EMAX4SPC-0001
Ver.0.9: Oct. 20 2014
Ver.1.0: Apr. 20 2015
Ver.1.1: May. 6 2015

EMAX2asic/ZYNQ and EMAX4/bsim Architecture Handbook – Energy-aware Multimode Accelerator eXtension –

Nara Institute of Science and Technology
Computing Architecture Laboratory
Accelerator Group

Table of contents

List of Figures

List of Tables

Chapter 1

EMAX2asic/ZYNQ Software

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, local memory module (LMM) 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 LMM 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 LMMs.

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 LMM 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.

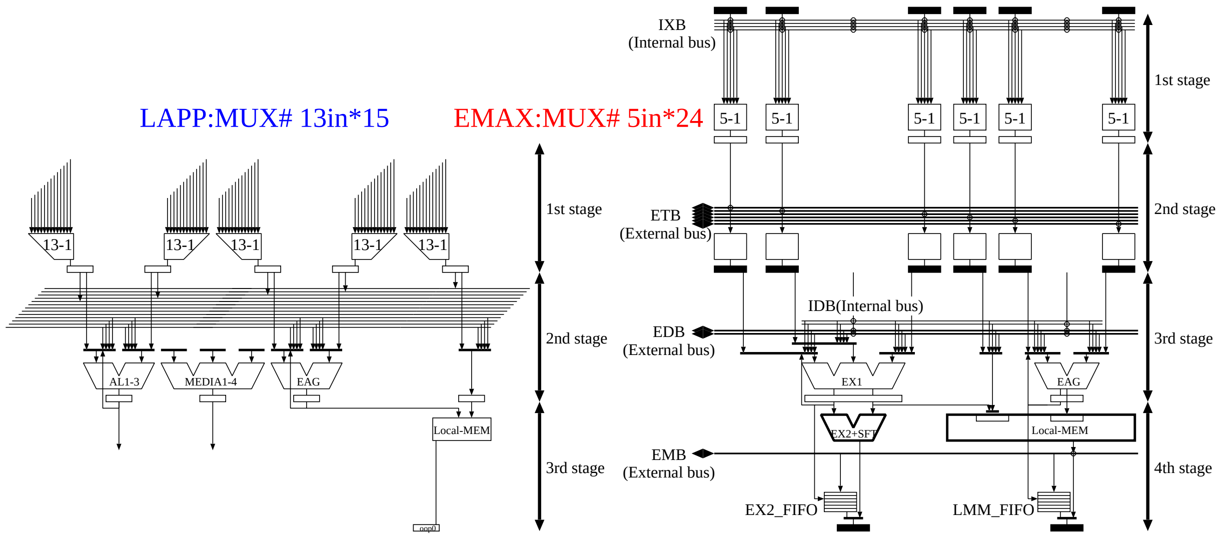


Figure.1.1: EMAX2 basic function.

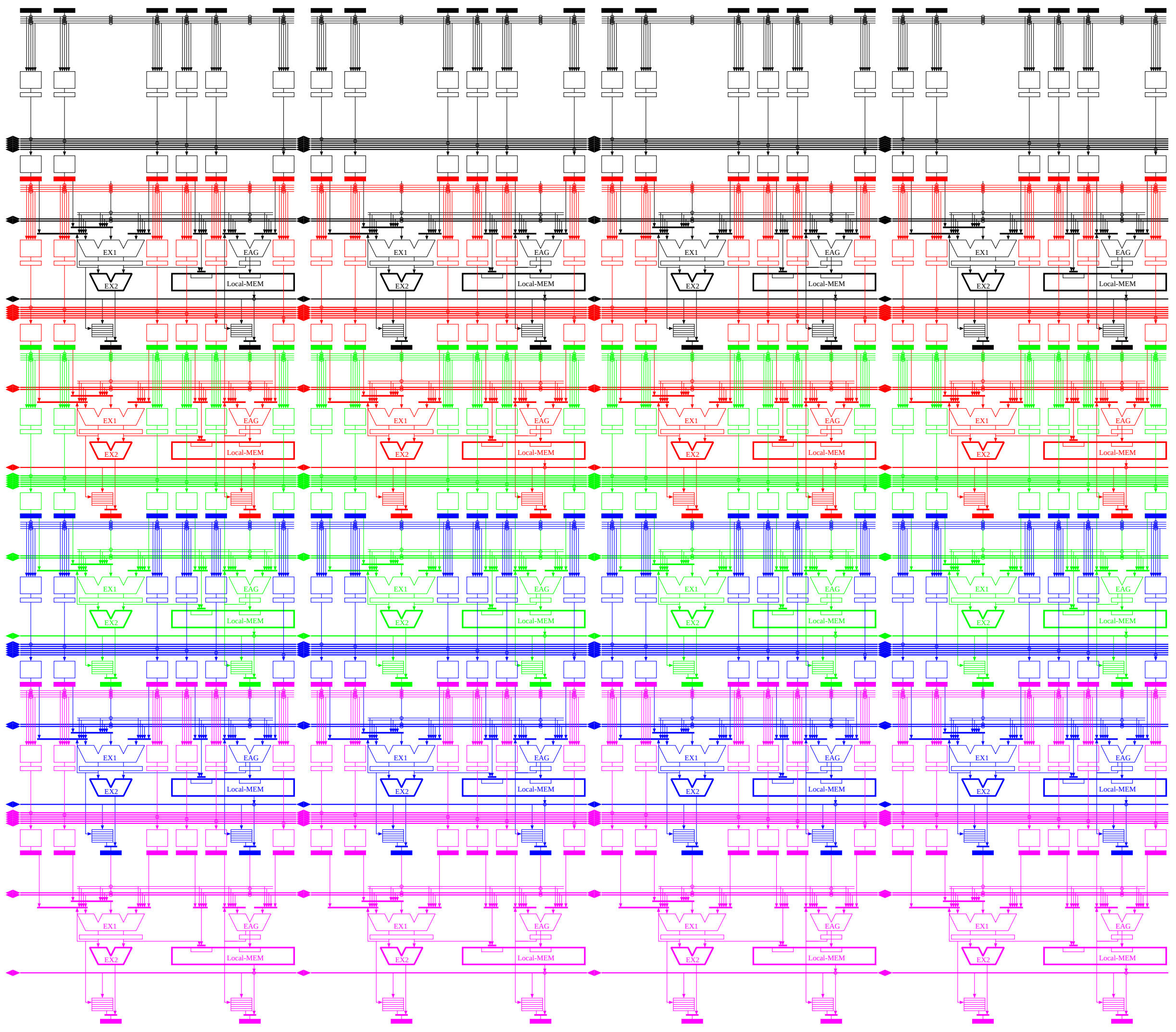


Figure.1.2: Interconnection of functions.

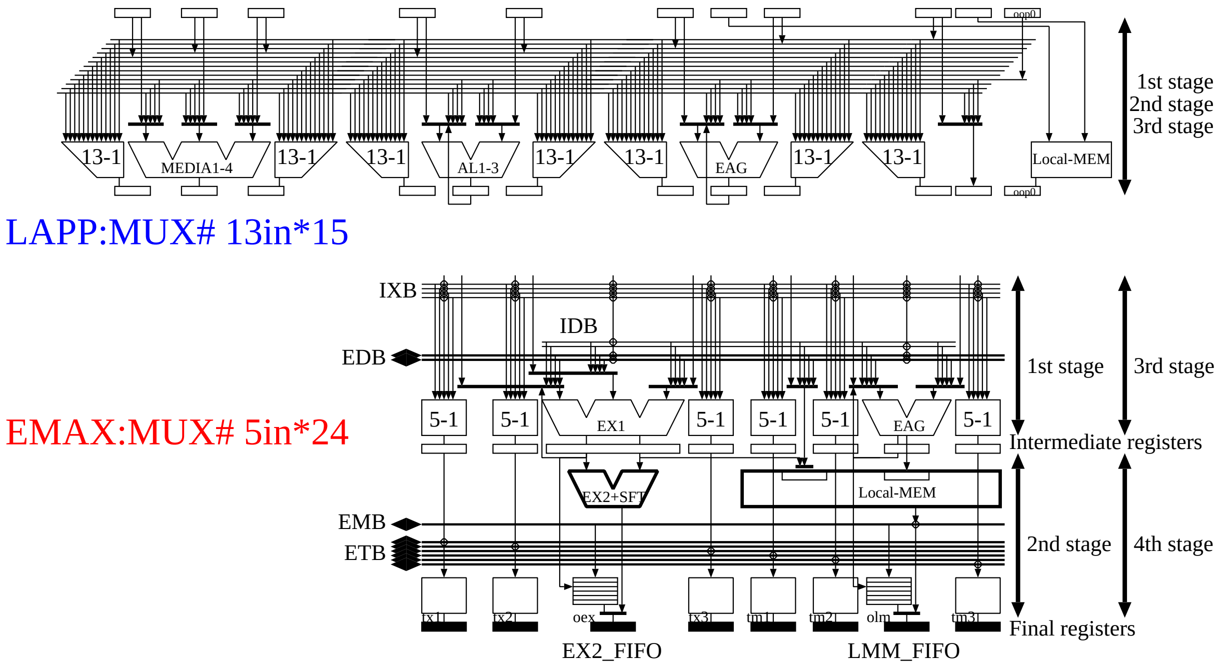


Figure.1.3: EMAX2 basic unit.

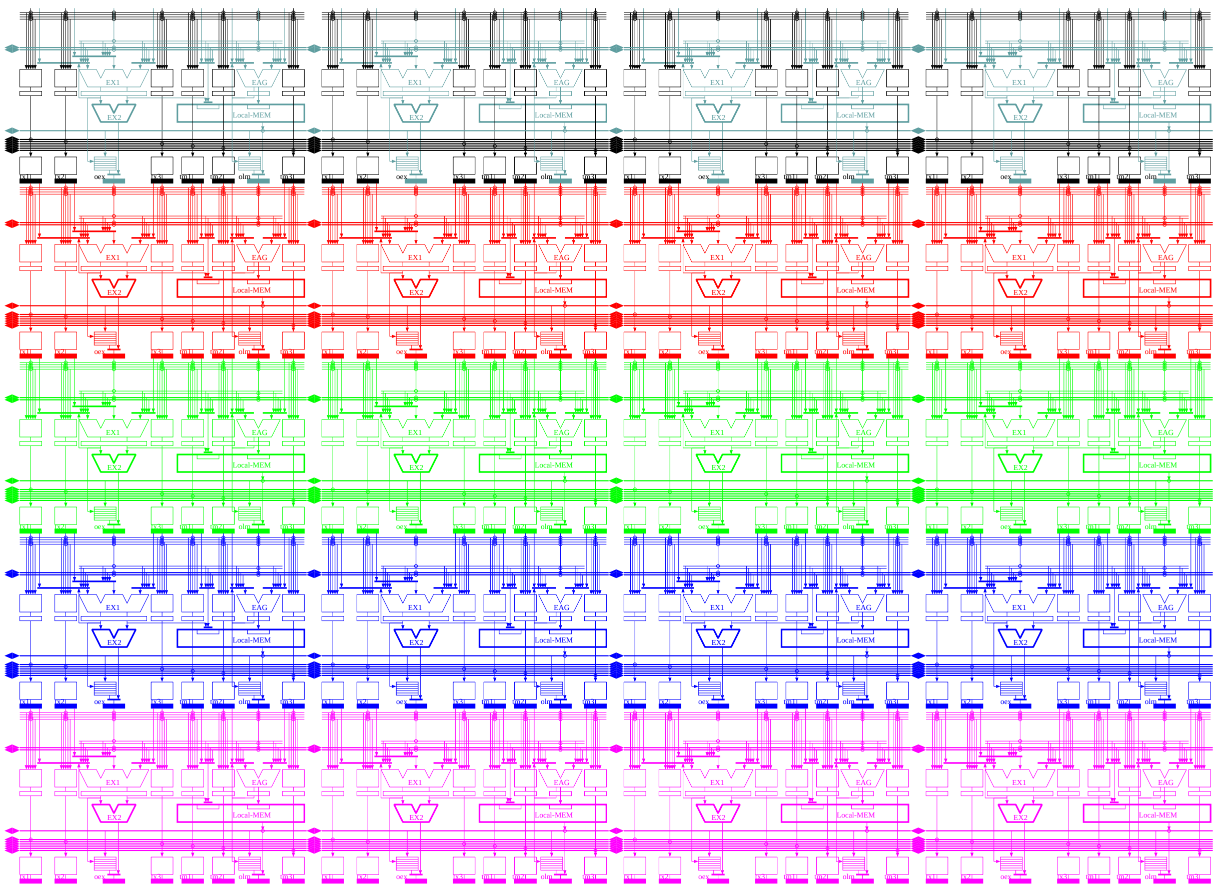


Figure.1.4: Interconnection of units.

1.3 Programming model

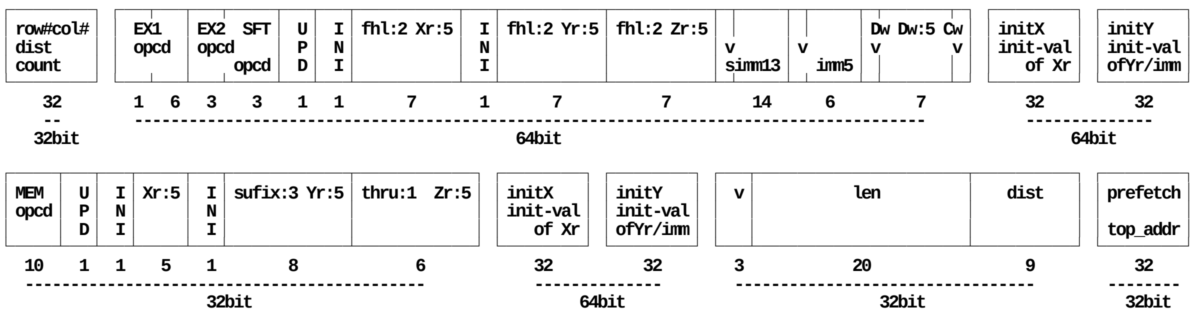
Application programs should be written according to the interface provided by ARM application binary interface (ABI). “bl emax2_start” is a trigger to invoke EMAX2asic from ARM. The first argument (r0) and the second argument (r1) correspond to starting and last address of EMAX2 codes respectively. Notice that the prefix of EMAX2asic/ZYNQ instruction is “//EMAX4A” that has different format from EMAX2/intel (specified in proj-emax/doc/emax2/emax2.pdf). The address space referred by EMAX2asic should be inside the physical memory space (0x3fffffff-0x20000000) where virtual address is not mapped by OS as shown in 2.2.

```
        .text
        .align 2
        .globl  template
        .type   template,%function
template:
        stmfd   sp!, {r4, r5, r6, r7, r8, r9, r10, fp, lr}
        sub     sp, sp, #20

        ldr     r0, .emax_loc_start_template
        ldr     r1, .emax_loc_end_template
        bl      emax2_start                         // start EMAX2asic
        mov     r0, #0
        add     sp, sp, #20
        ldmfd   sp!, {r4, r5, r6, r7, r8, r9, r10, fp, lr}
        bx      lr

.emax_loc_start_template:  .word .emax_start_template
.emax_loc_end_template:    .word .emax_end_template

        .data
        .p2align 4
//EMAX4A start .emax_start_template:
//EMAX4A ctl map_dist=1
//EMAX4A @0,0 while (320)             & ld (ri+=,4),r9  rgi[.label0:,] lmf[.label2:,0,320]
//EMAX4A @1,0                         & st r9,(ri+=,4)  rgi[.label1:,] lmw[.label3:,0,320]
//EMAX4A end .emax_end_template:
```

1.4 Instruction format

The instruction format for each unit is shown in Figure 1.5. “start”, “ctl” and “end” are common definition to all included instructions. “map_dist” specifies row-distance from previous instruction mapping (should be used to effectively reuse LMMs which are not affected by therow-distance). The execution count (number of cycles) the units should work is specified by “while” statement. 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. 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 and LMM.

```
start .emax_start_grapes:
ctl map_dist=1
while (320)
 Case 1:  @row#,col# ALU_OP rgi[labelX:,labelY:] & MEM_OP rgi[labelX:,labelY:] LMM_CONTROL
 Case 2:  @row#,col# ALU_OP rgi[labelX:,labelY:]
 Case 3:  @row#,col#                             & MEM_OP rgi[labelX:,labelY:] LMM_CONTROL
end .emax_end_grapes:
```



```
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), 6:LMT, 7:LMD */
    Uint len  : 20;
    Uint dist : 9;
    Uint pre_top;
  } ctl;
} insn[INSN_DEPTH][INSN_WIDTH]; /* 10words/unit */
```

Figure.1.5: Instruction format for each unit.

1.4.1 ALU_OP

```
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.1.6: Format of ALU operations.

The format of ALU operations is shown in Figure 1.6. 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).

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 1.1, Table 1.2 and Table 1.3 respectively.

Table 1.1, Table 1.2 and Table 1.3 describe all instructions available on EMAX2/intel. EMAX2asic/ZYNQ has limited instructions marked with '★'.

Table.1.1: EX1 operations (★ is available on EMAX2asic).

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.1.2: EX2 operations (★ is available on EMAX2asic).

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 ⇒ H16bit
suml	(-/Xr)	H16bit+L16bit ⇒ L16bit

Table.1.3: SFT operations (★ is available on EMAX2asic).

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)

1.4.2 MEM_OP

Table.1.4: Memory operations (★ is available on EMAX2asic).

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

Memory operations are listed in Table 1.4. Table 1.4 describes all instructions available on EMAX2/intel. EMAX2asic/ZYNQ has limited instructions marked with ' ★'.

1.4.3 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 ARM can identify the location where some initial value should be set before the instructions are sent to EMAX2. 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”

1.4.4 LMM_CONTROL

LMM_CONTROL specifies the type, start address of LMM, distance, start address of main memory and length for transmitting data between main memory and LMM. 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:, dist, length]
This format is used for reading data from main memory to LMM. “label:” corresponds to the start address of an array in LMM. “dist” expresses the distance between elements of array by 4<<dist. In the case of normal array with single words, dist should be 0. “length” is the number of words to prefetch.

lmw [label:, dist, length]
This format is used for writing data from LMM to main memory after array execution completes.

lmx [label:, dist, length]
This format is used when the LMM should execute both of reading and writing.

lmf [label:, dist, length]
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 again.

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

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

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

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

```
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.1.7: Lower level configuration data (1/2).

1.5 Application binary interface

1.5.1 Configuration data

The binary data of the instructions is not directly sent to each unit. Each unit requires lower level information representing input signals for all selectors and tri-state buffers as shown in Figure 1.7 and Figure 1.8, so that no complicated instruction decoder is required on each unit. Such binary translation from the instructions to the configuration data will be performed by EMAX2asic compiler (conv-a2b).

```
  /* 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.1.8: Lower level configuration data (2/2).

1.5.2 Initial values of registers

```
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.1.9: 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 1.9 are calculated by EMAX2asic compiler and transmitted to EMAX2asic with other source data.

1.5.3 LMM information

```
struct lmmi {
  struct e2ctl {
    Uint v    : 3; /* 0:nop, 1:LMR, 2:LMW, 3:LMX, 5:LMF(force read) */
    Uint len  : 20;
    Uint dist : 9;
    Uint top;
  } e2ctl; /* 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 skip   :  1; /* 0:fsm.v read DDR3, 1:fsm.v skip DDR3 */
    Uint base_offset : 1; /* for macro-pipelining 0:regv=6000,lmmi=a000 1:regv=8000,lmmi=c000 */
                         /* See proj-arm32/sample/stencil-pipe/emax2.c and
                                proj-emax/fpga/step4008-GP6X-fpu/RTL/pe0/fsm.v */
    Uint _dmy1 : 12;
    Uint _dmy2 : 32;
  } ddr3_tlb; /* 2word */
} lmmi[UNIT_DEPTH][UNIT_WIDTH]; /* 128bit(4words)/unit */
```

Figure.1.10: LMM Information.

Unlike the configuration data, LMM information should be transmited to EMAX2 every time before starting execution. LMM information shown in Figure 1.10 are calculated by EMAX2asic compiler and transmitted to EMAX2asic with other source data.

