EMAX2SPC-0001
Ver.1.0: Oct. 20 2012
Ver.1.1: Mar. 10 2013

EMAX2/intel ARCHITECTURE HANDBOOK (Energy-aware Multimode Accelerator eXtension)

Nara Institute of Science and Technology
Computing Architecture Laboratory
Accelerator Group

Table of contents

1 Hardware structure **7**
1.1 Basic function . . . 7
1.2 Basic structure . . . 7

2 Software specification **11**
2.1 Instruction format . . . 11
2.2 ALU_OP . . . 13
2.3 MEM_OP . . . 16
2.4 RGI . . . 16
2.5 LMM_CONTROL . . . 16
2.6 Hints for sophisticated use of hardware . . . 17
2.7 Application Binary Interface . . . 18
2.8 Configuration data . . . 19
2.9 Initial Values of Registers . . . 21
2.10 LMM Information . . . 21

3 Examples **23**
3.1 tone_curve . . . 23
3.2 hokan1 . . . 24
3.3 hokan2 . . . 25
3.4 hokan3 . . . 26

4 Control **27**
4.1 LMM の仕様 . . . 27
4.2 使用するメモリ空間 . . . 28
4.3 各 UNIT に対する制御情報の伝達 . . . 29
4.3.1 Idle (pe0_status=STATUS_IDLE) . . . 29
4.3.2 Unit configuration in progress (pe0_status=STATUS_CONF) . . . 30
4.3.3 Register initialization in progress (pe0_status=STATUS_REGV) . . . 31
4.3.4 LMM tag initialization in progress (pe0_status=STATUS_LMMI) . . . 32
4.3.5 LMM loading in progress (pe0_status=STATUS_LMM_LOAD) . . . 33
4.3.6 Start execution (pe0_status=STATUS_START) . . . 33
4.3.7 Execution in progress (pe0_status=STATUS_EXEC) . . . 33
4.3.8 LMM drainage in progress (pe0_status=STATUS_LMM_DRAIN) . . . 34
4.3.9 Terminate execution (pe0_status=STATUS_TERM) . . . 34

5 How to use EMAX2 simulator **35**
5.1 Compiling application programs . . . 35
5.2 Executing application programs . . . 35

A 評価システム **37**
A.1 GP6X760MP システムの構成 37
A.2 プロセッサ機能と HOST 機能のインタフェース 40
A.3 プロセッサ機能と DDR3 のインタフェース 41

B ASIC 拡張機能 **45**
B.1 QFP208 パッケージ用インタフェース 45

C References **49**
C.1 GP6X760 documents 49
C.2 EMAX2 documents/sources 49
C.3 EMAX2 toolchains 50

List of Figures

1.1 EMAX2 basic function. 9
1.2 Interconnection of functions. 9
1.3 EMAX2 basic unit. 10
1.4 Interconnection of units. 10

2.1 Instruction format for each unit. 12
2.2 Format of ALU operation. 13
2.3 Lower level configuration data (1/2). 19
2.4 Lower level configuration data (2/2). 20
2.5 Initial Values of Registers. 21
2.6 LMM Information. 21

4.1 Structure of LMM. 27
4.2 Timing chart of LMM. 27
4.3 General control information of EMAX2. 29
4.4 conf[][] → unit[][] のタイミングチャート 30
4.5 regv[][] → unit[][] のタイミングチャート 31
4.6 lmmi[][] → ctl_new[][]/ddr3_tlb[][] のタイミングチャート 32
4.7 source_DDR3 → lmm[][] のタイミングチャート 33
4.8 lmm[][] → result_DDR3 のタイミングチャート 34

A.1 全体構成 38
A.2 プロセッサ制御インタフェースのタイミングチャート (WRITE/READ) 40
A.3 DDR3 インタフェースのタイミングチャート (WRITE) 41
A.4 DDR3 インタフェースのタイミングチャート (WRITE バーストモード) 42
A.5 DDR3 インタフェースのタイミングチャート (READ) 43
A.6 DDR3 インタフェースのタイミングチャート (READ バーストモード) 44

B.1 GP6X760 ボードの構成 46
B.2 QFP208 のピン配置（周辺の記号は XILINX の対応ピン番号（現状は仮）） 47

List of Tables

2.1 EX1 operations 14
2.2 EX2 operations 15
2.3 SFT operations 15
2.4 Memory operations 16

4.1 初期モデルにおけるメモリ空間の運用 28

A.1 制御レジスタ空間と DDR3 空間 39
A.2 プロセッサ-HOST 機能のインタフェース信号（ASIC インタフェースとしては 40 本） . . . 40
A.3 プロセッサ-DDR3 のインタフェース信号（ASIC インタフェースとしては 82 本） 41

Chapter 1

Hardware structure

1.1 Basic function

The basic function of a minimal component of EMAX2 is shown in Figure 1.1. At the first stage, the values stored in six input registers are read and transferred to second level intermediate registers dedicated to each fixed portion of EX1 (ALU) or EAG (Effective Address Generator) through four internal data bus (IXB). At the second stage, one of the registers at the same portion group is selected among neighbor components and transferred to third level intermediate registers through 6 external data bus (ETB). The main role of second and third level registers is to transfer any input registers in neighbor components to any portion of EX1 or EAG across different components. Two stage pipelining is employed to reduce the impact on frequency caused by the delay of long wires and many selectors connecting the components.

At the third stage, EX1 (3-in ALU) and EAG (2-in adder) produce the results and write into fourth level intermediate registers dedicated to each of EX1 and EAG. At the fourth stage, EX2 (2-in ALU) gets the result of EX1, produces final result and writes into final (fifth) level dedicated register. In the same way, when a load instruction is allocated to the component, LMM (Local Memory Module) produces the load data and writes into final level dedicated register.

Each of final level registers can select the input from dedicated EX2/LMM or dedicated FIFO. Each FIFO is filled with the data supplied from some neighbor LMM through an external data bus (EMB). When tri-state buffer between EMB is cut off, each component can use EMB independently, so that each LMM can supply the data to EX2_FIFO in the same component. When tri-state buffer between EMB is opened, neighbor FIFOs can share the output of some neighbor LMM connected to the EMB. The main usage of FIFOs is executing some kind of `"load (I-12); load (I-8); load (I-4); load (I)"` instructions in the same row.

The combination of EX1 and EX2 is suitable for multimedia or floating-point add/multiply operations. Moreover, the combination of EX1, EAG and local memory module can directly execute some kind of `"store (A+B+C)->(base+offset)"` instructions.

1.2 Basic structure

Many minimal components are connected to form EMAX2 as shown in Figure 1.2. For the pipelined execution, the third level registers in the first components (colored with black) are merged into the first level registers in the second components (colored with red). Also the outputs of final level registers in the first components are connected to IDB and EDB in the second components and to IXB in the third components (colored with green). In the same way, the outputs of final level registers in the final components (colored with purple) are connected to IDB and EDB in the first components and to IXB in the second components. Consequently, EMAX2 has a vertical ring network of ALUs and local memory modules.

However, it is difficult to design the hardware of EMAX2 from the view of the function, because several components in different levels are mixed in the same physical area as shown in Figure 1.2. For alleviating this complexity, "unit" which is a folded form of function is introduced as shown in Figure 1.3. From the view of unit, each unit has single lane of intermediate registers and single lane of final registers. All data is supplied from the final registers in the previous units and EX1, EX2, EAG and local memory module finally store the results into the final registers. The whole structure of EMAX2 based on unit is shown in Figure 1.4.

Above array structure is specially designed for executing a loop with no dependency between different iterations. After each of the registers and the instructions for a loop are mapped on each unit, EMAX2 can execute all instruction in a loop simultaneously and can produce the result of each iteration every cycle.

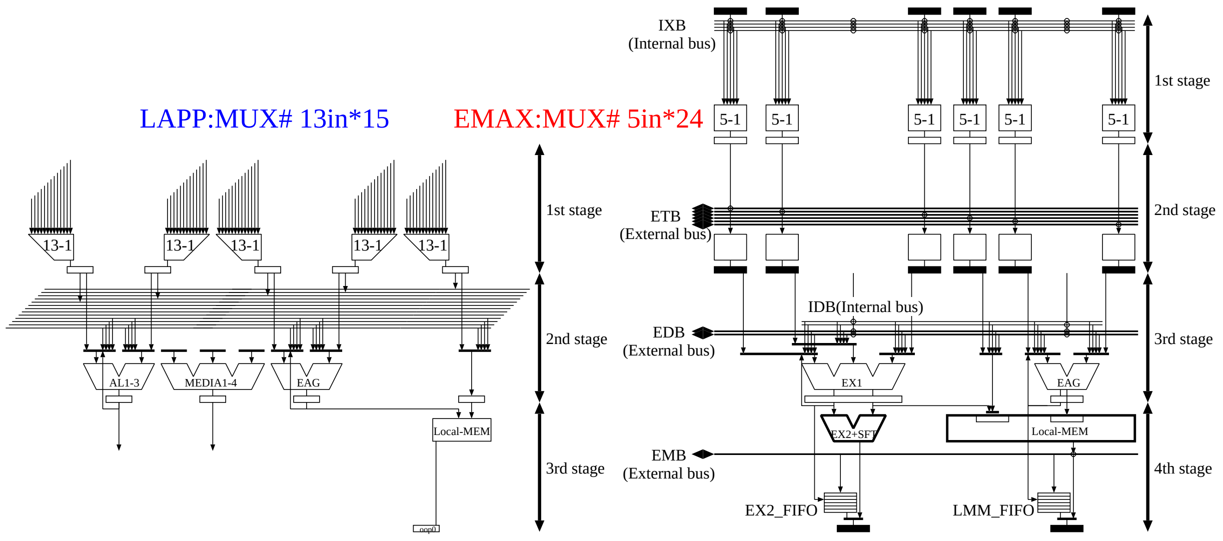


Figure.1.1: EMAX2 basic function.

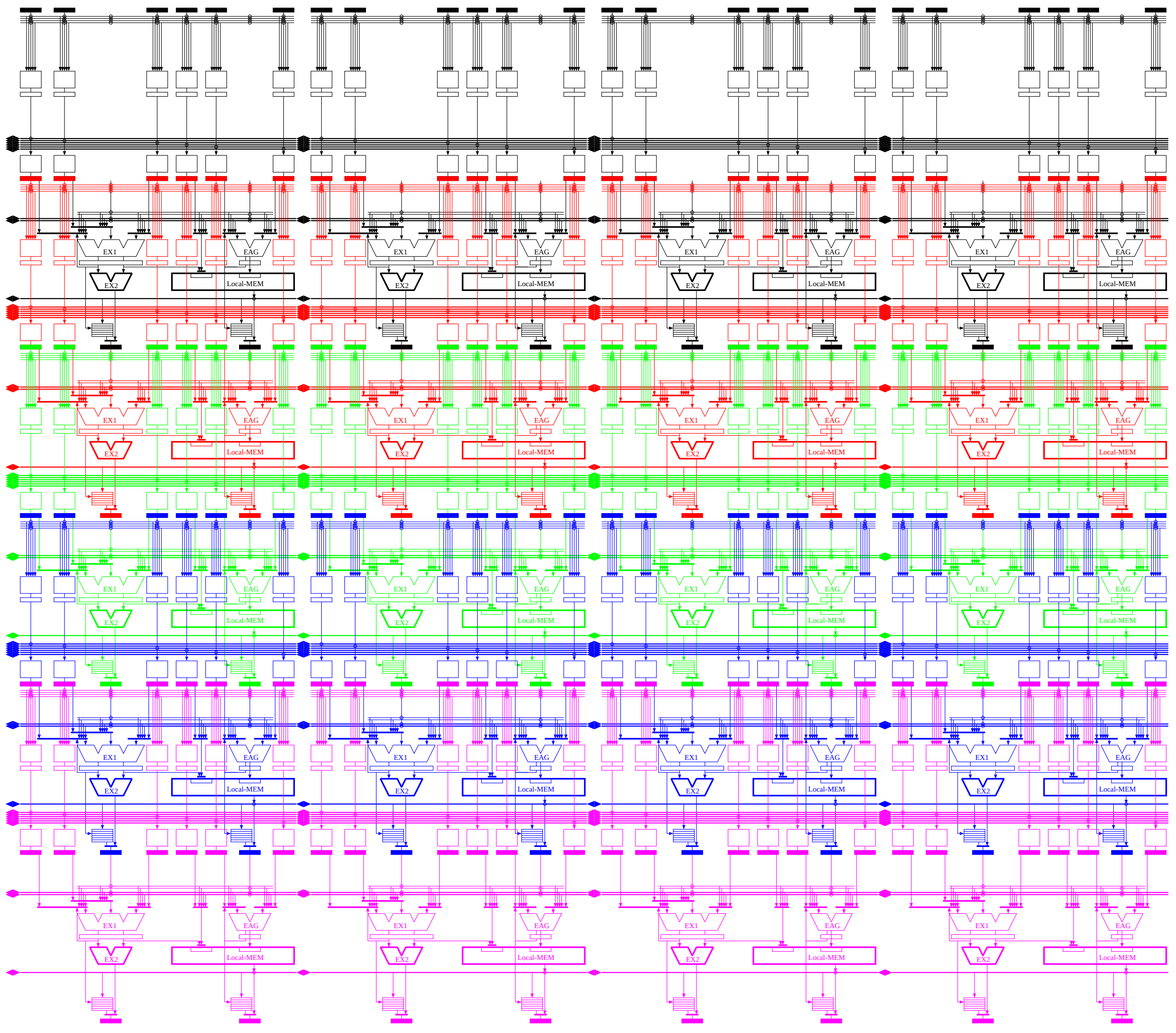


Figure.1.2: Interconnection of functions.

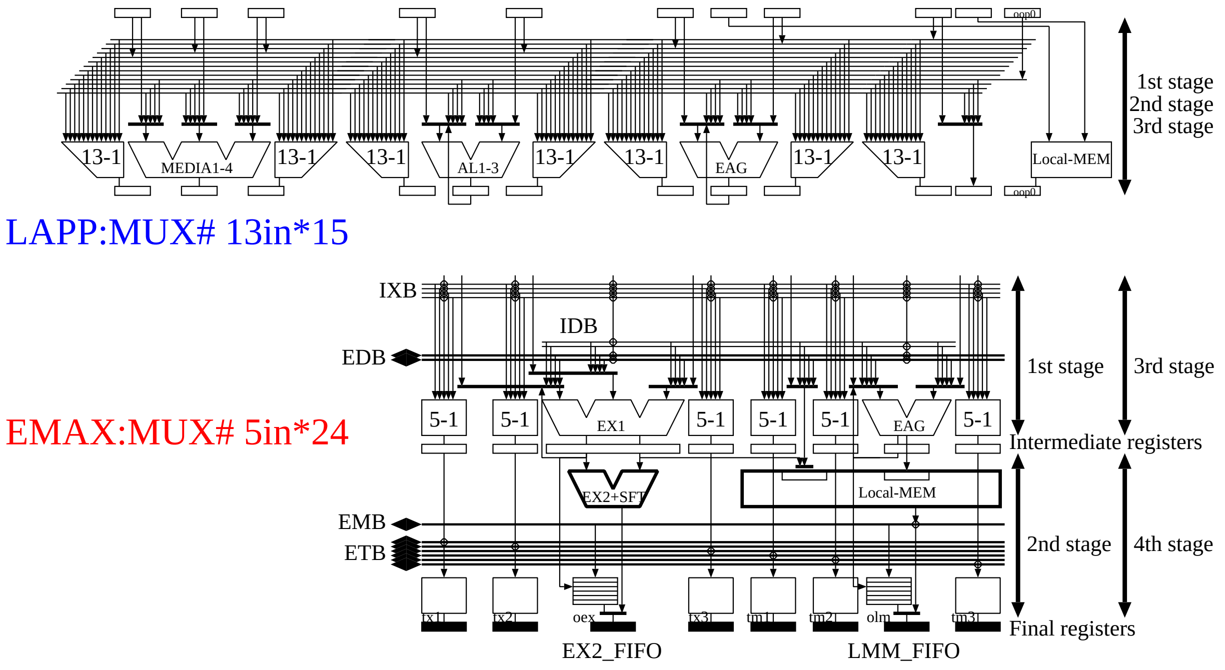


Figure.1.3: EMAX2 basic unit.

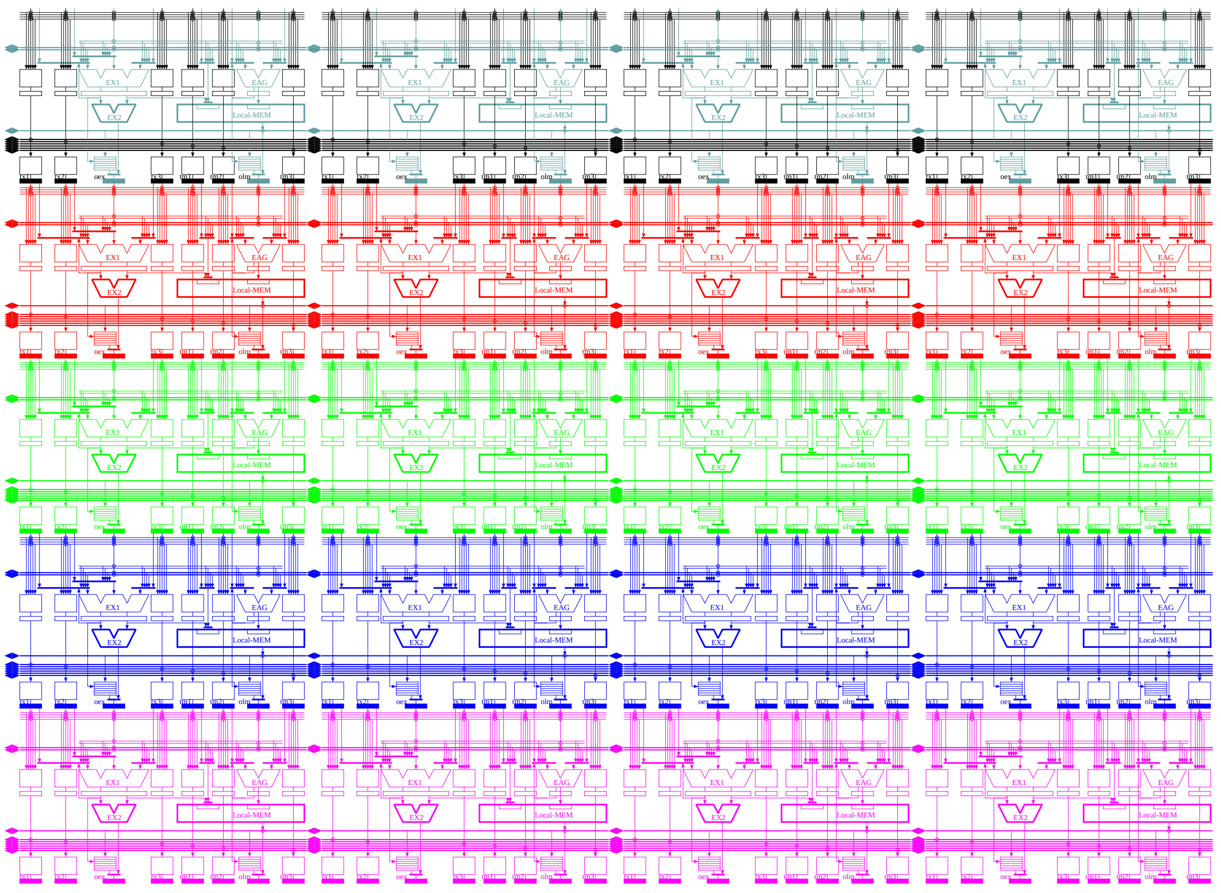


Figure.1.4: Interconnection of units.

