

IN THE UNITED STATES DISTRICT COURT
FOR THE WESTERN DISTRICT OF TEXAS
WACO DIVISION

BiTMICRO LLC,

Plaintiff,

v.

KIOXIA AMERICA, INC.
And KIOXIA CORPORATION,

Defendants.

Case No. 6:22-cv-00331-ADA

BiTMICRO LLC,

Plaintiff,

v.

INTEL CORPORATION,

Defendant.

Case No. 6:22-cv-00335-ADA

DEFENDANTS' OPENING CLAIM CONSTRUCTION BRIEF

TABLE OF CONTENTS

	Page
I. LEGAL STANDARD.....	1
II. '939 PATENT.....	1
A. Technical Background	1
B. “predetermined level”	2
C. “means for deactivating the plurality of super capacitors to cut off power to the computing engine based upon the plurality of super capacitors discharging to a predetermined level”	5
D. “reversing the flow of current between the computing engine and the plurality of super capacitors”	9
E. “means for reversing the flow of current between the computing engine and the plurality of super-capacitors”	11
III. '243 PATENT.....	12
A. Technical Background	12
B. “passive port”	13
C. “a plurality of modules each comprising”	15
IV. '740 PATENT.....	18
A. Technical Background	18
B. “optimized memory operations”	19
C. “optimally distributed”	22
D. “memory store”	23
V. '190 PATENT.....	26
A. Technical Background	26
B. “optimal”	27
VI. '084 AND '694 PATENTS.....	30
A. Technical Background	30

TABLE OF CONTENTS
(continued)

	Page
B. “Direct Memory Access (DMA) descriptors” / “DMA controller descriptor”	31
VII. CONCLUSION.....	33

TABLE OF AUTHORITIES

	Page(s)
CASES	
<i>Allen Eng’g Corp. v. Bartell Indus., Inc.</i> , 299 F.3d 1336 (Fed. Cir. 2002).....	9, 11
<i>Apple, Inc. v. Ameranth, Inc.</i> , 842 F.3d 1229 (Fed. Cir. 2016).....	24
<i>Arendi S.A.R.L. v. LG Elecs., Inc.</i> , No. 12-1595-LPS, 2019 WL 3891150 (D. Del. Aug. 19, 2019).....	6
<i>Bell N. Research, LLC v. Coolpad Techs., Inc.</i> , No. 18-CV-1783-CAB-BLM, 2019 WL 3766688 (S.D. Cal. Aug. 9, 2019)	8, 9, 12
<i>Bio-Rad Labs., Inc. v. 10X Genomics Inc.</i> , 967 F.3d 1353 (Fed. Cir. 2020).....	18
<i>Biogen MA Inc. v. EMD Serono, Inc.</i> , 976 F.3d 1326 (Fed. Cir. 2020).....	15
<i>BookIT Oy v. Bank of Am. Corp.</i> , 817 F. App’x 990 (Fed. Cir. 2020)	13
<i>Chewy Inc. v. IBM Corp.</i> , 571 F. Supp. 3d 133 (S.D.N.Y. 2021).....	21, 22
<i>Cont’l Circuits LLC v. Intel Corp.</i> , 915 F.3d 788 (Fed. Cir. 2019).....	15
<i>Dig. Retail Apps, Inc., v. H-E-B, LP</i> , No. 6-19-CV-00167-ADA, 2020 WL 376664 (W.D. Tex. Jan. 23, 2020)	5, 8, 12
<i>Eon Corp. IP Holdings v. Silver Spring Networks, Inc.</i> , 815 F.3d 1314 (Fed. Cir. 2016).....	4
<i>Ergo Licensing, LLC v. CareFusion 303, Inc.</i> , 673 F.3d 1361 (Fed. Cir. 2012).....	6, 8, 12
<i>Ex Parte Gittleman</i> , 2008 WL 696137 (B.P.A.I. 2008).....	9, 11
<i>GE Lighting Sols., LLC v. AgiLight, Inc.</i> , 750 F.3d 1304 (Fed. Cir. 2014).....	14

TABLE OF AUTHORITIES
(continued)

	Page(s)
<i>In re Cohn</i> , 438 F.2d 989 (C.C.P.A. 1971)	9, 11
<i>In re Donaldson Co.</i> , 16 F. 3d 1189 (Fed. Cir. 1994).....	8, 12
<i>In re Fought</i> , 941 F.3d 1175 (Fed. Cir. 2019).....	17
<i>In re Horvitz</i> , 168 F.2d 522 (C.C.P.A. 1948)	24
<i>Intellectual Ventures I LLC v. T Mobile USA, Inc.</i> , No. 2:17-CV-00577-JRG, 2018 WL 5809267 (E.D. Tex. Nov. 6, 2018).....	29
<i>Intellectual Ventures I LLC v. T-Mobile USA, Inc.</i> , 902 F.3d 1372 (Fed. Cir. 2018).....	27
<i>Interval Licensing LLC v. AOL, Inc.</i> , 766 F.3d 1364 (Fed. Cir. 2014).....	27
<i>Kyocera Senco Indus. Tools Inc. v. Int’l Trade Comm’n</i> , 22 F.4th 1369 (Fed. Cir. 2022)	4, 15
<i>Medtronic, Inc. v. Advanced Cardiovascular Sys., Inc.</i> , 248 F.3d 1303 (Fed. Cir. 2001).....	5
<i>Merck & Co. v. Teva Pharms. USA, Inc.</i> , 347 F.3d 1367 (Fed. Cir. 2003).....	3
<i>Nautilus, Inc. v. Biosig Instruments, Inc.</i> , 572 U.S. 898 (2014).....	1, 19
<i>Nazomi Commc’ns, Inc. v. Arm Holdings, PLC</i> , 403 F.3d 1364 (Fed. Cir. 2005).....	23
<i>Nichia Corp. v. Document Sec. Sys., Inc.</i> , 2022 WL 1218036 (Fed. Cir. Apr. 26, 2022)	15
<i>Phillips v. AWH Corp.</i> , 415 F.3d 1303 (Fed. Cir. 2005).....	1, 2, 21, 23
<i>Pi-Net Int’l Inc. v. JPMorgan Chase & Co.</i> , No. 12-282-SLR, 2014 WL 1997039 (D. Del. May 14, 2014).....	6

TABLE OF AUTHORITIES
(continued)

	Page(s)
<i>Pitney Bowes, Inc. v. Hewlett-Packard Co.</i> , 182 F.3d 1298 (Fed. Cir. 1999).....	18
<i>Portus Singapore Pte. Ltd. v. Ecobee, Inc.</i> , No. 6:20-cv-00694-ADA, ECF No. 36, slip op. (W.D. Tex. Jul. 23, 2021).....	15
<i>Renishaw PLC v. Marposs Societa per Azioni</i> , 158 F.3d 1243 (Fed. Cir. 1998).....	4
<i>Sinorgchem Co., Shandong v. Int’l Trade Comm’n</i> , 511 F.3d 1132 (Fed. Cir. 2007).....	14
<i>Tech. Properties Ltd. LLC v. Huawei Techs. Co.</i> , 849 F.3d 1349 (Fed. Cir. 2017).....	14
<i>Vitronics Corp. v. Conceptronic, Inc.</i> , 90 F.3d 1576 (Fed. Cir. 1996).....	2
<i>Williamson v. Citrix Online, LLC</i> , 792 F.3d 1339 (Fed. Cir. 2015).....	5
STATUTES	
35 U.S.C. § 112.....	1, 13, 27
35 U.S.C. § 112(2).....	9
35 U.S.C. § 112(6).....	5, 11

Defendants KIOXIA America, Inc. and KIOXIA Corporation (together, “KIOXIA”) and Intel Corporation (“Intel”) (collectively, “Defendants”) hereby submit their Opening Claim Construction Brief for the disputed terms of U.S. Patent Nos. 6,496,939 (“the ’939 patent”), 8,010,740 (“the ’740 patent”), and 9,135,190 (“the ’190 patent”). Intel further submits its Opening Claim Construction Brief for the disputed terms of U.S. Patent Nos. 7,826,243 (“the ’243 patent”), 9,858,084 (“the ’084 patent”), and 10,120,694 (“the ’694 patent”).¹

I. LEGAL STANDARD

Claim terms are given their customary meaning as understood by a person of ordinary skill in the art. *Phillips v. AWH Corp.*, 415 F.3d 1303, 1312-13 (Fed. Cir. 2005). To determine that meaning within the field of art, a court should look to “the words of the claims themselves, the remainder of the specification, the prosecution history, and extrinsic evidence concerning relevant scientific principles, the meaning of technical terms, and the state of the art.” *Id.* at 1314. A claim is invalid for indefiniteness under 35 U.S.C. § 112, ¶ 2 when, read in light of the specification and the prosecution history, it “fail[s] to inform, with reasonable certainty, those skilled in the art about the scope of the invention.” *Nautilus, Inc. v. Biosig Instruments, Inc.*, 572 U.S. 898, 901 (2014).

II. ’939 PATENT

A. Technical Background

The ’939 patent relates to systems and methods for controlling data in a computer system when the computer system loses power. ’939 patent at 1:8-11. As the ’939 patent explains, computer systems typically comprise two types of memory: (1) volatile “cache” memory, which is erased when external power is shut down, and (2) non-volatile memory, which preserves data when external power is removed. *Id.* at 1:15-33. Since volatile cache memory is much faster than

¹ The ’243 patent, the ’084 patent, and the ’694 patent are not asserted against KIOXIA.

non-volatile memory, data is typically written to cache memory before being transferred to non-volatile memory. *Id.* However, when the computer system loses external power, there is insufficient time to transfer all newly written data from the volatile cache memory to the non-volatile memory, thereby creating a risk that data will be lost. *Id.* at 1:34-39. Accordingly, the '939 patent identifies an alleged need to ensure that all newly written data to the volatile memory is safely transferred to non-volatile memory when external power is lost. *Id.* at 2:13-16.

The '939 patent purports to address this need by disclosing the use of “super-capacitors”—which operate as a “short term power source”—to provide sufficient energy to allow data to be transferred from volatile memory to non-volatile memory before the computer system shuts down. *Id.* at 2:27-31; 3:3-5. To accomplish this function, the '939 patent explains that, upon loss of external power, the super capacitors are “activated” to maintain the internal power of the system. *Id.* at 5:8-11. Using this power supply, data is then transferred from the volatile memory to the non-volatile memory to ensure that it is preserved. *Id.* at 5:33-36. Once the super capacitors have discharged to a predetermined level—a preset minimum operating voltage necessary for proper operation, in which the super capacitors are not fully discharged—power is cut off from the super capacitors to the computer system. *Id.* at 5:39-43; 5:48-52; 6:49-53.

B. “predetermined level”

Patent	Plaintiff’s Proposal	Defendants’ Proposal
'939 patent: Claims 1, 10	Plain and ordinary meaning; not indefinite.	“a preset minimum operating voltage necessary for proper operation, in which the super capacitors are not fully discharged”

“[T]he specification is always highly relevant to the claim construction analysis.” *Vitronics Corp. v. Conceptronic, Inc.*, 90 F.3d 1576, 1582 (Fed. Cir. 1996). “Usually, it is dispositive; it is the single best guide to the meaning of a disputed term.” *Phillips*, 415 F.3d at 1315; *id.* at 1316

(“[T]he specification necessarily informs the proper construction of the claims.”); *Merck & Co. v. Teva Pharms. USA, Inc.*, 347 F.3d 1367, 1371 (Fed. Cir. 2003) (“[C]laims must be construed so as to be consistent with the specification, of which they are a part.”) (citations omitted). Here, Defendants’ proposed construction should be adopted because the specification makes clear that the “predetermined level” is “a preset minimum operating voltage that is necessary for proper operation, in which the super capacitors are not fully discharged.”

In particular, the specification explains that the “predetermined level” is a preset minimum operating voltage at which “the down-converter has **insufficient voltage differential to continue proper operation** and shuts down.” ’939 patent at 5:39-43 (emphasis added); *see also id.* at 6:40-44; 5:44-48 (explaining that voltage cannot be allowed to fall below “minimum operating voltage”); 5:48-52; 6:49-53 (explaining that super capacitors should not be “fully discharged”). The specification also demonstrates why these parameters are critically important, explaining that if the voltage “were allowed to **fall below the minimum operating voltage** of the computer system, the computer system could continue to run with **unpredictable and potentially serious results.**” *Id.* at 5:44-48 (emphases added); 6:45-49. This would seriously undermine the ’939 patent’s stated goals of ensuring that “a user is able to correctly store large amounts of newly written and modified data” and “rapidly and irretrievably erase data.” *Id.* at 2:33-41. Further, “by not fully discharging the super-capacitors they will take only a few seconds to recharge once the external system power is restored.” *Id.* at 5:48-52; 6:49-53.

The applicant also relied on these specific characteristics to distinguish the alleged invention during prosecution. As the applicant explained, “performance and reliability of the computer system are improved” by ensuring that the predetermined level is “set above the minimum operative voltage of the computer system, preventing the computer system from running

with unpredictable results.” See Ta Decl.,² Ex. I at 7 (amend. filed on May 8, 2002). The applicant further explained that, by setting the predetermined level “above a zero level of power for the computing system . . . it may take only a few seconds to recharge the super capacitors, rather than several minutes, rapidly providing protection against power loss.” *Id.* The applicant also distinguished the prior art based on these characteristics, emphasizing that “[a]ny combination of [the prior art] . . . would not deactivate super capacitors when the super capacitors are discharged to a predetermined level to cut off power to the computer system.” *Id.* at 8-9.

BiTMICRO’s argument that this term should be afforded a generic “plain and ordinary meaning” is incorrect. *Eon Corp. IP Holdings v. Silver Spring Networks, Inc.*, 815 F.3d 1314, 1321 (Fed. Cir. 2016) (“The question is not whether there is a settled ordinary meaning of the terms in some abstract sense of the words” since “[t]he only meaning that matters in claim construction is the meaning in the context of the patent.”). As explained above, the intrinsic evidence not only confirms that the patentee acted as its own lexicographer in giving this term a specific meaning, but also that the patentee relied on this specific meaning as the critical feature that differentiated the alleged invention from the prior art. As a result, the description of “predetermined level” in the intrinsic record, which is reflected in Defendants’ proposed construction, should be applied. See, e.g., *Kyocera Senco Indus. Tools Inc. v. Int’l Trade Comm’n*, 22 F.4th 1369, 1378 (Fed. Cir. 2022) (finding that the patentee acted as its own lexicographer in clearly defining a term); *Renishaw PLC v. Marposs Societa per Azioni*, 158 F.3d 1243, 1250 (Fed. Cir. 1998) (“The construction that stays true to the claim language and most naturally aligns with the patent’s description of the invention will be, in the end, the correct construction.”).³

² All references to “Ta Decl.” are to the Declaration of Janice L. Ta in Support of this Brief.

³ BiTMICRO’s proposal also states that this term is “not indefinite.” Because Defendants do not contend that “predetermined level” is indefinite, Defendants do not address this assertion.

C. “means for deactivating the plurality of super capacitors to cut off power to the computing engine based upon the plurality of super capacitors discharging to a predetermined level”

Patent	Plaintiff’s Proposal	Defendants’ Proposal
’939 patent: Claim 10	Function: deactivating the plurality of super capacitors to cut off power to the computing engine based upon the plurality of super capacitors discharging to a predetermined level. Structure: Col. 3:3-12, 3:28-4:8, 4:20-35, 4:46-52, 4:64-5:5, 5:35-42, 6:37-44, Figs. 1, 3, 5, of the ’939 Patent, and equivalents thereof.	Indefinite under 35 U.S.C. § 112(6): Lacks corresponding structure.

A means plus function claim element “limits the scope of the functional term ‘to only the structure, materials, or acts described in the specification as corresponding to the claimed function and equivalents thereof.’” *Dig. Retail Apps, Inc., v. H-E-B, LP*, No. 6-19-CV-00167-ADA, 2020 WL 376664, at *3 (W.D. Tex. Jan. 23, 2020) (citing *Williamson v. Citrix Online, LLC*, 792 F.3d 1339, 1347 (Fed. Cir. 2015)). Critically, however, “[a] ‘structure disclosed in the specification is ‘corresponding’ structure only if the specification or prosecution history clearly links or associates that structure to the function recited in the claim.”” *Id.* (citing *Medtronic, Inc. v. Advanced Cardiovascular Sys., Inc.*, 248 F.3d 1303, 1311 (Fed. Cir. 2001)). “The focus of the ‘corresponding structure’ inquiry is not merely whether a structure is capable of performing the recited function, but rather whether the corresponding structure is ‘clearly linked or associated with the [recited] function.”” *Id.* (citing *Medtronic*, 248 F.3d at 1311). In this case, the specification does not clearly link any structure to the recited function of “deactivating a plurality of super-capacitors to cut off power to the computing engine based upon the plurality of super capacitors discharging to predetermined level.” As a result, this claim term is indefinite.

As an initial matter, all instances of “deactivating” described in the specification use generic functional language that simply states a result—*i.e.*, “deactivation” of a component—rather than describing a structure that achieves the result. *Id.* at 5:19-22 (stating that an up-converter “is deactivated”); 6:22-25 (same); Figs. 3 and 5 (same); 5:52-57 (stating that a down-converter “is deactivated”). On that basis alone, the specification fails to identify a sufficiently definite structure. *See Ergo Licensing, LLC v. CareFusion 303, Inc.*, 673 F.3d 1361, 1365 (Fed. Cir. 2012) (affirming indefiniteness where the “specification merely provides functional language” to describe a means plus function term); *Arendi S.A.R.L. v. LG Elecs., Inc.*, No. 12-1595-LPS, 2019 WL 3891150, at *14 (D. Del. Aug. 19, 2019) (holding that a means plus function term was indefinite where the “specification describes the ‘operation’ in terms of a functional result” and not an actual structure); *Pi-Net Int’l Inc. v. JPMorgan Chase & Co.*, No. 12-282-SLR, 2014 WL 1997039, at *9 (D. Del. May 14, 2014) (finding term indefinite where the specification “describes the [term] in functional language . . . [but] is devoid of any disclosure regarding **how** the [term] performs these various functions”).

More importantly, however, the specification does not disclose a structure that deactivates a plurality of super capacitors to cut off power to a computing engine. As explained above, the specification uses functional language to discuss “deactivation” of an up-converter and a down-converter. But annotated Figure 1, shown at right, establishes that the **up-converter** and **down-**

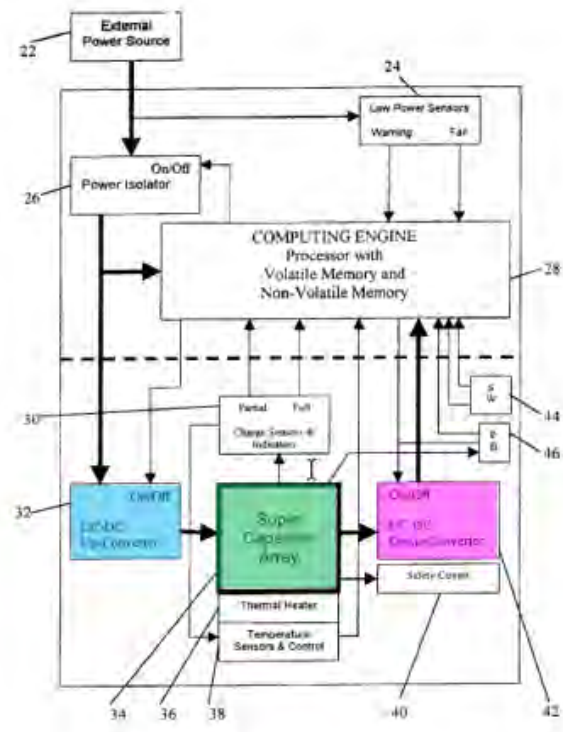


Figure 1

converter are separate and distinct from the **super capacitor array**. '939 patent, Fig. 1. Accordingly, the specification's discussion of deactivating an up-converter and down-converter does not identify a specific structure for deactivating a plurality of **super capacitors** as recited in the claim.

The specification also does not contain any link between the deactivation of the up-converter or down-converter and the deactivation of the super capacitors. For example, the specification states that an up-converter is deactivated when external power has fallen below a specific level. *Id.* at 5:19-22; 6:22-25; Figs. 3 and 5. Because the up-converter is simply a component that “up converts” external power to the voltage necessary to charge the super capacitors, however, it cannot itself deactivate the super capacitors and deactivation of up-converter will not cause deactivation of the super capacitors.⁴ *Id.* at 3:39-41. The only other mention of “deactivation” in the specification is a single sentence indicating that a “down-converter” can be deactivated once “the external power system is risen back to normal levels.” *Id.* at 5:52-57. However, this passage says nothing about deactivating the super capacitors themselves, much less doing so once the super capacitors have discharged to a predetermined level. To the contrary, this passage indicates that the down-converter is deactivated once the external power has turned back on—not when “the plurality of super capacitors discharge[] to a predetermined level” as required by the claim. Similarly, while other passages mention that the down-converter may shut down when it has insufficient voltage differential, these passages are utterly silent regarding “deactivation” of the super capacitors. *See id.* at 5:39-43; 6:40-44. There is no

⁴ Notably, Figure 3 shows that the super capacitors only supply power **after the up-converter is deactivated**, establishing that deactivation of the up-converter does not—and cannot—deactivate the super-supercapacitors. *See* '939 patent, Fig. 3 (demonstrating that up-converter is deactivated in step 100, after which current is discharged from the super capacitors in steps 104-110).

indication at all that the down-converter’s shutdown renders the super capacitors inactive. The claim recites something very specific—deactivating the super capacitors—and the patent does not describe whether or how super-capacitor deactivation occurs. At most, the patent states that the down converter shuts down. These passages fail to “clearly link” any structure to the recited function of “deactivating the plurality of super capacitors” as required for a means plus function element. *See Dig. Retail*, 2020 WL 376664, at *3.

Accordingly, because the specification of the ’939 patent “does not contain an adequate disclosure of the structure that corresponds to the claimed function,” the patent “fail[s] to particularly point out and distinctly claim the invention as required by the second paragraph of section 112, which renders the claim invalid for indefiniteness.” *Id.*, at *7 (citing *In re Donaldson Co.*, 16 F. 3d 1189, 1195 (Fed. Cir. 1994)); *Ergo Licensing*, 673 F.3d at 1363 (“If an applicant does not disclose structure for a means-plus-function term, the claim is indefinite.”).

While BiTMICRO acknowledges that this is a means plus function term, BiTMICRO fails to identify a specific structure for performing the claimed function. Rather, BiTMICRO cites a laundry list of figures and paragraphs from the specification and contends that the allegedly corresponding structure is present ***somewhere*** in that collection. Such an “absurdly overinclusive designation fails to identify a sufficiently definite structure that corresponds to the claimed function.” *Bell N. Research, LLC v. Coolpad Techs., Inc.*, No. 18-CV-1783-CAB-BLM, 2019 WL 3766688, at *6 (S.D. Cal. Aug. 9, 2019). BiTMICRO would instead have Defendants and the Court speculate as to the allegedly corresponding structure by “scour[ing] the Plaintiff’s submission to locate, or otherwise ascertain from the blanket proffer made by Plaintiff what structure . . . is disclosed to provide the function.” *Id.* This is improper.

BiTMICRO’s inability to identify a specific structure simply affirms that no such structure is disclosed. *See id.* (finding means plus function claim indefinite based, in part, on plaintiff’s failure to identify a specific structure). This is further confirmed by the portions of the specification cited by BiTMICRO, the majority of which make no mention of “deactivating” anything, much less a super capacitor. ’939 patent at 3:3-12, 3:28-4:8, 4:20-35, 4:46-52, 4:64-5:5, 5:35-42, 6:37-44. The remainder of BiTMICRO’s citations discuss deactivation of ***different*** components—such as an up-converter and down-converter—not a super capacitor, ***as required by the claims***. *See, e.g., id.* at Figs. 3 and 5. As such, BiTMICRO’s inadequate and imprecise proposal underscores the indefiniteness of this claim element.

D. “reversing the flow of current between the computing engine and the plurality of super capacitors”

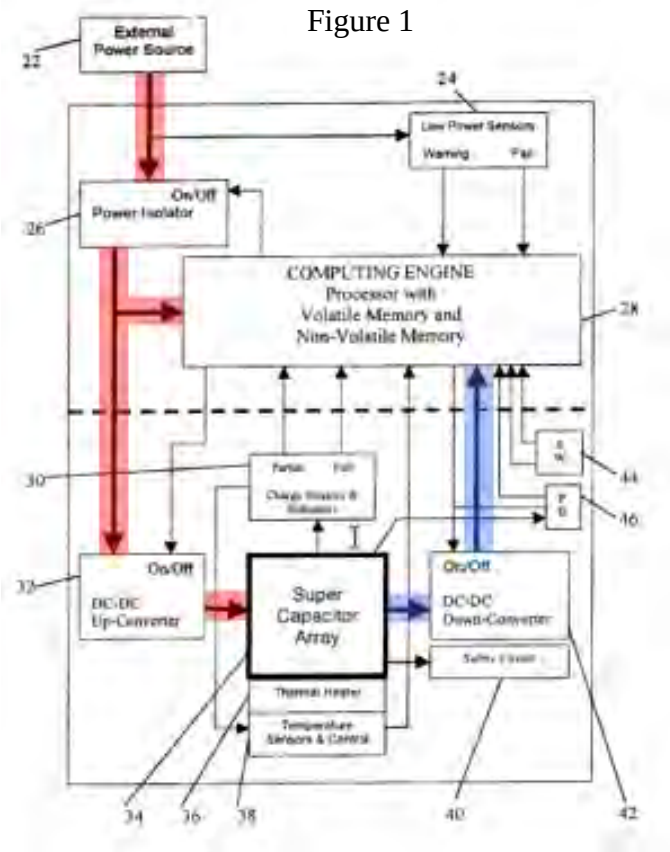
Patent	Plaintiff’s Proposal	Defendants’ Proposal
’939 patent: Claim 6	Plain and ordinary meaning; not indefinite.	Indefinite.

A claim element that is inconsistent with the description of the alleged invention in the specification is indefinite under 35 U.S.C. § 112(2). *See Allen Eng’g Corp. v. Bartell Indus., Inc.*, 299 F.3d 1336, 1349 (Fed. Cir. 2002) (finding that a claim requiring a plane to be “perpendicular” to another plane was indefinite where the recited orientation was inconsistent with the orientation disclosed in the specification); *see also In re Cohn*, 438 F.2d 989, 993 (C.C.P.A. 1971) (finding claims indefinite where “the description, definitions and examples appearing in appellant’s specification” were “inherently inconsistent” with the claims); *Ex Parte Gittleman*, 2008 WL 696137, at *4 (B.P.A.I. 2008) (“A claim also runs afoul of § 112, ¶ 2 when it is inherently inconsistent with what is described in the specification.”). In this case, the plain language of the claim requires a two-way flow of current between the computing engine (processor and memories)

and super capacitors such that current can be “reversed.” But the specification of the ’939 patent establishes that current only flows in one direction between the external power source, the super capacitors, and the computing engine. As a result, the requirement that current flow be “reversed” between the computing engine and the super capacitors upon activation of the super capacitors is both non-sensical and inconsistent with the specification, rendering this term indefinite.

As the specification explains, power is supplied to the computing engine from an external power source (“EPS”), in only one direction, during normal operation. ’939 patent at 3:32-35. The EPS also supplies current to charge the super capacitors in the same, singular direction. *Id.* at

3:39-41. Once external power has been lost, the super capacitors are activated to supply current to the computing engine. In that event, however, the flow of current between the super capacitors and the computing engine is still maintained in the same, singular direction. This is demonstrated in annotated Figure 1 at right, which illustrates (1) the unidirectional flow of current from the external power source to both the computing engine and super capacitors (highlighted in **red**), and (2) the unidirectional flow of current from the super capacitors to the



computing engine upon activation (highlighted in **blue**). The specification’s references to reversing current flow do not teach otherwise. In one passage, current flow reverses “between the

computer system and the super-capacitors.” *Id.* at 5:24-26 (emphasis added). Computer system 20 (represented by the whole of Figure 1) differs from and is much broader than the computing engine (processor and memories) recited in the claim. A second passage describes current flow that reverses “between the volatile memory and the super-capacitors.” *Id.* at 6:27-28. This statement is clearly an error: the patent nowhere suggests, let alone discloses, a current flow from the volatile memory within the computing engine to the super capacitors.

As annotated Figure 1 and its corresponding disclosure further confirm, current does not flow in two directions between the computing engine and the super capacitors. It only flows in one direction—it is not “reversed” upon activation of the super capacitors. In order for current reversal to occur, current from the external power source would need to flow ***from the computing engine to the super capacitors*** during normal operation, which is the exact opposite of the current flow that is illustrated and described in the specification. *See, e.g., id.*, Fig. 1 (demonstrating that current flows directly from the external power source to the super capacitors); 3:32-41 (same). Accordingly, because the claim’s requirement that current be reversed is inconsistent with the description in the specification, this term is indefinite. *Allen Eng’g*, 299 F.3d at 1349; *In re Cohn*, 438 F.2d at 993; *Ex Parte Gittleman*, 2008 WL 696137 at *4.

E. “means for reversing the flow of current between the computing engine and the plurality of super-capacitors”

Patent	Plaintiff’s Proposal	Defendants’ Proposal
’939 patent: Claim 15	Function: reversing the flow of current between the computing engine and the plurality of super capacitors. Structure: Col. 3:3-12, 3:28-4:8, 4:20-35, 4:64-5:5, 5:18-26, 6:22-28, Figs. 1, 3, 5, of the ’939 Patent, and equivalents thereof.	Indefinite under 35 U.S.C. § 112(6): Lacks corresponding structure.

As explained above, the specification does not teach “reversing” the flow of current

between the computing engine and super capacitors upon activation of the super capacitors. On that basis alone, this claim term is indefinite for failing to disclose the recited function. However, the specification also does not disclose a structure for performing the recited function. As Figure 1 makes clear, there is no structure that “reverses” current flow between the super capacitors and computing engine such that current flows in the opposite direction when the super capacitors are activated. *See* Section II.D, *supra*. As such, this claim element is indefinite for failure to disclose sufficient corresponding structure. *Dig. Retail*, 2020 WL 376664, at *7 (citing *In re Donaldson*, 16 F.3d at 1195); *Ergo Licensing*, 673 F.3d at 1363.

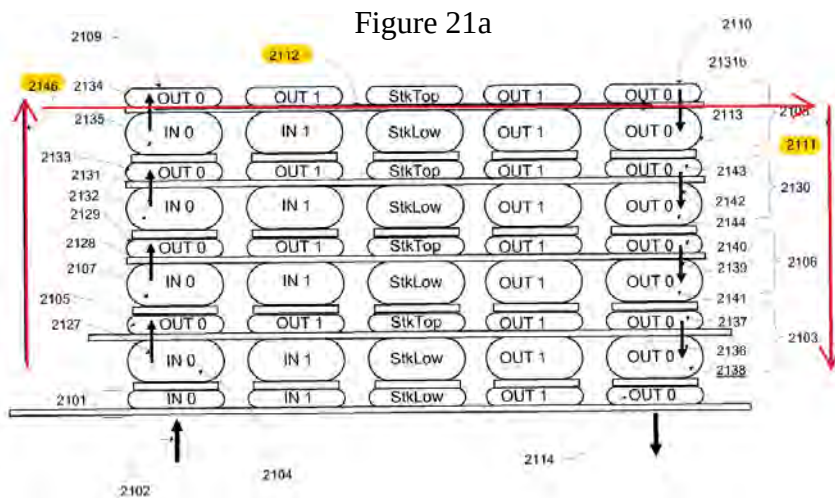
As with the other means plus function terms, BiTMICRO fails to identify a specific structure that allegedly corresponds to the recited “reversing” function, instead citing the same laundry list of figures and paragraphs from the specification that it cites for the other means plus function elements. This is both improper and confirms the lack of corresponding structure as discussed above. *See* Section II.C, *supra*; *Bell N.*, 2019 WL 3766688, at *6.

III. '243 PATENT

A. Technical Background

The '243 patent purports to “support miniaturization and memory scalability” by stacking and interconnecting multiple memory modules to form packages. *Id.* at 1:7-12, 32-42. Through this stacking technique, the patent states, “large capacity storage devices are implemented in a small package device, and larger form factors achieve bigger memory capacities.” *Id.* at 1:54-57.

The '243 patent describes several memory module stacking embodiments. Relevant here is the embodiment of Figures 21a-b, the sole embodiment that includes the “serial chain” and “control circuit” components recited by asserted claims 1, 2, 11, and 12. Figure 21a shows a first serial chain route 2146 that is connected to a second serial chain route 2111 via routing path 2112. *Id.* at 9:67 (as modified by Certificate of Correction), Fig. 21a (annotated at right). A control circuit in each memory



module determines how signals are routed between modules. *Id.* at 10:8-35, Fig. 21b. The patent states that “[s]ince the modules are identical,” a signal can be routed through “all modules in the stack” to the output of the main board for external access. *Id.* at 9:61-64, 10:5-8.

B. “passive port”

Patent	Plaintiff’s Proposal	Intel’s Proposal
'243 patent: Claims 1, 2	“connection between a passive ball and a passive pad”	“connection between a passive ball on one surface of a SDRAM module and a passive pad on another surface of the same SDRAM module.”

During prosecution, and in response to a rejection under 35 U.S.C. § 112, the applicants for the '243 patent amended their specification to add an express definition of “passive port”: “The connection between a passive ball on one surface of a SDRAM module and a passive pad on another surface of the same SDRAM module . . . is named ‘passive port.’” '243 patent at 9:8 (as modified by Certificate of Correction); Ta Decl., Ex. F at 3, 22-23. This is an unambiguous definition of “passive port.” See *BookIT Oy v. Bank of Am. Corp.*, 817 F. App'x 990, 994 (Fed. Cir. 2020) (holding, where specification stated that “the mediator is a network based service,” that

“[t]here can be no clearer definition[]” of the disputed claim term) (emphasis added); *Sinorgchem Co., Shandong v. Int’l Trade Comm’n*, 511 F.3d 1132, 1136-37 (Fed. Cir. 2007) (observing that “the word ‘is,’ . . . used here in the specification, may signify that a patentee is serving as its own lexicographer”) (quotation omitted). This statement also operates as a clear and unambiguous disavowal of claim scope, given that the applicants chose to add it to the specification to overcome a rejection. *E.g., Tech. Properties Ltd. LLC v. Huawei Techs. Co.*, 849 F.3d 1349, 1358 (Fed. Cir. 2017) (holding that “statements made to overcome [the rejection]” during prosecution operated to disclaim claim scope, where they “were clear and unmistakable”).

To the extent there was ambiguity about the definitional nature of applicants’ statement, they ***told the Patent Office expressly*** that they were acting as “their own lexicographers”:

Applicants may be their own lexicographers. MPEP 2173.01. The Specification at p. 21, line 7 through p. 22, line 4, as amended in the “Amendments to the Specification” section above, recites in part:

. . . A ***connection*** between a passive ball on one surface of a SDRAM module and a passive pad on another surface of the same SDRAM module, such as the X1 passive ball and X1 passive pad connection in Figure 19, is named “passive port”, such as passive ports 1909 illustrated in Figure 19. (Emphasis added).

. . . ***Thus, “a connection between a passive ball on one surface of a SDRAM module and a passive pad on another surface of the same SDRAM module” is a “passive port”.***

Ta Decl., Ex. F at 23 (emphases added). Applicants also acknowledged to the Patent Office that this definition was consistent with the specification, which “shows [in Fig. 19] that a passive port 1909 ***connects*** ‘a passive ball’, such as passive ball X2, ‘on one surface of a SDRAM module’, . . . and a passive pad[], such as passive pad X3, ‘on another surface of the same SDRAM module.’” *Id.*

There is no more straightforward example of a “clearly express[ed] . . . intent to define the term”: not only did applicants use the word “lexicographers,” they pointed specifically to the

MPEP section discussing lexicography. *See GE Lighting Sols., LLC v. AgiLight, Inc.*, 750 F.3d 1304, 1309 (Fed. Cir. 2014). The Federal Circuit has recognized that “[w]hen the patentee acts as its own lexicographer, that definition governs.” *Cont’l Circuits LLC v. Intel Corp.*, 915 F.3d 788, 796 (Fed. Cir. 2019); *see also Kyocera*, 22 F.4th at 1378 (“Because the patentee clearly defined ‘driven position’ within the written description, that definition controls.”); *Biogen MA Inc. v. EMD Serono, Inc.*, 976 F.3d 1326, 1335-36 (Fed. Cir. 2020) (noting that patentee’s “argument fails to give effect to [the] explicit definition of ‘polypeptide’ in the specification. We must respect this lexicographic choice.”). This Court has also recognized that a patentee’s definition controls. *See Portus Singapore Pte. Ltd. v. Ecobee, Inc.*, No. 6:20-cv-00694-ADA, ECF No. 36, slip op. at 8-10 (W.D. Tex. Jul. 23, 2021) (holding that “the intrinsic record demonstrates the Patents provide a specific definition of the term [extranet],” where the specification stated that “[a]n extranet *is* a private network” with specific characteristics). And, by applicants’ own admission, their definition here is consistent with the specification. That definition should control.

C. “a plurality of modules each comprising”

Patent	Plaintiff’s Proposal	Intel’s Proposal
’243 patent: claim 1	Part of the preamble and not a limitation of the claim.	“each module in the stacked module comprises”

The disputed term “a plurality of modules each comprising” is part of the claim body, not the preamble. The preamble of claim 1 recites “[a] stacked module comprising.” The “plurality of modules each comprising” follows this preamble language and is thus a limitation. *See Nichia Corp. v. Document Sec. Sys., Inc.*, 2022 WL 1218036, *4 (Fed. Cir. Apr. 26, 2022) (where claim recited “[a] display comprising a plurality of plastic leaded chip carrier LEDs, the plastic leaded chip carrier LEDs each comprising:”, the preamble included only “[a] display comprising”).

As this term is part of the claim body, not the preamble, Intel’s construction is consistent with the plain language—“each” means “each”—and the specification, which repeatedly describes a stacked module that includes multiple **identical** modules. For example, each of Figures 19 (shown at right), 20b, 21a, and 21b depicts a stacked module made up of identical modules with the same configuration. ’243 patent at 9:6-

8 (“**All SDRAM modules are identical**, and in the example shown in FIG. 19 include the connections as follows: X1 ball is connected to X1 pad, X2 ball to X2 pad, and so on.”) (as modified by Certificate of Correction); *id.* at 3:29-33 (“FIG. 20a shows . . . a rotational stacking orientation

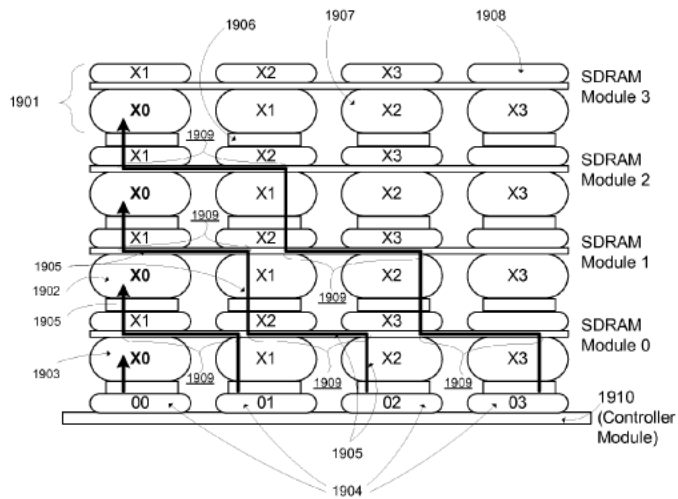


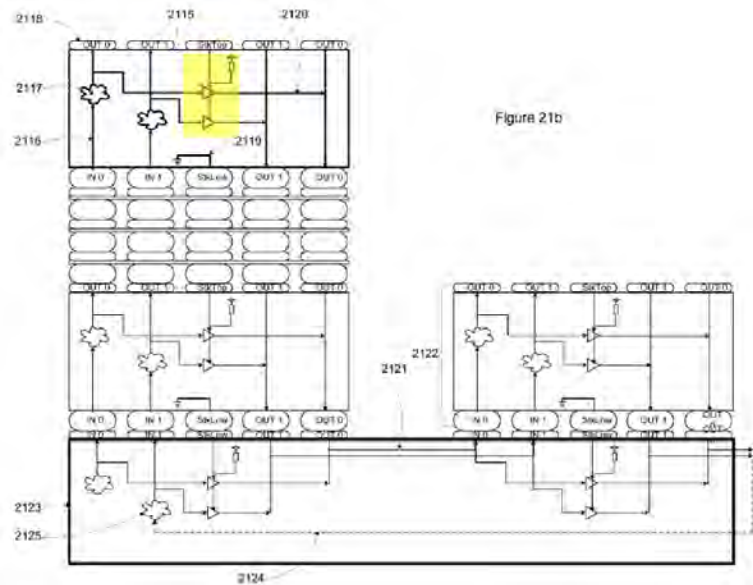
Figure 19

to allow stacking of **four identical modules**”). The specification states that the rotational stacking technique of certain embodiments is feasible only **because** the modules are identical. *Id.* at 9:32-38 (“**Since the stacked modules are identical**, we can use rotational stacking. . . .”). Dependent claims also contemplate identical modules. *Id.* at cls. 3, 10 (“same location on one or more modules”), cl. 13 (“same location on the plurality of modules”).