1.6 Example

1.6.1 Grapes kernel

```
void grapes( b, a, c )
     float *b, *a, *c;
{
  int i;
  float t0, t1;

  for (i=0; i<WD; i++) {
    *(c+i) = *(b+HT*WD  +WD  +i) * *(a+DP*HT*WD*9+HT*WD  +WD  +i)
           + *(b+HT*WD     +1+i) * *(a+DP*HT*WD*8+HT*WD     +1+i)
           + *(b+HT*WD       +i) * *(a+DP*HT*WD*7+HT*WD       +i)
           + *(b+HT*WD     -1+i) * *(a+DP*HT*WD*6+HT*WD     -1+i)
           + *(b+HT*WD  -WD  +i) * *(a+DP*HT*WD*5+HT*WD  -WD  +i)
           + *(b        +WD+1+i) * *(a+DP*HT*WD*4        +WD+1+i)
           + *(b        +WD  +i) * *(a+DP*HT*WD*3        +WD  +i)
           + *(b        +WD-1+i) * *(a+DP*HT*WD*2        +WD-1+i)
           + *(b           +1+i) * *(a+DP*HT*WD*1           +1+i)
           + *(b             +i)
           + *(b           -1+i) * *(a-DP*HT*WD*1           -1+i)
           + *(b        -WD+1+i) * *(a-DP*HT*WD*2        -WD+1+i)
           + *(b        -WD  +i) * *(a-DP*HT*WD*3        -WD  +i)
           + *(b        -WD-1+i) * *(a-DP*HT*WD*4        -WD-1+i)
           + *(b-HT*WD  +WD  +i) * *(a-DP*HT*WD*5-HT*WD  +WD  +i)
           + *(b-HT*WD     +1+i) * *(a-DP*HT*WD*6-HT*WD     +1+i)
           + *(b-HT*WD       +i) * *(a-DP*HT*WD*7-HT*WD       +i)
           + *(b-HT*WD     -1+i) * *(a-DP*HT*WD*8-HT*WD     -1+i)
           + *(b-HT*WD  -WD  +i) * *(a-DP*HT*WD*9-HT*WD  -WD  +i);
  }
}
```

```
        .text
        .align 2
        .global  grapes
        .type   grapes,%function
grapes: /* void grapes( b, a, c ) float *b, *a, *c; */
        stmfd   sp!, {r4, r5, r6, r7, r8, r9, r10, fp, lr}
        sub     sp, sp, #20
        add     r0, r0, #-409600;add r0, r0, #-4
        ldr     r3, .emax_loc_rgi_p0____grapes;        str     r0, [r3]
        add     r0, r0, #4;add r0, r0, #-1280
        ldr     r3, .emax_loc_lmrla_PREV_B0_grapes;    str     r0, [r3]
        add     r0, r0, #1280
        ldr     r3, .emax_loc_lmrla_CURR_B0_grapes;    str     r0, [r3]
        add     r0, r0, #1280
        ldr     r3, .emax_loc_lmrla_NEXT_B0_grapes;    str     r0, [r3]
        add     r0, r0, #-1280;add r0, r0, #409600;add r0, r0, #-4
        ldr     r3, .emax_loc_rgi_p1____grapes;        str     r0, [r3]
        add     r0, r0, #4;add r0, r0, #-1280
        ldr     r3, .emax_loc_lmrla_PREV_B1_grapes;    str     r0, [r3]
        add     r0, r0, #1280
        ldr     r3, .emax_loc_lmrla_CURR_B1_grapes;    str     r0, [r3]
        add     r0, r0, #1280
        ldr     r3, .emax_loc_lmrla_NEXT_B1_grapes;    str     r0, [r3]
        add     r0, r0, #-1280;add r0, r0, #409600;add r0, r0, #-4
        ldr     r3, .emax_loc_rgi_p2____grapes;        str     r0, [r3]
        add     r0, r0, #4;add r0, r0, #-1280
        ldr     r3, .emax_loc_lmrla_PREV_B2_grapes;    str     r0, [r3]
        add     r0, r0, #1280
        ldr     r3, .emax_loc_lmrla_CURR_B2_grapes;    str     r0, [r3]
        add     r0, r0, #1280
        ldr     r3, .emax_loc_lmrla_NEXT_B2_grapes;    str     r0, [r3]
        add     r1, r1, #-8192000;add r1, r1, #-8192000;add r1, r1, #-8192000;add r1, r1, #-8192000;add r1, r1, #-8192000
        add     r1, r1, #-8192000;add r1, r1, #-8192000;add r1, r1, #-8192000;add r1, r1, #-8192000;add r1, r1, #-409600;add r1, r1, #-1280
        ldr     r3, .emax_loc_lmrla_A0_grapes;         str     r1, [r3]
        add     r1, r1, #-4
        ldr     r3, .emax_loc_rgi_A0_grapes;           str     r1, [r3]
        add     r1, r1, #4;add r1, r1, #8192000;add r1, r1, #1280;add r1, r1, #-4
        ldr     r3, .emax_loc_lmrla_A1_grapes;         str     r1, [r3]
        add     r1, r1, #-4
        ldr     r3, .emax_loc_rgi_A1_grapes;           str     r1, [r3]
        add     r1, r1, #4;add r1, r1, #8192000;add r1, r1, #4
        ldr     r3, .emax_loc_lmrla_A2_grapes;         str     r1, [r3]
        add     r1, r1, #-4
        ldr     r3, .emax_loc_rgi_A2_grapes;           str     r1, [r3]
        add     r1, r1, #4;add r1, r1, #8192000;add r1, r1, #4
        ldr     r3, .emax_loc_lmrla_A3_grapes;         str     r1, [r3]
        add     r1, r1, #-4
        ldr     r3, .emax_loc_rgi_A3_grapes;           str     r1, [r3]
        add     r1, r1, #4;add r1, r1, #8192000;add r1, r1, #1280;add r1, r1, #-4
        ldr     r3, .emax_loc_lmrla_A4_grapes;         str     r1, [r3]
        add     r1, r1, #-4
        ldr     r3, .emax_loc_rgi_A4_grapes;           str     r1, [r3]
        add     r1, r1, #4;add r1, r1, #8192000;add r1, r1, #409600;add r1, r1, #-2560;add r1, r1, #-4
        ldr     r3, .emax_loc_lmrla_A5_grapes;         str     r1, [r3]
        add     r1, r1, #-4
        ldr     r3, .emax_loc_rgi_A5_grapes;           str     r1, [r3]
        add     r1, r1, #4;add r1, r1, #8192000;add r1, r1, #4
        ldr     r3, .emax_loc_lmrla_A6_grapes;         str     r1, [r3]
        add     r1, r1, #-4
        ldr     r3, .emax_loc_rgi_A6_grapes;           str     r1, [r3]
        add     r1, r1, #4;add r1, r1, #8192000;add     r1, r1, #4
        ldr     r3, .emax_loc_lmrla_A7_grapes;         str     r1, [r3]
        add     r1, r1, #-4
        ldr     r3, .emax_loc_rgi_A7_grapes;           str     r1, [r3]
        add     r1, r1, #4;add r1, r1, #8192000;add r1, r1, #1280;add r1, r1, #-8
        ldr     r3, .emax_loc_lmrla_A8_grapes;         str     r1, [r3]
        add     r1, r1, #-4
        ldr     r3, .emax_loc_rgi_A8_grapes;           str     r1, [r3]
        add     r1, r1, #4;add r1, r1, #8192000;add r1, r1, #4;add r1, r1, #8192000;add r1, r1, #4
        ldr     r3, .emax_loc_lmrla_A10_grapes;        str     r1, [r3]
        add     r1, r1, #-4
        ldr     r3, .emax_loc_rgi_A10_grapes;          str     r1, [r3]
        add     r1, r1, #4;add r1, r1, #8192000;add r1, r1, #1280;add r1, r1, #-8
        ldr     r3, .emax_loc_lmrla_A11_grapes;        str     r1, [r3]
        add     r1, r1, #-4
        ldr     r3, .emax_loc_rgi_A11_grapes;          str     r1, [r3]
        add     r1, r1, #4;add r1, r1, #8192000;add r1, r1, #4
        ldr     r3, .emax_loc_lmrla_A12_grapes;        str     r1, [r3]
        add     r1, r1, #-4
        ldr     r3, .emax_loc_rgi_A12_grapes;          str     r1, [r3]
        add     r1, r1, #4;add r1, r1, #8192000;add r1, r1, #4
        ldr     r3, .emax_loc_lmrla_A13_grapes;        str     r1, [r3]
        add     r1, r1, #-4
        ldr     r3, .emax_loc_rgi_A13_grapes;          str     r1, [r3]
        add     r1, r1, #4;add r1, r1, #8192000;add r1, r1, #409600;add r1, r1, #-2560;add r1, r1, #-4
        ldr     r3, .emax_loc_lmrla_A14_grapes;        str     r1, [r3]
        add     r1, r1, #-4
        ldr     r3, .emax_loc_rgi_A14_grapes;          str     r1, [r3]
        add     r1, r1, #4;add r1, r1, #8192000;add r1, r1, #1280;add r1, r1, #-4
        ldr     r3, .emax_loc_lmrla_A15_grapes;        str     r1, [r3]
        add     r1, r1, #-4
        ldr     r3, .emax_loc_rgi_A15_grapes;          str     r1, [r3]
        add     r1, r1, #4;add     r1, r1, #8192000;add r1, r1, #4
        ldr     r3, .emax_loc_lmrla_A16_grapes;        str     r1, [r3]
        add     r1, r1, #-4
        ldr     r3, .emax_loc_rgi_A16_grapes;          str     r1, [r3]
        add     r1, r1, #4;add r1, r1, #8192000;add r1, r1, #4
        ldr     r3, .emax_loc_lmrla_A17_grapes;        str     r1, [r3]
        add     r1, r1, #-4
        ldr     r3, .emax_loc_rgi_A17_grapes;          str     r1, [r3]
        add     r1, r1, #4;add r1, r1, #8192000;add r1, r1, #1280;add r1, r1, #-4
        ldr     r3, .emax_loc_lmrla_A18_grapes;        str     r1, [r3]
        add     r1, r1, #-4
        ldr     r3, .emax_loc_rgi_A18_grapes;          str     r1, [r3]
        add     r1, r1, #4
        ldr     r3, .emax_loc_lmwla_store_grapes;      str     r2, [r3]
        add     r2, r2, #-4
        ldr     r3, .emax_loc_rgi_store_grapes;        str     r2, [r3]
        ldr     r0, .emax_loc_start_grapes
        ldr     r1, .emax_loc_end_grapes
        bl      emax2_start
        mov     r0, #0;add sp, sp, #20;ldmfd sp!, {r4, r5, r6, r7, r8, r9, r10, fp, lr}
        bx      lr
```

```
.emax_loc_start_grapes:         .word   .emax_start_grapes
.emax_loc_rgi_p0____grapes:     .word   .emax_rgi_p0____grapes
.emax_loc_lmrla_PREV_B0_grapes: .word   .emax_lmrla_PREV_B0_grapes
.emax_loc_rgi_A0_grapes:        .word   .emax_rgi_A0_grapes
.emax_loc_lmrla_A0_grapes:      .word   .emax_lmrla_A0_grapes
.emax_loc_lmrla_CURR_B0_grapes: .word   .emax_lmrla_CURR_B0_grapes
.emax_loc_rgi_A1_grapes:        .word   .emax_rgi_A1_grapes
.emax_loc_lmrla_A1_grapes:      .word   .emax_lmrla_A1_grapes
.emax_loc_rgi_A2_grapes:        .word   .emax_rgi_A2_grapes
.emax_loc_lmrla_A2_grapes:      .word   .emax_lmrla_A2_grapes
.emax_loc_lmrla_NEXT_B0_grapes: .word   .emax_lmrla_NEXT_B0_grapes
.emax_loc_rgi_A3_grapes:        .word   .emax_rgi_A3_grapes
.emax_loc_lmrla_A3_grapes:      .word   .emax_lmrla_A3_grapes
.emax_loc_rgi_A4_grapes:        .word   .emax_rgi_A4_grapes
.emax_loc_lmrla_A4_grapes:      .word   .emax_lmrla_A4_grapes
.emax_loc_rgi_p1____grapes:     .word   .emax_rgi_p1____grapes
.emax_loc_rgi_A5_grapes:        .word   .emax_rgi_A5_grapes
.emax_loc_lmrla_A5_grapes:      .word   .emax_lmrla_A5_grapes
.emax_loc_rgi_A6_grapes:        .word   .emax_rgi_A6_grapes
.emax_loc_lmrla_A6_grapes:      .word   .emax_lmrla_A6_grapes
.emax_loc_lmrla_PREV_B1_grapes: .word   .emax_lmrla_PREV_B1_grapes
.emax_loc_rgi_A7_grapes:        .word   .emax_rgi_A7_grapes
.emax_loc_lmrla_A7_grapes:      .word   .emax_lmrla_A7_grapes
.emax_loc_rgi_A8_grapes:        .word   .emax_rgi_A8_grapes
.emax_loc_lmrla_A8_grapes:      .word   .emax_lmrla_A8_grapes
.emax_loc_lmrla_CURR_B1_grapes: .word   .emax_lmrla_CURR_B1_grapes
.emax_loc_rgi_A10_grapes:       .word   .emax_rgi_A10_grapes
.emax_loc_lmrla_A10_grapes:     .word   .emax_lmrla_A10_grapes
.emax_loc_rgi_A11_grapes:       .word   .emax_rgi_A11_grapes
.emax_loc_lmrla_A11_grapes:     .word   .emax_lmrla_A11_grapes
.emax_loc_lmrla_NEXT_B1_grapes: .word   .emax_lmrla_NEXT_B1_grapes
.emax_loc_rgi_A12_grapes:       .word   .emax_rgi_A12_grapes
.emax_loc_lmrla_A12_grapes:     .word   .emax_lmrla_A12_grapes
.emax_loc_rgi_A13_grapes:       .word   .emax_rgi_A13_grapes
.emax_loc_lmrla_A13_grapes:     .word   .emax_lmrla_A13_grapes
.emax_loc_rgi_p2____grapes:     .word   .emax_rgi_p2____grapes
.emax_loc_rgi_A14_grapes:       .word   .emax_rgi_A14_grapes
.emax_loc_lmrla_A14_grapes:     .word   .emax_lmrla_A14_grapes
.emax_loc_lmrla_PREV_B2_grapes: .word   .emax_lmrla_PREV_B2_grapes
.emax_loc_rgi_A15_grapes:       .word   .emax_rgi_A15_grapes
.emax_loc_lmrla_A15_grapes:     .word   .emax_lmrla_A15_grapes
.emax_loc_rgi_A16_grapes:       .word   .emax_rgi_A16_grapes
.emax_loc_lmrla_A16_grapes:     .word   .emax_lmrla_A16_grapes
.emax_loc_lmrla_CURR_B2_grapes: .word   .emax_lmrla_CURR_B2_grapes
.emax_loc_rgi_A17_grapes:       .word   .emax_rgi_A17_grapes
.emax_loc_lmrla_A17_grapes:     .word   .emax_lmrla_A17_grapes
.emax_loc_rgi_A18_grapes:       .word   .emax_rgi_A18_grapes
.emax_loc_lmrla_A18_grapes:     .word   .emax_lmrla_A18_grapes
.emax_loc_lmrla_NEXT_B2_grapes: .word   .emax_lmrla_NEXT_B2_grapes
.emax_loc_rgi_store_grapes:     .word   .emax_rgi_store_grapes
.emax_loc_lmwla_store_grapes:   .word   .emax_lmwla_store_grapes
.emax_loc_end_grapes:           .word   .emax_end_grapes

        .data
        .p2align 4
//EMAX4A start .emax_start_grapes:
//EMAX4A ctl map_dist=1
//EMAX4A @0,0  while (320)
//EMAX4A @0,1  add (ri+=,4),r9       rgi[.emax_rgi_p0____grapes:,] &
//EMAX4A @1,0                                                      & ld (r9,-1276),r19 lmr[.emax_lmrla_PREV_B0_grapes:,0,320]
//EMAX4A @1,1                                                      & ld (r9,-1280),r20
//EMAX4A @1,2                                                      & ld (ri+=,4),r0    rgi[.emax_rgi_A0_grapes:,]   lmr[.emax_lmrla_A0_grapes:,0,320]
//EMAX4A @2,0                                                      & ld (r9,4),r23     lmr[.emax_lmrla_CURR_B0_grapes:,0,320]
//EMAX4A @2,1                                   ld (r9,-4),r21     & ld (r9,0),r22
//EMAX4A @2,2  fmul (r20,r0),r30                                   & ld (ri+=,4),r1    rgi[.emax_rgi_A1_grapes:,]   lmr[.emax_lmrla_A1_grapes:,0,320]
//EMAX4A @2,3                                                      & ld (ri+=,4),r2    rgi[.emax_rgi_A2_grapes:,]   lmr[.emax_lmrla_A2_grapes:,0,320]
//EMAX4A @3,0                                                      & ld (r9,1284),r19  lmr[.emax_lmrla_NEXT_B0_grapes:,0,320]
//EMAX4A @3,1                                                      & ld (r9,1280),r24
//EMAX4A @3,2  fma3 (r21,r1,r30),r30                               & ld (ri+=,4),r3    rgi[.emax_rgi_A3_grapes:,]   lmr[.emax_lmrla_A3_grapes:,0,320]
//EMAX4A @3,3  fmul (r22,r2),r31                                   & ld (ri+=,4),r4    rgi[.emax_rgi_A4_grapes:,]   lmr[.emax_lmrla_A4_grapes:,0,320]
//EMAX4A @4,0  add (ri+=,4),r9       rgi[.emax_rgi_p1____grapes:,] &
//EMAX4A @4,2  fma3 (r23,r3,r30),r30                               & ld (ri+=,4),r5    rgi[.emax_rgi_A5_grapes:,]   lmr[.emax_lmrla_A5_grapes:,0,320]
//EMAX4A @4,3  fma3 (r24,r4,r31),r31                               & ld (ri+=,4),r6    rgi[.emax_rgi_A6_grapes:,]   lmr[.emax_lmrla_A6_grapes:,0,320]
//EMAX4A @5,0                                                      & ld (r9,-1276),r27 lmr[.emax_lmrla_PREV_B1_grapes:,0,320]
//EMAX4A @5,1                                   ld (r9,-1284),r25  & ld (r9,-1280),r26
//EMAX4A @5,2                                                      & ld (ri+=,4),r7    rgi[.emax_rgi_A7_grapes:,]   lmr[.emax_lmrla_A7_grapes:,0,320]
//EMAX4A @5,3                                                      & ld (ri+=,4),r8    rgi[.emax_rgi_A8_grapes:,]   lmr[.emax_lmrla_A8_grapes:,0,320]
//EMAX4A @6,0                                                      & ld (r9,4),r20     lmr[.emax_lmrla_CURR_B1_grapes:,0,320]
//EMAX4A @6,1                                   ld (r9,-4),r28     & ld (r9,0),r29
//EMAX4A @6,2  fma3 (r25,r5,r30),r30                               & ld (ri+=,4),r10   rgi[.emax_rgi_A10_grapes:,]   lmr[.emax_lmrla_A10_grapes:,0,320]
//EMAX4A @6,3  fma3 (r26,r6,r31),r31                               & ld (ri+=,4),r11   rgi[.emax_rgi_A11_grapes:,]   lmr[.emax_lmrla_A11_grapes:,0,320]
//EMAX4A @7,0                                                      & ld (r9,1284),r23  lmr[.emax_lmrla_NEXT_B1_grapes:,0,320]
//EMAX4A @7,1                                   ld (r9,1276),r21   & ld (r9,1280),r22
//EMAX4A @7,2  fma3 (r27,r7,r30),r30                               & ld (ri+=,4),r12   rgi[.emax_rgi_A12_grapes:,]   lmr[.emax_lmrla_A12_grapes:,0,320]
//EMAX4A @7,3  fma3 (r28,r8,r31),r31                               & ld (ri+=,4),r13   rgi[.emax_rgi_A13_grapes:,]   lmr[.emax_lmrla_A13_grapes:,0,320]
//EMAX4A @8,1  fadd (r29,r30),r30                                  &
//EMAX4A @8,2  fma3 (r20,r10,r31),r31                              &
//EMAX4A @8,3  add (ri+=,4),r9       rgi[.emax_rgi_p2____grapes:,] &
//EMAX4A @9,1  fma3 (r21,r11,r30),r30                              & ld (ri+=,4),r14   rgi[.emax_rgi_A14_grapes:,]   lmr[.emax_lmrla_A14_grapes:,0,320]
//EMAX4A @9,2  fma3 (r23,r13,r31),r31                              & ld (r9,-1276),r19 lmr[.emax_lmrla_PREV_B2_grapes:,0,320]
//EMAX4A @9,3                                                      & ld (r9,-1280),r24
//EMAX4A @10,0                                                     & ld (ri+=,4),r15   rgi[.emax_rgi_A15_grapes:,]   lmr[.emax_lmrla_A15_grapes:,0,320]
//EMAX4A @10,1 fma3 (r22,r12,r30),r30                              & ld (ri+=,4),r16   rgi[.emax_rgi_A16_grapes:,]   lmr[.emax_lmrla_A16_grapes:,0,320]
//EMAX4A @10,2 fma3 (r24,r14,r31),r31                              & ld (r9,4),r27     lmr[.emax_lmrla_CURR_B2_grapes:,0,320]
//EMAX4A @10,3                                  ld (r9,-4),r25     & ld (r9,0),r26
//EMAX4A @11,0 fma3 (r25,r15,r31),r31                              & ld (ri+=,4),r17   rgi[.emax_rgi_A17_grapes:,]   lmr[.emax_lmrla_A17_grapes:,0,320]
//EMAX4A @11,1                                                     & ld (ri+=,4),r18   rgi[.emax_rgi_A18_grapes:,]   lmr[.emax_lmrla_A18_grapes:,0,320]
//EMAX4A @11,2                                                     & ld (r9,1284),r19  lmr[.emax_lmrla_NEXT_B2_grapes:,0,320]
//EMAX4A @11,3                                                     & ld (r9,1280),r28
//EMAX4A @12,0 fma3 (r27,r17,r31),r31                              &
//EMAX4A @12,3 fma3 (r26,r16,r30),r30                              &
//EMAX4A @13,2 fadd (r30,r31),r31                                  &
//EMAX4A @13,3 fmul (r28,r18),r19                                  &
//EMAX4A @14,2 fadd (r31,r19),r19
//EMAX4A @15,2                                                     & st r19,(ri+=,4)   rgi[.emax_rgi_store_grapes:,] lmw[.emax_lmwla_store_grapes:,0,320]
//EMAX4A end .emax_end_grapes:
```

1.6.2 Jacobi kernel

```
void jacobi( B, C )
  float *B, *C;
{
    int x;
    float C1 = 0.2;
    float C2 = 0.3;

    for (x=0; x<WD; x++) {
      *(C+x) = C2 * (*(B-HT*WD+x) + *(B-WD+x) + *(B-1+x) + *(B+1+x) + *(B+WD+x) + *(B+HT*WD+x))
     + C1 * *(B+x);
    }
}
```

```
        .text
        .align 2
        .global  jacobi
        .type   jacobi,%function
jacobi:
        /* void jacobi( B, C ) float *B, *C; */
        stmfd   sp!, {r4, r5, r6, r7, r8, r9, r10, fp, lr}
        sub     sp, sp, #20

        add     r0, r0, #-409600
        add     r0, r0, #-4
        ldr     r3, .emax_loc_rgi_CURR_A0_ja
        str     r0, [r3]
        add     r0, r0, #4
        ldr     r3, .emax_loc_lmrla_CURR_A0_ja
        str     r0, [r3]
        add     r0, r0, #409600
        add     r0, r0, #-4
        ldr     r3, .emax_loc_rgi_p0_ja
        str     r0, [r3]
        add     r0, r0, #4
        add     r0, r0, #-1280
        ldr     r3, .emax_loc_lmrla_PREV_A1_ja
        str     r0, [r3]
        add     r0, r0, #1280
        ldr     r3, .emax_loc_lmrla_CURR_A1_ja
        str     r0, [r3]
        add     r0, r0, #1280
        ldr     r3, .emax_loc_lmrla_NEXT_A1_ja
        str     r0, [r3]
        add     r0, r0, #-1280
        add     r0, r0, #409600
        add     r0, r0, #-4
        ldr     r3, .emax_loc_rgi_CURR_A2_ja
        str     r0, [r3]
        add     r0, r0, #4
        ldr     r3, .emax_loc_lmrla_CURR_A2_ja
        str     r0, [r3]

        ldr     r3, .emax_loc_lmwma_store_ja
        str     r1, [r3]
        ldr     r3, .emax_loc_lmwla_store_ja
        str     r1, [r3]
        add     r1, r1, #-4
        ldr     r3, .emax_loc_rgi_store_ja
        str     r1, [r3]

        ldr     r0, .emax_loc_start_ja
        ldr     r1, .emax_loc_end_ja

        bl      emax2_start                           // start EMAX2asic

        mov     r0, #0
        add     sp, sp, #20
        ldmfd   sp!, {r4, r5, r6, r7, r8, r9, r10, fp, lr}
        bx      lr

.emax_loc_start_ja:             .word .emax_start_ja
.emax_loc_rgi_p0_ja:            .word   .emax_rgi_p0_ja
.emax_loc_rgi_CURR_A0_ja:       .word   .emax_rgi_CURR_A0_ja
.emax_loc_lmrla_CURR_A0_ja:     .word   .emax_lmrla_CURR_A0_ja
.emax_loc_lmrla_PREV_A1_ja:     .word   .emax_lmrla_PREV_A1_ja
.emax_loc_lmrla_CURR_A1_ja:     .word   .emax_lmrla_CURR_A1_ja
.emax_loc_lmrla_NEXT_A1_ja:     .word   .emax_lmrla_NEXT_A1_ja
.emax_loc_rgi_CURR_A2_ja:       .word   .emax_rgi_CURR_A2_ja
.emax_loc_lmrla_CURR_A2_ja:     .word   .emax_lmrla_CURR_A2_ja
.emax_loc_rgi_store_ja:         .word   .emax_rgi_store_ja
.emax_loc_lmwla_store_ja:       .word   .emax_lmwla_store_ja
.emax_loc_lmwma_store_ja:       .word   .emax_lmwma_store_ja
.emax_loc_end_ja:               .word   .emax_end_ja

        .data
        .p2align 4
//EMAX4A start .emax_start_ja:
//EMAX4A ctl map_dist=1
//EMAX4A @0,0 while (320)
//EMAX4A @0,1 add (ri+=,4),r0      rgi[.emax_rgi_p0_ja:,] & ld (ri+=,4),r1 rgi[.emax_rgi_CURR_A0_ja:,] lmr[.emax_lmrla_CURR_A0_ja:,0,320]
//EMAX4A @1,1 fmul (ri,r1),r10     rgi[0x3e99999a,]       & ld (r0,-1280),r2 lmr[.emax_lmrla_PREV_A1_ja:,0,320]
//EMAX4A @2,1 fma3 (ri,r2,r10),r10 rgi[0x3e99999a,]       & ld (r0,4),r5     lmr[.emax_lmrla_CURR_A1_ja:,0,320]
//EMAX4A @2,2                                             & ld (r0,0),r4
//EMAX4A @2,3                                             & ld (r0,-4),r3
//EMAX4A @3,1 fma3 (ri,r5,r10),r10 rgi[0x3e99999a,]       & ld (r0,1280),r6  lmr[.emax_lmrla_NEXT_A1_ja:,0,320]
//EMAX4A @3,2 fmul (ri,r4),r11     rgi[0x3e4ccccd,]       & ld (ri+=,4),r7 rgi[.emax_rgi_CURR_A2_ja:,] lmr[.emax_lmrla_CURR_A2_ja:,0,320]
//EMAX4A @3,3 fmul (ri,r3),r12     rgi[0x3e99999a,]       &
//EMAX4A @4,1 fma3 (ri,r6,r10),r10 rgi[0x3e99999a,]       &
//EMAX4A @4,2 fma3 (ri,r7,r11),r11 rgi[0x3e99999a,]       &
//EMAX4A @5,1 fadd (r10,r11),r10                          &
//EMAX4A @6,1 fadd (r10,r12),r10                          &
//EMAX4A @7,1                                             & st r10,(ri+=,4) rgi[.emax_rgi_store_ja:,] lmw[.emax_lmwla_store_ja:,0,320]
//EMAX4A end .emax_end_ja:
```