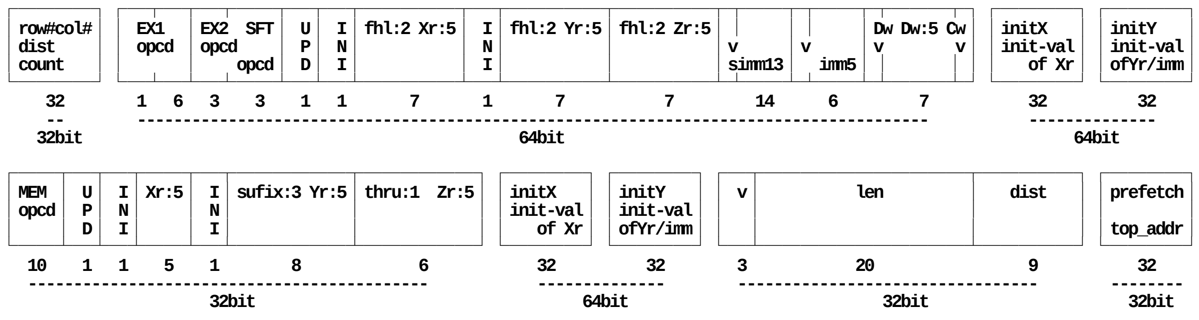
Chapter 2

Software specification

2.1 Instruction format

The instruction format for each unit is shown in Figure 2.1. Case 1 includes both of ALU operation and memory operation. Case 2 includes only ALU operation and case 3 includes only memory operation. The header specifies the target unit to be initialized, row-distance from previous instruction mapping (should be used to effectively reuse LMMs which are not affected by the row-distance) and execution count (number of cycles) the unit should work. ALU_OP specifies each operation of EX1 (3-in ALU) and EX2 (2-in ALU and 2-in shifter are cascaded), register numbers and some sort of attributes. Initial values of the registers can be specified by RGI. MEM_OP also specifies the load/store operation and the register numbers. In the same way as ALU, initial values of the registers can be specified by RGI. LMM_CONTROL specifies how to transmit data between main memory of host computer and local memory module including FIFO.

```
Case 1:  @row#,col#,dist [count] ALU_OP rgi[labelX:,labelY:] & MEM_OP rgi[labelX:,labelY:] LMM_CONTROL
Case 2:  @row#,col#,dist [count] ALU_OP rgi[labelX:,labelY:]
Case 3:  @row#,col#,dist [count]                              & MEM_OP rgi[labelX:,labelY:] LMM_CONTROL
```



```
struct insn { /* EMAX2 instruction format */
  struct header {
    Uint v            : 1;  /* insn on */
    Uint insn_row     : 6;  /* max 64 */
    Uint insn_col     : 3;  /* max  8 */
    Uint insn_dist    : 6;  /* max 64 */
    Uint count        : 16; /* max 65536 */
  } header;
  struct alu {
    Uint ex1_use_regZ : 1;
    Uint ex1_op : 6; /* 0:ex2_op use regX others:ex2_op use (-) */
    Uint ex2_op : 3;
    Uint sft_op : 3;
    Uint upd    : 1;
    Uint Xini   : 1; /* 0:noinit 1:ri     */
    Uint Xfhl   : 2; /*       1:SUFLO 2:SUFHI 3:ri/SUFFL */
    Uint Xr     : 5;
    Uint Yini   : 1; /* 0:noinit 1:ri/imm */
    Uint Yfhl   : 2; /* 0:imm 1:SUFLO 2:SUFHI 3:ri/SUFFL */
    Uint Yr     : 5;
    Uint Zfhl   : 2; /*       1:SUFLO 2:SUFHI 3:SUFFL */
    Uint Zr     : 5;
    Uint simmS_v: 1; /* 0:simmS is not used */
    Uint simmS  : 13;
    Uint immT_v : 1; /* 0:immT is not used */
    Uint immT   : 5;
    Uint Dw_v   : 1; /* 0:Dr is not used */
    Uint Dw     : 5;
    Uint Cw_v   : 1; /* 0:CC is not used */
    Uint initX;
    Uint initY;
  } alu;
  struct mem {
    Uint op     : 10;
    Uint upd    : 1;
    Uint Xini   : 1; /* 0:noinit 1:ri     */
    Uint Xr     : 5;
    Uint Yini   : 1; /* 0:noinit 1:ri/imm */
    Uint Ysuffix: 3; /* 0:imm 1:SUFLO 2:SUFHI 3:SUFFL 4:SUFB0 5:SUFB1 6:SUFB2 7:SUFB3 */
    Uint Yr     : 5;
    Uint Zthru  : 1; /* for EX1->store 0:none 1:thru */
    Uint Zr     : 5;
    Uint initX;
    Uint initY;
  } mem;
  struct ctl {
    Uint v    : 3; /* 0:nop, 1:LMR, 2:LMW, 3:LMX, 5:LMF(force read) */
    Uint len  : 20;
    Uint dist : 9;
    Uint top;
  } ctl;
} insn[INSN_DEPTH][INSN_WIDTH]; /* 10words/unit */
```

Figure.2.1: Instruction format for each unit.

2.2 ALU_OP

The format of ALU operations is shown in Figure 2.2. EX1, EX2 and SFT are mnemonic such as add/sub/and. Xr, Yr and Zr are register numbers (r0-r31) of source operands. "−" means the output of EX1 is used as the first operand of EX2. Imm32, simm13 and imm5 are 32bit immediate, 13bit signed immediate and 5bit unsigned immediate values respectively. The combination of "−expr", "∼expr", "(expr<<expr)" and "(expr>>expr)" are also allowed. "|" indicates following mnemonic is assigned to EX2. In the case that only SFT operation is required, use `"or (Xr, 0) SFT Zr/imm5"` (type-8 and 9).

```
type-1: EX1 (Xr, Yr/imm32)     | EX2 (-,        Zr) [SFT imm5], Dw Cw
type-2: EX1 (Xr, Yr/imm32)     | EX2 (-,    simm13) [SFT Zr],   Dw Cw
type-3: EX1 (Xr, Yr/imm32)     | EX2 (-,    simm13) [SFT imm5], Dw Cw
type-4: EX1 (Xr, Yr/imm32)     | EX2 (-)            [SFT Zr],   Dw Cw
type-5: EX1 (Xr, Yr/imm32)     | EX2 (-)            [SFT imm5], Dw Cw
type-6: EX1 (Xr, Yr/imm32, Zr) | EX2 (-,    simm13) [SFT imm5], Dw Cw
type-7: EX1 (Xr, Yr/imm32, Zr) | EX2 (-)            [SFT imm5], Dw Cw
type-8:                        | EX2 (Xr, Yr/imm32) [SFT Zr],   Dw Cw
type-9:                        | EX2 (Xr, Yr/imm32) [SFT imm5], Dw Cw
type-a:                        | EX2 (Xr)           [SFT Zr],   Dw Cw
type-b:                        | EX2 (Xr)           [SFT imm5], Dw Cw
type-c: EX1 (Xr, Yr/imm32)    , Dw Cw
type-d: EX1 (Xr, Yr/imm32, Zr), Dw Cw
type-e: EX1 (Xr, Yr/imm32)
type-f: EX1 (Xr, Yr/imm32, Zr)
```

Figure.2.2: Format of ALU operation.

Dw and Cw are register numbers (r0-15 and c0) of destination registers. If Dw is omitted, no register is updated (type-e and f). This instruction format is used to pass the result of EX1 to the input of a store operation (one of memory operations). In the same way, if Cw is omitted, no condition code register is updated. Condition code register (only c0 is available) has 4bit information (Negative, Zero, oVerflow and Carry) and is updated only by add or sub instruction.

In the case that a floating-point or load operation is specified in EX1, no EX2/SFT operation can be specified. EX1, EX2 and SFT operations are listed in Table 2.1, Table 2.2 and Table 2.3 respectively.

Table.2.1: EX1 operations

32bit operations		
add	(Xr[+=], Yr/imm32)	Xr+Yr or Xr+imm32 (∗)
add3	(Xr[+=], Yr/imm32, Zr)	Xr+Yr+Zr or Xr+imm32+Zr (∗)
sub	(Xr[+=], Yr/imm32)	Xr-Yr or Xr-imm32 (∗)
sub3	(Xr[+=], Yr/imm32, Zr)	Xr-Yr-Zr or Xr-imm32-Zr (∗)
16bit[2] operations		
mauh	(Xr.{fhl}, Yr.{fhl})	16bit[2] Xr+Yr (†)
mauh3	(Xr.{fhl}, Yr.{fhl}, Zr.{fhl})	16bit[2] Xr+Yr+Zr (†)
msuh	(Xr.{fhl}, Yr.{fhl})	16bit[2] Xr-Yr (†)
msuh3	(Xr.{fhl}, Yr.{fhl}, Zr.{fhl})	16bit[2] Xr-Yr-Zr (†)
misc operations		
mluh	(Xr.{fhl}, Yr.{fhl})	10bit[2]*9bit ⇒ 16bit[2]
mmrg3	(Xr, Yr, Zr)	merge Xr.byte3 | Yr.byte2 | Zr.byte1 | 0
msad	(Xr, Yr)	sum-of-absolute-difference 8bit[4] ⇒ 16bit[2]
minl	(Xr, Yr)	select Xr or Yr based on min(Xr.L16bit,Yr.L16bit)
minl3	(Xr, Yr, Zr)	merge min(Zr.H16bit,Zr.L16bit) and Xr.H16bit or Yr.H16bit
mh2bw	(Xr, Yr)	merge sat(Xr.H16bit),sat(Xr.L16bit),sat(Yr.H16bit),sat(Yr.L16bit) (‡)
mcas	(Xr, Yr)	(Xr<Yr) ? 0 : 0xff
mmid3	(Xr, Yr, Zr)	bytewise compare and collect middle value
mmax	(Xr, Yr)	bytewise compare and collect maximum value
mmax3	(Xr, Yr, Zr)	bytewise compare and collect maximum value
mmin	(Xr, Yr)	bytewise compare and collect minimum value
mmin3	(Xr, Yr, Zr)	bytewise compare and collect minimum value
load from FIFO		
ldb	(Xr[+=], Yr/imm32)	load signed byte from EX2_FIFO
ldub	(Xr[+=], Yr/imm32)	load unsigned byte from EX2_FIFO
ldh	(Xr[+=], Yr/imm32)	load signed half from EX2_FIFO
lduh	(Xr[+=], Yr/imm32)	load unsigned half from EX2_FIFO
ld	(Xr[+=], Yr/imm32)	load word from EX2_FIFO
floating-point operations		
fmul	(Xr, Yr)	floating-point multiply
fma3	(Xr, Yr, Zr)	floating-point multiply and add
fadd	(Xr, Yr)	floating-point add

(∗) += ... Get source from previous result of EX1 after first cycle. rgi[ ] should also be specified.

(†) {fhl} ... f:fullword h:byte3,byte2 ⇒ H16bit,L16bit l:byte1,byte0 ⇒ H16bit,L16bit

(‡) sat ... Saturate 16bit ⇒ 8bit.

Table.2.2: EX2 operations

32bit operations		
and	(-/Xr, Zr/simm13/Yr/imm32)	Xr and Zr/simm13/Yr/imm32
or	(-/Xr, Zr/simm13/Yr/imm32)	Xr or Zr/simm13/Yr/imm32
xor	(-/Xr, Zr/simm13/Yr/imm32)	Xr xor Zr/simm13/Yr/imm32
16bit[2] operations		
sumh	(-/Xr)	H16bit+L16bit $\Rightarrow$ H16bit
suml	(-/Xr)	H16bit+L16bit $\Rightarrow$ L16bit

Table.2.3: SFT operations

shift operations		
$<<$	Zr/imm5	logical shift to left
$>>$	Zr/imm5	logical shift to right
$>M$	Zr/imm5	logical shift high/low 16bit to right
$>A$	Zr/imm5	arithmetic shift to right (original bit31 is sign extended)
$>B$	Zr/imm5	arithmetic shift to right (original bit23 is sign extended)
$>C$	Zr/imm5	arithmetic shift to right (original bit15 is sign extended)
$>D$	Zr/imm5	arithmetic shift to right (original bit07 is sign extended)

2.3 MEM_OP

Memory operations are listed in Table 2.4.

Table.2.4: Memory operations

load from LMM or LMM_FIFO		
ldb	(Xr[+=], Yr.suffix/imm32), Dw	load signed byte from LMM or LMM_FIFO (∗)(†)
ldub	(Xr[+=], Yr.suffix/imm32), Dw	load unsigned byte from LMM or LMM_FIFO (∗)(†)
ldh	(Xr[+=], Yr.suffix/imm32), Dw	load signed half from LMM or LMM_FIFO (∗)(†)
lduh	(Xr[+=], Yr.suffix/imm32), Dw	load unsigned half from LMM or LMM_FIFO (∗)(†)
ld	(Xr[+=], Yr.suffix/imm32), Dw	load word from LMM or LMM_FIFO (∗)(†)
store to LMM		
stb	-/Zr, (Xr[+=], Yr.suffix/imm32)	store byte to LMM (∗)(†)
sth	-/Zr, (Xr[+=], Yr.suffix/imm32)	store half to LMM (∗)(†)
st	-/Zr, (Xr[+=], Yr.suffix/imm32)	store word to LMM (∗)(†)
cst	-/Zr, (Xr[+=], Yr.suffix/imm32)	if (c0.Z==1) store word to LMM (∗)(†)

(∗) += ... Get source from previous result of EAG after first cycle. rgi[ ] should also be specified.
(†) suffix ... f:fullword h:H16bit l:L16bit 3:byte3 2:byte2 1:byte1 0:byte0

2.4 RGI

Each of ALU_OP and MEM_OP has dedicated rgi[labelX:, labelY:] section for initializing Xr and Yr. RGI is just used for inserting labels at the location of initX and initY in each instruction, so that host computer can identify the location where some initial value should be set before the instructions are sent to EMAX2 device driver. Then, labelX and labelY should be unique name among the program. Notice that even when same constant values should be set to several places, each location should have different label each other.

In the case of register with an initial value, no register number is required because there is no register dependency between such type of register and previous instructions. For saving the register numbers, "ri" can be used instead of "r0-31". Typical usage of "ri" is `"ld (ri+=,4),r0 rgi[array_A_minus_4:,]"` for sequential load starting at address array_A. Notice that "the address of array_A minus 4" should be stored at label "array_A_minus_4:" because the first result of EAG is "ri+4"

2.5 LMM_CONTROL

LMM_CONTROL specifies the type, start address, length, distance and some timing adjustment information for transmitting data between main memory of host computer and local memory module including FIFO. The format of LMM_CONTROL depends on the type field which should be one of "lmr", "lmw" , "lmx" or "lmf" (lmp and lmd are reserved for future extention).

lmr [**label:, len, dist**]
This format is used for reading data from main memory of host computer to local memory module including FIFO. "label:" corresponds to the start address of an array in main memory. "len" is the number of words to prefetch. "dist" expresses the distance between elements of array by 4<<dist. In the case of normal array with single words, dist should be 0.

lmw [**label:, len, dist**]
This format is used for writing data from local memory module to main memory of host computer

after array execution completes.

lmx [**label:, len, dist**]
: This format is used when the local memory module should execute both of reading and writing.

lmf [**label:, len, dist**]
: This format is the same as “lmr” except the data in LMM is never reused, so that to read the latest value of the same location.

lmp [**label:, len, dist**]
: This format is used for overlapping array execution and prefetching (reserved for future extension: macro-pipelining).

lmd [**label:, len, dist**]
: This format is used to write back (drain) LMM to host memory (reserved for future extension: macro-pipelining).

For the correct and effective control of FIFO, it is important to carefully combine a base register number (Xr) of load operation, rgi specification and lmr/lmf.

```
@2,0,0 [320] ld (r12,0),r0 & ld (r12+=,4),r31 rgi[A_minus_4:,] lmr[A:,320,0]
@2,1,0 [320]               & ld (r12,  4),r1
@2,2,0 [320]               & ld (r12,  8),r2
@2,3,0 [320]               & ld (r12, 12),r3
```

In the case of above instructions, unit@2,1, unit@2,2, unit@2,3 have no lmr specification. The load instruction without lmr is scheduled to get data from neighbor LMM through EMB and FIFO. The appropriate source LMM is determined only by the base register number (r12). In this case, the memory operation in unit@2,0 should be "`ld (r12+=,4)`", because "`ld (ri+=,4)`" gives no information about connecting LMM and FIFOs. However, r12 in "`ld (r12+=,4)`" with rgi specification does not mean the value of previous r12. It just works as a marker to show the LMM is a source for FIFOs of neighbor units using the same base register r12. Note that the value of the base register r12 in the load instruction without rgi specification is supplied from previous r12 (different from the value of r12 in "`ld (r12+=,4)`").

2.6 Hints for sophisticated use of hardware

Following is an example of image processing. The EAG part of each load and the destination register of each sad are omitted to focus on the register dependency between load and sad. In this case, each result of load instructions in @2,X is referred by sad at different column in @5,X. EMAX2 tries to route the dependency using several ETBs in @4,X. However, the routing is failed because each ETB is dedicated to the same portion to simplify the switching network and only a path can be used to route the second operand of sad (Yr). If "`sad (r1,r26)`" is changed to "`sad (r26,r1)`", other ETB in @4,X is used and the routing is successfully completed.

```
@2,3,0 ld->r28 & ld  ->r27     @2,2 ld  ->r26     @2,1 ld  ->r25     @2,0 ld  ->r24
@3,3,0           ld  ->r3      @3,2 ld  ->r2      @3,1 ld  ->r1      @3,0 ld  ->r0
@4,3,0           sad (r3,r27)  @4,2 sad (r2,r26)  @4,1 sad (r1,r25)  @4,0 sad (r0,r24)
@5,3,0           sad (r3,r28)  @5,2 sad (r2,r27)  @5,1 sad (r1,r26)  @5,0 sad (r0,r25)
```

However, following is more sophisticated scheduling. This example swaps @2.X and @3.X, @4.X and @5.X respectively. This means cross-column dependency should be put into adjacent row to save ETB resource.

```
@2,3,0          ld  ->r3      @2,2 ld  ->r2      @2,1 ld  ->r1      @2,0 ld  ->r0
@3,3,0 ld->r28 & ld  ->r27     @3,2 ld  ->r26     @3,1 ld  ->r25     @3,0 ld  ->r24
@4,3,0          sad (r3,r28)  @4,2 sad (r2,r27)  @4,1 sad (r1,r26)  @4,0 sad (r0,r25)
@5,3,0          sad (r3,r27)  @5,2 sad (r2,r26)  @5,1 sad (r1,r25)  @5,0 sad (r0,r24)
```

2.7 Application Binary Interface

Application programs should be written according to the interface provided by EMAX2 device driver. Following is a typical style for calling EMAX. "call emax2_start" is a function leading to pwritev(usb, iov, iovcnt, 0x10000100) system_call. The first argument (.emax_wvect) is the address of write vector and the second argument (2) is the number of elements in the vector. The first element of write vector is a pair of the start address of EMAX2 instructions (.emax_wctrl) and its length in bytes (.emax_wctrl_end-.emax_wctrl). The second element of write vector is a pair of the start address of data to be transmitted to EMAX2 and its length (320*4). Additional pair of the start address of read vector (.emax_rvect) and the number of elements (1) should be attached next to the final element of write vector, so that EMAX2 can setup some address translation information for writing the result back to the host computer through a buffer memory (DDR3 located between LMM and host computer is smaller than the main memory).