Further, the sole claimed embodiment **requires** identical stacked modules. The ’243 patent describes the claimed “control circuit” and “control input signal” in connection with the embodiment of Figures 21a-b. See *id.* at 10:10-11 (as modified by Certificate of Correction). These illustrate a “connection technique . . . used for serial routing of all modules in the stack” that “allow[s] them to be externally accessible for horizontal expansion on a PCB.” *Id.* at 9:61-64. The patent states that “**Since the modules are identical**,” a routed signal can be passed through the

pads and balls of the stacked modules until it reaches an externally accessible “OUT pad.” *Id.* at 10:5-8. More specifically—and as shown in Figure 21b—“**All modules** have a control circuit that includes a selectable buffer 2115 that tri-states the input when pulled low.” *Id.* at 10:10-11 (as modified by Certificate of Correction), Fig. 21b (at right, with control circuit of top-left module highlighted). When a module is on top of the stack, the buffer 2115 control line is pulled high and

“let[s] the input signal branch out 2120 to the other pads, thus creating a connection route or trace that loops back to the base module solder balls.” *Id.* at 10:13-20; *supra* § III.A (annotated Fig. 21a showing serial chain routes connected via a routing path). Conversely, when a module is stacked below another



module, the buffer 2115 control line is pulled low and “disallow[s] the branching effect.” ’243 patent at 10:14-17, 20-24. This achieves “horizontal expansion” by routing the output signal to the OUT pads of the base module on the bottom of the stack. *Id.* at 10:24-26. But it is only **because** each module operates identically—with the same control circuit and the same tri-state buffer—that this arrangement works. *Id.* at 10:31-32 (“The tri-stated buffer technique is redundant to all locations.”).

Even if this Court determines that the disputed term is part of the preamble of claim 1, it is still limiting because the body of the claim depends on it for antecedent basis. *See In re Fought*, 941 F.3d 1175, 1178 (Fed. Cir. 2019) (“We have repeatedly held a preamble limiting when it

serves as antecedent basis for a term appearing in the body of a claim.”). The last limitation of claim 1 recites “the plurality of modules.” ’243 patent at 10:59. This depends upon “a plurality of modules” for antecedent basis; there is no other mention of a “plurality of modules” in the claim.⁵ Additionally, because “a plurality of modules” is intertwined with “each comprising,” the entire phrase “a plurality of modules each comprising” is limiting. *See Bio-Rad Labs., Inc. v. 10X Genomics Inc.*, 967 F.3d 1353, 1370-72 (Fed. Cir. 2020) (language intertwined with terms in preamble providing antecedent basis was also limiting). Further, as the discussion above shows, the “each comprising” aspect is “necessary to give life, meaning, and vitality” to the claimed module. *See Pitney Bowes, Inc. v. Hewlett-Packard Co.*, 182 F.3d 1298, 1305-06 (Fed. Cir. 1999).

IV. ’740 PATENT

A. Technical Background

The ’740 patent is directed to purported solutions for optimizing memory operations by “optimally distribut[ing]” operational load across resources of the memory system, thereby “increasing operational efficiency, reducing memory operation latency or both.” ’740 patent at 1:16-24; 3:15-21.

The ’740 patent teaches a mapping table 9 (see Fig. 2 below) that maps LBA sets 40, 42, 44, and 45 from a host addressing system to PBAs 46, 48, 50, and 51 that can be used by an electronic storage device to

9

Logical Fields 26	PBA Fields 28			
	Access Parameter Fields 30			Memory Index Fields 32
	Bus Id Fields 34	FDE Id Fields 36	Group Id Fields 38	
40 LBAs 00-07	8-0	22-01	15-0	0x00 46
42 LBAs 08-15	8-0	22-02	15-1	0x08 48
44 LBAs 16-23	8-1	22-11	15-0	0x00 50
45 LBAs 24-31	8-1	22-12	15-1	0x08 51
⋮	⋮	⋮	⋮	⋮
⋮	⋮	⋮	⋮	⋮

FIG. 2

⁵ The other references to “module” in claim 1 are to “[an / the] end module” (’243 patent at 10:55, 10:59-60), and “[another module / the another module] from the plurality of modules” (*id.* at 10:58-60). These refer to specific modules *within* the plurality, not to the plurality as a whole.

access the corresponding physical memory locations. *Id.* at 5:27-30. “LBA” refers to logical block addresses that “represent an address that is part of a logical addressing system . . . used by a host.” *Id.* at 6:37-54. LBA sets 40, 42, 44, and 45 “respectively represent consecutive LBAs used by host 18, such as LBAs 00 through 07, LBAs 08 through 15, LBAs 16 through 23, and LBAs 24-31.” *Id.* at 6:51-54.

“PBA” fields refer to physical block address fields that each represent a unique addressable physical memory location in a memory store, such as PBA fields 46, 48, 50 or 51. *Id.* For each PBA field, the physical memory location is represented by a set of access parameter fields 30 and memory index fields 32. *Id.* Each access parameter field 30 includes a Bus Id field 34, FDE Id field 36, and Group Id field 38, which essentially provide a path for accessing a specific physical memory location.

B. “optimized memory operations”

Patent	Plaintiff’s Proposal	Defendants’ Proposal
’740 patent: Claim 1	Plain and ordinary meaning; not indefinite.	Indefinite.

The phrase “optimized memory operations” in independent claim 1 of the ’740 patent is indefinite. Whether memory operations are within or outside the scope of the claim is impossible to decipher because the claim provides no criteria for assessing whether memory operations have been optimized. *Nautilus*, 572 U.S. at 901 (claim is indefinite when it “fail[s] to inform, with reasonable certainty, those skilled in the art about the scope of the invention.”). Nor does resort to the written description solve the problem. Although the written description includes one approach for optimizing memory transactions, constraining claim 1 to that example would render dependent claim 2 redundant and meaningless. Thus, claim 1 cannot be constrained to the illustration in the written description.

Asserted claim 1 requires a mapping table wherein, “in response to receiving the I/O transaction request, said mapping table causes the electronic storage device to perform **optimized memory operations** on memory locations respectively associated with said first PBA and said second PBA, if the I/O transaction request is associated with said first and second LBA sets.” ’740 patent, cl. 1 (emphasis added). The ’740 patent explains that “optimized memory operations” can be accomplished by “increasing the likelihood that, in response to an I/O transaction initiated by a host, the operational load will be ‘**optimally distributed**’ across different storage device resources, such as by interleaving or parallel memory operations, reducing memory operation latency, increasing operational device efficiency, or both.” See *id.* at 2:14-22 (emphasis added); see also 1:16-24, 3:12-21, 7:9-18, 9:19-25. The ’740 patent teaches only one way to “optimally distribute” operational load and thus optimize memory transactions—i.e., to map LBA sets to PBA fields such that memory transactions across multiple LBA sets can be conducted in parallel.

Figure 2 of the ’740 patent shows a mapping table in which LBA (logical block address) sets are associated with PBA (physical block address) fields consisting of access parameter fields 30 and memory index fields 32. ’740 2:14-22; 7:9-18, 9:19-25. For memory transactions across two or more LBA sets to be conducted in parallel, one or more of the access parameters associated with the LBA sets must be different. *Id.* As highlighted in red, the FDE

Logical Fields 26	PBA Fields 28			
	Access Parameter Fields 30			Memory Index Fields 32
	Bus Id Fields 34	FDE Id Fields 36	Group Id Fields 38	
LBAs 00-07	8-0	22-01	15-0	0x00
LBAs 08-15	8-0	22-02	15-1	0x08
LBAs 16-23	8-1	22-11	15-0	0x00
LBAs 24-31	8-1	22-12	15-1	0x08
⋮	⋮	⋮	⋮	⋮
⋮	⋮	⋮	⋮	⋮

FIG. 2

and Group ID access parameters associated with LBA sets 40 and 42 differ, which permits overlapping memory operations tied to those two LBA sets.

While claim 1 requires an I/O transaction to be associated with more than one LBA set, it does not require that the access parameters associated with said first and second LBA sets must be different as taught in the written description. *See* '740 patent, cl. 1 (“the I/O transaction request is associated with said first and second LBA sets”). Claim 2, which depends from claim 1, does require that difference. *See id.* at cl. 2 (“said first and second access parameters are different.”). Under the doctrine of claim differentiation, independent claim 1 presumptively has a broader scope from dependent claim 2. *See Phillips*, 415 F.3d at 1315. But interpreting “optimized memory operations” to mean the type of optimal distribution taught in the '740 patent would leave claim 1 and dependent claim 2 without any difference of scope, which is plainly not the correct reading of the claims. If on the other hand, claim 1 is given broader scope according to the ordinary meaning of “optimized memory operations,” claim 1 is indefinite because it fails to provide reasonably certain guidance about how to assess what is or is not optimal.

The court in *Chewy Inc. v. IBM Corp.*, 571 F. Supp. 3d 133, 153 (S.D.N.Y. 2021), addressed an analogous situation. There, the patentee argued that the term “on demand by said user” was not indefinite because the specification explained that “[f]or each search result item, a graphical user interface (GUI) selection is presented, allowing the user to select the GUI, on demand if so desired, to investigate related advertisement.” *Id.* Relying on this teaching in the specification, the patentee argued that “on demand by said user” is not indefinite and should be construed to mean “upon the user selecting a graphical user interface to investigate related advertisement.” *Id.* The court held that this interpretation failed as a matter of claim differentiation because another dependent claim recited the step of “displaying a graphical user interface to [the]

user.” *Id.* Because the patentee could not offer a construction that would render the disputed term meaningful without encroaching on the explicit language of another claim, the court found the disputed claim term to be indefinite. *Id.* As in *Chewy*, claim 1 is indefinite because there is no construction of claim 1 that would render the term “optimized memory operations” meaningful without encroaching on the explicit language of claim 2.

C. “optimally distributed”

Patent	Plaintiff’s Proposal	Defendants’ Proposal
’740 patent: Claim 32	Plain and ordinary meaning; not indefinite.	Indefinite.

The term “optimally distributed” as used in claim 32 of the ’740 patent is also indefinite for similar reasons as the term “optimized memory operations.” Claim 32 requires:

An electronic storage device that includes a memory system that uses a memory table for increasing the likelihood that an operational load imposed on the storage apparatus during the processing of an I/O transaction request will be **optimally distributed** across storage device resource ... wherein said mapping table increases the likelihood that that [sic] the operational load imposed on the storage apparatus during the processing of the I/O transaction request will be **optimally distributed** across storage device resources.

’740 patent, cl. 32 (emphasis added).

Claim 32 describes “at least one difference between said first and second PBAs,” which is one of the predicate requirements for optimizing memory transactions (as described above), but it does not recite that the I/O transaction is associated with two LBA sets—the other predicate requirement. The latter requirement is part of claim 34, which depends from claim 32. *See id.* at cl. 34 (“wherein the processing of the I/O transaction request includes a first memory operation performed using said first PBA and a second memory operation performed using said second PBA.”). Although claim 32 relates to two memory operations performed using two PBAs, that is the same as requiring an I/O transaction to be associated with two LBA sets because claim 32

requires that “a mapping table which maps a first LBA set to a first PBA, and a second LBA set to a second PBA.” Again, claim differentiation suggests that claim 32 is broader than claim 34. *See Phillips*, 415 F.3d at 1315. It follows that claim 32 covers optimal distribution in the case where there is only a single set of LBAs. Here too, the patent provides no guidance as to how that is possible, leaving the scope of the claim indefinite.

D. “memory store”

Patent	Plaintiff’s Proposal	Defendants’ Proposal
’740 patent: Claims 1, 15, 32	“non-volatile, solid-state storage, e.g., flash”	Plain and ordinary meaning.

The term “memory store” should be given its plain and ordinary meaning, which is not limited to “non-volatile, solid-state storage, e.g., flash” memory as plaintiff suggests. For example, independent claim 1 recites a “memory store,” while claims 7, 8, and 15—each of which ultimately depends from claim 1—specifically recite that the memory store comprises either “a plurality of flash devices” or a “set of solid state memory devices.” Likewise, claim 17 recites a memory store that includes a first and second set of memory devices, while claim 19—which depends from claim 17—further recites that the first and second memory devices “include non-volatile solid state memory cells.” Under the doctrine of claim differentiation, “limitations stated in dependent claims are not to be read into the independent claim from which they depend.” *Nazomi Commc’ns, Inc. v. Arm Holdings, PLC*, 403 F.3d 1364, 1370 (Fed. Cir. 2005) (citations omitted); *Phillips*, 415 F.3d at 1314-15 (“[T]he presence of a dependent claim that adds a particular limitation gives rise to a presumption that the limitation in question is not present in the independent claim.”). Accordingly, because the dependent claims expressly limit a “memory store” to solid state and flash devices (which reflect the outer boundaries of plaintiff’s proposed constructions), these limitations cannot be read into the broader construction of “memory device.”

This conclusion is further confirmed by asserted claims 15 and 32, which respectively state that the “memory store” “**includes** a set of solid state memory devices” and “**includes** a set of flash devices,” signifying an open-ended use of the term “memory store” that does not foreclose the inclusion of other types of memory devices. ’740 patent, cl. 15, 32 (emphases added); *see also*, *e.g.*, *In re Horvitz*, 168 F.2d 522, 525 (C.C.P.A. 1948) (holding that “comprises” is synonymous with “includes” and does not preclude the presence of other elements as part of the described structure). Indeed, construing a “memory store” to be limited to a solid state memory device or flash device would render the language in these claims redundant, further confirming that Plaintiff’s proposed construction is incorrect. *See Apple, Inc. v. Ameranth, Inc.*, 842 F.3d 1229, 1237 (Fed. Cir. 2016) (affirming that construction of a term should not include features “expressly recited in the claims” because “[c]onstruing a claim term to include features of that term already recited in the claims would make those expressly recited features redundant.”)

The specification also refutes Plaintiff’s proposed, narrow construction. As an initial matter, nothing in the specification expressly disclaims the use of devices other than solid-state storage devices. To the contrary, the specification states that “[m]emory store 6 **may be configured to include** a set of solid state memory devices that are coupled to a set of buses and that are controlled by a set of FDEs.” ’740 patent at 4:54-56 (emphasis added). Use of the phrase “may be configured to include” confirms that solid state memory devices are permitted but not required. The specification also states that “[t]he term ‘flash memory device,’” which constitutes one example of a memory store, “is intended to include **any form** of non-volatile memory, including those that use blocks of non-volatile memory cells, named flash blocks.” *Id.* at 4:59-62 (emphasis added). As a result, the specification establishes that the claims are not limited to solid-state storage devices.

BiTMICRO’s own Complaint demonstrates that BiTMICRO understands that the plain and ordinary meaning of the term “memory store” is not limited to non-volatile memory stores. For example, the Complaint, in discussing the ’190 patent, refers to Dynamic Random Access Memory (“DRAM”)—a volatile memory—as a type of memory store: “The Optane Persistent Memory product includes ‘a memory store.’ For example, the Optane Persistent Memory product **includes both DRAM memory** and persistent memory modules, **which separately** or in combination **are a memory store.**” No. 6:22-cv-00335-ADA, ECF No. 1 at ¶ 106 (emphases added). BiTMICRO’s proposed construction, which limits “memory store” to “non-volatile, solid-state storage, e.g., flash,” is inconsistent with even its own understanding of the plain and ordinary meaning of that term.

Since the intrinsic evidence alone is sufficient to rebut Plaintiff’s proposal, the Court need not consult extrinsic evidence. Nonetheless, Defendants’ extrinsic evidence confirms that the plain and ordinary meaning of “memory store” is not limited to “non-volatile, solid-state storage, e.g., flash” as BiTMICRO proposes. For example, the extrinsic illustrates the following:

- Definition of “memory”: “All of the addressable storage in a processing unit and other internal storage that is used to execute instructions.” Ta Decl., Ex. G, *The Authoritative Dictionary of IEEE Standards Terms* (7th ed.) at 684 (Inst. of Elec. & Electronics Engineers, Inc. 2000).
- Definition of “storage medium”: “Any device or recording medium into which data can be stored and held until some later time, and from which the entire original data can be obtained.” *Id.* at 1113.
- Definition of “memory”: “A device where information can be stored and retrieved.” Ta Decl., Ex. H, *Microsoft Computer Dictionary* (5th ed.) at 333 (Microsoft Corp.

2002).

- Definition of “storage”: “In computing, any device in or on which information can be kept. Microcomputers have two main types of storage: random access memory (RAM) and disk drives and other external storage media. Other types of storage include read-only memory (ROM) and buffers.” *Id.* at 498.

As the plain and ordinary meaning of “memory store” is well supported by the intrinsic and extrinsic record, the Court should reject BiTMICRO’s attempt to limit that term to “non-volatile, solid-state storage, e.g., flash” memory.

V. ’190 PATENT

A. Technical Background

The ’190 patent pertains to a “multi-profile memory controller” where memory locations are associated with different memory devices having different memory attributes. *See* ’190 patent at Abstract. The patent alleges that prior art memory controllers—components that control the flow of information going to and from a computer’s main memory—often managed memory devices that have the same memory device characteristics, e.g., flash memory devices that have the same large block size. *Id.* at 1:20-37. But in some instances, that shared characteristic might be undesirable: large block sizes may spawn latency and other issues. *Id.* at 1:38-43.

The ’190 patent is directed to memory controllers that “can perform memory transactions using different device profiles.” *Id.* at 3:61-63. The patent discloses that a memory store can include addressable memory locations associated with various memory attributes, such as the block size, the page size, the memory protocol, the memory device type, and the partial page size. *Id.* at 2:63-3:13. The memory attributes are organized into a “device profile.” *Id.* at 3:14-15. For example, in Figures 1 and 2B below, memory location 9 in memory store 4 may have a device profile 26 that includes attributes like “page size 28 associated with an attribute value of 1024

bytes, block size 30 associated with an attribute value of 2048 pages, a memory protocol 32 associated with an attribute type that represents the ONFi specification, a memory device type 34 associated with an attribute type that represents a NAND flash memory device, and a partial page size 36 associated with an attribute value of 512 bytes.” *Id.* at Fig. 1, 3:40-49. The patent further teaches that the alleged invention is not limited by the type of memory device or any number of innumerable attributes or device profiles. *Id.* at 3:61-64, 4:26-28. 5:4-6.

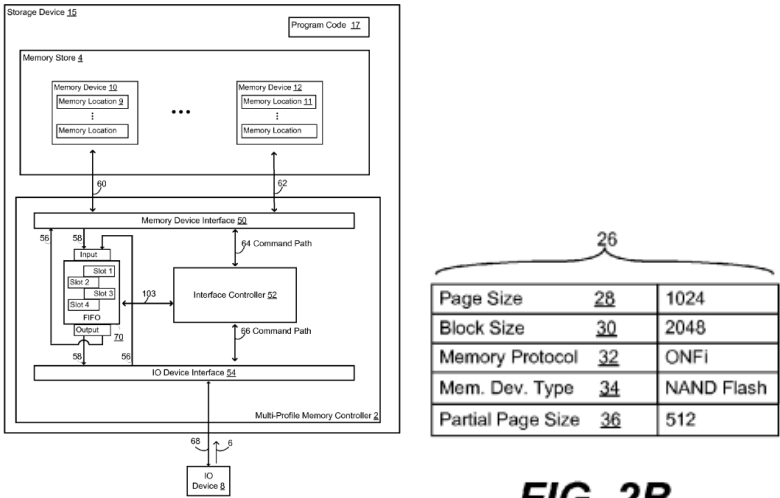


FIG. 1

FIG. 2B

B. “optimal”

Patent	Plaintiff’s Proposal	Defendants’ Proposal
’190 patent: Claims 1, 29, 45, 59, 80, 98	Plain and ordinary meaning; not indefinite.	Indefinite.

The term “optimal” as used in the ’190 patent to describe a “first device profile” is indefinite for failing to satisfy 35 U.S.C. § 112, ¶ 2. “[A] term of degree fails to provide sufficient notice of its scope if it depends on the unpredictable vagaries of any one person’s opinion.” *Interval Licensing LLC v. AOL, Inc.*, 766 F.3d 1364, 1371 (Fed. Cir. 2014) (citation and internal quotation marks omitted). Here, the term “optimal” is a subjective term of degree, dependent on the characteristic sought to be optimized for any given device profile. *See, e.g., Intellectual*

Ventures I LLC v. T-Mobile USA, Inc., 902 F.3d 1372, 1381 (Fed. Cir. 2018) (“‘[O]ptimiz[ing] . . . QoS’ is a ‘term of degree’ that . . . is ‘purely subjective’ . . .”).

Asserted claims 1, 29, 45, 59, 80, and 98 each claim a “first device profile” that is “optimal for a data type subject to the memory transaction, wherein said data type comprises one of a random data type or a sequential data type.” ’190 patent at cl. 1, 29, 45, 59, 80, 98. Here, the specification notes that a “multi-profile” memory controller can “support[] memory locations that have different device profiles” that are optimal for different data types. *Id.* at 9:7-16. “One target memory location” “is optimal for the type of data subject to a memory transaction request,” while “another memory location” “is associated with another device profile that is optimal for the type of data subject to another memory transaction request.” *Id.* at 9:7-16. But it is unclear from the specification what criteria would make a device profile optimal for a particular data type.

At most, the specification provides three examples of device profiles that “may be desired” when a memory transaction request relates to a particular data type:

- (1) a device profile “that includes a block size attribute value that is the largest available from the other device profiles” when the data type is sequential (*id.* at 9:16-26);
- (2) a device profile “that includes a block size attribute value that is the smallest available from the other device profiles” when the data type is random (*id.* at 9:27-35); and
- (3) a device profile “that includes a partial page size attribute value” when the data type is random and “the data size of the data subject to the write transaction is at most equal to the partial page size attribute value” (*id.* at 9:35-43).

Despite these suggestions, the patent does not explain how to determine when one proposed approach would actually be desired and therefore “optimal.” Furthermore, the specification states that these three examples are not limiting (*see id.* at 9:43-44), and that the memory controller could

include code or use a map that “determines or causes” which memory locations are used for particular data types (*id.* at 9:44-48). The patent even states that “numerous implementation-specific decisions must be made to achieve [a] developer’s specific goals” and such “specific goals will vary from one implementation to another and from one developer to another.” *Id.* at 2:32-36. The patent fails to provide any explanation for how the memory device would select an optimal device profile based on the data type in any of those innumerable situations. The specification’s conclusory statements and three incomplete examples thus fail to provide the metes and bounds for when a particular device profile would be optimal for a particular data type, as required by the claim language.

A similar situation was present in *Intellectual Ventures I LLC v. T Mobile USA, Inc.*, No. 2:17-CV-00577-JRG, 2018 WL 5809267, at *4-5 (E.D. Tex. Nov. 6, 2018). There, the court found indefiniteness where the specification of the patent-at-issue provided two potential definitions for what would constitute “‘optimal’ performance in terms of handling communications in ways that depend on the types of data being communicated,” but failed to provide the metes and bounds of when performance would actually be considered optimal. In one example in *T Mobile*, optimal performance depended on the type of data being communicated—similar to how an optimal device profile for a given data type here may depend on block size, page size, or some other memory device attributes. *Id.* at 4. In another example recited in *T Mobile*, optimal performance “depend[ed] on how ‘the user defines it’”—a definition that is just as open-ended as the code or map implementation disclosed in the specification here. *Id.* at *5. As a result, the Court found that the specification failed to provide “sufficient ‘objective boundaries’ for those of skill in the art to understand the scope of the claims.” *Id.* at *5 (citation omitted).

Because the intrinsic record here fails to inform one of ordinary skill in the art with

reasonable certainty what objective boundaries and criteria are relevant to assessing an “optimal” device profile, the asserted claims should be found to be indefinite.

VI. '084 AND '694 PATENTS

A. Technical Background

The '084 and '694 patents describe techniques that use “a minimum amount of nonvolatile memory” to boot an embedded system through the use of a specialized data structure called a Power-On Reset (“POR”) Sequencer Descriptor. '084 patent at 2:23-25, 3:47-51.⁶ The embedded system of “[t]he present invention comprises . . . a Power-On Reset Sequencer which controls the overall boot process of the embedded system; nonvolatile memory and controller used to store data used by the Power-On Reset Sequencer in the boot process; [and] at least one direct memory access (DMA) controller which initiates and tracks data transfers.” *Id.* at 2:31-42. When the system powers up, the POR Sequencer and DMA controller are released from reset, and the POR Sequencer detects the POR Sequencer Descriptor in nonvolatile memory. *Id.* at 3:16-21. The POR Sequencer Descriptor

includes information for configuring system registers and DMA descriptor information. *Id.* at 6:19-28, Fig. 4 (annotated at right, with

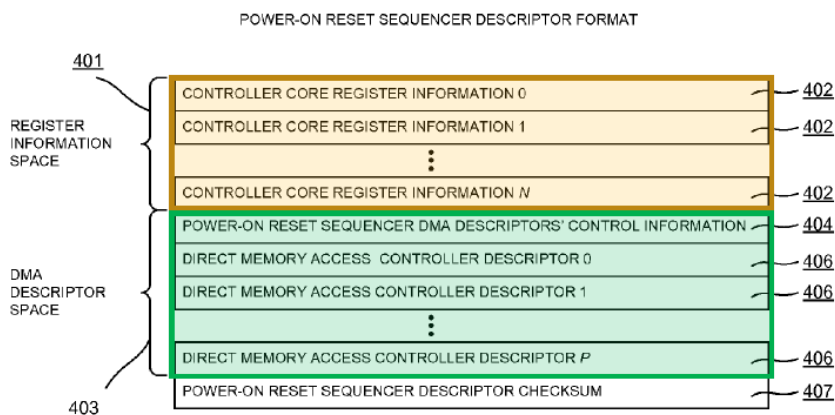


Figure 4

The POR Sequencer copies the POR Sequencer Descriptor to RAM and uses the register information to update the system’s configuration registers. *Id.* at 2:54-56, 3:28-34. Then it loads

⁶ The '084 and '694 patent specifications are nearly identical. Intel cites to the '084 patent specification for simplicity; the '694 patent contains parallel disclosure unless otherwise noted.

the DMA descriptors to the DMA controller. *Id.* at 2:56-58. If the DMA descriptor points to more register information, the DMA controller fetches that information and updates the POR Sequencer Descriptor; if the DMA descriptor points to firmware, the DMA controller copies that firmware to RAM. *Id.* at 3:35-45. When the DMA controller has finished copying all firmware, system CPU(s) are released from reset and execute the firmware to complete the boot process. *Id.* at 3:62-67.

B. “Direct Memory Access (DMA) descriptors” / “DMA controller descriptor”

Patent	Plaintiff’s Proposal	Intel’s Proposal
’084 Patent: Claims 1, 19 ’694 Patent: Claims 1, 6, 12	Plain and ordinary meaning; not indefinite.	“data structure including at least a source and destination address and byte count”

The ’084 patent independent claims recite a “POR sequencer descriptor” that includes “Direct Memory Access (DMA) descriptors used to fetch other POR sequencer descriptor fragments.” ’084 patent at 9:18-22, 10:56-60; *accord* ’694 patent at 9:31-32, 10:8-9, 10:60-61 (“POR sequencer descriptor includes . . . DMA controller descriptors”). Thus, while the claims plainly require that the POR sequencer descriptor includes DMA descriptors, it is not clear what type of information would qualify as a “DMA descriptor.”

Direct Memory Access, or DMA, is a technique for transferring data or code in a system without using the CPU. Alpert Decl. ¶ 15. DMA transfers are typically handled by a specialized device called a DMA controller. *Id.*; *see* ’084 patent at 2:31-42 (“The present invention comprises . . . at least one direct memory access (DMA) controller which initiates and tracks data transfers”). To perform a DMA transfer, a DMA controller requires, at a minimum, three pieces of information: the source location of the data/code to be transferred; the amount (*i.e.*, the length) of whatever is to be transferred; and the destination where the data/code is to be sent. Alpert Decl. ¶ 14. A DMA “descriptor” is a data structure that includes this basic information. *Id.* ¶¶ 15-16. DMA descriptors may have additional information as well, such as a pointer linking to another

DMA descriptor that describes a subsequent transfer operation. *Id.* ¶ 16; *see* '084 patent at Fig. 5, 6:44-46 (“The last DMA controller descriptor 506 . . . points to the next POR sequencer descriptor fragment 502.”).

The '084/'694 patents describe using DMA descriptors in precisely this way. Alpert Decl. ¶ 18. The POR Sequencer loads “DMA descriptors . . . to the DMA controllers of the system.” '084 patent at 2:56-58; *id.* at 7:29-32. The DMA controllers use the DMA descriptors to “initiate[] and track[] data transfers.” *Id.* at 5:3-4. A DMA controller processes each DMA descriptor depending on the information the descriptor “points to,” *i.e.*, what is stored at a particular source address. Specifically, “[i]f the DMA descriptor points to another set of register information,” the DMA controller transfers that information to RAM and updates the POR Sequencer Descriptor. *Id.* at 3:36-40; *id.* at 7:33-37, 7:41-45. “If the DMA descriptor points to a system firmware fragment,” the DMA controller “cop[ies] the system firmware fragments to the RAM.” *Id.* at 3:40-51.

The '084/'694 patents thus describe DMA descriptors as used to *fetch and transfer* data or firmware. *E.g.*, '084 at 7:47-50 (DMA descriptors are used “to fetch POR sequencer descriptor fragments . . . [and] also used to transfer system firmware . . . to the RAM”). The patents also repeatedly describe DMA descriptors as “pointing to” content to be fetched and transferred. *E.g.*, '084 patent at 3:36-42, 3:47-49, 3:54-60, 6:44-46, 6:52-55, 7:41-44, 7:63-66, 8:3-9. And to perform transfers, the DMA controller must know, at a minimum, where to find the content to be transferred (*i.e.*, the source location “pointed to” by the descriptor); the size of that content; and the destination where the content should be transferred. Alpert Decl. ¶¶ 15-18. Thus, a “DMA descriptor” or “DMA controller descriptor,” as used in the claims of the '084 and '694 patents, is a data structure that includes a source address, a byte count, and a destination address.

VII. CONCLUSION

Defendants respectfully request that the Court adopt their proposed constructions for the claim terms at issue.

Dated: November 9, 2022

Respectfully submitted,

/s/ Chad S. Campbell

Chad S. Campbell
CSCampbell@perkinscoie.com
Tyler Bowen (*admitted pro hac vice*)
tbowen@perkinscoie.com
PERKINS COIE LLP
2901 North Central Avenue, Suite 2000
Phoenix, AZ 85012
Tel: (602) 351-8000
Fax: (602) 648-7000

Christina J. McCullough, WA 47147
cmccullough@perkinscoie.com
PERKINS COIE LLP
1201 Third Avenue, Suite 4900
Seattle, WA 98101-3099
Tel: (206) 359-6897
Fax: (206) 359-7897

Nancy Cheng (*admitted pro hac vice*)
ncheng@perkinscoie.com
PERKINS COIE LLP
3150 Porter Drive
Palo Alto, CA 94304
Tel: (650) 838-4325
Fax: (650) 838-4525

Janice L. Ta, Texas 24075138
JTa@perkinscoie.com
PERKINS COIE LLP
405 Colorado St., Suite 1700
Austin, TX 78701
Tel: (737) 256-6100
Fax: (737) 256-6300

Counsel for Defendant Intel Corporation

/s/ Douglas F. Stewart

Douglas F. Stewart
WA State Bar # 34068
Jared D. Schuettenhelm
WA State Bar # 46181
Patrick J. Connolly
WA State Bar # 46767
Bracewell LLP
701 5th Avenue, Suite 6200
Seattle, WA 98104
Telephone: (206) 204-6200
Facsimile: (800) 404-3970
doug.stewart@bracewell.com
jared.schuettenhelm@bracewell.com
patrick.connolly@bracewell.com

David J. Ball
NY State Bar # 4315099
Bracewell LLP
31 W 52nd Street, Suite 1900
New York, NY 10019-0019
Telephone: (212) 508-6100
Facsimile: (800) 404-3970
david.ball@bracewell.com

Timothy R. Geiger
Texas State Bar No. 24078552
Bracewell LLP
711 Louisiana Street, Suite 2300
Houston, Texas 77002-2781
Telephone: (713) 223-2300
Facsimile: (800) 404-3970
tim.geiger@bracewell.com

*Counsel for KIOXIA America, Inc. and
KIOXIA Corporation*

CERTIFICATE OF SERVICE

I hereby certify that on November 9, 2022, a copy of the foregoing DEFENDANTS' OPENING CLAIM CONSTRUCTION BRIEF was served upon the following counsel via electronic mail.

Denise M. De Mory
California State Bar No. 168076
Corey Johanningmeier
California State Bar No. 251297
Richard C. Lin
California State Bar No. 209233
Michael Flynn-O'Brien
California State Bar No. 291301
Bunsow De Mory LLP
701 El Camino Real
Redwood City, CA 94063
(650) 351-7241 Telephone
(415) 426-4744 Facsimile
ddemory@bdiplaw.com
cjohanningmeier@bdiplaw.com
rlin@bdiplaw.com
mflynno'Brien@bdiplaw.com
BDIP-BiTMicro-KIOXIA@bdiplaw.com

B. Russell Horton
State Bar No. 10014450
George Brothers Kincaid & Horton, L.L.P.
114 West 7th Street, Ste. 1100
Austin, Texas 78701
(512) 495-1400 Telephone
(512) 499-0094 Facsimile
rhorton@gbkh.com

Counsel for Plaintiff BiTMICRO LLC

/s/ Chad S. Campbell

**IN THE UNITED STATES DISTRICT COURT
FOR THE WESTERN DISTRICT OF TEXAS
WACO DIVISION**

BiTMICRO LLC,

Plaintiff,

v.

KIOXIA AMERICA, INC.
And KIOXIA CORPORATION,

Defendants.

Case No. 6:22-cv-00331-ADA

BiTMICRO LLC,

Plaintiff,

v.

INTEL CORPORATION,

Defendant.

Case No. 6:22-cv-00335-ADA

EXPERT DECLARATION OF DONALD ALPERT, PH.D.

I. INTRODUCTION

1. My name is Donald Alpert, PhD. I am an independent consultant and the president of Camelback Computer Architecture LLC, through which I have provided computer architecture consulting services for twenty years.

2. I have been asked by counsel for Intel Corporation to offer my opinions and analysis regarding claim construction of certain terms of U.S. Patent Number 9,858,084 (“the ’084 patent”) and U.S. Patent Number 10,120,694 (“the ’694 patent”). This report sets forth my opinions and the bases for those opinions. I reserve the right to supplement this report to the extent permitted under the rules if additional information becomes available to me. If called upon to testify, I may create demonstrative exhibits or other visual aides to assist with the presentation of my opinions.

3. I am being compensated for my work in this matter at my customary consulting rate of \$650 per hour. My compensation in no way depends on the substance of my opinions or the outcome of this proceeding.

II. PROFESSIONAL BACKGROUND

4. I have 45 years of academic and industrial experience in applying, designing, studying, teaching, and writing about microprocessors and computer systems. I received an Electrical Engineering Ph.D. degree in 1984 from Stanford University. I earlier received an Electrical Engineering B.S. degree from MIT in 1973 and an Electrical Engineering M.S. degree from Stanford University in 1978. I have taught classes in computer architecture at Stanford, Tel Aviv, and Arizona State Universities.

5. From 1976 to 1977, I worked at Burroughs Corporation, where I designed peripheral interface controllers, including those for serial data communications based on Intel 8080 microprocessor components. From 1980 to 1989, I was the lead architect for the design of three high-performance microprocessors at Zilog and National Semiconductor. Later, at Intel, I was the

lead architect of the Pentium® Processor from 1989 to 1992 and of the 815 chipset from 1999 to 2000, both of which became the most widely used PC components of their time. The 815 chipset comprised two components: (1) a memory controller hub (MCH) that included a graphics controller and memory controller with interfaces to the CPU, 133 MHz SDRAM system memory modules, an optional, external graphics controller and (2) an I/O controller hub (ICH) that included various I/O controllers (e.g., network, hard drive, USB) for system peripheral devices, as well as Direct Memory Access (DMA) controllers. Additionally, I served as co-manager for the Itanium processor design from 1993 to 1997.

6. I am a Senior Member of the Institute of Electrical and Electronics Engineers (IEEE), and served as the chair of the IEEE Technical Committee on Microprocessors and Microcomputers from 1999 to 2000. I was the keynote speaker at the first Cool Chips conference, dedicated to the study of low-power microprocessors and systems. I have given invited lectures at several universities, and published ten papers in various professional journals and conference proceedings. My paper entitled “Architecture of the Pentium Processor” was selected as best paper in IEEE Micro for 1993. I am a named inventor on over 30 U.S. patents that pertain to microprocessors, computer systems, and related technology.

7. I have reviewed the '084 and '694 patents, and I am familiar with their subject matter, which is within the scope of my education and professional experience. Based at least on my background in academia, industry, and consulting, I am familiar with the issues and technology relating to processors, chipsets, memory, and peripheral devices for computer systems. I have personally analyzed, developed, and tested such computer components and systems. More specifically, the Pentium® Processor and 815 chipset for which I was the lead architect at Intel implemented DMA controllers and various features for system initialization.

8. My curriculum vitae, including my publications and patents, is submitted herewith as Ex. A.

III. LEGAL PRINCIPLES

9. I am not a lawyer, but counsel have informed me about certain legal principles that apply in construing patent claims. Generally, I understand that the words of a claim are construed to have the plain and ordinary meaning they would have to a person of ordinary skill in the art (“POSITA”) at the time of the invention based upon the claim language itself, the patent specification including figures, and the file history of the patent and related patents. I also understand that terms must be construed based upon what the inventor actually invented and intended to be covered (as evidenced in the patent and file history) and the claim in which the terms appear.

10. I also understand that the “intrinsic evidence,” which includes the claims, the specification, and the file history, is the most important tool for determining the meaning of a claim term. I understand further that evidence such as dictionaries, textbooks, and other publications may also provide guidance about the meaning of a claim term.

11. I also understand that if a claim term, read in light of the specification and the prosecution history, fails to inform a POSITA with reasonable certainty about the scope of the invention, then the claim is indefinite.