1.6.3 Fd6 kernel

```
void fd6( B, C )
  float *B, *C;
{
    int x;
    float t0, t1, t2;
    float C1 = 0.1;
    float C2 = 0.2;
    float C3 = 0.3;
    float C4 = 0.4;

    for (x = 0; x < WD; x++) {
      *(C+x) = C4 * (*(B-3*HT*WD+x) + *(B-3*WD+x) + *(B-3+x)
                   + *(B+3+x)       + *(B+3*WD+x) + *(B+3*HT*WD+x))
             + C3 * (*(B-2*HT*WD+x) + *(B-2*WD+x) + *(B-2+x)
                   + *(B+2+x)       + *(B+2*WD+x) + *(B+2*HT*WD+x))
             + C2 * (*(B-1*HT*WD+x) + *(B-1*WD+x) + *(B-1+x)
                   + *(B+1+x)       + *(B+1*WD+x) + *(B+1*HT*WD+x))
             + C1 * *(B+x);
    }
}
```

```
        .text
        .align 2
        .global  fd6
        .type   fd6,%function
fd6:    /* void fd6( B, C ) float *B, *C; */
        stmfd   sp!, {r4, r5, r6, r7, r8, r9, r10, fp, lr}
        sub     sp, sp, #20;add r0, r0, #-1228800;add r0, r0, #-4
        ldr     r3, .emax_loc_rgi_CURR_A0_fd6;        str     r0, [r3]
        add     r0, r0, #4
        ldr     r3, .emax_loc_lmrla_CURR_A0_fd6;      str     r0, [r3]
        add     r0, r0, #409600;add r0, r0, #-4
        ldr     r3, .emax_loc_rgi_CURR_A1_fd6;        str     r0, [r3]
        add     r0, r0, #4
        ldr     r3, .emax_loc_lmrla_CURR_A1_fd6;      str     r0, [r3]
        add     r0, r0, #409600;add r0, r0, #-4
        ldr     r3, .emax_loc_rgi_CURR_A2_fd6;        str     r0, [r3]
        add     r0, r0, #4
        ldr     r3, .emax_loc_lmrla_CURR_A2_fd6;      str     r0, [r3]
        add     r0, r0, #409600;add r0, r0, #-4
        ldr     r3, .emax_loc_rgi_p0____fd6;          str     r0, [r3]
        ldr     r3, .emax_loc_rgi_p1____fd6;          str     r0, [r3]
        add     r0, r0, #4;add r0, r0, #-3840
        ldr     r3, .emax_loc_lmrla_PREV3_A3_fd6;     str     r0, [r3]
        add     r0, r0, #1280
        ldr     r3, .emax_loc_lmrla_PREV2_A3_fd6;     str     r0, [r3]
        add     r0, r0, #1280
        ldr     r3, .emax_loc_lmrla_PREV1_A3_fd6;     str     r0, [r3]
        add     r0, r0, #1280
        ldr     r3, .emax_loc_lmrla_CURR_A3_fd6;      str     r0, [r3]
        add     r0, r0, #1280
        ldr     r3, .emax_loc_lmrla_NEXT1_A3_fd6;     str     r0, [r3]
        add     r0, r0, #1280
        ldr     r3, .emax_loc_lmrla_NEXT2_A3_fd6;     str     r0, [r3]
        add     r0, r0, #1280
        ldr     r3, .emax_loc_lmrla_NEXT3_A3_fd6;     str     r0, [r3]
        add     r0, r0, #-3840;add r0, r0, #409600;add r0, r0, #-4
        ldr     r3, .emax_loc_rgi_CURR_A4_fd6;        str     r0, [r3]
        add     r0, r0, #4
        ldr     r3, .emax_loc_lmrla_CURR_A4_fd6;      str     r0, [r3]
        add     r0, r0, #409600;add r0, r0, #-4
        ldr     r3, .emax_loc_rgi_CURR_A5_fd6;        str     r0, [r3]
        add     r0, r0, #4
        ldr     r3, .emax_loc_lmrla_CURR_A5_fd6;      str     r0, [r3]
        add     r0, r0, #409600;add r0, r0, #-4
        ldr     r3, .emax_loc_rgi_CURR_A6_fd6;        str     r0, [r3]
        add     r0, r0, #4
        ldr     r3, .emax_loc_lmrla_CURR_A6_fd6;      str     r0, [r3]
        ldr     r3, .emax_loc_lmwla_store_fd6;        str     r1, [r3]
        add     r1, r1, #-4
        ldr     r3, .emax_loc_rgi_store_fd6;          str     r1, [r3]
        ldr     r0, .emax_loc_start_fd6;ldr r1, .emax_loc_end_fd6;bl emax2_start
        mov     r0, #0;add sp, sp, #20; ldmfd   sp!, {r4, r5, r6, r7, r8, r9, r10, fp, lr};bx lr

.emax_loc_start_fd6:            .word   .emax_start_fd6
.emax_loc_rgi_CURR_A0_fd6:      .word   .emax_rgi_CURR_A0_fd6
.emax_loc_lmrla_CURR_A0_fd6:    .word   .emax_lmrla_CURR_A0_fd6
.emax_loc_rgi_p0____fd6:        .word   .emax_rgi_p0____fd6
.emax_loc_rgi_CURR_A1_fd6:      .word   .emax_rgi_CURR_A1_fd6
.emax_loc_lmrla_CURR_A1_fd6:    .word   .emax_lmrla_CURR_A1_fd6
.emax_loc_rgi_CURR_A2_fd6:      .word   .emax_rgi_CURR_A2_fd6
.emax_loc_lmrla_CURR_A2_fd6:    .word   .emax_lmrla_CURR_A2_fd6
.emax_loc_lmrla_PREV3_A3_fd6:   .word   .emax_lmrla_PREV3_A3_fd6
.emax_loc_lmrla_PREV2_A3_fd6:   .word   .emax_lmrla_PREV2_A3_fd6
.emax_loc_lmrla_PREV1_A3_fd6:   .word   .emax_lmrla_PREV1_A3_fd6
.emax_loc_rgi_p1____fd6:        .word   .emax_rgi_p1____fd6
.emax_loc_lmrla_CURR_A3_fd6:    .word   .emax_lmrla_CURR_A3_fd6
.emax_loc_lmrla_NEXT1_A3_fd6:   .word   .emax_lmrla_NEXT1_A3_fd6
.emax_loc_lmrla_NEXT2_A3_fd6:   .word   .emax_lmrla_NEXT2_A3_fd6
.emax_loc_lmrla_NEXT3_A3_fd6:   .word   .emax_lmrla_NEXT3_A3_fd6
.emax_loc_rgi_CURR_A4_fd6:      .word   .emax_rgi_CURR_A4_fd6
.emax_loc_lmrla_CURR_A4_fd6:    .word   .emax_lmrla_CURR_A4_fd6
.emax_loc_rgi_CURR_A5_fd6:      .word   .emax_rgi_CURR_A5_fd6
.emax_loc_lmrla_CURR_A5_fd6:    .word   .emax_lmrla_CURR_A5_fd6
.emax_loc_rgi_CURR_A6_fd6:      .word   .emax_rgi_CURR_A6_fd6
.emax_loc_lmrla_CURR_A6_fd6:    .word   .emax_lmrla_CURR_A6_fd6
.emax_loc_rgi_store_fd6:        .word   .emax_rgi_store_fd6
.emax_loc_lmwla_store_fd6:      .word   .emax_lmwla_store_fd6
.emax_loc_end_fd6:              .word   .emax_end_fd6

        .data
        .p2align 4
//EMAX4A start .emax_start_fd6:
//EMAX4A ctl map_dist=1
//EMAX4A @0,0  while (320)                                  & ld (ri+=,4),r0     rgi[.emax_rgi_CURR_A0_fd6:,] lmr[.emax_lmrla_CURR_A0_fd6:,0,320]
//EMAX4A @0,1  add (ri+=,4),r10     rgi[.emax_rgi_p0____fd6:,]  & ld (ri+=,4),r1     rgi[.emax_rgi_CURR_A1_fd6:,] lmr[.emax_lmrla_CURR_A1_fd6:,0,320]
//EMAX4A @0,2                                               & ld (ri+=,4),r2     rgi[.emax_rgi_CURR_A2_fd6:,] lmr[.emax_lmrla_CURR_A2_fd6:,0,320]
//EMAX4A @1,0  fmul(ri,r0),r20      rgi[0x3ecccccd,]            & ld (r10,-3840),r3  lmr[.emax_lmrla_PREV3_A3_fd6:,0,320]
//EMAX4A @1,1  fmul(ri,r1),r21      rgi[0x3e99999a,]            &
//EMAX4A @1,2  fmul(ri,r2),r22      rgi[0x3e4ccccd,]            &
//EMAX4A @2,0  fma3(ri,r3,r20),r20  rgi[0x3ecccccd,]            & ld (r10,-2560),r4  lmr[.emax_lmrla_PREV2_A3_fd6:,0,320]
//EMAX4A @2,1  fadd(r21,r22),r21
//EMAX4A @3,0  fma3(ri,r4,r20),r20  rgi[0x3e99999a,]            & ld (r10,-1280),r5  lmr[.emax_lmrla_PREV1_A3_fd6:,0,320]
//EMAX4A @3,1
//EMAX4A @4,0  add (ri+=,4),r10     rgi[.emax_rgi_p1____fd6:,]  & ld (r10,12),r12    lmr[.emax_lmrla_CURR_A3_fd6:,0,320]
//EMAX4A @4,1                                   ld (r10,4),r10  & ld (r10,8),r11
//EMAX4A @4,2                                   ld (r10,-4),r8  & ld (r10,0),r9
//EMAX4A @4,3                                   ld (r10,-12),r6 & ld (r10,-8),r7
//EMAX4A @5,0  fma3(ri,r5,r20),r20  rgi[0x3e4ccccd,]            & ld (r10,1280),r13  lmr[.emax_lmrla_NEXT1_A3_fd6:,0,320]
//EMAX4A @5,1  fma3(ri,r10,r21),r21 rgi[0x3e4ccccd,]            &
//EMAX4A @5,2  fmul(ri,r8),r22      rgi[0x3e4ccccd,]            &
//EMAX4A @5,3  fmul(ri,r6),r23      rgi[0x3ecccccd,]            &
//EMAX4A @6,0  fma3(ri,r12,r20),r20 rgi[0x3ecccccd,]            & ld (r10,2560),r14  lmr[.emax_lmrla_NEXT2_A3_fd6:,0,320]
//EMAX4A @6,1  fma3(ri,r11,r21),r21 rgi[0x3e99999a,]            &
//EMAX4A @6,2  fma3(ri,r9,r22),r22  rgi[0x3dcccccd,]            &
//EMAX4A @6,3  fma3(ri,r7,r23),r23  rgi[0x3e99999a,]            &
//EMAX4A @7,0                                               & ld (r10,3840),r15  lmr[.emax_lmrla_NEXT3_A3_fd6:,0,320]
//EMAX4A @7,1                                               & ld (ri+=,4),r16    rgi[.emax_rgi_CURR_A4_fd6:,] lmr[.emax_lmrla_CURR_A4_fd6:,0,320]
//EMAX4A @7,2  fma3(ri,r13,r22),r22 rgi[0x3e4ccccd,]            & ld (ri+=,4),r17    rgi[.emax_rgi_CURR_A5_fd6:,] lmr[.emax_lmrla_CURR_A5_fd6:,0,320]
//EMAX4A @7,3  fma3(ri,r14,r23),r23 rgi[0x3e99999a,]            & ld (ri+=,4),r18    rgi[.emax_rgi_CURR_A6_fd6:,] lmr[.emax_lmrla_CURR_A6_fd6:,0,320]
//EMAX4A @8,0  fma3(ri,r15,r20),r20 rgi[0x3ecccccd,]            &
//EMAX4A @8,1  fma3(ri,r16,r21),r21 rgi[0x3e4ccccd,]            &
//EMAX4A @8,2  fma3(ri,r17,r22),r22 rgi[0x3e99999a,]            &
//EMAX4A @8,3  fma3(ri,r18,r23),r23 rgi[0x3ecccccd,]            &
//EMAX4A @9,1  fadd(r20,r21),r21                            &
//EMAX4A @9,2  fadd(r22,r23),r22                            &
//EMAX4A @10,2 fadd(r21,r22),r22                            &
//EMAX4A @11,0                                              & st r22,(ri+=,4)    rgi[.emax_rgi_store_fd6:,] lmw[.emax_lmwla_store_fd6:,0,320]
//EMAX4A end .emax_end_fd6:
```

1.6.4 Resid kernel

```
void resid( B, C, D )
      float *B, *C, *D;
{
    int x;

    float A0 = 0.1;
    float A1 = 0.2;
    float A2 = 0.3;
    float A3 = 0.4;

    for (x=0; x<WD; x++) {
        *(D+x) = *(C+x)
                - A0 * *(B+x)
                - A1 * ( *(B-HT*WD     +x) + *(B+HT*WD   +x)
                       + *(B      -WD  +x) + *(B      +WD+x)
                       + *(B         -1+x) + *(B       +1+x) )
                - A2 * ( *(B-HT*WD-WD  +x) + *(B+HT*WD-WD+x) + *(B-HT*WD+WD+x) + *(B+HT*WD+WD+x)
                       + *(B      -WD-1+x) + *(B    +WD-1+x) + *(B    -WD+1+x) + *(B    +WD+1+x)
                       + *(B-HT*WD   -1+x) + *(B-HT*WD +1+x) + *(B+HT*WD -1+x) + *(B+HT*WD +1+x) )
                - A3 * ( *(B-HT*WD-WD-1+x) + *(B+HT*WD-WD-1+x)
                       + *(B-HT*WD+WD-1+x) + *(B-HT*WD-WD+1+x)
                       + *(B+HT*WD+WD-1+x) + *(B+HT*WD-WD+1+x)
       + *(B-HT*WD+WD+1+x) + *(B+HT*WD+WD+1+x) );
    }
}
```

```
        .text
        .align 2
        .global  resid
        .type   resid,%function
resid:
        /* void resid( B, C, D ) float *B, *C, *D; */
        stmfd   sp!, {r4, r5, r6, r7, r8, r9, r10, fp, lr}
        sub     sp, sp, #20
        add     r0, r0, #-409600;add r0, r0, #-4
        ldr     r3, .emax_loc_rgi_p0____resid;       str     r0, [r3]
        add     r0, r0, #4;add r0, r0, #-1280
        ldr     r3, .emax_loc_lmrla_PREV_B0_resid;   str     r0, [r3]
        add     r0, r0, #1280
        ldr     r3, .emax_loc_lmrla_CURR_B0_resid;   str     r0, [r3]
        add     r0, r0, #1280
        ldr     r3, .emax_loc_lmrla_NEXT_B0_resid;   str     r0, [r3]
        add     r0, r0, #409600;add r0, r0, #-4
        ldr     r3, .emax_loc_rgi_p1____resid;       str     r0, [r3]
        add     r0, r0, #4;add r0, r0, #-1280
        ldr     r3, .emax_loc_lmrla_PREV_B1_resid;   str     r0, [r3]
        add     r0, r0, #1280
        ldr     r3, .emax_loc_lmrla_CURR_B1_resid;   str     r0, [r3]
        add     r0, r0, #1280
        ldr     r3, .emax_loc_lmrla_NEXT_B1_resid;   str     r0, [r3]
        add     r0, r0, #409600;add r0, r0, #-4
        ldr     r3, .emax_loc_rgi_p2____resid;       str     r0, [r3]
        add     r0, r0, #4;add r0, r0, #-1280
        ldr     r3, .emax_loc_lmrla_PREV_B2_resid;   str     r0, [r3]
        add     r0, r0, #1280
        ldr     r3, .emax_loc_lmrla_CURR_B2_resid;   str     r0, [r3]
        add     r0, r0, #1280
        ldr     r3, .emax_loc_lmrla_NEXT_B2_resid;   str     r0, [r3]
        ldr     r3, .emax_loc_lmrla_C_resid;         str     r1, [r3]
        add     r1, r1, #-4
        ldr     r3, .emax_loc_rgi_C_resid;           str     r1, [r3]
        ldr     r3, .emax_loc_lmwla_store_resid;     str     r2, [r3]
        add     r2, r2, #-4
        ldr     r3, .emax_loc_rgi_store_resid;       str     r2, [r3]
        ldr     r0, .emax_loc_start_resid
        ldr     r1, .emax_loc_end_resid
        bl      emax2_start
        mov     r0, #0
        add     sp, sp, #20
        ldmfd   sp!, {r4, r5, r6, r7, r8, r9, r10, fp, lr}
        bx      lr

.emax_loc_start_resid:          .word   .emax_start_resid
.emax_loc_rgi_p0____resid:      .word   .emax_rgi_p0____resid
.emax_loc_lmrla_PREV_B0_resid:  .word   .emax_lmrla_PREV_B0_resid
.emax_loc_lmrla_CURR_B0_resid:  .word   .emax_lmrla_CURR_B0_resid
.emax_loc_rgi_p1____resid:      .word   .emax_rgi_p1____resid
.emax_loc_lmrla_NEXT_B0_resid:  .word   .emax_lmrla_NEXT_B0_resid
.emax_loc_lmrla_PREV_B1_resid:  .word   .emax_lmrla_PREV_B1_resid
.emax_loc_lmrla_CURR_B1_resid:  .word   .emax_lmrla_CURR_B1_resid
.emax_loc_rgi_p2____resid:      .word   .emax_rgi_p2____resid
.emax_loc_lmrla_NEXT_B1_resid:  .word   .emax_lmrla_NEXT_B1_resid
.emax_loc_lmrla_PREV_B2_resid:  .word   .emax_lmrla_PREV_B2_resid
.emax_loc_lmrla_CURR_B2_resid:  .word   .emax_lmrla_CURR_B2_resid
.emax_loc_lmrla_NEXT_B2_resid:  .word   .emax_lmrla_NEXT_B2_resid
.emax_loc_rgi_C_resid:  .word   .emax_rgi_C_resid
.emax_loc_lmrla_C_resid:        .word   .emax_lmrla_C_resid
.emax_loc_rgi_store_resid:      .word   .emax_rgi_store_resid
.emax_loc_lmwla_store_resid:    .word   .emax_lmwla_store_resid
.emax_loc_end_resid:            .word   .emax_end_resid

        .data
        .p2align 4
//EMAX4A start .emax_start_resid:
//EMAX4A ctl map_dist=1
//EMAX4A @0,0  while (320)
//EMAX4A @0,1  add (ri+=,4),r0      rgi[.emax_rgi_p0____resid:,]  &
//EMAX4A @1,1                                                     & ld (r0,-1276),r3  lmr[.emax_lmrla_PREV_B0_resid:,0,320]
//EMAX4A @1,2                                                     & ld (r0,-1280),r2
//EMAX4A @1,3                                                     & ld (r0,-1284),r1
//EMAX4A @2,1  fmul (ri,r3),r29     rgi[0xbecccccd,]              & ld (r0,4),r6      lmr[.emax_lmrla_CURR_B0_resid:,0,320]
//EMAX4A @2,2  fmul (ri,r2),r30     rgi[0xbe99999a,]              & ld (r0,0),r5
//EMAX4A @2,3  fmul (ri,r1),r31     rgi[0xbecccccd,]              & ld (r0,-4),r4
//EMAX4A @3,0  add (ri+=,4),r0      rgi[.emax_rgi_p1____resid:,]  &
//EMAX4A @3,1  fma3 (ri,r6,r29),r29 rgi[0xbe99999a,]              & ld (r0,1284),r9   lmr[.emax_lmrla_NEXT_B0_resid:,0,320]
//EMAX4A @3,2  fma3 (ri,r5,r30),r30 rgi[0xbe4ccccd,]              & ld (r0,1280),r8
//EMAX4A @3,3  fma3 (ri,r4,r31),r31 rgi[0xbe99999a,]              & ld (r0,1276),r7
//EMAX4A @4,1  fma3 (ri,r9,r29),r29 rgi[0xbecccccd,]              & ld (r0,-1276),r12 lmr[.emax_lmrla_PREV_B1_resid:,0,320]
//EMAX4A @4,2  fma3 (ri,r8,r30),r30 rgi[0xbe99999a,]              & ld (r0,-1280),r11
//EMAX4A @4,3  fma3 (ri,r7,r31),r31 rgi[0xbecccccd,]              & ld (r0,-1284),r10
//EMAX4A @5,1  fma3 (ri,r12,r29),r29 rgi[0xbe99999a,]             & ld (r0,4),r15     lmr[.emax_lmrla_CURR_B1_resid:,0,320]
//EMAX4A @5,2  fma3 (ri,r11,r30),r30 rgi[0xbe4ccccd,]             & ld (r0,0),r14
//EMAX4A @5,3  fma3 (ri,r10,r31),r31 rgi[0xbe99999a,]             & ld (r0,-4),r13
//EMAX4A @6,0  add (ri+=,4),r0      rgi[.emax_rgi_p2____resid:,]  &
//EMAX4A @6,1  fma3 (ri,r15,r29),r29 rgi[0xbe4ccccd,]             & ld (r0,1284),r18  lmr[.emax_lmrla_NEXT_B1_resid:,0,320]
//EMAX4A @6,2  fma3 (ri,r14,r30),r30 rgi[0xbdcccccd,]             & ld (r0,1280),r17
//EMAX4A @6,3  fma3 (ri,r13,r31),r31 rgi[0xbe4ccccd,]             & ld (r0,1276),r16
//EMAX4A @7,1  fma3 (ri,r18,r29),r29 rgi[0xbe99999a,]             & ld (r0,-1276),r21 lmr[.emax_lmrla_PREV_B2_resid:,0,320]
//EMAX4A @7,2  fma3 (ri,r17,r30),r30 rgi[0xbe4ccccd,]             & ld (r0,-1280),r20
//EMAX4A @7,3  fma3 (ri,r16,r31),r31 rgi[0xbe99999a,]             & ld (r0,-1284),r19
//EMAX4A @8,1  fma3 (ri,r21,r29),r29 rgi[0xbecccccd,]             & ld (r0,4),r24     lmr[.emax_lmrla_CURR_B2_resid:,0,320]
//EMAX4A @8,2  fma3 (ri,r20,r30),r30 rgi[0xbe99999a,]             & ld (r0,0),r23
//EMAX4A @8,3  fma3 (ri,r19,r31),r31 rgi[0xbecccccd,]             & ld (r0,-4),r22
//EMAX4A @9,1  fma3 (ri,r24,r29),r29 rgi[0xbe99999a,]             & ld (r0,1284),r27  lmr[.emax_lmrla_NEXT_B2_resid:,0,320]
//EMAX4A @9,2  fma3 (ri,r23,r30),r30 rgi[0xbe4ccccd,]             & ld (r0,1280),r26
//EMAX4A @9,3  fma3 (ri,r22,r31),r31 rgi[0xbe99999a,]             & ld (r0,1276),r25
//EMAX4A @10,1 fma3 (ri,r27,r29),r29 rgi[0xbecccccd,]             & ld (ri+=,4),r28   rgi[.emax_rgi_C_resid:,] lmr[.emax_lmrla_C_resid:,0,320]
//EMAX4A @10,2 fma3 (ri,r26,r30),r30 rgi[0xbe99999a,]             &
//EMAX4A @10,3 fma3 (ri,r25,r31),r31 rgi[0xbecccccd,]             &
//EMAX4A @11,1 fadd (r29,r28),r28                                 &
//EMAX4A @11,2 fadd (r30,r31),r31                                 &
//EMAX4A @12,1 fadd (r31,r28),r28                                 &
//EMAX4A @13,3                                                    & st r28,(ri+=,4)   rgi[.emax_rgi_store_resid:,] lmw[.emax_lmwla_store_resid:,0,320]
//EMAX4A end .emax_end_resid:
```

