboardI	SysDmaI (4)	GspCTxI	SysDmaI (8)	GspDaCpI	SysDmaI (12)
7	--	SysDmaI (1)	Sytem PLL	SysDmaI (5)	GspERxI	SysDmaI (9)	RtcI	SysDmaI (13)
8	SimE	SysDmaI (2)	Cipher (GPRS)	SysDmaI (6)	GspETxI	SysDmaI (10)	--	SysDmaI (14)
9	UscSwitchE	SysDmaI (3)	RtcI	SysDmaI (7)	GspACpI	SysDmaI (11)	--	SysDmaI (15)
10	--	CbaI (0)	GspDaRxI	--	GspBCpI	--	KeyboardI	--
11	EAPE	DspFlagI (0)	GspDaTxI	DspFlagI (1)	GspCCpI	DspFlagI (2)	--	DspFlagI (0)
12	USBE	TimerI	GspDaCpI	MCUCSportTxI	GspDaCpI	MCUCSportRxI	TimerI	DspFlagI (1)
13	--	MMC	DSPdm (15)	USBRxTx	GspDbCpI	UscSwitchRxI	MCUCSportRxI	DspFlagI (2)
14	--	--	ASPORT	USBTx	GspECpI	UscSwitchTxI	MCUCSportTxI	UscSwitchRxI
LO:15	--	--	--	DbusRdySnap	SnapShotRdy	SPI	SimI	UscSwitchTxI
Groups								
	8	9	10	11	12	13	14	15
	McuSwi (0)	McuSwi (2)	McuSwi (4)	McuSwi (6)	McuSwi (8)	McuSwi (10)	McuSwi (12)	McuSwi (14)
	McuSwi (1)	McuSwi (3)	McuSwi (5)	McuSwi (7)	McuSwi (9)	McuSwi (11)	McuSwi (13)	McuSwi (15)

DSP Interrupt Controller

Priority	Groups							
	0	1	2	3	4	5	6	7
HI: 0	Null Condition							
1	GpioEdgeE	GspARxI	SimI	GspERxI	GspARxI	GspBRxI	GspDbRxI	GspCRxI
2	GpioLevelE	GspATxI	DSPCSportTxI	GspETxI	GspATxI	GspBTxI	GspDbTxI	GspCTxI
3	TEPE	GspACpI	DSPCSportRxI	GspECpI	GspBRxI	GspBCpI	GspDbCpI	GspCCpI
4	DspSwiOverE	GpioI (0)	TimerI	GpioI (1)	GspBTxI	GpioI (2)	GspDaRxI	GpioI (3)
5	DspSwiUnderE	GpioI (4)	--	GpioI (5)	GspCRxI	GpioI (6)	GspDaTxI	GpioI (7)
6	WdtE	DSPDmaI (0)	Viterbi IINT	DSPDmaI (4)	GspCTxI	DSPDmaI (8)	GspDaCpI	DSPDmaI (12)
7	DSPDmaE	DSPDmaI (1)	Viterbi OINT	DSPDmaI (5)	GspERxI	DSPDmaI (9)	RtcI	DSPDmaI (13)
8	SimE	DSPDmaI (2)	Cipher (GSM)	DSPDmaI (6)	GspETxI	DSPDmaI (10)	Cipher (GSM)	DSPDmaI (14)
9	UscSwitchE	DSPDmaI (3)	RtcI	DSPDmaI (7)	GspACpI	DSPDmaI (11)	Viterbi OINT	DSPDmaI (15)
10	--	CbaI (0)	ECI	--	GspBCpI	--	Viterbi IINT	CbaI (3)
11	EAPE	DspFlagI (0)	GspDaRxI	DspFlagI (1)	GspCCpI	DspFlagI (2)	--	DspFlagI (0)
12	USBE	TimerI	GspDaRxI	DSPCSportTxI	GspDaCpI	DSPCSportRxI	TimerI	DspFlagI (1)
13	--	MMC	GspDaRxI	USB	GspDbCpI	UscSwitchRxI	DSPCSportRxI	DspFlagI (2)
14	--	BSPORT TX	ASPORT	USBTx	GspECpI	UscSwitchTxI	DSPCSportTxI	UscSwitchRxI
LO:15	--	BSPORT RX	--	DbusRdySnap	SnapShotRdy	SPI	SimI	UscSwitchTxI
Groups								
	8	9	10	11	12	13	14	15
	DspSwi (0)	DspSwi (2)	DspSwi (4)	DspSwi (6)	DspSwi (8)	DspSwi (10)	DspSwi (12)	DspSwi (14)
	DspSwi (1)	DspSwi (3)	DspSwi (5)	DspSwi (7)	DspSwi (9)	DspSwi (11)	DspSwi (13)	DspSwi (15)

Table 31 : Interrupt Group Assignments for the MCU and DSP.

7.5.8 System and DSP DMA Interrupt Controllers

The Monaco device integrates a new 16-channel DMA controller in the DSP Sub-system. The interrupt controller for this DMA is identical to that for the System DMA.

SELECTOR INPUTS							
0 '1'	1 UscSwitchTxI	2 UscSwitchRxI	3 —	4 MCUCsportTx I	5 MCUCsportRx I	6 DspFlagI[0]	7 DspFlagI[1]
8 SimI	9 GpioIrqI[4]	10 GpioIrqI[5]	11 GpioIrqI[6]	12 GpioIrqI[7]	13 MCUASport	14	15 GspETxI
16 GspERxI	17 GspCTxI	18 GspCRxI	19 GspDbTxI	20 GspDbRxI	21 GspDaTxI	22 GspDaRxI	23 GspBTxI
24 GspBRxI	25 GspATxI	26 GspARxI	27 USBRx	28 USBTx	DSPBSportRx	DSPBSportT	—
SOFTWARE INTERRUPT INPUTS							
DmaSwi[15:0]							

Table 32: Interrupt sources for the DMA.

This table lists the possible DMA event sources together with the event ID number to be programmed into the respective channel event select register. The SWIs control directly the 'ChanEnable' function. It should be noted that the signals in the table appended with an 'I' are ordinary interrupt sources. All signals appended with an 'E' are exceptions and therefore signify unexpected behavior.

Opens:

- *The DMA interrupt sources table needs to be updated for the TEP interrupt sources.*
- *System and DSP DMA interrupt sources are not identical, need separate tables.*

7.6 System DMA

The only change to this section is the addition of the Chaining Mode.

7.6.1 Chaining DMA channels

Chaining DMA channels implies the transfer-completion interrupt from one channel initiates another DMA channel. The particular usage of this is for implementing an A/B buffer system, e.g. for servicing a serial communication channel like an UART.

As an example, say DMA channel 4 is setup to serve a UART reception stream. Once the setup number of data has been received the DMA channel will generate an interrupt to the MCU/DSP system. At the same time another DMA channel takes over the service of the UART stream until its setup number of data has been received. After this, the task can either be halted or by issuing an interrupt, the DMA channel 4 can take over again. By proper setup of the memory areas, the two DMA channels writings form the A- and B-buffer system. The benefits are the usual for A/B buffer systems, relaxed timing constraints for post processing data and decreased risks for simultaneously working on same set of data. The drawbacks are

the increased amount of data memory and the use of two DMA channels for servicing one data stream.

7.6.1.1 Functionality

At least two approaches exist,

- Use two DMA channels with a few modifications concerning changing interrupt assignment triggered by another signal.
- Add a set of shadow configurations registers for each DMA channel supporting the chaining feature. The two sets of registers are then swapped at the transfer-completion interrupt event.

The cost of the latter approach will be comparable to adding a DMA channel. For this reason the former seems more feasible when combined with addition of some more DMA channels for chaining. In this way the chaining mode will not interfere with already existing assignments of channels.

The idea is that the original 16 channels will be kept as is, and an additional 16 channels will be added for chaining purposes only. Thus, for every channel it is possible to enable (or disable) the chaining mode by use of a control bit in a **ChainingReg** register. When enabled, the “shadowed” chaining channel is available, so that when for example channel 7s chaining mode is enabled, the chaining channel 7 is available. It should be noted that it is possible to set up the chaining channel even when they are not enabled, but it will not be possible to use these channels as ordinary DMA channels. An example: if the DMA channel 7 has been configured for its chaining mode, the bit7 in the **ChainingReg** will be checked once the IRQOut fires, and if set, the seventh bit in the **ChanEnable** register [1] remains set. The purpose of this operation is it to put the ordinary channel x (seven in this example) on hold while letting the chained channel x taking over serving the interrupt source – and once the chained channels IRQOut fires, the channels are interchanged again. If the bit for chaining channel 7 in the **ChainingReg** is still set, the ordinary channel 7 is taking over again and the process repeats itself. But if the bit for chaining channel 7 has been cleared, the DMA will perform the next task in the priority list. No matter if the task just finished were performed by the ordinary channel or the chained channel, the default channel will be the ordinary channel once a task has completed. By proper setting up the two DMA channels pointing to different memory areas, this usage implements an A/B buffer system.

In this way both the ordinary channel and the chained channel will use the same interrupt signal and thus, there will be no need for changing the DMA interrupt controller. This also means that both the ordinary channel and its chained channel will have the same level of priority.

7.6.1.2 Channel Programming

The programming of the channels are done in a similar way no matter if the channel in question is an ordinary channel (i.e. from channel zero to 15) or a chained channel. However, a write to the currently active channel should only modify the channel control registers and not transfer source and destination address since this would result in serious transfer errors.

The programming of a channel is done via the PBUS and involves writing the channel configuration data to the registers that specifies the channel in question. The DMA controller is placed in the Hermes peripheral memory map at location 0x801C 0000 to 0x801C FFFF. In this address range the lower 10 bit is given by the PBUS-arbiter to the DMA controller where they are used internally to differentiate between the various registers, channels and the channels control registers. The address PAdr[9:0] is shown in Figure 25.

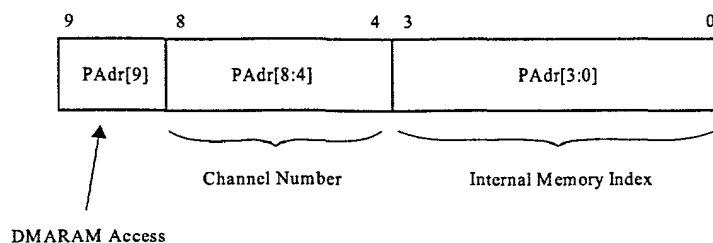


Figure 25: The lower 10 address bit (PAdr[9:0]) to the DMA controller from the PBUS arbiter.

The DMARAM access bit (PAdr[9]) differentiates between making an access to the internal DMA data registers and the channel specific registers. If PAdr[9] = 1'b1 the internal DMA data control registers are chosen. The Channel number (PAdr[8:4]) yields the vertical index to the DMARAM i.e. the channel number and the Internal Memory Index (PAdr[3:0]) yields the horizontal index of the channel i.e. the various setup/control registers of the channel. The following table shows the address map internal to the DMA controller.