"call emax2_end" is a function leading to preadv(usb, iov, iovcnt, 0x10000100+read_vect_offset) system_call. The first argument (.emax_rvect) is the address of read vector and the second argument (1) is the number of elements in the vector. The first element of read vector is a pair of the start address of data to be transmitted from EMAX2 and its length (320*4).

```
        movl  8(%ebp),%eax
        movl  %eax,.emax_mmr04
        movl  %eax,.emax_lmr04
        movl  %eax,.emax_mmw08
        movl  %eax,.emax_lmw08
        addl  $-4,%eax
        movl  %eax,.emax_rgi04
        movl  %eax,.emax_rgi08

        pushl $2
        pushl $.emax_wvect
        call  emax2_start
        addl  $8, %esp

        pushl $1
        pushl $.emax_rvect
        call  emax2_end
        addl  $8, %esp

        .data
        .p2align 4
.emax_wvect: .long .emax_wctrl, .emax_wctrl_end-.emax_wctrl
.emax_mmr04: .long 0x00000000, 320*4
.emax_rvtlb: .long .emax_rvect, 1
.emax_wctrl:
//EMAX2 @3,0 [320]               & ld (ri+=,4),r11 rgi[.emax_rgi04:,] lmr[.emax_lmr04:,320,0]
//EMAX2 @4,0 [320] minl (r10,r11) & st -,(ri+=,4)   rgi[.emax_rgi08:,] lmw[.emax_lmw08:,320,0]
.emax_wctrl_end:
.emax_rvect:
.emax_mmw08: .long 0x00000000, 320*4
```

2.8 Configuration data

The binary data of the instructions is not directly sent to each unit. Each unit requires lower level information representing selection signal of each selectors and tri-state buffers as shown in Figure 2.3 and Figure 2.4, so that no complicated instruction decoder is required on each unit. Such binary translation from the instructions to the configuration data will be executed in EMAX2 device driver which can memorize many configuration data and reuse them to speed up the translation.

```
struct conf { /* final information for EMAX2 hardware */
  /* struct ixbc: select any portion in the same unit */
  Uint ixbc0_sel_r : 4;  /* 0:off 1:ex2.p1 2:ex2.p2 3:ex2.p3 4:mem.p1 5:mem.p2 6:mem.p3 8:ex2.d 9:mem.d */
  Uint ixbc1_sel_r : 4;  /* 0:off 1:ex2.p1 2:ex2.p2 3:ex2.p3 4:mem.p1 5:mem.p2 6:mem.p3 8:ex2.d 9:mem.d */
  Uint ixbc2_sel_r : 4;  /* 0:off 1:ex2.p1 2:ex2.p2 3:ex2.p3 4:mem.p1 5:mem.p2 6:mem.p3 8:ex2.d 9:mem.d */
  Uint ixbc3_sel_r : 4;  /* 0:off 1:ex2.p1 2:ex2.p2 3:ex2.p3 4:mem.p1 5:mem.p2 6:mem.p3 8:ex2.d 9:mem.d */
  /* struct ixcc: select any portion in the same unit */
  Uint ixcc0_sel_r : 4;  /* 0:off 7:prev_mem.pc 10:prev_ex2.c */
  /* struct idbc: dedicated to prev_ex2.d and prev_mem.d */
  Uint idbc0_sel_r : 1;  /* 0:off 1:on */
  Uint idbc1_sel_r : 1;  /* 0:off 1:on */
  /* struct idcc: dedicated to prev_ex2.d and prev_mem.d */
  Uint idcc0_sel_r : 1;  /* 0:off 1:on */
  /* struct edbc: select dst portion among neighbor units */
  Uint edbc0_sel_r : 4;  /* 0:off 8:prev_ex2.d 9:prev_mem.d */
  Uint edbc0_dir_r : 2;  /* 0:off 1:to-left 2:to-right 3:inhibited */
  Uint _dmy0       : 3;

  Uint edbc1_sel_r : 4;  /* 0:off 8:prev_ex2.d 9:prev_mem.d */
  Uint edbc1_dir_r : 2;  /* 0:off 1:to-left 2:to-right 3:inhibited */
  /* struct edcc: select dst portion among neighbor units */
  Uint edcc0_sel_r : 4;  /* 0:off 10:prev_ex2.c */
  Uint edcc0_dir_r : 2;  /* 0:off 1:to-left 2:to-right 3:inhibited */
  Uint ex1c_s1_r   : 3;  /* 5to1 selector 0:prev_p1 1:self_loop 2:idb0 3:idb1 4:edb0 5:edb1 */
  Uint ex1c_s1_fhl : 2;  /*       1:SUFLO 2:SUFHI 3:ri/SUFFL */
  Uint ex1c_s2_r   : 3;  /* 5to1 selector 0:prev_p2             2:idb0 3:idb1 4:edb0 5:edb1 */
  Uint ex1c_s2_fhl : 2;  /* 0:imm 1:SUFLO 2:SUFHI 3:ri/SUFFL */
  Uint ex1c_s3_r   : 3;  /* 5to1 selector 0:prev_p3             2:idb0 3:idb1 4:edb0 5:edb1 */
  Uint ex1c_s3_fhl : 2;  /*       1:SUFLO 2:SUFHI 3:ri/SUFFL */
  Uint ex1c_urZ_r  : 1;  /* opcd-extension */
  Uint _dmy1       : 4;

  /* struct ex1c */
  Uint ex1c_op_r   : 6;  /* ex1_opcd */
  Uint ex1c_px1_r  : 2;  /* 0:off 2:s2 */
  Uint ex1c_px2_r  : 2;  /* 0:off 3:s3 */
  Uint ex1c_x1_r   : 3;  /* 5to1 selector 0:prev_p[p] 4:ixb0 5:ixb1 6:ixb2 7:ixb3 */
  Uint ex1c_x2_r   : 3;  /* 5to1 selector 0:prev_p[p] 4:ixb0 5:ixb1 6:ixb2 7:ixb3 */
  Uint ex1c_x3_r   : 3;  /* 5to1 selector 0:prev_p[p] 4:ixb0 5:ixb1 6:ixb2 7:ixb3 */
  /* struct ex2c */
  Uint ex2c_simmS_r:13;

  Uint ex2c_immT_r : 5;
  Uint ex2c_s1_r   : 3;  /* 0:d_r(ex1) 4:dx1_r(ex1) 5:dx2_r(ex1) 6:simmS_r 7:immT_r */
  Uint ex2c_s2_r   : 3;  /* 0:d_r(ex1) 4:dx1_r(ex1) 5:dx2_r(ex1) 6:simmS_r 7:immT_r */
  Uint ex2c_s3_r   : 3;  /* 0:d_r(ex1) 4:dx1_r(ex1) 5:dx2_r(ex1) 6:simmS_r 7:immT_r */
  Uint ex2c_op_r   : 3;  /* ex2_opcd */
  Uint ex2c_sft_r  : 3;  /* sft_opcd */
  Uint ex2c_dsel_r : 1;  /* ex2-selector 0:ex2 direct 1:fifo */
  Uint ex2c_x1_r   : 2;  /* ex2-output-selector 0:fixed_for_constant 1:t1_direct 2:from etb[] */
  Uint ex2c_x2_r   : 2;  /* ex2-output-selector 0:fixed_for_constant 1:t2_direct 2:from etb[] */
  Uint ex2c_x3_r   : 2;  /* ex2-output-selector 0:fixed_for_constant 1:t3_direct 2:from etb[] */
  Uint _dmy3       : 5;
```

Figure.2.3: Lower level configuration data (1/2).

```
  /* struct eagc */
  Uint eagc_s1_r   : 3;  /* 5to1 selector 0:prev_p1 1:self_loop 2:idb0 3:idb1 4:edb0 5:edb1 */
  Uint eagc_s2_r   : 3;  /* 5to1 selector 0:prev_p2             2:idb0 3:idb1 4:edb0 5:edb1 */
  Uint eagc_s2_suffix:3; /* 0:imm 1:SUFLO 2:SUFHI 3:SUFFL 4:SUFB0 5:SUFB1 6:SUFB2 7:SUFB3 */
  Uint eagc_s3_r   : 3;  /* 5to1 selector 0:prev_p3             2:idb0 3:idb1 4:edb0 5:edb1 */
  Uint eagc_sc_r   : 3;  /* 3to1 selector 0:prev_c              2:idc0        4:edc0        */
  Uint eagc_op_r   : 10; /* mem_opcd */
  Uint eagc_x1_r   : 3;  /* 5to1 selector 0:prev_p[p] 4:ixb0 5:ixb1 6:ixb2 7:ixb3 */
  Uint eagc_x2_r   : 3;  /* 5to1 selector 0:prev_p[p] 4:ixb0 5:ixb1 6:ixb2 7:ixb3 */
  Uint _dmy4       : 1;

  Uint eagc_x3_r   : 3;  /* 5to1 selector 0:prev_p[p] 4:ixb0 5:ixb1 6:ixb2 7:ixb3 */
  Uint eagc_xc_r   : 1;  /* 2to1 selector 0:prev_p[p] 1:ixc0 */
  /* struct lmmc */
  Uint lmmc_ssel_r : 1;  /* lmem-selector */
  Uint lmmc_dsel_r : 1;  /* lmm-selector 0:lmm direct 1:fifo */
  Uint lmmc_x1_r   : 2;  /* mem-output-selector 0:fixed_for_constant 1:t1_direct 2:from etb[] */
  Uint lmmc_x2_r   : 2;  /* mem-output-selector 0:fixed_for_constant 1:t2_direct 2:from etb[] */
  Uint lmmc_x3_r   : 2;  /* mem-output-selector 0:fixed_for_constant 1:t3_direct 2:from etb[] */
  Uint lmmc_xc_r   : 2;  /* mem-output-selector                      1:tc_direct 2:from etc[] */
  Uint lmmc_pc_r   : 4;  /* mem output(data) */
  /* struct embc: select memory portion among neighbor units */
  Uint embc0_sel_r : 1;  /* 0:off 1:lmem */
  Uint embc0_dir_r : 2;  /* 0:off 1:to-left    2:to-right   3:inhibited */
  Uint etbc0_sel_r : 1;  /* 0:off 1:t[1-3] */
  Uint etbc0_dir_r : 2;  /* 0:off 1:to-left    2:to-right   3:inhibited */
  Uint etbc1_sel_r : 1;  /* 0:off 1:t[1-3] */
  Uint etbc1_dir_r : 2;  /* 0:off 1:to-left    2:to-right   3:inhibited */
  Uint etbc2_sel_r : 1;  /* 0:off 1:t[1-3] */
  Uint etbc2_dir_r : 2;  /* 0:off 1:to-left    2:to-right   3:inhibited */
  Uint _dmy5       : 2;

  Uint etbc3_sel_r : 1;  /* 0:off 1:t[1-3] */
  Uint etbc3_dir_r : 2;  /* 0:off 1:to-left    2:to-right   3:inhibited */
  Uint etbc4_sel_r : 1;  /* 0:off 1:t[1-3] */
  Uint etbc4_dir_r : 2;  /* 0:off 1:to-left    2:to-right   3:inhibited */
  /* struct etbc: select same portion among neighbor units */
  Uint etbc5_sel_r : 1;  /* 0:off 1:t[1-3] */
  Uint etbc5_dir_r : 2;  /* 0:off 1:to-left    2:to-right   3:inhibited */
  /* struct etcc: select same portion among neighbor units */
  Uint etcc0_sel_r : 1;  /* 0:off 1:c */
  Uint etcc0_dir_r : 2;  /* 0:off 1:to-left    2:to-right   3:inhibited */
  Uint _dmy6       : 20;

  Uint v1          : 1; /* unit 1/2 (ex1/eag) on */
  Uint v2          : 1; /* unit 2/2 (ex2/lmm) on */
  Uint dist        : 6; /* unit_map distance */
  Uint count       : 16;
  Uint _dmy7       : 8;
} conf[UNIT_DEPTH][UNIT_WIDTH]; /* 221bit(8words)/unit */
```

Figure.2.4: Lower level configuration data (2/2).

2.9 Initial Values of Registers

Unlike the configuration data, the initial values of registers should be transmited to EMAX2 every time before starting execution. The initial values of registers shown in Figure 2.5 are calculated by EMAX2 device driver and transmitted to EMAX2 with other source data.

```
struct regv { /* final information for EMAX2 hardware */
  struct ex2v {
    /* inputs are connected to ex1.d*_r */
    Uint p1_r  : 32; /* ex2 output(prop) */
    Uint p2_r  : 32; /* ex2 output(prop) */
  } ex2v; /* 64bit */

  struct lmmv {
    Uint p1_r  : 32; /* mem output(prop) */
    Uint p2_r  : 32; /* mem output(prop) */
  } lmmv; /* 64bit */
} regv[UNIT_DEPTH][UNIT_WIDTH]; /* 128bit(4words)/unit */
```

Figure.2.5: Initial Values of Registers.

2.10 LMM Information

Unlike the configuration data, LMM information should be transmited to EMAX2 every time before starting execution. LMM information shown in Figure 2.6 are calculated by EMAX2 device driver and transmitted to EMAX2 with other source data.

```
struct lmmi {
  struct ctl ctl; /* 2words */

  struct ddr3_tlb { /* DDR3 address translation (aligned by DDR3_MINALIGN) */
    Uint v      :  2; /* 0:nop, 1:LMR, 2:LMW, 3:LMX */
    Uint msksft:  4;
    Uint base   : 12; /* ddr3 = (ddr3_base*DDR3_MINALIGN)|(intel_addr&((DDR3_MINALIGN<<ddr3_msksft)-1)) */
    Uint _dmy1 : 14;
    Uint _dmy2 : 32;
  } ddr3_tlb; /* 2word */
} lmmi[UNIT_DEPTH][UNIT_WIDTH]; /* 128bit(4words)/unit */
```

Figure.2.6: LMM Information.

Chapter 3

Examples

3.1 tone_curve

```
  for(i=0; i<HT; i++)
    tone_curve( &R[i*WD], &D[i*WD], lut );
//-----------------------------------------------------------------------------
        .text
        .p2align 2
        .globl  tone_curve
        .type   tone_curve,@function
tone_curve: // void tone_curve(r, d, t) unsigned int *r, *d; unsigned char *t;
// int j;
// for (j=0; j<WD; j++) {
//   *d = ((t)[*r>>24])<<24 | (t[256+((*r>>16)&255)])<<16 | (t[512+((*r>>8)&255)])<<8;
//   r++; d++;
// }
        pushl %ebp
        movl  %esp,%ebp
        pushl %edi    /* reg for any */
        pushl %esi    /* reg for any */
        pushl %ebx    /* reg for any */

        movl  8(%ebp),%eax
        movl  %eax,.emax_mmf00_tone_curve /* MMFILL src Outer-Loop */
        movl  %eax,.emax_lmf00_tone_curve /* LMFILL src Inner-Loop */
        addl  $-4,%eax
        movl  %eax,.emax_rgi00_tone_curve /* RGINIT src Inner-Loop */
        movl  16(%ebp),%eax
        movl  %eax,.emax_mmr01_tone_curve /* MMFILL table Outer-Loop */
        movl  %eax,.emax_lmr01_tone_curve /* LMFILL table Inner-Loop */
        movl  %eax,.emax_rgi01_tone_curve /* RGINIT table Inner-Loop */
        addl  $256,%eax
        movl  %eax,.emax_lmr02_tone_curve /* LMFILL table Inner-Loop */
        movl  %eax,.emax_rgi02_tone_curve /* RGINIT table Inner-Loop */
        addl  $256,%eax
        movl  %eax,.emax_lmr03_tone_curve /* LMFILL table Inner-Loop */
        movl  %eax,.emax_rgi03_tone_curve /* RGINIT table Inner-Loop */
        movl  12(%ebp),%eax
        movl  %eax,.emax_mmw04_tone_curve /* MDRAIN dst Outer-Loop */
        movl  %eax,.emax_lmw04_tone_curve /* LDRAIN dst Inner-Loop */
        addl  $-4,%eax
        movl  %eax,.emax_rgi04_tone_curve /* RGINIT dst Inner-Loop */
        pushl $3
        pushl $.emax_wvect_tone_curve
        call  emax2_start
        addl  $8, %esp
        pushl $1
        pushl $.emax_rvect_tone_curve
        call  emax2_end
        addl  $8, %esp

        popl  %ebx
        popl  %esi
        popl  %edi
        leave
        ret

        .data
        .p2align 4
.emax_wvect_tone_curve: .long .emax_wctrl_tone_curve, .emax_wctrl_end_tone_curve-.emax_wctrl_tone_curve
.emax_mmf00_tone_curve: .long 0x00000000, 320*4
.emax_mmr01_tone_curve: .long 0x00000000, 768
.emax_rvtlb_tone_curve: .long .emax_rvect_tone_curve, 1
.emax_wctrl_tone_curve:
//EMAX2 @0,0,0 [320]                           & ld   (ri+=,4),r9    rgi[.emax_rgi00_tone_curve:,] lmf[.emax_lmf00_tone_curve:,320,0]
//EMAX2 @1,0,0 [320]                           & ldub (ri,r9.3),r10  rgi[.emax_rgi01_tone_curve:,] lmr[.emax_lmr01_tone_curve:, 64,0]
//EMAX2 @1,1,0 [320]                           & ldub (ri,r9.2),r11  rgi[.emax_rgi02_tone_curve:,] lmr[.emax_lmr02_tone_curve:, 64,0]
//EMAX2 @1,2,0 [320]                           & ldub (ri,r9.1),r12  rgi[.emax_rgi03_tone_curve:,] lmr[.emax_lmr03_tone_curve:, 64,0]
//EMAX2 @2,0,0 [320] mmrg3 (r10,r11,r12) rgi[,] & st   -,(ri+=,4)     rgi[.emax_rgi04_tone_curve:,] lmw[.emax_lmw04_tone_curve:,320,0]
.emax_wctrl_end_tone_curve:
.emax_rvect_tone_curve:
.emax_mmw04_tone_curve: .long 0x00000000, 320*4
```