1.6.5 Wave2d kernel

```
void wave2d( Z0, Z1, Z2 )
  float *Z0, *Z1, *Z2;
{
    int x;

    float value = 0.0025;

    for (x=0; x<WD; x++) {
      *(Z2+x) = 2.0 * *(Z1+x)
- *(Z0+x)
+ value * ( *(Z1+WD+x) + *(Z1-WD+x) + *(Z1-1+x) + *(Z1+1+x) - 4.0 * *(Z1+x) );
    }
}
```

```
        .text
        .align 2
        .global  wave2d
        .type   wave2d,%function
wave2d:
        /* void wave2d( Z0, Z1, Z2 ) float *Z0, *Z1, *Z2; */
        stmfd   sp!, {r4, r5, r6, r7, r8, r9, r10, fp, lr}
        sub     sp, sp, #20

        ldr     r3, .emax_loc_lmrla_Z0_wave2d
        str     r0, [r3]
        add     r0, r0, #-4
        ldr     r3, .emax_loc_rgi_Z0_wave2d
        str     r0, [r3]

        add     r1, r1, #-4
        ldr     r3, .emax_loc_rgi_p0____wave2d
        str     r1, [r3]
        add     r1, r1, #4
        add     r1, r1, #-1280
        ldr     r3, .emax_loc_lmrla_PREV_Z1_wave2d
        str     r1, [r3]
        add     r1, r1, #1280
        ldr     r3, .emax_loc_lmrla_CURR_Z1_wave2d
        str     r1, [r3]
        add     r1, r1, #1280
        ldr     r3, .emax_loc_lmrla_NEXT_Z1_wave2d
        str     r1, [r3]

        ldr     r3, .emax_loc_lmwla_store_wave2d
        str     r2, [r3]
        add     r2, r2, #-4
        ldr     r3, .emax_loc_rgi_store_wave2d
        str     r2, [r3]

        ldr     r0, .emax_loc_start_wave2d
        ldr     r1, .emax_loc_end_wave2d
        bl      emax2_start

        mov     r0, #0
        add     sp, sp, #20
        ldmfd   sp!, {r4, r5, r6, r7, r8, r9, r10, fp, lr}
        bx      lr

.emax_loc_start_wave2d:         .word   .emax_start_wave2d
.emax_loc_rgi_p0____wave2d:     .word   .emax_rgi_p0____wave2d
.emax_loc_rgi_Z0_wave2d:        .word   .emax_rgi_Z0_wave2d
.emax_loc_lmrla_Z0_wave2d:      .word   .emax_lmrla_Z0_wave2d
.emax_loc_lmrla_PREV_Z1_wave2d: .word   .emax_lmrla_PREV_Z1_wave2d
.emax_loc_lmrla_CURR_Z1_wave2d: .word   .emax_lmrla_CURR_Z1_wave2d
.emax_loc_lmrla_NEXT_Z1_wave2d: .word   .emax_lmrla_NEXT_Z1_wave2d
.emax_loc_rgi_store_wave2d:     .word   .emax_rgi_store_wave2d
.emax_loc_lmwla_store_wave2d:   .word   .emax_lmwla_store_wave2d
.emax_loc_end_wave2d:           .word   .emax_end_wave2d

        .data
        .p2align 4
//EMAX4A start .emax_start_wave2d:
//EMAX4A ctl map_dist=1
//EMAX4A @0,0  while (320)                                               &
//EMAX4A @0,1  add (ri+=,4),r0       rgi[.emax_rgi_p0____wave2d:,]       & ld (ri+=,4),r10   rgi[.emax_rgi_Z0_wave2d:,] lmr[.emax_lmrla_Z0_wave2d:,0,320]
//EMAX4A @1,1  fmul (ri,r10),r31     rgi[0xbf800000,]                    & ld (r0,-1280),r1  lmr[.emax_lmrla_PREV_Z1_wave2d:,0,320]
//EMAX4A @2,1  fma3 (ri,r1,r31),r31  rgi[0x3b23d70a,]                    & ld (r0,4),r4      lmr[.emax_lmrla_CURR_Z1_wave2d:,0,320]
//EMAX4A @2,2                                                            & ld (r0,0),r3
//EMAX4A @2,3                                                            & ld (r0,-4),r2
//EMAX4A @3,1  fma3 (ri,r4,r31),r31  rgi[0x3b23d70a,]                    & ld (r0,1280),r5   lmr[.emax_lmrla_NEXT_Z1_wave2d:,0,320]
//EMAX4A @3,2  fmul (ri,r3),r30      rgi[0x3b23d70a,]                    &
//EMAX4A @3,3  fmul (ri,r2),r29      rgi[0x3b23d70a,]                    &
//EMAX4A @4,1  fma3 (ri,r5,r31),r31  rgi[0x3b23d70a,]                    &
//EMAX4A @4,2  fmul (ri,r3),r29      rgi[0x40000000,]                    &
//EMAX4A @4,3  fma3 (ri,r30,r29),r30 rgi[0xc0800000,]                    &
//EMAX4A @5,1  fadd (r31,r29),r31                                        &
//EMAX4A @6,1  fadd (r31,r30),r31                                        &
//EMAX4A @7,1                                                            & st r31,(ri+=,4)   rgi[.emax_rgi_store_wave2d:,] lmw[.emax_lmwla_store_wave2d:,0,320]
//EMAX4A end .emax_end_wave2d:
```

1.6.6 Initialization in user program

```
In user program:
sysinit()
{
  Uint memsize;

#define EMAX2CTRLSPACE 0x00100000
  umem_open();
  udev_open();
  memsize = EMAX2CTRLSPACE+sizeof(struct GrA)+sizeof(struct B3D)+sizeof(struct C3D)
                          +sizeof(struct D3D)+sizeof(int)*BITMAP*4;
  membase_logical = (void*)umem_malloc(memsize);
  usrbase_logical = membase_logical+EMAX2CTRLSPACE;
  membase_phys = umem_get_phys(membase_logical); /* for real EMAX2 */
  usrbase_phys = umem_get_phys(usrbase_logical); /* for real EMAX2 */
  udev_write_4b(0, 1); /* reset EMAX2 */
}
```

1.6.7 Inside of emax2.c

```
In emax2.c:
#define UIO_MEMSPACE "/dev/uio0"
#define UMEMMAP_BASE 0x20000000
#define UMEMMAP_SIZE 0x20000000

#define UIO_DEVSPACE "/dev/uio1"
#define UMEMPAGESIZE 0x00001000
#define UDEVMAP_SIZE 0x00001000

void umem_open()
{
  int fd;

  if ((fd = open(UIO_MEMSPACE, O_RDWR)) < 1) {
    printf("umem_open(): Invalid UIO device: '%s'\n", UIO_MEMSPACE);
    exit(1);
  }
  umembase_logical = (volatile void*)mmap(NULL, UMEMMAP_SIZE, PROT_READ|PROT_WRITE, MAP_SHARED, fd, 0);
  umembase_phys    = (volatile void*)UMEMMAP_BASE;
  umem_allocated = 0;
}

void udev_open()
{
  int fd;

  if ((fd = open(UIO_DEVSPACE, O_RDWR)) < 1) {
    printf("udev_open: Invalid UIO device: '%s'\n", UIO_DEVSPACE);
    exit(1);
  }
  udevbase_logical = (volatile int*)mmap(NULL, UDEVMAP_SIZE, PROT_READ|PROT_WRITE, MAP_SHARED, fd, 0);
}
```

```
In emax2.c:
emax2_start(start, end) Uint start, end; /* start_of conf[][] - next_of last lmmi[][] */
{
  /* conf[][] 元先頭は start                          (stencil の場合,conf[][] 位置は必ずずれる conf 再利用しない) */
  /* regv[][] 元先頭は start+sizeob(conf)             (macropipe の場合,regv[][] 書き込み先は毎回変更)           */
  /* lmmi[][] 元先頭は start+sizeob(conf)+sizeof(regv)(macropipe の場合,lmmi[][] 書き込み先は毎回変更)           */

  struct conf_new {struct conf d[UNIT_DEPTH][UNIT_WIDTH];} *conf_new=(struct conf_new*)(start);
  struct regv_new {struct regv d[UNIT_DEPTH][UNIT_WIDTH];} *regv_new=(struct regv_new*)(start+sizeof(conf));
  struct lmmi_new {struct lmmi d[UNIT_DEPTH][UNIT_WIDTH];} *lmmi_new=(struct lmmi_new*)(start+sizeof(conf)+sizeof(regv));
  int i, j, k, nonzero_dist, M_STAT;

  if (last_start != start) {
    last_start = start;
    emax2_evenodd = 0;
    conf_addr = DDR3_CONFTOP;
    regv_addr = DDR3_REGVTOP;
    lmmi_addr = DDR3_LMMITOP;
  }
  else {
    emax2_evenodd = ~emax2_evenodd;
    if (!emax2_evenodd) {
      conf_addr = DDR3_CONFTOP;
      regv_addr = DDR3_REGVTOP;
      lmmi_addr = DDR3_LMMITOP;
    }
    else {
      conf_addr = DDR3_CONFTOP;
      regv_addr = DDR3_REGVTOP + DDR3_VARDIST;
      lmmi_addr = DDR3_LMMITOP + DDR3_VARDIST;
    }
  }

  memcpy((void*)conf_addr, conf_new, sizeof(conf));
  memcpy((void*)regv_addr, regv_new, sizeof(regv));
  for (i=0; i<UNIT_DEPTH; i++) {
    int old_i = (i + conf_new->d[0][0].dist)&(INSN_DEPTH-1);
    for (j=0; j<UNIT_WIDTH; j++) {
      ((struct lmmi_new*)lmmi_addr)->d[i][j].e2ctl                = lmmi_new->d[i][j].e2ctl;
      ((struct lmmi_new*)lmmi_addr)->d[i][j].ddr3_tlb.v           = lmmi_new->d[i][j].ddr3_tlb.v;
      ((struct lmmi_new*)lmmi_addr)->d[i][j].ddr3_tlb.msksft      = lmmi_new->d[i][j].ddr3_tlb.msksft;
      ((struct lmmi_new*)lmmi_addr)->d[i][j].ddr3_tlb.base_offset = emax2_evenodd;
      switch (lmmi_new->d[i][j].e2ctl.v) {
      case 1:
      case 3:
      case 5:
        if (lmmi_old[old_i][j].e2ctl.v    && lmmi_new->d[i][j].e2ctl.v < 4
         && lmmi_old[old_i][j].e2ctl.top  == lmmi_new->d[i][j].e2ctl.top
         && lmmi_old[old_i][j].e2ctl.len  >= lmmi_new->d[i][j].e2ctl.len
         && lmmi_old[old_i][j].e2ctl.dist == lmmi_new->d[i][j].e2ctl.dist)
          ((struct lmmi_new*)lmmi_addr)->d[i][j].ddr3_tlb.skip = 1;
        else {
          ((struct lmmi_new*)lmmi_addr)->d[i][j].ddr3_tlb.skip = 0;
          ((struct lmmi_new*)lmmi_addr)->d[i][j].ddr3_tlb.base = (lmmi_new->d[i][j].e2ctl.top)/DDR3_MINALIGN;
          /* EMAX2 転送前に, phys_addr を代入 */
        }
        break;
      case 2: /* LMW */
        ((struct lmmi_new*)lmmi_addr)->d[i][j].ddr3_tlb.base = (lmmi_new->d[i][j].e2ctl.top)/DDR3_MINALIGN;
        /* EMAX2 転送前に, phys_addr を代入 */
        break;
      }
    }
  }
  for (i=0; i<UNIT_DEPTH; i++) { /* for each unit be assigned */
    for (j=0; j<UNIT_WIDTH; j++) { /* for each unit to be assigned */
      lmmi_old[i][j].e2ctl.v    = ((struct lmmi_new*)lmmi_addr)->d[i][j].e2ctl.v;
      lmmi_old[i][j].e2ctl.top  = ((struct lmmi_new*)lmmi_addr)->d[i][j].e2ctl.top;
      lmmi_old[i][j].e2ctl.len  = ((struct lmmi_new*)lmmi_addr)->d[i][j].e2ctl.len;
      lmmi_old[i][j].e2ctl.dist = ((struct lmmi_new*)lmmi_addr)->d[i][j].e2ctl.dist;
    }
  }

  udev_write_4b(4, 1); /* start EMAX2 */
  do {
    udev_read_4b(8, &pe0_status);
  } while (pe0_status != STATUS_IDLE);
}
```

1.6.8 Compiling application programs

See “all:” tag in proj-arm32/sample/stencil-pipe/Makefile-zynq.emax2+asic.

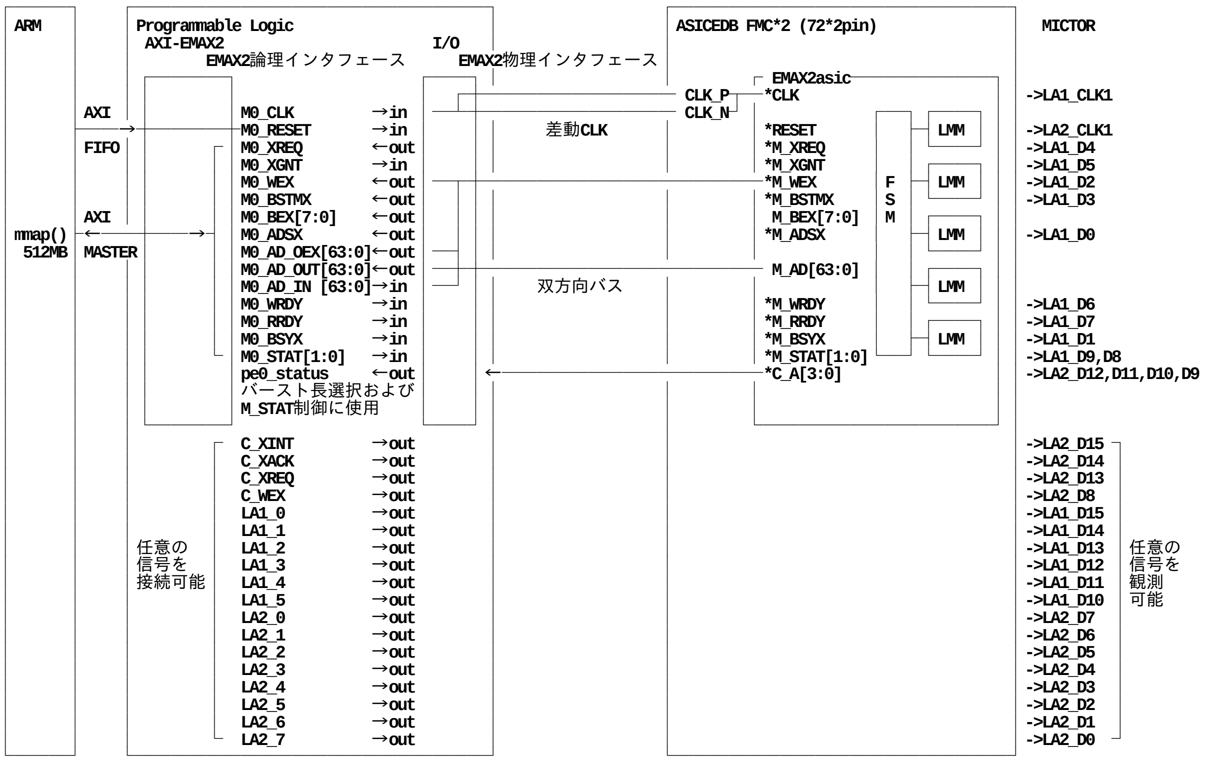
1.6.9 Executing application programs on simulator

See “run:” tag in proj-arm32/sample/stencil-pipe/Makefile-zynq.emax2+asic.

Chapter 2

EMAX2asic/ZYNQ Hardware

2.1 Structure of ZYNQ platform



※ M_AD[28:3] 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).

Figure.2.1: 全体構成

ZYNQ706 システムは，XC7Z045-FFG900-2，ASIC ソケットボード（ASICEDB），1GB-DDR3 を備え，内蔵 ARM コアと FPGA が密結合している．ARM コアおよび DDR3 主記憶上で LINUX が稼働しており，ARM-EMAX2 インタフェースを FPGA 上に，また，EMAX2 機能を ASICEDB 上に実装して，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 2.1 に全体構成を示す．論理インタフェース M0_CLK は 52.6MHz の供給を受け，FPGA 内の差動 CLK 変換器を経由して物理インタフェース CLK_P および CLK_M を駆動し，最終的に EMAX2asic の CLK ピンを駆動する．AXI_FIFO には，論理インタフェース M0_RESET が接続されており，ARM からのリセット指示に従い，一定期間（512 サイクル/52.6MHz 程度），M0_RESET を 1 とした後に 0 とすることにより，物理インタフェース RESET を通じて EMAX2asic 内部をリセットする．AXI MASTER には，M0_CLK および M0_RESET を除く論理インタフェース M0_*が接続される．ロジックアナライザには CLK（LA1_CLK1），RESET（LA2_CLK1），M_XREQ（LA1_D4）， M_XGNT（LA1_D5），M_WEX（LA1_D2），M_BSTMX（LA1_D3），M_ADSX（LA1_D0），M_WRDY（LA1_D6），M_RRDY （LA1_D7），M_BSYX（LA1_D1），M_STAT[1:0]（LA1_D9,D8）が接続されている．

EMAX2asic 内部状態 pe0_status[3:0] は C_A[3:0] に出力されており（FPGA の I/O から駆動してはならない）FPGA を経由せずに（LA2_D12,D11,D10,D9）にて観測できる．なお，バースト長情報を必要としない EMAX2asic インタフェースと，バースト長情報を必要とする AXI MASTER インタフェースを接続する際，C_A[3:0] に基づきバースト長を決定する．

後述の「EMAX2asic–AXI MASTER ブリッジ仕様」において

物理ピン C_XINT（AG26→LA2_D15)，C_XACK（AE27→LA2_D14)，C_XREQ（AG27→LA2_D13)，C_WEX（AK26 → LA2_D8）は EMAX2asic 内部には接続されていないため，同様に EMAX2asic に接続されていない LA1_0（R21 → LA1_D15），LA1_1（P21 → LA1_D14），LA1_2（N27 → LA1_D13），LA1_3（N26 → LA1_D12），LA1_4（P24 → LA1_D11），LA1_5（P23 → LA1_D10），LA2_0（AF12 → LA2_D7），LA2_1（AE12 → LA2_D6），LA2_2（AF13 → LA2_D5），LA2_3（AE13 → LA2_D4），LA2_4（AG16 → LA2_D3），LA2_5（AG17 → LA2_D2），LA2_6（AD28 → LA2_D1），LA2_7（AC28 → LA2_D0）と併せて FPGA 内部の信号を最大 18 まで観測できる．

2.2 ブロック間インタフェース

2.2.1 LMM

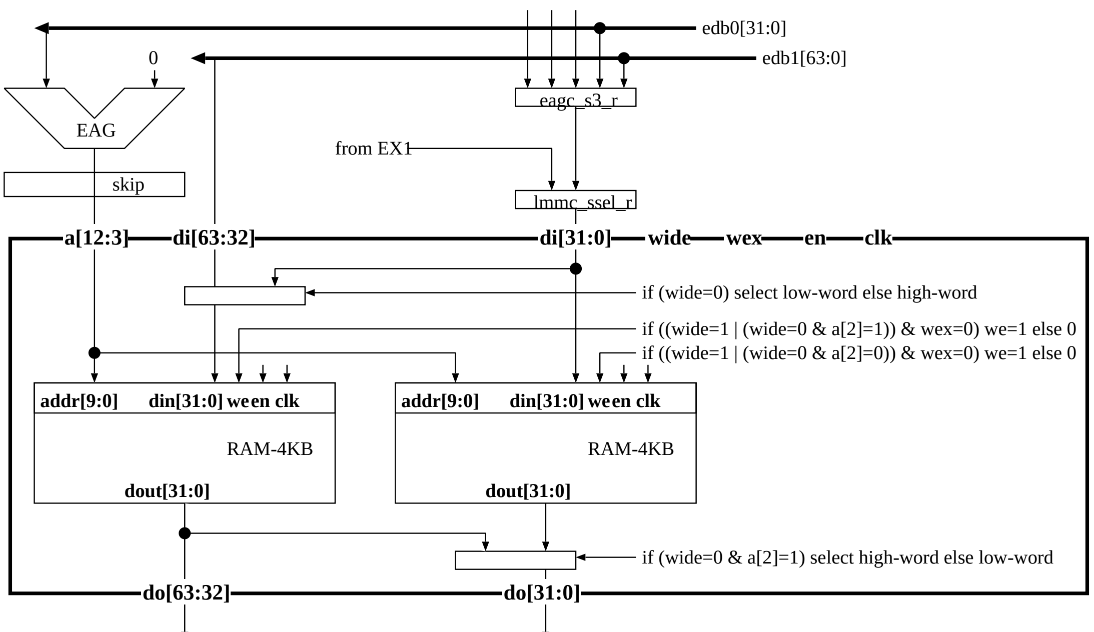


Figure.2.2: Structure of LMM.

