

ASYNCHRONOUS INTERFACE FOR LOCALLY CLOCKED MODULES IN ULSI SYSTEMS

Pasi Liljeberg, Juha Plosila, and Jouni Isoaho

University of Turku, Department of Applied Physics
Laboratory of Electronics and Information Technology
FIN-20014 Turku, Finland
{pakrli | juplos | jisoaho}@utu.fi

ABSTRACT

This paper describes an approach to construct globally asynchronous locally synchronous ULSI systems. The architecture employs interface units which communicate asynchronously with each other and synchronously with locally clocked modules. Interface units feature elastic burst-mode data transfer to reduce the number of synchronizations. The architecture allows the use of different clocks in synchronous modules and makes thereby modular design possible.

1. INTRODUCTION

The distribution of a high frequency clock signal will be an essential problem in the future ULSI systems containing hundreds of millions of transistors. As the clock frequency goes higher the clock skew and system level timing will develop into even more important issues as they are in today's systems. Also, the system-on-chip design strategy brings new challenges to manage system level timing and to arrange an efficient interaction between separate modules.

A viable alternative to overcome the problems of a globally clocked system is the globally asynchronous locally synchronous approach [1, 4], where each distinct module of the system is synchronized to an optimized local clock, and communication between these modules is arranged asynchronously. As a solution, we present in this paper an interface design which manages asynchronous data transfer and communication between synchronous modules. To accomplish this, each synchronous module is equipped with an interface unit which transfers data and control signals from asynchronous domain to synchronous domain and vice versa. Each of these synchronous modules can use an independent local clock which is unrelated to the other clocks in the system. Therefore, the problem of global clocking and clock skew are not any more system level design issues. Instead, the clock skew and routing problems are limited only to a locally synchronous module which is relatively small compared to the whole system. This simplifies the design process of a complex system enabling easy re-usage

of different synchronous modules. Furthermore, the asynchronously controlled stoppable clocks make the automatic power-down operation of currently idle blocks possible [5].

2. SYSTEM ARCHITECTURE

Every locally synchronous module in the proposed architecture is equipped with a synchronous/asynchronous interface unit (*SAInt*) [3] which takes care of inter-block communication working as a "wrapper" around a synchronous module. *SAInt* has an asynchronous external interface with other system blocks, i.e., other *SAInt* units, and a synchronous internal interface with the local module. The structure of a system with one master module and two server modules is depicted in Figure 1.

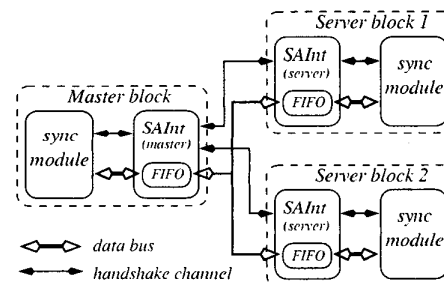


Figure 1: Architecture of system with *SAInt* units.

Two different types of interface units are needed to construct a flexible and fast communication network. A *master-type* unit initiates data transmission and fetching. A *server-type* unit, in turn, basically just responds to the action requested by the master, receive or send, and informs the master its current status: ready, busy or idle.

The asynchronous interface between two modules is a 4-phase handshake channel [2] which consists of a bidirectional data bus, two status signals indicating that the last

data word is sent to or has been fetched from the server party, and three request and four acknowledgement signals which are used to initialize communication and control data transfer between the master and server units.

A master-type *SAInt* can have any number of server units to which it allocates different operations in a sequential fashion. As soon as a server receives its input data, it starts to operate in parallel with the other servers. Correspondingly, a server type interface unit can be connected to arbitrarily many master units via a set of arbitration circuits. In the proposed architecture, the frequency of the local clock in a server party can be optimized for the involved synchronous module. The system can also be configured to use a pauseable ring oscillator-type [4, 7] clock source.

The system structure presented in this paper is targeted to ULSI systems, where each involved locally synchronous module requires a large number of input data words per operation, and produces a large number of output data words. Furthermore, it is assumed that the latency of each operation is several clock cycles.