IV. LEVEL OF ORDINARY SKILL IN THE ART

12. A person of ordinary skill in the art (“POSITA”) at the time of the alleged inventions of the ’084 and ’694 patents (which were filed on March 17, 2014) would have had a bachelor’s degree in electrical engineering or computer engineering and at least two years of experience working in the field of embedded system design, including booting of such systems, or an equivalent combination of education, work, and/or experience. My opinion as to this level of

ordinary skill in the art would be the same even if BiTMICRO contends that the time of the alleged inventions is as early as March 15, 2013, which is the filing date of the provisional application of the '084 patent.

V. DISPUTED CLAIM TERMS

Term / Patent	Plaintiff's Proposal	Intel's Proposal
"DMA descriptors" / '084 Patent: Claims 1, 19 "DMA controller descriptors" / '694 Patent: Claims 1, 6, 12	Plain and ordinary meaning; not indefinite.	"data structure including at least a source and destination address and byte count"

13. The independent claims of the '084 and '694 patents recite a "POR sequencer descriptor" that includes "Direct Memory Access (DMA) descriptors used to fetch other POR sequencer descriptor fragments," for the '084 patent claims, and "DMA controller descriptors," for the '694 patent claims.

14. Direct Memory Access (often abbreviated as "DMA") is a technique for transferring data or code in embedded systems, without using the CPU. As recognized in the textbook DESIGNING EMBEDDED HARDWARE (excerpt attached as Ex. B):

The second option in moving large amounts of data around the system [besides using the processor] is DMA. A special device, called a *DMA Controller (DMAC)*, performs high-speed transfers between memory and I/O devices. Using DMA bypasses the processor by setting up *a channel* between the I/O device and the memory. Thus, data is read from the I/O device and written into memory without the need to execute code to perform the transfer on a byte-by-byte (or word-by-word) basis.

DESIGNING EMBEDDED HARDWARE at 20; *see also* HARDWARE VERIFICATION WITH SYSTEMVERILOG (excerpt attached as Ex. C) at 32 ("DMA is a common hardware feature for transferring data from one memory location to another without putting a load on the CPU."). The EMBEDDED HARDWARE industry reference book (excerpt attached as Ex. D) illustrates typical

DMA flows, in which a DMA controller performs transfers between system memory and/or peripherals:

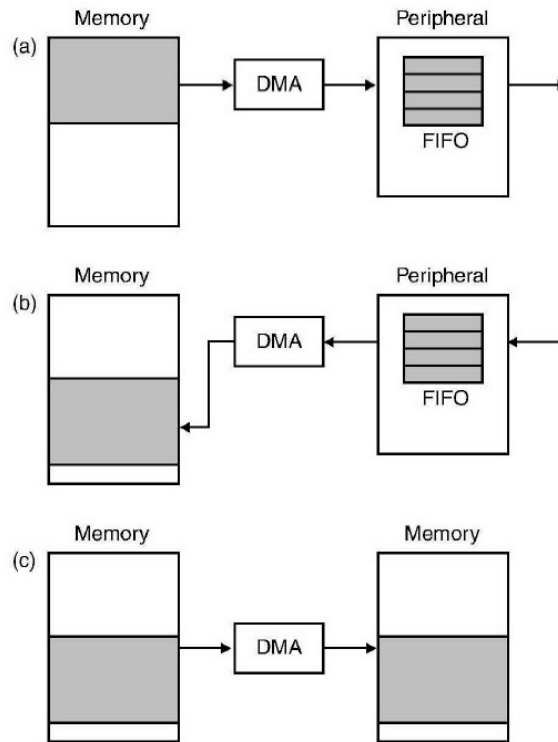


Figure 5.17: Typical DMA flows.

EMBEDDED HARDWARE at 216. DMA controllers can perform transfers of data as well as code.

15. Any DMA transfer requires that the DMA controller know at least three pieces of information: the source address of the data (or code) to be transferred, *i.e.*, where to find that data/code; the size of the data/code to be transferred, *i.e.*, how many bytes to be transferred; and the destination address where the data (or code) is to be transferred. The DMA controller must be configured with this information. If the DMA controller does not have each of these pieces of information, it cannot perform a DMA transfer. See DESIGNING EMBEDDED HARDWARE at 21 (“A DMA transfer begins by the processor configuring the DMAC for the transfer. This setup involves specifying the source, destination, and size of the data, as well as other parameters.”).

16. DMA “descriptors” are data structures that specify this information for a DMA controller. DMA descriptors frequently include other information for the DMA controller, as well. Often, DMA descriptors include pointers to other DMA descriptors, such that DMA transfers can be “specified by a linked-list in memory.” DESIGNING EMBEDDED HARDWARE at 21; *id*:

Data chaining is started by specifying a pointer to a descriptor in memory. The descriptor is a table specifying byte count, source address, destination address, and a pointer to the next descriptor. The DMAC loads the relevant information about the transfer from this table and begins moving data. The transfer continues until the number of bytes transferred is equal to the entry in the byte-count field. On completion, the pointer to the next descriptor is loaded. This continues until a null pointer is found.

17. The term “DMA descriptor” is often used in embedded system design to refer to a data structure for a DMA controller that includes a source address, destination address, byte count, and often a pointer to another descriptor. For example, many embedded systems use ARM IP, including the PL081 DMA controller. The Technical Reference Manual for the PL081 DMA controller (excerpted as Ex. E) specifies a “chain of descriptors” for the DMA controller that define blocks of data to be transferred, and each of these descriptors includes the “source start address,” the “destination address” and the “transfer size” in bytes, along with additional information. *Id.* at 96800DOC00033481-82. Similarly, the HARDWARE VERIFICATION WITH SYSTEMVERILOG text—which describes functional verification techniques for hardware systems—describes a test scenario for a “DMA chip that accepts DMA descriptors,” where “[e]ach descriptor has a source and destination memory address, as well as the number of bytes (called ‘length’) to transfer.” Ex. C at 32.

18. The ’084 and ’694 patents (which have similar disclosures; I have cited to the ’084 patent for convenience) describe using “DMA descriptors” and “DMA controller descriptors” in a manner consistent with how that term is used in the field of embedded system design. The patents

state that “DMA controllers 203 initiates and tracks data transfers.” ’084 patent at 5:3-4. To initiate and track a data transfer, a DMA controller must have the three pieces of information mentioned above: the source and destination of whatever is to be transferred, along with its size in bytes. The patents further state that “DMA descriptors are loaded by the POR Sequencer to the DMA controllers of the system” and that these DMA descriptors are used by the DMA controller to perform the transfer: “[t]he DMA controller 203 then processes the DMA controller's descriptor 406 that it received and initiates transfer of data from the storage device 209 to the RAM 208 via the storage device controller 204 in block 706.” *Id.* at 2:56-58, 7:33-37. Similarly, the patents state that DMA descriptors are used to both “fetch POR sequencer descriptor fragments” and also “to transfer system firmware” to memory. *Id.* at 7:47-50. This description confirms that, consistent with the normal understanding of “DMA descriptor,” the DMA descriptors of the ’084 and ’694 patents contain the information necessary for the DMA controller to perform a transfer—namely, source/destination address and a length in bytes. Additionally, the patents refer to DMA descriptors as “pointing to” particular information. For example, DMA descriptors are described as “point[ing] to a system firmware fragment” (*id.* at 3:40-51). Again, this indicates that the DMA descriptors include source address information. Additionally, U.S. App. No. 2005/0120146—which the ’084 and ’694 patents cite as “related art” (*id.* at 1:21, 1:56-2:8)—describes “program[ming] the DMA engine with source and target device addresses and an initial transfer length,” which allows the DMA engine to perform a DMA transfer. U.S. App. No. 2005/0120146 at [0053]. Yet again, this indicates that the applicants for the ’084/’694 patents understood, consistent with a POSITA’s understanding, that a DMA descriptor requires at least source and destination address and length (*i.e.*, number of bytes) to be transferred.

19. For these reasons, I agree with Intel that a POSITA would view “DMA descriptors” and “DMA controller descriptors,” as those terms are used in the ’084 and ’694 patent claims, to mean data structures that include at least a source address, a byte length, and a destination address.

I declare under penalty of perjury under the laws of the United States that the foregoing is true and correct.

EXECUTED this 7th day of November, 2022, in Sacramento, California.

A handwritten signature in black ink, appearing to read "Donald Alpert", is written over a horizontal line.

Donald Alpert, Ph.D.

EXHIBIT A

Donald Alpert

Camelback Computer Architecture, LLC

(916) 515-8332



Donald Alpert, Ph.D.
Camelback Computer Architecture, LLC
1809 S Street, Suite 101
Box 340
Sacramento CA 95811
(916) 515-8332

Donald Alpert, Ph.D., is an independent consultant, specializing in microprocessors and computer systems. For more than a decade in his role as president of Camelback Computer Architecture LLC, Dr. Alpert has provided clients with expert services for patent litigation, technology assessment, and product development. He has over 40 years of wide-ranging academic and industry experience that enable him to provide expert consulting services in the following areas:

- Computers, Computer Systems
- Computer architecture
- Microprocessors
- Semiconductors
- PC Systems
- Embedded Systems
- Design Validation and Debug
- Performance Evaluation
- Cache Memory
- Instruction Sets
- Media Processing
- Power Management

Patent Analysis: Dr. Alpert has served as an expert on dozens of patent litigation cases, handling both infringement and invalidity issues. Dr. Alpert has worked as both a consulting expert and testifying expert assisting attorneys representing major high-tech corporations. He has substantial experience testifying in depositions and at trials before U.S. Federal Courts and the International Trade Commission. Dr. Alpert also consults to assess patents for acquisition and for pre-litigation review. See [here](#) for a summary of testifying experience.

Product Development: Dr. Alpert is a widely recognized innovator with over 30 patents. Dr. Alpert has worked at some of the most ground-breaking technology firms, including Zilog, National Semiconductor, and Intel, where he was the principal architect of the Pentium® Processor. See [here](#) for more information and example projects.

Donald Alpert

Camelback Computer Architecture, LLC

(916) 515-8332

Employment Experience

Camelback Computer Architecture, LLC, Sacramento, CA

Consultant

2/02-Present

Computer architecture consulting services to assess technology, develop products, and protect IP.

Intel Corporation, Chandler, Arizona

Principal Engineer

12/99-1/02

Led architecture team of 10 responsible for definition and performance evaluation of Itanium™ Processor Family microprocessors.

Principal Engineer

5/98-12/99

Led architecture team of 4 responsible for technical strategy and product definition of PC graphics processors and chipsets.

Stanford University, Stanford, California

Consulting Assistant Professor

9/97-4/98

Responsible for teaching undergraduate and graduate classes in computer architecture and participating in processor-design research projects.

Intel Corporation, Santa Clara, California

Engineering Manager

10/93-6/97

Responsible for microarchitecture definition and architecture validation of the Itanium™ microprocessor, which was the first implementation of Intel's 64-bit extension architecture. Led a team of 60 engineers.

Architecture Manager

1/89-10/93

Responsible for the Pentium® Processor technical definition. Worked with customers and Intel marketing to identify product requirements and with the design engineering team to evaluate design options. Led a team of seven engineers responsible for creating technical specifications and for modeling and simulating performance using traces from DOS, Windows, and UNIX execution. After completing the Pentium Processor, led an interdisciplinary team of 30 engineers to define a new 64-bit instruction set architecture.

National Semiconductor Corporation, Herzlia, Israel

Architecture Manager

5/85-1/89

Responsible for defining new microprocessors in the Series 32000 family, including the NS32532 and "Swordfish" high-performance microprocessors. Led a team of five engineers working in the following areas: modeling and simulating high-performance microprocessors, evaluating novel computer architectures, and writing technical specifications for new products.

Zilog, Incorporated, Campbell, California

Component Architect

6/80-4/85

Worked on the Z80,000 CPU, a 32-bit microprocessor that integrated an instruction pipeline, cache, and memory management unit. Responsible for specifying the non-privileged instruction set, evaluating performance, and writing the reference manual for the microprocessor.

Burroughs Corporation, Pasadena, California

Associate Engineer

7/76-7/77

Designed computer peripheral interface controllers using MSI-TTL and 8080 microprocessors.

Alpex Computer Corporation, Danbury, Connecticut

8/75-6/76

Consulting engineer working on design of video games.

Action, Washington, DC

6/73-6/75

Peace Corps volunteer teaching mathematics and physics in a secondary school in Ghana, West Africa.

Donald Alpert Camelback Computer Architecture, LLC (916) 515-8332

Education

Stanford University 9/77-6/84

Graduate study in computer design. National Science Foundation Graduate Fellowship.
Awarded MSEE in 1978 and PhDEE in 1984.

M.I.T. 9/69-6/73

Member of Tau Beta Pi and Eta Kappa Nu. Awarded BSEE in 1973.

Professional Activities

Member of Microprocessor Report editorial board (2002-present)

Anthem College Commencement Address (18-Dec-2009)

Panel session about history of microprocessor development at National Semiconductor Corporation, Computer History Museum (2008).

[\[http://archive.computerhistory.org/resources/access/text/Oral_History/102658246.05.01.acc.pdf\]](http://archive.computerhistory.org/resources/access/text/Oral_History/102658246.05.01.acc.pdf)

IEEE Technical Committee on Microprocessors and Microcomputers, Chair (1999-2000) and Vice-Chair (2001-present).

Lectures about the Itanium™ Processor Family at U.C. Irvine (1999, 2000).

Cool Chips I, Keynote speaker (Tokyo, 1998)

IEEE Hot Chips Symposium Program Co-chair (1994), member of program committee (1993, 1995).

Lectures about the Pentium® Processor at Stanford University, University of Illinois, and Cornell (1993-1994).

Appeared in Stanford UVC Educational Video production about the Pentium Processor (1993).

IEEE International Symposium on Computer Architecture member of program committee (1996) and Session Chair for Cache Design (1989).

Teaching Experience

Faculty Associate at Arizona State University teaching undergraduate class on assembly language programming (Spring/2001) and computer architecture (Spring/2002).

Consulting Associate Professor at Stanford University teaching undergraduate and graduate classes in computer architecture (Fall/1997, Winter/1998).

Consulting Professor at Tel Aviv University teaching graduate class in computer architecture (Spring/1987)

Peace Corps volunteer in a secondary school in Ghana, West Africa teaching mathematics and physics (1973-1975).

**Donald Alpert
Publications**

Camelback Computer Architecture, LLC

(916) 515-8332

Don Alpert, "How Microprocessors Upset the Computer Industry," Microprocessor Report, December 11, 2006.

Don Alpert, "Evolving PC Chip-Set Topologies," Microprocessor Report, April 17, 2006.

Don Alpert, "Will Microprocessors Become Simpler?" Microprocessor Report, November 17, 2003.

Don Alpert, "Itanium Processor Status Report," Microprocessor Report, July 28, 2003.

Don Alpert, "Scalable MicroSupercomputers," Microprocessor Report, March 17, 2003.

Don Alpert, "ColdFire Goes Fully Superscalar," Microprocessor Report, October 28, 2002.

Donald Alpert and Alan Jay Smith, "Guest Editor Introduction," IEEE Micro special issue on Hot Chips VI, April 1995.

Donald Alpert and Dror Avnon, "Architecture of the Pentium Processor," IEEE Micro, June 1993, pp. 11-21. Selected best paper in IEEE Micro for 1993.

D. Alpert, A. Averbuch, and O. Danielli, "Performance Comparison of Load/Store and Symmetric Instruction Set Architectures," Proceedings, 18th International Symposium on Computer Architecture, June 1990, pp. 172-181.

Benjamin Maytal, Sorin Iacobovici, Donald Alpert, et al, "Design Considerations for a General-Purpose Microprocessor," Computer, January 1989, pp. 66-76.

Donald Alpert and Michael Flynn, "Performance Trade-offs for Microprocessor Cache Memories," IEEE Micro, August 1988, pp. 44-54.

D. Alpert, J. Levy, and B. Maytal, "Architecture of the NS32532 Microprocessor," Proceedings, 1987 IEEE International Conference on Computer Design, October 1987, pp. 168-172. Awarded best presenter at Design and Test sessions.

D. Alpert, D. Biran, L. Epstein, et al, "Trends in VLSI Microprocessor Design," Proceedings, First Annual Conference on Computer Technology, Systems and Applications (CompEuro '87), May 1987, pp. 564-567.

Y Sidi, D. Alpert, D. Biran, et al, "Design Considerations of an Advanced 32-Bit Microprocessor," Proceedings, 15th Conference of Electrical and Electronic Engineers in Israel, April 1987.

D. Alpert, "Trends in VLSI Microprocessor Design," Proceedings, 15th Conference of Electrical and Electronic Engineers in Israel, April 1987.

Donald Alpert, Michael J. Flynn, and Scott Wakefield, "Directly Executed Languages for VLSI Processor Design," Proceedings, 1983 IEEE International Conference on Computer Design, October 1983, pp. 609-612.

Donald Alpert, "Powerful 32-Bit Micro Includes Memory Management," Computer Design, October 1983, pp. 213-220.

Don Alpert, Dean Carberry, Mike Yamamura, et al, "32-Bit Processor Chip Integrates Major System Functions," Electronics, July 14, 1983, pp. 113-119.

Donald Alpert

Camelback Computer Architecture, LLC

(916) 515-8332

Patents

4,802,085, Apparatus and method for detecting and handling memory-mapped I/O by a pipelined microprocessor

5,249,286, Selectively locking memory locations within a microprocessor's on-chip cache

5,263,153, Monitoring control flow in a microprocessor

5,416,913, Method and apparatus for dependency checking in a multi-pipelined microprocessor

5,442,756, Branch prediction and resolution apparatus for a superscalar computer processor

5,475,824, Microprocessor with apparatus for parallel execution of instructions

5,479,652, Microprocessor with an external command mode for diagnosis and debugging

5,481,751, Apparatus and method for storing partially-decoded instructions in the instruction cache of a CPU having multiple execution units

5,559,986, Interleaved cache for multiple accesses per clock cycle in a microprocessor

5,606,676, Branch prediction and resolution apparatus for a superscalar computer processor

5,617,554, Physical address size selection and page size selection in an address translator

5,621,886, Method and apparatus for providing efficient software debugging

5,638,525, Processor capable of executing programs that contain RISC and CISC instructions

5,657,253, Apparatus for monitoring the performance of a microprocessor

5,659,679, Method and apparatus for providing breakpoints on taken jumps and for providing software profiling in a computer system

5,669,011, Partially decoded instruction cache

5,671,435, Technique for software to identify features implemented in a processor

5,675,825, Apparatus and method for identifying a computer microprocessor

5,692,167, Method for verifying the correct processing of pipelined instructions including branch instructions and self-modifying code in a microprocessor

5,729,724, Adaptive 128-bit floating point load and store operations for quadruple precision compatibility

5,740,413, Method and apparatus for providing address breakpoints, branch breakpoints, and single stepping

5,764,959, Adaptive 128-bit floating point load and store instructions for quad-precision compatibility

5,774,686, Method and apparatus for providing two system architectures in a processor

5,790,834, Apparatus and method using an ID instruction to identify a computer microprocessor

5,802,605, Physical address selection and page size selection in an address translator

5,958,037, Apparatus and method for identifying features and the origin of a computer microprocessor

5,991,874, Conditional move using a compare instruction generating a condition field

6,052,801, Method and apparatus for providing breakpoints on a selectable address range

6,219,774, Address translation with/bypassing intermediate segmentation translation to accommodate two different instruction set architecture

6,385,718, Computer system and method for executing interrupt instructions in operating modes

6,408,386, Method and apparatus for providing event handling functionality in a computer system

6,584,558, Article for providing event handling functionality in a processor supporting different instruction sets

7,010,671, Computer system and method for executing interrupt instructions in two operating modes

7,389,403, Adaptive computing ensemble microprocessor architecture

7,802,073, Virtual core management

EXHIBIT B

SECOND EDITION

Designing Embedded Hardware

John Catsoulis

O'REILLY®

Beijing • Cambridge • Farnham • Köln • Sebastopol • Tokyo

96800DOC00033148

Petitioner Intel Corp., Ex. 1012
IPR2023-00783

Designing Embedded Hardware, Second Edition

by John Catsoulis

Copyright © 2005, 2002 O'Reilly Media, Inc. All rights reserved.
Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*safari.oreilly.com*). For more information, contact our corporate/institutional sales department: (800) 998-9938 or *corporate@oreilly.com*.

Editor: Andy Oram
Production Editor: Sanders Kleinfeld
Cover Designer: Emma Colby
Interior Designer: David Futato

Printing History:

November 2002: First Edition.
May 2005: Second Edition.

Nutshell Handbook, the Nutshell Handbook logo, and the O'Reilly logo are registered trademarks of O'Reilly Media, Inc. *Designing Embedded Hardware*, the image of a porcelain crab, and related trade dress are trademarks of O'Reilly Media, Inc.

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and O'Reilly Media, Inc. was aware of a trademark claim, the designations have been printed in caps or initial caps.

While every precaution has been taken in the preparation of this book, the publisher and author assume no responsibility for errors or omissions, or for damages resulting from the use of the information contained herein.

ISBN: 978-0-596-00755-3
[LSI]

[2011-04-08]

96800DOC00033149

Petitioner Intel Corp., Ex. 1012
IPR2023-00783

Input/Output

The address space of the processor can contain devices other than memory. These are input/output devices (I/O devices, also known as *peripherals*) and are used by the processor to communicate with the external world. Some examples are serial controllers that communicate with keyboards, mice, modems, etc.; parallel I/O devices that control some external subsystem; or disk-drive controllers, video and audio controllers, or network interfaces.

There are three main ways in which data may be exchanged with the external world:

Programmed I/O

The processor accepts or delivers data at times convenient to it (the processor).

Interrupt-driven I/O

External events control the processor by requesting the current program be suspended and the external event be serviced. An external device will interrupt the processor (assert an interrupt control line into the processor), at which time the processor will suspend the current task (program) and begin executing an interrupt service routine. The service of an interrupt may involve transferring data from input to memory, or from memory to output.

Direct Memory Access (DMA)

DMA allows data to be transferred from I/O devices to memory directly without the continuous involvement of the processor. DMA is used in high-speed systems, where the rate of data transfer is important. Not all processors support DMA.

DMA

DMA is a way of streamlining transfers of large blocks of data between two sections of memory, or between memory and an I/O device. Let's say you want to read in 100M from disk and store it in memory. You have two options.

One option is for the processor to read one byte at a time from the disk controller into a register and then store the contents of the register to the appropriate memory location. For each byte transferred, the processor must read an instruction, decode the instruction, read the data, read the next instruction, decode the instruction, and then store the data. Then the process starts over again for the next byte.

The second option in moving large amounts of data around the system is DMA. A special device, called a *DMA Controller (DMAC)*, performs high-speed transfers between memory and I/O devices. Using DMA bypasses the processor by setting up a *channel* between the I/O device and the memory. Thus, data is read from the I/O device and written into memory without the need to execute code to perform the transfer on a byte-by-byte (or word-by-word) basis.

In order for a DMA transfer to occur, the DMAC must have use of the address and data buses. There are several ways in which this could be implemented by the system designer. The most common approach (and probably the simplest) is to suspend the operation of the processor and for the processor to “release” its buses (the buses are tristate). This allows the DMAC to “take over” the buses for the short period required to perform the transfer. Processors that support DMA usually have a special control input that enables a DMAC (or some other processor) to request the buses.

There are four basic types of DMA:

Standard block transfer

Accomplished by the DMA controller performing a sequence of memory transfers. The transfers involve a load operation from a source address followed by a store operation to a destination address. Standard block transfers are initiated under software control and are used for moving data structures from one region of memory to another.

Demand-mode transfers

Similar to standard mode except that the transfer is controlled by an external device. Demand-mode transfers are used to move data between memory and I/O or vice versa. The I/O device requests and synchronizes the movement of data.

Fly-by transfer

Provides high-speed data movement in the system. Instead of using multiple bus accesses as with conventional DMA transfers, fly-by transfers move data from source to destination in a single access. The data is not read into the DMAC before going to its destination. During a fly-by transfer, memory and I/O are given different bus control signals. For example, an I/O device is given a read request at the same time that memory is given a write request. Data moves from the I/O device straight into the memory device.

Data-chaining transfers

Allow DMA transfers to be performed as specified by a linked-list in memory. Data chaining is started by specifying a pointer to a *descriptor* in memory. The descriptor is a table specifying byte count, source address, destination address, and a pointer to the next descriptor. The DMAC loads the relevant information about the transfer from this table and begins moving data. The transfer continues until the number of bytes transferred is equal to the entry in the byte-count field. On completion, the pointer to the next descriptor is loaded. This continues until a null pointer is found.

To illustrate the use of DMA, let's consider the example of a fly-by transfer of data from a hard-disk controller to memory. A DMA transfer begins by the processor configuring the DMAC for the transfer. This setup involves specifying the source, destination, and size of the data, as well as other parameters. The disk controller generates a request for service to the DMAC (not the processor). The DMAC then

generates a HOLD or BR (bus request) to the processor. The processor completes the current instruction; places the address, control, and data buses in a high-impedance state (*floats*, tristates, or releases them); and responds to the DMAC with a HOLD-acknowledge or BG (bus granted) and enters a dormant state. Upon receiving a HOLD-acknowledge, the DMAC places the address of the memory location where the transfer to memory will begin onto the address bus and generates a WRITE to the memory while the disk controller places the data on the data bus. Hence, a direct memory access is accomplished from the disk controller to the memory.

In a similar fashion, transfers from memory to I/O devices are also possible. DMACs are capable of handling block transfers of data. The DMAC automatically increments the address on the address bus to point to each successive memory location as the I/O device generates (or receives) data. Once the transfer is complete, the buses are returned to the processor and it resumes normal operation.

Not all DMA controllers support all forms of DMA. Some DMA controllers simply read data from a source, hold it internally, and then store it to a destination. They perform the transfer in exactly the same way that a processor would. The advantage in using a DMA controller instead of a processor is that if the transfer were to be performed by the processor, each transfer would still have program fetches associated with it. Thus, even though the transfer takes place by sequential reads and writes, the DMA controller does not also have to fetch and execute code, thereby providing a faster transfer than a processor.

Support for DMA is normally not found in small microcontrollers. Some mid-range processors (16-bit, low-end 32-bit) may have DMA support. All high-end processors (32-bit and above) will have DMA support, and many include a DMA controller on-chip. Similarly, peripherals intended for small-scale computers will not provide DMA support, whereas peripherals intended for high-speed and powerful computers definitely *will* have DMA support.

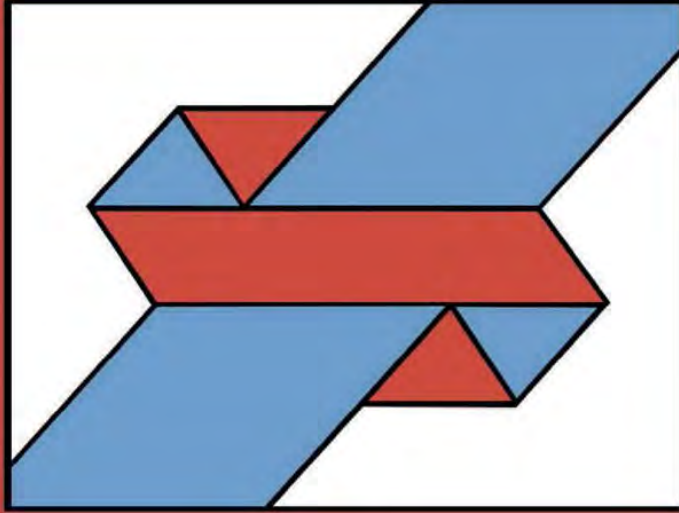
Parallel and Distributed Computers

Some embedded applications require greater performance than is achievable from a single processor. For cost reasons, it may not be practical to implement a design with the latest superscalar RISC processor, or perhaps the application lends itself to distributed processing where the tasks are run across several communicating machines. It may make more sense to use a fleet of lower-cost processors, distributed throughout the installation. It is becoming increasingly common to see embedded systems implemented using parallel processors.

Introduction to parallel architectures

The traditional architecture for computers follows the conventional, Von Neumann serial architecture. Computers based on this form usually have a single, sequential

EXHIBIT C



Hardware Verification with SystemVerilog

An Object-Oriented Framework

Mike Mintz
Robert Ekendahl

 Springer

96800DOC00033113

Petitioner Intel Corp., Ex. 1012
IPR2023-00783

Hardware Verification with SystemVerilog

An Object-Oriented Framework

Mike Mintz
Robert Ekendahl

Hardware Verification with SystemVerilog

An Object-Oriented Framework

Cover art from the original painting “Dementia #10” by John E.
Bannon, johnebannon.com

 Springer

Mike Mintz
Harvard, MA
USA

Robert Ekendahl
Somerville, MA
USA

Library of Congress Control Number: 2007923923

ISBN 0-387-71738-2

e-ISBN 0-387-71740-4

ISBN 978-0-387-71738-8

e-ISBN 978-0-387-71740-1

Printed on acid-free paper.

© 2007 Springer Science+Business Media, LLC.

All rights reserved. This work may not be translated or copied in whole or in part without the written permission of the publisher (Springer Science+Business Media, LLC, 233 Spring Street, New York, NY 10013, USA), except for brief excerpts in connection with reviews or scholarly analysis. Use in connection with any form of information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed is forbidden. The use in this publication of trade names, trademarks, service marks and similar terms, even if they are not identified as such, is not to be taken as an expression of opinion as to whether or not they are subject to proprietary rights.

9 8 7 6 5 4 3 2 1

springer.com

96800DOC00033116

Petitioner Intel Corp., Ex. 1012
IPR2023-00783

*For Joan, Alan, and Brian.
Thanks again for your patience.*

Mike

*For Chantal.
Thanks again for your understanding,
love, and active support.*

*And to Newton—and now Darwin.
For many more missed walks.*

Robert

Contents

Preface xix

Acknowledgments xxi

Chapter 1: Introduction 1

 Background 3

 What is Functional Verification? 4

 Why Focus on SystemVerilog? 5

 A Tour of the Handbook 5

 For Further Reading 6

Part I:

SystemVerilog and Verification

(The Why and How) 7

Chapter 2: Why SystemVerilog? 9

 Overview 10

 SystemVerilog as a Verification Language 11

 Main Benefits of Using SystemVerilog 13

 Drawbacks of Using SystemVerilog 13

 SystemVerilog Traps and Pitfalls 14

 SystemVerilog is not Verilog 14

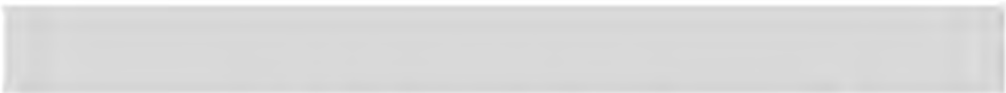
 Errors and run-time crashes 15

Contents

Five languages in one!	15
The assertions language	15
The constraint language	16
The coverage language	18
SystemVerilog features not discussed	19
Summary	20
For Further Reading	20

Chapter 3: OOP and SystemVerilog 23

Overview	24
The Evolution of OOP and SystemVerilog	25
Assembly programming: The early days	25
Procedural languages: The next big step	25
OOP: Inheritance for functionality	26
OOP: Inheritance for interface	28
A word or two about “interface”	28
The Evolution of Functional Verification	29
Verification through inspection	29
Verification through randomness	29
The emergence of hardware verification languages	30
OOP: A current trend in verification	31
OOP: A possible next step	31
OOP and SystemVerilog	32
Data abstraction through classes	32
A DMA descriptor example	32
Access control	33
Constructors	34
Member methods and variables	35
Inheritance for functionality	36
Inheritance for code interface	37
What’s a header file?	39
Packages	40
Separating HDL and testbench code	42



Wiggling wires: the interface concept. 42

Building and using interfaces 44

Summary 46

For Further Reading 46

Chapter 4: A Layered Approach 47

Overview. 48

A Whiteboard Drawing 50

 An “ends-in” approach. 51

 Refining the whiteboard blocks 52

The “Common-Currency” Components 52

The Component Layer in Detail 53

 The connection layer. 54

 The agent layer 56

 The transaction layer 57

The Top-Layer Components 58

What is a Test? 60

The Test Component 62

The Test Irritator 64

A Complete Test 65

Summary 67

For Further Reading 67

Part II:

An Open-Source Environment with SystemVerilog69

Chapter 5: Teal Basics 71

Overview. 72

Teal’s Main Components. 72

Using Teal 74

 A simple test 74

Logging Output. 74

Using Test Parameters 77

Accessing Memory 79

Contents

A memory example	80
Constrained Random Numbers	84
Required initialization	84
Using random numbers	85
Working with Simulation Events	86
Summary	87

Chapter 6: Truss: A Standard Verification Framework	89
Overview	90
General Considerations	91
SystemVerilog considerations	91
Keeping it simple	92
Major Classes and Their Roles	93
Key test algorithm: The “dance”	94
The verification_component Virtual Base Class	97
Detailed Responsibilities of the Major Components	98
The testbench class	99
Watchdog timer	101
Test class	102
Test Component and Irritator Classes	106
The test component virtual base class	106
An AHB example	108
Test-component housekeeping functionality	109
The irritator virtual base class	110
Using the irritator	112
Summary	113



Chapter 7: Truss Flow 115

Overview 116

About truss_verification_top.sv 116

The Test Component Dance 119

The Irritator Dance 121

Compiling and Running Tests. 122

 The truss run script 123

 Switches 124

 Using “-f” files 125

The First Test: A Directed Test 125

The Second Test:

Adding Channels and Random Parameters 127

 The channel pseudo-templated classes 128

 Building the second test. 129

 Building the second test’s test_component 131

 Adjusting the second test’s parameters 132

The Remaining Tests:

Mix-and-Match Test Components 135

Summary 136

Chapter 8: Truss Example 137

Overview 138

Directory Structure 138

Theory of Operation 140

Running the Simple ALU Example 142

Points of Interest 142

Power-on Reset 143

Driver and Monitor Protocol 144

The alu_test_component 145

Checking the Chip 146

Completing the Test 147

Summary 149

Contents

Part III:
Using OOP for Verification
(Best Practices). 151

Chapter 9: Thinking OOP 153

Overview. 154

Sources of Complexity 155

Essential complexity vs.
implementation complexity 155

Flexibility vs. complexity 156

Apparent simplicity vs.
hiding inherent complexity. 159

Example: How hiding complexity
can create confusion. 159

Example: How apparent simplicity
leads to later problems 160

Team dynamics 162

Team roles 162

Using a “code buddy” 163

Creating Adaptable Code 163

Achieving adaptability 163

Why is adaptability tricky? 164

Architectural Considerations
to Maximize Adaptability 165

Changes are easy—or just plain impossible. 166

Where is adaptation likely to happen? 167

Separating Interface from Implementation 168

Code Interface, Implementation, and Base Classes . . .169

Summary. 170

For Further Reading 171



Chapter 10: Designing with OOP 173

Overview 174

Keeping the Abstraction Level Consistent 174

Using “Correct by Construction” 176

The Value of Packages 178

Data Duplication—A Necessary Evil 180

Designing Well, Optimizing Only When Necessary 181

Using the Protocol, Only the Protocol 182

Verification Close to the Programming Model 183

The Three Parts of Checking 184

Separating the Test from the Testbench 186

Summary 187

For Further Reading 188

Chapter 11: OOP Classes 189

Overview 190

Defining Classes 191

How Much Electricity? 191

 Classes 192

 Packages 192

 Pointers and virtual functions 192

Global Services 193

 Package it up! 193

 Static methods 194

 Singletons—A Special Case of Static Methods 194

 Packages or static methods? 195

 Other considerations 196

Class Instance Identifiers 197

 Strings as identifiers 197

 Static integers as identifiers 197

 Combination identifiers 198

Class Inheritance for Reuse 198

 A BFM base-class example 199

Contents

A BFM agent class	200
Reusing the BFM class	200
Class Inheritance for Code Interfaces	201
Inheritance for a verification component	201
Inheritance for a payload code interface.	202
Summary	203
For Further Reading	204
 Chapter 12: OOP Connections	 205
Overview	206
How Tight a Connection?	207
Types of Connections	209
Peer-to-peer connections	209
Master-to-slave and push-vs.-pull connections	209
Two Tight Connection Techniques	211
Using pointers	211
Using inheritance	212
Threads and Connections	214
Events—explicit blocking interconnects	214
Hiding the thread block in a method	216
Fancier Connections	217
Listener or callback connections	218
Channel connections	219
Action object connections	220
Summary	221
For Further Reading	222

Chapter 13: Coding OOP	223
Overview	224
“If” Tests—A Necessary Evil	224
“If” tests and abstraction levels	225
“If” tests and code structure	226
Repeated “if” expressions	227
“If” tests and factory functions	228
A factory function example	229
Coding Tricks	232
Coding only what you need to know	232
Reservable resources	233
The register: an int by any other name	234
Using data members carefully	234
Coding Idioms	236
The singleton idiom	237
Public nonvirtual methods:	
Virtual protected methods	238
Enumeration for Data, Integer for Code Interface	240
What’s in a Name?	241
Keeping class name the same as file name	241
Keeping class and instance names related	241
Coding with Style	242
Proceeding with caution	243
General syntax conventions	243
Identifying local and protected members	244
Summary	245
For Further Reading	246

Contents

Part IV:
Examples
(Putting It All Together) 247

Chapter 14: Block-Level Testing 249

Overview 250

Theory of Operation 251

Verification environment 252

Verification IP 253

 UART VIPs 253

 Wishbone VIP 254

The verification dance 255

Running the UART Example 255

Points of Interest 256

Configuration 256

 VIP UART package 257

 VIP UART configuration class 258

 Randomization of parameters 258

 UART 16550 configuration class 260

Configuring the Chip 261

 Register access 262

 The wishbone_memory_bank and wishbone_driver 263

Traffic Generation 265

 The generator_agent and uart_bfm_agent classes 265

The Checker 267

 Checking the data 268

Connecting It All Together 270

 The testbench 270

 Building the channels 271

 Building the configuration and interface port 271

 Building the component-layer objects 273

 The wishbone objects 274

 The test component 275

The uart_basic_test_component::do_randomize() method.....	277
The basic data test	278
More Tests.....	280
Summary.....	280
 Chapter 15: Chip-Level Testing	281
Overview.....	282
Theory of Operation	282
Verification environment	283
Running the UART Example.....	284
The quad_uart_test_components Test	284
The quad_uart_irritators Test	286
UART irritator class.....	286
The test	288
The quad_uart_vectors Test	292
The block_uart Test	293
Summary.....	293
 Chapter 16: Things to Remember.....	295
Part I: Use SystemVerilog and Layers!	296
Part II: An Open-Source Approach	296
Part III: OOP—Best Practices	297
Part IV: Examples—Copy and Adapt!	298
Conclusion to the Conclusion.....	298
 Index	301

Preface

This is the second of our books designed to help the professional verifier manage complexity. This time, we have responded to a growing interest not only in object-oriented programming but also in SystemVerilog. The writing of this second handbook has been just another step in an ongoing masochistic endeavor to make your professional lives as painfree as possible.

The authors are not special people. We have worked in several companies, large and small, made mistakes, and generally muddled through our work. There are many people in the industry who are smarter than we are, and many coworkers who are more experienced. However, we have a strong desire to help.

We have been in the lab when we bring up the chips fresh from the fab, with customers and sales breathing down our necks. We've been through software bring-up and worked on drivers that had to work around bugs¹ in production chips.

What we feel makes us unique is our combined broad experience from both the software and hardware worlds. Mike has over 20 years of experience from the software world that he applies in this book to hardware verification. Robert has over 12 years of experience with hardware verification, with a focus on environments and methodology.

What we bring to the task of functional verification is over three decades of combined experience, from design, verification, software development, and management. It is our experiences that speak in this handbook. It is our desire that others might learn and benefit from these experiences.

We have had heated discussions over each line of code in this book and in our open-source libraries. We rarely agree at first, but by having to argue our cases we arrive at what we feel are smart, efficient, flexible, and simple solutions. Most of these we have "borrowed" from the software industry but have applied to the field of verification.