図 2.2 と図 2.3 に，EMAX2 内 LMM の構造およびタイミングチャートを示す．

2.2.2 EMAX2asic インタフェース

図 2.4 に示すように，EMAX2asic 物理ピンは，メモリバス信号（82 本），および，ステータス信号（pe0_status が接続された C_A[3:0] の 4 本）の合計 86 本を収容する．さらに，CLK，RESET の 2 本を加えた合計 88 本が EMAX2asic に接続される．赤字により示された信号は EMAX2asic では使用しない．図 2.1 に示すように，MICTOR コネクタには 88 本のうち 16 本が接続され，MICTOR コネクタの残り 18 本には，FPGA のピンが 1 対 1 に接続される．

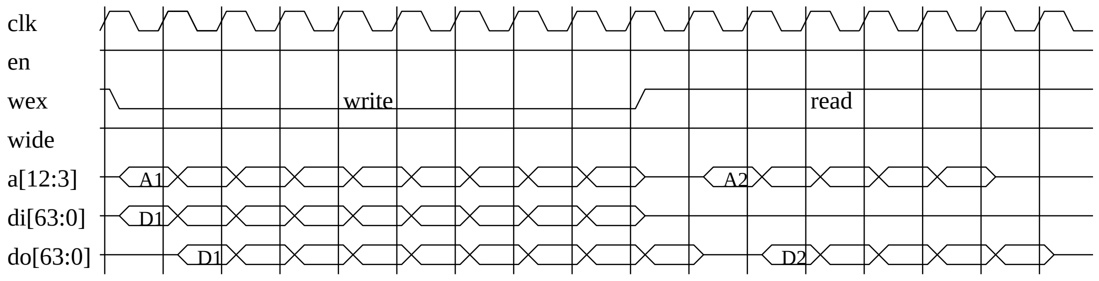


Figure.2.3: Timing chart of LMM.

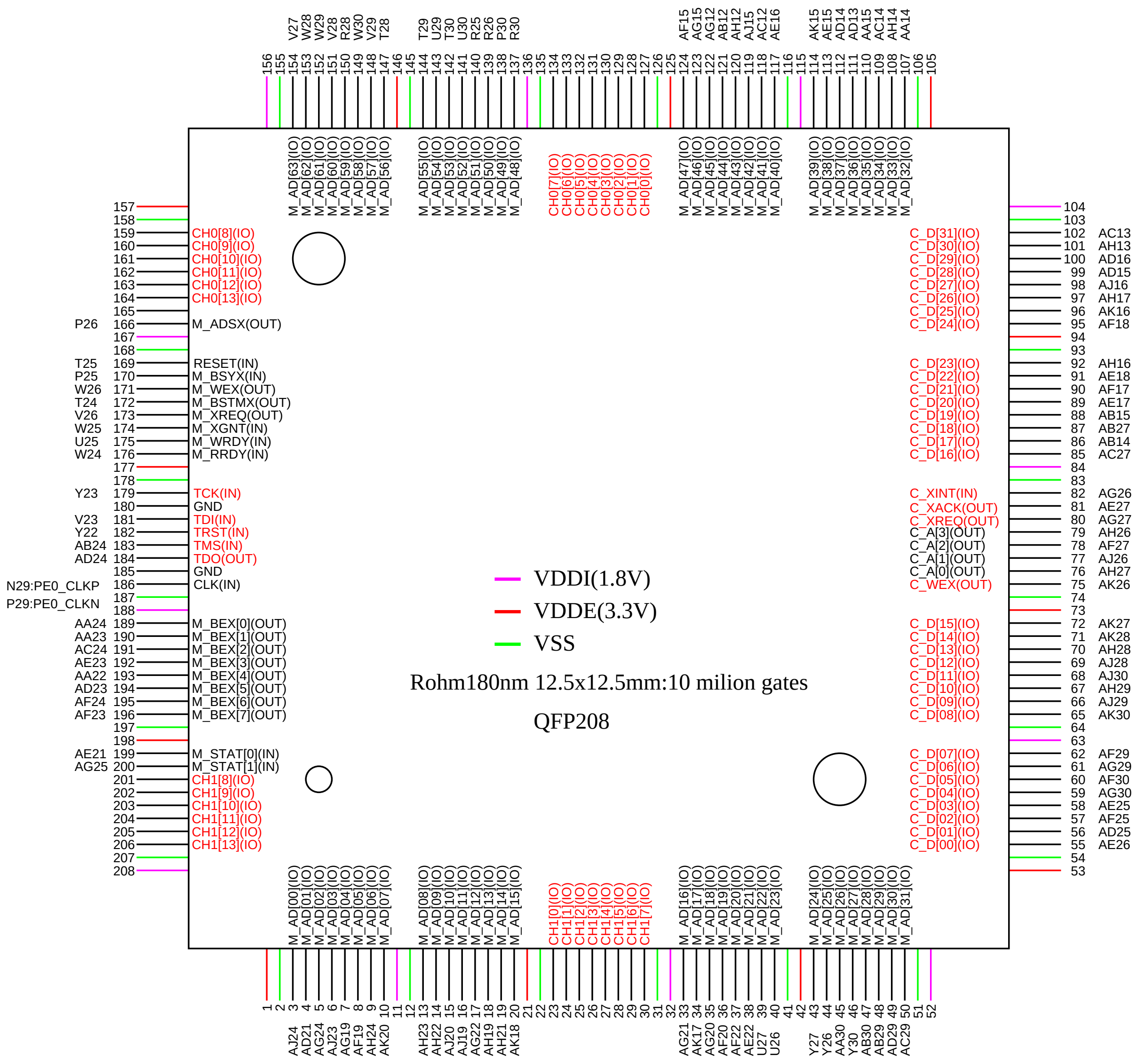


Figure.2.4: QFP208 のピン配置（周辺の記号は ZYNQ の対応ピン番号）

ASIC（ROHM/TSMC：QFP208）のピン配置制約は次の通りである．

- 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）
- CLK，RESET 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．ASIC の I/O には，駆動力 2mA のドライバを推奨（ROHM0.18 の場合は以下）．
  - PC30B01 入出力用．出力用 I ピンおよび OEN の容量は各 22fF（INV × 3 に相当）．OEN × 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 × 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．

図 2.6 から図 2.7 に EMAX2asic インタフェースのタイミングチャートを示す．

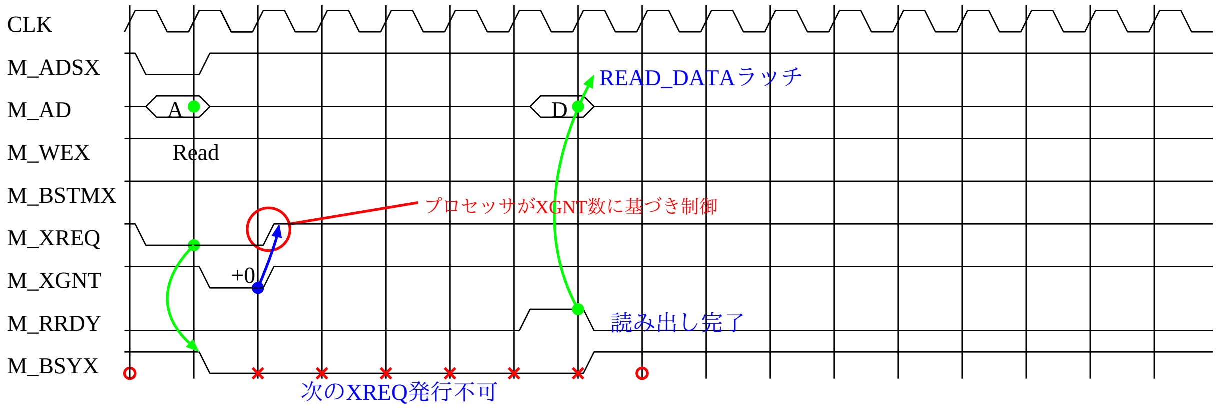


Figure.2.5: EMAX2asic インタフェースのタイミングチャート (READ)

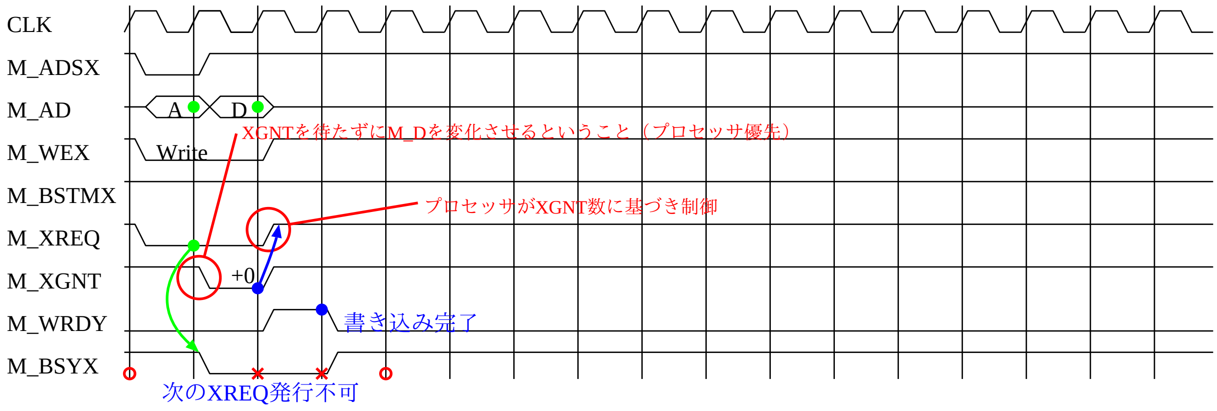


Figure.2.6: EMAX2asic インタフェースのタイミングチャート (WRITE)

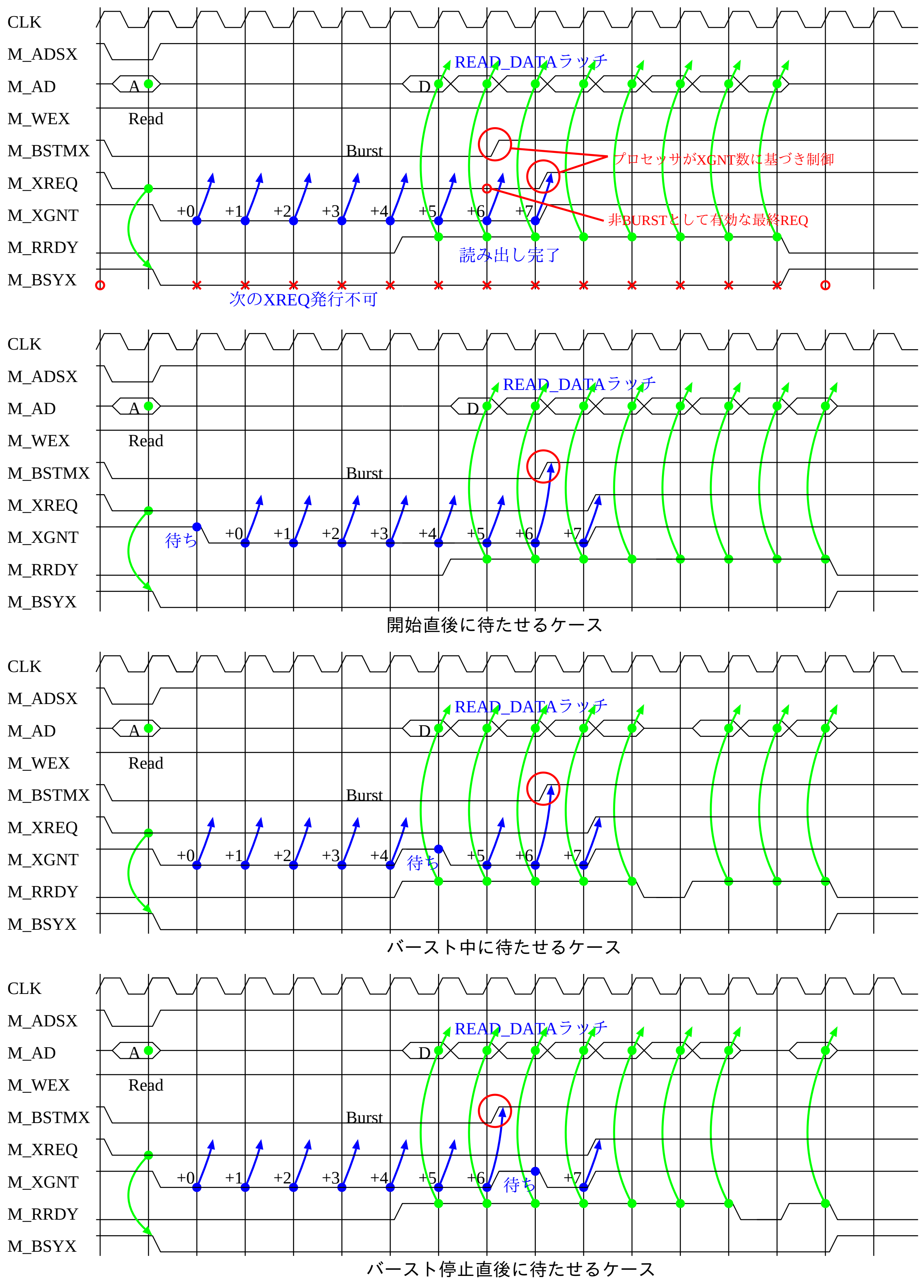


Figure.2.7: EMAX2asic インタフェースのタイミングチャート (READ バースト)

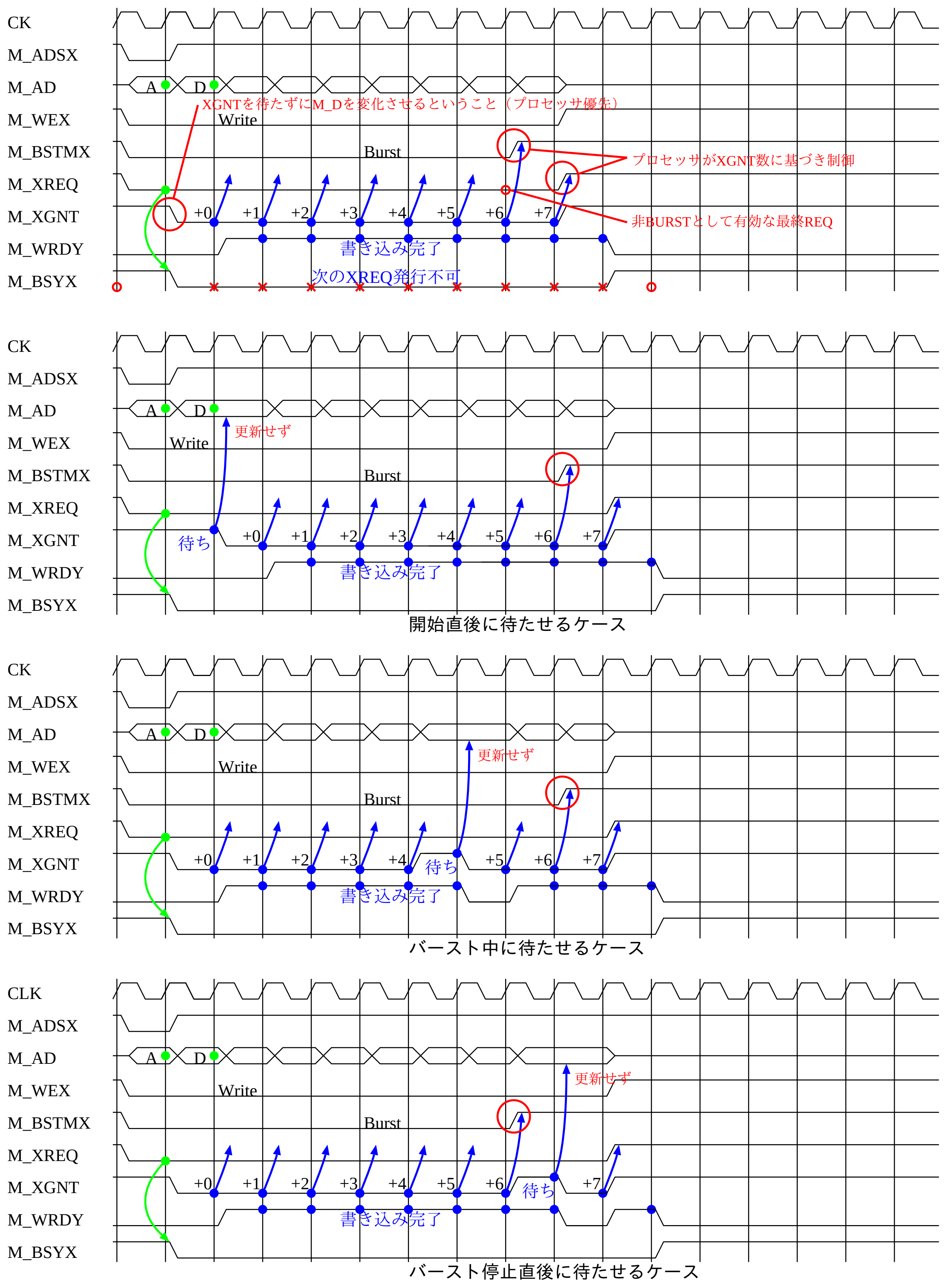

CK
M_ADSX
M_AD
M_WEX
M_BSTMX
M_XREQ
M_XGNT
M_WRDY
M_BSYX
A
D
XGNTを待たずにM_Dを変化させるということ（プロセッサ優先）
Write
Burst
プロセッサがXGNT数に基づき制御
非BURSTとして有効な最終REQ
+0
+1
+2
+3
+4
+5
+6
+7
書き込み完了
次のXREQ発行不可
更新せず
待ち
開始直後に待たせるケース
バースト中に待たせるケース
CLK
バースト停止直後に待たせるケース

Figure.2.8: EMAX2asic インタフェースのタイミングチャート (WRITE バースト)

2.2.3 EMAX2 論理インタフェース

Table.2.1: EMAX2 論理インタフェース

信号名	方向	説明
M0_XREQ	out	主記憶参照要求信号，Active-LOW
M0_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
M0_WEX	out	書き込みイネーブル，Active-LOW
M0_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
M0_BEX[7:0]	out	M0_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
M0_ADSX	out	0 の場合 M0_AD_OUT はアドレス，1 の場合データ
M0_AD_OEX[63:0]	out	M0_AD_OUT の各 bit に対応する出力イネーブル, Active-LOW．全 bit に (M0_ADSX & M0_WEX) が接続される
M0_AD_OUT[63:0]	out	M0_ADSX が 0 の場合アドレス (bit27-3 のみ有効)，1 の場合データ
M0_AD_IN[63:0]	in	入力データ
M0_WRDY	in	書き込み完了信号，Active-HIGH．本信号のカウントにより EMAX2 は書き込み完了を知る
M0_RRDY	in	読み出し完了信号，Active-HIGH．本信号により EMAX2 は M0_AD_IN の有効値を取り込む
M0_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
M0_STAT[1:0]	in	状態表示信号，STAT1-0（0:empty 1:EMAX2end 3:EMAX2start）は，特定アドレスへの書き込み値を反映する

表 2.1 に EMAX2 論理インタフェースのタイミングチャートを示す.