3. INTERFACE UNITS

The interface units contain a synchronous controller *Synctl* and an asynchronous controller *Asynctl*. The latter takes care of the external interface with another *SAInt* unit, while the synchronous unit controls internal communication with the local synchronous module. A dual-port RAM is used as a FIFO queue for bidirectional data transfer. The ports of this memory can be accessed simultaneously. One of them is controlled asynchronously by *Asynctl*, the other synchronously by *Synctl*. The controllers have the related counters *Scount* and *Account* which provide the required addresses for the memory access. In order to prevent FIFO queue overruns and underruns, *Synctl* has to be able to read the value of *Account*, and *Asynctl* has to be able to read the value of *Scount*. To accomplish this, the write and read accesses to the auxiliary counter register, containing a past value of the asynchronous counter *Account*, take place through an arbiter unit. The *Asynctl* can read the value of *Scount* through a synchronizer.

The server unit, depicted in Figure 2, contains also a clock control unit *Clkctl* which pauses the local clocks of the interface unit and local synchronous module if no data is available or the module is idle. The unit executes repeatedly three successive tasks. In the (1) *data receiving mode*, data provided by a master-type *SAInt* unit of another system block, are transferred to the local synchronous module through the FIFO queue. The asynchronous counter *Account* generates the write addresses for the memory, the synchronous counter *Scount* the read addresses, respectively. In the (2) *computation mode*, *SAInt* awaits the clocked module to complete its data processing. In the (3) *data trans-*

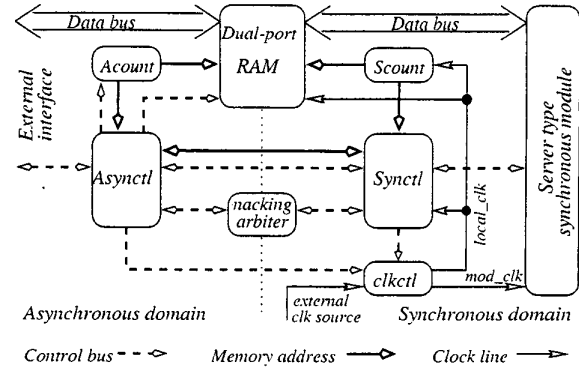


Figure 2: Server-type *SAInt* architecture

mission mode, the computation results are transferred to the master system block through the FIFO queue. The roles of the counters are now interchanged. We call the implemented flexible data transfer protocol between the asynchronous and synchronous domains the *elastic burst-mode* transfer, where the data words are transferred in variable-length bursts. The length of a write or read burst depends on the size of the FIFO queue and the difference between the speeds of the controllers. The fundamental property of the elastic burst-mode transfer is that the controllers interact only when it is absolutely necessary, i.e., when the FIFO queue appears to be empty or full. This improves performance and reliability, because the number of time-consuming synchronizations and thereby the rate of possible synchronization failures are reduced.

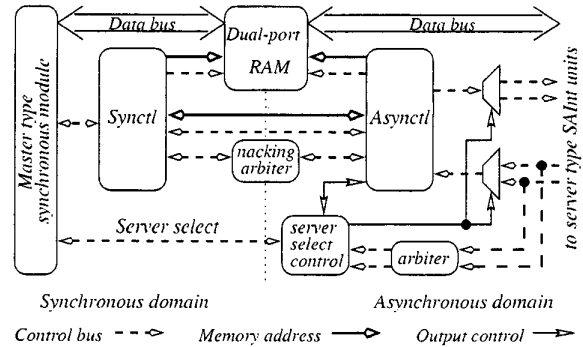


Figure 3: Master-type *SAInt* architecture

The master executes basically the same three tasks as the server, but in a different order. First it awaits activation

from the local module, transmits then data to the server, and finally fetches data from the server, when there is data available and the master is inclined to do so. The structure of a master, which can be connected to two server units, is depicted in Figure 3. The memory address counters have been included in the related control units, *Synctl* and *Asynctl*.

The master controls the shared bidirectional data bus and allocates operations to the server units. The *server select controller* is used to select an appropriate server unit to be activated. The order of the incoming acknowledgement signals sent by the servers, and thereby the data source of the fetch procedure, can be chosen in two different ways. In the “first-ready first-served” approach, the acknowledgement signals are arbitrated, and the result of the arbitration determines which data source is to be selected. On the other hand, if the incoming data is needed in a specific order, the multiplexers can be easily forced to the appropriate state, and the unit is suspended until the required acknowledgement arrives.