We believe that the verification industry can benefit from the lessons learned from the software domain. By using industry-standard languages, the verification domain can adapt techniques and code from over twenty calendar years

¹. Features.

of software effort, the scope of which is nothing short of stunning. Many brilliant people have paved the way in the software field. Although the field of verification is much younger, we could benefit greatly from listening, learning, and adapting mature programming techniques to the production of products of the highest quality.

So why do we provide open-source software at our website, www.trusster.com? Open-source software is a key to uniting and increasing the productivity of our industry. There is almost no successful closed-source (“hard macro”) intellectual property (IP), for a good reason. Without the ability to look at the source and edit as necessary, the task is much more difficult and the chances for success are slim.

We hope that you enjoy this book—and better yet, find its principles increasingly useful in daily practice. We look forward to your comments. Please keep in touch with us at www.trusster.com.

Mike Mintz

Robert Ekendahl

Cambridge, Massachusetts, USA

March 2007

Chapter 3: OOP and SystemVerilog

However, we are getting ahead of ourselves, so before we dive into the practical problem of verification, let's look at how SystemVerilog expresses OOP techniques.

OOP and SystemVerilog

This section shows how SystemVerilog expresses the OOP concepts described above. It describes some of the techniques we use to build a successful verification environment in later chapters. For engineers experienced with other OOP languages such as C++, Vera or “e,” this chapter can serve as a way to map concepts from one language to another.

Data abstraction through classes

Using classes to express data abstraction is an important technique in building large verification systems. Data abstraction, by grouping the data and the operations together, allows engineers to reason about the code.

We will look at a direct memory access (DMA) descriptor class to show how a class can be constructed, then evolved by means of inheritance.

A DMA descriptor example

DMA is a common hardware feature for transferring data from one memory location to another without putting a load on the CPU. In this example, we verify a DMA chip that accepts DMA descriptors, puts them into an on-chip memory array, and then executes them. Each descriptor has a source and destination memory address, as well as the number of bytes (called “length”) to transfer.

In the verification environment, a DMA descriptor could be represented by a small class. The DMA generator is then responsible for building, or *instantiating*, DMA descriptors and “pushing” them to the chip and to the checker.

The following code describes the DMA descriptor class:

```
class descriptor;
  //Constructor
  extern function new (int src, int dest,
                      int length, int status);
  //Code (or Public) Interface:
  extern virtual function void print();
  int source_address_; //public as an example!
  extern virtual function bit equal(descriptor d);
  //Implementation (or local and protected) interface:
  extern local virtual function int unique_id ();
  protected int destination_address_;
  protected int length_;
  protected int status_;
  protected int verif_id_;
endclass
```

The descriptor class is divided into the code (or public) interface and the implementation (or local and protected) interface, as shown by the *access control* level on each line. The next section will explain what access control labels are for.

Access control

The keywords *protected* and *local*, as used in the descriptor class, are SystemVerilog *access control* labels. They indicate how methods and variables following the statements can be used. The absence of a label indicates that the methods and variables that follow are publicly accessible by any code that has access to an instance of the class.¹ A *local* label indicates that only the code inside the class itself can access the variable or method.

The keyword *protected* indicates a private variable or method that can be modified through inheritance. Public, local, and protected can be used to express and enforce the intent of the class quite clearly.

¹. This is an unfortunate default, as a vast majority of classes will have far fewer public methods and variables compared to the local and protected code used to implement the class. Be prepared to type “local” and “protected” a lot.

Chapter 3: OOP and SystemVerilog

Access control is needed to help separate the user or code interface from both the internal methods and the data needed to implement the class. Consequently, the *public* section of a class declaration is the “code interface.” These are the interesting methods and variables to look at when you want to *use* a new class. When you *implement* a class, on the other hand, you also need a space to store the “state” of your class between method calls. This is done in local or protected scope.

Implementing a class is similar to implementing a state machine, where each method call changes the state of the state machine (that is, modifies the data members of the class). This “change of state” must be recorded somehow. Variables for tracking the state as well as intermediate methods should be put in the *local* scope, not only to indicate to users that they shouldn’t focus on these methods and variables, but also to protect these variables from accidentally being modified. When a class is instantiated, only the public methods and calls can be accessed. Trying to access local scope results in an error during compilation. This is an example of how language enforces the “intent” of the class.

Enforcing intent can (and should) go beyond protecting state variables. For example, instead of printing an error message during run-time, when the code calls internal implementation-detail methods, one should declare those methods to be in local scope, so that a compile error occurs instead.¹

Constructors

When a class is instantiated, the special function `new()` is called. This special function is the *constructor*. A constructor is used to initialize member variables, reserve memory, and initialize the class.

So how do you actually create an instance of a class? Consider our descriptor class for a moment. The class could be instantiated as follows:

¹. Note that SystemVerilog does access checking first, then resolution checking. This is unfortunate, as it means the code can behave differently when the access control is changed. This problem is discussed in detail in the book *The Design and Evolution of C++*.

```

descriptor descriptor1 = new (source_addr,
                             destination_addr,
                             source, length);

descriptor descriptor2;
descriptor2 = descriptor1; //point to the same object!
descriptor descriptor3; //Careful: null pointer!

```

The first line declares `descriptor1`, and calls the constructor method as declared above, passing in variables as necessary. Next, `descriptor2` is created and then is just assigned the pointer to `descriptor1`, meaning that `descriptor2` is exactly the same as `descriptor1`. Note that the next descriptor, `descriptor3`, is not initialized at all, so SystemVerilog assigns a special keyword, `null`, to the variable, because constructors are not automatically called in SystemVerilog.

Each line is valid SystemVerilog, but this might not be what you intended.

Be aware that there are different ways of initializing an instance pointer.

When writing a class, try to express the intent of the class so that an unintended use of your class generates a compile error. Though annoying, compile errors are much easier to understand than run-time errors. Similarly, when you get an unexpected compile error, don't see it as an annoyance, but rather realize that the person who wrote the class may be trying to tell you something.

Member methods and variables

In the `descriptor` class example, a few member variables and member methods are declared. The member variables are simply integers for the fields of the DMA transaction. The `print()` method will simply print all the current fields of the projects. Member variables and methods can be accessed like this:

```

descriptor d1 = new ('h68000, 'h20563, 39, 0);
d1.source_address_ = 'h586;
d1.print ();

```

Inheritance for functionality

By using class inheritance, you can create larger and larger functional blocks, building upon existing functionality. By inheriting from another class, you are saying, “I want to start from the functionality of an existing class and expand upon or change it with the features I define in my new class.”

Consider our DMA project again. In the first generation of the product (as described above), the chip would simply store DMA descriptors in an array and signal when the array was full. For the second generation, this has been improved and the chip now implements a linked list, storing each descriptor in off-chip memory.

To enable this functionality, a hardware pointer field must be added for each descriptor. As a technique, the pointer can be set to 0 to stop the chip from processing, or it can be set to the first descriptor to implement a ring.

Instead of copying and editing the descriptor class, we can simply inherit from and expand upon the base descriptor class. This is called *inheritance for functionality*.

To create our new, fancy `linked_list_descriptor` class, we could declare it like this:

```
//in linked_list_descriptor.svh
#include "descriptor.svh"
class linked_list_descriptor extends descriptor;
    extern function new (uint32 src, uint32 dest,
                        uint32 length, uint32 next);
    extern virtual function void print();
    local uint32 next_descriptor; //Pointer to DMA memory
endclass
```

The `extends` keyword from the first line of the class states that the `linked_list_descriptor` class inherits from the `descriptor` class. Now the `linked_list_descriptor` class has all the functionality of the original `descriptor` class, and adds the `next_descriptor` variable.

Note that in an extended class, you must manually call the base class constructor:

```
//in linked_list_descriptor.sv
function linked_list_descriptor::new
    (uint32 src, uint32 dest, uint32 length, uint32 next);
    super.new (src, dest, length, 0);
    next_descriptor_ = next;
endfunction
```

Inheritance for code interface

As we have seen, *inheritance for code interface* means using a base class, with virtual methods, to describe the class framework. This base class identifies some or all of the methods as *virtual*; classes extending from the base class can, and sometime must, implement those virtual methods.

With this technique, standard code interfaces for similar, but different, components can be used. This is very useful in creating a verification framework, because in a large verification environment you must keep track of a large number of components—for example, verification components [such as bus functional models (BFMs), generators, checkers, and monitors], and test components. By defining a code interface to which each type of component must conform, the ability to reason about the environment increases.

It should be noted that defining appropriate virtual base classes is not easy. Overly complicated or overly generic base classes tend to make the problem of verification more confusing instead of less. In Part III of this handbook we'll talk about the trade-offs.

As an example of *inheritance for code interface*, let's consider building a virtual base class for a BFM for a verification project. The team could decide that all verification components need certain *phases* (expressed as method calls), including `do_randomize()`, `out_of_reset()`, `start()`, and `final_report()`.

These methods ensure that a verification component is randomized, has time to program its part of the chip, starts up any threads needed to run, and has a way to print its status once the simulation is done. This can be done by creating a *virtual base class* from which all actual drivers inherit.

Packages are very useful for grouping related enumerations and constants. Be careful with the `import` clause in header files.

Data Duplication—A Necessary Evil

There is an inherent trade-off between minimizing the complexity of connections among verification components and duplicating the data passed among them. The looser the connection, the more likely that there is a duplication of data. This is talked about in detail in the next chapter, but we'll discuss it as a design bias here.

Let's look at an example of duplicated data. Consider a DMA¹ chip's view of data versus a checker's view. A processor sends to the DMA engine the source address, the destination address, and the length of the transfer. When the DMA completes, selected memory contents are stored in an object that contains a start address, an end address, a length, and a completion status for the transaction. The chip synchronization mechanism for the monitor is probably an interrupt. The class and the chip memory both contain the data, so we have a data duplication situation. In this design the data duplication is good, because checkers can use the more abstract fields of a class, instead of reading memory directly. Also, the code is more portable, because the concepts of source, destination, and length are abstracted away from the actual memory layout.

In the previous example, the data representation changed from chip memory to fields in a class. However, there is another place where data are duplicated. This is a more subtle, yet useful, design technique.

We think nothing of calling a function and passing it an integer. We know that the integer's value will be copied. But what if this integer were the result of some computations in the function making the call? Now the situation is not so clear. Why? Because this integer is not just a value. It represents the result of an algorithm that computed some data. The fancy term for what this integer represents is *derived data*. This is a normal situation, and it is used all the time when abstraction layers are crossed.

¹. Direct memory access.

Designing Well, Optimizing Only When Necessary

In fact, derived data are common in multilayer protocols. Many protocols have a physical layer, which deals in bits and words. Then they have a transport layer, which deals in packets. Some protocols even have a third layer, which deals in higher-layer transfers. As each layer hands off to the next, data are copied. The concept of derived data also exists when abstraction layers are used in verification.

For example, consider the physical interface for Ethernet Media Access Control (MAC). Suppose a verification component called MAC supports both Media Independent Interface (MII) and Reduced Media Independent Interface (RMII), and expresses this in an enumeration. These two interfaces are quite close, with RMII being a reduced set of the MII interface. In order to simplify the control connections between the MAC and the physical layer, the connection could be a single bit set to one for “full,” to distinguish it from “reduced,” where the bit would be zero. Although the information in a single bit is weaker and simpler than the passing of an enumeration, the bit is more appropriate to the lower-level abstraction.

Be aware of the same data being in two places at once. This generally happens across abstraction levels. Be very aware when derived data must be refreshed.

Designing Well, Optimizing Only When Necessary

.....

When hardware is designed and implemented, efficiency is critical to its ability to be built (meeting timing, power, and size constraints). However, in software the emphasis is often on minimizing mental complexity. This is natural, as increasingly complex control and data structures can be built relatively quickly with software.

In verification you must focus on clarity in your overall design and implementation.¹ This does not mean, however, that you can ignore efficiency.

¹. The authors assume, of course, that the verification of the chip is the primary concern.

Chapter 10: Designing with OOP

The speed of a functional simulation almost always depends on just a small percentage of the actual code. You should optimize the code for simulation speed only after you have finished profiling it thoroughly. Premature code optimization leads to confusion. While it might run a simulation a little faster, it will more often than not slow down the project by being harder to understand and use.

Get the design working and understandable first. Then figure out what needs to be optimized—and possibly made more complicated.

Using the Protocol, Only the Protocol

The verification test system must both apply stimuli to the chip and then check to make sure the chip generated the correct response. Consequently, the verification system is aware of *both ends* of a data path—an awareness the software does not have. In a simulation, creating both ends of a data path is necessary, but it can lead to sloppy code that may mask chip problems. Be aware of what information the protocol provides, and use only that information for checking.

For example, the authors helped test a USB device. The original verification tests preloaded the chip with the “correct” data. Then, when the data were needed, as specified by the protocol, they were sent by the chip to the checker. The checker confirmed the data and all was well. Upon closer inspection by the software team, a critical parameter was found to be missing: the length of the response. By design, this parameter was being dropped by the chip, but because the verification code already knew the “right” answer and length, the design error was missed. Had the verification code been cleanly separated into a *generator* (the requestor) and a *checker* (the interpreter) of the request, the error would have been caught.

Another example concerned a DMA checker. The DMA checker required the verification components to register all the memory accesses that the chip would make. At the end of the test, the DMA checker would make sure there were no unmatched writes. What the checker failed to test,

Verification Close to the Programming Model

however, was the result of overlapped memory writes. There was a bug in the cache coherency unit that was masked because the tests did not write to the same address twice.

Minimize the assumptions made across different parts of the verification system. Strive to use only the “knowledge” that the production software has through the feature or interface.

Verification Close to the Programming Model

Verification is both similar to and different from the production software that will run on the system. However, the closer the verification architecture is to the software, the better the chance you have of finding errors in the programming model, with respect to condition status registers (CSRs), hardware algorithms, and so on.

So while it is true that the software you use for verification is inherently a lot more detailed than the production software, you should still strive to keep your test algorithms as close to the production ones as possible. This design bias not only helps the software and hardware folks communicate better, but it can also help shake out register/memory/interrupt design and programming model.

For example, the authors had a project where the design of the DMA part of a chip was changed after the verification team had their code reviewed by the software team. Yes, this change was humbling. Specifically, the chip used a bit to restart a DMA channel. The original design had made the transition of zero to one be the enabling action. This meant that the software would first have to set this bit to zero (it was a one from a previous enable!), and then set it back to one. Once the software team was aware of this issue, they asked how hard it would be just to restart with a write of one, even if the previous value were a one. This was done, making the DMA enable more intuitive to the software team. Another change concerned the ability to write the DMA offset and index. The original design allowed only a reset of the index to zero, which made resending a DMA buffer clumsy. The software had to recopy all the buffer descriptors in memory to the zero offset address. The hardware was

Chapter 10: Designing with OOP

changed to allow the index to be written whenever the DMA unit was paused.

A final change was in the address map. Previously, the address space for the DMA was contiguous, which made sense to the hardware team. However, because the various I/O subsystems were features that end users paid for individually, the software was separated according to I/O subsystems. Because of this division, there were two different address spaces to manage: one for the I/O subsystem, and another for the DMA. We changed this design and moved the specific DMA channel's CSR address space to the I/O subsystem that it served. While this change seems (and probably is) simple, it shows that the hardware and software teams viewed the chip differently. Working more closely together improves the overall quality and helps reduce total project time. Furthermore, using a software bias allowed the two teams to look at each other's code without too much complaining.

Try to design your verification system to use the same algorithms and subarchitecture as the production software. That way, you can catch clumsy or conflicting programming models.

The Three Parts of Checking

The major thrust of verification is checking the operation of the chip. We send data in, or turn on features, and check that the chip produces the correct response. A fancy term for the transmission of data and the enabling of features is called *stimulus*. As a design bias, it's a good idea to separate the generation of the stimulus from the acting on and checking of the stimulus. We'll look at this further in subsequent chapters, but for now let's look at the *checker*.

Once data have been injected into the chip, the checker recovers the actual data output from the chip's I/O and confirms that everything is as expected. There are three parts to this process and, to promote adaptability, the implementations of each part should be kept separate. The success

Reservable resources

The majority of hardware has a bus to access configuration registers. As the verification system we write consists of many threads, there is a danger that two threads can start using the bus at the same time. The hardware bus is considered a *reservable resource*, because code must first request access to the resource. The trick is to make the reservation as simple as possible for the code that uses the resource.

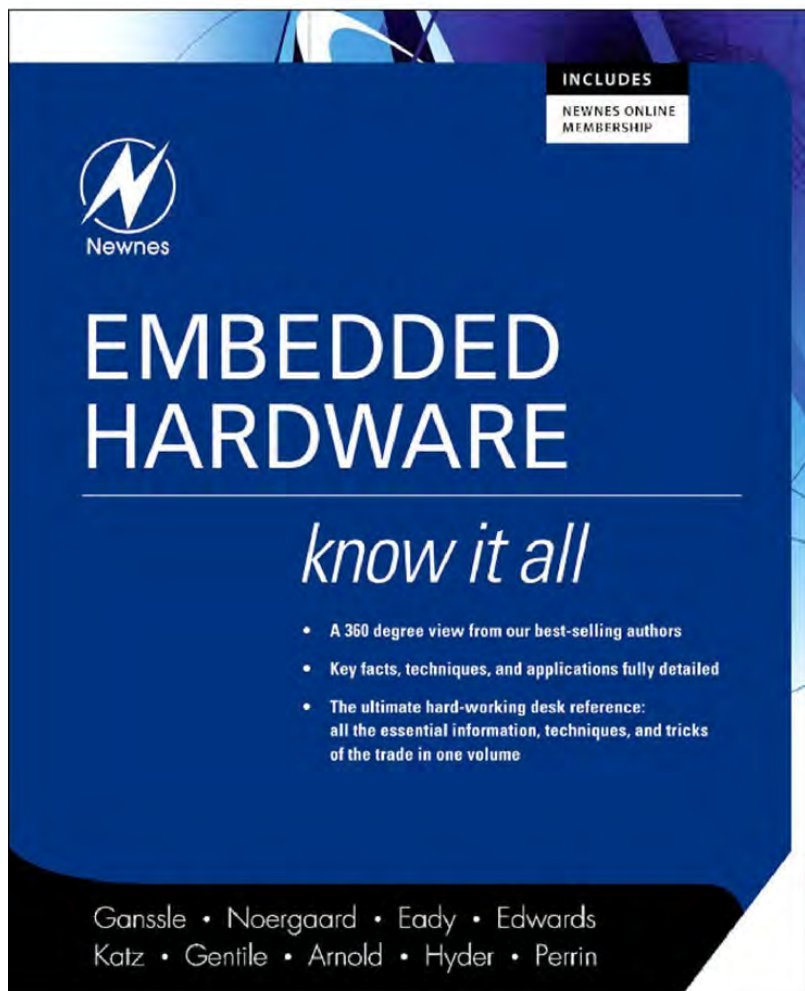
The simplest solution is to hide the reservation inside the implementation of the class. This is appropriate in most cases. A simple mutual exclusion (mutex) algorithm can be used for this purpose. A mutex only allows one thread at a time into a section of code. The latter parts of this handbook show examples of using a mutex in a register-access BFM.

Sometimes the hardware can process several requests at the same time. This is probably an implementation detail when used in a full chip test, but it is probably something you need to expose at the unit level test. In this case there are two classes. One class exposes a *key*, such as an integer tag or an instance in the interface. The other class uses this lower-level class and hides the key from the rest of the system.

The concept of reservable resources can also exist solely in the verification system itself. You might, for example, have a DMA descriptor queue and need to allocate and release descriptors. Of course, the hardware actually implements the queue, but the management of the queue is a verification concept.

Reservable resources may be an implementation detail, and thus use a mutex internally—or they may be an external property, in which case a “key” must be used. This key can be anything from an integer to an object, depending on how safe the management must be.

EXHIBIT D



96800DOC00033153

Petitioner Intel Corp., Ex. 1012
IPR2023-00783

Embedded Hardware

96800DOC00033154

Petitioner Intel Corp., Ex. 1012

IPR2023-00783

This book belongs to Christine Moua
(cmoua@perkinscoie.com)

Copyright Elsevier 2022

96800DOC00033155
Petitioner Intel Corp., Ex. 1012
IPR2023-00783

Newnes Know It All Series

PIC Microcontrollers: Know It All

Lucio Di Jasio, Tim Wilmshurst, Dogan Ibrahim, John Morton,
Martin Bates, Jack Smith, D.W. Smith, and Chuck Hellebuyck
ISBN: 978-0-7506-8615-0

Embedded Software: Know It All

Jean Labrosse, Jack Ganssle, Tammy Noergaard, Robert Oshana, Colin Walls, Keith Curtis,
Jason Andrews, David J. Katz, Rick Gentile, Kamal Hyder, and Bob Perrin
ISBN: 978-0-7506-8583-2

Embedded Hardware: Know It All

Jack Ganssle, Tammy Noergaard, Fred Eady, Creed Huddleston, Lewin Edwards,
David J. Katz, Rick Gentile, Ken Arnold, Kamal Hyder, and Bob Perrin
ISBN: 978-0-7506-8584-9

Wireless Networking: Know It All

Praphul Chandra, Daniel M. Dobkin, Alan Bensky, Ron Olexa,
David A. Lide, and Farid Dowla
ISBN: 978-0-7506-8582-5

RF & Wireless Technologies: Know It All

Bruce Fette, Roberto Aiello, Praphul Chandra, Daniel Dobkin,
Alan Bensky, Douglas Miron, David A. Lide, Farid Dowla, and Ron Olexa
ISBN: 978-0-7506-8581-8

For more information on these and other Newnes titles visit: www.newnespress.com

Embedded Hardware

Jack Ganssle
Tammy Noergaard
Fred Eady
Lewin Edwards
David J. Katz
Rick Gentile
Ken Arnold
Kamal Hyder
Bob Perrin
Creed Huddleston



AMSTERDAM • BOSTON • HEIDELBERG • LONDON
NEW YORK • OXFORD • PARIS • SAN DIEGO
SAN FRANCISCO • SINGAPORE • SYDNEY • TOKYO

Newnes is an imprint of Elsevier



Cover image by iStockphoto

Newnes is an imprint of Elsevier

30 Corporate Drive, Suite 400, Burlington, MA 01803, USA

Linacre House, Jordan Hill, Oxford OX2 8DP, UK

Copyright © 2008 by Elsevier Inc. All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the publisher.

Permissions may be sought directly from Elsevier's Science & Technology Rights Department in Oxford, UK: phone: (+44) 1865 843830, fax: (+44) 1865 853333, E-mail: permissions@elsevier.com. You may also complete your request online via the Elsevier homepage (<http://elsevier.com>), by selecting "Support & Contact" then "Copyright and Permission" and then "Obtaining Permissions."



Recognizing the importance of preserving what has been written, Elsevier prints its books on acid-free paper whenever possible.

Library of Congress Cataloging-in-Publication Data

Ganssle, Jack G.

Embedded hardware / Jack Ganssle ... [et al.].

p. cm.

Includes index.

ISBN 978-0-7506-8584-9 (alk. paper)

1. Embedded computer systems. I. Title.

TK7895.E42G37 2007

004.16—dc22

2007027559

British Library Cataloguing-in-Publication Data

A catalogue record for this book is available from the British Library.

For information on all Newnes publications
visit our Web site at www.books.elsevier.com

07 08 09 10 10 9 8 7 6 5 4 3 2 1

Typeset by Charon Tec Ltd (A Macmillan Company), Chennai, India

www.charontec.com

Printed in the United States of America

Working together to grow
libraries in developing countries

www.elsevier.com | www.bookaid.org | www.sabre.org

ELSEVIER

BOOK AID
International

Sabre Foundation

Contents

About the Authors.....xiii

Chapter 1: Embedded Hardware Basics 1

 1.1 Lesson One on Hardware: Reading Schematics..... 1

 1.2 The Embedded Board and the von Neumann Model 5

 1.3 Powering the Hardware 9

 1.3.1 A Quick Comment on Analog Vs. Digital Signals 10

 1.4 Basic Electronics 12

 1.4.1 DC Circuits 12

 1.4.2 AC Circuits 21

 1.4.3 Active Devices 28

 1.5 Putting It Together: A Power Supply 32

 1.5.1 The Scope 35

 1.5.2 Controls..... 35

 1.5.3 Probes..... 38

 Endnotes 41

Chapter 2: Logic Circuits..... 43

 2.1 Coding 43

 2.1.1 BCD 46

 2.2 Combinatorial Logic..... 47

 2.2.1 NOT Gate 47

 2.2.2 AND and NAND Gates..... 48

 2.2.3 OR and NOR Gates..... 49

 2.2.4 XOR 50

 2.2.5 Circuits 50

 2.2.6 Tristate Devices..... 53

 2.3 Sequential Logic..... 53

 2.3.1 Logic Wrap-Up 57

 2.4 Putting It All Together: The Integrated Circuit 58

 Endnotes 61

www.newnespress.com

vi Contents

Chapter 3: Embedded Processors.....	63
3.1 Introduction	63
3.2 ISA Architecture Models.....	65
3.2.1 Operations	65
3.2.2 Operands	68
3.2.3 Storage	69
3.2.4 Addressing Modes.....	71
3.2.5 Interrupts and Exception Handling	72
3.2.6 Application-Specific ISA Models	72
3.2.7 General-Purpose ISA Models	74
3.2.8 Instruction-Level Parallelism ISA Models.....	76
3.3 Internal Processor Design.....	78
3.3.1 Central Processing Unit (CPU).....	82
3.3.2 On-Chip Memory	99
3.3.3 Processor Input/Output (I/O).....	113
3.3.4 Processor Buses.....	130
3.4 Processor Performance	131
3.4.1 Benchmarks.....	133
Endnotes	133
Chapter 4: Embedded Board Buses and I/O	137
4.1 Board I/O.....	137
4.2 Managing Data: Serial vs. Parallel I/O.....	140
4.2.1 Serial I/O Example 1: Networking and Communications: RS-232	144
4.2.2 Example: Motorola/Freescale MPC823 FADS Board RS-232 System Model	146
4.2.3 Serial I/O Example 2: Networking and Communications: IEEE 802.11 Wireless LAN	148
4.2.4 Parallel I/O	153
4.2.5 Parallel I/O Example 3: “Parallel” Output and Graphics I/O	153
4.2.6 Parallel and Serial I/O Example 4: Networking and Communications—Ethernet.....	156
4.2.7 Example 1: Motorola/Freescale MPC823 FADS Board Ethernet System Model	158
4.2.8 Example 2: Net Silicon ARM7 (6127001) Development Board Ethernet System Model	160
4.2.9 Example 3: Adastra Neptune x86 Board Ethernet System Model	161
4.3 Interfacing the I/O Components	161
4.3.1 Interfacing the I/O Device with the Embedded Board.....	162
4.3.2 Interfacing an I/O Controller and the Master CPU	164

www.newnespress.com

This book belongs to Christine Moua (cmoua@perkinscoie.com)

Copyright Elsevier 2022

96800DOC00033160

Petitioner Intel Corp., Ex. 1012
IPR2023-00783

4.4 I/O and Performance	165
4.5 Board Buses.....	166
4.6 Bus Arbitration and Timing.....	168
4.6.1 Nonexpandable Bus: I ² C Bus Example	174
4.6.2 PCI (Peripheral Component Interconnect) Bus Example: Expandable	175
4.7 Integrating the Bus with Other Board Components	179
4.8 Bus Performance	180
Chapter 5: Memory Systems.....	183
5.1 Introduction	183
5.2 Memory Spaces	183
5.2.1 L1 Instruction Memory	186
5.2.2 Using L1 Instruction Memory for Data Placement.....	186
5.2.3 L1 Data Memory	187
5.3 Cache Overview	187
5.3.1 What Is Cache?	188
5.3.2 Direct-Mapped Cache	190
5.3.3 Fully Associative Cache.....	190
5.3.4 N-Way Set-Associative Cache	191
5.3.5 More Cache Details.....	191
5.3.6 Write-Through and Write-Back Data Cache.....	193
5.4 External Memory.....	195
5.4.1 Synchronous Memory	195
5.4.2 Asynchronous Memory.....	203
5.4.3 Nonvolatile Memories	206
5.5 Direct Memory Access.....	214
5.5.1 DMA Controller Overview	215
5.5.2 More on the DMA Controller	216
5.5.3 Programming the DMA Controller.....	218
5.5.4 DMA Classifications	228
5.5.5 Register-Based DMA	228
5.5.6 Descriptor-Based DMA	231
5.5.7 Advanced DMA Features.....	234
Endnotes	236
Chapter 6: Timing Analysis in Embedded Systems.....	239
6.1 Introduction	239
6.2 Timing Diagram Notation Conventions	239
6.2.1 Rise and Fall Times	241

viii Contents

6.2.2 Propagation Delays	241
6.2.3 Setup and Hold Time.....	241
6.2.4 Tri-State Bus Interfacing.....	243
6.2.5 Pulse Width and Clock Frequency	244
6.3 Fan-Out and Loading Analysis: DC and AC.....	244
6.3.1 Calculating Wiring Capacitance.....	247
6.3.2 Fan-Out When CMOS Drives LSTTL.....	249
6.3.3 Transmission-Line Effects	251
6.3.4 Ground Bounce	253
6.4 Logic Family IC Characteristics and Interfacing	255
6.4.1 Interfacing TTL Compatible Signals to 5 V CMOS	258
6.5 Design Example: Noise Margin Analysis Spreadsheet.....	261
6.6 Worst-Case Timing Analysis Example.....	270
Endnotes	272
 Chapter 7: Choosing a Microcontroller and Other Design Decisions	 273
7.1 Introduction	273
7.2 Choosing the Right Core	276
7.3 Building Custom Peripherals with FPGAs.....	281
7.4 Whose Development Hardware to Use—Chicken or Egg?.....	282
7.5 Recommended Laboratory Equipment.....	285
7.6 Development Toolchains	286
7.7 Free Embedded Operating Systems	289
7.8 GNU and You: How Using “Free” Software Affects Your Product.....	295
 Chapter 8: The Essence of Microcontroller Networking: RS-232.....	 301
8.1 Introduction	301
8.2 Some History.....	303
8.3 RS-232 Standard Operating Procedure	305
8.4 RS-232 Voltage Conversion Considerations	308
8.5 Implementing RS-232 with a Microcontroller	310
8.5.1 Basic RS-232 Hardware.....	310
8.5.2 Building a Simple Microcontroller RS-232 Transceiver	313
8.6 Writing RS-232 Microcontroller Routines in BASIC	333
8.7 Building Some RS-232 Communications Hardware.....	339
8.7.1 A Few More BASIC RS-232 Instructions.....	339
8.8 I ² C: The Other Serial Protocol	342
8.8.1 Why Use I ² C?.....	343
8.8.2 The I ² C Bus.....	344
8.8.3 I ² C ACKS and NAKS	347

www.newnespress.com

This book belongs to Christine Moua (cmoua@perkinscoie.com)

Copyright Elsevier 2022

96800DOC00033162

Petitioner Intel Corp., Ex. 1012
IPR2023-00783

8.8.4 More on Arbitration and Clock Synchronization.....	347
8.8.5 I ² C Addressing	351
8.8.6 Some I ² C Firmware.....	352
8.8.7 The AVR Master I ² C Code.....	352
8.8.8 The AVR I ² C Master-Receiver Mode Code.....	358
8.8.9 The PIC I ² C Slave-Transmitter Mode Code	359
8.8.10 The AVR-to-PIC I ² C Communications Ball	365
8.9 Communication Options.....	378
8.9.1 The Serial Peripheral Interface Port.....	378
8.9.2 The Controller Area Network	380
8.9.3 Acceptance Filters	386
Endnote.....	387
Chapter 9: Interfacing to Sensors and Actuators	389
9.1 Introduction	389
9.2 Digital Interfacing	389
9.2.1 Mixing 3.3 and 5 V Devices	389
9.2.2 Protecting Digital Inputs	392
9.2.3 Expanding Digital Inputs	398
9.2.4 Expanding Digital Outputs.....	402
9.3 High-Current Outputs.....	404
9.3.1 BJT-Based Drivers	405
9.3.2 MOSFETs	409
9.3.3 Electromechanical Relays	411
9.3.4 Solid-State Relays	417
9.4 CPLDs and FPGAs.....	418
9.5 Analog Interfacing: An Overview	420
9.5.1 ADCs.....	420
9.5.2 Project 1: Characterizing an Analog Channel.....	421
9.6 Conclusion.....	434
Endnote.....	435
Chapter 10: Other Useful Hardware Design Tips and Techniques	437
10.1 Introduction	437
10.2 Diagnostics	437
10.3 Connecting Tools.....	438
10.4 Other Thoughts.....	439
10.5 Construction Methods	440
10.5.1 Power and Ground Planes.....	441
10.5.2 Ground Problems	441

x Contents

10.6 Electromagnetic Compatibility	442
10.7 Electrostatic Discharge Effects	442
10.7.1 Fault Tolerance	443
10.8 Hardware Development Tools	444
10.8.1 Instrumentation Issues	445
10.9 Software Development Tools	445
10.10 Other Specialized Design Considerations	446
10.10.1 Thermal Analysis and Design	446
10.10.2 Battery-Powered System Design Considerations	447
10.11 Processor Performance Metrics	448
10.11.1 IPS	448
10.11.2 OPS	448
10.11.3 Benchmarks	449
Appendix A: Schematic Symbols	451
Appendix B: Acronyms and Abbreviations	459
Appendix C: PC Board Design Issues	469
C.1 Introduction	469
C.2 Resistance of Conductors	470
C.3 Voltage Drop in Signal Leads—"Kelvin" Feedback	471
C.4 Signal Return Currents	472
C.5 Grounding in Mixed Analog/Digital Systems	474
C.6 Ground and Power Planes	475
C.7 Double-Sided versus Multilayer Printed Circuit Boards	477
C.8 Multicard Mixed-Signal Systems	478
C.9 Separating Analog and Digital Grounds	479
C.10 Grounding and Decoupling Mixed-Signal ICs with Low Digital Currents	480
C.11 Treat the ADC Digital Outputs with Care	481
C.12 Sampling Clock Considerations	483
C.13 The Origins of the Confusion About Mixed-Signal Grounding: Applying Single-Card Grounding Concepts to Multicard Systems	485
C.14 Summary: Grounding Mixed-Signal Devices with Low Digital Currents in a Multicard System	486
C.15 Summary: Grounding Mixed-Signal Devices with High Digital Currents in a Multicard System	487
C.16 Grounding DSPs with Internal Phase-Locked Loops	487
C.17 Grounding Summary	488
C.18 Some General PC Board Layout Guidelines for Mixed-Signal Systems	489

www.newnespress.com

C.19 Skin Effect	491
C.20 Transmission Lines	493
C.21 Be Careful with Ground Plane Breaks.....	494
C.22 Ground Isolation Techniques	495
C.23 Static PCB Effects	497
C.24 Sample MINIDIP and SOIC Op Amp PCB Guard Layouts.....	500
C.25 Dynamic PCB Effects.....	502
C.26 Stray Capacitance	503
C.27 Capacitive Noise and Faraday Shields.....	504
C.28 The Floating Shield Problem.....	506
C.29 Buffering ADCs Against Logic Noise.....	506
Endnotes	509
Acknowledgments	509
Index	511

This page intentionally left blank

About the Authors

Ken Arnold (Chapters 6 and 10) is the author of *Embedded Controller Hardware Design*. He is the Embedded Computer Engineering Program Coordinator and an instructor at UCSD Extension, as well as founding director of the On-Line University of California, where he manages, develops and teaches courses in engineering and embedded systems design. Ken has been developing commercial embedded systems and teaching others how for more than two decades. As the champion of the embedded program at UCSD, he led the inception and growth of the program as well as introducing the world's first on-line embedded course well over a decade ago. Ken was also the founder and CEO of HiTech Equipment Corp., CTO of Wireless Innovation, and engineering chief at General Dynamics.

Fred Eady (Chapter 8) is the author of *Networking and Internetworking with Microcontrollers*. As an engineering consultant, he has implemented communications networks for the space program and designed hardware and firmware for the medical, retail and public utility industries. He currently writes a monthly embedded design column for a popular electronics enthusiast magazine. Fred also composes monthly articles for a popular robotics magazine. Fred has been dabbling in electronics for over 30 years. His embedded design expertise spans the spectrum and includes Intel's 8748 and 8051 microcontrollers, the entire Microchip PIC microcontroller family and the Atmel AVR microcontrollers. Fred recently retired from his consulting work and is focused on writing magazine columns and embedded design books.

Lewin Edwards (Chapter 7) is the author of *Embedded System Design on a Shoestring*. He hails from Adelaide, Australia. His career began with five years of security and encryption software at PC-Plus Systems. The next five years were spent developing networkable multimedia appliances at Digi-Frame in Port Chester, NY. Since 2004 he has been developing security and fire safety devices at a Fortune 100 company in New York. He has written numerous technical articles and three embedded systems books, with a fourth due in early 2008.

Jack Ganssle (Chapters 1, 2, and 10) is the author of *The Firmware Handbook*. He has written over 500 articles and six books about embedded systems, as well as a book about his sailing fiascos. He started developing embedded systems in the early 70s using the 8008. He's started and sold three electronics companies, including one of the bigger embedded tool businesses. He's developed or managed over 100 embedded products, from deep-sea navigation gear to

www.newnespress.com

the White House security system... and one instrument that analyzed cow poop! He's currently a member of NASA's Super Problem Resolution Team, a group of outside experts formed to advise NASA in the wake of Columbia's demise, and serves on the boards of several high-tech companies. Jack now gives seminars to companies world-wide about better ways to develop embedded systems.

Rick Gentile (Chapter 5) is the author of *Embedded Media Processing*. Rick joined ADI in 2000 as a Senior DSP Applications Engineer, and he currently leads the Processor Applications Group, which is responsible for Blackfin, SHARC and TigerSHARC processors. Prior to joining ADI, Rick was a Member of the Technical Staff at MIT Lincoln Laboratory, where he designed several signal processors used in a wide range of radar sensors. He has authored dozens of articles and presented at multiple technical conferences. He received a B.S. in 1987 from the University of Massachusetts at Amherst and an M.S. in 1994 from Northeastern University, both in Electrical and Computer Engineering.

Creed Huddleston (Chapter 8) is the author of *Intelligent Sensor Design Using the Microchip dsPIC*. With over twenty years of experience designing real-time embedded systems, he is President and founder of Real-Time by Design, LLC, a certified Microchip Design Partner based in Raleigh, NC that specializes in the creation of hard real-time intelligent sensing systems. In addition to his duties with Real-Time by Design, Creed also serves on the Advisory Board of Quickfilter Technologies Inc., a Texas-based company producing mixed-signal integrated circuits that provide high-speed analog signal conditioning and digital signal processing in a single package. A graduate of Rice University in Houston, TX with a BSEE degree, Creed performed extensive graduate work in digital signal processing at the University of Texas at Arlington before heading east to start Omnisys. To her great credit and his great fortune, Creed and his wife Lisa have been married for 23 years and have three wonderful children: Kate, Beth, and Dan.

Kamal Hyder (Chapter 9) is the author of *Embedded Systems Design Using the Rabbit 3000 Microprocessor*. He started his career with an embedded microcontroller manufacturer. He then wrote CPU microcode for Tandem Computers for a number of years, and was a Product Manager at Cisco Systems, working on next-generation switching platforms. He is currently with Brocade Communications as Senior Group Product Manager. Kamal's BS is in EE/CS from the University of Massachusetts, Amherst, and he has an MBA in finance/marketing from Santa Clara University.

David Katz (Chapter 5) is the author of *Embedded Media Processing*. He has over 15 years of experience in circuit and system design. Currently, he is the Blackfin Applications Manager at Analog Devices, Inc., where he focuses on specifying new convergent processors. He has published over 100 embedded processing articles domestically and internationally, and he has presented several conference papers in the field. Previously, he worked at Motorola, Inc., as a

senior design engineer in cable modem and automation groups. David holds both a B.S. and M. Eng. in Electrical Engineering from Cornell University.