2.2.4 AXI fifo インタフェース

```
----------------------------------------------------------------------------------------------------------------------------------------------------
module axi_master_fifo #
  (parameter integer C_M_AXI_THREAD_ID_WIDTH      = 1,
   parameter integer C_M_AXI_ADDR_WIDTH           = 32,
   parameter integer C_M_AXI_DATA_WIDTH           = 32,
   parameter integer C_M_AXI_AWUSER_WIDTH         = 1,
   parameter integer C_M_AXI_ARUSER_WIDTH         = 1,
   parameter integer C_M_AXI_WUSER_WIDTH          = 1,
   parameter integer C_M_AXI_RUSER_WIDTH          = 1,
   parameter integer C_M_AXI_BUSER_WIDTH          = 1,
   parameter integer C_M_AXI_SUPPORTS_WRITE       = 1,
   parameter integer C_M_AXI_SUPPORTS_READ        = 1,
   parameter integer FIFO_ADDR_WIDTH              = 4,
   parameter         C_M_AXI_TARGET               = 'h00000000)
'define AXIF_C_LOG_2(n) ((n)<=(1<< 0)? 0:(n)<=(1<< 1)? 1:(n)<=(1<< 2)? 2:(n)<=(1<< 3)? 3: (n)<=(1<< 4)? 4:(n)<=(1<< 5)? 5:(n)<=(1<< 6)? 6:(n)<=(1<< 7)? 7:
                        (n)<=(1<< 8)? 8:(n)<=(1<< 9)? 9:(n)<=(1<<10)?10:(n)<=(1<<11)?11: (n)<=(1<<12)?12:(n)<=(1<<13)?13:(n)<=(1<<14)?14:(n)<=(1<<15)?15:
                        (n)<=(1<<16)?16:(n)<=(1<<17)?17:(n)<=(1<<18)?18:(n)<=(1<<19)?19: (n)<=(1<<20)?20:(n)<=(1<<21)?21:(n)<=(1<<22)?22:(n)<=(1<<23)?23:
                        (n)<=(1<<24)?24:(n)<=(1<<25)?25:(n)<=(1<<26)?26:(n)<=(1<<27)?27: (n)<=(1<<28)?28:(n)<=(1<<29)?29:(n)<=(1<<30)?30:(n)<=(1<<31)?31:32)
  localparam integer C_M_AXI_BURST_COUNT_WIDTH = C_M_AXI_ADDR_WIDTH + 1;
  localparam integer ADDRMASK_WIDTH = 'AXIF_C_LOG_2(C_M_AXI_DATA_WIDTH / 8);
  (
   input wire ACLK,
   input wire ARESETN,
   //----------------------------------------------------------------------
   // User Interface ★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★
   //----------------------------------------------------------------------
   // Data Channel
   input                          user_write_enq,
   input [C_M_AXI_DATA_WIDTH-1:0]   user_write_data,
   output                         user_write_almost_full,
   input                          user_read_deq,
   output [C_M_AXI_DATA_WIDTH-1:0]  user_read_data,
   output                         user_read_empty,
   // Command Channel
   input [C_M_AXI_ADDR_WIDTH-1:0]   user_addr,
   input                          user_read_enable,
   input                          user_write_enable,
   input [8:0]                    user_word_size,
   output reg                     user_done,
   output wire                    ERROR,
   //----------------------------------------------------------------------
   // AXI Master Interface
   //----------------------------------------------------------------------
   // Master Interface Write Address
   output wire [C_M_AXI_THREAD_ID_WIDTH-1:0] M_AXI_AWID,      // assign M_AXI_AWID = 'b0;
   output wire [C_M_AXI_ADDR_WIDTH-1:0]      M_AXI_AWADDR,    // assign M_AXI_AWADDR = C_M_AXI_TARGET + awaddr_offset;  ★internal_reg★
   output wire [8-1:0]                       M_AXI_AWLEN,     // assign M_AXI_AWLEN = user_word_size_buf_m1;            ★internal_reg★
   output wire [3-1:0]                       M_AXI_AWSIZE,    // assign M_AXI_AWSIZE = 'AXIF_C_LOG_2(C_M_AXI_DATA_WIDTH/8);
   output wire [2-1:0]                       M_AXI_AWBURST,   // assign M_AXI_AWBURST = 2'b01;
   output wire                               M_AXI_AWLOCK,    // assign M_AXI_AWLOCK = 1'b0;
   output wire [4-1:0]                       M_AXI_AWCACHE,   // assign M_AXI_AWCACHE = 4'b0011;
   output wire [3-1:0]                       M_AXI_AWPROT,    // assign M_AXI_AWPROT = 3'h0;
   output wire [4-1:0]                       M_AXI_AWQOS,     // assign M_AXI_AWQOS = 4'h0;
   output wire [C_M_AXI_AWUSER_WIDTH-1:0]    M_AXI_AWUSER,    // assign M_AXI_AWUSER = 'b0;
   output wire                               M_AXI_AWVALID,   // assign M_AXI_AWVALID = awvalid;                        ★internal_reg★
   input  wire                               M_AXI_AWREADY,   //
   // Master Interface Write Data
   output wire [C_M_AXI_DATA_WIDTH-1:0]      M_AXI_WDATA,     // assign M_AXI_WDATA  = wdata;                           ★internal_reg★
   output wire [C_M_AXI_DATA_WIDTH/8-1:0]    M_AXI_WSTRB,     // assign M_AXI_WSTRB  = {(C_M_AXI_DATA_WIDTH/8){1'b1}};
   output wire                               M_AXI_WLAST,     // assign M_AXI_WLAST  = wlast;                           ★internal_reg★
   output wire [C_M_AXI_WUSER_WIDTH-1:0]     M_AXI_WUSER,     // assign M_AXI_WUSER  = 'b0;
   output wire                               M_AXI_WVALID,    // assign M_AXI_WVALID = wvalid;                          ★internal_reg★
   input  wire                               M_AXI_WREADY,    //
   // Master Interface Write Response
   input  wire [C_M_AXI_THREAD_ID_WIDTH-1:0] M_AXI_BID,
   input  wire [2-1:0]                       M_AXI_BRESP,
   input  wire [C_M_AXI_BUSER_WIDTH-1:0]     M_AXI_BUSER,
   input  wire                               M_AXI_BVALID,
   output wire                               M_AXI_BREADY,    // assign M_AXI_BREADY  = C_M_AXI_SUPPORTS_WRITE;
   // Master Interface Read Address
   output wire [C_M_AXI_THREAD_ID_WIDTH-1:0] M_AXI_ARID,      // assign M_AXI_ARID    = 'b0;
   output wire [C_M_AXI_ADDR_WIDTH-1:0]      M_AXI_ARADDR,    // assign M_AXI_ARADDR  = C_M_AXI_TARGET + araddr_offset; ★internal_reg★
   output wire [8-1:0]                       M_AXI_ARLEN,     // assign M_AXI_ARLEN   = user_word_size_buf_m1;          ★internal_reg★
   output wire [3-1:0]                       M_AXI_ARSIZE,    // assign M_AXI_ARSIZE  = 'AXIF_C_LOG_2(C_M_AXI_DATA_WIDTH/8);
   output wire [2-1:0]                       M_AXI_ARBURST,   // assign M_AXI_ARBURST = 2'b01;
   output wire [2-1:0]                       M_AXI_ARLOCK,    // assign M_AXI_ARLOCK  = 1'b0;
   output wire [4-1:0]                       M_AXI_ARCACHE,   // assign M_AXI_ARCACHE = 4'b0011;
   output wire [3-1:0]                       M_AXI_ARPROT,    // assign M_AXI_ARPROT  = 3'h0;
   output wire [4-1:0]                       M_AXI_ARQOS,     // assign M_AXI_ARQOS   = 4'h0;
   output wire [C_M_AXI_ARUSER_WIDTH-1:0]    M_AXI_ARUSER,    // assign M_AXI_ARUSER  = 'b0;
   output wire                               M_AXI_ARVALID,   // assign M_AXI_ARVALID = arvalid;                        ★internal_reg★
   input  wire                               M_AXI_ARREADY,
   // Master Interface Read Data
   input  wire [C_M_AXI_THREAD_ID_WIDTH-1:0] M_AXI_RID,
   input  wire [C_M_AXI_DATA_WIDTH-1:0]      M_AXI_RDATA,
   input  wire [2-1:0]                       M_AXI_RRESP,
   input  wire                               M_AXI_RLAST,
   input  wire [C_M_AXI_RUSER_WIDTH-1:0]     M_AXI_RUSER,
   input  wire                               M_AXI_RVALID,
   output wire                               M_AXI_RREADY     // assign M_AXI_RREADY = !axi_read_almost_full;           ★internal_reg★
   );
  //------------------------------------------------------------------------
  // Data Channel (Read FIFO / Write FIFO)
  //------------------------------------------------------------------------
  wire                          axi_write_deq;
  wire [C_M_AXI_DATA_WIDTH-1:0] axi_write_data;
  wire                          axi_write_empty;
  reg                           axi_read_enq;
  reg  [C_M_AXI_DATA_WIDTH-1:0] axi_read_data;
  wire                          axi_read_almost_full;
  // Write
  axi_data_fifo
  #(.DATA_WIDTH(C_M_AXI_DATA_WIDTH),
    .ADDR_WIDTH(FIFO_ADDR_WIDTH))
  inst_write_fifo
  (.ACLK(ACLK), .ARESETN(ARESETN),
   .enq(user_write_enq), .data_in(user_write_data), .almost_full(user_write_almost_full),
   .deq(axi_write_deq), .data_out(axi_write_data), .empty(axi_write_empty));
  // Read
  axi_data_fifo
  #(.DATA_WIDTH(C_M_AXI_DATA_WIDTH),
    .ADDR_WIDTH(FIFO_ADDR_WIDTH))
  inst_read_fifo
  (.ACLK(ACLK), .ARESETN(ARESETN),
   .enq(axi_read_enq), .data_in(axi_read_data), .almost_full(axi_read_almost_full),
   .deq(user_read_deq), .data_out(user_read_data), .empty(user_read_empty));
endmodule
```

Figure.2.9: AXI FIFO インタフェース (1/2)

```
-------------------------------------------------------------------------------------------------
module axi_data_fifo #
  (
   parameter integer DATA_WIDTH = 32,
   parameter integer ADDR_WIDTH = 4,
   parameter integer ALMOST_FULL_THRESHOLD = 3,
   parameter integer ALMOST_EMPTY_THRESHOLD = 1)
  (
   input                   ACLK,
   input                   ARESETN,
   input [DATA_WIDTH-1:0]  data_in,
   input                   enq,
   output reg              full,
   output reg              almost_full,
   output [DATA_WIDTH-1:0] data_out,
   input                   deq,
   output reg              empty,
   output reg              almost_empty
   );
  axi_data_fifo_ram
  #(
    .DATA_WIDTH(DATA_WIDTH),
    .ADDR_WIDTH(ADDR_WIDTH)
    )
  inst_ram
  (
   .ACLK(ACLK),
   .addr0(ram_addr0), .data_in0(ram_data_in0), .write_enable0(ram_we0),
   .data_out0(ram_data_out0),
   .addr1(ram_addr1), .data_in1('h0), .write_enable1(1'b0),
   .data_out1(ram_data_out1)
   );
endmodule

module axi_data_fifo_ram #
  (
   parameter integer DATA_WIDTH = 32,
   parameter integer ADDR_WIDTH = 4)
  (
   input                   ACLK,
   input  [ADDR_WIDTH-1:0] addr0,
   input  [DATA_WIDTH-1:0] data_in0,
   input                   write_enable0,
   output [DATA_WIDTH-1:0] data_out0,
   input  [ADDR_WIDTH-1:0] addr1,
   input  [DATA_WIDTH-1:0] data_in1,
   input                   write_enable1,
   output [DATA_WIDTH-1:0] data_out1
   );
endmodule
```

Figure.2.10: AXI FIFO インタフェース (2/2)

2.2.5 AXI master インタフェース

```
-----------------------------------------------------------------------------------------------------------------------------------
module axi_master_interface #
  (parameter integer C_M_AXI_THREAD_ID_WIDTH      = 1,
   parameter integer C_M_AXI_ADDR_WIDTH           = 32,
   parameter integer C_M_AXI_DATA_WIDTH           = 32,
   parameter integer C_M_AXI_AWUSER_WIDTH         = 1,
   parameter integer C_M_AXI_ARUSER_WIDTH         = 1,
   parameter integer C_M_AXI_WUSER_WIDTH          = 1,
   parameter integer C_M_AXI_RUSER_WIDTH          = 1,
   parameter integer C_M_AXI_BUSER_WIDTH          = 1,
   parameter integer C_M_AXI_SUPPORTS_WRITE       = 1,
   parameter integer C_M_AXI_SUPPORTS_READ        = 1,
   parameter         C_M_AXI_TARGET               = 'h00000000)
  localparam BURST_FIXED = 2'b00;
  localparam BURST_INCR  = 2'b01;
  localparam BURST_WRAP  = 2'b10;
‘define AXIF_C_LOG_2(n) ((n)<=(1<< 0)? 0:(n)<=(1<< 1)? 1:(n)<=(1<< 2)? 2:(n)<=(1<< 3)? 3:(n)<=(1<< 4)? 4:(n)<=(1<< 5)? 5:(n)<=(1<< 6)? 6:(n)<=(1<< 7)? 7:
                         (n)<=(1<< 8)? 8:(n)<=(1<< 9)? 9:(n)<=(1<<10)?10:(n)<=(1<<11)?11:(n)<=(1<<12)?12:(n)<=(1<<13)?13:(n)<=(1<<14)?14:(n)<=(1<<15)?15:
                         (n)<=(1<<16)?16:(n)<=(1<<17)?17:(n)<=(1<<18)?18:(n)<=(1<<19)?19:(n)<=(1<<20)?20:(n)<=(1<<21)?21:(n)<=(1<<22)?22:(n)<=(1<<23)?23:
                         (n)<=(1<<24)?24:(n)<=(1<<25)?25:(n)<=(1<<26)?26:(n)<=(1<<27)?27:(n)<=(1<<28)?28:(n)<=(1<<29)?29:(n)<=(1<<30)?30:(n)<=(1<<31)?31:32)
  localparam integer C_M_AXI_ADDRMASK_WIDTH = ‘AXII_C_LOG_2(C_M_AXI_DATA_WIDTH / 8);
  (
   input wire ACLK,
   input wire ARESETN,
   //------------------------------------------------------------------------
   // User Bus Interface ★★★★★★★★★★★★★★★★★★★★★★★★★★★★
   //------------------------------------------------------------------------
   // Write Address
   input wire                          awvalid,    // awaddr有効（先行awaddrがawlen分使われた後に使われる）
   input wire  [C_M_AXI_ADDR_WIDTH-1:0] awaddr,     // awaddr
   input wire  [8-1:0]                  awlen,      // バースト長(len+1)
   output wire                          awready,    // awready=0ならawvalidとawaddrをHOLD
   // Write Data
   input wire                           wvalid,     // wdata有効
   input wire  [C_M_AXI_DATA_WIDTH-1:0] wdata,      // wdata
   input wire                           wlast,      // wreadyが常時1のslaveの場合,burst最後に1としなければならない
   output wire                          wready,     // wready=0ならwvalidとwdataをHOLD
   // Write Response
   output wire                          bvalid,     // awvalidに対するトランザクション完了
   // Read Address
   input wire                           arvalid,    // araddr有効（先行araddrがarlen分使われた後に使われる）
   input wire  [C_M_AXI_ADDR_WIDTH-1:0] araddr,     // araddr
   input wire  [8-1:0]                  arlen,      // バースト長(len+1)
   output wire                          arready,    // arready=0ならarvalidとaraddrをHOLD
   // Read Data
   output wire                          rvalid,     // rdata有効
   output wire [C_M_AXI_DATA_WIDTH-1:0] rdata,      // rdata
   output wire                          rlast,      // burst最後に1
   input wire                           rready,     // rready=0にするとrvalidとrdataをHOLDしてくれる

   //------------------------------------------------------------------------
   // AXI Master Interface
   //------------------------------------------------------------------------
   // Master Interface Write Address
   output wire [C_M_AXI_THREAD_ID_WIDTH-1:0] M_AXI_AWID,        // assign M_AXI_AWID    = 'b0;
   output wire [C_M_AXI_ADDR_WIDTH-1:0]      M_AXI_AWADDR,      // assign M_AXI_AWADDR  = C_M_AXI_TARGET + awaddr; ★in★
   output wire [8-1:0]                       M_AXI_AWLEN,       // assign M_AXI_AWLEN   = awlen;                   ★in★
   output wire [3-1:0]                       M_AXI_AWSIZE,      // assign M_AXI_AWSIZE  = C_M_AXI_ADDRMASK_WIDTH;
   output wire [2-1:0]                       M_AXI_AWBURST,     // assign M_AXI_AWBURST = BURST_INCR;
   output wire                               M_AXI_AWLOCK,      // assign M_AXI_AWLOCK  = 1'b0;
   output wire [4-1:0]                       M_AXI_AWCACHE,     // assign M_AXI_AWCACHE = 4'b0011;
   output wire [3-1:0]                       M_AXI_AWPROT,      // assign M_AXI_AWPROT  = 3'h0;
   output wire [4-1:0]                       M_AXI_AWQOS,       // assign M_AXI_AWQOS   = 4'h0;
   output wire [C_M_AXI_AWUSER_WIDTH-1:0]    M_AXI_AWUSER,      // assign M_AXI_AWUSER  = 'b0;
   output wire                               M_AXI_AWVALID,     // assign M_AXI_AWVALID = awvalid;                 ★in★
   input  wire                               M_AXI_AWREADY,     // assign awready       = M_AXI_AWREADY;           ★out★
   // Master Interface Write Data
   output wire [C_M_AXI_DATA_WIDTH-1:0]      M_AXI_WDATA,       // assign M_AXI_WDATA   = wdata;                   ★in★
   output wire [C_M_AXI_DATA_WIDTH/8-1:0]    M_AXI_WSTRB,       // assign M_AXI_WSTRB   = {(C_M_AXI_DATA_WIDTH/8){1'b1}};
   output wire                               M_AXI_WLAST,       // assign M_AXI_WLAST   = wlast;                   ★in★
   output wire [C_M_AXI_WUSER_WIDTH-1:0]     M_AXI_WUSER,       // assign M_AXI_WUSER   = 'b0;
   output wire                               M_AXI_WVALID,      // assign M_AXI_WVALID  = wvalid;                  ★in★
   input  wire                               M_AXI_WREADY,      // assign wready        = M_AXI_WREADY;            ★out★
   // Master Interface Write Response
   input  wire [C_M_AXI_THREAD_ID_WIDTH-1:0] M_AXI_BID,
   input  wire [2-1:0]                       M_AXI_BRESP,
   input  wire [C_M_AXI_BUSER_WIDTH-1:0]     M_AXI_BUSER,
   input  wire                               M_AXI_BVALID,      // assign bvalid        = M_AXI_BVALID;            ★out★
   output wire                               M_AXI_BREADY,      // assign M_AXI_BREADY  = C_M_AXI_SUPPORTS_WRITE;
   // Master Interface Read Address
   output wire [C_M_AXI_THREAD_ID_WIDTH-1:0] M_AXI_ARID,        // assign M_AXI_ARID    = 'b0;
   output wire [C_M_AXI_ADDR_WIDTH-1:0]      M_AXI_ARADDR,      // assign M_AXI_ARADDR  = C_M_AXI_TARGET + araddr; ★in★
   output wire [8-1:0]                       M_AXI_ARLEN,       // assign M_AXI_ARLEN   = arlen;                   ★in★
   output wire [3-1:0]                       M_AXI_ARSIZE,      // assign M_AXI_ARSIZE  = C_M_AXI_ADDRMASK_WIDTH;
   output wire [2-1:0]                       M_AXI_ARBURST,     // assign M_AXI_ARBURST = BURST_INCR;
   output wire [2-1:0]                       M_AXI_ARLOCK,      // assign M_AXI_ARLOCK  = 1'b0;
   output wire [4-1:0]                       M_AXI_ARCACHE,     // assign M_AXI_ARCACHE = 4'b0011;
   output wire [3-1:0]                       M_AXI_ARPROT,      // assign M_AXI_ARPROT  = 3'h0;
   output wire [4-1:0]                       M_AXI_ARQOS,       // assign M_AXI_ARQOS   = 4'h0;
   output wire [C_M_AXI_ARUSER_WIDTH-1:0]    M_AXI_ARUSER,      // assign M_AXI_ARUSER  = 'b0;
   output wire                               M_AXI_ARVALID,     // assign M_AXI_ARVALID = arvalid;                 ★in★
   input  wire                               M_AXI_ARREADY,     // assign arready       = M_AXI_ARREADY;           ★out★
   // Master Interface Read Data
   input  wire [C_M_AXI_THREAD_ID_WIDTH-1:0] M_AXI_RID,
   input  wire [C_M_AXI_DATA_WIDTH-1:0]      M_AXI_RDATA,       // assign rdata         = M_AXI_RDATA;             ★out★
   input  wire [2-1:0]                       M_AXI_RRESP,
   input  wire                               M_AXI_RLAST,       // assign rlast         = M_AXI_RLAST;             ★out★
   input  wire [C_M_AXI_RUSER_WIDTH-1:0]     M_AXI_RUSER,
   input  wire                               M_AXI_RVALID,      // assign rvalid        = M_AXI_RVALID;            ★out★
   output wire                               M_AXI_RREADY       // assign M_AXI_RREADY  = rready;                  ★in★
   );
endmodule
```

Figure.2.11: AXI MASTER インタフェース

2.2.6 EMAX2asic–AXI master ブリッジ仕様

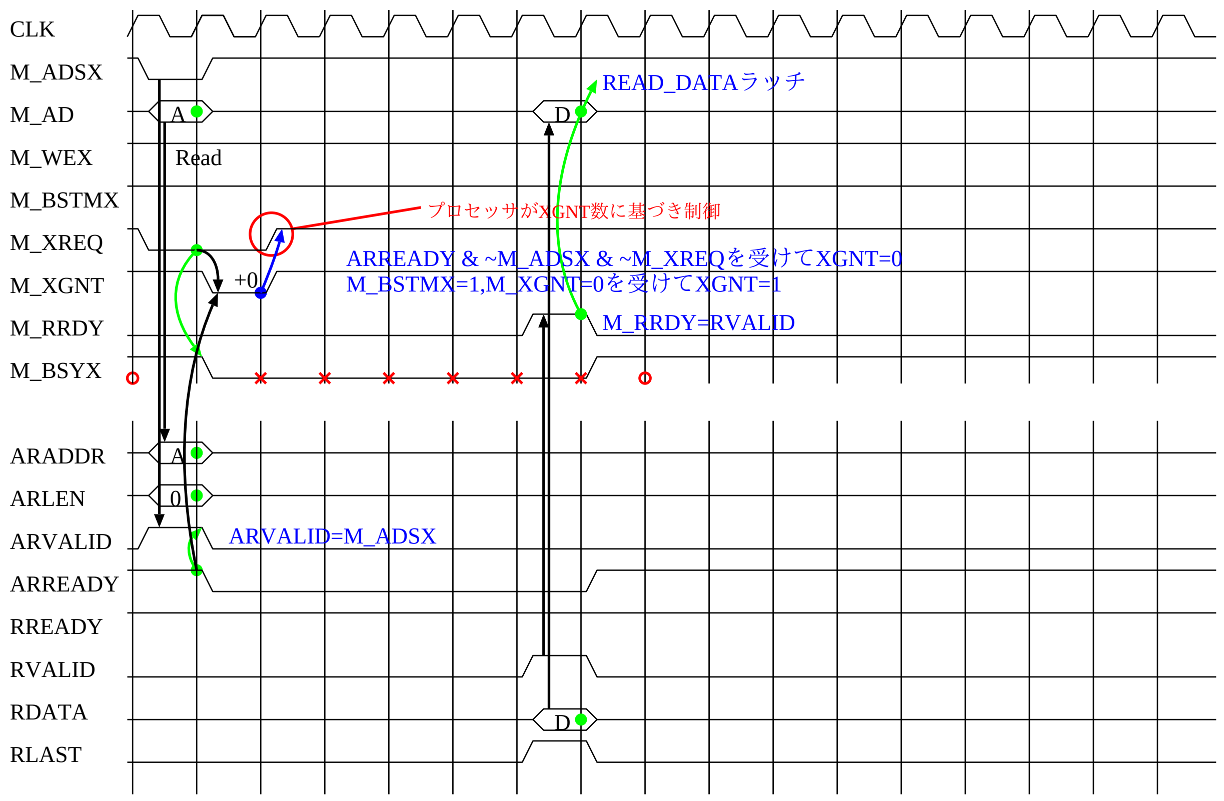


Figure.2.12: EMAX2 AXI ブリッジ仕様 (READ)

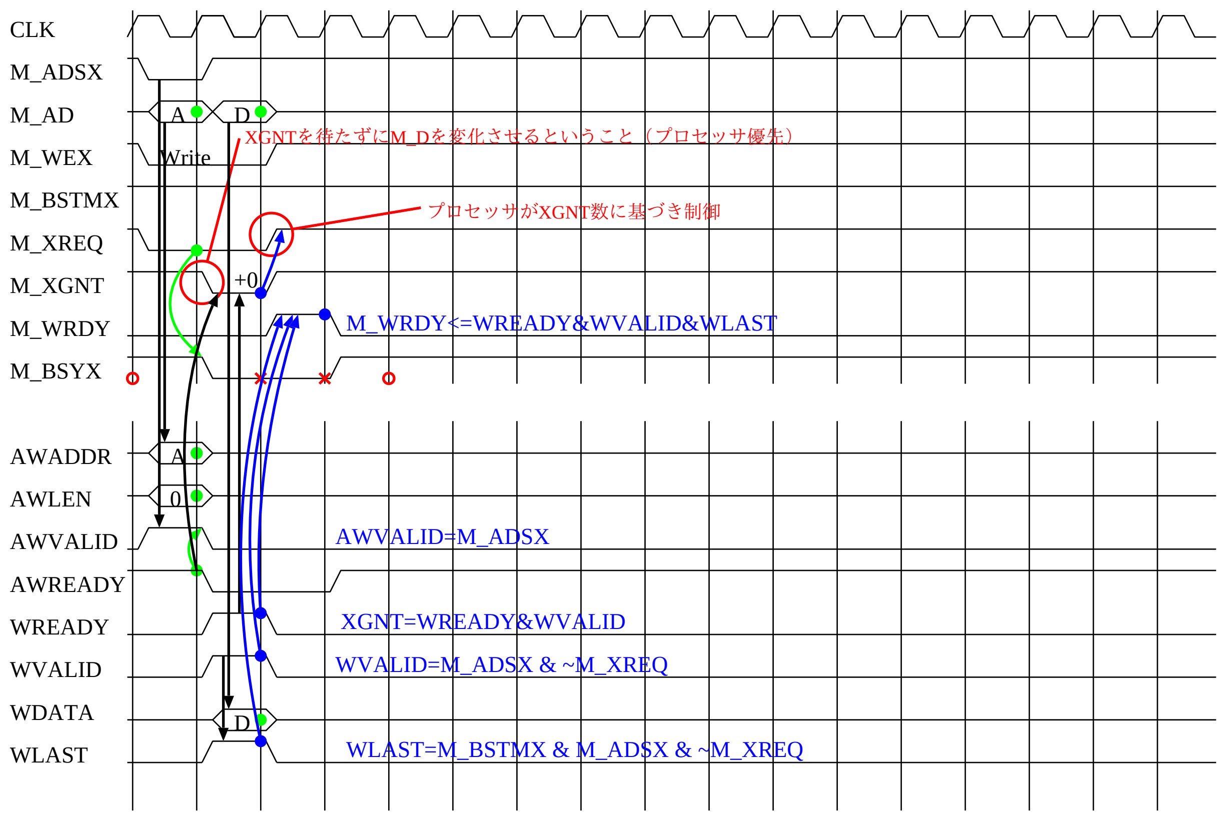


Figure.2.13: EMAX2 AXI ブリッジ仕様 (WRITE)