Address	Register	R/W	Description
0x801C 0200	Reset	W	Any data value resets the DMA state machine.
0x801C 0202	IRQOut[15:0]	R/W	Each bit corresponds to a channel output interrupt when set high. A DMA channel generates an interrupt on completion of a block transfer.
0x801C 0204	IRQOutClr[15:0]	W	Allows bit-wise clearing of IRQOut bits. If corresponding data bit is set high the bit is cleared.
0x801C 0206	SysEx[0]	R/W	When asserted it indicates a DMA system exception.
0x801C 0208	{MasterIdle, ActivChan[4:0]}	R	MasterIdle is high when statemachine no current channel transfer is made. The ActivChan yields the currently active channel code (0-15). When no channel is active, the value 16 is read.
0x801C 020A	ChainingReg[15:0]	R/W	The corresponding ChainingReg bit must be set to enable chainingmode for the channel. Bit 0 corresponds to channel 0 and bit 15 corresponds to channel 15.
0x801C 0000	Chan0SrcLo[15:0]	R/W	Channel 0 source address lower halfword.
0x801C 0002	Chan0SrcHi[12:0]	R/W	Channel 0 source address upper halfword
0x801C 0004	Chan0DestLo[15:0]	R/W	Channel 0 dest addr lower halfword
0x801C 0006	Chan0DestHi[12:0]	R/W	Channel 0 dest addr upper halfword
0x801C 0008	Chan0Config[6:0]	R/W	Channel 0 configuration

0x801C 000A	Chan0NotifyAddrMem[14:0]	R/W	Channel 0 Notify address for Linear
0x801C 000C	Chan0NotifyAddrIO[10:0]	R/W	Channel 0 Notify address for Circular
0x801C 000E	Chan0N[3:0]	R/W	Channel 0 Circular buffer size (2^N bytes)
0x801C 0010	Chan1SrcLo[15:0]	R/W	Channel 1 source address lower halfword.
0x801C 0012	Chan1SrcHi[12:0]	R/W	Channel 1 source address upper halfword
0x801C 0014	Chan1DestLo[15:0]	R/W	Channel 1 dest addr lower halfword
0x801C 0016	Chan1DestHi[12:0]	R/W	Channel 1 dest addr upper halfword
0x801C 0018	Chan1Config[6:0]	R/W	Channel 1 configuration
0x801C 001A	Chan1NotifyAddrMem[14:0]	R/W	Channel 1 Notify address for Linear
0x801C 001C	Chan1NotifyAddrIO[10:0]	R/W	Channel 1 Notify address for Circular
0x801C 001E	Chan1N[3:0]	R/W	Channel 1 Circular buffer size (2^N bytes)
...the registers for channel 2 to 30 continues as for channel 0 and 1..			
0x801C 01F0	Chan31SrcLo[15:0]	R/W	Channel 31 source address lower halfword.
0x801C 0122	Chan31SrcHi[12:0]	R/W	Channel 31 source address upper halfword
0x801C 0124	Chan31DestLo[15:0]	R/W	Channel 31 dest addr lower halfword
0x801C 0126	Chan31DestHi[12:0]	R/W	Channel 31 dest addr upper halfword
0x801C 0128	Chan31Config[6:0]	R/W	Channel 31 configuration
0x801C 012A	Chan31NotifyAddrMem[14:0]	R/W	Channel 31 Notify address for Linear
0x801C 012C	Chan31NotifyAddrIO[10:0]	R/W	Channel 31 Notify address for Circular
0x801C 012E	Chan31N[3:0]	R/W	Channel 31 Circular buffer size (2^N bytes)

Table 33 DMA Internal Address Map

The DMA channel registers can be modified at any time, including that of the currently active channel, which will take immediate effect. It is important that only the NotifyAddr (or the TranOneStop) register is written to in this case, to avoid data corruption. The active channel register data is written back to the RAM when a higher priority channel becomes active.

The channel data is stored in the internal DMA 32x80bit memory where each vertical entry looks like shown in Figure 26. An important detail in this figure is that the ChanSrcHi and ChanDestHi registers are only 13bit wide. It is shown how the PBUS data is latched into these registers where PDW[10:8] is discarded. It should therefore be noted that whenever a transfer is being done, the 32 bit source or destination address from the DMA controller (Adr[31:0]) for accessing the Pbus, Ebus and Rbus will always have ADR[26:24] = 00.

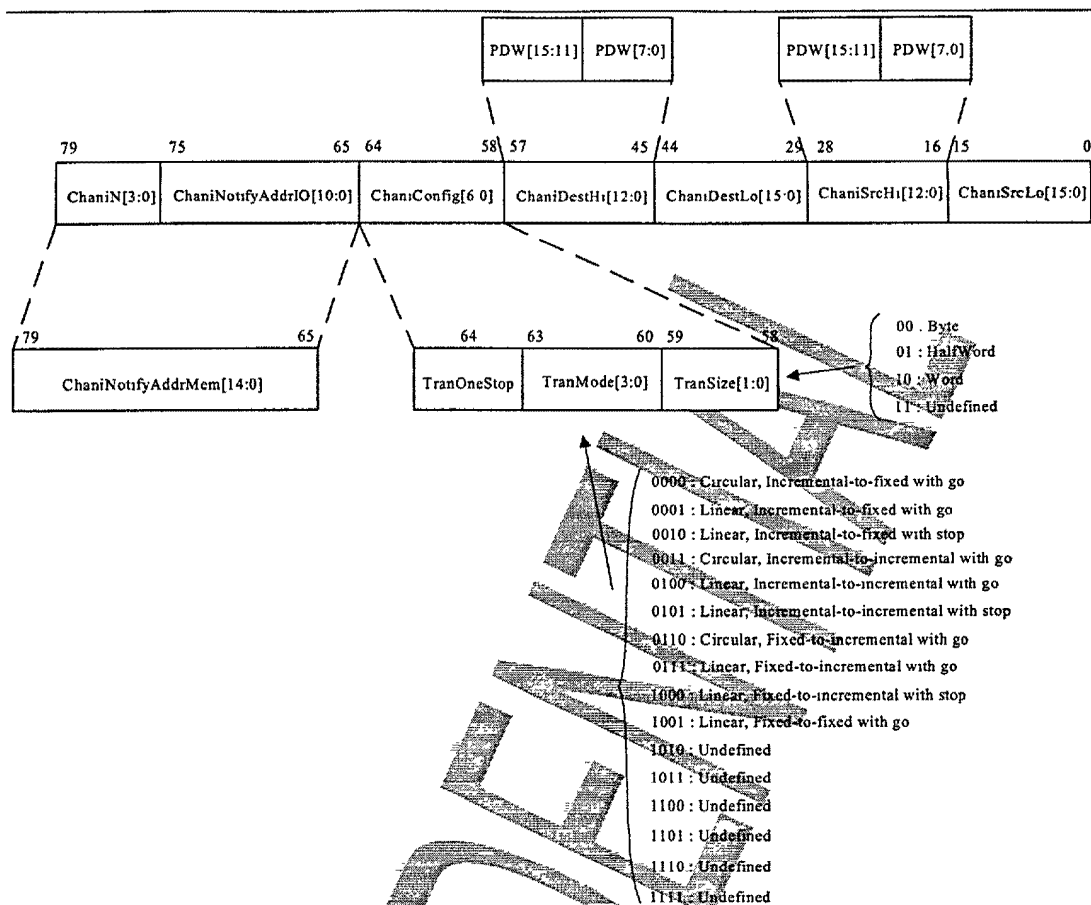


Figure 26: Vertical entry for channel i in the internal DMA memory.

7.6.1.3 Using the Chaining Mode

When the chaining mode is used, the first thing to ensure is that the corresponding bit in the ChainingReg is set. Once this is done, and the channel (plus the corresponding chained channel) have been programmed, the setup and transfer runs in the same way as with a normal DMA channel. It is first when the normal channel transmission finishes that the DMA starts to behave differently.

Once the normal channel transmission finishes, IRQOut is set (ChanDisable is not!) and the chained channel takes over. In this way the interrupt for the master who initiated the transfer is notified, but the ChanDisable bit that corresponds to the current channel is not cleared. At this time the ChainingReg bit is cleared automatically and the chained channel will take over. Seen from the outside, this change to the chaining channel is transparent since it will look like the normal channel is just performing a new transfer. When the chaining transmission finishes, the corresponding bit in both IRQOut and ChanDisable will be set.

After the IRQOut is spotted from the normal channel (i.e. the first IRQOut for that channel), a possible usage could be for the master to reprogram the chaining channel data and set the ChainingReg bit again before the first chaining transmission ends. If this is done, the DMA will regard this second chaining transmission in the same way as the first chaining transmission and

once the first chained transmission ends, the IRQOut will be set but the ChanDisable will not. This could go on and on and the ChanDisable will only be set by the very last transmission.

It is possible to reprogram the chaining channel even though it is executing because once the chaining channel transfer has begun, it will also be treated internally in the DMA as if it was the normal channel and not the chaining channel performing the transfer. This means that if a chaining transfer is being conducted and it is interrupted by a transfer done on a higher priority channel, the current status of the chaining transfer is stored in the DMA memory that belongs to the corresponding normal channel. Therefore, (given that no higher priority channels have been queued) when the chained transfer resumes after the higher priority interrupt, the transfer is continued from the IRQIn asserted bit which will correspond to the normal channel.

7.6.2 General operation mode

The Hermes device provides more flexibility in the transfer modes than available in the Hercules device. This means that the Hermes device will support the following

- Any combination between fixed address mode (typically an I/O device) and incremental address modes (memory).
- “All” of these combinations supports Stop and Go modes.
- The incremental addressing modes supports Circular incremental mode (ring buffers and FIFO implementations) and Linear incremental addressing mode (“conventional” buffer addressing).

These combinations are shown in the following table:

Where '√' = valid, And '+' = invalid.		Incremental to Fixed address mode	Incremental to Incremental address mode	Fixed to Incremental address mode	Fixed to Fixed address mode
Circular	<i>GO</i>	√	√	√	√
	<i>STOP</i>	√	+	+	+
Linear	<i>GO</i>	√	√	√	+
	<i>STOP</i>	√	√	√	+

Table 34: Transfer modes supported by the DMA.

As can be seen from this table, some of the modes are marked as not supported. The reason for this is the following:

- In case of Incremental-to-Fixed Circular address mode there is no need for circular combined with stop mode since this is just Incremental-to-fixed in stop mode.
- In case of the Incremental-to-Incremental address mode, a Circular Stop mode is the same as a Linear Stop mode.
- In case of the Incremental-to-Incremental address mode, a Linear Go mode is given implicitly.
- In case of the Fixed-to-Fixed address mode, a Circular Stop mode, a Linear Go mode or a Linear Stop mode will make no sense.

7.7 Communication System Functions

Changes to this section include dropping the Direct Memory Access block, and the uSM and slow clocking modules, and adding the MCU Cipher module, renaming A Sport to C Sport, and adding the GSP to support the audio A SPort function (which used to reside in the DSP IDMA block on Hercules).

8. External Bus Specification

Changes to this section include adding the burst FLASH support of the Hermes specification, and possible support for the new low-power memories (e.g., V-Sync FLASH, BAT RAM, pseudo-RAM). Also, the Display Interface function is not supported.

8.1 FLASH interface

In addition to an industry standard, single access type Flash memory, Monaco supports the industry standard burst Flash memory types (16 bit wide), in asynchronous, page, and burst modes of operation. All read/write operations are half-word wide. The Burst Mode Flash interface included in the Monaco device supports the following system requirements:

- Support of asynchronous and synchronous read protocols
- Fetch of one half word at a time.
- Address pipelining.
- No external glue logic necessary
- Once a burst access has begun i.e. an active EBUS access, it cannot be interrupted.
- Intel and AMD burst- and page-mode flash devices are supported.
- For Intel devices the Read-While-Write (RWW) and Read-While-Erase (RWE) features are supported [7].
- FLASH wrap mode is NOT supported (incompatible w/ burst length from DSP).

The interface supports asynchronous read mode, asynchronous page-mode read, single synchronous read, synchronous continuous read, and programmable mode. Upon initial power up or return from reset, both the AMD burst mode Flash device and the Intel burst mode Flash device start up in asynchronous read mode and thereby behaves as a conventional Flash circuit. Therefore, upon reset, the burst mode Flash interface is configured to support this mode. The following table shows the required signals for supporting these modes.