Walt Kester (Appendix C) is the editor of *Data Conversion Handbook*. He is a corporate staff applications engineer at Analog Devices. For more than 35 years at Analog Devices, he has designed, developed, and given applications support for high-speed ADCs, DACs, SHAs, op amps, and analog multiplexers. Besides writing many papers and articles, he prepared and edited eleven major applications books which form the basis for the Analog Devices world-wide technical seminar series including the topics of op amps, data conversion, power management, sensor signal conditioning, mixed-signal, and practical analog design techniques. Walt has a BSEE from NC State University and MSEE from Duke University.

Tammy Noergaard (Chapters 1, 2, 3, 4, Appendices B and C) is the author of *Embedded Systems Architecture*. Since beginning her embedded systems career in 1995, she has had wide experience in product development, system design and integration, operations, sales, marketing, and training. Noergaard worked for Sony as a lead software engineer developing and testing embedded software for analog TVs. At Wind River she was the liaison engineer between developmental engineers and customers to provide design expertise, systems configuration, systems integration, and training for Wind River embedded software (OS, Java, device drivers, etc.) and all associated hardware for a variety of embedded systems in the Consumer Electronic market. Most recently she was a Field Engineering Specialist and Consultant with Esmertec North America, providing project management, system design, system integration, system configuration, support and expertise for various embedded Java systems using Jbed in everything from control systems to medical devices to digital TVs. Noergaard has lectured to engineering classes at the University of California at Berkeley and Stanford, the Embedded Internet Conference, and the Java User's Group in San Jose, among others.

Bob Perrin (Chapter 9) is the author of *Embedded Systems Design Using the Rabbit 3000 Microprocessor*. He got his start in electronics at the age of nine when his mother gave him a "150-in-one Projects" kit from Radio Shack for Christmas. He grew up programming a Commodore PET. In 1990, Bob graduated with a BSEE from Washington State University. Since then Bob has been working as an engineer designing digital and analog electronics. He has published about twenty technical articles, most with *Circuit Cellar*.

USB/Firewire

These drives are nothing more than Parallel ATA disk drives with a USB 2.0 high-speed or Firewire (IEEE 1394) front end, usually used to facilitate access to a PC. However, in the embedded world, where USB and Firewire are gaining traction, these interfaces provide a handy way to add storage to a system.

USB pen drives, also called keychain drives, are not related to these ATA drives. Instead, they are Flash memory devices with a USB front end, similar in all respects to the NAND memory devices described previously.

5.4.3.4 Emerging Nonvolatile Memory Technologies

There are some exciting new technologies on the horizon that will probably be very important in high-performance embedded applications. Two of particular interest are FRAM and MRAM. FRAM, or Ferroelectric RAM, uses electric field orientation to store charge. This gives it almost infinite write capability. By comparison, conventional EEPROMs can only be written on the order of 10,000 times. MRAM, or Magnetoresistive RAM, uses electron spin to store information. It represents the best of several memory domains: it's nonvolatile, it has bit densities rivaling DRAM, and it operates at SRAM-like speeds.

5.5 Direct Memory Access

The processor core is capable of doing multiple operations, including calculations, data fetches, data stores, and pointer increments/decrements, in a single cycle. In addition, the core can orchestrate data transfer between internal and external memory spaces by moving data into and out of the register file.

All this sounds great, but in reality you can only achieve optimum performance in your application if data can move around without constantly bothering the core to perform the transfers.

This is where a direct memory access (DMA) controller comes into play. Processors need DMA capability to relieve the core from these transfers between internal/external memory and peripherals or between memory spaces. There are two main types of DMA controllers. "Cycle-stealing" DMA uses spare (idle) core cycles to perform data transfers. This is not a workable solution for systems with heavy processing loads like multimedia flows. Instead, it is much more efficient to employ a DMA controller that operates independently from the core.

Why is this so important? Well, imagine if a processor's video port has a FIFO that needs to be read every time a data sample is available. In this case, the core has to be interrupted tens of millions of times each second. As though that's not disruptive enough, the core has to perform an equal amount of writes to some destination in memory. For every core processing cycle spent on this task, a corresponding cycle would be lost in the processing loop.

We know from experience that PC-based software designers transitioning to the embedded world are hesitant to rely on a DMA controller for moving data around in an application. This usually stems from their impression that the complexity of the programming model increases exponentially when DMA is factored in. Yes, it is true that a DMA controller adds another dimension to your solution. We will, in fact, explore some intricacies that DMA introduces—such as contention for shared resources and new challenges in maintaining coherency between data buffers. Our goal, however, is to put your mind at ease, to show you how DMA is truly your friend. In this chapter, we'll focus on the DMA controller itself.

5.5.1 DMA Controller Overview

Because you'll typically configure a DMA controller during code initialization, the core should only need to respond to interrupts after data set transfers are complete. You can program the DMA controller to move data in parallel with the core while the core is doing its basic processing tasks—the jobs on which it's supposed to be focused! In an optimized application, the core would never have to move any data but rather only access it in L1 memory. The core wouldn't need to wait for data to arrive, because the DMA engine would have already made it available by the time the core was ready to access it. Figure 5.16 shows a typical interaction between the processor and the DMA controller. The steps allocated to the processor involve setting up the transfer, enabling interrupts, and running code when an interrupt is generated. The dashed lines/arrows between memory and the peripheral indicate operations the DMA controller makes to move data independent of the processor. Finally, the interrupt input back to the processor can be used to signal that data is ready for processing.

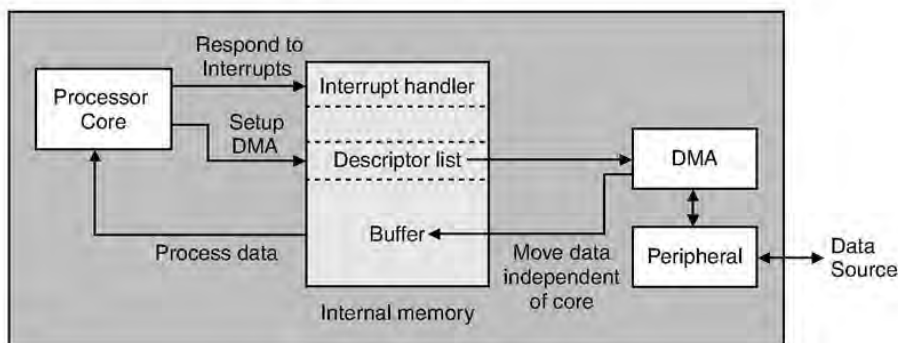


Figure 5.16: DMA controller.

In addition to moving to and from peripherals, data also needs to move from one memory space to another. For example, source video might flow from a video port straight to L3 memory, because the working buffer size is too large to fit into internal memory. We don't want to make the processor fetch pixels from external memory every time we need to perform a calculation,

so a memory-to-memory DMA (MemDMA for short) can bring pixels into L1 or L2 memory for more efficient access times. Figure 5.17 shows some typical DMA data flows.

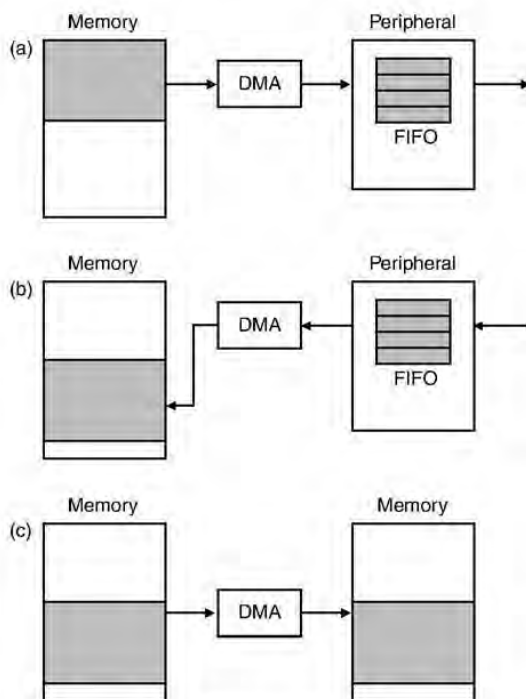


Figure 5.17: Typical DMA flows.

So far we've focused on data movement, but a DMA transfer doesn't always have to involve data. We can use code overlays to improve performance, configuring the DMA controller to move code into L1 instruction memory before execution. The code is usually staged in larger external memory.

In this chapter, we will use the Blackfin processor's DMA controller as a model to illustrate the basic concepts of direct memory access and how it can boost system performance. We will also offer some helpful ways to manage the DMA controller and review examples of "two-dimensional" transfers that can save valuable data passes by markedly reducing the time an application spends traversing a data buffer.

5.5.2 More on the DMA Controller

A DMA controller is a unique peripheral devoted to moving data around a system. Think of it as a controller that connects internal and external memories with each DMA-capable peripheral via

a set of dedicated buses. It is a peripheral in the sense that the processor programs it to perform transfers. It is unique in that it interfaces to both memory and selected peripherals. Notably, only peripherals where data flow is significant (kbytes per second or greater) need to be DMA-capable. Good examples of these are video, audio, and network interfaces. Lower-bandwidth peripherals can also be equipped with DMA capability, but it's less an imposition on the core to step in and assist with data transfer on these interfaces.

In general, DMA controllers will include an address bus, a data bus, and control registers. An efficient DMA controller will possess the ability to request access to any resource it needs, without having the processor itself get involved. It must have the capability to generate interrupts. Finally, it has to be able to calculate addresses within the controller.

A processor might contain multiple DMA controllers. Each controller has multiple DMA channels, as well as multiple buses that link directly to the memory banks and peripherals, as shown in Figure 5.18. There are two types of DMA controllers in the Blackfin processor. The first category, usually referred to as a System DMA controller, allows access to any resource (peripherals and memory). Cycle counts for this type of controller are measured in *system clocks* (SCLKs) at frequencies up to 133 MHz. The second type, an Internal Memory DMA (IMDMA) controller, is dedicated to accesses between internal memory locations. Because the accesses are internal (L1 to L1, L1 to L2, or L2 to L2), cycle counts are measured in *core clocks* (CCLKs), which can exceed 600 MHz rates.

Figure 5.18 also shows the Blackfin DMA bus structure, where the DMA External Bus (DEB) connects the DMA controller to external memory, the DMA Core Bus (DCB) connects the

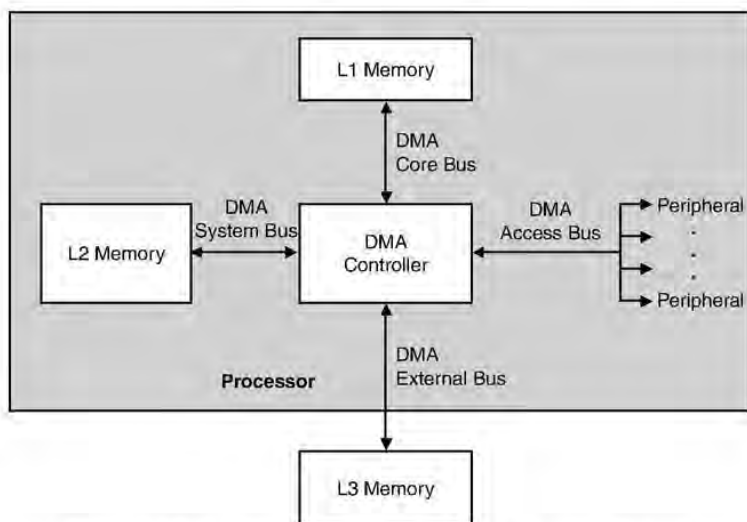


Figure 5.18: System and internal memory DMA architecture.

controller to internal memory, and the DMA Access Bus (DAB) connects to the peripherals. An additional DMA bus set is also available when L2 memory is present, to move data within the processor's internal memory spaces.

Each DMA channel on a Blackfin DMA controller has a programmable priority associated with it. If more than one channel requests the DMA bus in a single cycle, the highest-priority channel wins access. For memory DMA channels, a "round robin" access capability exists. That is, one memory channel can access the bus for a programmable number of cycles before turning the bus over to the next MemDMA channel, which also gets the same number of cycles on the bus. When more than one DMA controller is present on a processor, the channels from one controller can run at the same time as channels on the other controller. This is possible, for example, when a memory-to-memory transfer takes place from L3 to L2 memory while the second controller feeds a peripheral from L1 memory. If both DMA controllers try to access the same resource (L3 memory, for example), arbitration must take place. In this case, one of the controllers can be programmed to a higher priority than the other.

Each DMA controller has a set of FIFOs that act as a buffer between the DMA subsystem and peripherals or memory. For MemDMA, a FIFO exists on both the source and destination sides of the transfer. The FIFO improves performance by providing a place to hold data while busy resources are preventing a transfer from completing.

5.5.3 Programming the DMA Controller

Let's take a look at the options we have in specifying DMA activity. We will start with the simplest model and build up to more flexible models that, in turn, increase in setup complexity.

For any type of DMA transfer, we always need to specify starting source and destination addresses for data. In the case of a peripheral DMA, the peripheral's FIFO serves as either the source or the destination. When the peripheral serves as the source, a memory location (internal or external) serves as the destination address. When the peripheral serves as the destination, a memory location (internal or external) serves as the source address.

In the simplest MemDMA case, we need to tell the DMA controller the source address, the destination address and the number of words to transfer. With a peripheral DMA, we specify either the source or the destination, depending on the direction of the transfer. The word size of each transfer can be 8, 16, or 32 bits. This type of transaction represents a simple one-dimensional (1D) transfer with a unity "stride." As part of this transfer, the DMA controller keeps track of the source and destination addresses as they increment. With a unity stride, as in Figure 5.19a, the address increments by 1 byte for 8-bit transfers, 2 bytes for 16-bit transfers, and 4 bytes for 32-bit transfers.

www.newnespress.com

We can add more flexibility to a one-dimensional DMA simply by changing the stride, as in Figure 5.19b. For example, with nonunity strides, we can skip addresses in multiples of the transfer sizes. That is, specifying a 32-bit transfer and striding by four samples results in an address increment of 16 bytes (four 32-bit words) after each transfer.

Couching this discussion in Blackfin DMA controller lingo, we have now described the operations of the XCOUNT and XMODIFY registers. XCOUNT is the number of *transfers* that need to be made. Note that this is not necessarily the same as the number of bytes to transfer. XMODIFY is the number of bytes to increment the address pointer after the DMA controller moves the first data element. Regardless of the transfer word size, XMODIFY is always expressed in bytes. XMODIFY can also take on the value of 0, which has its own advantage, as we'll see later in this chapter.

Whereas the 1D DMA capability is widely used, the two-dimensional (2D) capability is even more useful, especially in video applications. The 2D feature is a direct extension to what we discussed for 1D DMA. In addition to an XCOUNT and XMODIFY value, we also program corresponding YCOUNT and YMODIFY values. It is easiest to think of the 2D DMA as a nested loop, where the inner loop is specified by XCOUNT and XMODIFY, and the outer loop is

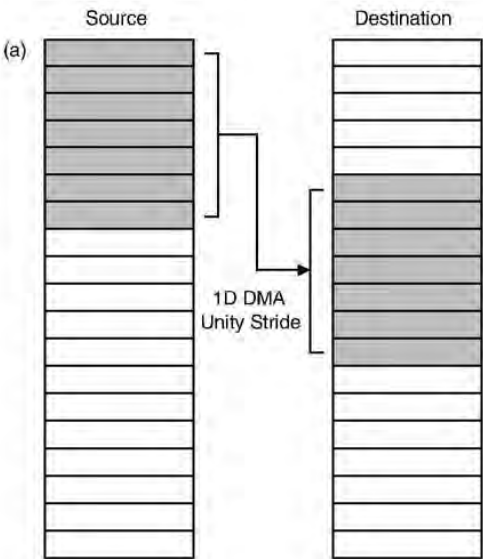


Figure 5.19: 1D DMA examples.
(a) 1D DMA with unity stride.

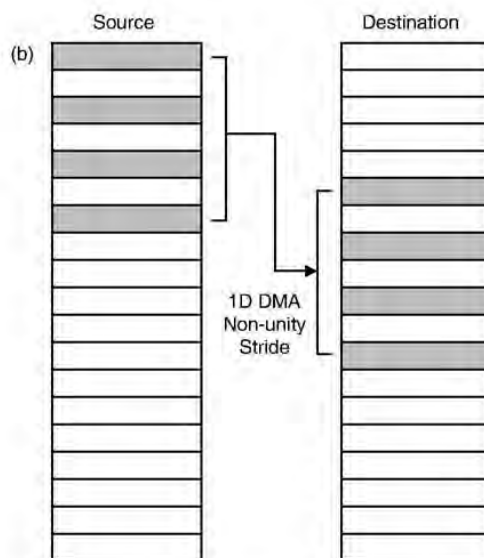


Figure 5.19: (Continued)
(b) 1D DMA with nonunity stride.

specified by YCOUNT and YMODIFY. A 1D DMA can then be viewed simply as an “inner loop” of the 2D transfer of the form:

```
for y = 1 to YCOUNT STEP YMODIFY /* 2D with outer loop */
for x = 1 to XCOUNT STEP XMODIFY /* 1D inner loop */
{
    /*Loop goes here */
}
```

While XMODIFY determines the stride value the DMA controller takes every time XCOUNT decrements, YMODIFY determines the stride taken whenever YCOUNT decrements. As is the case with XCOUNT and XMODIFY, YCOUNT is specified in terms of the number of transfers, while YMODIFY is specified as a number of bytes. Notably, YMODIFY can be negative, which allows the DMA controller to wrap back around to the beginning of the buffer. We’ll explore this feature shortly.

For a peripheral DMA, the “memory side” of the transfer can be either 1D or 2D. On the peripheral side, though, it is always a 1D transfer. The only constraint is that the total number of bytes transferred on each side (source and destination) of the DMA must be the same. For example, if we were feeding a peripheral from three 10-byte buffers, the peripheral would have to be set to transfer 30 bytes using any possible combination of supported transfer width and transfer count values available.

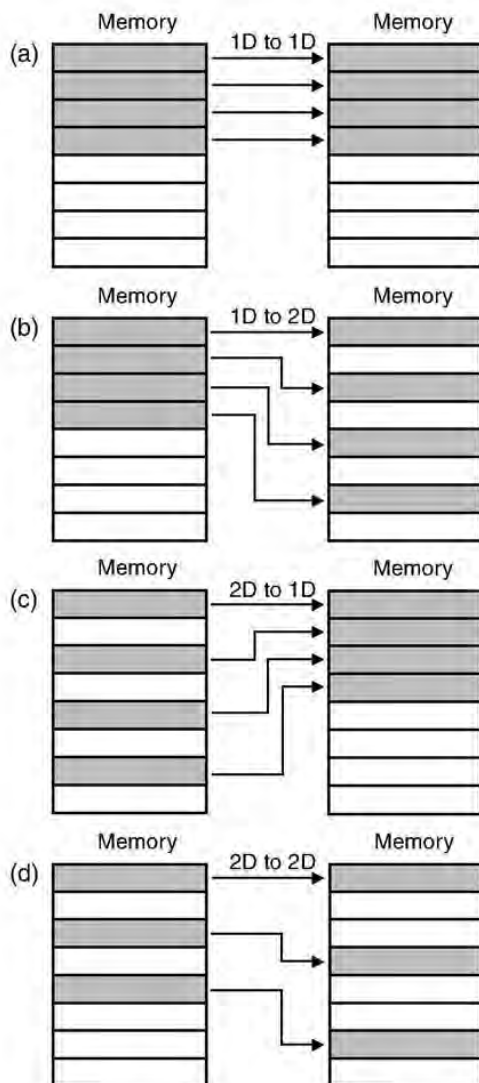


Figure 5.20: Possible Memory DMA configurations.

MemDMA offers a bit more flexibility. For example, we can set up a 1D-to-1D transfer, a 1D-to-2D transfer, a 2D-to-1D transfer, and of course a 2D-to-2D transfer, as shown in Figure 5.20. The only constraint is that the total number of bytes being transferred on each end of the DMA transfer block has to be the same.

has to complete before the next transfer is initiated, Stop mode can guarantee this sequencing.

5.5.6 Descriptor-Based DMA

DMA transfers that are descriptor-based require a set of parameters stored within memory to initiate a DMA sequence. The descriptor contains all of the same parameters normally programmed into the DMA control register set. However, descriptors also allow the chaining together of multiple DMA sequences. In descriptor-based DMA operations, we can program a DMA channel to automatically set up and start another DMA transfer after the current sequence completes. The descriptor-based model provides the most flexibility in managing a system's DMA transfers.

Blackfin processors offer two main descriptor models—a *Descriptor Array* scheme and a *Descriptor List* method. The goal of these two models is to allow a tradeoff between flexibility and performance. Let's take a look at how this is done.

In the Descriptor Array mode, descriptors reside in consecutive memory locations. The DMA controller still fetches descriptors from memory, but because the next descriptor immediately follows the current descriptor, the two words that describe where to look for the next descriptor (and their corresponding descriptor fetches) aren't necessary. Because the descriptor does not contain this Next Descriptor Pointer entry, the DMA controller expects a group of descriptors to follow one another in memory like an array.

A Descriptor List is used when the individual descriptors are not located "back-to-back" in memory. There are actually multiple sub-modes here, again to allow a tradeoff between performance and flexibility. In a "small descriptor" model, descriptors include a single 16-bit field that specifies the lower portion of the Next Descriptor Pointer field; the upper portion is programmed separately via a register and doesn't change. This, of course, confines descriptors to a specific 64 K ($=2^{16}$) page in memory. When the descriptors need to be located across this boundary, a "large" model is available that provides 32 bits for the Next Descriptor Pointer entry.

Regardless of the descriptor mode, using more descriptor values requires more descriptor fetches. This is why Blackfin processors specify a "flex descriptor model" that tailors the descriptor length to include only what's needed for a particular transfer, as shown in Figure 5.27. For example, if 2D DMA is not needed, the YMODIFY and YCOUNT registers do not need to be part of the descriptor block.

5.5.6.1 Descriptor Management

So what's the best way to manage a descriptor list? Well, the answer is application-dependent, but it is important to understand what alternatives exist.

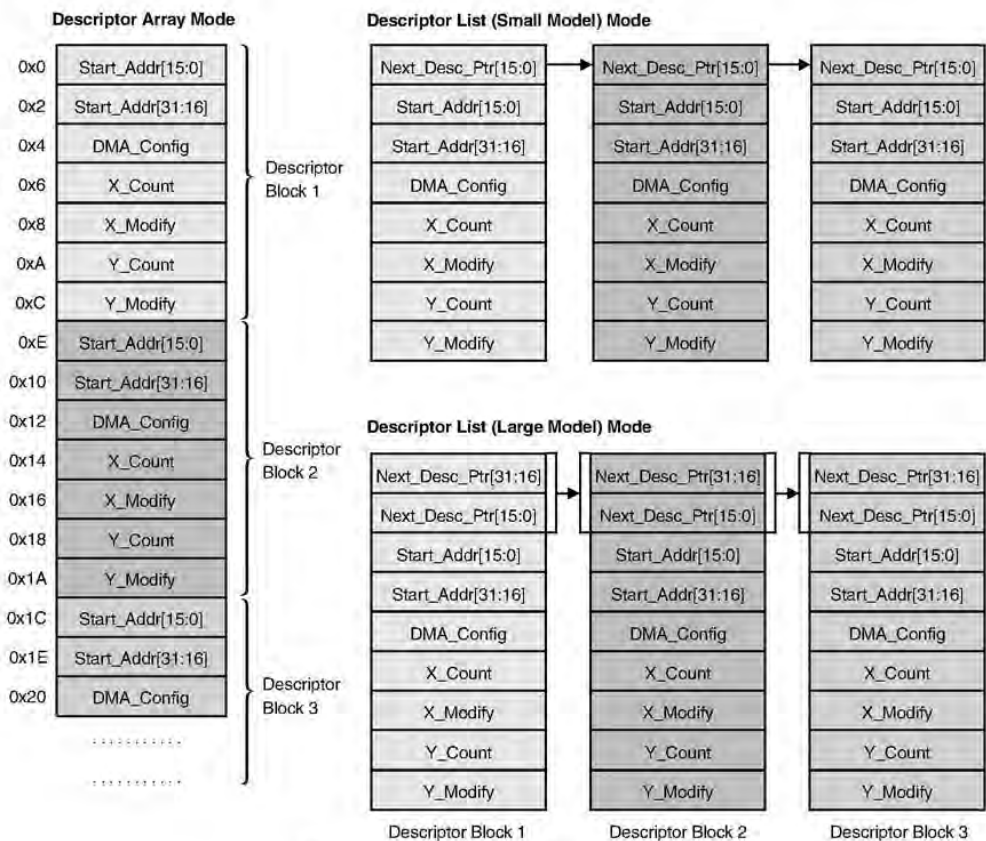


Figure 5.27: DMA descriptor models.

The first option we will describe behaves very much like an Autobuffer DMA. It involves setting up multiple descriptors that are chained together as shown in Figure 5.28a. The term “chained” implies that one descriptor points to the next descriptor, which is loaded automatically once the data transfer specified by the first descriptor block completes. To complete the chain, the last descriptor points back to the first descriptor, and the process repeats. One reason to use this technique rather than the Autobuffer mode is that descriptors allow more flexibility in the size and direction of the transfers. In our YCbCr example (Example 5.5), the Y buffer is twice as large as the other buffers. This can be easily described via descriptors and would be much harder to implement with an Autobuffer scheme.

The second option involves the processor manually managing the descriptor list. Recall that a descriptor is really a structure in memory. Each descriptor contains a configuration word, and each configuration word contains an “Enable” bit which can regulate when a transfer starts. Let’s

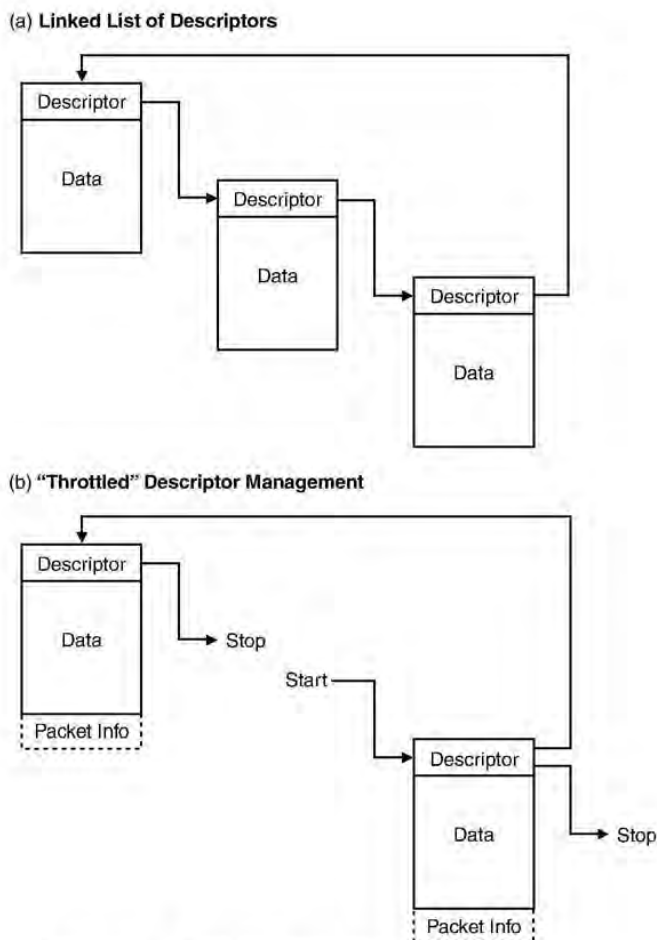


Figure 5.28: DMA descriptor throttled by the processor.

assume we have four buffers that have to move data over some given task interval. If we need to have the processor start each transfer specifically when the processor is ready, we can set up all of the descriptors in advance, but with the “Enable” bits cleared. When the processor determines the time is right to start a descriptor, it simply updates the descriptor in memory and then writes to a DMA register to start the stalled DMA channel. Figure 5.28b shows an example of this flow.

When is this type of transfer useful? EMP applications often require us to synchronize an input stream to an output stream. For example, we may receive video samples into memory at a rate that is different from the rate at which we display output video. This will happen in real systems even when you attempt to make the streams run at exactly the same clock rate.

In cases where synchronization is an issue, the processor can manually regulate the DMA descriptors corresponding to the output buffer. Before the next descriptor is enabled, the processor can synchronize the stream by adjusting the current output descriptor via a semaphore mechanism. For now, you can simply consider semaphores tools that guarantee only one entity at a time accesses a shared resource.

When using internal DMA descriptor chains or DMA-based streams between processors, it can also be useful to add an extra word at the end of the transferred data block that helps identify the packet being sent, including information on how to handle the data and, possibly, a time stamp. The dashed area of Figure 5.28b shows an example of this scheme.

Most sophisticated applications have a “DMA Manager” function implemented in software. This may be provided as part of an operating system or real-time kernel, but it can also run without either of these. In both cases, an application submits DMA descriptor requests to the DMA Queue Manager, whose responsibility it is to handle each request. Usually, an address pointer to a “callback” function is part of the system as well. This function carries out the work you want the processor to perform when a data buffer is ready, without needlessly making the core linger in a high-priority interrupt service routine.

There are two general methods for managing a descriptor queue using interrupts. The first is based on interrupting upon the completion of every descriptor. Use this method only if you can guarantee that each interrupt event will be serviced separately, with no interrupt overrun. The second involves interrupting only on completion of the work transfer specified by the last descriptor of a work block. A work block is a collection of one or more descriptors.

To maintain synchronization of the descriptor queue, you need to maintain in software a count of descriptors added to the queue, while the interrupt handler maintains a count of completed descriptors removed from the queue. The counts are then equal only when the DMA channel pauses after having processed all the descriptors.

5.5.7 Advanced DMA Features

5.5.7.1 System Performance Tuning

To effectively use DMA in a multimedia system, there must be enough DMA channels to support the processor’s peripheral set fully, with more than one pair of Memory DMA streams. This is an important point, because there are bound to be raw media streams incoming to external memory (via high-speed peripherals), while at the same time data blocks will be moving back and forth between external memory and L1 memory for core processing. What’s more, DMA engines that allow direct data transfer between peripherals and external memory, rather than requiring a stopover in L1 memory, can save extra data passes in numerically intensive algorithms.

As data rates and performance demands increase, it becomes critical to have “system performance tuning” controls at your disposal. For example, the DMA controller might be optimized

EXHIBIT E

PrimeCell® Single Master DMA Controller (PL081)

Revision: r1p2

Technical Reference Manual

ARM®

Copyright © 2001, 2003-2005 ARM Limited. All rights reserved.
ARM DDI 0218E

96800DOC00033366

Petitioner Intel Corp., Ex. 1012
IPR2023-00783

PrimeCell Single Master DMA Controller (PL081) Technical Reference Manual

Copyright © 2001, 2003-2005 ARM Limited. All rights reserved.

Release Information

The following changes have been made to this document.

Change history

Date	Issue	Confidentiality	Change
May 2001	A	Non-Confidential	First release.
July 2001	B	Non-Confidential	Section 3.8.1 and 3.8.2 revised. Figure B-9 and B-17 revised. Section added: Memory-to-peripheral transaction under DMA Controller flow control.
February 2003	C	Non-Confidential	Release for revision r1p1.
28 January 2004	D	Non-Confidential	First Issue for r1p2. Incorporating errata from Issue C.
20 December 2005	E	Non-Confidential	Second Issue for r1p2. Addressing defects and general maintenance.

Proprietary Notice

Words and logos marked with ® or ™ are registered trademarks or trademarks owned by ARM Limited. Other brands and names mentioned herein may be the trademarks of their respective owners.

Neither the whole nor any part of the information contained in, or the product described in, this document may be adapted or reproduced in any material form except with the prior written permission of the copyright holder.

The product described in this document is subject to continuous developments and improvements. All particulars of the product and its use contained in this document are given by ARM in good faith. However, all warranties implied or expressed, including but not limited to implied warranties of merchantability, or fitness for purpose, are excluded.

This document is intended only to assist the reader in the use of the product. ARM Limited shall not be liable for any loss or damage arising from the use of any information in this document, or any error or omission in such information, or any incorrect use of the product.

Confidentiality Status

This document is Non-Confidential. The right to use, copy and disclose this document may be subject to license restrictions in accordance with the terms of the agreement entered into by ARM and the party that ARM delivered this document to.

Product Status

The information in this document is final, that is for a developed product.

Web Address

<http://www.arm.com>

Contents

PrimeCell Single Master DMA Controller (PL081) Technical Reference Manual

	Preface	
	About this manual	xii
	Feedback	xvi
Chapter 1	Introduction	
	1.1 About the SMDMAC	1-2
	1.2 Product revisions	1-4
Chapter 2	Functional Overview	
	2.1 SMDMAC functional description	2-2
	2.2 System considerations	2-9
	2.3 System connectivity	2-10
	2.4 Use with memory management unit based systems	2-16
Chapter 3	Programmer's Model	
	3.1 About the programmer's model	3-2
	3.2 Programming the SMDMAC	3-3
	3.3 Summary of SMDMAC registers	3-6
	3.4 Register descriptions	3-8
	3.5 Address generation	3-34

Contents

	3.6	Scatter/gather	3-35
	3.7	Interrupt requests	3-37
	3.8	SMDMAC data flow	3-40
Chapter 4		Programmer's Model for Test	
	4.1	SMDMAC test harness overview	4-2
	4.2	Scan testing	4-3
	4.3	Test register descriptions	4-4
	4.4	Integration test	4-7
Appendix A		Signal Descriptions	
	A.1	DMA interrupt request signals	A-2
	A.2	DMA request and response signals	A-3
	A.3	AHB slave signals	A-4
	A.4	AHB master signals	A-6
	A.5	AHB master bus request signals	A-8
	A.6	Scan test control signals	A-9
Appendix B		DMA Interface	
	B.1	DMA request signals	B-2
	B.2	DMA response signals	B-3
	B.3	Flow control	B-4
	B.4	Transfer types	B-5
	B.5	Signal timing	B-16
	B.6	Functional timing diagram	B-17
	B.7	SMDMAC transfer timing diagram	B-18
Appendix C		Scatter/Gather	
	C.1	Scatter/gather through linked list operation	C-2
		Glossary	

Appendix C

Scatter/Gather

This section describes scatter/gather through LLI. It contains the following section:

- *Scatter/gather through linked list operation* on page C-2.

Scatter/Gather

C.1 Scatter/gather through linked list operation

A series of linked lists define the source and destination data areas. Each LLI controls the transfer of one block of data, and then optionally loads another LLI to continue the DMA operation, or stops the DMA stream. The first LLI is programmed into the SMDMAC.

The data to be transferred, described by a LLI, referred to as the packet of data, usually requires one or more DMA bursts, to each of the source and destination.

Figure C-1 shows an example of an LLI. A rectangle of memory has to be transferred to a peripheral. The addresses of each line of data are given, in hexadecimal, at the left-hand side of the figure. The LLIs describing the transfer are to be stored contiguously from address 0x20000.

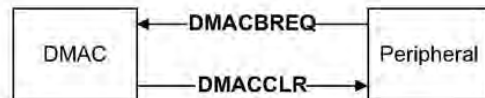


Figure C-1 LLI example

The first LLI, stored at 0x20000, defines the first block of data to be transferred. This is the data stored between addresses 0x0A200 and 0x0AE00:

- source start address 0x0A200
- destination address set to the destination peripheral address
- transfer width, word, 32-bit
- transfer size, 3 072 bytes, 0xC00
- source and destination burst sizes, 16 transfers
- next LLI address, 0x20010.

The second LLI, stored at 0x20010, describes the next block of data to be transferred:

- source start address 0x0B200
- destination address set to the destination peripheral address
- transfer width, word, 32-bit
- transfer size, 3 072 bytes, 0xC00
- source and destination burst sizes, 16 transfers
- next LLI address, 0x20020.

A chain of descriptors is built up, each one pointing to the next in the series. To initialize the DMA stream, the first LLI, 0x20000, is programmed into the SMDMAC. When the first packet of data has been transferred, the next LLI is automatically loaded.

Scatter/Gather

The final LLI is stored at 0x20070 and contains:

- source start address 0x11200
- destination address set to the destination peripheral address
- transfer width, word, 32-bit
- transfer size, 3 072 bytes, 0xC00
- source and destination burst sizes, 16 transfers
- next LLI address, 0x0.

Because the next LLI address is set to zero, this is the last descriptor, and the DMA channel is disabled after transferring the last item of data. The channel is probably set to generate an interrupt at this point to indicate to the ARM processor that the channel can be reprogrammed.

Cache way	A group of cache lines (or blocks). It is 2 to the power of the number of index bits in size. <i>See also</i> Cache terminology diagram on the last page of this glossary.
Cold reset	Also known as power-on reset. Starting the processor by turning power on. Turning power off and then back on again clears main memory and many internal settings. Some program failures can lock up the processor and require a cold reset to enable the system to be used again. In other cases, only a warm reset is required. <i>See also</i> Warm reset.
Coprocessor	A processor that supplements the main processor. It carries out additional functions that the main processor cannot perform. Usually used for floating-point math calculations, signal processing, or memory management.
Core	A core is that part of a processor that contains the ALU, the datapath, the general-purpose registers, the Program Counter, and the instruction decode and control circuitry.
CoreSight	The infrastructure for monitoring, tracing, and debugging a complete system on chip.
Data Abort	An indication from a memory system to the core of an attempt to access an illegal data memory location. An exception must be taken if the processor attempts to use the data that caused the abort. <i>See also</i> Abort, External Abort, and Prefetch Abort.
Data cache	A block of on-chip fast access memory locations, situated between the processor and main memory, used for storing and retrieving copies of often used data. This is done to greatly increase the average speed of memory accesses and so improve processor performance.
Direct Memory Access (DMA)	An operation that accesses main memory directly, without the processor performing any accesses to the data concerned.
DMA	<i>See</i> Direct Memory Access.
Endianness	Byte ordering. The scheme that determines the order that successive bytes of a data word are stored in memory. An aspect of the system's memory mapping. <i>See also</i> Little-endian and Big-endian
Event	1 (Simple) An observable condition that can be used by an ETM to control aspects of a trace. 2 (Complex) A boolean combination of simple events that is used by an ETM to control aspects of a trace.

IN THE UNITED STATES DISTRICT COURT
FOR THE WESTERN DISTRICT OF TEXAS
WACO DIVISION

BiTMICRO LLC,

Plaintiff,

v.

KIOXIA AMERICA, INC.
And KIOXIA CORPORATION,
Defendants.

Case No. 6:22-cv-00331-ADA

BiTMICRO LLC,

Plaintiff,

v.

INTEL CORPORATION,
Defendant.

Case No. 6:22-cv-00335-ADA

**DECLARATION OF JANICE L. TA IN SUPPORT OF
DEFENDANTS' OPENING CLAIM CONSTRUCTION BRIEF**

I, Janice L. Ta, declare as follows:

1. I am admitted to practice in the Western District of Texas and am a partner at the law firm of Perkins Coie, LLP, counsel of record for Intel Corporation (“Intel”) in the above-captioned action. I submit this declaration in support of Defendants’ Opening Claim Construction Brief in the above-captioned matter.

2. Attached hereto as **Exhibit F** is a true and correct copy of an excerpt from the prosecution File History for U.S. Patent No. 7,826,243.

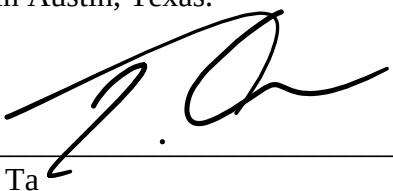
3. Attached hereto as **Exhibit G** is a true and correct copy of an excerpt from *The Authoritative Dictionary of IEEE Standard Terms* (7th ed.) (Inst. of Elec. & Electronics Engineers, Inc. 2000).