3.2 hokan1

```
  for (i=4; i<HT-4; i++) { /* scan-lines */
    for (k=-4; k<4; k++)
      hokan1(&W[i*WD], &R[(i+k)*WD], SAD1->SAD1[i/4][k+4]);
  }
//------------------------------------------------------------------------------
        .text
        .p2align 2
        .globl  hokan1
        .type   hokan1,@function
hokan1: // void hokan1(c, p, s) unsigned int *c, *p; unsigned short *s;
// int j;
// for (j=0; j<WD; j++) {
//   int j2 = j/4*4;
//   int k = j%4*2;
//   * s    += df(c[j2],p[j2+k-4]) + df(c[j2+1],p[j2+k-3]) + df(c[j2+2],p[j2+k-2]) + df(c[j2+3],p[j2+k-1]); p[-4],p[-3],p[-2],p[-1] -> p[-2],p[-1],p[0],p[1]
//   *(s+1) += df(c[j2],p[j2+k-3]) + df(c[j2+1],p[j2+k-2]) + df(c[j2+2],p[j2+k-1]) + df(c[j2+3],p[j2+k  ]); p[-3],p[-2],p[-1],p[ 0] -> p[-1],p[ 0],p[1],p[2]
//   s += 2;
// }
        pushl %ebp
        movl  %esp,%ebp
        pushl %edi    /* reg for any */
        pushl %esi    /* reg for any */
        pushl %ebx    /* reg for any */

        movl  8(%ebp),%eax
        addl  $16,%eax
        movl  %eax,.emax_mmr00_hokan1 /* MMFILL src1 Outer-Loop */
        movl  %eax,.emax_lmr00_hokan1 /* LMFILL src1 Inner-Loop */
        addl  $-4,%eax
        movl  %eax,.emax_rgix0_hokan1 /* LMFILL src2 Inner-Loop */
        addl  $-12,%eax
        movl  %eax,.emax_rgi00_hokan1 /* RGINIT src1 Inner-Loop */
        movl  12(%ebp),%eax
        addl  $16,%eax
        movl  %eax,.emax_mmr01_hokan1 /* MMFILL src2 Outer-Loop */
        movl  %eax,.emax_lmr01_hokan1 /* LMFILL src2 Inner-Loop */
        addl  $-4,%eax
        movl  %eax,.emax_rgix1_hokan1 /* LMFILL src2 Inner-Loop */
        addl  $-12,%eax
        movl  %eax,.emax_rgi01_hokan1 /* RGINIT src2 Inner-Loop */
        movl  16(%ebp),%eax
        movl  %eax,.emax_mmf02_hokan1 /* MMFILL src3 Outer-Loop */
        movl  %eax,.emax_lmf02_hokan1 /* LMFILL src3 Inner-Loop */
        movl  %eax,.emax_mmw05_hokan1 /* MDRAIN dst Outer-Loop */
        movl  %eax,.emax_lmw05_hokan1 /* LDRAIN dst Inner-Loop */
        addl  $-4,%eax
        movl  %eax,.emax_rgi02_hokan1 /* RGINIT src3 Inner-Loop */
        movl  %eax,.emax_rgi05_hokan1 /* RGINIT dst Inner-Loop */
        movl  $-1,%eax
        movl  %eax,.emax_rgi03_hokan1 /* loop_counter */
        movl  %eax,.emax_rgi04_hokan1 /* loop_counter */
        pushl $4
        pushl $.emax_wvect_hokan1
        call  emax2_start
        addl  $8, %esp
        pushl $1
        pushl $.emax_rvect_hokan1
        call  emax2_end
        addl  $8, %esp

        popl  %ebx
        popl  %esi
        popl  %edi
        leave
        ret

        .data
        .p2align 4
.emax_wvect_hokan1: .long .emax_wctrl_hokan1, .emax_wctrl_end_hokan1-.emax_wctrl_hokan1
.emax_mmr00_hokan1: .long 0x00000000, 320*4
.emax_mmr01_hokan1: .long 0x00000000, 320*4
.emax_mmf02_hokan1: .long 0x00000000, 320*4
.emax_rvtlb_hokan1: .long .emax_rvect_hokan1, 1
.emax_wctrl_hokan1: /* +=は self_loop 指示 */
//EMAX2 @0,0,0 [320] add  (ri+=,1) | and (-,~3)<<2,r12 rgi[.emax_rgi03_hokan1:,]
//EMAX2 @0,1,0 [320] add  (ri+=,1) | and (-, 3)<<3,r13 rgi[.emax_rgi04_hokan1:,]
//EMAX2 @1,0,0 [320] add  (ri,r12),r12                 rgi[.emax_rgi00_hokan1:,]
//EMAX2 @1,1,0 [320] add3 (ri,r12,r13),r13             rgi[.emax_rgi01_hokan1:,]
//EMAX2 @2,0,0 [320]                        ld  (r12,  0),r0  & ld  (r12+=,4),r31 rgi[.emax_rgix0_hokan1:,] lmr[.emax_lmr00_hokan1:,320,0]
//EMAX2 @2,1,0 [320]                                          & ld  (r12,  4),r1
//EMAX2 @2,2,0 [320]                                          & ld  (r12,  8),r2
//EMAX2 @2,3,0 [320]                                          & ld  (r12, 12),r3
//EMAX2 @3,0,0 [320]                        ld  (r13,-16),r24 & ld  (r13+=,4),r31 rgi[.emax_rgix1_hokan1:,] lmr[.emax_lmr01_hokan1:,320,0]
//EMAX2 @3,1,0 [320]                                          & ld  (r13,-12),r25
//EMAX2 @3,2,0 [320]                                          & ld  (r13, -8),r26
//EMAX2 @3,3,0 [320]                        ld  (r13,  0),r28 & ld  (r13, -4),r27
//                          @2.3           ld  ->r3      @2.2 ld  ->r2      @2.1 ld  ->r1      @2.0 ld  ->r0
//                          @3.3 ld->r28 & ld  ->r27     @3.2 ld  ->r26     @3.1 ld  ->r25     @3.0 ld  ->r24
//                          @4.3           sad (r3,r28)  @4.2 sad (r2,r27)  @4.1 sad (r1,r26)  @4.0 sad (r0,r25)
//                          @5.3           sad (r3,r27)  @5.2 sad (r2,r26)  @5.1 sad (r1,r25)  @5.0 sad (r0,r24)
//EMAX2 @4,0,0 [320] msad (r0,r25),r11
//EMAX2 @4,1,0 [320] msad (r1,r26),r13  ! swap r1 and r26 to avoid collision of pos2
//EMAX2 @4,2,0 [320] msad (r2,r27),r15
//EMAX2 @4,3,0 [320] msad (r3,r28),r17
//EMAX2 @5,0,0 [320] msad (r0,r24),r10
//EMAX2 @5,1,0 [320] msad (r1,r25),r12
//EMAX2 @5,2,0 [320] msad (r2,r26),r14
//EMAX2 @5,3,0 [320] msad (r3,r27),r16
//EMAX2 @6,0,0 [320] mauh (r10,r12),r10
//EMAX2 @6,1,0 [320] mauh (r11,r13),r11
//EMAX2 @6,2,0 [320] mauh (r14,r16),r14
//EMAX2 @6,3,0 [320] mauh (r15,r17),r15
//EMAX2 @7,0,0 [320] mauh (r10,r14) | suml (-),r10
//EMAX2 @7,1,0 [320] mauh (r11,r15) | sumh (-),r11
//EMAX2 @7,2,0 [320]                                        & ld (ri+=,4),r0 rgi[.emax_rgi02_hokan1:,] lmf[.emax_lmf02_hokan1:,320,0]
//EMAX2 @8,0,0 [320] mauh3 (r0,r10,r11)                     & st -,(ri+=,4)  rgi[.emax_rgi05_hokan1:,] lmw[.emax_lmw05_hokan1:,320,0]
.emax_wctrl_end_hokan1:
.emax_rvect_hokan1:
.emax_mmw05_hokan1: .long 0x00000000, 320*4
```

3.3 hokan2

```
   for (i=4; i<HT-4; i+=4) { /* scan-lines */
     for (k=-4; k<4; k++)
       hokan2(SAD1->SAD1[i/4][k+4], &W[i*WD], (((k/2)&0xff)<<16)); /* 8 回走査して SAD 最小位置を求める */
   }
 //-----------------------------------------------------------------------------
         .text
         .p2align 2
         .globl  hokan2
         .type   hokan2,@function
 hokan2: //void hokan2(s, sminxy, k) unsigned short *s; unsigned int *sminxy; int k;
 // int j;
 // for (j=0; j<WD; j++) { /* j%4==0 の時のみ sminxy[j] に有効値. 他はゴミ */
 //   if ((sminxy[j]&0xffff) > *(s  )) sminxy[j] = ((-2)<<24)|k|*(s  );
 //   if ((sminxy[j]&0xffff) > *(s+1)) sminxy[j] = ((-1)<<24)|k|*(s+1);
 //   if ((sminxy[j]&0xffff) > *(s+2)) sminxy[j] = ((-1)<<24)|k|*(s+2);
 //   if ((sminxy[j]&0xffff) > *(s+3)) sminxy[j] = (( 0)<<24)|k|*(s+3);
 //   if ((sminxy[j]&0xffff) > *(s+4)) sminxy[j] = (( 0)<<24)|k|*(s+4);
 //   if ((sminxy[j]&0xffff) > *(s+5)) sminxy[j] = (( 0)<<24)|k|*(s+5);
 //   if ((sminxy[j]&0xffff) > *(s+6)) sminxy[j] = (( 1)<<24)|k|*(s+6);
 //   if ((sminxy[j]&0xffff) > *(s+7)) sminxy[j] = (( 1)<<24)|k|*(s+7);
 //   s += 2;
 // }
         pushl %ebp
         movl  %esp,%ebp
         pushl %edi    /* reg for any */
         pushl %esi    /* reg for any */
         pushl %ebx    /* reg for any */

         movl  8(%ebp),%eax
         addl  $8,%eax
         movl  %eax,.emax_mmr03_hokan2 /* MMFILL src1 Outer-Loop */
         movl  %eax,.emax_lmr03_hokan2 /* LMFILL src1 Inner-Loop */
         addl  $-12,%eax
         movl  %eax,.emax_rgi00_hokan2 /* RGINIT src1 Inner-Loop */
         addl  $4,%eax
         movl  %eax,.emax_rgi01_hokan2 /* RGINIT src1 Inner-Loop */
         addl  $4,%eax
         movl  %eax,.emax_rgi02_hokan2 /* RGINIT src1 Inner-Loop */
         addl  $4,%eax
         movl  %eax,.emax_rgi03_hokan2 /* RGINIT src1 Inner-Loop */
         movl  12(%ebp),%eax
         movl  %eax,.emax_mmf04_hokan2 /* MMFILL src2 Outer-Loop */
         movl  %eax,.emax_lmf04_hokan2 /* LMRENEW src2 Inner-Loop */
         movl  %eax,.emax_mmw08_hokan2 /* MDRAIN dst Outer-Loop */
         movl  %eax,.emax_lmw08_hokan2 /* LDRAIN dst Inner-Loop */
         addl  $-4,%eax
         movl  %eax,.emax_rgi04_hokan2 /* RGINIT src2 Inner-Loop */
         movl  %eax,.emax_rgi08_hokan2 /* RGINIT dst Inner-Loop */
         movl  16(%ebp),%eax
         movl  %eax,.emax_rgi05_hokan2 /* RGINIT src3 Inner-Loop */
         movl  %eax,.emax_rgi06_hokan2 /* RGINIT src3 Inner-Loop */
         movl  %eax,.emax_rgi07_hokan2 /* RGINIT src3 Inner-Loop */
         movl  %eax,.emax_rgi10_hokan2 /* RGINIT src3 Inner-Loop */
         movl  %eax,.emax_rgi11_hokan2 /* RGINIT src3 Inner-Loop */
         movl  %eax,.emax_rgi12_hokan2 /* RGINIT src3 Inner-Loop */
         pushl $3
         pushl $.emax_wvect_hokan2
         call  emax2_start
         addl  $8, %esp
         pushl $1
         pushl $.emax_rvect_hokan2
         call  emax2_end
         addl  $8, %esp

         popl  %ebx
         popl  %esi
         popl  %edi
         leave
         ret

         .data
         .p2align 4
 .emax_wvect_hokan2: .long .emax_wctrl_hokan2, .emax_wctrl_end_hokan2-.emax_wctrl_hokan2
 .emax_mmr03_hokan2: .long 0x00000000, 320*4
 .emax_mmf04_hokan2: .long 0x00000000, 320*4
 .emax_rvtlb_hokan2: .long .emax_rvect_hokan2, 1
 .emax_wctrl_hokan2: /* +=は self_loop 指示 */
 //EMAX2 @0,0,0 [320] |or  (ri,(-2<<24)),r28 rgi[.emax_rgi05_hokan2:,] & ld (r31+=,4),r10 rgi[.emax_rgi00_hokan2:,]
 //EMAX2 @0,1,0 [320] |or  (ri,(-1<<24)),r29 rgi[.emax_rgi06_hokan2:,] & ld (r31+=,4),r12 rgi[.emax_rgi01_hokan2:,]
 //EMAX2 @0,2,0 [320] |or  (ri,( 1<<24)),r31 rgi[.emax_rgi07_hokan2:,] & ld (r31+=,4),r14 rgi[.emax_rgi02_hokan2:,]
 //EMAX2 @0,3,0 [320]                                                  & ld (r31+=,4),r16 rgi[.emax_rgi03_hokan2:,] lmr[.emax_lmr03_hokan2:,320,0]
 //EMAX2 @1,0,0 [320] minl3 (r29,r28,r10),r10
 //EMAX2 @1,1,0 [320] minl3 (ri,r29, r12),r12 rgi[.emax_rgi10_hokan2:,]
 //EMAX2 @1,2,0 [320] minl3 (ri, ri, r14),r14 rgi[.emax_rgi12_hokan2:,.emax_rgi11_hokan2:]
 //EMAX2 @1,3,0 [320] minl3 (r31,r31,r16),r16
 //EMAX2 @2,0,0 [320] minl (r10,r12),r10
 //EMAX2 @2,2,0 [320] minl (r14,r16),r14
 //EMAX2 @3,0,0 [320] minl (r10,r14),r10                               & ld (ri+=,4),r11 rgi[.emax_rgi04_hokan2:,] lmf[.emax_lmf04_hokan2:,320,0]
 //EMAX2 @4,0,0 [320] minl (r10,r11)                                   & st -,(ri+=,4)   rgi[.emax_rgi08_hokan2:,] lmw[.emax_lmw08_hokan2:,320,0]
 .emax_wctrl_end_hokan2:
 .emax_rvect_hokan2:
 .emax_mmw08_hokan2: .long 0x00000000, 320*4
```

3.4 hokan3

```
   for (i=0; i<HT; i++) { /* scan-lines */
     for (k=-2; k<2; k++)
       hokan3(&W[(i/4*4)*WD], &R[(i+k)*WD], &D[i*WD], k);
   }
//-------------------------------------------------------------------------------
        .text
        .p2align 2
        .globl  hokan3
        .type   hokan3,@function
hokan3: //void hokan3(sminxy, r, d, k) unsigned int *sminxy; unsigned int *r, *d; int k;
// int j;
// for (j=0; j<WD; j++) {
//   int x = (int) sminxy[j/4*4]>>24;
//   int y = (int)(sminxy[j/4*4]<<8)>>24;
//   if (y == k) d[j] = r[j+x];
// }
        pushl %ebp
        movl  %esp,%ebp
        pushl %edi    /* reg for any */
        pushl %esi    /* reg for any */
        pushl %ebx    /* reg for any */

        movl  8(%ebp),%eax
        movl  %eax,.emax_mmr00_hokan3 /* MMFILL src1 Outer-Loop */
        movl  %eax,.emax_lmr00_hokan3 /* LMFILL src1 Inner-Loop */
        movl  %eax,.emax_rgi00_hokan3 /* RGINIT src1 Inner-Loop */
        movl  12(%ebp),%eax
        movl  %eax,.emax_mmr01_hokan3 /* MMFILL src2 Outer-Loop */
        movl  %eax,.emax_lmr01_hokan3 /* LMFILL src2 Inner-Loop */
        movl  %eax,.emax_rgi01_hokan3 /* RGINIT src2 Inner-Loop */
        movl  16(%ebp),%eax
        movl  %eax,.emax_mmx02_hokan3 /* RGINIT src3 Inner-Loop */
        movl  %eax,.emax_lmx02_hokan3 /* RGINIT src3 Inner-Loop */
        movl  %eax,.emax_mmx08_hokan3 /* MDRAIN dst Outer-Loop */
        addl  $-4,%eax
        movl  %eax,.emax_rgi02_hokan3 /* RGINIT src3 Inner-Loop */
        movl  20(%ebp),%eax
        movl  %eax,.emax_rgi03_hokan3 /* RGINIT src3 Inner-Loop */
        movl  $-1,%eax
        movl  %eax,.emax_rgi04_hokan1 /* loop_counter */
        movl  %eax,.emax_rgi05_hokan1 /* loop_counter */
        pushl $4
        pushl $.emax_wvect_hokan3
        call  emax2_start
        addl  $8, %esp
        pushl $1
        pushl $.emax_rvect_hokan3
        call  emax2_end
        addl  $8, %esp

        popl  %ebx
        popl  %esi
        popl  %edi
        leave
        ret

        .data
        .p2align 4
.emax_wvect_hokan3: .long .emax_wctrl_hokan3, .emax_wctrl_end_hokan3-.emax_wctrl_hokan3
.emax_mmr00_hokan3: .long 0x00000000, 320*4
.emax_mmr01_hokan3: .long 0x00000000, 320*4
.emax_mmx02_hokan3: .long 0x00000000, 320*4
.emax_rvtlb_hokan3: .long .emax_rvect_hokan3, 1
.emax_wctrl_hokan3: /* +=は self_loop 指示 */
//EMAX2 @0,0,0 [320] add (ri+=,1) |  and (-,~3)<<2,r12 rgi[.emax_rgi04_hokan3:,]
//EMAX2 @0,1,0 [320] add (ri+=,1) |  or  (-, 0)<<2,r14 rgi[.emax_rgi05_hokan3:,]
//EMAX2 @1,0,0 [320]                                                             & ld (ri,r12),r16  rgi[.emax_rgi00_hokan3:,] lmr[.emax_lmr00_hokan3:,320,0]
//EMAX2 @2,0,0 [320] add (ri,r14),r13                  rgi[.emax_rgi01_hokan3:,]
//EMAX2 @2,1,0 [320] |and (r16,0xff000000)>A22,r17  ! >A は SRA
//EMAX2 @2,2,0 [320] |and (r16,0x00ff0000)>B16,r18  ! >B は bit23 を符号拡張 >C は bit15 を符号拡張 >D は bit7 を符号拡張
//EMAX2 @3,0,0 [320] sub (r18,ri),r0 c0                rgi[,.emax_rgi03_hokan3:] & ld (r13,r17),r16 rgi[,]                    lmr[.emax_lmr01_hokan3:,320,0]
//EMAX2 @4,0,0 [320]                                                             & cst r16,(ri+=,4) rgi[.emax_rgi02_hokan3:,] lmx[.emax_lmx02_hokan3:,320,0]
.emax_wctrl_end_hokan3:
.emax_rvect_hokan3:
.emax_mmx08_hokan3: .long 0x00000000, 320*4
```