Table 35: Burst FLASH Interface Signals

FLASH Memory Signal Name	Direction	Description
ADR[23:0]	Out	External address bus.
DATA[15:0]	Bidir	External data bus.
nCE	Out	Select signal.
CLK	Out	Clock signal for Flash device burst counter.
ADV	Out	Address Latch Valid.

nWAIT	In	Wait signal from Flash device used during continuous burst mode operation or when data is not valid
nOE	Out	Output enable.
nRST		Reset, system level reset, not an output of Monaco
nWE	Out	Write Enable.
WP		Write Protect, not supported

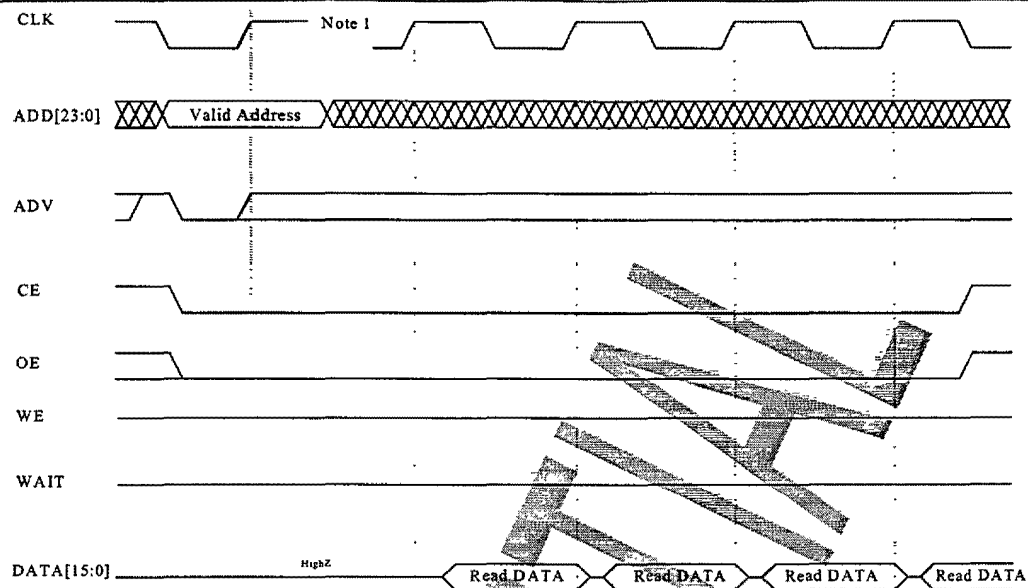
The output delay (indicated by nWAIT = Logic '0') occurs when the burst sequence crosses a line boundary or when the read data is not valid. WAIT is inactive during single synchronous, asynchronous read cycles and in programmable mode.

The flexibility of burst mode memory functions can be achieved by controlling the content of the Flash's internal device configuration registers. It is possible to configure the device (in synchronous burst mode) for either one, four, eight, sixteen or continuous burst mode. However, the Monaco chip does not support wrapping mode.

Since the MCU processor can run at a frequency of at least 75MHz, and the maximum clock frequency for current FLASH memories is 52MHz, single cycle MCU access to the FLASH is not possible at maximum MCU frequencies. There exists some frequency, well below 52 MHz, at which the FLASH can support single cycle access from the MCU. However, at higher BCLK frequencies, the EBUS clock divider must be enabled, to stay within the supported operating parameters of the FLASH. The current clock divider function provides only a divide by 2 option. When enabled, and the BCLK is switching at 75 MHz, the FLASH will be cycled at a 37.5 MHz rate.

Only a sequence of less than sixteen words can be fetched at a time. This helps to limit the occupation of the EBUS to a single master. Once the appropriate number of half words has been fetched from the Flash, the arbiter grants the EBUS to the master of highest priority in the queue.

Since the Flash devices are 16 bit data devices, the burst flash mode interface always performs data writes and fetches of half words no matter what the requested data is: a byte, half word, or a word. The arbiter simply adjusts the accesses to the wanted access type. The following figure shows a synchronous burst read operation of four half words. Once the burst flash device has latched the address, the subsequent rising edges of the clock will increment the flash internal address counter.

**NOTE:**

1. Depending on frequency configuration code value in the read configuration register of the burst FLASH memory (2 up to 6 clock cycles delay). See FLASH datasheet for further information.

Figure 27: Synchronous burst read operation Timing, four half word burst length.

The nWAIT signal is only used when the flash device is in its burst mode and will tri-stated otherwise. This signal indicates when the interface should wait one clock cycle (this number is however configurable) before expecting the next half word of data. An example of the assertion of the wait signal is shown in Figure 29.

[TBD]

Figure 28: Asynchronous burst read operation Timing, four half word burst length.

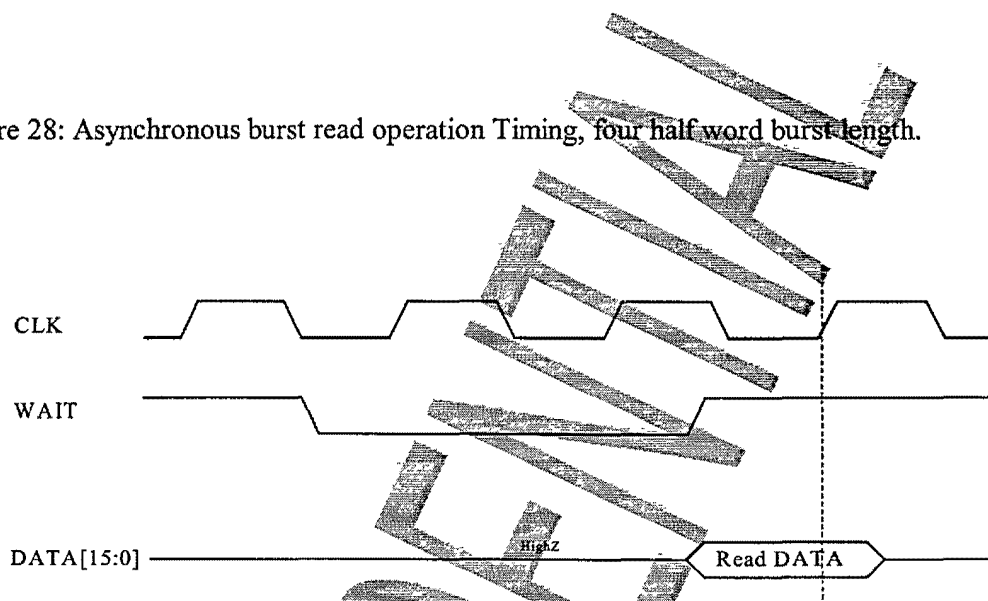


Figure 29: Wait signal timing during continuous burst read operation

8.1.1 Performance gain when using burst mode

Add spreadsheet graph here, demonstrating performance delta for same initial access time.

[TBD]

8.1.2 Internal Wrapping Burst Transfer Support

In order to take the most advantage of the burst mode and decide when to do burst accesses, the external interface module (EBS) from Monaco receives information from the MCU (sequential access) and DSP caches (if enabled).

The DSP cache line fill operations are 32-byte, wrapping burst read transfers. The initial address for the burst transfer from the cache can be 1st, 2nd, 3rd, or 4th 64-bit address of the cache line, depending on the DSP core transfer address which generated the cache line fill operation. The following table describes the possible scenarios

Table 36: Supported Internal Address-wrapping Burst Transfers for Burst FLASH

DSP Burst Read Initial Address[3:0]	Resulting Burst FLASH Transfers (16-bit data bus)
--	--

000	Single Incrementing burst of 16
010	Incrementing burst of 12, starting at address 010, followed by an incrementing burst of 4 from address 000
100	Incrementing burst of 8, starting at address 100, followed by an incrementing burst of 8 from address 000
110	Incrementing burst of 4, starting at address 110, followed by an incrementing burst of 12 from address 000

8.1.3 EBS Pipelining FLASH Accesses

The SBIU decodes whether the target of the DSP/DMA access is external memory (EBUS), or System RAM (RBUS). If the target is external memory, the SBIU signals the incrementing burst indicator to the EBS, and where appropriate, performs up to two incrementing burst transfer requests to complete the entire 32 byte transfer. The SBIU also locks the EBUS arbitration during the entire cache line fill/copyback transfer.

The MCU asserts a sequential address indicator (SEQ), valid with the MCU request, which indicates that the next access from the MCU is a sequential 16-bit offset from the current address. This signal provides the equivalent function to the burst indicator from the DSP SBIU. In either case, the EBS is responsible for pipelining the external burst FLASH read operations until it detects a new discontinuous half-word address, as indicated by:

- The burst indicator is deasserted by the SBIU, even if the arbitration is still locked
- The SEQ indicator is deasserted by the MCU, or the MCU loses arbitration to another requester, even if the SEQ indicator is still asserted
- No further accesses, such that the FLASH accesses should stop to conserve power consumption

The signals added for Burst Flash interface are brought out on the Monaco pins as presented on the table:

Table 37: Burst FLASH Pinout Multiplexing

The burst flash functionality can be mapped in one the following off-chip address segments:

Recommended modes

[TBD]

Opens:

- *Should we add support for the new "mobile" memory products? Which ones? What specific support is required?*
- *Do we need to have a mode bit in the SBIU to not lock EBUS arbitration for the entire cache line fill? Perhaps arbitration should at least open up between the two burst transfers if the access is not 32-byte address aligned. This would bound the worst case latency for MCU requests to the burst of 16 half-word case.*

9. RAM Bus Specification

Changes in this section to decode DSP sub-system access from MCU, system DMA. RBUS arbitration control has moved from SBUS, to the PBUS. Also possible size changes to the System RAM.

9.1 RBUS

The RBUS arbiter module has programmable configuration registers. For Monaco, both the DSP and the microcontroller can access the configuration registers (RBARBCR0,1), in the HouseKeeping PBUS group. Correctly written software must prevent simultaneous access to the configuration registers. If a conflict occurs, the hardware resolves the conflict according to the following priority (highest priority completes first, lowest priority completes last):

1. DSP
2. Microcontroller

Note that the lower-priority requester writes last, so if both requesters attempted a simultaneous write, the lower-priority requester would appear to have won the arbitration. We favor the DSP in the arbiter only because the waiting the DSP generates many more cycles of stall, due to the difference in clock rate between the two controllers.

The arbiter configuration address space is not mapped into the DMA controller address space; therefore, the DMA controllers cannot inadvertently generate accesses to this region.

9.2 System RAM

The Monaco device integrates larger system memory made available to the MCU, DSP, System DMA, DSP DMA, and EAPI. The SRAM area is placed on the RBUS and is increased from 1Mbits on Hercules to 4 Mbits on Monaco. The 4 Mbit system memory area is accessible by the MCU at full MCU clock frequency (up to 75MHz, zero wait state access area). The following access cycles are therefore required for the respective masters.

Master	Access Time (BCLKs)
MCU	1 cycle
DMA	1 cycle
DSP (DSP-DMA)	1 cycle
EAPI	4 cycles

Table 38: Cycle accesses to the SRAM places on RBUS (with a 75MHz MCU).

In this scheme, the MCU, DSP (and DSP DMA), DMA and the EAPI Controller are the masters and the SRAM modules are considered slaves. The handling of byte, half-word, and word accesses, as well as the arbitration between the masters is performed into the RBUS arbitration and interface logic.

9.2.1 MONACO memory maps

The SRAM is always mapped to the 1Gbyte segment from address 0x4000 0000 to address 0x7FFF FFFF. Considering the memory map for the SRAM, a few requirements must be fulfilled. The requirements are

- Possibility of placing the interrupt service routine in the internal SRAM area.

-
- Fastest possible execution of the interrupt service routine.
 - A minimum of introduced delay at jumps between internal SRAM and FLASH.

To comply with these requirements, the low address area will be shadowed by the internal SRAM so that the SRAM is conditionally mapped (by the MEM_CTR bus configuration register) to 16Mbytes segment from address 0x0000 0000 - 0x00FF FFFF.

Thus, both in the 16Mbytes segments from 0x0000 0000 to 0x00FF FFFF and in the 1Gbyte segment from 0x4000 0000 to 0x7FFF FFFF, the 4 Mbit SRAM is aliased to comply with the MCU address map. In this way it will be possible to place the interrupt service routine in the internal SRAM. The SRAM mapping is shown in the following diagram. The full memory maps are shown in Section 14.1 Memory Maps.