4. Attached hereto as **Exhibit H** is a true and correct copy of an excerpt from *Microsoft Computer Dictionary* (5th ed.) (Microsoft Corp. 2002).

5. Attached hereto as **Exhibit I** is a true and correct copy of an excerpt from the prosecution File History of U.S. Patent No. 6,496,939.

I declare under penalty of perjury under the laws of the United States that to the best of my knowledge and recollection the foregoing is true and correct.

EXECUTED this 9th day of November, 2022, in Austin, Texas.



Janice L. Ta

EXHIBIT F

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Application of: BRUCE et al.	)	Group Art Unit: 2892
	)	
Serial No.: 11/322,442	)	Examiner: ULLAH, Elias
	)	
Application Filing Date: 29 Dec 2005	)	
	)	
For: MULTIPLE CHIP MODULE AND PACKAGE	)	
STACKING FOR STORAGE DEVICES	)	
	)	

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

AMENDMENT AND REPLY TO SECOND OFFICE ACTION

Dear Examiner:

In response to the Office Action, mailed 13 May 2009, please kindly amend this Application as follows and consider the remarks set forth below.

Amendments to the Specification begin on page 2;

A listing of claims begins on page 7;

Amendment(s) to the Drawings begins on page 19; and

Remarks begin on page 21 of this paper.

Replacement sheets of drawings incorporating the changes proposed in the Amendments to the Drawings and corresponding to the written description, respectively, are attached to this paper.

Amendments to the Specification

Text in underline indicates new text, while text in strike-out format or between a pair of double brackets indicates deletion of the text.

In the Specification, as previously amended, please enter the amendments shown with respect to the replacement paragraphs identified below.

Please replace the title on page 1, lines 1-2, with the following replacement title.

Multiple Chip Module and Package Stacking For Storage Devices

Please replace the paragraph on page 8, line 13 through 15, with the following replacement paragraph.

Figure 21a shows another pin assignment and connection technique for creating be able to ~~connect~~ a serial chain route through ~~from~~ multiple modules in a stack by connecting a set of serial chain connections, according to another example embodiment of the present invention.

Please replace the paragraph on page 20, line 19 through page 21, line 16, with the following replacement paragraphs.

Figure 19 shows a first pin assignment and connection technique that enables a selection of specific module in a multiple module stack. The figure illustrates the controller and SDRAM modules as an example. The SDRAM module balls, such as **1907**, are represented by big oblong shapes, and the pads such as **1908** are represented by smaller oblong shapes. A pad to ball connection is represented by a rectangle, such as **1906**. The controller module has four active pads such as **1904**, one each for the four SDRAM modules such as **1901**. An active pad may be also

referred to herein as an “active port”. The controller module’s active pad 00 **1904** is ultimately connected to ~~SDRAM module 0’s~~ X0 ball **1903** of SDRAM module 0, active pad 01 is connected to ~~SDRAM module 1’s~~ X0 ball **1902** of SDRAM module 1, and so on. These[[The]] X0 balls are the active balls for these[[the]] SDRAM modules. Thus, as illustrated in Figure 19, there is one active ball, such as X0, ultimately coupled to one active pad per module.

All SDRAM modules are identical, and in the example shown in Figure 19, [[which]] include the following connections as follows: an X1 passive ball is connected to an X1 passive pad, an X2 passive ball to X2 passive pad, and so on. The connection between a passive ball on one surface of a SDRAM module and a passive pad on another surface of the same SDRAM module, such as the X1 passive ball and X1 passive pad connection in Figure 19, is named “passive port”, such as passive ports 1909 illustrated in Figure 19. The technique involves using a ladder-like routing path, such as **1905** on the stacked modules. This technique enables the controller module’s active pad to be routed to the desired specific module in the stack. Thus, an active signal applied at an active pad, such as active pad 02, shown with reference number 1904 on controller module 1910, is routed through at least one passive port connecting a passive ball and passive pad, such as the passive ports 1905 connecting passive ball X2 to passive pad X2 and passive ball X1 to passive pad X2, respectively, so that this active signal eventually reaches an active ball, such as X0 ball SDRAM Module 2 in fig. 19. The example embodiment illustrated in Figure 19 comprises a repeating pattern for the balls and pads. Passive balls are connected within each module to passive pads with an offset distance equaling one periodic distance of the repeating pattern. For example in Figure 19, an X1 passive ball is connected to an X1 passive pad, X2 passive ball to X2 passive pad, etc. The importance is the approach By using repeating patterns and offsetting of the passive ball to passive pad within each module, other offset distances

can be used ~~[[also]]~~ in ~~alternative~~~~other~~ embodiments of the present invention.~~[[,]] e.g. For~~
example, an alternative embodiment may include using an offset of two or any multiple of the
periodic distance of the repeating pattern. This~~[[The]] alternative-example~~ embodiment is not
intended to~~does not~~ limit the scope of the present invention in any way.

Please amend the paragraph on page 23, line 1 through page 24, line 6, with the following replacement paragraphs.

The last connection technique is used for serial routing of all modules in the stack and allowing them to be externally accessible for horizontal expansion on a PCB, such as a main board. Figure 21a shows how this technique is implemented in the stacking method. The main board **2101** ~~[[]]~~ contains the input signals **2102** which will then be connected to IN ball **2104** of the base module **2103**. The modules, including base module **2103**, stack modules **2106** and **2130**, and an end module, such as top module **2108**, each have a ball and a pad that are coupled together by a serial chain circuit. For example, base module **2103** includes a serial chain circuit **2127** that couples IN ball **2104** with OUT pad **2105**, and a serial chain circuit **2136** that couples OUT pad **2137** with OUT ball **2138**; module **2106** includes a serial chain circuit **2128** that couples IN ball **2107** with OUT pad **2129**, and a serial chain circuit **2139** that couples OUT pad **2140** with OUT ball **2141**; module **2130** includes a serial chain circuit **2131** that couples IN ball **2132** with OUT pad **2133**, and a serial chain circuit **2142** that couples OUT pad ball **2143** with OUT ball **2144**; and top module **2108** includes a serial chain circuit **2134** that couples IN ball **2135** with OUT pad **2109** and a serial chain circuit **2145** that couples OUT pad **2110** with OUT ball **2113**. Thus, the~~[[The]]~~ base module's OUT pad **2105** is connected to IN ball **2107** of the first stack module **2106**, and so on.

A signal route that includes a serial chain circuit coupling an ball and pad is herein after referred to as a "serial chain connection", and at least two serial chain connections connected in

series are herein after referred to as a “serial chain route”, such as serial chain route **2146** or **2111**. The top module **2108**[[,]] terminates the signal to the top pad **2109** and routes, with connection **2112**, the signal internally to another pad, such as OUT pad **2110**, which is connected directly to the ball [[**2113**]] underneath, such as OUT ball **2113**. Since the modules are identical, this routed signal passes through another set of [[the]] serial chain connections that comprise another serial chain route, such as serial chain route pads and balls-2111 for[[of]] the stack until it reaches the OUT pad **2114** of the main board for external access. A serial chain connection includes a serial chain input and a serial chain output. A serial chain input is disposed to receive a signal, while a serial chain output is disposed to provide the signal. For example in Figure 21a, IN balls **2104**, **2107**, **2132**, and **2135**, and OUT pads **2110**, **2143**, and **2140** are serial chain inputs, respectively. OUT pads **2105**, **2129**, **2133**, and **2109**, and OUT balls **2113**, **2144**, **2141**, and **2138** are serial chain outputs, respectively. At least one of the balls and pads that are coupled to their respective chain circuit may be in the form of a passive ball and passive pad, respectively, since an active signal, such as signal **2102**, applied at active pad **2101**, can be routed through a series of serial chain circuits that couple respective balls and pads. In effect, each of these serial chain circuits functions as a passive port. A serial chain circuit, however, can also terminate a serial chain route, such as serial chain route **2146**, and branching or routing this serial chain route to another pad that functions as a serial chain input of another serial chain route, such as serial chain route **2111**, as described further below.

Regarding~~How~~ the top module, ~~terminates~~ the serial chain termination and branchings ~~them out~~ to other pads is shown in Figure 21b. All modules have a control circuit that includes a selectable buffer **2115** that tri-states the input when pulled low. The signal from the IN balls **2116** enters the internal circuit **2117** and exits the circuit connecting to the OUT pad **2118**. The buffer’s **2115** control line is weakly pulled up internally. The “StkLow” ball is connected internally to

GND **2119**, thus pulling down the buffer control line when a module is stacked above it, and it is pulled up when no module is stacked directly above it. When the buffer control line is pulled high, it will let the input signal branch out **2120** to the other pads, thus creating a connection route or trace that loops back to the base module solder balls. When a module is stacked above a module that is disposed with selectable buffer **2115** in the manner shown, the buffer control signal is pulled low, tri-stating the input signal and disallowing the branching effect. This technique allows the signal to be accessible to the external balls of the base module, thus horizontal expansion for a serial signal is achievable. The balls are then routed, with connection **2121**, to the other modules **2122** on the other locations. The same technique is used on the main board **2123**, when no package is detected on the “StkLow” pad, the buffer allows the input to connect to the designated pads on the other locations. The tri-stated buffer technique is redundant to all locations. An example is the JTAG TDI-TDO signal. The driver circuit **2125** sends out TDI signals to the pads and the closing TDO signal **2124** loops back to the driver circuit.

Listing of Claims:

This listing of claims will replace all prior versions, and listings, of claims in the application.

1. (PREVIOUSLY CANCELLED) A method for stacking a plurality of modules comprising the steps of:

providing one or more active port on the plurality of modules for carrying one or more active signal;

providing one or more passive port on the plurality of modules for passing through the one or more active signals; and

stacking the plurality of modules.

2. (PREVIOUSLY CANCELLED) The method of claim 1 wherein the one or more passive port forms a ladder like routing path.

3. (PREVIOUSLY CANCELLED) The method of claim 1 wherein one or more of the plurality of module is rotated before being stacked and the one or more passive port forms a rotational routing path.

4. (PREVIOUSLY CANCELLED) The method of claim 1 further comprising the step of providing one or more pair of daisy chain input port and daisy chain output port for forming a first daisy chain connection wherein each pair of the daisy chain input port and the daisy chain output port are connected through a daisy chain circuit.

5. (PREVIOUSLY CANCELLED) The method of claim 4 further comprising the step of providing one or more control port for enabling a routing path extending the first daisy chain connection within an end module.

6. (PREVIOUSLY CANCELLED) The method of claim 5 wherein the routing path within the end module is connected to a second daisy chain connection.

7. (PREVIOUSLY CANCELLED) The method of claim 1 wherein one or more of the plurality of modules is a main board.

8. (PREVIOUSLY CANCELLED) The method of claim 7 wherein one or more vertically stacked module is connected to the main board.

9. (PREVIOUSLY CANCELLED) The method of claim 2:

wherein each of the plurality of modules comprises a first side and a second side;

wherein the one or more active port is connected to one or more active ball on the first side of one or more corresponding module;

wherein the one or more passive port is connected to one or more passive ball on the first side and one or more passive pad on the second side of one or more corresponding module;

wherein each of the one or more passive ball and each of the one or more passive pad connected to a same passive port are located between a pre-determined distance to each other in a repeating pattern, the pre-determined distance equaling a multiple of a

periodic distance of the repeating pattern;

wherein one or more active port of a base module is routed to a same location on one or more modules through the ladder like routing path; and

wherein the ladder like routing path comprises one or more passive ball and one or more passive pad.

10. (PREVIOUSLY CANCELLED) The method of claim 3:

wherein each of the plurality of modules comprises a first side and a second side;

wherein the one or more active port is connected to one or more active ball on the first side of one or more corresponding module;

wherein the one or more passive port is connected to one or more passive ball on the first side and one or more passive pad on the second side of one or more corresponding module;

wherein each of the one or more passive ball and each of the one or more passive pad connected to a same passive port are located opposite each other in a repeating pattern;

wherein one or more active port of a base module is routed to a same location on one or more modules through the rotational routing path; and

wherein the rotational routing path comprises one or more passive ball and one or more passive pad.

11. (PREVIOUSLY CANCELLED) The method of claim 4:

wherein each of the plurality of modules comprises a first side and a second side;

wherein the one or more daisy chain input port is connected to one or more passive ball on the first side and the one or more daisy chain output port is connected to one or more passive pad on the second side of one or more corresponding module;

wherein each of the one or more passive ball and each of the one or more passive pad corresponding to a same daisy chain port are located opposite each other;

wherein one or more active port of a base module is routed to a same location on one or more modules through the daisy chain connection; and

wherein the daisy chain connection comprises one or more daisy chain circuit, one or more passive ball and one or more passive pad.

12. (PREVIOUSLY CANCELLED) The method of claim 5:

wherein each of the plurality of modules comprises a first side and a second side;

wherein the one or more daisy chain input port is connected to one or more passive ball on the first side and the one or more daisy chain output port is connected to one or more passive pad on the second side of one or more corresponding module;

wherein each of the one or more passive ball and each of the one or more passive pad corresponding to a same daisy chain port are located opposite each other;

wherein one or more active port of a base module is routed to a same location on one or more modules through the daisy chain connection;

wherein the daisy chain connection comprises one or more daisy chain circuit, one or more passive ball and one or more passive pad; and

wherein the routing path comprises one or more tri-state driver controlled by the one or more control port.

13. (PREVIOUSLY CANCELLED) The method of claim 6:

wherein each of the plurality of modules comprises a first side and a second side;

wherein the one or more daisy chain input port is connected to one or more passive ball on the first side and the one or more daisy chain output port is connected to one or more passive pad on the second side of one or more corresponding module;

wherein each of the one or more passive ball and each of the one or more passive pad corresponding to a same daisy chain port are located opposite each other;

wherein one or more active port of a base module is routed to a same location on one or more modules through the daisy chain connection;

wherein the daisy chain connection comprises one or more daisy chain circuit, one or more passive ball and one or more passive pad;

wherein the routing path comprises one or more tri-state driver controlled by the one or more control port; and

wherein one or more daisy chain connection are extended through one or more vertically stacked module connected to a main board.

14. (CURRENTLY AMENDED) A stacked module comprising a plurality of modules each comprising:

one or more active ports for carrying one or more active signal; [[and]]

one or more passive ports for passing through the one or more active signals;[[.]]

a first serial chain route that includes at least one serial chain connection, the serial chain connection including: a serial chain circuit, a serial chain input, and serial chain output; said serial chain input coupled to said serial chain output through said serial chain circuit;

a second serial chain route and a control circuit for enabling a routing path that connects the first serial chain route with the second serial chain route within an end module; and

said control circuit is disposed to enable the routing path in response to a control input signal received from another module from the plurality of modules when the end module is coupled to the another module.

15. (CURRENTLY AMENDED) The stacked module of claim 14 wherein the one or more passive port forms a ladder like routing path, and the passive port connects a passive ball on one surface of a module to a passive pad on another surface of the module.

16. (CURRENTLY AMENDED) The stacked module of claim 14 wherein one or more of the plurality of modules is rotated before being stacked and the one or more passive port forms a rotational routing path.

17. (CANCELLED) The stacked module of claim 1 further comprising one or more pair of daisy chain input port and daisy chain output port for forming a first daisy chain connection wherein each pair of the daisy chain input port and the daisy chain output port are connected through a daisy chain circuit.

18. (CANCELLED) The stacked module of claim 17, further comprising one or more control port for enabling a routing path extending the first daisy chain connection within an end module.

19. (CANCELLED) The stacked module of claim 18 wherein the routing path within the end module-is connected to a second daisy chain connection.

20. (CURRENTLY AMENDED) The stacked module of claim 14 wherein one or more of the plurality of modules includes ~~is~~-a main board.

21. (ORIGINAL) The method of claim 20 wherein one or more vertically stacked module is connected to the main board.

22. (CURRENTLY AMENDED) The stacked module of claim 15:

wherein each of the plurality of modules comprises a first side and a second side;

wherein the one or more active port is connected to one or more active ball on the first side of one or more corresponding module;

wherein the one or more passive port is connected to one or more passive ball on the first side and one or more passive pad on the second side of one or more corresponding module;

wherein each of the one or more passive ball and each of the one or more passive pad connected to a same passive port are located between a pre-determined distance to each other in a repeating pattern, the pre-determined distance equaling a multiple of a periodic distance of the repeating pattern;

wherein one or more active port of a base module is routed to a same location on one or more modules through the ladder like routing path; and

wherein at least one of the ladder-like routing path comprises one or more passive ball and one or more passive pad forms a ladder like routing path.

23. (CURRENTLY AMENDED) The stacked module of claim 16:

wherein each of the plurality of modules comprises a first side and a second side;

wherein the one or more active port is connected to one or more active ball on the first side of one or more corresponding module;

wherein the one or more passive port is connected to one or more passive ball on the first side and one or more passive pad on the second side of one or more corresponding module;

wherein each of the one or more passive ball and each of the one or more passive pad connected to a same passive port are located opposite each other in a repeating pattern;

wherein one or more active port of a base module is routed to a same location on one or more modules through the rotational routing path; and

wherein at least one of the ~~rotational routing path~~ comprises one or more passive ball and one or more passive pad form a rotational routing path.

24. (CURRENTLY AMENDED) The stacked module of claim 14[[17]]:

wherein each of the plurality of modules comprises a first side and a second side, and the one or more passive ports includes a first passive port connected to a passive ball disposed on the first side and to a passive pad disposed on the second side of a module from the plurality of modules;

wherein the ~~one or more~~ serial[[daisy]] chain input ~~port is~~ includes connected to ~~the one or more~~ passive ball ~~on the first side~~ and the ~~one or more~~ serial daisy chain output ~~port includes the~~ is connected to one or more passive pad ~~on the second side of one or more corresponding module;~~

wherein ~~each of the one or more~~ passive ball and ~~each of the one or more~~ passive pad ~~corresponding to a same daisy chain port~~ are located opposite each other; and

further including a base module having an active port for providing a signal, said signal routed to a same location on the plurality of modules through the first serial chain route.

~~wherein one or more active port of a base module is routed to a same location on one or more modules through the daisy chain connection; and~~

~~wherein the daisy chain connection comprises one or more daisy chain circuit, one or more passive ball and one or more passive pad.~~

25. (CURRENTLY AMENDED) The stacked module of claim 14[[18]]:

~~wherein each of the plurality of modules comprises a first side and a second side;~~

~~wherein the one or more daisy chain input port is connected to one or more passive ball on the first side and the one or more daisy chain output port is connected to one or more passive pad on the second side of one or more corresponding module;~~

~~wherein each of the one or more passive ball and each of the one or more passive pad corresponding to a same daisy chain port are located opposite each other;~~

~~wherein one or more active port of a base module is routed to a same location on one or more modules through the daisy chain connection;~~

~~wherein the daisy chain connection comprises one or more daisy chain circuit, one or more passive ball and one or more passive pad; and~~

~~wherein the routing path includes comprises one or more tri-state driver controlled by the one or more control circuit port.~~

26. (CURRENTLY AMENDED) The stacked module of claim 25[[19]] further including:

~~wherein each of the plurality of modules comprises a first side and a second side;~~

~~wherein the one or more daisy chain input port is connected to one or more passive ball on the first side and the one or more daisy chain output port is connected to one or more passive pad on the second side of one or more corresponding module;~~

~~wherein each of the one or more passive ball and each of the one or more passive pad corresponding to a same daisy chain port are located opposite each other;~~

~~wherein one or more active port of a base module is routed to a same location on one or more modules through the daisy chain connection;~~

~~wherein the daisy chain connection comprises one or more daisy chain circuit, one or more passive ball and one or more passive pad;~~

~~wherein the routing path comprises one or more tri-state driver controlled by the one or more control port;~~

~~wherein one or more daisy chain connection are extended through one or more vertically stacked module connected to a main board.~~

a main board coupled to another stacked module having a third serial chain route; and

another control circuit for enabling another routing path that connects the second serial chain route with the third serial chain route.

27. (CURRENTLY AMENDED) The stacked module of claim 15, wherein:

the stacked module further includes a first outside surface and a second outside surface, the first and second outside surfaces disposed to face opposite each other; and

at least one of the plurality of modules includes a first module having a first target location terminating between the first and second outside surfaces.

28 (PREVIOUSLY ADDED) The stacked module of claim 27, wherein the plurality of modules further includes a second module and a third module, the

second module including a second module surface that includes the first outside surface and the third module including a third module surface that includes the second outside surface.

29. (PREVIOUSLY ADDED) The stacked module of claim 15, further including a second ladder like routing path; and wherein:

the plurality of modules includes a first module and a second module, the first and second modules respectively including a first target location and a second target location, the second target location located within the stacked module; and

the ladder like routing path for routing one of the active port to the first target location, and the second ladder like routing path for routing another one of the active port to the second target location.

30. (PREVIOUSLY ADDED) The stacked module of claim 29, further including a second ladder like routing path; and wherein:

the stacked module includes a first outside surface and a second outside surface, the first and second outside surfaces disposed to face opposite each other;

the ladder like routing path terminates at the first target location; and

the second ladder like routing path terminates at the second target location, the second target terminating within the stacked module and located between the first and second outside surfaces;

31. (PREVIOUSLY ADDED) The stacked module of claim 29, wherein the second target location is located between the first and second outside surfaces.

Amendment(s) to the Drawings

The following proposed amendments to the Drawings are provided in the form of replacement drawing sheets, which are each labeled "Replacement Sheet" in their respective page headers pursuant to 37 CFR 1.84(c). Support for the amendments below is provided on pages XX in the Substitute Specification previously submitted on 12 February 2009. No new matter has been added.

Figure 19 has been amended as described below.

A set of parenthesis has been placed around the text "Controller Module".

Reference number "1910" has been placed above the text "(Controller Module)".

Multiple instances of reference number "1909" has been inserted as shown in the Replacement Sheet; Reference number "1909" corresponds to the element named "passive port" in the Specification as amended at p. 22.

Multiple instances of reference number "1904" has been inserted to the drawing elements annotated with "01", "02", and "03", and referred to in the Specification as "active pads" or "active ports" at p. 21.

Figure 21a in drawing sheet 27/28 has been amended as described below:

Element numbers 2103, 2106, and 2108 have been moved from the left to the right side of the Drawing sheet.

Element numbers 2127 through 2135, inclusive, element numbers 2137, 2138, 2140, 2141, and 2143 have been added as shown.

Drawing elements corresponding to new element numbers 2136, 2139, 2142, 2145, 2146 have been added.

Drawing element corresponding to element number 2111 has been adjusted in length and position as shown.

The figure in drawing sheet 28/28 has been amended by replacing the figure label "Figure 21a" to "Figure 21b". The drawing on drawing sheet 28/28 as originally filed was represented with the figure label of "Figure 21b" but the Amendments to the Drawings submitted on 13 February 2009 was inadvertently amended to include "Figure 21a".

Remarks

The specification and drawings have been amended to correct minor errors noted in the Office Action and otherwise. These corrections are of a clerical nature and do not add "new matter".

Claims 1 and 13 have been previously cancelled without prejudice.

Claims 14-16, 20 and 22-27 are amended.

Claims 17-19 are cancelled without prejudice.

The text of original claim 21 and of previously added claims 28-31 remain unchanged but their meaning have changed because they depend from amended claims.

Request for Acknowledgement and Acceptance of Substitute Drawings Submitted on
13 February 2009

In the Office Action, the drawings filed on "29 December 2005", named the "Acknowledgment Date", were "accepted" by the Examiner. Applicants kindly note that the Drawings were amended and provided in the form of Replacement Sheets, collectively referred to herein as the "Amended Drawings", with the USPTO by the Applicants through the Response and Amendment to Office Action submitted with the USPTO on 13 February 2009, named the "Prior Response". The Acknowledgment Date used in the Office Action, however, reflects the original Application filing date of this application and thus, refers to the original Drawings rather than the Amended Drawings. With due respect, the Acknowledgement seems to be inadvertent, and consequently, Applicants request that the Amended Drawings previously submitted

in the Prior Response also be reviewed and acknowledged if applicable.

Amendments to The Current Drawings

Figures 19 and 21a, and the Figure on sheet 28/28 have been amended, and are provided in the form of replacement drawing sheets in accordance with 37 CFR 1.121. These amendments are described in detail in the "Amendments to the Drawings" section above.

Examiner Interview Conducted on 8 May 2009

Withdrawn claims 17-19 and 24-26 have been reconsidered by the Examiner because claim 17 has been amended to depend from claim 14 instead of claim 1 during the telephone interview, named "Examiner Interview", conducted between the undersigned and the Examiner on 8 May 2009. Accordingly, claims 17-19 and 24-26 have been entered for further consideration by the Examiner. The undersigned on behalf of the applicants extend much thanks to the Examiner for his responsiveness during the Examiner Interview.

The First 35 USC §112, Second Paragraph, Rejection

Claims 14-26 are rejected under 35 USC, second paragraph, as being indefinite for failing to particularly point out and distinctly claiming the subject matter which the applicant regards as the invention. The Applicants respectfully traverse this rejection.

In the Office Action, it is asserted that in:

base claim 1, applicants claimed "one or more passive port for passing though the one or more active signals". However, in light of applicants'

discloser, as it would be interpreted by one of ordinary skill in the art, above claim limitations do not set out and define the invention with a reasonable degree of precision and particularity. For purpose of examination, Examiner assumes the "passive port" is a connection on active port for carrying signal.

Definiteness of claim language must be analyzed, not in a vacuum, but in light of Applicants written disclosure, the prior art, and the claim interpretation that would be given by one of ordinary skill in the art at the time the invention was made. MPEP 2173.02. Applicants may be their own lexicographers. MPEP 2173.01. The Specification at p. 21, line 7 through p. 22, line 4, as amended in the "Amendments to the Specification" section above, recites in part:

...A **connection** between a passive ball on one surface of a SDRAM module and a passive pad on another surface of the same SDRAM module, such as the X1 passive ball and X1 passive pad connection in Figure 19, is named "passive port", such as passive ports 1909 illustrated in Figure 19. (Emphasis added).

The above description should be read in conjunction with amended Figure 19, which shows that a passive port 1909 **connects** "a passive ball", such as passive ball X2, "on one surface of a SDRAM module", such as SDRAM module 0, and a passive pad", such as passive pad X3, "on another surface of the same DRAM module", such as SDRAM module 0. Thus, "a connection between a passive ball on one surface of a SDRAM module and a passive pad on another surface of the same SDRAM module" is a "passive port". The above amendments render moot the First 35 USC § 112,

Second Paragraph rejection as to claims 14-26, and the Specification and claims now comply with the statutory requirements.

In order to properly provide a complete response to the Office Action, please kindly note that the Office Action refers to "base claim 1". The applicants believe that this reference to "base claim 1" is inadvertent because base claim 1 has been withdrawn from consideration due to the earlier received Restriction Requirement, and consequently, the applicants have responded to this rejection with respect to base claim 14 from which claims 15 through 26 depend.

The Second 35 USC §112, Second Paragraph, Rejection

Claims 22-26 are rejected under 35 USC, second paragraph, as being indefinite for failing to particularly point out and distinctly claiming the subject matter which the applicant regards as the invention. The Applicants respectfully traverse this rejection.

In the Office Action, it is asserted that

in claims 22-23, applicants claimed "one or more passive ball and one or more passive pad". However, in light of applicants' disclosure, as it would be interpreted by one of ordinary skill in the art, above claim limitations do not set out and define the invention with a reasonable degree of precision and particularity. For purpose of examination, Examiner assumes the "passive port inherently comprises passive ball and passive pad".

The Specification as amended beginning from p. 21, line 7 through p. 22, line 4,

recites in part:

...A **connection** between a passive ball on one surface of a SDRAM module and a passive pad on another surface of the same SDRAM module, such as the X1 passive ball and X1 passive pad connection in Figure 19, is named "passive port", such as passive ports 1909 illustrated in Figure 19. (Emphasis added).

The above description should be read in conjunction with amended Figure 19, which shows that a passive port 1909 **connects** "a passive ball", such as passive ball X2, "on one surface of a SDRAM module", such as SDRAM module 0, and a passive pad", such as passive pad X3, "on another surface of the same DRAM module", such as SDRAM module 0. Thus, the recitation of "one or more passive ball and one or more passive pad" by the rejected claims particularly points out and distinctly claims the invention. By these amendments, the Second 35 USC § 112, Second Paragraph rejection has been rendered moot as to claims 14-26, and the Specification and claims now comply with the statutory requirements.

The Third 35 USC §112, Second Paragraph, Rejection

Claims 17-19 and 24-26 are rejected under 35 USC, second paragraph, as being indefinite for failing to particularly point out and distinctly claiming the subject matter which the applicant regards as the invention. The Applicants respectfully traverse this rejection.

In the Office Action with respect to claim 17, it is asserted that:

...applicants claimed "one or more pair of daisy chain input port and daisy chain output port ". However, in light of applicants' disclosure, as it would be interpreted by one of ordinary skill in the art, above claim limitations do not set out and define the invention with a reasonable degree of precision and particularity. For purpose of examination, Examiner assumes the "limitation associated with daisy chain will read as ladder like routing path connection".

The Specification at page 23, line 14 through page 24, line 20; Figure 21a; and the above claims have been amended so that these claims now particularly point out and distinctly claim the subject matter regarded as the invention by the Applicants. Support for the amendment made in the Specification is supported by the original claims, including claim 17, 18, 24, and 25. The Specification and these claims now comply with the statutory requirements, rendering the Third Section 112 Rejection moot.

The First 35 USC §102 Rejection

Claims 14-15 and 17-31 are rejected under 35 USC 102(b) as being anticipated by Kyoungoku et al. (Kyoungoku, US 5,995,379) of record.

The First 35 USC §102(b) Rejection

In the Office Action, claims 14-15 and 17-31 have been rejected under 35 USC 102(b) as being allegedly anticipated by Kyoungoku et al. (Kyoungoku, US 5,995,379) of record. This rejection is respectfully traversed.

In the Office Action it is asserted that with respect to claim 14, "Kyoungoku

shows a stacked module (Fig. 5) comprising: one or more active port (B1) for carrying one or more active signal (Fig. 5); and one or more passive port (L1, L4) for passing through the one or more active signals (Fig. 5).

Independent Claim 14

Amended claim 14, recites in part:

a first serial chain route that includes at least one serial chain connection, the serial chain connection including: a serial chain circuit, a serial chain input, and serial chain output; said serial chain input coupled to said serial chain output through said serial chain circuit;

a second serial chain route and a control circuit for enabling a routing path that connects the first serial chain route with the second serial chain route within an end module; and

said control circuit is disposed to enable the routing path in response to a control input signal received from another module from the plurality of modules when the end module is coupled to the another module.

Kyougoku fails to teach or disclose the above claim limitations, and hence, claim 14 is allowable.

Dependent Claim 15

In the Office Action it is also asserted that with respect to claim 15, "Kyougoku shows the one or more passive port forms a ladder like routing path (L1-L6)."

Applicants respectfully traverse this rejection.

Claim 15 depends on allowable base claim 14. Thus, claim 15 is also allowable for the same reasons stated for base claim 14 since an allowable base claim renders its dependent claim(s) also allowable.

Dependent Claims 17-19

Dependent claims 17-19 have been cancelled without prejudice. In addition, claim 14, recites additional claim limitations that are not shown or disclosed by the arguments presented in the Office Action with respect to claims 17-19. For example, claim 14, recites a “control circuit disposed to enable the routing path in response to a control input signal”, which is not shown in Kyougoku. Consequently, claim 14 is also allowable for this additional reason.

Dependent Claims 20-21

Claims 20-21 ultimately depend on allowable base claim 14. Thus, claims 20-21 are also allowable for the same reasons stated for base claim 14 above since an allowable base claim renders its dependent claims also allowable.

Dependent Claims 22-23

Claims 22-23 ultimately depend on allowable base claim 14. Thus, claims 22-23 are also allowable for the same reasons stated for base claim 14 above since an allowable base claim renders its dependent claims also allowable.

Dependent Claims 24-26

Claims 24-26 ultimately depend on allowable base claim 14. Thus, claims 24-26 are also allowable for the same reasons stated for base claim 14 above since an allowable base claim renders its dependent claims also allowable.

In addition, claim 25 recites in part, “wherein the routing path includes one or more tri-state driver controlled by the control circuit”, which is not taught or disclosed in Kyoungoku. For this additional, reason claim 25 is allowable.

In addition, claim 26 recites one or more additional claim limitations that are not disclosed or taught in Kyoungoku. For example, claim 26 recites in part, “a main board coupled to another stacked module having a third serial chain route”. In another example, claim 26 recites in part, “another control circuit for enabling another routing path that connects the second serial chain route with the third serial chain route”. Consequently, since Kyoungoku does not disclose or teach one or both of these additional claim limitations, claim 26 is allowable for these additional reasons.

Dependent Claims 27-31

Claims 27-31 ultimately depend on allowable base claim 14. Thus, claims 27-31 are also allowable for the same reasons stated for base claim 14 above since an allowable base claim renders its dependent claims also allowable.

The Second 35 USC §102(b) Rejection

In the Office Action, claims 14-15 have been rejected under 35 USC 102(b) as being allegedly anticipated by Beausoleil et al. (Beausoleil, US 6,469, 375). This rejection is respectfully traversed.

In the Office Action it is asserted that with respect to claim 14, “Beausoleil shows a plurality of modules each comprising comprising: one or more active port 18 (see attached Fig. 1) for carrying one or more active signal (Fig. 1); and one or

more passive port (see attached Fig. 1) for passing through the one or more active signals.

Independent Claim 14

Amended claim 14, recites in part:

a first serial chain route that includes at least one serial chain connection, the serial chain connection including: a serial chain circuit, a serial chain input, and serial chain output; said serial chain input coupled to said serial chain output through said serial chain circuit;

a second serial chain route and a control circuit for enabling a routing path that connects the first serial chain route with the second serial chain route within an end module; and

said control circuit is disposed to enable the routing path in response to a control input signal received from another module from the plurality of modules when the end module is coupled to the another module.

Beausoleil fails to teach or disclose the above claim limitations, and hence, claim 14 is allowable.

Dependent Claim 15

In the Office Action it is also asserted that with respect to claim 15, "Beausoleil shows wherein the one or more passive port forms a ladder like routing path (see attached Fig. 1)." Applicants respectfully traverse this rejection.

Claim 15 depends on allowable base claim 14. Thus, claim 15 is also allowable

for the same reasons stated for base claim 14 since an allowable base claim renders its dependent claim(s) also allowable.

The First 35 USC §103 Rejection

The Examiner rejects claim 16 by asserting this claim is allegedly unpatentable over Beausoleil et al. (Beausoleil, US 6,469,375). This rejection is respectfully traversed.

In the Office Action it is asserted that with respect to claim 16:

Beausoleil teaches one or more of the plurality of module 14 (Fig. 1) and stacked one or more passive port (see attached Fig. 1), but fails teach module is rotated before being stacked and the one or more passive port forms a rotational routing path. However, Applicants have no support data, which convinces that the particular claimed configuration is significant or is anything more than one of numerous configurations a person of ordinary skill in the art would find obvious for the purpose of providing mating surfaces. In re Dailey 149 USPQ 47,50 (CCPA 1966). See also Glue Co. v. Upton 97 US 3,24 (USSC 1878)., see MPEP 2144.04.

Claim 16 depends from claim 1, which recites limitations not taught or suggested by Beausoleil. For example, claim 1 recites in part:

a first serial chain route that includes at least one serial chain connection, the serial chain connection including: a serial chain circuit, a

serial chain input, and serial chain output; said serial chain input coupled to said serial chain output through said serial chain circuit;

a second serial chain route and a control circuit for enabling a routing path that connects the first serial chain route with the second serial chain route within an end module; and

said control circuit is disposed to enable the routing path in response to a control input signal received from another module from the plurality of modules when the end module is coupled to the another module.

Consequently, claim 1 is not rendered obvious by Beausoleil for the above reason, rendering this Section 103 Rejection inapplicable here since an allowable base claim renders its dependent claims also allowable.

In addition, Applicants agree with the Examiner that Beausoleil “fails to teach” module is rotated before being stacked and the one or more passive port forms a rotational routing path.” Applicants, however, kindly disagree that the limitation in claim 16 that recites “one or more passive port forms a rotational routing path” is without significance or “is anything more than one of a numerous configurations a person of ordinary skill in the art would find obvious”. The cases cited in the Office Action do not apply here because In re Daily or Glue Co. v. Upton, simply hold that structures that are a “mere matter of choice”, or are not a new means “by which a desirable change in the form of a common article of commerce is obtained”, respectively. Indeed, if the configuration recited by this claim limitation is so “numerous” so that it would be found obvious, a suitable prior art reference would be

readily available. The lack of such a reference precludes the Applicants from adequately responding to the Section 103 rejection on the merits.

Further, pursuant to MPEP 2144.04, the significance of the limitations recited in claim 16 is presented in by the Specification (Substitute Specification submitted on 13 February 2009) at p. 21, line 18 through p. 22, line 21; and Figures 20a and 20b.

Consequently, under any of the above additional reasons, the Section 103 rejection does not apply here, rendering claim 16 allowable.

Petition for Extension of Time

A Petition for an Extension of Time is submitted in a separate paper and filed on even date herewith.

Request for Allowance

It is believed that this Amendment places this Application into condition for allowance. Early favorable consideration of this Amendment is earnestly solicited.