Chapter 4

Control

評価システムおよび ASIC 拡張機能のハードウェア仕様については付録を参照のこと．

4.1 LMM の仕様

図 4.1 と図 4.2 に，LMM の構造およびタイミングチャートを示す．

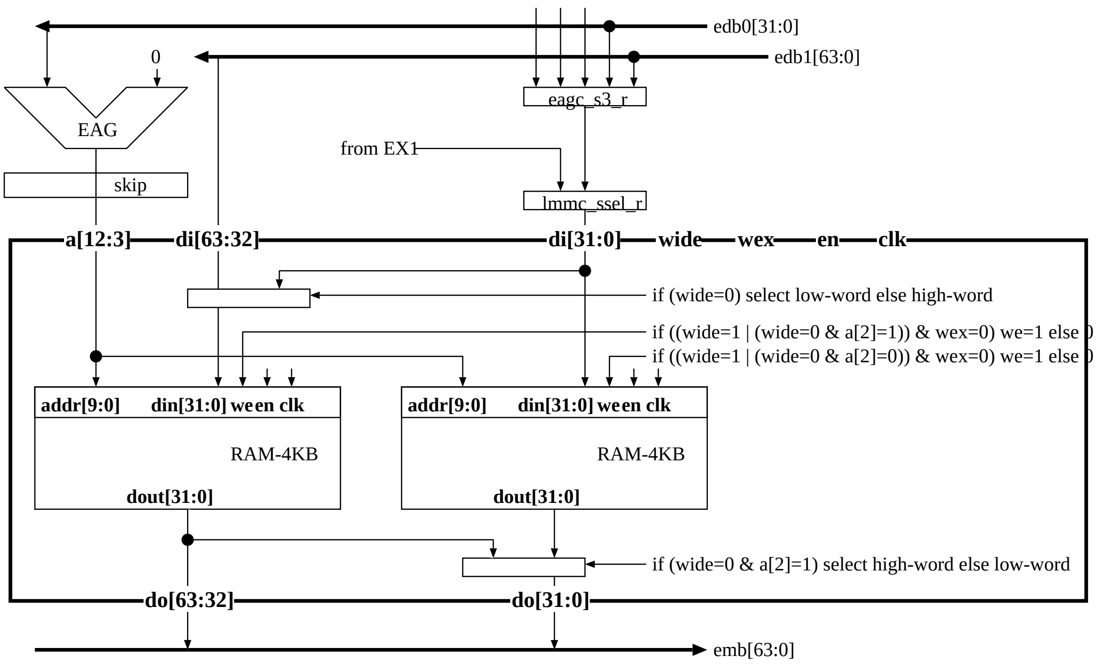

edb0[31:0]
edb1[63:0]
0
EAG
eagc_s3_r
from EX1
skip
lmmc_ssel_r
a[12:3]
di[63:32]
di[31:0]
wide
wex
en
clk
if (wide=0) select low-word else high-word
if ((wide=1 | (wide=0 & a[2]=1)) & wex=0) we=1 else 0
if ((wide=1 | (wide=0 & a[2]=0)) & wex=0) we=1 else 0
addr[9:0] din[31:0] we en clk
RAM-4KB
dout[31:0]
addr[9:0] din[31:0] we en clk
RAM-4KB
dout[31:0]
if (wide=0 & a[2]=1) select high-word else low-word
do[63:32]
do[31:0]
emb[63:0]

Figure.4.1: Structure of LMM.

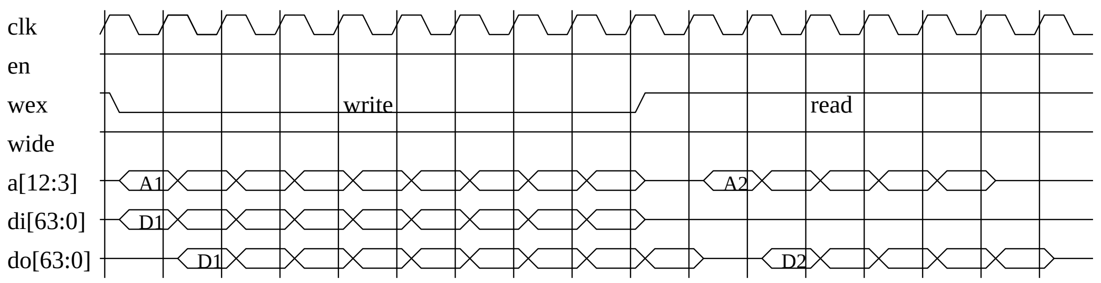


Figure.4.2: Timing chart of LMM.

4.2 使用するメモリ空間

前述した制御情報（conf[][]，regv[][]，および，lmmi[][][]）は，演算対象データ（wdata 群）とともに，HOST から DDR3 へ転送される．メモリマップを表 4.1 に示す．なお，EMAX2 の初期モデルでは，PE0 に対応する空間のみが使用される．

```
USB0

命令変換/TLB生成後
usb_space_mem_writev(usb, iov, iovcnt, ddr3_atop=10000100);
    iov[0].iov_base = ddr3_wlist[]={10002000,10004000,10006000,10100000,...}
    iov[0].iov_len  = (iovcnt-1+readiov)*4
    iov[1].iov_base = conf[][]         ->10002000
    iov[1].iov_len  = sizeof(conf)
    iov[2].iov_base = regv[][]         ->10004000
    iov[2].iov_len  = sizeof(regv)
    iov[3].iov_base = lmmi[][]         ->10006000
    iov[3].iov_len  = sizeof(lmmi)
    iov[4].iov_base = wdata1[]         ->10100000
    iov[4].iov_len  = bytes1
    iov[5].iov_base = wdata2[]         ->10200000
    iov[5].iov_len  = bytes2
                    :

usb_space_mem_pe0_l2ct0000.l2ct0000.m_stat = 3;
usb_space_mem_write(usb, &usb_space_mem_pe0_l2ct0000, 4, USB_SPACE_MEM_PE0_L2CT0000);
                                                                      0x10000078
do {
 usb_space_mem_read(usb, &usb_space_mem_pe0_l2ct0000, 4, USB_SPACE_MEM_PE0_L2CT0000);
} while (usb_space_mem_pe0_l2ct0000.l2ct0000.m_stat != 1);            0x10000078

usb_space_mem_readv(usb, iov, iovcnt, ddr3_atop=10000100+readiovoffset);
    iov[0].iov_base = ddr3_rlist[]={10300000,...} iovcnt-1
    iov[0].iov_len  = (iovcnt-1)*4
    iov[1].iov_base = rdata1[]         <-10300000
    iov[1].iov_len  = bytes1
    iov[2].iov_base = rdata2[]         <-10400000
    iov[2].iov_len  = bytes2
                    :
```

```
  0x0000003f-0x00000000:DMA制御
  0x0000017f-0x00000100:PE0.REG
[ 0x000001ff-0x00000180:PE1.REG             ] reserved
  0x1000007f-0x10000000:PE0.L2CT
[ 0x100000ff-0x10000080:PE1.L2CT            ] reserved
--0x1fffffff-0x10000100:USER---------------------------------------------------------
  0x10003fff-0x10002000:PE0.emax_wctrl(8KB)   for conf
  0x10005fff-0x10004000:PE0.emax_wregv(8KB)   for regv
  0x10007fff-0x10006000:PE0.emax_wlmmi(8KB)   for lmmi
  0x17ffffff-0x10100000:PE0.data(1MB*127)     for data
[ 0x18003fff-0x18002000:PE1.emax_wctrl(8KB) ] reserved
[ 0x18005fff-0x18004000:PE1.emax_wregv(8KB) ] reserved
[ 0x18007fff-0x18006000:PE1.emax_wlmmi(8KB) ] reserved
[ 0x1fffffff-0x18100000:PE1.data(1MB*127)   ] reserved
[ 0x2fffffff-0x20000000:other-side-DDR3     ] (unused)
```

```
USB1
  0x0000003f-0x00000000:DMA制御
[ 0x0000017f-0x00000100:PE2.REG             ] reserved
[ 0x000001ff-0x00000180:PE3.REG             ] reserved
[ 0x1000007f-0x10000000:PE2.L2CT            ] reserved
[ 0x100000ff-0x10000080:PE3.L2CT            ] reserved
--0x1fffffff-0x10000100:USER---------------------------------------------------------
[ 0x10003fff-0x10002000:PE2.emax_wctrl(8KB) ] reserved
[ 0x10005fff-0x10004000:PE2.emax_wregv(8KB) ] reserved
[ 0x10007fff-0x10006000:PE2.emax_wlmmi(8KB) ] reserved
[ 0x17ffffff-0x10100000:PE2.data(1MB*127)   ] reserved
[ 0x18003fff-0x18002000:PE3.emax_wctrl(8KB) ] reserved
[ 0x18005fff-0x18004000:PE3.emax_wregv(8KB) ] reserved
[ 0x18007fff-0x18006000:PE3.emax_wlmmi(8KB) ] reserved
[ 0x1fffffff-0x18100000:PE3.data(1MB*127)   ] reserved
[ 0x2fffffff-0x20000000:other-side-DDR3     ] PE0空間を参照可能
```

Table.4.1: 初期モデルにおけるメモリ空間の運用

4.3 各 UNIT に対する制御情報の伝達

```
struct emax2 {
  Uint pe0_status        :  4;
  Uint unit_offset       :  4; /* current mapped insn_row[0] */

  Uint unit_edb_cmd      :  3; /* 0:idle, 1:conf, 2:regv, 3:lmmi, 4:lmm_load, 5:exec, 6:lmm_drain */
  Uint unit_edb_cmd_d1   :  3; /* delay1 */
  Uint unit_edb_cmd_d2   :  3; /* delay2 */
  Uint unit_ctl_count    :  7; /* unit counter */
  Uint unit_select_row   : 16; /* bitmap 0:off 1:selected */
  Uint unit_select_col   :  4; /* bitmap 0:off 1:selected */
  Uint unit_select_row_d1: 16; /* delay1 */
  Uint unit_select_col_d1:  4; /* delay1 */
  Uint unit_edb_valid    :  1; /* edb_valid (for HDL only) */
  Uint unit_edb0         : 32; /* write_data regno/lmm_address */
  Ull  unit_edb1         : 64; /* config/write_data to unit */
  Uint unit_emb_valid    :  1; /* emb_valid (for HDL only) */
  Ull  unit_emb          : 64; /* read_data from unit */

  struct ctl      ctl_old[UNIT_DEPTH][UNIT_WIDTH];
  struct ctl      ctl_new[UNIT_DEPTH][UNIT_WIDTH];
  struct ddr3_tlb ddr3_tlb[UNIT_DEPTH][UNIT_WIDTH];

  Uint prev2_status;
  Ull  unit1_status[UNIT_WIDTH]; /* 1bit corresponds to each unit (ex1,eag), 0:stop 1:run */
  Uint prev1_status;
  Ull  unit2_status[UNIT_WIDTH]; /* 1bit corresponds to each unit (ex2,lmm), 0:stop 1:run */
} emax2;
```

Figure.4.3: General control information of EMAX2.

HOST により DDR3 に格納された制御情報を各 unit に伝達するために使用する制御信号を図 4.3 に示す．各制御信号は，pe0_status の値に応じて以下のように動作する．

4.3.1 Idle (pe0_status=STATUS_IDLE)

M_STAT が 3 でない場合は状態を維持する．M_STAT が 3 の場合，pe0_status=STATUS_CONF に遷移し，unit_edb_cmd および wrdyc（カウンタ）を 0 にリセットする．

4.3.2 Unit configuration in progress (pe0_status=STATUS_CONF)

conf[][] を元に各 unit の構成を変更する．conf[][] 再利用機構が有効である場合，DDR3 は参照せず，EMAX2 内部のシフト機構により conf[][] を再利用する（初期モデルでは本機構は無効である）．DDR3 を参照する場合，M_WEX=1，M_BSTMX=0 （バーストリード）を使用して DDR3 から conf[][] を読み出す．各 unit は，unit_edb_cmd，unit_edb_valid，行指定ビットマップ（unit_select_row）の該当行ビット，および，列指定ビットマップ（unit_select_col）の該当列ビットを監視しており，unit_edb_cmd=1，かつ，unit_edb_valid=1，かつ，該当ビットが 1 の場合に，EDB0[1:0] の内容を書き込み先構成情報レジスタグループ識別子（0-3），EDB1[63:0] の内容を構成情報レジスタ値として書き込みを行う．全ての unit に対して書き込みが完了すると，pe0_status=STATUS_REGV に遷移する．タイミングチャートを図 4.4 に示す．

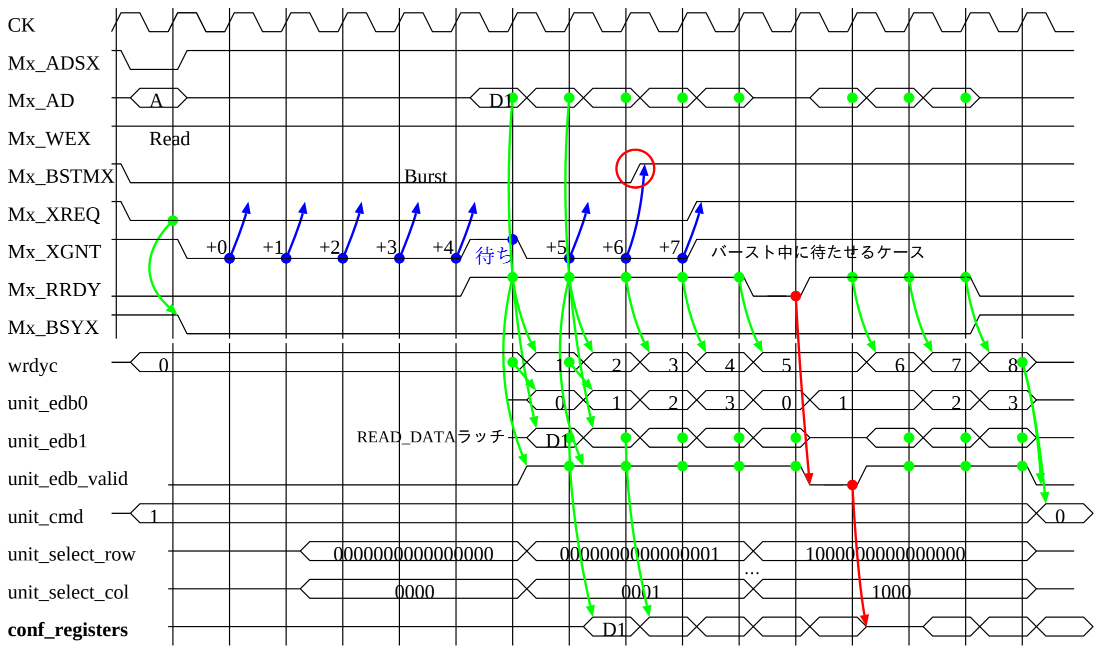


Figure.4.4: conf[][] → unit[][] のタイミングチャート

4.3.3 Register initialization in progress (pe0_status=STATUS_REGV)

regv[][] を元に各 unit のレジスタ値を初期化する．M_WEX=1，M_BSTMX=0（バーストリード）を使用して DDR3 から regv[][] を読み出す．各 unit は，unit_edb_cmd，unit_edb_valid，行指定ビットマップ（unit_select_row）の該当行ビット，および，列指定ビットマップ（unit_select_col）の該当列ビットを監視しており，unit_edb_cmd=2，かつ，unit_edb_valid=1，かつ，該当ビットが 1 の場合に，EDB0[0] の内容を書き込み先レジスタグループ識別子（0-1），EDB1[63:0] の内容をレジスタ値として書き込みを行う．全ての unit に対して書き込みが完了すると，pe0_status=STATUS_LMMI に遷移する．タイミングチャートを図 4.5 に示す．

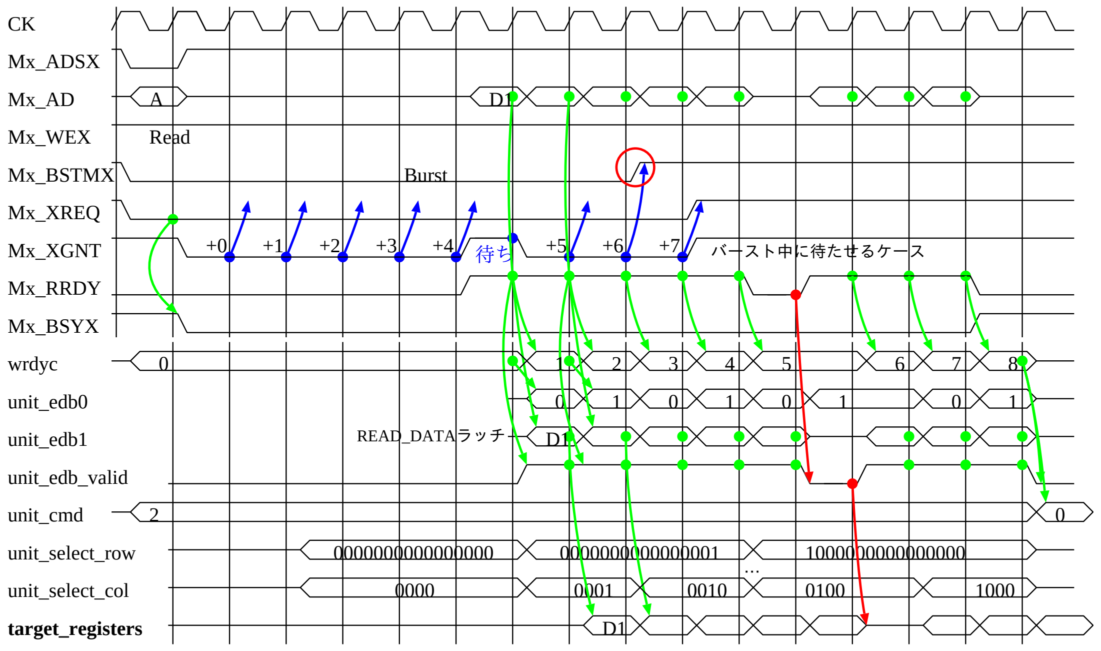


Figure.4.5: regv[][] → unit[][] のタイミングチャート

4.3.4 LMM tag initialization in progress (pe0_status=STATUS_LMMI)

conf[][] を元に LMM タグ情報（unit 毎ではなく全体に 1 つ存在）を初期化する．M_WEX=1，M_BSTMX=0（バーストリード）を使用して DDR3 から lmmi[][] を読み出す．EDB0[0] の内容を書き込み先識別子（0 の場合 ctl_new，1 の場合 ddr3_tlb），EDB1[63:0] の内容を設定値として書き込みを行う．全ての lmmi[][] の書き込みが完了すると，pe0_status=STATUS_LMM_LOAD に遷移する．タイミングチャートを図 4.6 に示す．