CONFIDENTIAL

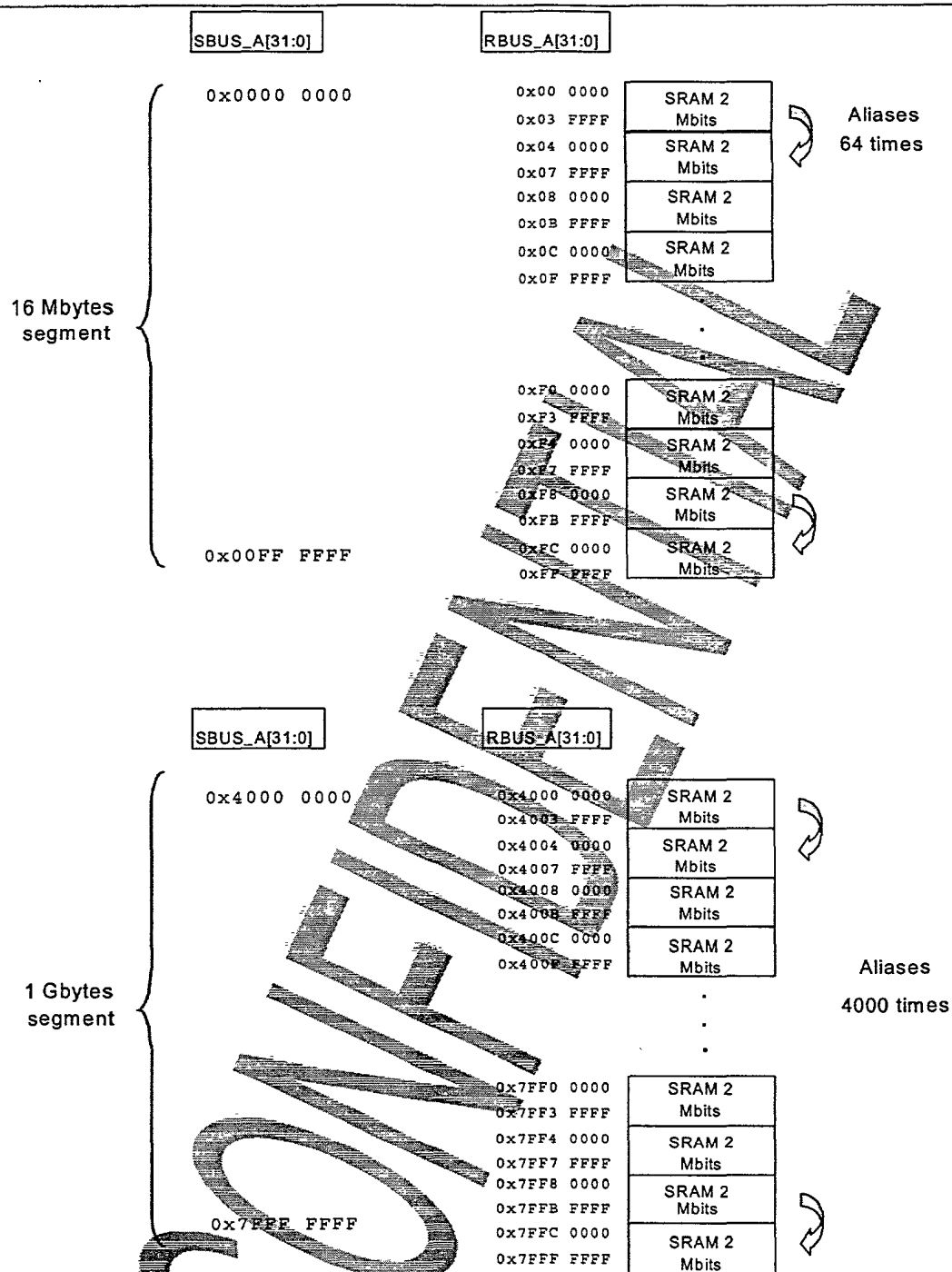


Figure 30: Alias of the SRAM.

10. SBUS Specification

The SBUS of Monaco represents the raw MCU core bus signals. This is the local bus of MCU, and is not visible to the rest of the Monaco system. The modules on this bus in the Hercules architecture include the MCU clock gen, system arbiter control, bus select decoders, and the BootROM Test

interface. For Monaco, we require DSP access to all of these functions, with the exception of the bus select decoders and the BootROM interface (Note that the MCU clock gen function changes significantly for Monaco). For this reason, the MCU clock gen and system arbiter control registers have been moved off the SBUS, to the House Keeping group on the PBUS, according to the following table. For the specific PBUS Address offsets, see the Ch.12 PBUS Memory Maps.

The registers for all SBUS local peripherals are word aligned, starting at MCU address 0xC000 0000 used as base for all peripherals. The table below summarizes the peripherals and their address offsets relative to the base.

Table 39: SBUS Peripheral Set and Registers

Peripheral	Function	SBUS Address Offset	Moves to PBUS?
MCU CLK GEN	MCU Clock Control	0x0000 – 0x007F	Yes
	System Clock and DSP PLL Control	0x0080 – 0x008F	Yes
	DMD Module ⁸	0x0100 – 0x01FF	No
	Reserved	0x0400 – 0x04FF	No
ARBITERS	PBUS Arbitration Control	0x1000 – 0x10FF	Yes
	RBUS Arbitration Control	0x1100 – 0x11FF	Yes
	EBUS Arbitration Control	0x1200 – 0x12FF	Yes
BUS SELECT DECODER	MCU Memory Map Control	0x2000 – 0x20FF	Yes
	(Becomes a combined MCU/DSP Memory Map Control)		
DBGEN	ARM Debug Control	0x2100 – 0x21FF	No
BootROM	Test Mode Access	0x4000 – 0x4FFF	No

10.1 MCU Clock Generator

This section has moved to the following chapter, Clock Control and Power Management. This is because both DSP and MCU must be able to access system clock control.

10.2 Arbitration Registers

Same definition as Hercules, except for:

1. Access to these registers is now through the PBUS, rather than the SBUS, as described above.
2. The PBUS MAI access to DSP memory, does not exist in Monaco. All MCU accesses to the DSP sub-system go through the RBUS path.
3. PBUS arbiter is replaced by an R/EBUS arbiter. This is because the ABUS (supporting External App. Processor Interface, EAPI), must have access to the PBUS.
4. The PMMU input to EBUS and RBUS arbiters is replaced by the ABUS input from the EAPI. The ABUS does not support burst transfers. However, the DSP BUS input to the E/RBUS arbiters DOES support burst transfers. The SBIU is responsible for ensuring that line wrapping burst requests from the L1 cache are broken into incrementing burst requests to the EBUS (and therefore the RBUS, as well).
5. Changes (TBD) to the E/RBUS Bandwidth Calculations sections

Opens:

⁸ Dual Modulus Divider/ICO is not supported in Monaco

- *DPBUS also has arbiter control, as it manages requests from DSP, DSP DMA, and MCU/System DMA (via RBUS into SBIU). These control registers are in the SBIU, and not accessed via PBUS. Since we are moving the P/E/Rbus arbiter regs anyway, should we bring these up to the DPBUS, to have a more consistent programming model?*
- *Need to revisit the E/RBUS Bandwidth Calculations*

10.3 Bus Select Decoders

Same definition as Hercules.

10.4 Security Features & Boot ROM

The Monaco device features a unique serial number for each device. This number consists of a sequence of 32 bits. A field of this serial number is used to WDT out of reset (see the WDT discussion in the MMI for more detail).

One area of the boot ROM can be reserved for program code and be protected so that the content is not visible from outside of the chip even when the emulator is enabled.

The Monaco device supports 4kB Boot ROM connected to the SBUS. The ROM aliases as shown in the following diagram.

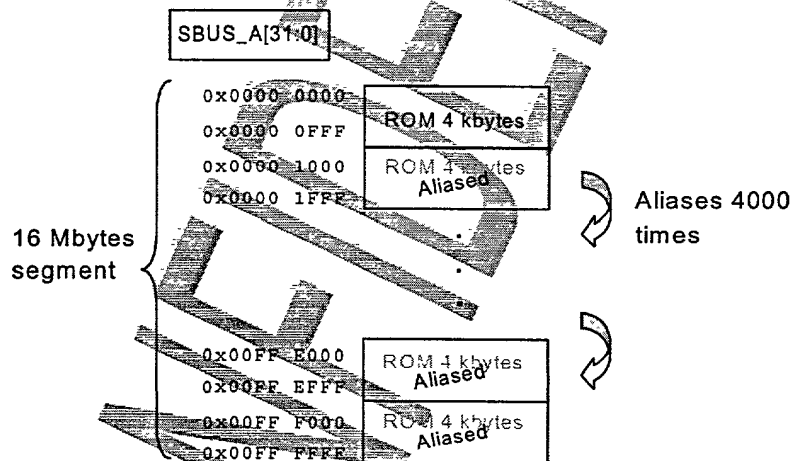


Figure 31: Alias of the Boot ROM on the SBUS.

More [TBD]

Opens:

- *Serial ID definition*

10.5 DBGEN Control

Same definition as Hercules.

11. System Clock Control and Power Management

11.1 System Clock Control

The CLOCK CTL module is responsible for clock control within the Monaco system, including System PLL programming for both the MCU, DSP, as well as the Peripheral PLL. Since this module is accessed through the PBUS, both the MCU and DSP have access to the CLOCK CTL. Care must be taken (i.e., software resource locking may need to be provided) to ensure that only 1 master attempts to modify the state of this module, at any given time.

The CLOCK CTL module contains the following primary functions:

1. System clock and PLL control
2. MCU clock control
3. DSP clock control
4. Peripheral clock and PLL control

Note that the MCU PLL and Dual Modulus Divider functions of Hercules are not supported in Monaco.

Clock Gen Control block goes here. Programming model changes include moving the memory map from local SBUS, to the PBUS House Keeping section. Additionally, the register definitions change to support the single PLL clocking structure of Monaco.

See also 2.5 System Clocking, for a discussion on Monaco clock definition.

11.1.1 Monaco Clock States

The following operational states have been identified for the Monaco device:

State	SysCLK	DCLK	DSCLK	BCLK	MCLK	SysPLL	Comments
0	Off	Off	Off	Off	Off	Off	Idle, back-bias enabled to all modules except RTC domain (TEP, RTC)
1	On	Off	Off	Off	Off	Off	TEP active on SysCLK.
2	On	Off	Off	On	Off	On/Off	PDA mode DMA, UART, etc. active, MCU idle
3	On	Off	Off	On	On	On/Off	PDA mode DMA, UART, etc. active, MCU active
4	On	Off	On	On	Off	On/Off	Paging Mode I/Q samples DMAed into L2
5	On	Off	On	On	On	On/Off	DSP Idle, L2, MCU active
6	On	On	On	On	Off	On	All active, except MCU idle
7	On	On	On	On	On	On	Full On State

Table 40 Clock States

Each of these states is addressed in the configuration flexibility of the following clock architecture.

11.1.2 System Clock Control

The clock trees for each of the major clock domains can be disabled in the different low-power states, to conserve power otherwise lost in charging the clock tree even though the modules driven by that clock tree are idled. Additional clock gating at the module level further provides a finer granularity power conservation feature.

11.1.3 System Clock Control Registers

System Clock Control Registers are mapped as shown in the following table. Since both the DSP and MCU may access the SCCR and SYSPLLCR registers frequently, each has separate set and clear access functions such that read-modify-write accesses are not generally required. A write to a set address offset, with 1 or more bits high, sets the corresponding register bit(s) to a logic high value. A write to a clear address offset, with 1 or more bits high, sets the corresponding register bit(s) to a logic low value.