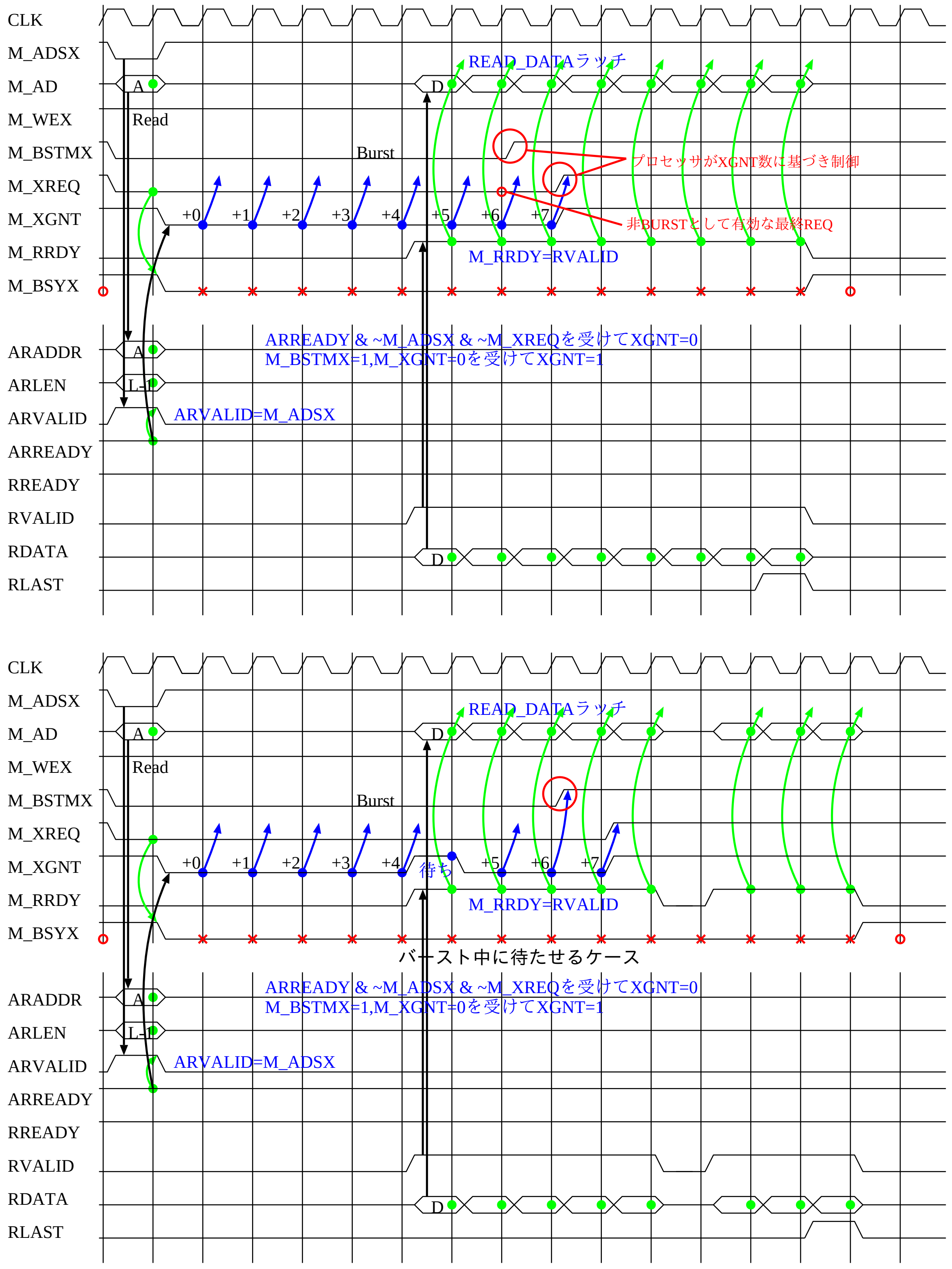


Figure.2.14: EMAX2 AXI ブリッジ仕様 (READ バースト)

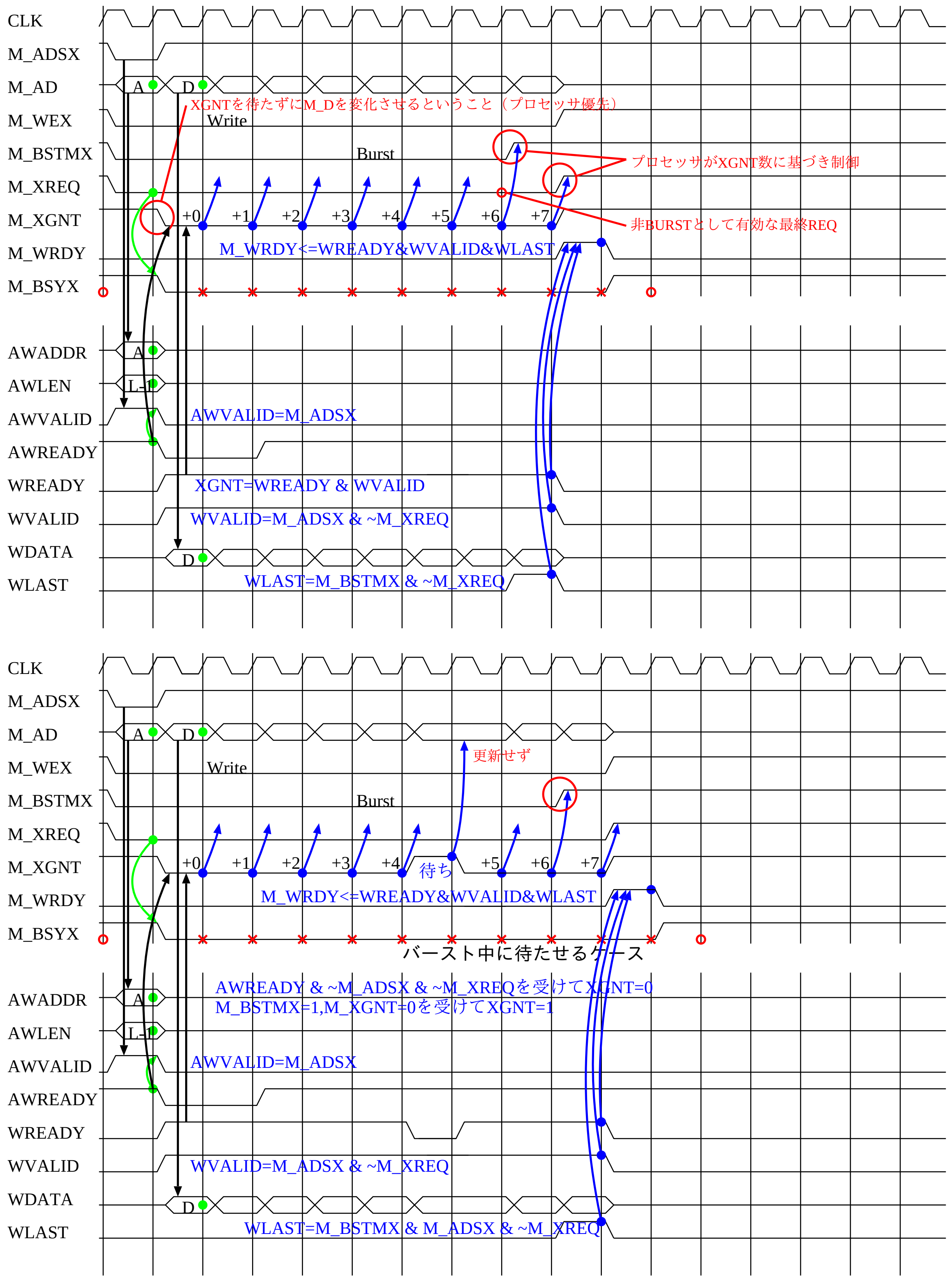

CLK
M_ADSX
M_AD
M_WEX
M_BSTMX
M_XREQ
M_XGNT
M_WRDY
M_BSYX
AWADDR
AWLEN
AWVALID
AWREADY
WREADY
WVALID
WDATA
WLAST
A
D
L-1
Write
Burst
+0
+1
+2
+3
+4
+5
+6
+7
XGNTを待たずにM_Dを変化させるということ（プロセッサ優先）
プロセッサがXGNT数に基づき制御
非BURSTとして有効な最終REQ
M_WRDY<=WREADY&WVALID&WLAST
AWVALID=M_ADSX
XGNT=WREADY & WVALID
WVALID=M_ADSX & ~M_XREQ
WLAST=M_BSTMX & ~M_XREQ
更新せず
待ち
バースト中に待たせるケース
AWREADY & ~M_ADSX & ~M_XREQを受けてXGNT=0
M_BSTMX=1,M_XGNT=0を受けてXGNT=1
WLAST=M_BSTMX & M_ADSX & ~M_XREQ

Figure.2.15: EMAX2 AXI ブリッジ仕様 (WRITE バースト)

2.3 リセット/起動/終了手順および主記憶空間

Table.2.2: リセット/起動/終了インタフェースと主記憶空間

AXI-FIFO	Notes	from EMAX2	from ARM
offset=0x00000000	Reset (EMAX2.RST=1 -> 0)	-- out of space --	OK for write
offset=0x00000004	Start EMAX2 (M_STAT=3)	-- out of space --	OK for write
offset=0x00000008	Status of EMAX2	-- out of space --	OK for read rdata[3:0]=pe0_status
AXI-MASTER	**Notes**	**from EMAX2**	**from ARM**
0x1fffffff-0x00000000	DDR3 Virtual (low-512MB)	-- out of space --	RW Virtual Space by OS
0x3fffffff-0x20000000	DDR3 Physical (high-512MB)	RW burst/non-burst	RW Physical by mmap()

0x2000007f-0x20000078	EMAX2 Handshake Status Reg
	L2CT_0000 bit5-4
		W=0書き込み時M_STAT=0
		W=1書き込み時M_STAT=1
		バースト時も有効
		
		※本機能は実装されない
		pe0_status[3:0]==9の
		観測時にM_STAT=1と
		なる
			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
			
			※本機能は実装されない
			AXI-FIFOへ0x0003を
			書き込むことで
			M_STAT=3になる

0x200027ff-0x20002000	conf 8word*64 = 2KB	RD burst	WR non-burst
0x200063ff-0x20006000	regv0 4word*64 = 1KB	RD burst base_offset=0	WR non-burst
0x200083ff-0x20008000	regv1 4word*64 = 1KB	RD burst base_offset=1	WR non-burst
0x2000a3ff-0x2000a000	lmmi0 4word*64 = 1KB	RD burst base_offset=0	WR non-burst
0x2000c3ff-0x2000c000	lmmi1 4word*64 = 1KB	RD burst base_offset=1	WR non-burst
0x201fffff-0x20100000	data-io#001 aligned to 1MB	RW burst	RW non-burst
	: (max 511 blocks)		
0x3fffffff-0x3ff00000	data-io#511 aligned to 1MB	RW burst	RW non-burst

EMAX2asic のリセット/起動/終了インタフェースおよび参照可能な ZYNQ 主記憶空間を表 2.2 に示す．ARM により AXI FIFO にエンキューされたリセットコマンド（offset=0, val=1）は，一定期間（512 サイクル/52.6MHz 程度），M0_RESET を 1 とした後に 0 とする FPGA 内回路を起動し，物理インタフェース RESET を通じて EMAX2asic 内部をリセットする．EMAX2asic が M_AD を通じて出力するアドレス情報は，8 バイト境界アドレスであり，M_AD[28:3] のみが有効である．これに 0x20000000 を OR した値が主記憶アドレスとして使用される．すなわち，EMAX2asic が参照可能な主記憶空間は，上位 512MB である．EMAX2asic は，EMAX2asic インタフェースハードウェアが，AXI FIFO にエンキューされた起動コマンド（offset=4, val=1）に基づき M_STAT[1:0] に 3 を反映することにより起動する．なお，起動前に，EMAX2asic コンパイラが生成した制御情報（conf[][]，regv[][]，および，lmmi[][]）を演算対象データとともに，ARM により主記憶上に配置しなければならない．EMAX2asic では，マクロパイプライニング機能が常に有効であり，次の連続演算に使用する regv[][] および lmmi[][] は，base_offset=0 の場合，各々 0x20006000 および 0x2000a000 から，base_offset=1 の場合，各々0x20008000 および 0x2000c000 から読み出される．演算対象データは，0x20100000-0x3fffffff の範囲に配置しなければならない．EMAX2asic が動作を終了すると，pe0_status に STATUS_WAIT(9) を表示する．EMAX2asic インタフェースハードウェアが，STATUS_WAIT に基づき M_STAT[1:0] に 1 を反映させることにより，EMAX2asic は待機状態に復帰する．ソフトウェアは，AXI_FIFO から読み出した値（pe0_status）が STATUS_IDLE(0) に変化したことをもって，EMAX2asic の動作完了を知ることができる．

2.4 動作状態と遷移

```
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.2.16: 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 2.16 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．

2.4.1 Idle (pe0_status=STATUS_IDLE(0))

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．

2.4.2 Unit configuration in progress (pe0_status=STATUS_CONF(1))

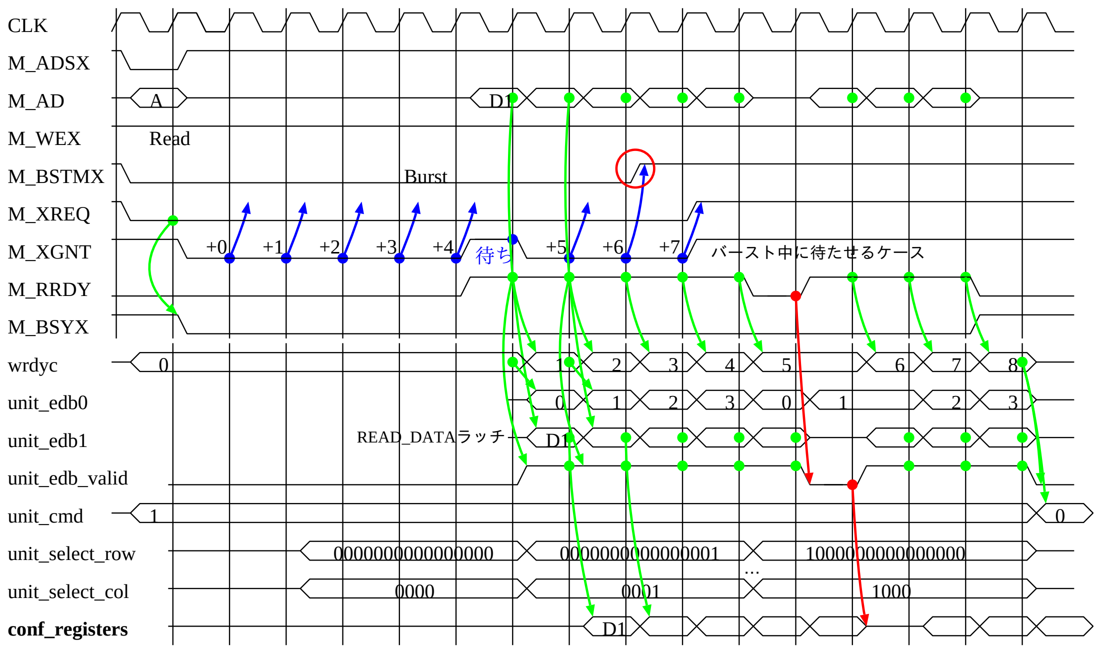


Figure.2.17: EMAX2asic インタフェース conf[][] → unit[][] のタイミングチャート

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．8 バイト幅バーストリード長は 256 である（ARLEN=255 を指定）．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 2.17 に示す．

2.4.3 Register initialization in progress (pe0_status=STATUS_REGV(2))

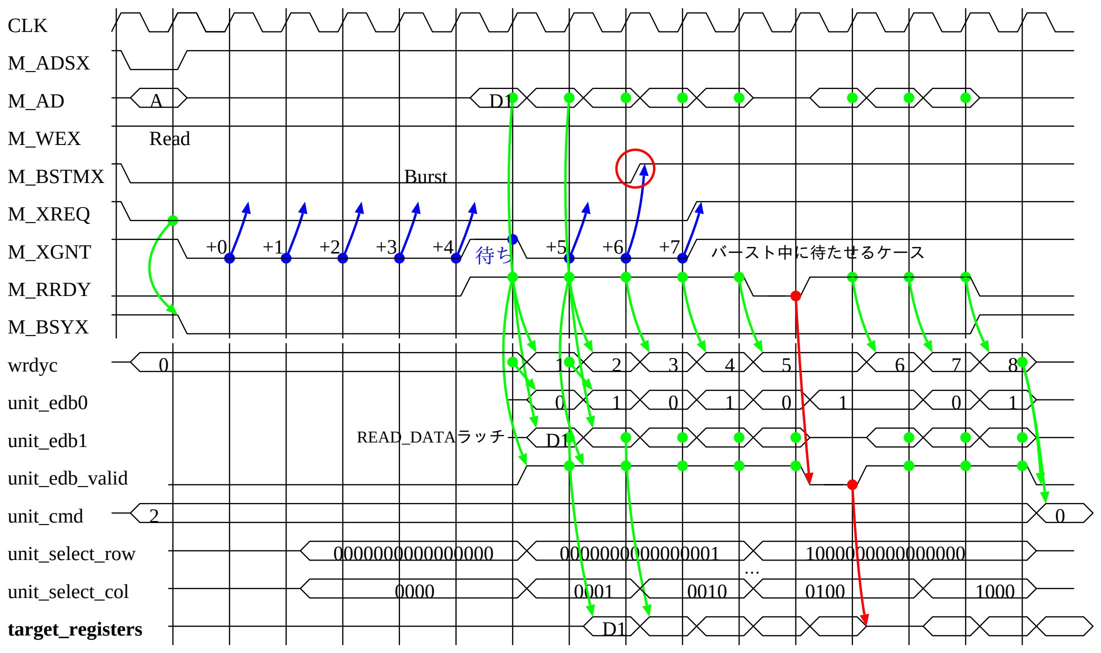


Figure.2.18: EMAX2asic インタフェース regv[][] → unit[][] のタイミングチャート

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．8 バイト幅バーストリード長は 128 である（ARLEN=127 を指定）．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 2.18 に示す．

2.4.4 LMM tag initialization in progress (pe0_status=STATUS_LMMI(3))

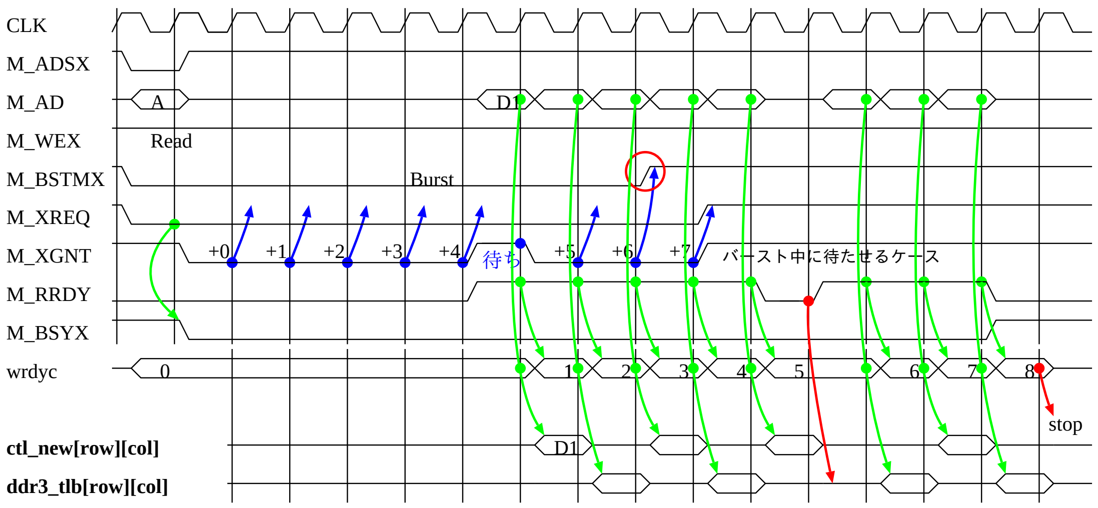


Figure.2.19: EMAX2asic インタフェース lmmi[][] → ctl_new[][]/ddr3_tlb[][] のタイミングチャート

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．8 バイト幅バーストリード長は 128 である（ARLEN=127 を指定）．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．なお，EMAX2/intel の場合，アプリケーションプログラムが指定した Intel 仮想アドレスを EMAX2 ドライバが FPGA ボード上の DDR3 物理アドレスに変換（管理単位は 1MB）し，EMAX2 が仮想アドレスを使用して DDR3 を直接参照する際のアドレス変換に ddr3_tlb[][] を使用していた．一方，EMAX2asic/ZYNQ の場合，EMAX2 は ARM の主記憶空間を直接参照できるものの，アドレスの管理単位を 1MB にできない．このため，EMAX2asic/ZYNQ では，アドレス変換機構を無効化して ARM 物理アドレスを直接使用する．すなわち，EMAX2asic を使用するアプリケーションプログラムは，emax2_start() を呼び出す際のアドレス引数に，ARM 物理アドレスを使用しなければならない．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 2.19 に示す．

2.4.5 LMM loading in progress (pe0_status=STATUS_LMM_LOAD(4))

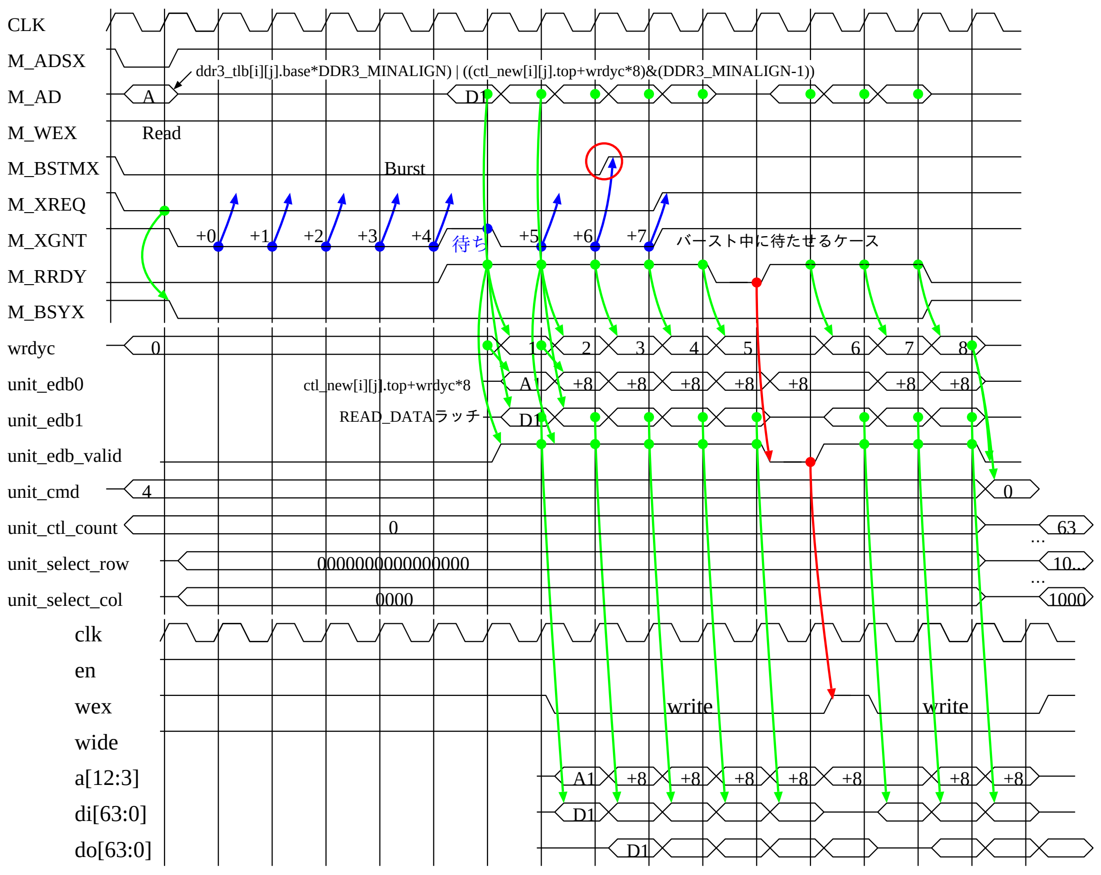


Figure.2.20: EMAX2asic インタフェース→ lmm[][] のタイミングチャート

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，EMAX2asic では，AXI MASTER があらかじめバースト長を必要とするため，8 バイト幅バーストリード長は 160 固定とする（ARLEN=159 を指定）．また，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 2.20 に示す．

2.4.6 Start execution (pe0_status=STATUS_START(5))

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．

2.4.7 Execution in progress (pe0_status=STATUS_EXEC(6))

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．

2.4.8 LMM drainage in progress (pe0_status=STATUS_LMM_DRAIN(7))

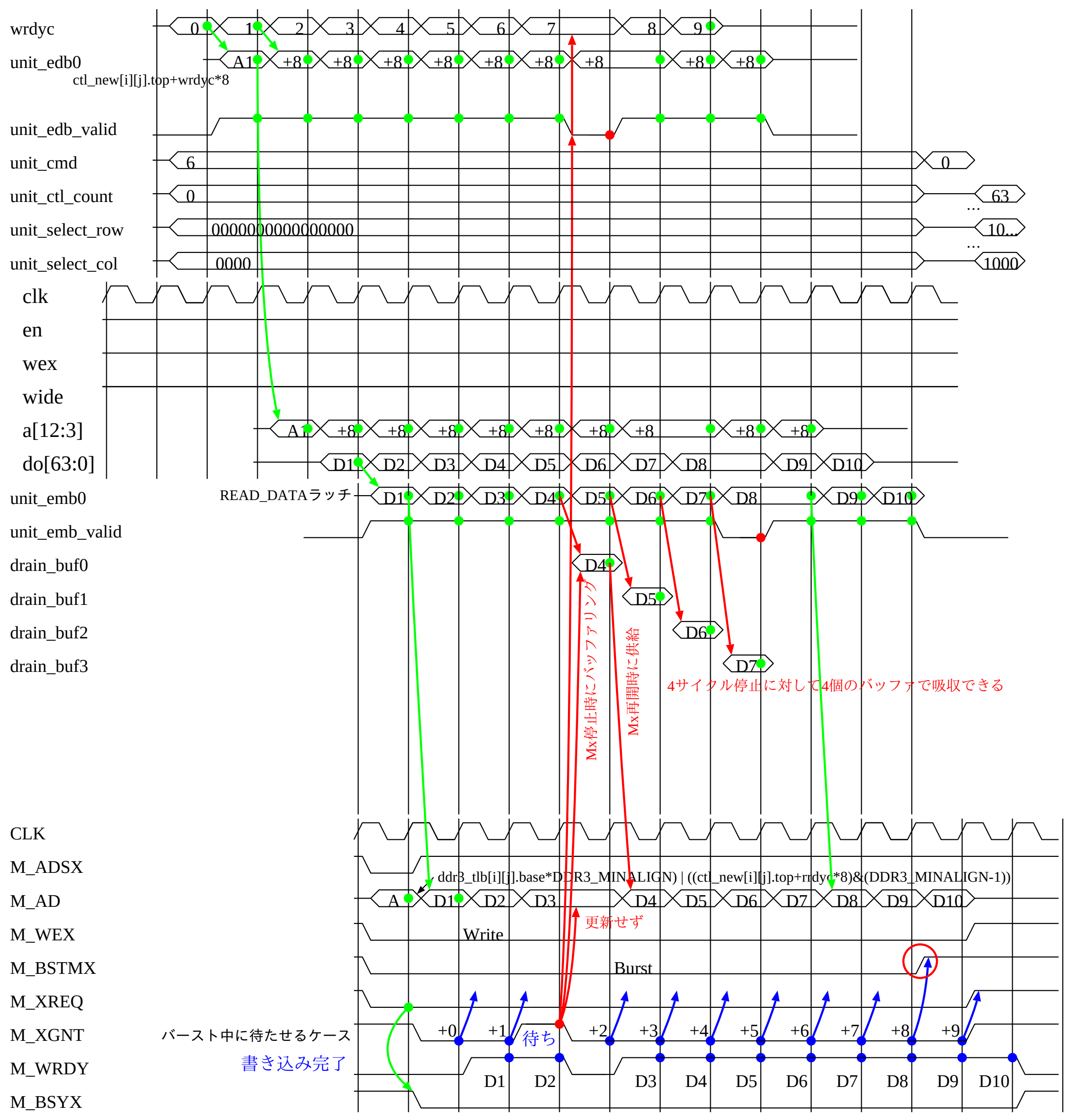


Figure.2.21: lmm[][] → EMAX2asic インタフェースのタイミングチャート

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．なお，EMAX2asic では，AXI MASTER があらかじめバースト長を必要とするため，8 バイト幅バーストリード長は 160 固定とする（AWLEN=159 を指定）．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 2.21 に示す．

2.4.9 Wait for execution (pe0_status=STATUS_TERM(8))

後処理を行い，pe0_status=STATUS_WAIT に遷移する．

2.4.10 Terminate execution (pe0_status=STATUS_WAIT(9))

base_offset を更新し，M_STAT が 1 でない場合は状態を維持する．M_STAT が 1 の場合，pe0_status=STATUS_IDLE に遷移する．

Chapter 3

EMAX4/bsim

3.1 Basic function

EMAX4/bsim は，EMAX2/intel にトランザクション機能を追加したアーキテクチャである．トランザクション機能を利用することにより，グラフ処理に見られる主記憶の不規則参照を CGRA のパイプライン動作に組み込むことが可能となる．EMAX4/bsim のシミュレータである bsim は，ARM をベースとするマルチコアおよびマルチスレッド機能を備えている．また，各コアは 1 基の EMAX4 を備えている．

3.2 Instruction format

See proj-arm32/src/conv-a2c/{conv-a2c.h, conv-a2c.l, conv-a2c.y}.

3.3 Application binary interface

See proj-arm32/src/conv-a2c/{emax4.h, emax4.c} and proj-arm32/src/bsim/emax4.c.

3.4 Examples (2D-imaging)

See proj-arm32/sample/filter/filter-emax4.S. Many stencil kernels for image processing are implemented.

3.4.1 Tone_curve with SIMD