Respectfully submitted,

Dated: 13 November 2009

/Stephen R. Uriarte/

Stephen R. Uriarte, Reg. No. 40,854

Uriarte Law
2021 The Alameda,
Suite 225
San Jose, CA 95126
Tel: 408.244.2164 x111
Fax: 408.716.2651
Email: mail@uriartelaw.com

REPLACEMENT SHEET

Multiple Chip Module and Package Stacking for Storage Devices

Inventors: Rey H. Bruce et al. Application #: 11/322,442 Docket#: 6005.011

24/28

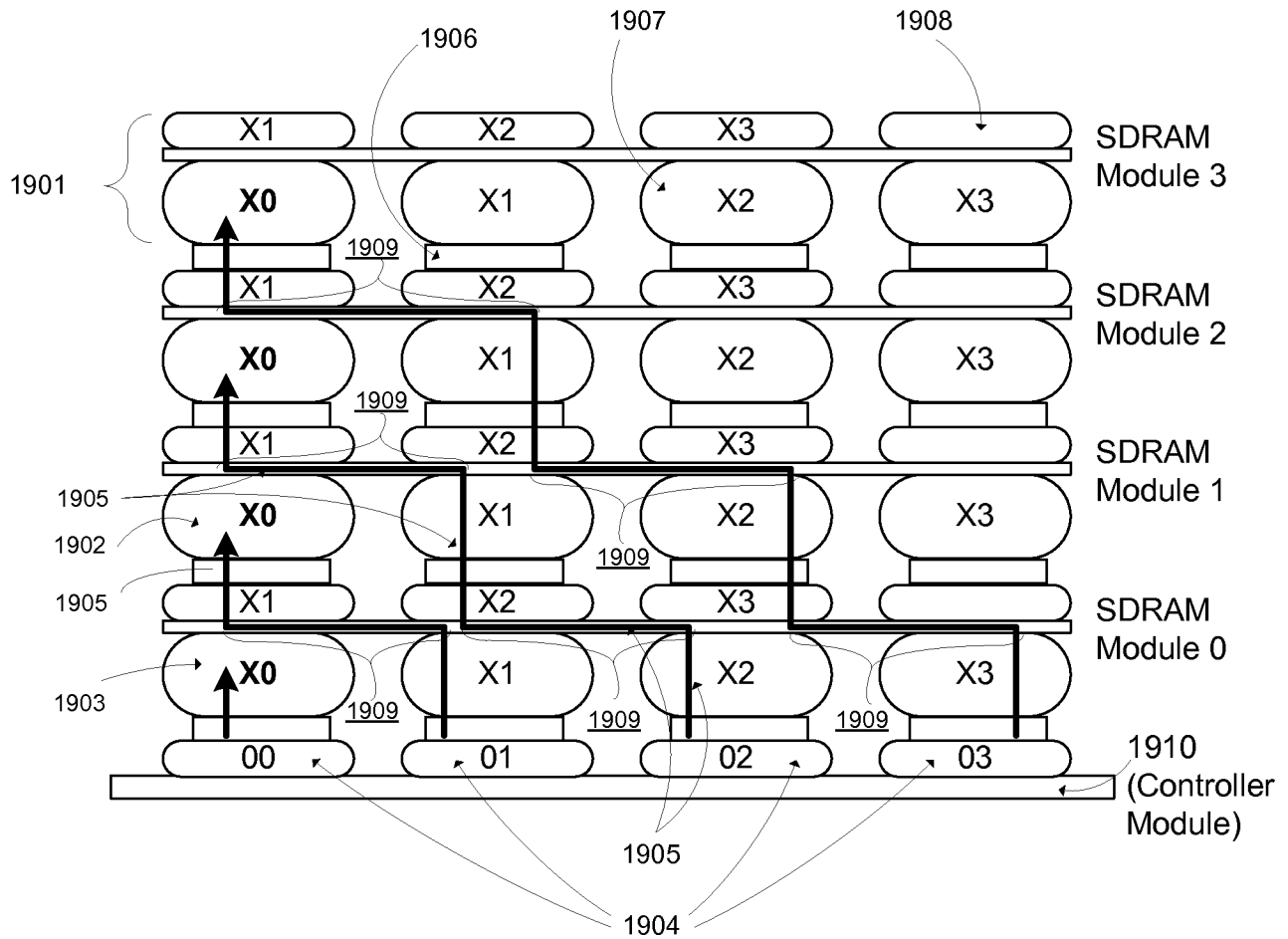


Figure 19

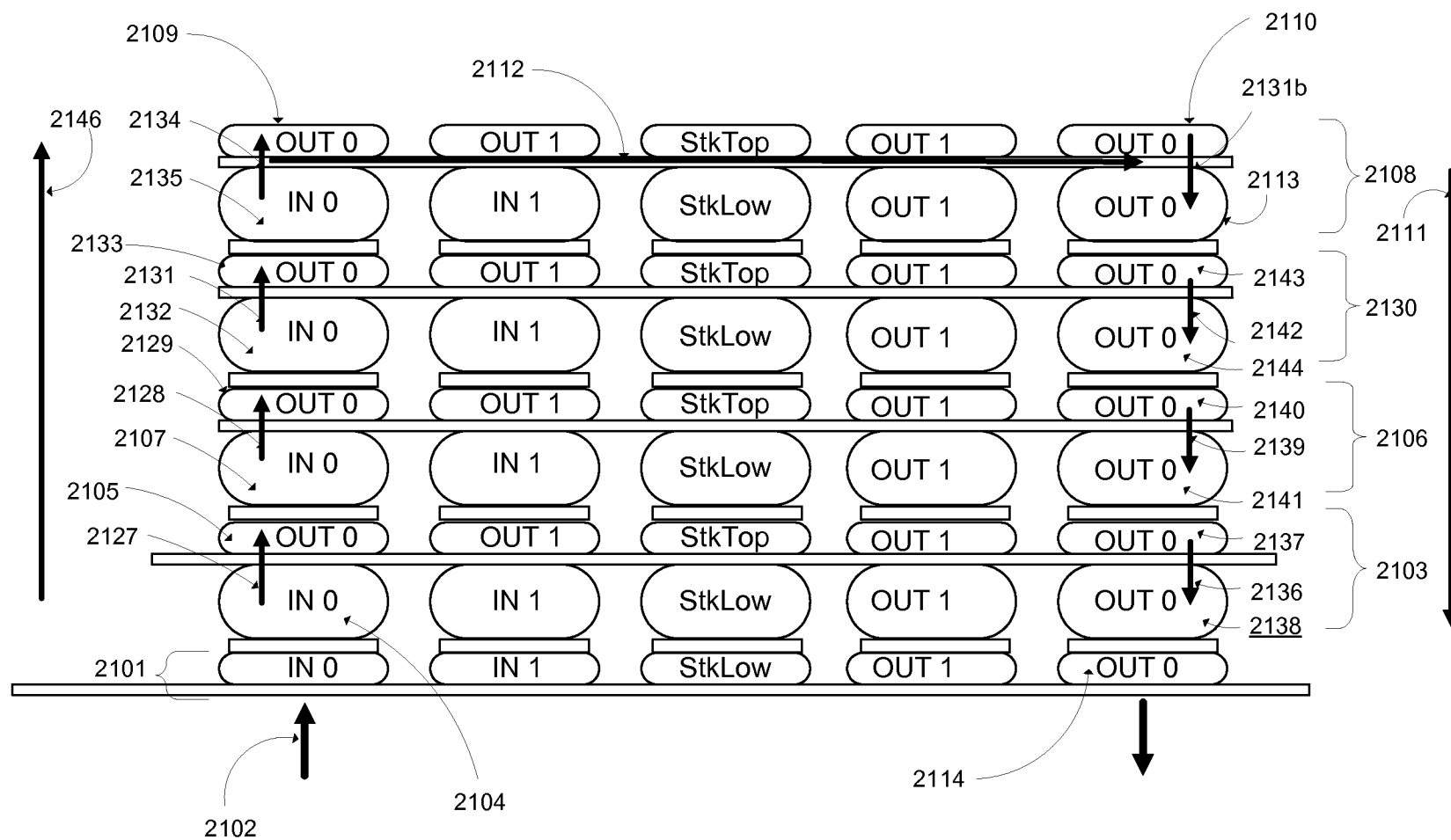


Figure 21a

REPLACEMENT SHEET
 Multiple Chip Module and Package Stacking for Storage Devices
 Inventors: Rey H. Bruce et al. Application #: 11/322,442 Docket#: 6005.011
 27/28

Inventors: Rey H. Bruce et al. Application #: 11/322,442 Docket#: 6005.011
28/28

Figure 21b

EXHIBIT G

IEEE 100
The Authoritative Dictionary of
IEEE Standards Terms

Seventh Edition



Published by
Standards Information Network
IEEE Press

Trademarks and disclaimers

IEEE believes the information in this publication is accurate as of its publication date; such information is subject to change without notice. IEEE is not responsible for any inadvertent errors.

Other tradenames and trademarks in this document are those of their respective owners.

*The Institute of Electrical and Electronics Engineering, Inc.
3 Park Avenue, New York, NY, 10016-5997, USA*

Copyright © 2000 by the Institute of Electrical and Electronics Engineers, Inc. All rights reserved. Published December 2000. Printed in the United States of America.

No part of this publication may be reproduced in any form, in an electronic retrieval system or otherwise, without the prior written permission of the publisher.

To order IEEE Press publications, call 1-800-678-IEEE.

Print: ISBN 0-7381-2601-2

SP1122

See other standards and standards-related product listings at: <http://standards.ieee.org/>

The publisher believes that the information and guidance given in this work serve as an enhancement to users, all parties must rely upon their own skill and judgement when making use of it. The publisher does not assume any liability to anyone for any loss or damage caused by any error or omission in the work, whether such error or omission is the result of negligence or any other cause. Any and all such liability is disclaimed.

This work is published with the understanding that the IEEE is supplying information through this publication, not attempting to render engineering or other professional services. If such services are required, the assistance of an appropriate professional should be sought. The IEEE is not responsible for the statements and opinions advanced in this publication.

Library of Congress Cataloging-in-Publication Data

IEEE 100 : the authoritative dictionary of IEEE standards terms.—7th ed.
p. cm.

ISBN 0-7381-2601-2 (paperback : alk. paper)

1. Electric engineering—Dictionaries. 2. Electronics—Dictionaries. 3. Computer engineering—Dictionaries. 4. Electric engineering—Acronyms. 5. Electronics—Acronyms. 6. Computer engineering—Acronyms. I. Institute of Electrical and Electronics Engineers.

TK9 .I28 2000
621.3'03—dc21

00-050601

medium interface connector

medium interface connector (MIC) A connector interface at which signal transmit and receive characteristics are specified for attaching stations and concentrators. One class of MICs is the connection between the attaching stations and the lobe cabling. A second set is the attachment interface between the concentrator and its lobes. A third set is the interface between the concentrator and the trunk cabling. Two types of connectors are specified: one for connecting to STP media and one for connecting to UTP media. (C/LM) 8802-5-1998

medium ion (dc electric-field strength and ion-related quantities) Ion comprised of several molecules or molecular clusters bound together by charge that is larger and less mobile than a small ion due to more mass or a greater number of molecular clusters. Typical radius is in the range of 10^{-9} m to 2×10^{-8} m. Mobility is in the range of 10^{-7} m²/Vs to 10^{-5} m²/Vs. (T&D/PE) 539-1990, 1227-1990r

medium noise (sound recording and reproducing system) The noise that can be specifically ascribed to the medium. *See also:* noise. 191-1953w

medium-pulse-repetition-frequency waveform A pulsed-radar waveform whose pulse-repetition frequency (PRF) is such that targets of interest are ambiguous with respect to both range and Doppler shift. *See also:* high-pulse-repetition-frequency waveform; low-pulse-repetition-frequency waveform. (AES) 686-1997

medium scale integration (MSI) (A) Pertaining to an integrated circuit containing between 100 and 500 transistors in its design. *Contrast:* large scale integration; very large scale integration; small scale integration; ultra-large scale integration. (B) Pertaining to an integrated circuit containing between 10 and 100 elements. (C) 610.10-1994

medium voltage (1) (cable systems in power generating stations) 601 to 15 000 V. (PE/EDPG) 422-1977

(2) (system voltage ratings) A class of nominal system voltages greater than 1000 V and less than 100 000 V. *See also:* low voltage; nominal system voltage; high voltage. (IA/PSE/APP) 241-1990r, [80]

medium-voltage aluminum-sheathed power cable (aluminum sheaths for power cables) Cable used in an electric system having a maximum phase-to-phase rms ac voltage above 1 000 V to 72 500 V, the cable having an aluminum sheath as a major component in its construction. (PE/IC) 635-1989r

medium-voltage power cable (1) (cable systems in power generating stations) Cables designed to supply power to utilization devices of the plant auxiliary system, operated at 601 to 15 000 V. (PE/EDPG) 422-1977

(2) Cable designed to supply power to utilization devices of the plant auxiliary system, operated at 5000–46 000 V in sizes ranging from 8 AWG (8.37 mm²) to 2000 kcmil (1010.0 mm²). (PE/IC) 1185-1994

medium-voltage system (electric power for industrial and commercial systems only) An electric system having a maximum root-mean-square alternating-current voltage above 1000 volts to 72 500 volts. *See also:* voltage classes. (IA/PSE) 570-1975w

meet *See:* AND.

meg Colloquial reference for megabyte. (C) 610.10-1994w

mega (M) (A) (mathematics of computing) A prefix indicating one million (10⁶). (B) (mathematics of computing) In statements involving size of computer storage, a prefix indicating 2²⁰, or 1 048 576. (C) 1084-1986

megabyte Either 1 000 000 bytes or 2²⁰ bytes. *Notes:* 1. The user of these terms shall specify the applicable usage. If the usage is 2¹⁰ or 1024 bytes, or multiples thereof, then note 2 below shall also be included with the definition. 2. As used in IEEE Std 610.10-1994, the terms kilobyte (kB) means 2¹⁰ or 1024 bytes, megabyte (MB) means 1024 kilobytes, and gigabyte (GB) means 1024 megabytes. *See also:* gigabyte. (C) 610.10-1994w

megacycle One million cycles. (C) 610.10-1994w

memory allocation and protection

megahertz (MHz) (1) A unit of frequency equal to 1 000 000 cycles per second. (LM/C) 802.7-1989r

(2) A unit of frequency equal to 1 000 000 Hz, that is, 10⁶ Hz. (C) 610.7-1995

Meissner oscillator An oscillator that includes an isolated tank circuit inductively coupled to the input and output circuits of an amplifying device to obtain the proper feedback and frequency. *See also:* oscillatory circuit. (AP/ANT) 145-1983s

mel A unit of pitch. By definition, a simple tone of frequency 1000 hertz, 40 decibels above a listener's threshold, produces a pitch of 1000 mels. *Note:* The pitch of any sound that is judged by the listener to be n times that of the 1-mel tone is n mels. (SP) [32]

melting channel The restricted portion of the charge in a submerged resistor or horizontal-ring induction furnace in which the induced currents are concentrated to effect high energy absorption and melting of the charge. *See also:* induction heating. (IA) 54-1955w, 169-1955w

melting-speed ratio (1) The ratio between between 0.1 s and 300 s or 600 s minimum melting currents, whichever is specified, which designates the relative speed of the fuse link. (SWG/PE) C37.40-1993

(2) (of a fuse) A ratio of the current magnitudes required to melt the current-responsive element at two specified melting times. *Notes:* 1. Specification of the current wave shape is required for time less than one-tenth of a second. 2. The lower melting time in present use is 0.1 s, and the higher minimum melting current times are 100 a for low-voltage fuses and 300 s or 600 s, whichever specified, for high-voltage fuses. (SWG/PE) C37.100-1992

melting time (1) (protection and coordination of industrial and commercial power systems) The time required to melt the current-responsive element on a specified overcurrent. Where the fuse is current limiting in less than half-cycle, the melting time may be approximately half or less of the clearing time. (IA/PSP) 242-1986r

(2) (of a fuse) The time required for overcurrent to sever the current-responsive element. (SWG/PE/SWG-OLD) C37.100-1992, C37.40-1993, C37.40b-1996

member In data management, a subunit contained in a partitioned data set. (C) 610.5-1990w

membrane keyboard A type of keyboard in which the keys are not raised, rather it is composed of a semi-flexible plastic sheet with a conductive surface below. *Synonym:* pressure-sensitive keyboard. (C) 610.10-1994w

membrane potential The potential difference, of whatever origin, between the two sides of a membrane. *See also:* electrophysiology. (EMB) [47]

memory (1) All of the addressable storage in a processing unit and other internal storage that is used to execute instructions. *See also:* main storage. (C) 610.10-1994w

(2) *See also:* storage medium; storage.

memory action (of a relay) A method of retaining an effect of an input after the input ceases or is greatly reduced, so that this input can still be used in producing the typical response of the relay. *Note:* For example, memory action in a high-speed directional relay permits correct response for a brief period after the source of voltage input necessary to such response is short-circuited. (PE) C37.100-1992

memory address An address of a particular storage location in memory. (C) 610.10-1994w

memory address register A register containing the address of the memory location to be accessed. (C) 610.10-1994w

memory agent A module that uses split transactions to assume all the rights and responsibilities of some number of remote memory modules. (C/BA) 896.4-1993w

memory allocation and protection (A) To allocate physical sections of memory into logical partitions with read/write protection provided within each partition. **(B)** Pertaining to the hardware components that perform the allocation as in (A). (C) 610.10-1994

memory allocation and protection

of frequency equal to 1 000 000 (LM/C) 802.7-1989r
al to 1 000 000 Hz, that is, 10⁶
(C) 610.7-1995

tor that includes an isolated tank
to the input and output circuits of
in the proper feedback and fre-
circuit. (AP/ANT) 145-1983s

tion, a simple tone of frequency
e a listener's threshold, produces
The pitch of any sound that is
n times that of the 1-mel tone is
(SP) [32]

l portion of the charge in a sub-
-ring induction furnace in which
icentrated to effect high energy
he charge. *See also:* induction
(IA) 54-1955w, 169-1955w

atio between between 0.1 s and
ting currents, whichever is spec-
lative speed of the fuse link.
(SWG/PE) C37.40-1993

: current magnitudes required to
element at two specified melting
n of the current wave shape is
e-tenth of a second. 2. The lower
s 0.1 s, and the higher minimum
a for low-voltage fuses and 300
ed, for high-voltage fuses.
(SWG/PE) C37.100-1992

**and coordination of industrial
lems** The time required to melt
ent on a specified overcurrent.
iting in less than half-cycle, the
nately half or less of the clearing
(IA/PSP) 242-1986r
ired for overcurrent to sever the

b) C37.100-1992, C37.40-1993,
C37.40b-1996

a subunit contained in a parti-
(C) 610.5-1990w

f keyboard in which the keys are
osed of a semi-flexible plastic
ace below. *Synonym:* pressure-
(C) 610.10-1994w

tial difference, of whatever or-
f a membrane. *See also:* electro-
(EMB) [47]

ible storage in a processing unit
t is used to execute instructions.
(C) 610.10-1994w

n; storage.

method of retaining an effect of
s or is greatly reduced, so that
producing the typical response
ple, memory action in a high-
its correct response for a brief
oltage input necessary to such
(PE) C37.100-1992

f a particular storage location in
(C) 610.10-1994w

gister containing the address of
cessed. (C) 610.10-1994w
ises split transactions to assume
ties of some number of remote
(C/BA) 896.4-1993w

ction (A) To allocate physical
al partitions with read/write pro-
partition. **(B)** Pertaining to the
rform the allocation as in (A).
(C) 610.10-1994

memory array

memory array (1) A matrix of memory locations arranged in
a rectangular geometric pattern on an integrated circuit.
(C) 610.10-1994w
(ED) 1005-1998

(2) *See also:* array.

memory bank *See:* bank.

memory board A circuit board that provides random-access
memory to a system. (C) 610.10-1994w

memory boundary The last address of an aligned data block.
The maximum data block size that can be transferred by an
IUT Master is the product of data width and data length.
(C/BA) 896.4-1993w

memory buffer register A register in which a word is stored
as it is read from memory or as it is written to memory. *Syn-*
onym: memory data register. (C) 610.10-1994w

memory bus A bus connecting memory to the devices which
can access it, including the processor and peripheral devices.
(C) 610.10-1994w

memory capacity (1) The maximum number of bits that a mem-
ory is capable of storing. (ED) 641-1987w

(2) **(software)** The maximum number of items that can be
held in a given computer memory; usually measured in words
or bytes. (C) 610.12-1990

(3) *See also:* capacity. (ED) 1005-1998

(4) **(electronic computation)** *See also:* channel capacity;
storage capacity.

memory cell (1) The smallest subdivision of a memory into
which a unit of data has been or can be entered, in which it
is or can be stored, and from which it can be retrieved.
(ED) 641-1987w

(2) The combination of one or more single or merged trans-
istors formed to provide a means of accessing, changing, and
storing data. (ED) 1005-1998

memory compaction (A) A storage allocation technique in
which the contents of all allocated storage areas are moved
to the beginning of the storage space and the remaining stor-
age blocks are combined into a single block. *Synonym:* gar-
bage collection. **(B)** A storage allocation technique in which
contiguous blocks of nonallocated storage are combined to
form single blocks. (C) 610.12-1990

memory core *See:* magnetic core.

**memory cycle (1) (test, measurement, and diagnostic equip-
ment)** The time required to read information from memory
and replace it. (ML) [2]

(2) A single complete access (read or write) of memory.
(C) 610.10-1994w

memory data register *See:* memory buffer register.

memory device A device that contains only memory and im-
plements configuration registers. (C/MM) 1155-1992

memory dump A display of the contents of all or part of a
computer's internal storage, usually in binary, octal, or hex-
adecimal form. *See also:* static dump; selective dump; snap-
shot dump; dynamic dump; change dump. (C) 610.12-1990

memory image A series of bits that can be stored within a
contiguous portion of transponder memory and that may be
passed as a parameter within commands initiated by the road-
side equipment (RSE). (SCC32) 1455-1999

memory integrated circuit An integrated circuit consisting of
memory cells and usually including associated circuits such
as signal amplification and address selection. (ED) 1005-1998

memory location A subdivision of a memory, including one or
several memory cells, that is the smallest part of the memory
that can be addressed. *Note:* The content of a memory loca-
tion is usually called a bit, a byte, or a word, as appropriate.
(ED) 1005-1998

memory management unit (MMU) A device that performs
address translation between a CPU's virtual addresses and the
physical addresses of some bus; typically, the bus represented
by the root node. (C/BA) 1275-1994

memory map (1) A diagram that shows where programs and
data are stored in a computer's memory. (C) 610.12-1990

685

mercury-arc rectifier

(2) A list of all the current addresses in a computer. *Note:*
This may indicate what is currently allocated, who is using it
and where it is located. *Synonym:* memory map list.
(C) 610.10-1994w

memory map list *See:* memory map.

memory mapping (A) The manner in which an address is trans-
lated into a physical address of a storage location. *See also:*
biasing; segmenting; paging. **(B)** The process of translating
addresses as in definition (A). (C) 610.10-1994

memory-mode agent An agent that communicates with others
by using memory and/or I/O space on the parallel system bus.
(C/MM) 1296-1987s

memory-mode system A system in which the agents commu-
nicate with one another with data structures in memory and/
or I/O space. (C/MM) 1296-1987s

memory object (1) Either a file or shared memory object. When
used in conjunction with *mmap()*, a memory object will ap-
pear in the address space of the calling process.
(C/PA) 9945-1-1996

(2) Either a file or shared memory object. When used in con-
junction with *Map.Memory*, *Open.And.Map.Shared.Memory*,
or *Open.Or.Create.And.Map.Shared.Memory*, a memory
object will appear in the address space of the calling process.
(C) 1003.5-1999

memory organization The arrangement of memory cells, either
by geometrical arrangement in rows and columns or by or-
ganization of the data to be stored. (ED) 1005-1998, 641-1987w

memory page A segment of transponder memory that is as-
signed a unique location by which it may be referenced.
(SCC32) 1455-1999

memory relay (A) A relay having two or more coils, each of
which may operate independent sets of contacts, and another
set of contacts that remain in a position determined by the
coil last energized. **(B)** Sometimes erroneously used for po-
larized relay. *See also:* relay. (EEC/REE) [87]

memory-resident Managed by the implementation in such a
way as to provide an upper bound on memory access times.
(C/PA) 9945-1-1996, 1003.5-1999

memory space The address space used for accessing physical
memory devices for storage and retrieval of code and data.
(C/MM) 1296-1987s

memory window The difference in threshold voltage between
the low- and high-conductance logic states of a memory cell.
(ED) 641-1987w

MENTOR A block-structured language used widely in com-
puter-aided instruction; characterized by its ability to model
a student's knowledge. (C) 610.13-1993w

menu (1) A list of options available for selection by the user of
a computer system. *Synonyms:* display menu; help menu;
menu selection. (C) 610.2-1987, 610.6-1991w

(2) A rectangular visual user interface control containing a
group of controls used to select an action from a group of
choices. (C) 1295-1993w

menu bar A visual user interface control that is the bounded
area near the top of a window, below the title bar, and above
the rest of the window that contains cascade buttons that pro-
vide access to other menus. (C) 1295-1993w

menu by-pass In a menu-driven system, a feature that permits
advanced users to perform functions in a command-driven
mode without selecting options from the menus. (C) 610.12-1990

menu-driven Pertaining to a system or mode of operation in
which the user directs the system through menu selections.
Contrast: command-driven. *See also:* menu by-pass.
(C) 610.12-1990

menu selection (A) The process of choosing an item from a
menu. **(B)** The item chosen from a menu. (C) 610.2-1987

mercury-arc converter, pool-cathode *See:* pool-cathode mer-
cury-arc converter; oscillatory circuit.

mercury-arc rectifier A gas-filled rectifier tube in which the
gas is mercury vapor. *See also:* rectification. (ED) [45], [84]

stop list.

or wading pool A pool with a maximum ft and a maximum wall height of 3 ft and is at it may be readily disassembled for storage to its original integrity.

(NESC/NEC) [86]
electronic computation The act of storing (electronic computation) Any device in which data can be stored, sometimes called a memory **tronic computation** In a computer, a section of a memory or store (British). *Notes:* 1. The storing information may be electrostatic, magnetic, acoustic, optical, chemical, mechanical, etc., in nature. 2. Pertaining to data that can be entered, in which it can be held, and can be retrieved at a later time. *See also:* (MIL/C) [2], [85], [20]

ement In a computer, one or more bytes of data. (C) 610.5-1990w
tion of data in a storage device. (B) The data into a storage device. (C) A storage medium in which data can be retained. (C) 610.10-1994

access.

(1) (computers) The assignment of sequential instructions to specified blocks of storage. (C) [20], [85]

element of computer resource allocation, assigning storage areas to specific jobs and procedures, such as transfer of data between storage, to support the assignments made, buffer; contiguous allocation; cyclic search; relay; memory compaction. (C) 610.12-1990

storage tubes An assembly of electrodes that contains the target together with electrical control of the storage process, those that regulate, and other members used for structural storage tube. (ED) 158-1962w

battery comprised of one or more rechargeable acid, nickel-cadmium, or other rechargeable types. (NESC/NEC) [86]

See: data breakpoint.

A low leakage capacitor on which a data is stored. (C) 610.10-1994w

The amount of data that can be contained in a storage device. *Notes:* 1. The units of capacity are bits, etc. For example, capacity might be "32 cimal digits," "16 384 words with 10 alters each." 2. When comparisons are made using different character sets and word convenient to express the capacity in equivalents the number obtained by taking the logarithm of the number of usable distinguishable storage can exist. 3. The storage (or memory) computer usually refers only to the internal storage. (C) 162-1963w

e maximum number of items that can be stored in a storage device; usually measured in words or bytes. (C) 610.12-1990

f data that can be contained in a storage device in binary characters, bytes, words, or other units. (C) 610.10-1994w

f data that can be contained in a storage device. (ED) 1005-1998

tronic energy) (secondary cell or accumulator) cell for the generation of electric energy in which, after being discharged, may be restored to its original condition by an electric current flowing in a direction opposite to the flow of current when the cell is discharged. (EEC/PE) [119]

storage channel

(2) (A) One or more storage elements considered as a unit.
(B) The smallest subdivision of storage into which a unit of data can be placed, retained, and from which the unit can be retrieved. *Synonym:* data cell. *See also:* binary cell; magnetic cell. (C) 610.10-1994

(3) An elementary unit of storage (e.g., a binary cell or a decimal cell). (ED/C) 1005-1998, [85], [20]

storage channel A channel that can be used to access a storage device. (C) 610.10-1994w

storage device (1) A device in which data can be stored and from which it can be copied at a later time. The means of storing data may be chemical, electrical, mechanical, etc. *See also:* storage. (C) 162-1963w

(2) A device into which data can be placed, in which they can be retained, and from which they can be retrieved. *See also:* store. (C) 610.10-1994w

storage display *See:* storage tube display device.

storage efficiency The degree to which a system or component performs its designated functions with minimum consumption of available storage. *See also:* execution efficiency. (C) 610.12-1990

storage element (1) (storage tubes) An area of a storage surface that retains information distinguishable from that of adjacent areas. *Note:* The storage element may be a portion of a continuous storage surface or a discrete area such as a dielectric island. *See also:* storage tube. (ED) 158-1962w, 161-1971w

(2) The basic unit of a storage device, such as a sector, or a track. (C) 610.10-1994w

storage-element equilibrium voltage (storage tubes) A limiting voltage toward which a storage element charges under the action of primary electron bombardment and secondary emission. At equilibrium voltage the escape ratio is unity. *Note:* Cathode equilibrium voltage, second-crossover equilibrium voltage, and gradient-established equilibrium voltage are typical examples. *See also:* charge-storage tube. (ED) 158-1962w

storage-element equilibrium voltage, cathode (storage tubes) The storage element equilibrium voltage near cathode voltage and below first-crossover voltage. *See also:* charge-storage tube. (ED) 158-1962w

storage-element equilibrium voltage, collector *See:* charge-storage tube.

storage-element equilibrium voltage, gradient established (storage tubes) The storage-element equilibrium voltage, between first- and second-crossover voltages, at which the escape ratio is unity. *See also:* charge-storage tube. (ED) 158-1962w

storage-element equilibrium voltage, second-crossover (storage tubes) The storage-element equilibrium voltage at the second-crossover voltage. *See also:* charge-storage tube. (ED) 158-1962w

storage error An error in which the data retrieved from storage is different from that which was originally stored in that location. *See also:* soft error; hard error; transient error. (C) 610.10-1994w

storageeid (microprocessor operating systems parameter types) An identifier for a block of data. The identifier is not guaranteed to be valid outside the allocating process and should not be passed between processes. (C/MM) 855-1985s

storage integrator In an analog computer, a device used to store a voltage in the hold condition for future use. *See also:* electronic analog computer. (C) 610.10-1994w, 165-1977w

storage life (accelerometer) (gyros) (inertial sensors) The nonoperating time interval under specified conditions, after which a device will still exhibit a specified operating life and performance. *See also:* operating life. (AES/GYAC) 528-1994

storage light A light found on a storage device indicating that a parity check error has occurred on a character as it was read into storage. (C) 610.10-1994w

storage light-amplifier *See:* image-storage panel.

storage location (1) An area in a storage device that can be explicitly and uniquely specified by means of an address. (C) 610.5-1990w

(2) A location in a storage device that is uniquely specified by means of an address. (C) 610.10-1994w

storage medium Any device or recording medium into which data can be stored and held until some later time, and from which the entire original data can be obtained. (IA) [61]

storage protection (computers) An arrangement for preventing access to storage for either reading or writing, or both. (C) [20]

storage rate The frequency with which sampled signals are recorded in crashworthy nonvolatile memory. The event recorder may store any signal less often than it samples. (VT) 1482.1-1999

storage, reservoir *See:* reservoir storage.

storage schema In a CODASYL database, statements expressed in data storage definition language that describe storage areas, stored records, and any associated indices and access paths supporting the records and sets defined by a given schema. *See also:* CODASYL database. (C) 610.5-1990w

storage stack *See:* stack.

storage station (power operations) A hydroelectric generating station associated with a water storage reservoir.

(PE/PSE) 858-1987s, 346-1973w

storage structure (A) The manner in which data structures are represented in storage. **(B)** The configuration of a database resident on computer storage devices after mapping the data elements of the logical structure of the database onto their respective physical counterparts. *Note:* The relationships and associations that provide the physical means for accessing the information stored in the database are preserved. (C) 610.5-1990

storage surface (storage tubes) The surface upon which information is stored. *See also:* storage tube. (ED) 158-1962w

storage temperature (1) (power supply) The range of environmental temperatures in which a power supply can be safely stored (for example, -40°C to $+85^{\circ}\text{C}$). (AES/IA) [41], [12]

(2) (light-emitting diodes) The temperature at which the device, without any power applied, is stored. (ED) [127]

storage temperature range The range of temperatures over which the Hall generators may be stored without any voltage applied, or without exceeding a specified change in performance. (MAG) 296-1969w

storage time *See:* decay time; maximum retention time.

storage tube An electron tube into which information can be introduced and read at a later time. *Note:* The output may be an electric signal and/or a visible image corresponding to the stored information. (ED) 161-1971w, 158-1962w

storage tube display device A type of cathode ray tube display device that retains a display image on its surface in the form of a pattern of electric charges. *Synonyms:* storage display; display storage tube; direct-view storage tube. *Contrast:* refresh display device. (C) 610.10-1994w

storage unit The length of an addressable element of storage in the machine, measured in bits. (Every storage element has the same size.). *Note:* The storage unit is very likely to be one byte, but this is not a requirement. For example, it might be 32 or 64 bits. (C) 1003.5-1999

store (A) A device into which data can be placed, in which they can be retained, and from which they can be retrieved. *Note:* This term is the equivalent of the term storage in British (U.K.) usage. **(B)** To place data into a device as in definition (A). **(C)** To retain data in a device as in definition (A). (C) 162-1963, 610.10-1994

(2) (A) To place or retain data in a storage device. **(B) (software) (data management)** To copy computer instructions or data from a register to internal storage or from internal storage to external storage. *Contrast:* retrieve; load. *See also:* move; fetch. (C) 610.12-1990, 610.5-1990

store-and-forward

1114

strain element

store-and-forward Pertaining to communications where a message is received completely before beginning transmission onto the next node. (C) 610.7-1995

store-and-forward buffer (local area networks) A first-in-first-out (FIFO) buffer in the network repeater that can provide temporary storage for an entire message packet prior to retransmission. The buffer acts as a shift register and must hold an entire, full-length packet. *See also:* elasticity buffer. (C) 8802-12-1998

store-and-forward switched network A switched network in which the store-and-forward principle is used to handle transmissions between the sender and the recipient. (C) 610.7-1995

store-and-forward switching A method of switching whereby messages are transferred directly or with interim storage, each in accordance with its own address. *See also:* packet switching. (LM/COM) 168-1956w

store-and-forward switching system (telephone switching systems) A switching system for the transfer of messages, each with its own address or addresses, in which the message can be stored for subsequent transmission. (COM) 312-1977w

stored-energy indicator An indicator that visibly shows that the stored-energy mechanism is in the charged or discharged position. (SWG/PE) C37.100-1992

stored-energy operation Operation by means of energy stored in the mechanism itself prior to the completion of the operation and sufficient to complete it under predetermined conditions. *Note:* This kind of operation may be subdivided according to: (1) how the energy is stored (spring, weight, etc.), (2) how the energy originates (manual, electric, etc.), and (3) how the energy is released (manual, electric, etc.). (SWG/PE/IA/PSP) C37.100-1992, 1015-1997

stored logic (telephone switching systems) Instructions in memory arranged to direct the performance of predetermined functions in response to readout. (COM) 312-1977w

stored paragraph *See:* boilerplate text.

stored program (telephone switching systems) A program in memory that a processor can execute. (COM) 312-1977w

stored-program computer (1) A digital computer that, under control of internally stored instructions, can synthesize, alter, and store instructions as though they were data and can subsequently execute these new instructions. (C) [20], [85] (2) A computer that is controlled by internally stored instructions that are treated as though they were data, and that can subsequently be executed. (C) 610.10-1994w

stored program control (telephone switching systems) A system control using stored logic. (COM) 312-1977w

stored record *See:* internal record.

stored-program switching system (telephone switching systems) An automatic switching system having stored program control. (COM) 312-1977w

storm guys Anchor guys, usually placed at right angles to direction of line, to provide strength to withstand transverse loading due to wind. *See also:* tower. (T&D/PE) [10]

storm loading The mechanical loading imposed upon the components of a pole line by the elements, that is, wind and/or ice, combined with the weight of the components of the line. *Note:* The United States has been divided into three loading districts, light, medium, and heavy, for which the amounts of wind and/or ice have been arbitrarily defined. *See also:* cable. (EEC/PE) [119]

STP (local area networks) A 150 Ω shielded balanced cable meeting the specifications in ISO/IEC 11801:1995. (C) 8802-12-1998

STR *See:* spanning tree route.

straggling, energy *See:* energy straggling.

straight air brake An arrangement of brakes whereby air is admitted from the main reservoir through a brake valve to the

straight air pipe to the brake cylinders in the operating unit. *Note:* In most rail transit vehicle applications, an electro-pneumatic overlay is utilized to assist in the straight air brake command transmission. (VT) 1475-1999

straight air pipe A method of transmitting a pneumatic command from the active cab to the straight air brake equipment on each vehicle in the operating unit. (VT) 1475-1999

straightaway A one-way measurement requiring no path or connection in the opposite direction. (COM/TA) 743-1995

straight binary *See:* binary.

straight condensing turbine (control systems for steam turbine-generator units) All the steam enters the turbine at one pressure and all the steam leaves the turbine exhaust at a pressure below atmospheric pressure. (PE/EDPG) 122-1985s

straight-cut control system (numerically controlled machines) A system in which the controlled cutting action occurs only along a path parallel to linear, circular, or other machine ways. (IA) [61]

straightforward trunking (manual telephone switchboard system) That method of operation in which the A operator gives the order to the B operator over the trunk on which talking later takes place. (PE/EEC) [119]

straight insertion sort *See:* insertion sort.

straight joint (power cable joints) A cable joint used for connecting two lengths of cable, each of which consists of one or more conductors. (PE/IC) 404-1986s

straight-line code A sequence of computer instructions in which there are no loops. (C) 610.12-1990

straight-line coding (1) (computers) Coding in which loops are avoided by the repetition of parts of the coding when required. (C) [20], [85]

(2) (software) A programming technique in which loops are avoided by stating explicitly and in full all of the instructions that would be involved in the execution of each loop. *See also:* unwind. (C) 610.12-1990

straight line sort *See:* linear sort.

straight noncondensing turbine (control systems for steam turbine-generator units) All the steam enters the turbine at one pressure and all the steam leaves the turbine exhaust at a pressure equal to or greater than atmospheric pressure. (PE/EDPG) 122-1985s

straight radix sort A radix sort in which items are sorted repeatedly on successive digits within the numeric representation of the sort key, starting with the least significant digit. (C) 610.5-1990w

straight-seated bearing (rotating machinery) (cylindrical bearing) A journal bearing in which the bearing liner is constrained about a fixed axis determined by the supporting structure. *See also:* bearing. (PE) [9]

straight selection sort *See:* selection sort.

straight storage system (electric power supply) A system in which the electrical requirements of a car are supplied solely from a storage battery carried on the car. *See also:* axle-generator system. (PE/EEC) [119]

straight two-way merge sort A variation of the natural two-way merge sort in which the set to be sorted is repeatedly divided into two ordered subsets of length 2 to the power of k , where k is the number of passes made so far. *Contrast:* natural two-way merge sort. (C) 610.5-1990w

strain attachment *See:* dead-end.

strain-bus structure A bus structure comprised of flexible conductors supported by strain insulators. (PE/SUB) 605-1998

strain element (of a fuse) That part of the current-responsive element that is connected in parallel with the fusible element in order to relieve it of tensile strain. *Note:* The fusible element melts and severs first, and then the strain element melts

EXHIBIT H

Microsoft

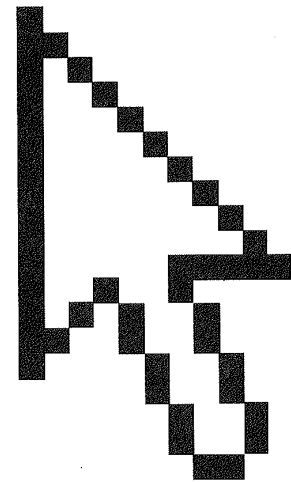
Computer Dictionary

Fifth Edition

Perkins Cole LLP

APR 09 2013

Law Library



PUBLISHED BY

Microsoft Press
A Division of Microsoft Corporation
One Microsoft Way
Redmond, Washington 98052-6399

Copyright © 2002 by Microsoft Corporation

All rights reserved. No part of the contents of this book may be reproduced or transmitted in any form or by any means without the written permission of the publisher.

Library of Congress Cataloging-in-Publication Data
Microsoft Computer Dictionary.--5th ed.

p. cm.

ISBN 0-7356-1495-4

1. Computers--Dictionaries. 2. Microcomputers--Dictionaries.

AQ76.5. M52267 2002
004'.03--dc21

200219714

Printed and bound in the United States of America.

2 3 4 5 6 7 8 9 QWT 7 6 5 4 3 2

Distributed in Canada by H.B. Fenn and Company Ltd.

A CIP catalogue record for this book is available from the British Library.

Microsoft Press books are available through booksellers and distributors worldwide. For further information about international editions, contact your local Microsoft Corporation office or contact Microsoft Press International directly at fax (425) 936-7329. Visit our Web site at www.microsoft.com/mspress. Send comments to mspinput@microsoft.com.

Active Desktop, Active Directory, ActiveMovie, ActiveStore, ActiveSync, ActiveX, Authenticode, BackOffice, BizTalk, ClearType, Direct3D, DirectAnimation, DirectDraw, DirectInput, DirectMusic, DirectPlay, DirectShow, DirectSound, DirectX, Entourage, FoxPro, FrontPage, Hotmail, IntelliEye, IntelliMouse, IntelliSense, JScript, MapPoint, Microsoft, Microsoft Press, Mobile Explorer, MS-DOS, MSN, Music Central, NetMeeting, Outlook, PhotoDraw, PowerPoint, SharePoint, UltimateTV, Visio, Visual Basic, Visual C++, Visual FoxPro, Visual InterDev, Visual J++, Visual SourceSafe, Visual Studio, Win32, Win32s, Windows, Windows Media, Windows NT, Xbox are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. Other product and company names mentioned herein may be the trademarks of their respective owners.