Register Name	Function	Address offsets
SCCR_SET	System Clock Control Register Set	TBD
SCCR_CLR	System Clock Control Register Clear	TBD
SYSPLLCR_SET	System PLL Control Register Set	TBD
SYSPLLCR_CLR	System PLL Control Register Clear	TBD
SFRSTCR_SET	Software Reset Control Register Set	TBD
DSPCR	DSP Control Register	TBD

CONFIDENTIAL

At the system level, clock control consists of on/off switches and clock divider selectors. These define clock activity on clock branches within the device for the MCU, DSP, Arbiters, and peripherals, and can therefore be used to save power when a clock source is not required. Note that all peripherals contain local clock control, which can be used to ensure that the clock is gated off to that module when not required. The Monaco clock tree schematic is shown below.

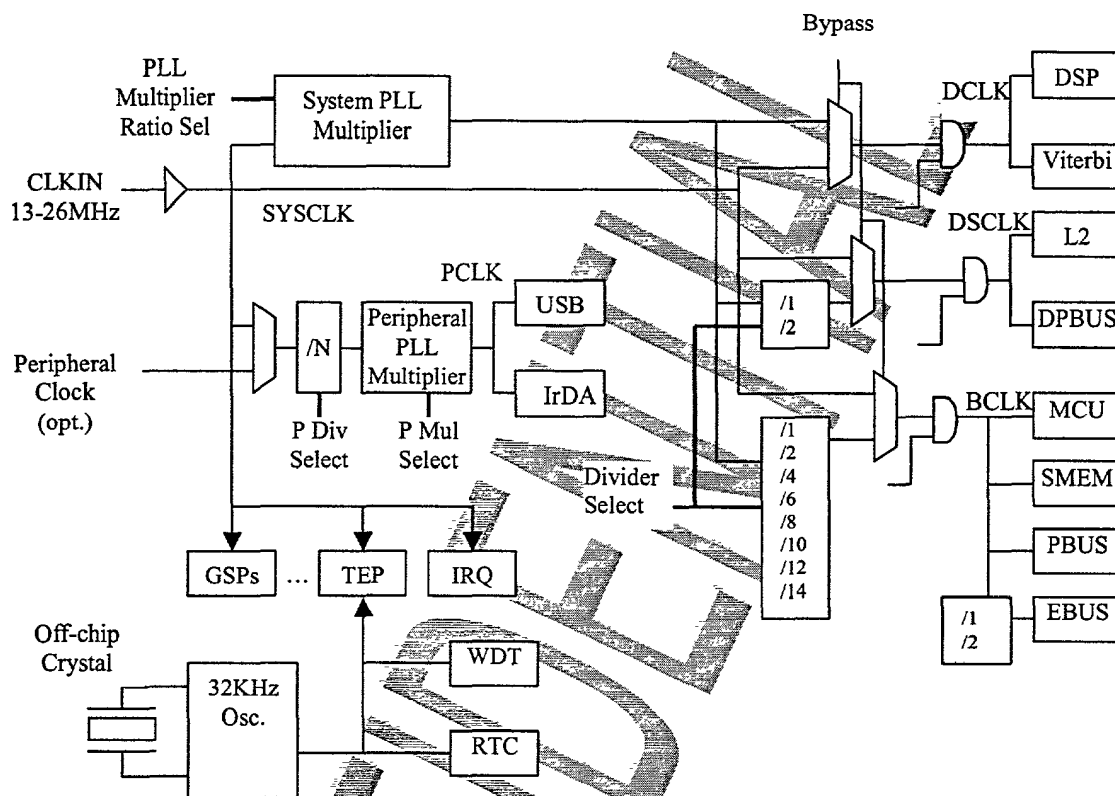


Figure 32 Monaco System Clock

The System Clock Control Register (SCCR) is defined in the following table. Entries in this register can be read directly. Values are written via the set and clear functions described in the overview.

Register	Add	[7]	[6]	[5]	[4]	[3]	[2]	[1]	[0]	R/W
SCCR	0xTBD	BCLKSEL[2:0]			BCLK OFF	DSCLK SEL	DSCLK OFF	TBD	DCLK OFF	r/w
(Reset)		011			0	1	0	0	0	
Register	Add	[15]	[14]	[13]	[12]	[11]	[10]	[9]	[8]	
SCCR	0xTBD	TBD	TBD	TBD	TBD	TBD	TBD	TBD	TBD	r/w
(Reset)		0	0	0	0	0	0	0	0	

Table 41: System Clock Control Register

DCLK control:

- If the DCLK OFF bit is cleared, the DCLK clock tree is enabled and driven. Note that the System PLL must be enabled if the DCLK OFF bit is cleared.
- If the DCLK OFF bit is set, and the DSP is in the IDLE state (as indicated by the assertion of the SLEEP output from the DSP), the DCLK tree is gated off. Upon assertion of the DSP wakeup function (e.g., by a DSP interrupt) hardware will re-enable the DCLK tree.
- DCLK always runs at the same frequency as the clock input selected by the Bypass Mux.

DSCLK control:

- DSCLK OFF:
 - If cleared, the DSCLK clock tree is enabled and driven.
 - If set, the DSCLK tree is gated off. This bit should not be set by the DSP, as it could possibly cause the DSP to hang. This bit would normally only be set if the MCU ran for extended periods in which none of the DSP system resources are required.
- DSCLK SEL:
 - If cleared, the DCLK:DSCLK divider ratio is 2:1
 - If set, the DCLK:DSCLK divider ratio is 1:1

BCLK control:

- BCLK OFF:
 - If cleared, the BCLK clock tree is enabled and driven.
 - If set, the BCLK tree is gated off. This bit is likely under TEP control only.
- The BCLK tree cannot be gated off. Assertion of the CLKIN disable bit will stop CLKIN, eliminating any power consumption in this domain.
- BCLK is always synchronized to DSCLK, such that a rising edge of BCLK is in phase with a rising edge of DSCLK.
- The BCLKSEL[2:0] selects the desired DCLK to BCLK ratio, according to the following table:

001	$f_{BCLK} = f_{DSCLK} / 1$
010	$f_{BCLK} = f_{DSCLK} / 2$
011	$f_{BCLK} = f_{DSCLK} / 4$
100	$f_{BCLK} = f_{DSCLK} / 6$
101	$f_{BCLK} = f_{DSCLK} / 8$
110	$f_{BCLK} = f_{DSCLK} / 10$
111	$f_{BCLK} = f_{DSCLK} / 12$
000	$f_{BCLK} = f_{DSCLK} / 14$

Opens:

- *Is the BCLK OFF bit redundant with the CLKIN enable control bit?*
- *Need to add the MCU sleep control and status bits somewhere (SLP2, SLP1, and DPS)*
- *What about the CLKMUX, BUSWRCLK, DSTG, ACCSTG, ARSTG test registers?*

11.1.4 DSP Control Register

The following register does not need separate set and clear functions, as it is not likely to be accessed frequently. The MCU is responsible for initialization of the DSP state, and sequencing (booting) the DSP to an operational mode.

Register	Add	[7]	[6:5]	[4]	[3]	[2]	[1]	[0]
DSPCR	0xTBD	DSP ENDM	DSP DMC[1:0]	PWD ACK	DPWD	TBD	DSRAM Init	DSP RESET
(Reset State)		1	00	0	0	0	1	1
(R/W)		r	r/w	r	r	r/w	r/w	r/w

Register	Add	[15]	[14]	[13]	[12]	[11]	[10]	[9]	[8]
DSPCR	0xTBD	TBD	TBD	TBD	TBD	TBD	TBD	DSP ENIM	DSP IMC
(Reset State)		1	1	1	1	0	0	1	0
(R/W)		r	r	r	r	r	r	r/w	r/w

Table 42: DSP Control Register

The DSP ENIM flag is set after reset, and is the instruction L1 memory enable. When this bit is cleared, instruction L1 memory is disabled.

The DSP IMC flag configures the DSP L1 instruction memory. After reset, the instruction L1 memory is configured as SRAM (only). This flag must be set to enable instruction L1 cacheing.

The DSP ENDM flag is set after reset, and is the data L1 memory enable. When this bit is cleared, data L1 memory is disabled.

The DSPDMC[1:0] bits configure the DSP L1 Data memory. After reset, the both banks of the data L1 memory are configured as SRAM (only). These bits must be written to one of the following values to enable data L1 cacheing:

- 0b10 – Superbank A configured as cache, Superbank B configured as SRAM
- 0b11 – Both Superbanks configured as cache

The PWDACK flag will be asserted high when the DSP has executed the IDLE instruction following a PWD request, and has therefore entered Power Down Mode. A PWD interrupt can be generated via the DSP CTRL Register in the MMI module. When the PWD flag is de-asserted, the PWDACK will be cleared after the DSP has exited Power Down mode.

The DPWD flag is a read only copy of the PWD flag in the MMI register set.

The DCLK OFF select can be asserted only once the nPWDACK flag is active (low), and can be used to save power. It must be cleared before the DSP exits Power Down, but after the DCLKIN Select has been configured.

The SRAM INIT flag is set after a System Reset. This bit has no effect, except immediately after the DSPRESET flag is cleared. If set, SRAM INIT stalls the DSP, allowing L1 memory, if configured as SRAM, to be initialize by the DSP DMA. Once cleared, the DSP begins fetching instructions.

The DSPRESET flag is set after a System Reset causing the DSP and its peripherals to be reset. The DSP clock must be active during a DSP Reset, therefore DCLKIN OFF must be cleared first. If the DSP is in Power Down, a DSP Reset will force it to exit Power Down. The flag must be cleared in order for the DSP to exit the reset state.

Opens:

- *Is FrioLP config identical to Frio?*
- *Frio reset vector – need to add a read/write register which specifies this.*

11.1.5 PLL Multiplier Ratios

The multiplication ratio is controlled by two set of signals, MSEL[6:0] and DF. These signals configure the clock dividers/multiplications performed in the PLL. The DF signal enables/disables the input divider placed before the phase detector, and MSEL[6] enables/disables the output divider placed after the VCO. The feedback divider is controlled by MSEL[4:0] which determines the state of the $\div N$ (1:32) divider. The block diagram is shown in Figure 33 and the multiplication ratios are shown in Table 43. Note that MSEL[5] is fixed to LOW.

With a 13-26MHz system clock, some of the multiplication ratios cannot be used since they would result in unsupported VCO operating conditions. For example, assuming a 13MHz CLKIN, the multiplication ratio must be limited to $< 23\times$.