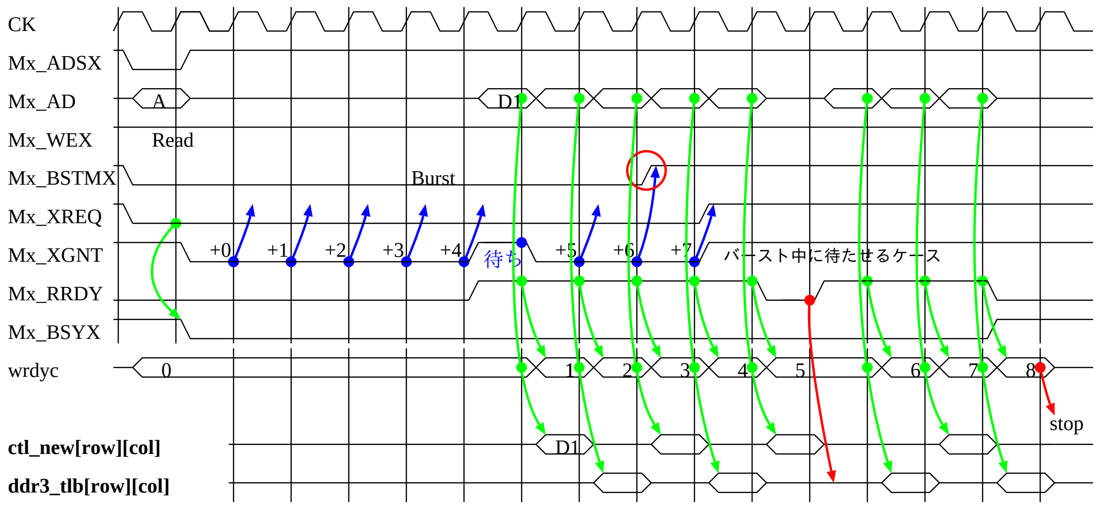


Figure.4.6: lmmi[][] → ctl_new[][]/ddr3_tlb[][] のタイミングチャート

4.3.5 LMM loading in progress (pe0_status=STATUS_LMM_LOAD)

前述の ctl_new 情報に基づき，1 つの LMM 毎に，M_WEX=1，M_BSTMX=0（バーストリード）を使用して DDR3 から各 unit の LMM に初期値をロードする．なお，前回実行時に使用した ctl_new[][] が ctl_old[][] に保存されており，該当 unit に対応する，ctl_old[i][j].top = ctl_new[i][j].top，ctl_old[i][j].len ≥ ctl_new[i][j].len，かつ，ctl_old[i][j].dist = ctl_new[i][j].dist の場合は LMM の内容を再利用できるためロードが省略される．各 unit は，unit_edb_cmd，unit_edb_valid，行指定ビットマップ（unit_select_row）の該当行ビット，および，列指定ビットマップ（unit_select_col）の該当列ビットを監視しており，unit_edb_cmd=4，かつ，unit_edb_valid=1，かつ，該当ビットが 1 の場合に，EDB0[12:3] の内容を書き込み先アドレス（最大 1K ダブルワード），EDB1[63:0] の内容を LMM 値として書き込みを行う．1 つの LMM の更新が完了する度に，該当する ctl_new[][] の内容が ctl_old[][] に保存される．必要な LMM の書き込みが完了すると，pe0_status=STATUS_START に遷移する．タイミングチャートを図 4.7 に示す．

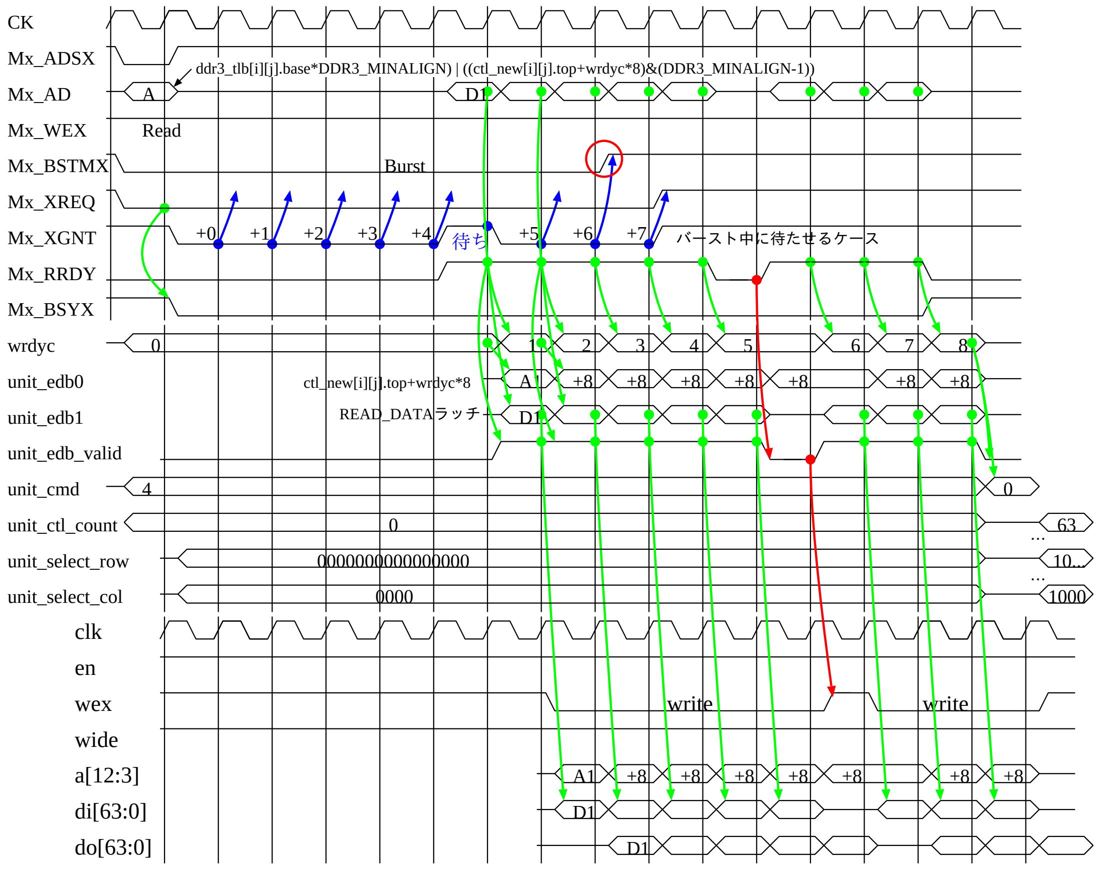


Figure.4.7: source_DDR3 → lmm[][] のタイミングチャート

4.3.6 Start execution (pe0_status=STATUS_START)

EMAX2 の演算を開始する．各 unit は，unit_edb_cmd，unit_edb_valid，行指定ビットマップ（unit_select_row）の該当行ビット，および，列指定ビットマップ（unit_select_col）の該当列ビットを監視しており，unit_edb_cmd=5，かつ，該当ビットが 1 の場合に，連続演算の起点として，当該 unit が起動する．なお，本状態に遷移した次のサイクルにおいて，pe0_status=STATUS_EXEC に遷移する．

4.3.7 Execution in progress (pe0_status=STATUS_EXEC)

unit_edb_cmd=5 を維持したまま，EMAX2 の連続演算を継続する．全 unit の演算が完了した場合，pe0_status=STATUS_LMM_DRAIN に遷移する．

4.3.8 LMM drainage in progress (pe0_status=STATUS_LMM_DRAIN)

前述の ctl_new 情報に基づき，1 つの LMM 毎に，M_WEX=0，M_BSTMX=0（バーストライト）を使用して各 unit の LMM から DDR3 に演算結果をストアする．各 unit は，unit_edb_cmd，unit_edb_valid，行指定ビットマップ（unit_select_row）の該当行ビット，および，列指定ビットマップ（unit_select_col）の該当列ビットを監視しており，unit_edb_cmd=6，かつ，unit_edb_valid=1，かつ，該当ビットが 1 の場合に，EDB0[12:3] の内容を読み出し元アドレス（最大 1K ダブルワード），EMB0[63:0] を LMM 値として読み出しを行う．必要な LMM の読み出しが完了すると，pe0_status=STATUS_TERM に遷移する．タイミングチャートを図 4.8 に示す．

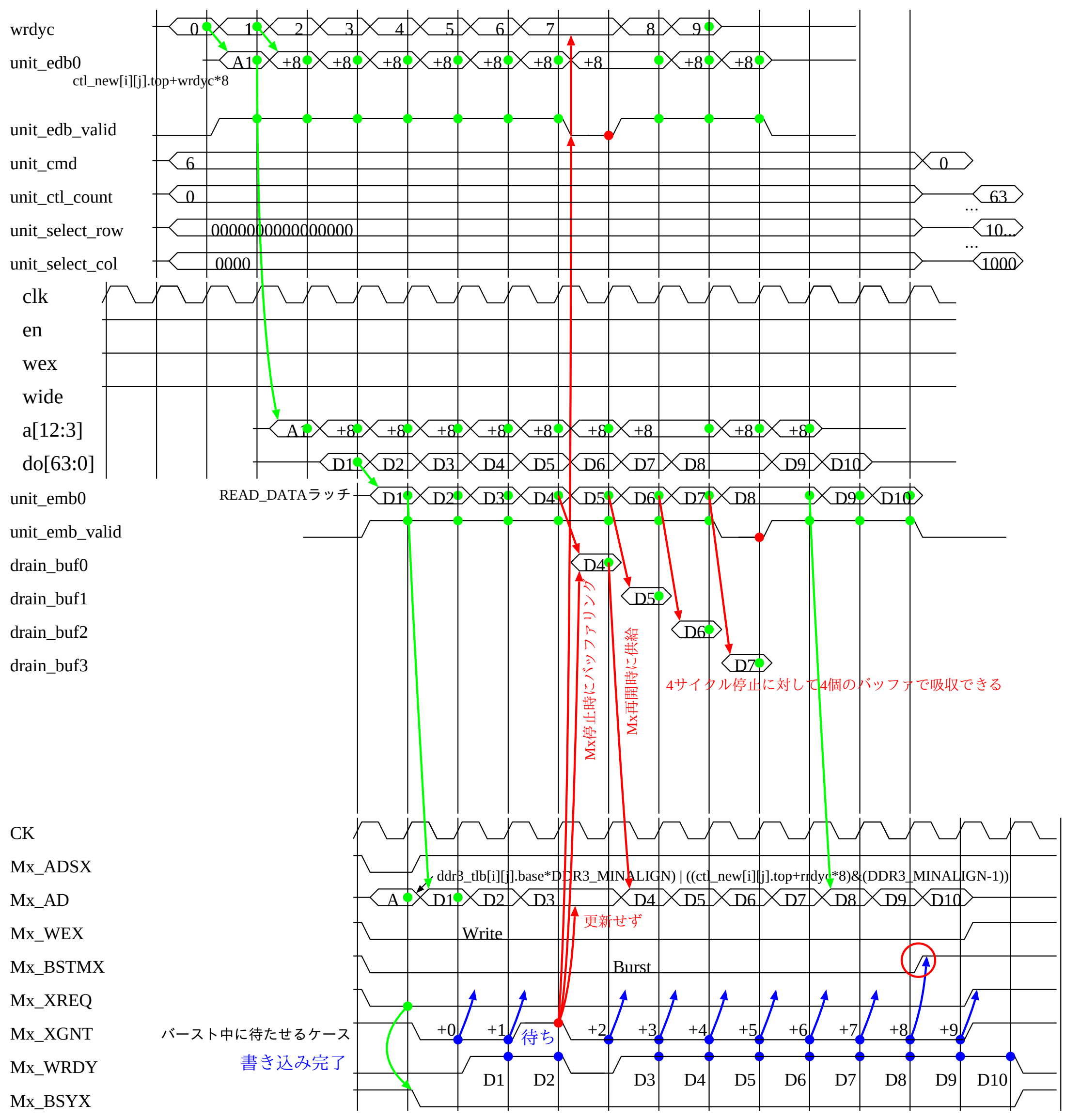


Figure.4.8: lmm[][] → result_DDR3 のタイミングチャート

4.3.9 Terminate execution (pe0_status=STATUS_TERM)

M_STAT に 1 を書き込み，unit_offset を更新し，pe0_status=STATUS_IDLE に遷移する．

Chapter 5

How to use EMAX2 simulator

5.1 Compiling application programs

See “all:” tag in proj-emax/sample/filter/Makefile-emax2.

5.2 Executing application programs

See ”run:” tag in proj-emax/sample/filter/Makefile-emax2. For monitoring the internal status of EMAX2, insert following code in the application program. These functions are defined in emax2.c.

```
emax2_show_status_start(); /* detailed information */
emax2_show_demo_start();   /* full-screen summary of execution */
```

Appendix A

評価システム

A.1 GP6X760MP システムの構成

GP6X760MP システムは，XC6V760 × 1，ASIC ソケットボード× 4，256MB-DDR3 × 2，および，USB3.0 インタフェース× 2 を備え，2 本の USB ケーブルにより PC/AT 互換機（HOST）に接続される．プロセッサ機能（以下 PE）を FPGA または ASIC ソケットボード上に，メモリ機能を DDR3 上に，また，主記憶機能および入出力機能を HOST 上に各々対応付けることにより，全体として実用アプリケーションを走行できる装置である．図 A.1 に全体構成，表 A.1 に，USB 毎に HOST に写像されるアドレス空間を示す．

各 USB インタフェースには，256MB の DDR3 および 2 組の PE が対応付けられる．「PE0/1 からの参照」列は，PE に許可されるアクセス種別，「HOST からの参照」列は，HOST に許可されるアクセス種別を示す．なお，HOST が pe_reset に非ゼロの 16 ビット値を書き込んだ場合，PE に至る PE0_RESET 信号が HIGH となる．pe_reset が毎サイクルデクリメントされて 0 に達した後に PE0_RESET 信号が LOW に復帰する．

HOST が h_p に対して非ゼロの制御コマンドを書き込んだ場合，f_stat に 1（STATUS_BUSY）がセットされ，その後，各機能毎に規定された値がセットされる．h_s に対して非ゼロの制御コマンドを書き込んだ場合，s_stat に 1（STATUS_BUSY）がセットされ，その後，各機能毎に規定された値がセットされる．PE は，HOST からの h_p または h_s への書き込みに伴って LOW になる C_XINT 信号により，h_p または h_s の更新を検知し，Cx_*を用いて制御レジスタを直接参照することにより，h_p，h_s，h_param，h_control を受け取る．PE が Cx_XINT を検出し Cx_XACK を LOW にした時点で，Cx_XINT は HIGH にならなければならない．Cx_XREQ は PE が制御レジスタに対して書き込みまたは読み出しを行う際に LOW にする信号線である．ただし，EMAX2 の初期モデルでは，本 HOST インタフェースは使用されない．

※ M_AD[27:3] is valid when M_ADSX=0 (M_AD holds the address for 64bit data bus of DDR3).
※ M_AD_OEX is connected to (M_ADSX & M_WEX). ※ C_D_OEX is connected to C_WEX.