4. CMOS IMPLEMENTATION

The *SAInt* units have been designed using Alcatel 0.35 μm technology in the Cadence 4.4.3. IC design environment. The synchronous part of the system and the 32x16-bit dual-port RAM has been designed with VHDL, and the asynchronous part by mixing VHDL and transistor level design. The total transistor count of the server-type *SAInt* which is connected only to a single master is 18482 of which 16088 is required by the dual-port memory and bidirectional bus-related multiplexers and tristate buffers. Hence, the control logic contains 2394 transistors. The size of the master-type *SAInt* is approximately the same, if it is connected only to one server unit. However, the transistor count of a *SAInt* unit increases, due to the multiplexing and arbitration circuits, if multiple units are connected together.

The implementation of the clock control unit of the server-type *SAInt* is presented in Figure 4. *Asynctl* activates this unit by sending an asynchronous request signal in parallel with writing the first data word to the FIFO queue. We have designed a customized pulse-driven synchronizer circuit which maximizes the metastability recovery time between two consecutive sampling points. If both address counter values are equal in receive and transmission mode, indicating that the FIFO queue is empty or full, *local_clk* and *mod_clk* are paused. Hence, at every rising edge of *mod_clk*, there is data available.

The arbitration circuitry, which is required to provide mutual exclusion between possible concurrent requests from *Asynctl* and *Synctl* to access the asynchronous address counter, is depicted in Figure 5. The interface unit employs a one-channel nacking arbiter to speed up the memory read and write sequences. One of the ports of this arbiter returns

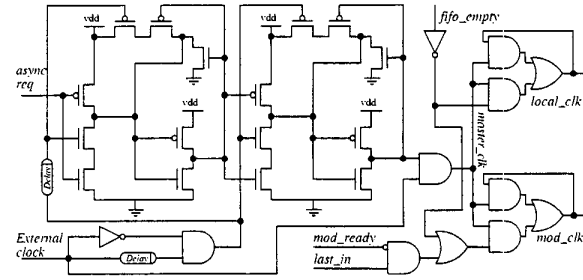


Figure 4: Clock control unit.

a negative acknowledgement, *nack_a*, in the case when the arbiter is already reserved by the other channel, and a positive acknowledgement, *ack_a*, if access is granted to *Asynctl*. The other port, connected to *Synctl*, works as a normal arbiter port. The nacking property can be disabled by a mode signal, in which case both ports operate as normal arbiter ports and no negative acknowledgements will be generated. The one-channel nacking arbiter is constructed by combining two mutual exclusion element (MUTEX) circuits [6] and modifying the state holding input stages.

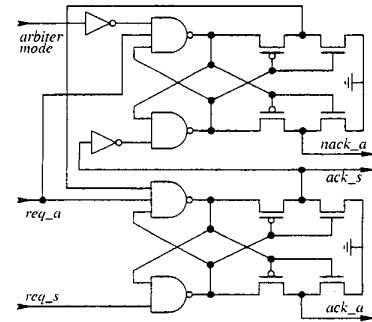


Figure 5: One-channel nacking arbiter.

The use of the nacking arbiter significantly improves the performance of the asynchronous domain. When the arbiter sends an acknowledgement to *Asynctl*, this can store the current value of the address counter *Acount* to a local register and start a new FIFO write or read procedure. In the case of a negative acknowledgement, it does not update the register, because the synchronous controller is reading this. However, *Asynctl* can start a new FIFO access procedure immediately. Hence, the nacking arbiter enables parallel operation of the controllers without interrupts as long as the FIFO is not full or empty.