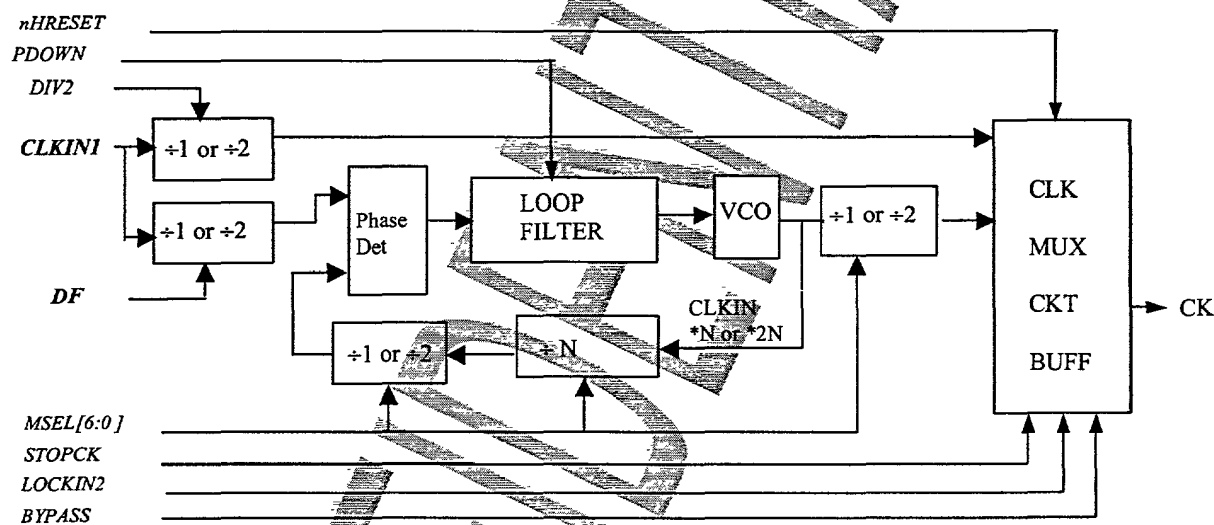


Figure 33: Block diagram of the System PLL

Signal name	MSEL[6]=0 & DF=0	MSEL[6]=1 & DF=0	MSEL[6]=0 & DF=1	MSEL[6]=1 & DF=1
MSEL[4:0]	DSP_CLK	DSP_CLK	DSP_CLK	DSP_CLK
0	32x	16x	16x	8x
1	1x	0.5x	0.5x	0.25x
2	2x	1x	1x	0.5x
3	3x	1.5x	1.5x	0.75x
4	4x	2x	2x	1x
5	5x	2.5x	2.5x	1.25x
6	6x	3x	3x	1.5x
7	7x	3.5x	3.5x	1.75x
8	8x	4x	4x	2x
9	9x	4.5x	4.5x	2.25x
10	10x	5x	5x	2.5x
11	11x	5.5x	5.5x	2.75x
12	12x	6x	6x	3x
13	13x	6.5x	6.5x	3.25x
14	14x	7x	7x	3.5x
15	15x	7.5x	7.5x	3.75x
16	16x	8x	8x	4x
17	17x	8.5x	8.5x	4.25x
18	18x	9x	9x	4.5x
19	19x	9.5x	9.5x	4.75x
20	20x	10x	10x	5x
21	21x	10.5x	10.5x	5.25x
22	22x	11x	11x	5.5x
23	23x	11.5x	11.5x	5.75x
24	24x	12x	12x	6x
25	25x	12.5x	12.5x	6.25x
26	26x	13x	13x	6.5x
27	27x	13.5x	13.5x	6.75x
28	28x	14x	14x	7x
29	29x	14.5x	14.5x	7.25x
30	30x	15x	15x	7.5x
31	31x	15.5x	15.5x	7.75x

Table 43: Multiplication ratios for the System PLL (MSEL[5]=0).

11.1.6 System PLL Control Register

The selection of the system PLL's programmable multiplication and clock division PLL module is done via the control register (SYSPLLCR). Entries in this register can be read directly. Values are written via the set and clear functions described in the overview.

SYSPLLCR Register

Address 0x TBD

Register	15															0
SYSPLLCR	LOCK				MSEL[6:0]							PWDN	TBD	BYPASS	STOPCK	DF
Reset State	0	0	0	0	0	0	0	0	0	0	1	1	0	1	0	0
Read/Write	R	0	0	0	RW	0	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW

MSEL[6] - enables the output divider

MSEL[5] - fixed to LOW

MSEL[4:0] - controls the divide ratio of the feedback divider

PDWN - when 1 PLL logic is in power down, the VCO is disabled.

BYPASS - when 1, PLL is in bypass mode (CK = CLKIN or CK = CLKIN/2)

STOPCK - when 1, stop CK LOW

DF - when 1, divides CLKIN/2 before the phase detector.

Figure 34: SYSDSPCR Control register for the DSP PLL.

After reset, the PLL is powered down, and bypassed, such that the internal system clock receives the CLKIN frequency.

Opens:

- *Lock indicator – not part of 219x PLL. For Tahoe, we had a dedicated Lock counter, which was programmable. What do we really need for Monaco? A reliable, deterministic lock indicator is difficult to build into the phase detector, but is the most desirable, since it can potentially assert earlier than a conservative counter-based approach. But the latency may be tolerable, if we do not have to wait for PLL Lock to begin ISR processing. Either way, we need a lock indicator which can be enabled as an interrupt to both/either MCU/DSP.*

11.1.7 Peripheral PLL

The Peripheral PLL is similar to the system PLL (SYS_PLL). The PLL is programmable; the control register resides on the PBUS. Both DSP and MCU processors can access the control register at address 0x8034 on PBUS. The Peripheral PLL can be used to feed the clock GSPa module in the case of IrDa protocol or feed the clock to the USB peripheral.

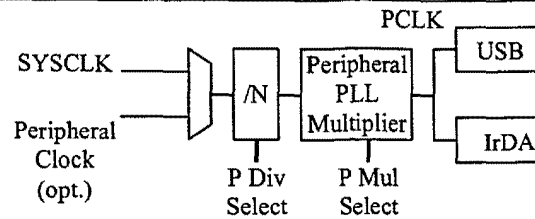


Figure 35 Peripheral Clock Block Diagram

The Peripheral PLL multiplier ratio is selectable across the same range as the system PLL, as can be seen from the PR_PLL control register definition.

As can be seen in the above figure, the Peripheral PLL is fed by a programmable divider function, which is clocked by either the SysCLK (13-26MHz), or an off-chip clock oscillator. The clock divider is programmable, providing a divide-by-N function, where $N = \{1, 2, \dots, 16\}$, which corresponds to the PDIV[3:0] field in the peripheral PLL divider control register.

When not in use, the PLL and clock divider can be completely shut off and powered down.

Opens:

- *Should this PLL have a separate back-bias enable, to further reduce leakage current?*
- *PLL jitter tolerance*

11.2 Software Reset

It is possible in the Monaco device to perform a software reset by setting a 16 bit register value to 0x0007. This register is placed on the PBUS. After being set in software, the bit is cleared automatically by the reset sequence. The soft reset is implemented in the reset path as shown in the following figure.

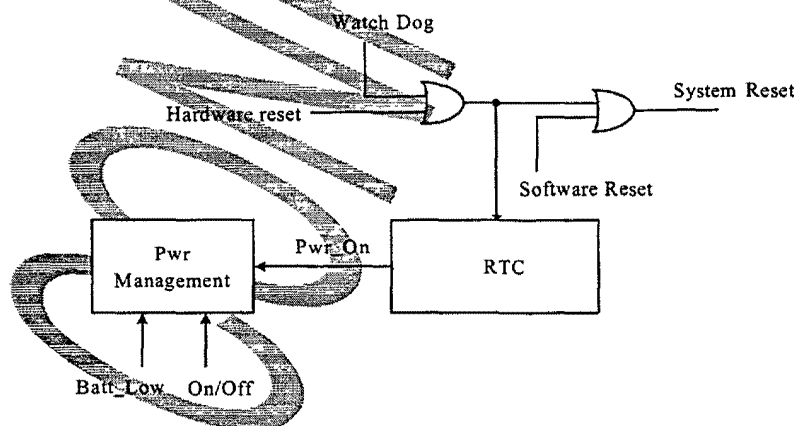


Figure 36: Illustration of how the software reset is implemented.

11.3 Power Management

Monaco includes several mechanisms to facilitate management of both dynamic and static power consumption. The dynamic power management mechanisms include:

- Dynamic voltage control
- Clock frequency control
- Local module clock gating
- Power state controller
- Substrate forward-bias control

The static power management mechanisms include:

- Substrate back-bias control
- Dynamic voltage control

Additionally, the Monaco design uses special circuit design/process features to minimize overall leakage currents, including:

- Use of special low-leakage current devices where possible (e.g., High Vt transistors employed in large memory arrays)
- Low-voltage swing signals where appropriate (e.g., CLKIN, memory cell bit lines)
- Fully static CMOS design

The following sections describe each of these mechanisms in detail.

11.3.1 Background

Static power consumption is the most critical power consumption issue, due to the extended periods of time in which most of the dynamic power consumption approaches zero (i.e., most of the time, the phone is in standby mode, in which the phone periodically must enter paging mode, but is otherwise relatively quiescent). Standby power consumption is comprised of the leakage currents which flow due to the circuits which are powered up.

Dynamic power consumption, when present, adds to the static power consumption factor, and is described by the following formula:

$$P_D = cV^2F, \text{ where:}$$

c is a constant determined by the circuits operating at that voltage and frequency

V is the voltage difference between Vdd and Gnd

F is the clock frequency

For Monaco, several different voltage and clock domains, or areas, exist. The total dynamic power consumption is the sum of the dynamic power consumed in each of these domains. For example, in the fully active state, the baseband may have six active clock domains:

1. DSP Core Clock (DCLK)
2. DSP System Clock (DSCLK)
3. MCU System Clock (BCLK)

4. System Clock (SysCLK)
5. USB Clock
6. Slow Clock

The internal logic driven by these clock domains are in one voltage domain (internal Vdd). The analog PLL is on an isolated voltage domain (PLL Vdd), as is the RTC oscillator (RTC Vdd), which generates the 32KHz clock of the Slow Clock domain. In standby mode, sections of the TEP are also powered by RTC Vdd.

11.3.2 Dynamic Voltage Control

For any of the active power states, there are likely to be periods of time during which maximum frequency of the system in that mode is not required. Whenever this scenario exists, power consumption can be reduced, perhaps significantly, by reducing internal Vdd to the minimum level required to support the minimum clock frequencies required to support the needed performance level. Monaco integrates dedicated hardware support for explicit software management of clock requirements, which implicitly manages the internal Vdd, when appropriately configured. This hardware provides a simple programming model which minimizes the impact to software, thereby maximizing the potential benefit of this power management tool.

The minimum power required to support any phone state is defined by the clock frequency requirements in that state. Software manages the clock frequency explicitly, by setting the PLL multiplier ratio, via the MSEL[5:0] control field. The dynamic voltage control block includes a lookup table (RAM), indexed by the PLL MSEL, used to determine the appropriate internal Vdd level. This function also includes a finite state machine, which sequences the power/PLL appropriately. The following block diagram describes the system:

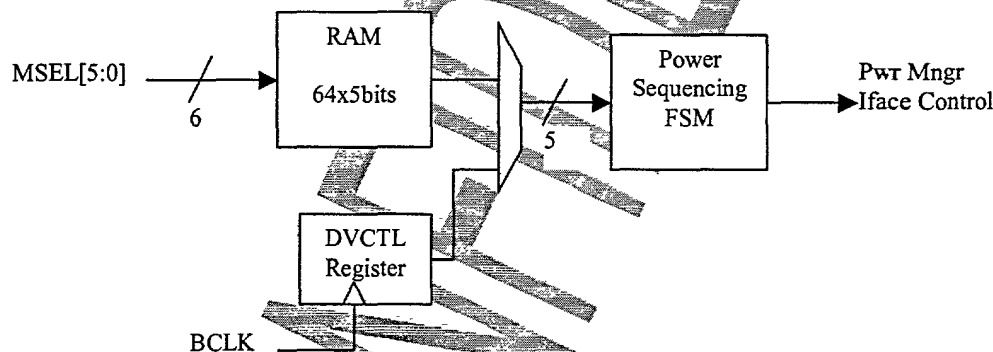


Figure 37: Dynamic Voltage Control Block Diagram

The DVCTL register includes the TLVAL[4:0], the default table lookup value used until the RAM is set up and enabled. Software loads the table lookup values into the RAM, as part of the reset initialization sequence. The table lookup values are determined by product engineering characterization of the silicon. The 6-bit MSEL field provides an index of up to 64 unique 5-bit values. The 5-bit lookup values provide a granularity of 32 voltage steps. 32 steps across the variable internal Vdd range of 800mV (assume max of $1.5V - 0.7V = 0.8V$) provides a granularity of 25mV.

To be added:

1. Add diagram of V^2f curve. Add text describing tradeoffs of fast re-lock time vs. absolute minimum V for a given frequency. In other words, if the voltage step requires a PLL re-lock, and latency is more critical than absolute maximum power savings, set the voltage higher than

that required to support the target frequency, by some TBD delta. This should ensure that any VCO overshoot of the target lock frequency will not violate maximum timing.