The example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious. No association with any real company, organization, product, domain name, e-mail address, logo, person, place, or event is intended or should be inferred.

Acquisitions Editor: Alex Blanton

Project Editor: Sandra Haynes

Body Part No. X08-41929

Melissa

gnals can be trans-
able. *See* media.

of the Intel 80x86
allows only 64 kilo-
byte for code. *See*

centration of circuit
hip. *Acronym:* MSI.

mputing, which is
g system, *mega-* has
he power of 2 (2²⁰)
4.

²⁰); sometimes inter-
Mb, Mbit.

(2²⁰); sometimes
ation: MB.

cles—usually used
bbreviation: MC.

equivalent to 1
r: MHz.

play.

esolution of one
l in reference to
rs, and computer

capable of dis-
mple, a video dis-
al pixels and 1024
lso

ord files in
first appeared in
s an attachment to
tant Message
g "Here is that
n the attachment
oft Outlook is
-mail addresses in
affected machine,

meltdown

the virus also changes the registry, infects the Normal.dot Word template (which, in turn, infects new documents), and, in Office 2000, disables the Word macro virus warning. Although the Melissa virus does not destroy data, it can affect e-mail performance through the increased volume of messages. If an infected document is open at a time when the day of the month is the same as the minute value of the current time, the virus inserts the text "Twenty-two points, plus triple-word-score, plus fifty points for using all my letters. Game's over. I'm outta here" at the current location of the cursor. The virus was named after an acquaintance of the hacker who developed it.

meltdown *n.* **1.** The complete collapse of a computer network caused by a higher level of traffic than the network can support. The term refers, by analogy, to the accidental melting down of a nuclear reactor core. **2.** Colloquially, the breakdown of a person, usually in a job situation, caused by overwork, stress, or failure.

member *n.* **1.** In object-oriented programming, a variable or routine that is part of a class. *See also* C++, class. **2.** A value that is part of a set data structure. *See also* set² (definition 1).

membrane keyboard *n.* A keyboard in which an unbroken plastic or rubber shell (a membrane) covers keys that have little or no travel (movement). Rather than use normal, full-travel keys, membrane keyboards use pressure-sensitive areas that are sometimes, but not always, defined by small bumps under the membrane.

memo field *n.* A field in a database file that can contain unstructured text.

memo pad *n.* A note-taking feature offered by many personal digital assistants and other handheld computing devices. Memo pad allows for the entry of short notes via typing or handwriting recognition applications. The notes can be categorized, organized, and edited later.

memory *n.* A device where information can be stored and retrieved. In the most general sense, memory can refer to external storage such as disk drives or tape drives; in common usage, it refers only to a computer's main memory, the fast semiconductor storage (RAM) directly connected to the processor. *See also* core, EEPROM, EPROM, flash memory, PROM, RAM, ROM. *Compare* bubble memory, mass storage.

memory management

memory bank *n.* The physical location on a motherboard where a memory module can be inserted. *See also* bank (definition 1).

memory board *n.* A plug-in printed circuit board that contains one or more memory chips. *See also* memory chip.

memory cache *n.* *See* CPU cache.

memory card *n.* A memory module that is used to extend RAM storage capacity or in place of a hard disk in a portable computer, such as a laptop, notebook, or handheld PC. The module is usually the size of a credit card and can be plugged into a PCMCIA-compliant portable computer. The module can be composed of EPROM, RAM, or ROM chips or flash memory. *Also called:* RAM card, ROM card. *See also* EPROM, flash memory, handheld PC, hard disk, memory cartridge, module (definition 2), PCMCIA, RAM, ROM.

memory cartridge *n.* A plug-in module containing RAM (random access memory) chips that can be used to store data or programs. Memory cartridges are used primarily in portable computers as smaller, lighter (but more expensive) substitutes for disk drives. Memory cartridges typically use either a nonvolatile form of RAM, which does not lose its contents when power is turned off, or battery-backed RAM, which maintains its contents by drawing current from a rechargeable battery within the cartridge. *Also called:* RAM cartridge. *See also* memory card, RAM. *Compare* ROM cartridge.

memory cell *n.* An electronic circuit that stores one bit of data. *See also* bit.

memory chip *n.* An integrated circuit devoted to memory storage. The memory storage can be *volatile* and hold data temporarily, such as RAM, or *nonvolatile* and hold data permanently, such as ROM, EPROM, EEPROM, or PROM. *See also* EEPROM, EPROM, integrated circuit, memory board, nonvolatile memory, PROM, RAM, volatile memory.

memory management *n.* **1.** In operating systems for personal computers, procedures for optimizing the use of RAM (random access memory). These procedures include selectively storing data, monitoring it carefully, and freeing memory when the data is no longer needed. Most current operating systems optimize RAM usage on their own; some older operating systems, such as early versions of MS-DOS, required the use of third-party utilities to optimize RAM usage and necessitated that the user be more

knowledgeable about how the operating system and applications used memory. *See also* memory management unit, RAM. **2.** In programming, the process of ensuring that a program releases each chunk of memory when it is no longer needed. In some languages, such as C and C++, the programmer must keep track of memory usage by the program. Java, a newer language, automatically frees any chunk of memory that is not in use. *See also* C, C++, garbage collection, Java.

memory management program *n.* **1.** A program used to store data and programs in system memory, monitor their use, and reassign the freed space following their execution. **2.** A program that uses hard disk space as an extension of the random access memory (RAM).

memory management unit *n.* The hardware that supports the mapping of virtual memory addresses to physical memory addresses. In some systems, such as those based on the 68020, the memory management unit is separate from the processor. In most modern microcomputers, however, the memory management unit is built into the CPU chip. In some systems, the memory management unit provides interfacing between the microprocessor and memory. This type of memory management unit is typically responsible for address multiplexing and, in the case of DRAMs, the refresh cycle. *Acronym:* MMU. *See also* physical address, refresh cycle, virtual address.

memory model *n.* The approach used to address the code and the data that are used in a computer program. The memory model dictates how much memory can be used in a program for code and how much for data. Most computers with a flat address space support only a single memory model. Computers with a segmented address space usually support multiple memory models. *See also* compact model, flat address space, large model, medium model, segmented address space, small model, tiny model.

memory module *n.* A removable circuit board, cartridge, or other carrier that contains one or more RAM memory chips. *See also* memory card, memory cartridge, RAM.

memory-resident *adj.* Permanently located in a computer's memory, rather than swapped in and out of memory as needed. *See also* memory, TSR.

memory scrubbing *n.* **1.** In mainframe computers, the process of a computer reading its own memory during idle periods in order to find and fix errors. **2.** The process of

examining and correcting errors as data is transferred from memory to the CPU of a computer.

memory size *n.* The memory capacity of a computer, usually measured in megabytes. *See also* megabyte, memory.

memory typewriter *n.* An electric typewriter with internal memory and typically a one-line liquid crystal display for viewing the contents of that memory. Memory typewriters can usually hold one page of text at a time, to which small modifications can be made. Memory typewriters usually do not retain the contents of memory when power is turned off.

MEMS *n.* Acronym for **micro-electromechanical systems**. A technology combining computers with extremely tiny mechanical devices. MEMS devices contain microcircuitry on a tiny silicon chip onto which a mechanical device such as a sensor or an actuator is attached. MEMS devices are used in switches, pacemakers, games, GPS tracking, data storage, and for accelerometers in air bags. Because MEMS devices have the potential to be manufactured in large quantities for little cost, many additional MEMS products are being planned or studied.

menu *n.* A list of options from which a user can make a selection in order to perform a desired action, such as choosing a command or applying a particular format to part of a document. Many application programs, especially those that offer a graphical interface, use menus as a means of providing the user with an easily learned, easy-to-use alternative to memorizing program commands and their appropriate usage.

menu bar *n.* A rectangular bar displayed in an application program's on-screen window, often at the top, from which menus can be selected by the user. Names of available menus are displayed in the menu bar; choosing one with the keyboard or with a mouse causes the list of options in that menu to be displayed.

menu-driven *adj.* Using menus to present choices of commands and available options. Menu-driven programs are usually considered friendlier and easier to learn than programs with a command-line interface. *Compare* command-line interface.

menu item *n.* A choice on a menu, selectable by either the keyboard or a mouse. In some instances, a menu item that is not available (that is, not appropriate) for a given

statistical multiplexer *n.* A multiplexing device that adds intelligence to time-division multiplexing by using buffering (temporary storage) and a microprocessor to combine transmission streams into a single signal and to allocate available bandwidth dynamically. *Also called:* stat mux. *See also* dynamic allocation, multiplexing, time-division multiplexing.

statistics *n.* The branch of mathematics that deals with the relationships among groups of measurements and with the relevance of similarities and differences in those relationships. *See also* binomial distribution, Monte Carlo method, probability, regression analysis, standard deviation, stochastic.

stat mux *n.* *See* statistical multiplexer.

status *n.* The condition at a particular time of any of numerous elements of computing—a device, a communications channel, a network station, a program, a bit, or other element—used to report on or to control computer operations.

status bar *n.* In Windows 9x and Windows NT 4 and later, a space at the bottom of many program windows that contains a short text message about the current condition of the program. Some programs also display an explanation of the currently selected menu command in the status bar. *See the illustration.*

Page 2 Sec 1 2/46 At 4" Ln 19 Col 9

Status bar.

status codes *n.* Strings of digits or other characters that indicate the success or failure of some attempted action. Status codes were commonly used to report the results of early computer programs, but most software today uses words or graphics. Internet users, especially those with UNIX shell accounts, are likely to encounter status codes while using the Web or FTP. *See also* HTTP status codes.

steganography *n.* A “hide-in-plain-sight” technique for concealing information by embedding a message within an innocuous cover message. In steganography, bits of unnecessary data within an image, sound, text, or even a blank file are replaced with bits of invisible information. The term steganography comes from the Greek for “covered writing” and has traditionally included any method of secret communication that conceals the existence of the message. Because steganography cannot be detected by decryption software, it is often used to replace or supplement encryption.

step-frame *n.* The process of capturing video images one frame at a time. This process is used by computers that are too slow to capture analog video images in real time.

stepper motor *n.* A mechanical device that rotates only a fixed distance each time it receives an electrical pulse. A stepper motor is part of a disk drive.

step-rate time *n.* The time required to move a disk actuator arm from one track to the next. *See also* actuator, stepper motor.

stereogram *n.* *See* autostereogram.

sticky *adj.* In reference to a Web site, properties such as targeted content or services that increase the amount of time users choose to spend at the site and increase user's desire to return to the site repeatedly.

StickyKeys *n.* An accessibility feature built into Macintosh and Windows computers that causes modifier keys such as Shift, Control, or Alt to “stay on” after they are pressed, eliminating the need to press multiple keys simultaneously. This feature facilitates the use of modifier keys by users who are unable to hold down one key while pressing another.

stochastic *adj.* Based on random occurrences. For example, a stochastic model describes a system by taking into account chance events as well as planned events.

stop bit *n.* In asynchronous transmission, a bit that signals the end of a character. In early electromechanical teleprinters, the stop bit provided time for the receiving mechanism to coast back to the idle position and, depending on the mechanism, had a duration of 1, 1.5, or 2 data bits. *See also* asynchronous transmission, Compare parity bit, start bit.

Stop error *n.* A serious error that affects the operating system and that could place data at risk. The operating system generates an obvious message, a screen with the Stop error, rather than continuing on and possibly corrupting data. *Also called:* blue screen error, fatal system error. *See also* Blue Screen of Death.

storage *n.* In computing, any device in or on which information can be kept. Microcomputers have two main types of storage: random access memory (RAM) and disk drives and other external storage media. Other types of storage include read-only memory (ROM) and buffers.

storage area network *n.* A high-speed network that provides a direct connection between servers and storage, including shared storage, clusters, and disaster-recovery

storage area network

storing video images one by one by computers that are images in real time.

device that rotates only as an electrical pulse. A device.

required to move a disk actuator. *See also* actuator,

um.

site, properties such as increase the amount of site and increase user's dily.

ature built into Macintosh causes modifier keys stay on" after they are pressed multiple keys simultaneously the use of modifier keys own one key while

occurrences. For example, a system by taking into planned events.

transmission, a bit that signals electromechanical time for the receiving device position and, a duration of 1, 1.5, or a transmission. *Com-*

affects the operating at risk. The operating age, a screen with the on and possibly corrupt-error, fatal system error.

ice in or on which informers have two main types (RAM) and disk drives Other types of storage and buffers.

-speed network that provides servers and storage, and disaster-recovery

storage device

devices. A storage area network, or SAN, includes components such as hubs and routers that are also used in local area networks (LANs), but it differs in being something of a "subnetwork" dedicated to providing a high-speed connection between storage elements and servers. Most SANs rely on fiber-channel connections that deliver speeds up to 1000 Mbps and can support up to 128 devices. SANs are implemented to provide the scalability, speed, and manageability required in environments that demand high data availability. *Acronym:* SAN. *Also called:* system area network.

storage device *n.* An apparatus for recording computer data in permanent or semipermanent form. When a distinction is made between primary (main) storage devices and secondary (auxiliary) storage devices, the former refers to random access memory (RAM) and the latter refers to disk drives and other external devices.

storage location *n.* The position at which a particular item can be found—either an addressed location or a uniquely identified location on a disk, tape, or similar medium.

storage media *n.* The various types of physical material on which data bits are written and stored, such as floppy disks, hard disks, tape, and optical discs.

storage tube *n.* *See* direct view storage tube.

store-and-forward *n.* A method of delivering transmissions in which messages are held temporarily by an intermediary before being sent on to their destination. Store and forward is used by some switches in delivering packets to their destinations. *Compare* cut-through switch.

stored procedure *n.* A precompiled collection of SQL statements and optional control-of-flow statements stored under a name and processed as a unit. They are stored in an SQL database and can be run with one call from an application.

stored program concept *n.* A system architecture scheme, credited largely to the mathematician John von Neumann, in which both programs and data are in direct-access storage (random access memory, or RAM), thereby allowing code and data to be treated interchangeably. *See also* von Neumann architecture.

storefront *n.* *See* virtual storefront.

storm *n.* On a network, a sudden, excessive burst of traffic. Storms are often responsible for network outages.

STP *n.* Acronym for shielded twisted pair. A cable consisting of one or more twisted pairs of wires and a sheath

streaming buffer

of foil and copper braid. The twists protect the pairs from interference by each other, and the shielding protects the pairs from interference from outside. Therefore, STP cable can be used for high-speed transmission over long distances. *See also* twisted-pair cable. *Compare* UTP.

straight-line code *n.* Program code that follows a direct sequence of statements rather than skipping ahead or jumping back via transfer statements such as GOTO and JUMP. *See also* GOTO statement, jump instruction. *Compare* spaghetti code.

stream¹ *n.* Any data transmission, such as the movement of a file between disk and memory, that occurs in a continuous flow. Manipulating a data stream is a programming task. Consumers, however, are likely to encounter references to streams and streaming in connection to the Internet, which has increased reliance on stream techniques to enable users (even those with slower equipment) to access large multimedia files—especially those containing audio and video components—and to display or play them before all the data has been transferred.

stream² *vb.* To transfer data continuously, beginning to end, in a steady flow. Many aspects of computing rely on the ability to stream data: file input and output, for example, and communications. If necessary, an application receiving a stream must be able to save the information to a buffer in order to prevent loss of data. On the Internet, streaming enables users to begin accessing and using a file before it has been transmitted in its entirety.

stream cipher *n.* A method for encrypting a data sequence of unlimited length using a key of fixed length. *See also* key (definition 3). *Compare* block cipher.

streaming *n.* **1.** On the Internet, the process of delivering information, especially multimedia sound or video, in a steady flow that the recipient can access as the file is being transmitted. **2.** In magnetic tape storage devices, a low-cost technique to control the motion of the tape by removing tape buffers. Although streaming tape compromises start/stop performance, it achieves highly reliable storage and retrieval of data, and is useful when a steady supply of data is required by a particular application or computer.

streaming buffer *n.* A small sound buffer that can play lengthy sounds because the application dynamically loads audio data into the buffer as it plays. For example, an application could use a buffer that can hold 3 seconds of audio data to play a 2-minute sound. A streaming buffer requires much less memory than a static buffer. *See also* static buffer.

EXHIBIT I

COPIES OF PAPERS
ORIGINALLY FILED

Attorney Docket No: 1457CPA



CERTIFICATE OF MAIL

I hereby certify that this correspondence is being deposited with the United States Postal Service as First Class Mail in an envelope addressed to the Assistant for Patents and Trademarks, Washington, D.C. 20231, on April 26, 2002.

Grace Alicea
Grace Alicea

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In Re Application of:

Date: April 26, 2002

Portman, et al.

Serial No. 09/399,908

Group Art Unit: 2181

Filed: Sept. 21, 1999

Examiner: Dharia, R.

For: METHOD AND SYSTEM FOR CONTROLLING DATA IN A
COMPUTER SYSTEM

RECEIVED

MAY 13 2002

Technology Center 210

Assistant Commissioner for Patents
Washington, D.C. 20231

AMENDMENT

Sir:

In response to the Final Office Action dated January 28, 2002, please amend the above-identified application in the following manner:

IN THE SPECIFICATION:

Please replace the title with: "METHOD AND SYSTEM FOR CONTROLLING DATA IN A
COMPUTER SYSTEM IN THE EVENT OF A POWER FAILURE"

In the Abstract (Page 18, line 2):

A method and system for controlling data in a computer system when the computer system loses power is disclosed. The method and system comprises activating a plurality of super capacitors to supply power to the computing engine based upon power being removed

Attorney Docket No: 1457CPA

Q1 from the computer system and reconfiguring the data in the computing engine. Through the use of a system and method, large amounts of newly written and modified data can be stored from the volatile memory to the non-volatile memory in the event of a sudden external system power loss. The system and method allows data to be rapidly and irretrievably erased from the non-volatile memory automatically, in the event of a sudden loss of external power, or manually. This capability consumes minimal space and weight and is implemented in an affordable manner.

IN THE CLAIMS:

Q2 1. (Amended) A method for controlling data in a computer system when the computer system loses power, the computer system comprising a computing engine, comprising the steps of:

- (a) activating a plurality of super capacitors to supply power to the computing engine based upon power being removed from the computer system;
- (b) reconfiguring the data in the computing engine; and
- (c) deactivating the plurality of super capacitors to cut off power to the computing engine based upon the plurality of super capacitors discharging to a predetermined level.

Q3 10. (Amended) A system for controlling data in a computer system when the computer system loses power, the computer system comprising a computer engine, comprising:

- means for activating a plurality of super capacitors to supply power to the computing engine based upon power being removed from the computer system;
- means for reconfiguring the data in the computing engine; and
- means for deactivating the plurality of super capacitors to cut off power to the computing engine based upon the plurality of super capacitors discharging to a predetermined level.

Please add claims:

19. A method for controlling data in a computer system when the computer system loses power, the computer system comprising a computing engine, comprising the steps of:

- (a) allowing the computer system to be placed in an erase mode;
- (b) activating a plurality of super capacitors to supply power to the computing engine based upon power being removed from the computer system; and
- (c) securely erasing data in the computing engine to prevent access to the data each time power is removed from the computing engine and the system is in the erase mode.

20. The method of claim 19 wherein the allowing step (a) further includes the step of:

- (a1) allowing the computer system to be placed in the erase mode or in a save mode.

21. The method of claim 20 further comprising the step of:

- (d) saving the data in the computing engine in a nonvolatile storage in each time power is removed from the computing engine and the system is in the save mode.

22. A method for controlling data in a computer system when the computer system loses power, the computer system comprising a computing engine, comprising the steps of:

- (a) activating a plurality of super capacitors to supply power to the computing engine based upon power being removed from the computer system;
- (b) placing the computer system in a standby mode based upon power being removed from the computer system;
- (c) providing power to the computer system using the plurality of super capacitors in response to a user interaction after the computer system has entered standby mode; and

Attorney Docket No: 1457CPA

- (d) erasing data in the computing engine based upon the user interaction.

23. A method for controlling data in a computer system when the computer system loses power, the computer system comprising a computing engine, comprising the steps of:

- (a) activating a plurality of super capacitors to supply power to the computing engine based upon power being removed from the computer system;
- (b) reconfiguring the data in the computing engine; and
- (c) ensuring that the plurality of super capacitors is maintained in a desired temperature range.

24. A system for controlling data in a computer system when the computer system loses power, the computer system comprising a computer engine, comprising:

means for activating a plurality of super capacitors to supply power to the computing engine based upon power being removed from the computer system; and

means for reconfiguring the data in the computing engine, the reconfiguring means being capable of being placed in an erase mode;

wherein the reconfiguring means further securely erase data in the computing engine to prevent access to the data each time each time power is removed from the computing engine and the system is in the erase mode.

25. The system of claim 24 wherein the reconfiguring means can further be placed in the erase mode or in a save mode.

26. The system of claim 25 wherein the reconfiguring means further save the data in the computing engine in a nonvolatile storage in each time power is removed from the computing engine and the system is in the save mode.

27. A system for controlling data in a computer system when the computer system loses power, the computer system comprising a computer engine, comprising:

means for activating a plurality of super capacitors to supply power to the computing engine based upon power being removed from the computer system, for placing the computer system in a standby mode based upon power being removed from the computer system, providing power to the computer system based upon a user interaction after the computer system has entered standby mode;

means for reconfiguring the data in the computing engine, the reconfiguring means further including means for erasing a portion of the data in the computing engine based upon the user interaction.

28. A system for controlling data in a computer system when the computer system loses power, the computer system comprising a computer engine, comprising:

means for activating a plurality of super capacitors to supply power to the computing engine based upon power being removed from the computer system;

means for reconfiguring the data in the computing engine; and

means for ensuring that the plurality of super capacitors is maintained in a desired temperature range.

Attorney Docket No: 1457CPA

REMARKS

Claims 1-18 are pending in the present Application. Applicant has amended claims 1 and 10. Applicant has added claims 19-28. Consequently, claims 1-28 remain pending in the present Application.

Applicant has amended claims 1 and 10 to recite that the super capacitors are deactivated when their charge reach a predetermined level. Support for the amendments to claims 1 and 10 can be found in the specification, page 9, line 23-page 10, line 3. Support for new claims 19-21 and 24-26 can be found in the specification, page 8, lines 1-6 and page 11, lines 3-10. Support for new claims 22 and 27 can be found in the specification, page 12, lines 11-19. Support for new claims 23 and 28 can be found in the specification, page 6, lines 8-16. Accordingly, Applicant respectfully submits that no new matter is added.

In the above-identified Office Action, the Examiner objected to portions of the present application and rejected the claims for the same reasons as in a previous Office Action in the parent case ("previous Office Action") dated June 15, 2001. In the previous Office Action, the Examiner objected to the Abstract and the title. Applicant has amended the Abstract in accordance with the Examiner's suggestion. In addition, Applicant has changed the title to: METHOD AND SYSTEM FOR CONTROLLING DATA IN A COMPUTER SYSTEM IN THE EVENT OF A POWER FAILURE, which Applicant believes is descriptive of the present invention as recited in varying scope in claims 1-26. Accordingly, Applicant respectfully submits that the Examiner's objection to the specification and the title has been overcome.

In the previous Office Action, the Examiner also rejected claims 1-4, 6, 7, 10-13, 15 and 16 under 35 U.S.C. § 102 as being anticipated by U.S. Patent No. 5,542,042 ("Manson"). The Examiner also rejection claims 5, 8, 9, 14, 17 and 18 under 35 U.S.C. § 103 as being obvious in light of Manson in view of U.S. Patent No. 5,768,208 ("Bruwer").

Attorney Docket No: 1457CPA

The Examiner has rejected claims 1-4, 6, 7, 10-13, 15 and 16 as being anticipated by Manson. The Examiner indicated that Manson transferring data from volatile to nonvolatile memory, and erasing the volatile memory.

Applicant respectfully traverses the Examiner's rejection. Claim 1 recites a method for controlling data in a computer system, which includes a computing system, when the computer system loses power. Claim 1 recites the steps of activating a plurality of super capacitors to supply power to the computing engine based upon power being removed from the computer system and reconfiguring the data in the computing engine. Claim 1 further recites the step of deactivating the plurality of super capacitors to cut off power to the computing engine based upon the plurality of super capacitors discharging to a predetermined level. Claim 10 recites an analogous system that includes means for performing the steps recited in claim 1.

Because the method and system of claims 1 and 10, respectively, teach that power is cut off to the computing engine when the super capacitors reach a predetermined level, performance and reliability of the computer system are improved. The predetermined level can be set above the minimum operating voltage of the computer system, preventing the computer system from running with unpredictable results. Specification, page 12, lines 5-8. Furthermore, the predetermined level can be set above a zero level of power for the computer system. When the super capacitors are not fully discharged, it may take only a few seconds to recharge the super capacitors, rather than several minutes, rapidly providing protection against power loss. Specification, page 12, lines 8-10.

Claims 1 and 10 are neither taught nor suggested by Manson. Similarly, claims 1 and 10 are neither taught nor suggested by Manson in view of Bruwer.

In contrast to the present invention as recited in varying scope in claims 1 and 10, Manson fails to teach or suggest the combination of elements recited in claims 1 and 10

Attorney Docket No: 1457CPA

including deactivating the super capacitors to cut off power to the computing engine based upon the plurality of super capacitors discharging to a predetermined level. Manson teaches a system which stores “critical” data upon a power loss. In particular, when a power loss is detected, the “critical” data is copied to an accumulator, then complemented. Manson, col. 3, lines 24-40. The complemented data is thus stored. Upon power on reset of the computer system, the complemented data is checked. Manson, col. 2, lines 29-30, 46-55. If the comparison indicates that the complemented data matches data generated after power is turned back on, then operation continues. However, if the comparison indicates that there was a memory loss during the power outage, then the memory is initialized. Manson, col. 2, lines 55-61.

Bruwer describes a system that is used to provide fail safe non-volatile memory programming, even in the event of a power loss. Bruwer, Abstract. Bruwer teaches that “soft writes” are a problem. Bruwer, col. 1, lines 23-28. A soft write occurs when an attempt is made to change the state of information in a cell of the memory, but the attempt is not completed due to some type of interrupt, such as a power failure. Bruwer, col. 1, lines 19-23. Bruwer also teaches that a standard write operation first erases a cell by writing a one, then changes the value in the cell from a 1 to a zero or leaves the cell unchanged, depending upon the desired data for the cell. Bruwer, col. 1, lines 29-34. The system of Bruwer determines whether sufficient charge exists in a bank of capacitors to program a non-volatile memory. Bruwer, col. 4, lines 51-60. If not, then the system of Bruwer does not operate upon a power loss. Bruwer, col. 4, lines 60-67. If the capacitors have sufficient charge, Bruwer completes a programming cycle interrupted by a power loss. Bruwer, col. 4, line 67-col. 5, line 4.

Although both Manson and Bruwer discuss operations performed in the event of a power failure, Applicant can find no mention in either Manson or Bruwer of any mechanism for cutting off power supplied by the capacitors. More specifically, Manson and Bruwer are devoid of

Attorney Docket No: 1457CPA

mention of deactivating super capacitors when the super capacitors are discharged to a predetermined level to cut power off to the computer system. Any combination of Manson and Bruwer, therefore, would not deactivate super capacitors when the super capacitors are discharged to a predetermined level to cut off power to the computer system. Consequently, Manson and Bruwer, separately or in combination, cannot teach or suggest the method and system recited in claims 1 and 10, respectively. Accordingly, Applicant respectfully submits that claims 1 and 10 are allowable over the cited references.

Claim 2-4, 6, and 7 depend upon independent claim 1. Similarly, claims 11-13, 15 and 16 depend upon independent claim 10. Consequently, the arguments herein apply with full force to claims 2-4, 6, 7, 11-13, 15 and 16. Accordingly, Applicant respectfully submits that claims 2-4, 6, 7, 11-13, 15 and 16 are allowable over cited references.

The Examiner also rejected claims 5, 8, 9, 14, 17 and 18 under 35 U.S.C. § 103 as being obvious in light of Manson in view of Bruwer. The Examiner cited Bruwer as teaching that it is known to erase a non-volatile memory and that one of ordinary skill in the art would have incorporated the teachings of Bruwer in the system of Manson to prevent the "soft writes" described by Bruwer.

Claims 5, 8 and 9 depend on independent claim 1. Claims 14, 17 and 18 depend on independent claim 10. Consequently, the arguments herein apply with full force to claims 2, 5-11, and 13-17. Accordingly, Applicant respectfully submits that claims 2-5, and 13-17 are also allowable over the cited references.

New claims 19 and 24 recite a method and system, respectively, for controlling data in a computer system when the computer system loses power. The method recited in claim 19 allows a user to place the computer system in an erase mode and activates a plurality of super capacitors to supply power to the computing engine based upon power being removed from the computer

system. The method recited in claim 19 also securely erases data in the computing engine to prevent access to the data each time power is removed from the computing engine and the system is in erase mode. Claim 23 recites a system analogous to the method recited in claim 19. The method and system recited in claims 19 and 23, respectively, securely erase data to prevent access to the data in response to power being removed from the computing engine. As a result, it can be ensured that sensitive data on the computer system cannot be accessed after power is removed from the computer system.

In contrast, neither Manson nor Bruwer, separately or in combination, teach or suggest the combination of elements in claims 19 and 23 including erasing data each time power is removed from the computer system and the computer system is in the erase mode. As described above, Manson teaches that “critical” data are saved in the event that power is removed from the computer system. It is only upon power on reset of the computer system that data might be erased only if a comparison indicates that there was a memory loss during the power outage. Manson, col. 2, lines 55-61. Applicant can find no mention in Manson of erasing data for any other reason. More specifically, Applicant can find no mention in Manson of erasing data in response to power being removed from the computer system and the computer system being in an erase mode. Manson also fails to mention an erase mode. Consequently, Manson cannot teach or suggest the method and system recited in claims 19 and 23, respectively.

Similarly, Bruwer teaches that a programming cycle is completed upon a power loss. In order to do so, data may be erased as part of the programming cycle if the data had not been erased prior to the power loss. Thus, data would only be erased if the programming cycle had not completed the erasing data in the computer system. Consequently, Bruwer would not erase data each time the computer system loses power and the computer system is in an erase mode. Furthermore, Bruwer fails to mention the existence of an erase mode in the computer system.

A

Attorney Docket No: 1457CPA

Consequently, Bruwer also fails to teach or suggest erasing data each time power is removed from a computer system and the computer system is in erase mode.

Neither Manson nor Bruwer discusses an erase mode or erasing data each time power is removed from a computer system and the computer system is in erase mode. As a result, any combination of Manson and Bruwer cannot teach or suggest this feature. Accordingly, claims 19 and 23 are allowable over the cited references.

Claims 20-21 and 25-26 depend upon claims 19 and 24, respectively. Therefore, the arguments herein apply with full force to claims 20-21 and 25-26. Accordingly, Applicant respectfully submits that claims 20-21 and 25-26 are allowable over the cited references. Furthermore, Applicant can find no mention in either Manson or Bruwer of having both a save mode and an erase mode available on the computer system. Manson and Bruwer also fail to discuss any mechanism for saving data in nonvolatile storage in the save mode, while erasing data in the computing engine in the erase mode. Thus, Manson and Bruwer, separately or in combination, cannot teach or suggest the method and system recited in claims 20-21 and 25-26, respectively. Accordingly, Applicant respectfully submits that claims 20-21 and 25-26 are each separately allowable over the cited references.

New claims 22 and 27 recite a method and system, respectively, for controlling data in a computer system when the computer system loses power. The computer system includes a computing engine. The method recited in claim 22 includes the steps of activating a plurality of super capacitors to supply power to the computing engine based upon power being removed from the computer system and placing the computer system in a standby mode based upon power being removed from the computer system. The method recited in claim 22 also includes the steps of providing power to the computer system using the plurality of super capacitors in response to a user interaction after the computer system has entered standby mode and erasing data in the

computing engine based upon the user interaction. Claim 27 recites an analogous system. Thus, using the method and system recited in claims 22 and 27, respectively, a user can select to erase data while the computer system is disconnected from a power.

In contrast, both Manson and Bruwer describe providing power to the computer system to keep the computer system operating and performing functions with computer system after power is removed. Neither Manson nor Bruwer describe placing the computer system in a standby mode based upon power being removed from the computer system. Neither Manson nor Bruwer describe providing power to the computer system using super capacitors in response to a user interaction after the computer system has entered standby mode. Furthermore, both Manson and Bruwer fail to mention erasing data in the computing engine based upon the user interaction. Consequently, any combination of Manson and Bruwer cannot teach or suggest placing the computer system in a standby mode based upon power being removed from the computer system, providing power to the computer system using the super capacitors in response to a user interaction after the computer system has entered standby mode and erasing data in the computing engine based upon the user interaction. Manson and Bruwer, separately or in combination, thus cannot teach or suggest the method and system recited in claims 22 and 27. Accordingly, Applicant respectfully submits that claims 22 and 27 are allowable over the cited references.

New claims 23 and 28 recite a method and system for controlling data in a computer system when the computer system loses power. The computer system includes a computing engine. The method recited in claim 23 includes the steps of activating a plurality of super capacitors to supply power to the computing engine based upon power being removed from the computer system and reconfiguring the data in the computing engine. The method recited in claim 23 also includes the step of ensuring that the plurality of super capacitors is maintained in

Attorney Docket No: 1457CPA

a desired temperature range. Claim 28 recites and analogous system. As a result, the super capacitors can be maintained in their operating temperature range, ensuring their ability to provide power to the computer system in the event of power being removed from the computer system.

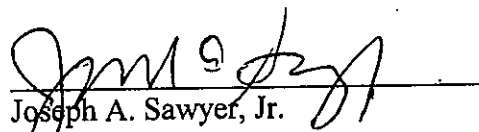
In contrast, Applicant can find no mention in either Bruwer or Manson of any mechanism for ensuring that the capacitors used to provide power to the computer system are maintained in a particular temperature range. Consequently, any combination of Manson and Bruwer would also fail to teach or suggest this feature. Manson and Bruwer, separately or in combination, therefore, cannot teach or suggest the method and system recited in claims 23 and 28. Accordingly, Applicant respectfully submits that claims 23 and 28 are allowable over the cited references.

Accordingly, for the above-mentioned reasons, Applicant respectfully submits that the claims are allowable over the cited reference. Consequently, Applicant respectfully requests reconsideration and allowance of the claims as currently presented.

Attached hereto is a marked-up version of the changes made to the specification and claims by the current amendment. The attached page is captioned **"Version with markings to show changes made"**.

Applicant's attorney believes that this application is in condition for allowance. Should any unresolved issue remain, the Examiner is invited to call Applicant's attorney at the telephone number indicated below.

Respectfully submitted,


Joseph A. Sawyer, Jr.
Sawyer Law Group LLP
Attorney for Applicant(s)
Reg. No. 30,801
(650) 493-4540

VERSION WITH MARKINGS TO SHOW CHANGES MADE

IN THE SPECIFICATION:

Please replace the title with: "METHOD AND SYSTEM FOR CONTROLLING DATA IN A COMPUTER SYSTEM IN THE EVENT OF A POWER FAILURE"

In the Abstract (Page 18, line 2):

[The present invention discloses] A method and system for controlling data in a computer system when the computer system loses power[, the computer system comprising a computing engine] is disclosed. The method and system comprises activating a plurality of super capacitors to supply power to the computing engine based upon power being removed from the computer system and reconfiguring the data in the computing engine. Through the use of a system and method[in accordance with the present invention, a user is able to correctly store], large amounts of newly written and modified data can be stored from the volatile memory to the non-volatile memory in the event of a sudden external system power loss. [Furthermore, the user of a]The system and method [in accordance with the present invention will be able to]allows data to be rapidly and irretrievably erased [data]from the non-volatile memory automatically, in the event of a sudden loss of external power, or manually. This capability consumes minimal space and weight and is implemented in an affordable manner.

IN THE CLAIMS:

1. (Amended) A method for controlling data in a computer system when the computer system loses power, the computer system comprising a computing engine, comprising the steps of:
 - (a) activating a plurality of super capacitors to supply power to the computing engine based upon power being removed from the computer system; [and]

Attorney Docket No: 1457CPA

- (b) reconfiguring the data in the computing engine; and
- (c) deactivating the plurality of super capacitors to cut off power to the computing engine based upon the plurality of super capacitors discharging to a predetermined level.

10. (Amended) A system for controlling data in a computer system when the computer system loses power, the computer system comprising a computer engine, comprising:
means for activating a plurality of super capacitors to supply power to the computing engine based upon power being removed from the computer system; [and]
means for reconfiguring the data in the computing engine; and
means for deactivating the plurality of super capacitors to cut off power to the computing engine based upon the plurality of super capacitors discharging to a predetermined level.

Please add claims:

19. A method for controlling data in a computer system when the computer system loses power, the computer system comprising a computing engine, comprising the steps of:

- (a) allowing the computer system to be placed in an erase mode;
- (b) activating a plurality of super capacitors to supply power to the computing engine based upon power being removed from the computer system; and
- (c) securely erasing data in the computing engine to prevent access to the data each time power is removed from the computing engine and the system is in the erase mode.

20. The method of claim 19 wherein the allowing step (a) further includes the step of:
(a1) allowing the computer system to be placed in the erase mode or in a save mode.

Attorney Docket No: 1457CPA

21. The method of claim 20 further comprising the step of:

(d) saving the data in the computing engine in a nonvolatile storage in each time power is removed from the computing engine and the system is in the save mode.

22. A method for controlling data in a computer system when the computer system loses power, the computer system comprising a computing engine, comprising the steps of:

(a) activating a plurality of super capacitors to supply power to the computing engine based upon power being removed from the computer system;

(b) placing the computer system in a standby mode based upon power being removed from the computer system;

(c) providing power to the computer system using the plurality of super capacitors in response to a user interaction after the computer system has entered standby mode; and

(d) erasing data in the computing engine based upon the user interaction.

23. A method for controlling data in a computer system when the computer system loses power, the computer system comprising a computing engine, comprising the steps of:

(a) activating a plurality of super capacitors to supply power to the computing engine based upon power being removed from the computer system;

(b) reconfiguring the data in the computing engine; and

(c) ensuring that the plurality of super capacitors is maintained in a desired temperature range.

24. A system for controlling data in a computer system when the computer system loses power, the computer system comprising a computer engine, comprising:

A

Attorney Docket No: 1457CPA

means for activating a plurality of super capacitors to supply power to the computing engine based upon power being removed from the computer system; and

means for reconfiguring the data in the computing engine, the reconfiguring means being capable of being placed in an erase mode;

wherein the reconfiguring means further securely erase data in the computing engine to prevent access to the data each time each time power is removed from the computing engine and the system is in the erase mode.

25. The system of claim 24 wherein the reconfiguring means can further be placed in the erase mode or in a save mode.

26. The system of claim 25 wherein the reconfiguring means further save the data in the computing engine in a nonvolatile storage in each time power is removed from the computing engine and the system is in the save mode.

27. A system for controlling data in a computer system when the computer system loses power, the computer system comprising a computer engine, comprising:

means for activating a plurality of super capacitors to supply power to the computing engine based upon power being removed from the computer system, for placing the computer system in a standby mode based upon power being removed from the computer system, providing power to the computer system based upon a user interaction after the computer system has entered standby mode;

Attorney Docket No: 1457CPA

means for reconfiguring the data in the computing engine, the reconfiguring means further including means for erasing a portion of the data in the computing engine based upon the user interaction.

28. A system for controlling data in a computer system when the computer system loses power, the computer system comprising a computer engine, comprising:

means for activating a plurality of super capacitors to supply power to the computing engine based upon power being removed from the computer system;

means for reconfiguring the data in the computing engine; and

means for ensuring that the plurality of super capacitors is maintained in a desired temperature range.