5. SIMULATION RESULTS

Extensive SPICE-level simulations were performed with the Cadence Analog Artist tool. Typical transient response waveforms of some control lines of the server-type *SAInt* in the data receiving mode are depicted in Figure 6 where the elastic burst-mode data transfer can be seen. One write burst and two read bursts are shown. The write burst is represented by the topmost signal which is the asynchronous request for memory access. A new data word is stored during each pulse. At the right end of the topmost signal the beginning of the next write burst can be seen. The second signal is the request from *Asynctl* to update the asynchronous memory address register. The third and fourth signals are the acknowledgement and negative acknowledgement to *Asynctl*, generated by the nacking arbiter. The next two signals are the synchronous request and acknowledgement signals which are used by *Synctl* to fetch the stored value of *Account* by communicating with the arbiter. They also separate read bursts, since new data cannot be read before a valid address has been fetched. The last signal, *mod_clk*, is the stoppable clock of the locally synchronous module. One data word is transferred in each clock cycle. The clock is paused when the FIFO is empty in the receiving mode or full in the transmission mode. It is derived from *master_clk* which is the intermediate signal generated by the clock control unit shown in Figure 4.

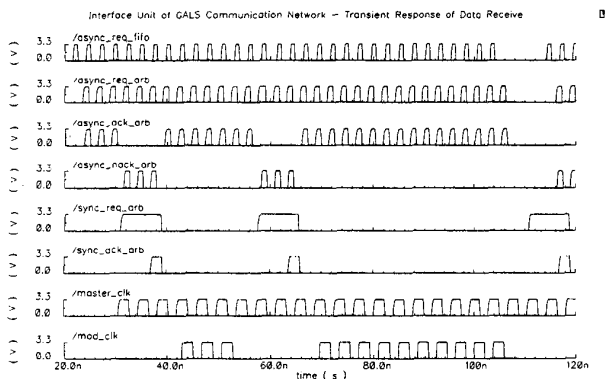


Figure 6: Simulation of elastic burst-mode transfer.

The maximum speed of the FIFO access in the asynchronous domain is 380 MHz at the current Alcatel 0.35 μm technology. The maximum clock frequency for the synchronous domain of *SAInt* is 420 MHz. The speed is restricted mainly by the relatively slow unoptimized memory. The size of the memory is another significant factor. If a small well-optimized FIFO queue is used, the control logic restricts the clock frequency.

6. CONCLUSION

We have presented an asynchronous design approach to interface locally clocked modules in a ULSI system. In this method, data is transferred asynchronously in variable-length packets between concurrently operating synchronous modules. The size of the FIFO queue of the interface units can be easily changed by just replacing the memory and increasing/decreasing the address counter widths, while the control logic can be left untouched.

The presented architecture with server- and master-type interface units allows a feasible way to construct a globally asynchronous locally synchronous system. This enables truly modular design and re-usage of synchronous modules, since no global timing assumptions have to be made. Furthermore, the physical timing problems are made local, and thereby the simulation and verification tasks are simplified. The proposed heterogeneous design method is targeted for systems with large locally synchronous modules each of which has an operation time of several clock cycles. The approach offers a viable method to reduce problems that arise in globally clocked ULSI systems and system-on-chip design.

7. REFERENCES

- [1] D. S. Bormann and P. Y. K. Cheung. Asynchronous Wrapper for heterogeneous Systems. In *Proc. of the IEEE ICCD'97*, pp. 307-314, 1997.
- [2] A. Davis and S.M. Nowick. Asynchronous circuit design: motivation, background and methods. In G. Birtwistle and A. Davis, editors, *Asynchronous Digital Circuit Design*, pages 1-49. Springer, 1995.
- [3] P. Liljeberg, J. Plosila and J. Isoaho. Synchronous/Asynchronous Interface Unit for IP based SoC Systems. In *Proc. of the 18th IEEE NORCHIP 2000 conference*, November 2000.
- [4] J. Muttersbach, T. Villiger and W. Fichtner. Practical Design of Globally-Asynchronous Locally- Synchronous Systems. In *Proc. of the IEEE ASYNC'2000*, pp. 52-59, April 2000.
- [5] L. Plana and S. Nowick. Architectural Optimization for Low-Power Nonpipelined Asynchronous Systems. In *IEEE Trans. on VLSI Systems*, 6(1) March 1998.
- [6] C. L. Seitz. System timing. Chapter 7 in Mead and Conway, editors, *Introduction to VLSI Systems*, Addison-Wesley, 1980.
- [7] K. Y. Yun and R. P. Donohue. Pausible Clocking: A First Step Towards Heterogeneous Systems. In *Proc. of IEEE ICCD'96*, 1996.