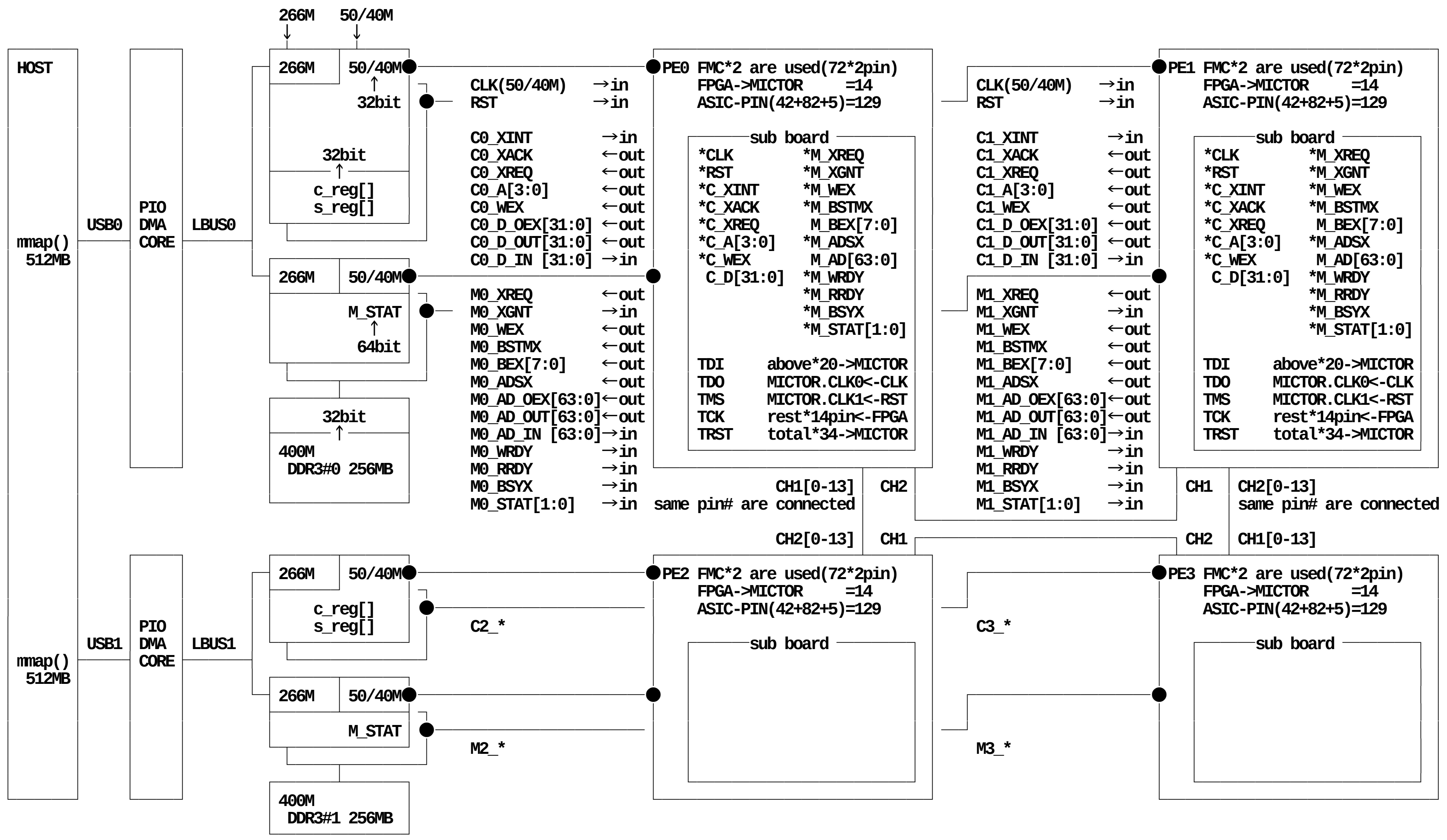


Figure.A.1: 全体構成

Table.A.1: 制御レジスタ空間と DDR3 空間

HOST address	Notes	from PE0/1	from HOST-PC
0x0000003f - 000	DMA control registers	-- out of space	RW
0x00000043 - 040	reset & status register GRST bit0	-- out of space	RW W:1 start reset R:1 reset in progress
0x0000013f - 100	HOST->PE0 control registers	R C_A[3:0]=0-7	RW
0x00000107 - 100 0x0000010f - 108 0x00000117 - 110 0x0000011f - 118 0x00000127 - 120 0x0000013f - 128	h_p bit15- 4 h_s bit31-16 h_param lower 4bytes h_control lower 4bytes -N.A.- pe_reset bit15- 0 -N.A.-	R R R R -N.A.- -- -N.A.-	RW:non-0 write ->f_stat=1 RW:non-0 write ->s_stat=1 RW RW RW -N.A.- RW:non-0 write -> PEx-reset -N.A.-
0x0000017f - 140	PE0->HOST status registers	RW C_A[3:0]=8-15	R
0x00000147 - 140 0x0000014f - 148 0x0000017f - 150	f_stat bit15-10 s_stat bit23-16 p_param lower 4bytes -N.A.-	RW RW RW -N.A.-	R:h_p non-0 write -> 1 R:h_s non-0 write -> 1 R -N.A.-
0x000001bf - 180	HOST->PE1 control registers	R C_A[3:0]=0-7	RW
0x00000187 - 180 0x0000018f - 188 0x00000197 - 190 0x0000019f - 198 0x000001a7 - 1a0 0x000001bf - 1a8	h_p bit15- 4 h_s bit31-16 h_param lower 4bytes h_control lower 4bytes -N.A.- pe_reset bit15- 0 -N.A.-	R R R R -N.A.- -- -N.A.-	RW:non-0 write ->f_stat=1 RW:non-0 write ->s_stat=1 RW RW RW -N.A.- RW:non-0 write -> PEx-reset -N.A.-
0x000001ff - 1c0	PE1->HOST status registers	RW C_A[3:0]=8-15	R
0x000001c7 - 1c0 0x000001cf - 1c8 0x000001ff - 1d0	f_stat bit15-10 s_stat bit23-16 p_param lower 4bytes -N.A.-	RW RW RW -N.A.-	R:h_p non-0 write -> 1 R:h_s non-0 write -> 1 R -N.A.-
0x1fffffff - 0x10000000	local DDR3 space(256MB)	RW M_A burst mode	RW PIO/DMA
0x10000077 - 000 0x1000007f - 078	通常データ PE0.L2CT_0000 bit5-4	PE0用RW W:0書き込み時M_STAT=0 W:1書き込み時M_STAT=1 バースト時も上記機能は有効	RW W:0書き込み時M_STAT=0 W:2書き込み時M_STAT=2 W:3書き込み時M_STAT=3 DMA時も上記機能は有効
0x100000f7 - 080 0x100000ff - 0f8	通常データ PE1.L2CT_0000 bit5-4	PE1用RW W:0書き込み時M_STAT=0 W:1書き込み時M_STAT=1 バースト時も上記機能は有効	RW W:0書き込み時M_STAT=0 W:2書き込み時M_STAT=2 W:3書き込み時M_STAT=3 DMA時も上記機能は有効
0x2fffffff - 0x20000000	other DDR3 space(256MB)	RW M_A burst mode	RW 他系USBからのみPIO/DMA可

A.2　プロセッサ機能と HOST 機能のインタフェース

表 A.2 に PE と HOST 機能のインタフェース，図 A.2 にタイミングチャートを示す．ただし，EMAX2 の初期モデルでは，本 HOST インタフェースは使用されない．

Table.A.2: プロセッサ-HOST 機能のインタフェース信号（ASIC インタフェースとしては 40 本）

信号名	説明
Cx_XINT(in)	HOST からのリクエスト信号，Active-LOW
Cx_XACK(out)	プロセッサからの INT 受付け信号，Active-LOW
Cx_XREQ(out)	プロセッサからのレジスタ参照信号，Active-LOW
Cx_A[3:0](out)	アドレス，A3-0
Cx_WEX(out)	書き込みイネーブル，Active-LOW
Cx_D_OEX[31:0](out)	Cx_D_OUT の各 bit に対応する出力イネーブル，Active-LOW．全 bit に Cx_WEX を 2 サイクル遅延させた信号が接続される．
Cx_D_OUT[31:0](out)	出力データ，D31-0
Cx_D_IN([31:0]in)	入力データ，D31-0

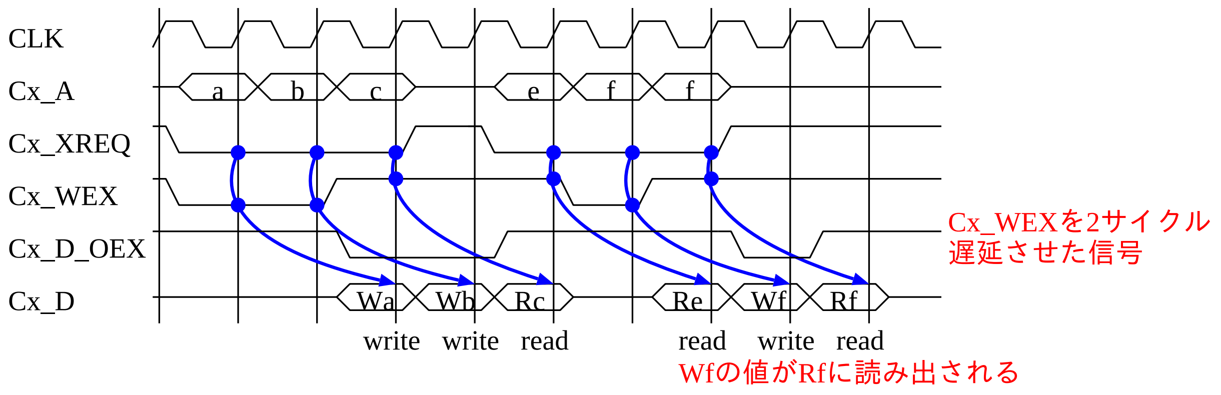


Figure.A.2: プロセッサ制御インタフェースのタイミングチャート (WRITE/READ)

A.3 プロセッサ機能と DDR3 のインタフェース

表 A.3 に PE と DDR3 のインタフェース，図 A.3 から図 A.6 にタイミングチャートを示す.

Table.A.3: プロセッサ-DDR3 のインタフェース信号（ASIC インタフェースとしては 82 本）

信号名	方向	説明
Mx_XREQ	out	DDR3 参照要求信号，Active-LOW
Mx_XGNT	in	要求受付完了信号，Active-LOW．バースト時，次のデータを送信できることを示す
Mx_WEX	out	書き込みイネーブル，Active-LOW
Mx_BSTMX	out	バーストモード，Active-LOW．HIGH 復帰後の最初の CLK ↑ がバースト転送の最終回
Mx_BEX[7:0]	out	Mx_AD_OUT がデータである場合の byte 単位書き込みイネーブル，Active-LOW
Mx_ADSX	out	0 の場合 Mx_AD_OUT はアドレス，1 の場合データ
Mx_AD_OEX[63:0]	out	Mx_AD_OUT の各 bit に対応する出力イネーブル，Active-LOW．全 bit に (Mx_ADSX & Mx_WEX) が接続される
Mx_AD_OUT[63:0]	out	Mx_ADSX が 0 の場合アドレス (bit27-3 のみ有効)，1 の場合データ
Mx_AD_IN[63:0]	in	入力データ
Mx_WRDY	in	書き込み完了信号，Active-HIGH．本信号のカウントによりプロセッサは書き込み完了を知る
Mx_RRDY	in	読み出し完了信号，Active-HIGH．本信号によりプロセッサは Mx_AD_IN の有効値を取り込む
Mx_BSYX	in	動作中を示すビジー信号，Active-LOW．HIGH 時は次の参照要求を発行できることを示す
Mx_STAT[1:0]	in	状態表示信号，STAT1-0（0:empty 2:OP-ok 3:IF-ok）は，特定アドレスへの書き込み値を反映する

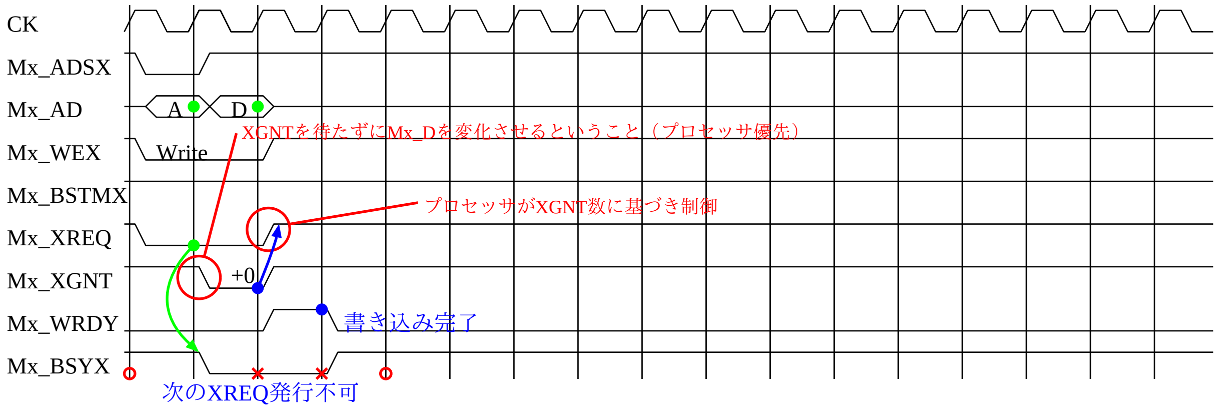


Figure.A.3: DDR3 インタフェースのタイミングチャート (WRITE)

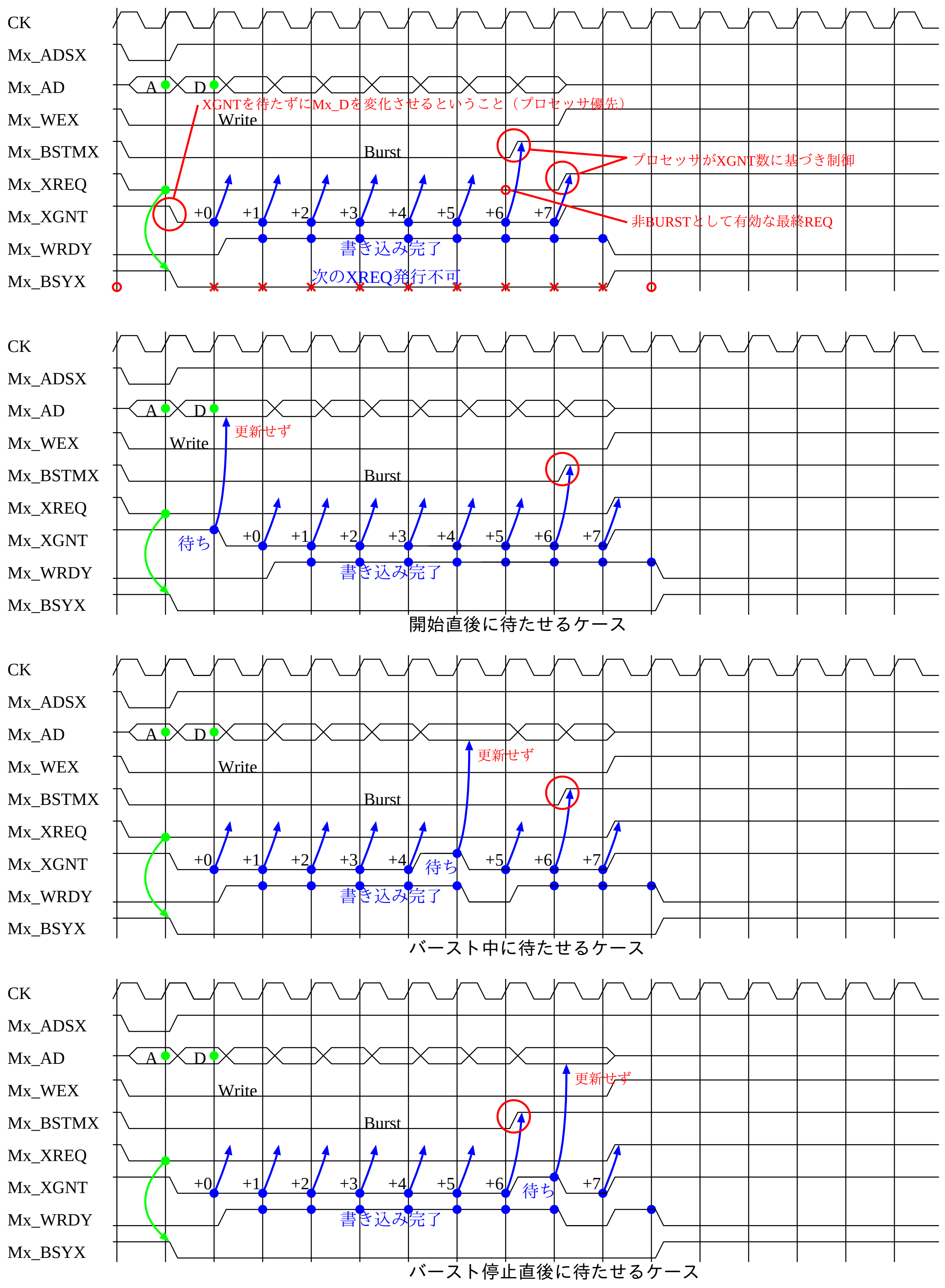

CK
Mx_ADSX
Mx_AD
Mx_WEX
Mx_BSTMX
Mx_XREQ
Mx_XGNT
Mx_WRDY
Mx_BSYX
A
D
Write
Burst
+0
+1
+2
+3
+4
+5
+6
+7
XGNTを待たずにMx_Dを変化させるということ（プロセッサ優先）
プロセッサがXGNT数に基づき制御
非BURSTとして有効な最終REQ
書き込み完了
次のXREQ発行不可
更新せず
待ち
開始直後に待たせるケース
バースト中に待たせるケース
バースト停止直後に待たせるケース

Figure.A.4: DDR3 インタフェースのタイミングチャート (WRITE バーストモード)

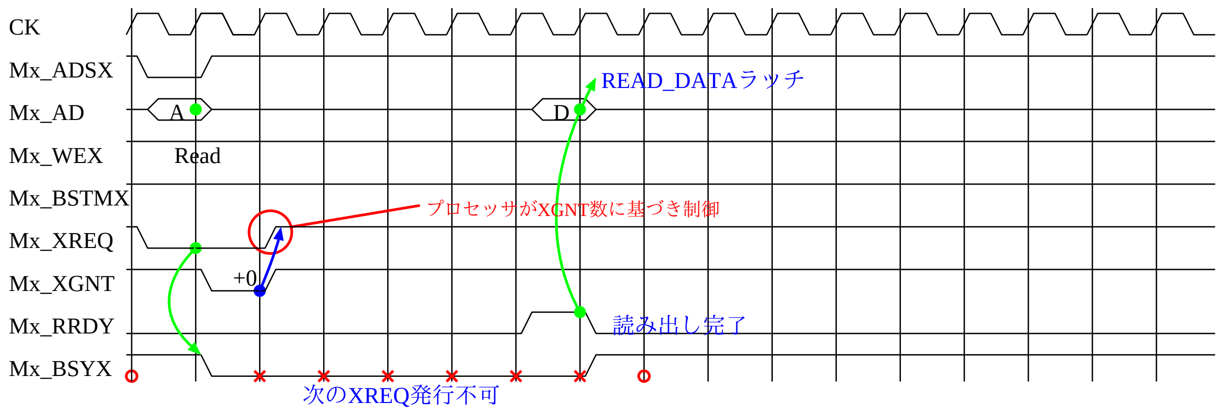
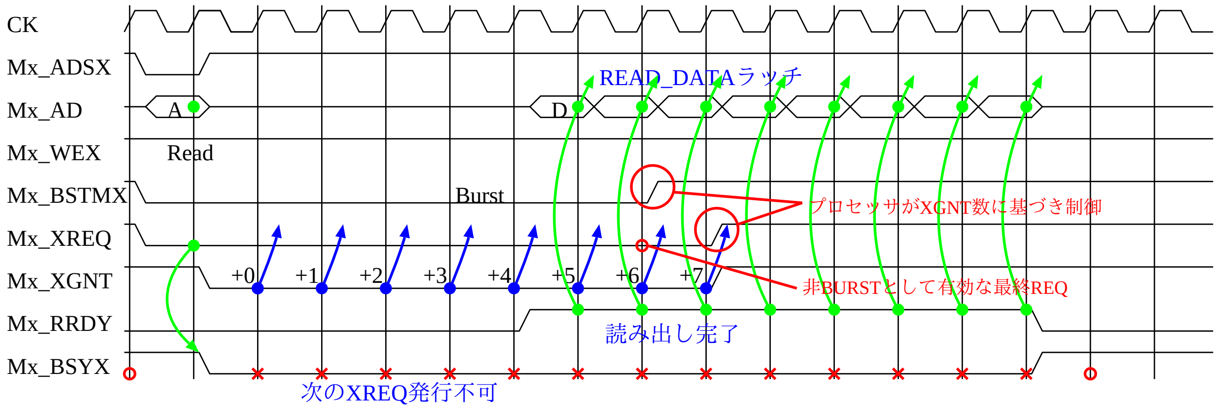
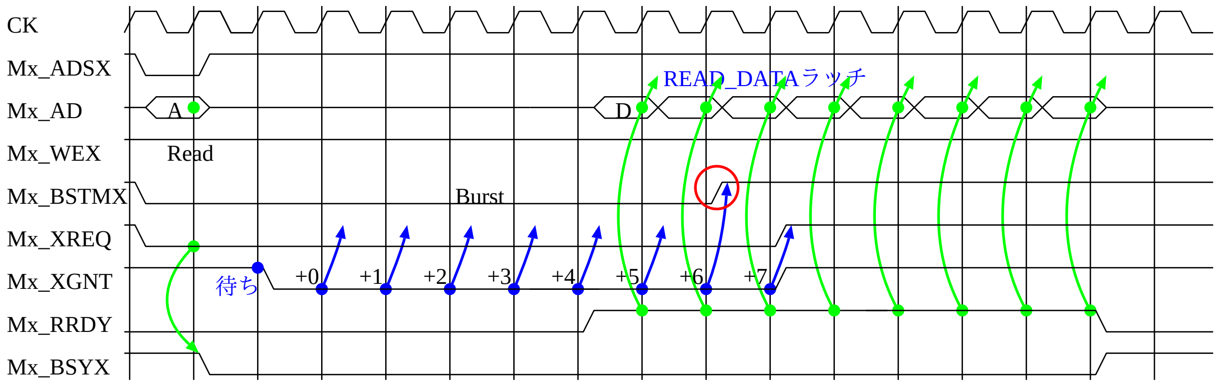
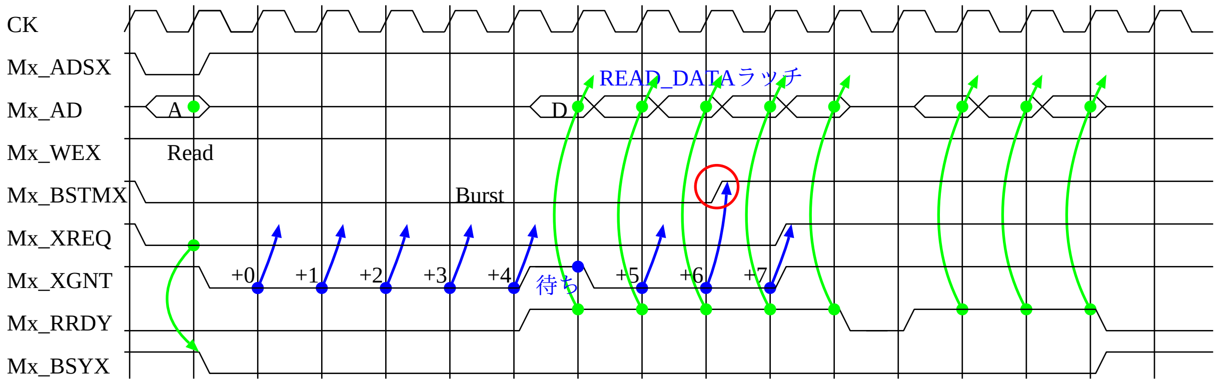