2. Description of power sequencer, differences between stepping voltage/frequency up, and stepping volt/freq down.
3. Description of power management interface (GSP).

11.3.3 Back-Bias Controller

To reduce leakage current to an acceptable level, Monaco will likely require substrate bias control capability, such that when a functional region of the chip is in an inactive state, back-bias can be enabled for that block. The back bias enables are created by logical AND-ing of the Back Bias Control Register enable bit for that respective block, with the its clock enable, such that the back bias state is realized when the clock to that functional block is disabled.

For Monaco, the following regions have been identified as having separate back-bias enables:

- System RAM
- L2 RAM
- DSP and L1 RAM
- MCU
- DSP Peripheral Group
- MCU PBUS Peripheral Group

The logic in the TEP, RTC, System and Peripheral PLLs, and IOs cannot be back-biased.

The back-bias enable time should be $< 1\ \mu\text{s}$. However, the critical timing parameter is the transition time from the back-biased state, to the normal operational state. This time needs to be $< 100\text{ns}$.

Editor's note: Add Transition V/T curve diagram here...

Opens:

- *Benefit vs. cost/risk of back-bias?*

12. Emulation

Lots of changes to DSP emulation, discussion of co-emulation.

13. Debug and Type Approval

14. System Overview

Lots of changes here.

14.1 Memory Maps

The following diagrams detail the memory map, as seen from the MCU, DSP, and DMA controllers perspectives.

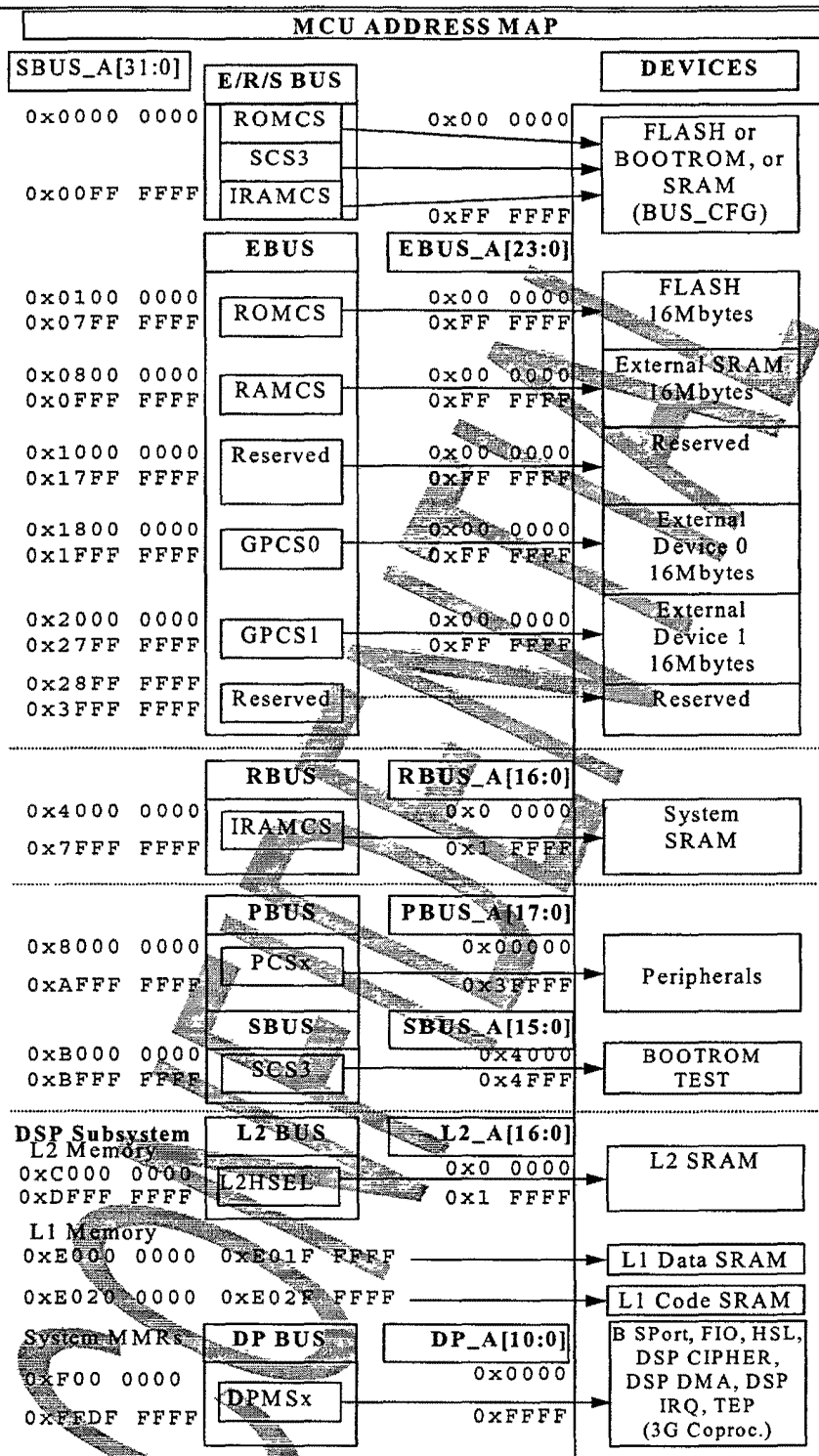


Figure 38 MCU Memory Map

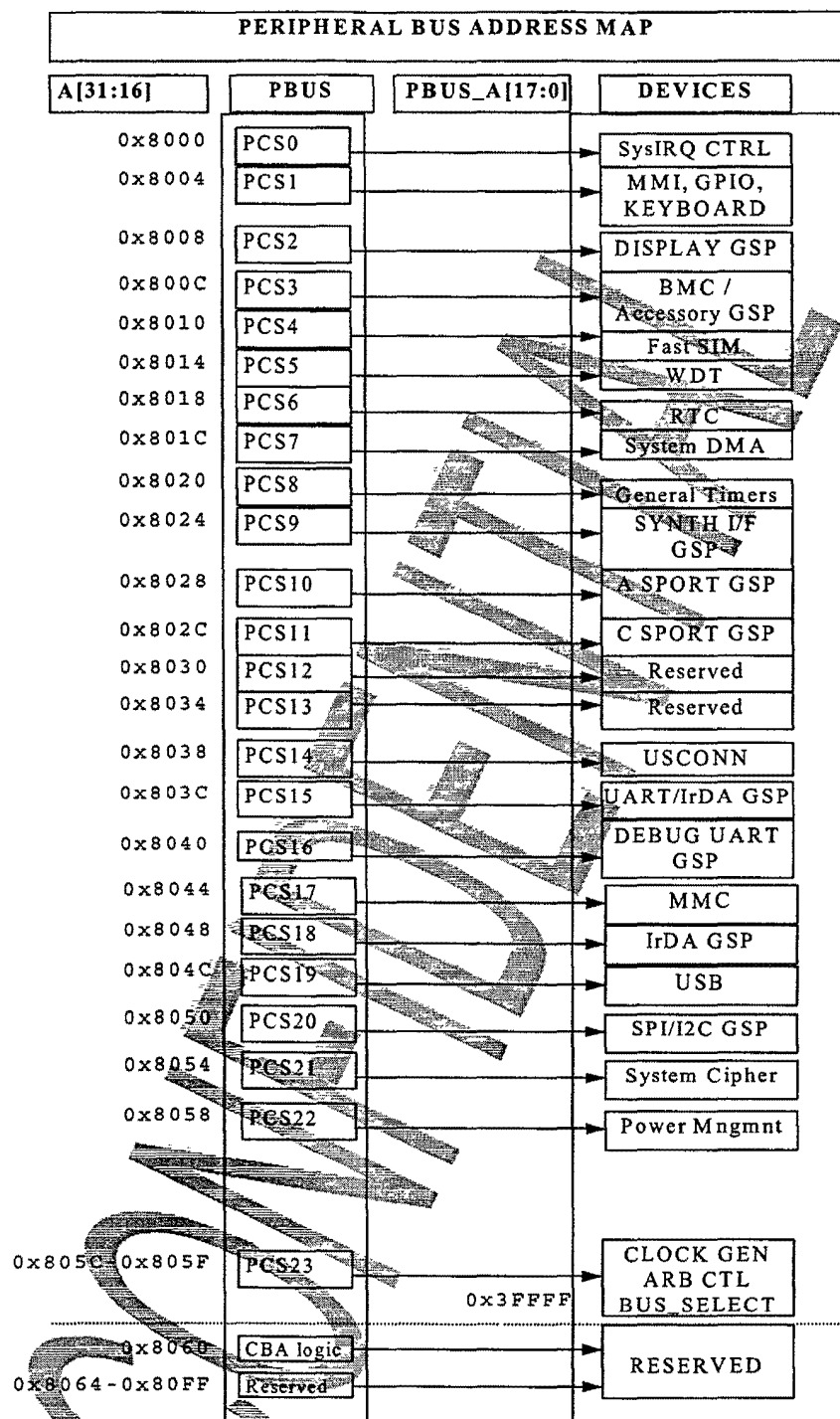


Figure 39 P Bus Slave Memory Map

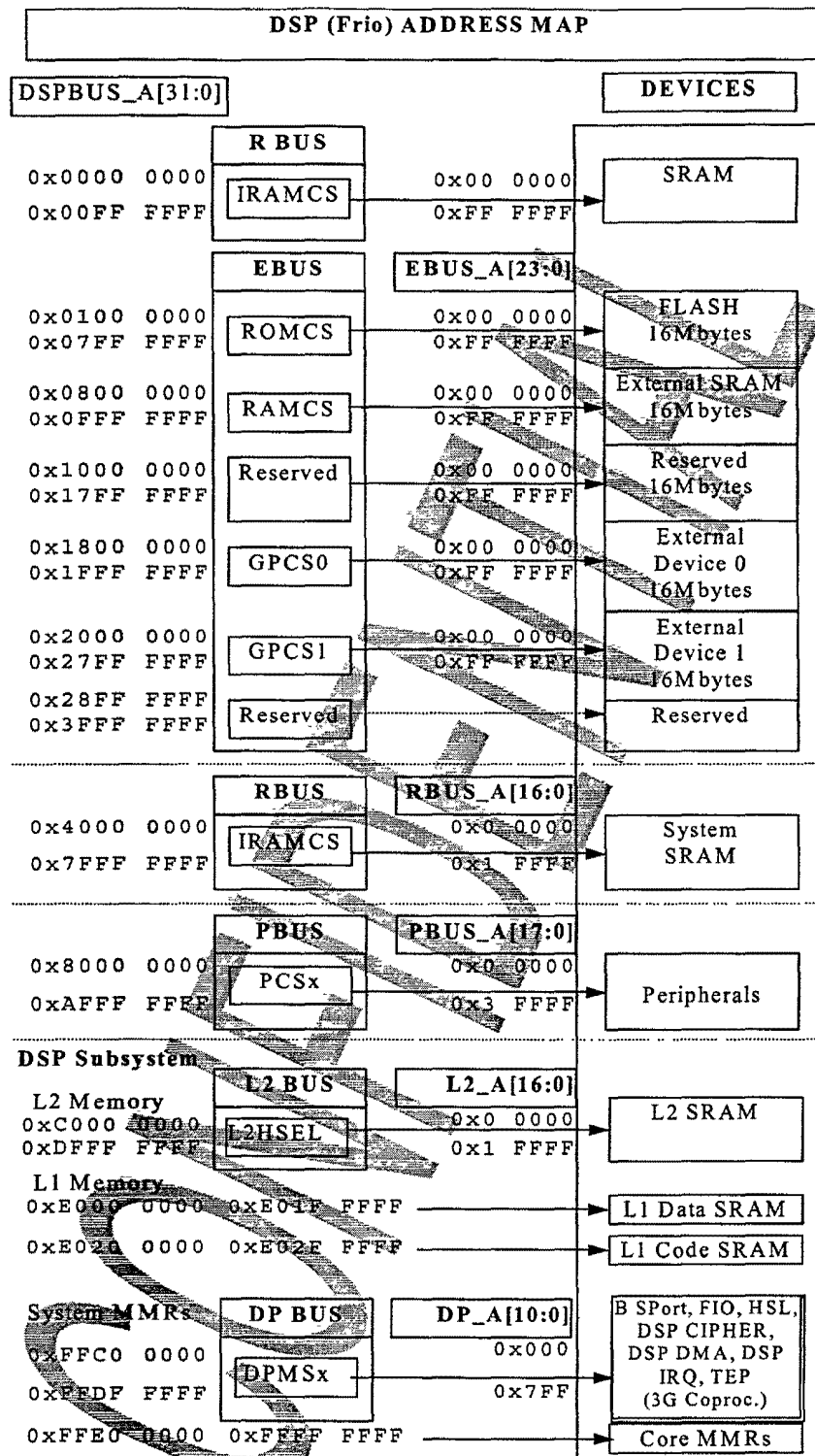


Figure 40 DSP Memory Map

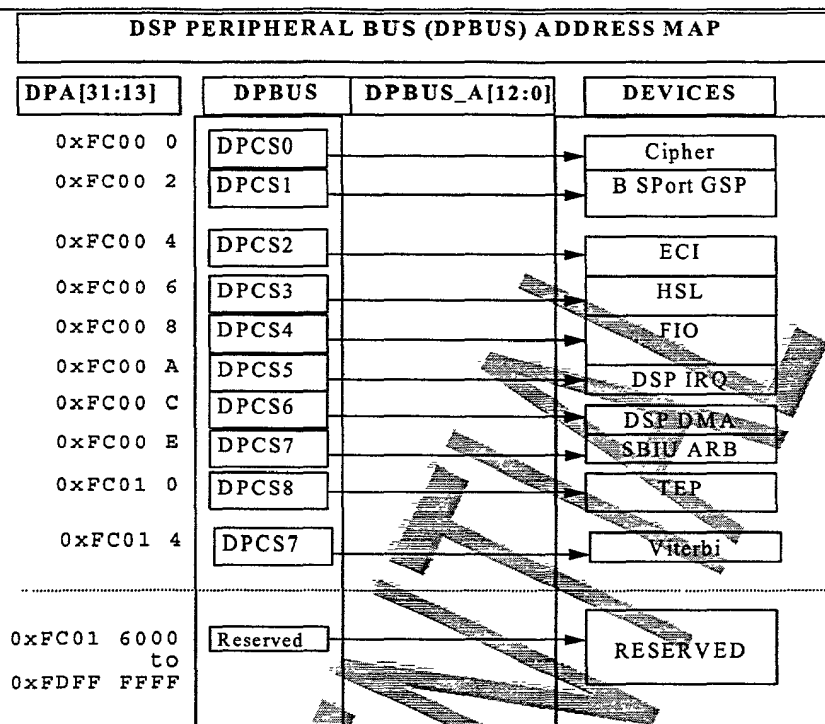


Figure 41 DP Bus Slave Memory Map

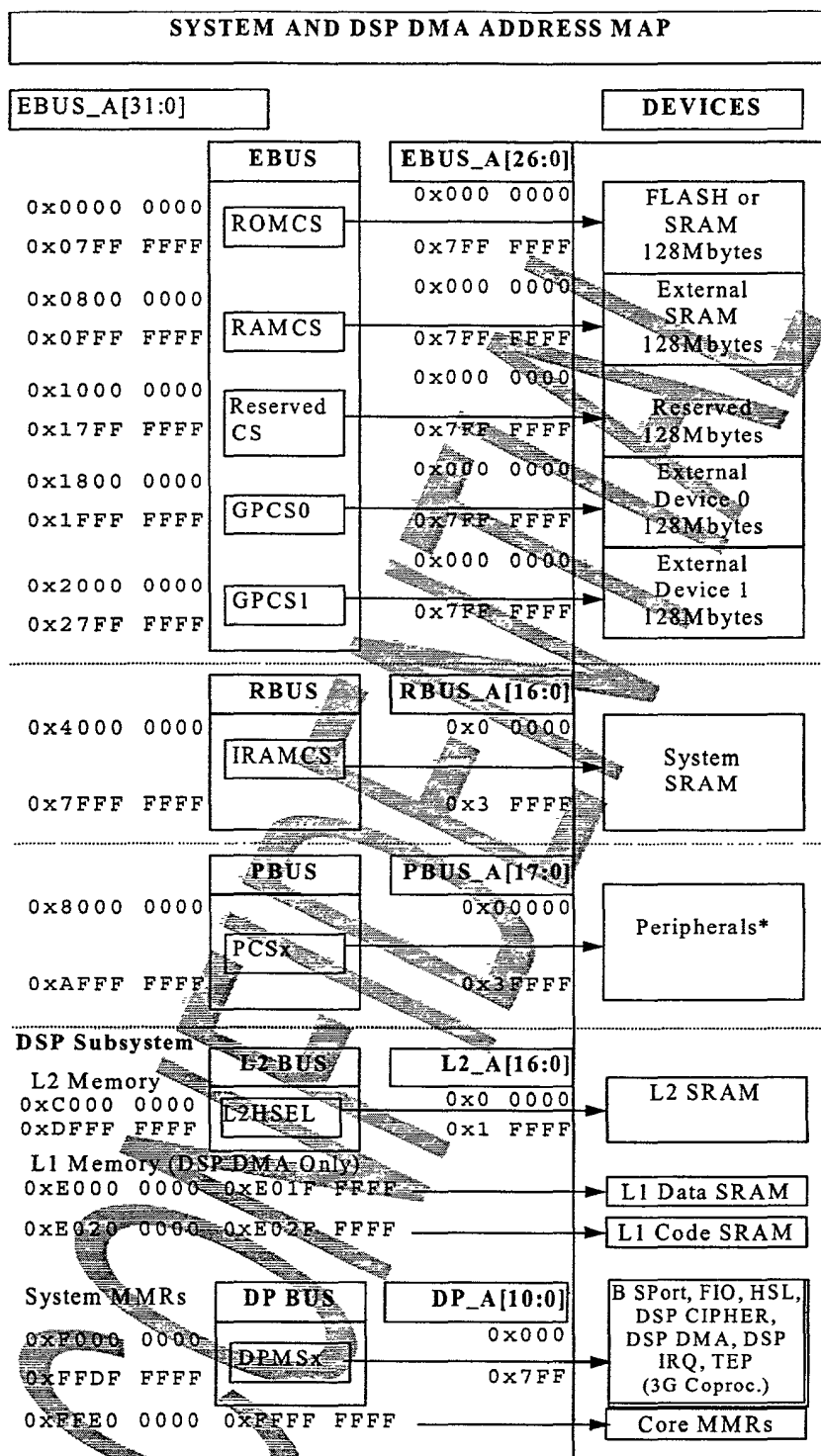


Figure 42: System DMA and DSP DMA Memory Map

15. Oscillator Specification

No change from the Hercules specification.

16. Vertical Function Descriptions

Add examples of:

MP SW synchronization through L2/SysRAM

17. Pin Out

Pinout does change from Hercules

18. JTAG and Boundary Scan

TBD

19. Manufacturing Test

TBD

20. Bootcode ROM Listing

TBD

21. Packaging

21.1 Production Pinout

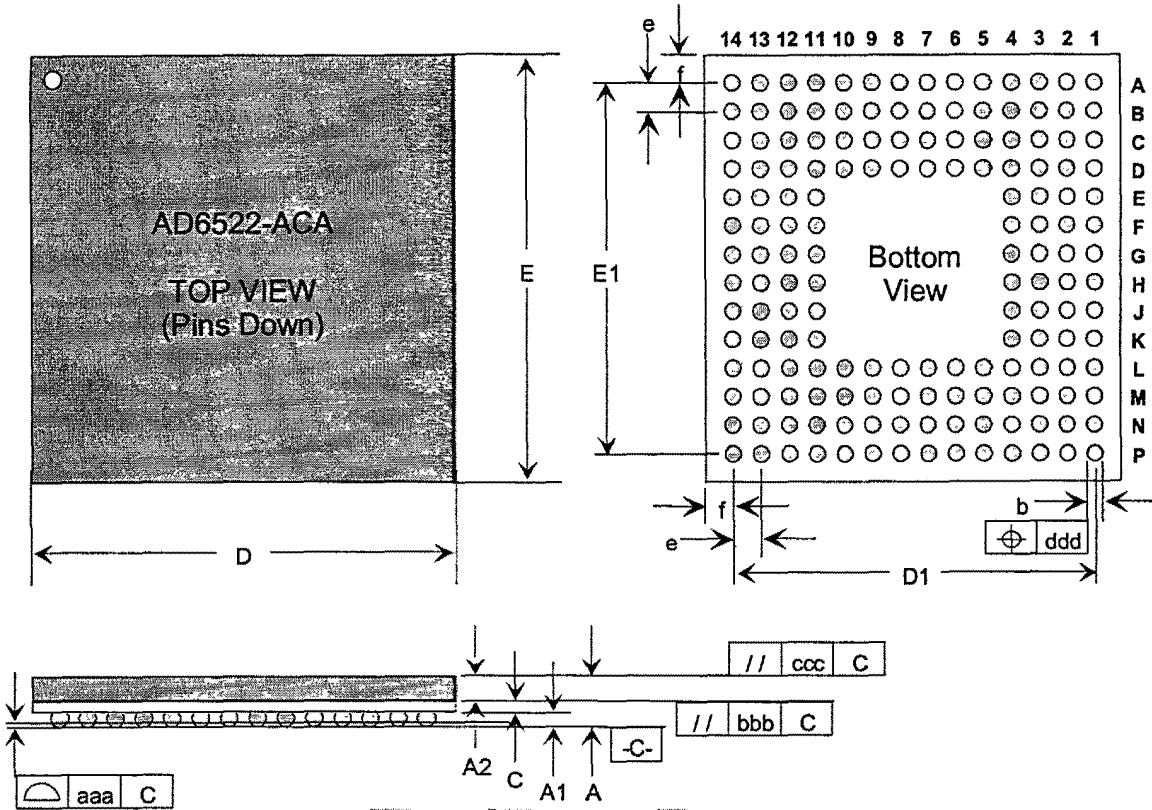
TBD

21.2 Extended Bondout Package

Monaco is also available in extended bond-out, PGA package; pinout is TBD

21.3 160 Pin MINI-BGA Outline Dimensions

Monaco production package will be higher ball count than this! This is a placeholder only!



Dim	MILLIMETERS			INCHES		
	Min.	Nom.	Max.	Min.	Nom.	Max.
A	1.35	1.45	1.55	0.05315	0.05709	0.06102
A1	0.35	0.40	0.45	0.01378	0.01575	0.01772
A2	0.65	0.70	0.75	0.02559	0.02756	0.02953
D	11.80	12.00	12.20	0.46457	0.47244	0.48031
D1	10.40 BSC			0.40945 BSC		
E	11.80	12.00	12.20	0.46457	0.47244	0.48031
E1	10.40 BSC			0.40945 BSC		
b	0.45	0.50	0.55	0.01772	0.01969	0.02165
c		0.35			0.01378	
e	0.725	0.80	0.875	0.02854	0.03140	0.03445
f	0.70	0.80	0.90	0.02756	0.03150	0.03543
aaa			0.15			0.00591

bbb			0.20			0.00787
ccc			0.25			0.00984
ddd			0.20 dia			0.00787 dia

Notes: BSC - Between Spacing Centers

Opens:

- *Currently, we believe the impedance of the bond wires must be eliminated to provide solid power to the device. This dictates flip-chip bonding to the package.*
- *Can mini-BGA support flip-chip die bonding?*

22. DC and AC Characteristics

TBD

----- END -----