Figure.A.5: DDR3 インタフェースのタイミングチャート (READ)

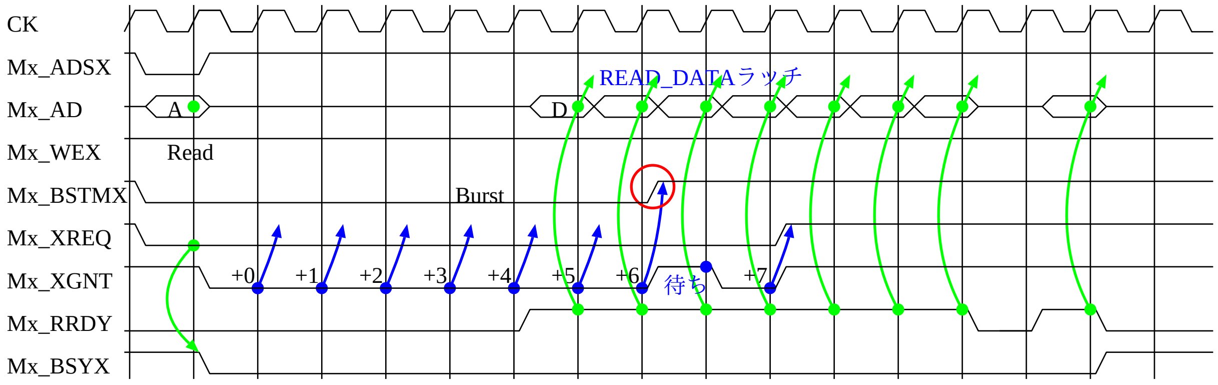




開始直後に待たせるケース



バースト中に待たせるケース



バースト停止直後に待たせるケース

Figure.A.6: DDR3 インタフェースのタイミングチャート (READ バーストモード)

Appendix B

ASIC 拡張機能

B.1 QFP208 パッケージ用インタフェース

XC6V760 と各 GPSUB-ASICEDB(ドータボード) の物理インタフェースは，ローカルメモリインタフェース信号（82 本），および，プロセッサ制御インタフェース信号（40 本）の合計 122 本を収容する．さらに，スキャン用信号 5 本，クロック，リセットの 2 本を加えた合計 129 本が ASIC に接続される I/O である．

また，図 A.1 に示すように，129 本のうち 20 本は MICTOR コネクタに接続され，MICTOR コネクタの残り 14 本には，FPGA のピンが 1 対 1 に接続される．すなわち，129+14=143 本が，XC6V760 と各 GPSUB-ASICEDB を接続する信号線である．

さらに，図 A.1 に示すように，ドータボード間接続用に，CH0 および CH1（各 14 本）が設けられている．CH0 と CH1 の同一番号ピンがケーブルにより直接接続され，CH0-CH0 間，または，CH1-CH1 間が接続されることはない．

ASIC（ROHM/TSMC : QFP208）のピン配置制約は次の通り．

- VDD(1.8V) : 計 12 ピン（11，32，52，63，84，104，115，136，156，167，188，208）
- VDDO(3.3V) : 計 12 ピン（1，21，42，53，73，94，105，125，146，157，177，198）
- VSS : 計 24 ピン（2，12，22，31，41，51，54，64，74，83，93，103，106，116，126，135，145，155，158，168，178，187，197，207）
- JTAG 用 TDI，TDO，TMS，TCK，TRST : 計 5 ピン（未使用時はプルアップ）
- クロック，リセット入力ピン : 計 2 ピン
- 一般信号用 : 残り 153 ピン．同時スイッチングする出力ピンおよび双方向ピンは，電源ピンに近接配置．電源ピンごとに分散配置．電源グループに対する同時スイッチング出力バッファの許容本数を守る．同時スイッチングする出力バッファや，その近傍の出力バッファを他 LSI のクロック入力としない．また，ドライバビリティの大きな出力ピンは中央に配置．

コア電圧は 1.8V．I/O は 3.3V．電源ピンを除く合計 160 ピンのうち 129 本が FPGA に接続される．ASIC の I/O には，駆動力 2mA のドライバを推奨する．なお，ROHM0.18 μ の I/O バッファには以下を推奨する．TSMC0.18 μ についても同様の駆動能力を推奨する．

- PC30B01 入出力用．出力用 I ピンおよび OEN の容量は各 22fF（INV $\times$ 3 に相当）．OEN $\times$ 8 ピンを駆動するには 8 倍 INV が必要．
- PC30D01 入力専用．PAD から CIN の遅延は負荷 FO4/FO32 で約 200ps/300ps．INV $\times$ 32 まで直接駆動可能．
- PC30O01 出力専用．I から PAD の遅延は負荷 20pF/60pF で 5ns/10ns．

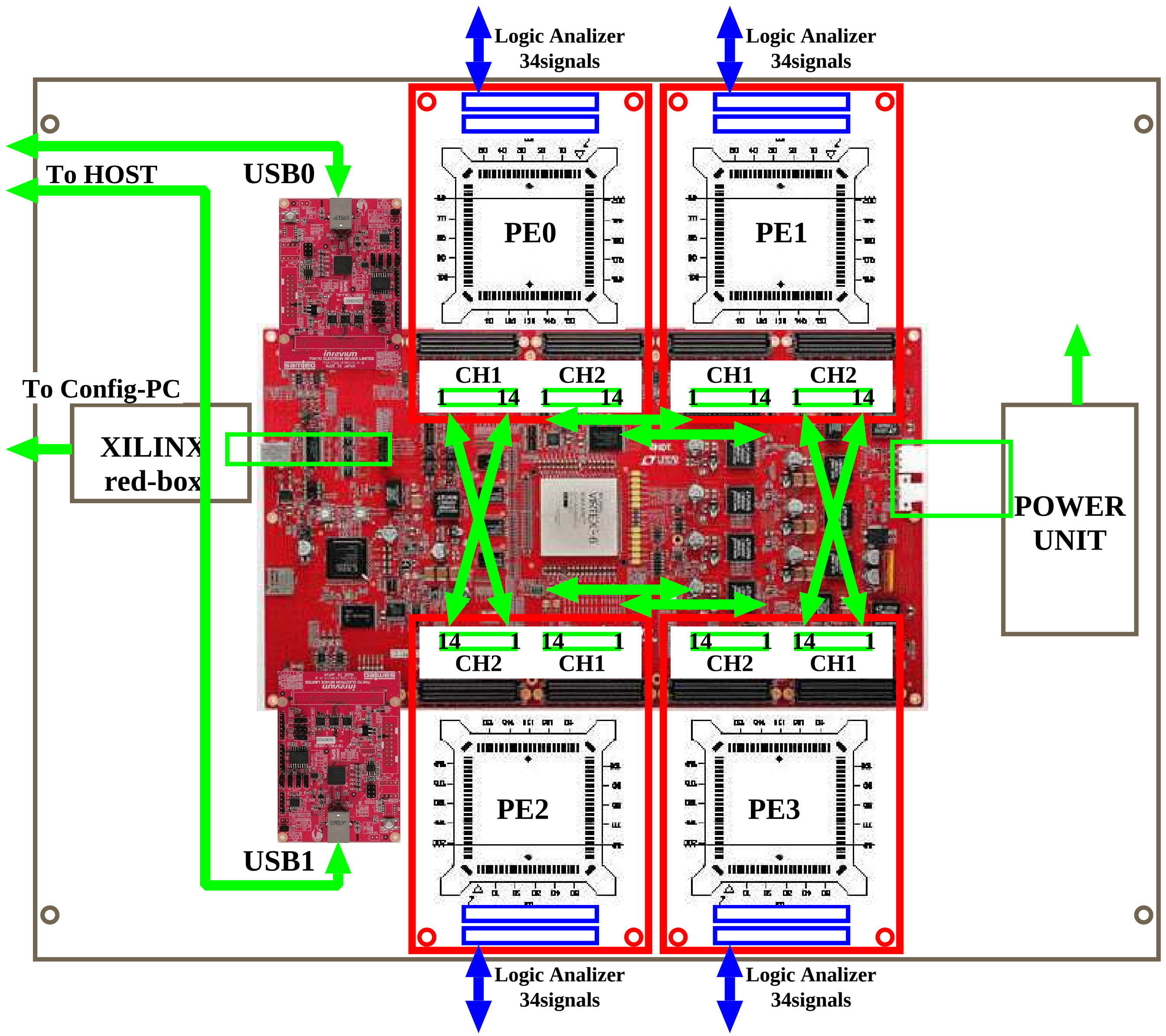


Figure.B.1: GP6X760 ボードの構成

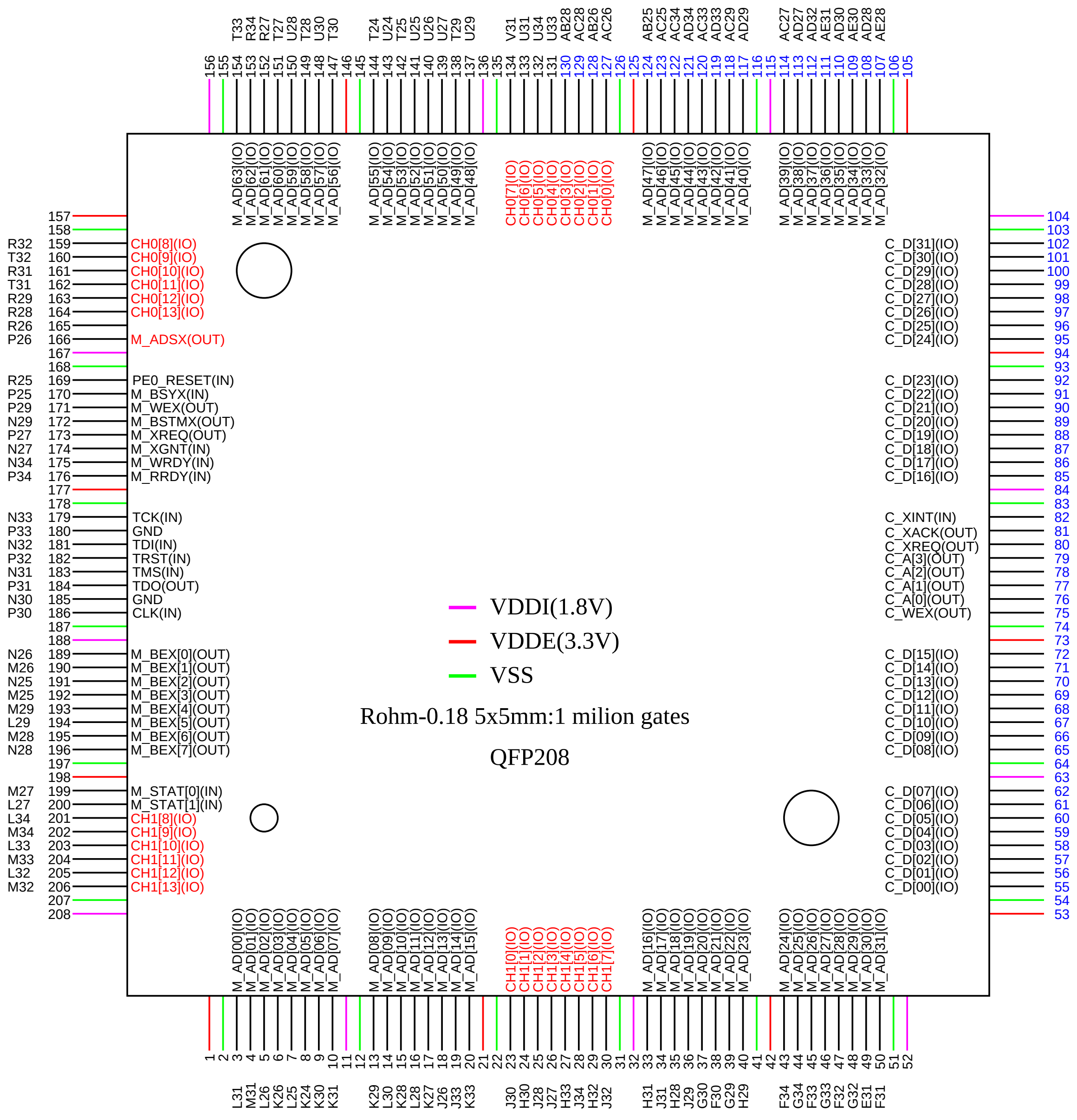


Figure.B.2: QFP208 のピン配置（周辺の記号は XILINX の対応ピン番号（現状は仮））

Appendix C

References

本章では，関連仕様書・規格，参考文献，関連ソースプログラム，および，EMAX2 ツールチェインを列挙する．

C.1 GP6X760 documents

- GP6X760 PIO-DMA コア概要仕様書.pdf
  . . . . proj-gp6x760/doc/GP6X760 用 PIO-DMA コア概要仕様書.pdf
- ChipScope 概要
  . . . . proj-gp6x760/doc/ChipScope.pdf
- TB-6V-LX760-LSI ハードユーザマニュアル
  . . . . proj-gp6x760/doc/TB-6V-LX760-LSI_HWUserManual_3.01.pdf
- TB-6V-LX760-LSI 回路図
  . . . . proj-gp6x760/doc/TB-6V-LX760-LSI_Schematic_2.00.pdf
- TB-6V-LX760-LSI_USB-DDR3 スタートアップガイド
  . . . . proj-gp6x760/doc/TB-6V-LX760-LSI_USB-DDR3_StartUpGuide_1.01.pdf
- TB-FMCH-CONNECTER 回路図
  . . . . proj-gp6x760/doc/TB-FMCH-CONNECTER_Schematic_2.00.pdf
- TB-FMCH-STACK 回路図
  . . . . proj-gp6x760/doc/TB-FMCH-STACK_Schematic_2.00.pdf
- TB-FMCL-USB30 動作確認済み PC 一覧
  . . . . proj-gp6x760/doc/TB-FMCL-USB30_Checked_PC_information_1.00.pdf
- TB-FMCL-USB30 ハードユーザマニュアル
  . . . . proj-gp6x760/doc/TB-FMCL-USB30_HWUserManual_1.00.pdf
- TB-FMCL-USB30 回路図
  . . . . proj-gp6x760/doc/TB-FMCL-USB30_Schematic_1.09.pdf
- TB-FMCL-USB30 スタートアップガイド
  . . . . proj-gp6x760/doc/TB-FMCL-USB30_StartupGuide_1.00.pdf
- uSD_CONF ユーザーズガイド
  . . . . proj-gp6x760/doc/uSD_CONF_UserManuarl_V6LSI_2.00.pdf

C.2 EMAX2 documents/sources

- EMAX2 アイデア履歴 . . . . proj-emax/doc/IDEA*
- EMAX2 基本特許 proj-emax/doc/pat33.tgz
- EMAX2 科研基盤 A proj-emax/doc/kaken2012.tgz

- Autovectorization in GCC .... proj-emax/doc/vectorization.pdf
- gcc proj-emax/src/gcc-4.4.6.tgz
- gmp proj-emax/src/gmp-4.3.2.tgz
- mpfr proj-emax/src/mpfr-3.0.1.tgz
- EMAX2 ディレクティブ解析 proj-emax/src/conv-a2i/*
- binutils proj-emax/src/binutils-2.21.tgz
- EMAX2 RTL シミュレータ（xsim 部分） .... proj-emax/src/xsim/emax2.*,soft_monitor.c
- EMAX2 性能・電力評価 RTL シミュレータ（filter 部分） .... proj-emax/sample/filter/*
- GP6X760-EMAX2 FreeBSD 用ドライバ proj-emax/src/xsim/GP6X760-Driver.c
- GP6X760-EMAX2 FreeBSD 用ユーザ関数およびヘッダ proj-emax/src/xsim/gp6x760.c
- GP6X760-EMAX2 簡易試験ツール proj-emax/src/xsim/gp6x760-test.c

C.3 EMAX2 toolchains

- EMAX2 FORTRAN コンパイラ .... proj-emax/bin/gfortran
- EMAX2 C プリプロセッサ .... proj-emax/bin/cpp
- EMAX2 C コンパイラ .... proj-emax/bin/gcc
- EMAX2 ライブラリ .... proj-emax/lib/*
- EMAX2 ディレクティブ解析 proj-emax/src/conv-a2i/conv-a2i
- EMAX2 アセンブラ .... proj-emax/bin/as
- EMAX2 リンカ .... proj-emax/bin/ld
- EMAX2 逆アセンブラ .... proj-emax/bin/objdump
- EMAX2 性能・電力評価 RTL シミュレータ（filter） proj-emax/sample/filter/frontend-emax2
- EMAX2 Intel 比較用バイナリ（filter） .... proj-emax/sample/filter/frontend-sse
- GP6X760-EMAX2 Verilog 雛型 proj-emax/fpga/step4000-GP6X-4way
- GP6X760-EMAX2 コンフィグレーションツール proj-emax/src/xsim/Config-gp6x760