```
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++;
  }
}
```

```
//EMAX4A start .emax_start_tone_curve:
//EMAX4A ctl map_dist=0
//EMAX4A @0,0 while (ri+=,-1) rgi[320,]  & ld   (ri+=,4),r9   rgi[.emax_rgi00_tone_curve:,] lmf[.emax_lmfla0_tone_curve:,0,0,0,0,.emax_lmfma0_tone_curve:,320]
//EMAX4A @1,0                            & ldub (ri,r9.3),r10 rgi[.emax_rgi01_tone_curve:,] lmr[.emax_lmrla1_tone_curve:,0,0,0,0,.emax_lmrma1_tone_curve:, 64]
//EMAX4A @1,1                            & ldub (ri,r9.2),r11 rgi[.emax_rgi02_tone_curve:,] lmr[.emax_lmrla2_tone_curve:,0,0,0,0,.emax_lmrma2_tone_curve:, 64]
//EMAX4A @1,2                            & ldub (ri,r9.1),r12 rgi[.emax_rgi03_tone_curve:,] lmr[.emax_lmrla3_tone_curve:,0,0,0,0,.emax_lmrma3_tone_curve:, 64]
//EMAX4A @2,0 mmrg3 (r10,r11,r12) rgi[,] & st   -,(ri+=,4)    rgi[.emax_rgi04_tone_curve:,] lmw[.emax_lmwla4_tone_curve:,0,0,0,0,.emax_lmwma4_tone_curve:,320]
//EMAX4A end .emax_end_tone_curve:
```

3.4.2 Hokan1 with SIMD and stencil

```
void hokan1(c, p, s)
     unsigned int *c, *p;
     unsigned short *s; /*[WD/4][8];*/
     /*hokan1(&W[i*WD], &R[(i+j)*WD], &SAD1[i/4][j+4]);*/
{
  int j;
  for (j=0; j<WD; j++) {
    int j2 = j/4*4;
    int k = j%4*2;                                                                                                   /* j2+k:0,2,4,6; 4,6,8,10; 8,10,12,14; 12,14,16,18; */
    * 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;
  }
}
```

```
//EMAX4A start .emax_start_hokan1:
//EMAX4A ctl map_dist=0
//EMAX4A @0,0 while (ri+=,-1) rgi[320,]
//EMAX4A @0,1 add  (ri+=,1) | and (-,~3)<<2,r12 rgi[-1,]
//EMAX4A @0,2 add  (ri+=,1) | and (-, 3)<<3,r13 rgi[-1,]
//EMAX4A @1,0 add  (ri,r12),r12                rgi[.emax_rgi00_hokan1:,]
//EMAX4A @1,1 add3 (ri,r12,r13),r13            rgi[.emax_rgi01_hokan1:,]
//EMAX4A @2,0                         ld  (r12,  0),r0  & ld  (r12+=,4),r31 rgi[.emax_rgix0_hokan1:,] lmr[.emax_lmrla00_hokan1:,0,0,0,0,.emax_lmrma00_hokan1:,320]
//EMAX4A @2,1                                           & ld  (r12,  4),r1
//EMAX4A @2,2                                           & ld  (r12,  8),r2
//EMAX4A @2,3                                           & ld  (r12, 12),r3
//EMAX4A @3,0                         ld  (r13,-16),r24 & ld  (r13+=,4),r31 rgi[.emax_rgix1_hokan1:,] lmr[.emax_lmrla01_hokan1:,0,0,0,0,.emax_lmrma01_hokan1:,320]
//EMAX4A @3,1                                           & ld  (r13,-12),r25
//EMAX4A @3,2                                           & ld  (r13, -8),r26
//EMAX4A @3,3                         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)
//EMAX4A @4,0 msad (r0,r25),r11
//EMAX4A @4,1 msad (r1,r26),r13  ! swap r1 and r26 to avoid collision of pos2
//EMAX4A @4,2 msad (r2,r27),r15
//EMAX4A @4,3 msad (r3,r28),r17
//EMAX4A @5,0 msad (r0,r24),r10
//EMAX4A @5,1 msad (r1,r25),r12
//EMAX4A @5,2 msad (r2,r26),r14
//EMAX4A @5,3 msad (r3,r27),r16
//EMAX4A @6,0 mauh (r10,r12),r10
//EMAX4A @6,1 mauh (r11,r13),r11
//EMAX4A @6,2 mauh (r14,r16),r14
//EMAX4A @6,3 mauh (r15,r17),r15
//EMAX4A @7,0 mauh (r10,r14) | suml (-),r10
//EMAX4A @7,1 mauh (r11,r15) | sumh (-),r11
//EMAX4A @7,2                                            & ld (ri+=,4),r0 rgi[.emax_rgi02_hokan1:,] lmf[.emax_lmfla02_hokan1:,0,0,0,0,.emax_lmfma02_hokan1:,320]
//EMAX4A @8,0 mauh3 (r0,r10,r11)                         & st -,(ri+=,4)  rgi[.emax_rgi05_hokan1:,] lmw[.emax_lmwla05_hokan1:,0,0,0,0,.emax_lmwma05_hokan1:,320]
//EMAX4A end .emax_end_hokan1:
```

3.4.3 Hokan2 with SIMD and stencil

```
void hokan2(s, sminxy, k)
     unsigned short *s; /*[WD/4][8];*/
     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 */
    int l1 = ((-2)<<24)|k|*(s  );
    int l2 = ((-1)<<24)|k|*(s+1);
    int l3 = ((-1)<<24)|k|*(s+2);
    int l4 = (( 0)<<24)|k|*(s+3);
    int l5 = (( 0)<<24)|k|*(s+4);
    int l6 = (( 0)<<24)|k|*(s+5);
    int l7 = (( 1)<<24)|k|*(s+6);
    int l8 = (( 1)<<24)|k|*(s+7);
    if ((sminxy[j]&0xffff) > *(s  )) sminxy[j] = l1;
    if ((sminxy[j]&0xffff) > *(s+1)) sminxy[j] = l2;
    if ((sminxy[j]&0xffff) > *(s+2)) sminxy[j] = l3;
    if ((sminxy[j]&0xffff) > *(s+3)) sminxy[j] = l4;
    if ((sminxy[j]&0xffff) > *(s+4)) sminxy[j] = l5;
    if ((sminxy[j]&0xffff) > *(s+5)) sminxy[j] = l6;
    if ((sminxy[j]&0xffff) > *(s+6)) sminxy[j] = l7;
    if ((sminxy[j]&0xffff) > *(s+7)) sminxy[j] = l8;
    s += 2;
  }
}
```

```
//EMAX4A start .emax_start_hokan2:
//EMAX4A ctl map_dist=0
//EMAX4A @0,0 while (ri+=,-1) rgi[320,]                      & ld (r31+=,4),r10 rgi[.emax_rgi00_hokan2:,]
//EMAX4A @0,1 |or (ri,(-2<<24)),r28 rgi[.emax_rgi05_hokan2:,] & ld (r31+=,4),r12 rgi[.emax_rgi01_hokan2:,]
//EMAX4A @0,2 |or (ri,(-1<<24)),r29 rgi[.emax_rgi06_hokan2:,] & ld (r31+=,4),r14 rgi[.emax_rgi02_hokan2:,]
//EMAX4A @0,3 |or (ri,( 1<<24)),r31 rgi[.emax_rgi07_hokan2:,] & ld (r31+=,4),r16 rgi[.emax_rgi03_hokan2:,] lmr[.emax_lmrla03_hokan2:,0,0,0,0,.emax_lmrma03_hokan2:,320]
//EMAX4A @1,0 minl3 (r29,r28,r10),r10
//EMAX4A @1,1 minl3 (ri,r29, r12),r12 rgi[.emax_rgi10_hokan2:,]
//EMAX4A @1,2 minl3 (ri, ri, r14),r14 rgi[.emax_rgi12_hokan2:,.emax_rgi11_hokan2:]
//EMAX4A @1,3 minl3 (r31,r31,r16),r16
//EMAX4A @2,0 minl (r10,r12),r10
//EMAX4A @2,2 minl (r14,r16),r14
//EMAX4A @3,0 minl (r10,r14),r10                             & ld (ri+=,4),r11 rgi[.emax_rgi04_hokan2:,] lmf[.emax_lmfla04_hokan2:,0,0,0,0,.emax_lmfma04_hokan2:,320]
//EMAX4A @4,0 minl (r10,r11)                                 & st -,(ri+=,4)   rgi[.emax_rgi08_hokan2:,] lmw[.emax_lmwla08_hokan2:,0,0,0,0,.emax_lmwma08_hokan2:,320]
//EMAX4A end .emax_end_hokan2:
```

3.4.4 Hokan3 with SIMD and stencil

```
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];
  }
}
```

```
//EMAX4A start .emax_start_hokan3:
//EMAX4A ctl map_dist=0
//EMAX4A @0,0 while (ri+=,-1) rgi[320,]
//EMAX4A @0,1 add (ri+=,1) |  and (-,~3)<<2,r12 rgi[-1,]
//EMAX4A @0,2 add (ri+=,1) |  or  (-, 0)<<2,r14 rgi[-1,]
//EMAX4A @1,0                                        & ld (ri,r12),r16 rgi[.emax_rgi00_hokan3:,] lmr[.emax_lmrla00_hokan3:,0,0,0,0,.emax_lmrma00_hokan3:,320]
//EMAX4A @2,0 add (ri,r14),r13   rgi[.emax_rgi01_hokan3:,]
//EMAX4A @2,1 |and (r16,0xff000000)>A22,r17 ! >A は SRA
//EMAX4A @2,2 |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
//EMAX4A @3,0 sub (r18,ri),r0 c0 rgi[,.emax_rgi03_hokan3:] & ld (r13,r17),r16 rgi[,]                   lmr[.emax_lmrla01_hokan3:,0,0,0,0,.emax_lmrma01_hokan3:,320]
//EMAX4A @4,0 cexe (,,,c0,0xaaaa)                          & st r16,(ri+=,4) rgi[.emax_rgi02_hokan3:,] lmx[.emax_lmxla02_hokan3:,0,0,0,0,.emax_lmxma02_hokan3:,320]
//EMAX4A end .emax_end_hokan3:
```

3.4.5 Expand4k with SIMD and stencil

```
void expand4k(p, r, kad, sk1, sk2)
     unsigned int *p, *r;
     int kad, sk1, sk2;
{
  /*    ┌───┬───┬───┐                */
  /*    │   │ k-1   │ p[k][l:320]  */
  /*    ├───┼───┼───┤ r[i][j:1024] */
  /*    l-1 │ kl │ l+1  i:1-767      */
  /*    ├───┼───┼───┤                */
  /*    │   │ l+1   │                */
  /*    └───┴───┴───┘                */

  /*  ┌─────┬──────────────────────────┐      */
  /*  │ ┌── │ ──────┐                  │      */
  /*  │ │   │       │                  │ k-1 */
  /*  ├─────┼──────────────────────────┤      */
  /*  │ │   │★      │                0 │     */
  /*  │ └── │ ──────┘                  │      */
  /*  │     │ (((i*HT)<<4)/768)&15 : 8  │ k   */
  /*  │     │                           │    */
  /*  │     │                        15 │    */
  /*  ├─────┼──────────────────────────┤      */
  /*  │     │                          │ k+1 */
  /*  └─────┴──────────────────────────┘      */
  /*  ★を中心とする正方形が２×２領域の個々と重なる面積比を kfraq と lfraq で表現する */

  int j;
  unsigned int ph, pl, x;

  for (j=0; j<1024; j++) { /* 本当は 4095 まで */
    int p1 = j*WD/1024;
    int lfraq = (((j*WD)<<4)/1024)&15; /* 4bit */
    int lad = 16-ad(lfraq,8);
    int sl1 = ss(lfraq,8);
    int sl2 = ss(8,lfraq);
    int r1 = kad*lad; /* 4bit*4bit */
    int r3 = kad*sl1; /* 4bit*4bit */
    int r2 = kad*sl2; /* 4bit*4bit */
    int r5 = sk1*lad; /* 4bit*4bit */
    int r9 = sk1*sl1; /* 4bit*4bit */
    int r8 = sk1*sl2; /* 4bit*4bit */
    int r4 = sk2*lad; /* 4bit*4bit */
    int r7 = sk2*sl1; /* 4bit*4bit */
    int r6 = sk2*sl2; /* 4bit*4bit */
    ph = madd(mmul(b2h(p[p1     ], 1), r1), mmul(b2h(p[p1-1], 1), r2));
    ph = madd(mmul(b2h(p[p1   +1], 1), r3), ph);
    ph = madd(mmul(b2h(p[p1-WD  ], 1), r4), ph);
    ph = madd(mmul(b2h(p[p1+WD  ], 1), r5), ph);
    ph = madd(mmul(b2h(p[p1-WD-1], 1), r6), ph);
    ph = madd(mmul(b2h(p[p1-WD+1], 1), r7), ph);
    ph = madd(mmul(b2h(p[p1+WD-1], 1), r8), ph);
    ph = madd(mmul(b2h(p[p1+WD+1], 1), r9), ph);
    pl = madd(mmul(b2h(p[p1     ], 0), r1), mmul(b2h(p[p1-1], 0), r2));
    pl = madd(mmul(b2h(p[p1   +1], 0), r3), pl);
    pl = madd(mmul(b2h(p[p1-WD  ], 0), r4), pl);
    pl = madd(mmul(b2h(p[p1+WD  ], 0), r5), pl);
    pl = madd(mmul(b2h(p[p1-WD-1], 0), r6), pl);
    pl = madd(mmul(b2h(p[p1-WD+1], 0), r7), pl);
    pl = madd(mmul(b2h(p[p1+WD-1], 0), r8), pl);
    pl = madd(mmul(b2h(p[p1+WD+1], 0), r9), pl);
    *r = h2b(msrl(ph, 8), 1) | h2b(msrl(pl, 8), 0);
    r++;
  }
}
```

```
//EMAX4A start .emax_start_expand4k:
//EMAX4A ctl map_dist=0
//EMAX4A @0,0 while (ri+=,-1) rgi[1024,]
//EMAX4A @0,1  add (ri+=,320) |  and (-,~1023)>>8,r0 rgi[-320,] ! p1*4
//EMAX4A @0,2  add (ri+=,320) |  and (-,0x3c0)>>6,r4 rgi[-320,] ! lfraq
//EMAX4A @1,0  add  (ri,r0),r0                       rgi[.emax_rgi_p____expand4k:,]
//EMAX4A @1,1  msuh (r4,ri),r1                        rgi[,8] ! sl1
//EMAX4A @1,2  msuh (ri,r4),r2                        rgi[8,] ! sl2
//EMAX4A @1,3  msad (r4,ri),r3                        rgi[,8]
//EMAX4A @2,3  msuh (ri,r3),r3                        rgi[16,] ! lad
//EMAX4A @3,1  mluh (ri,r1),r21  rgi[.emax_rgi_sk21_expand4k:,] & ld (r0,-1276),r10  lmr[.emax_lmrla_PREV_expand4k:,0,0,0,0,.emax_lmrma_PREV_expand4k:,320]
//EMAX4A @3,2  mluh (ri,r2),r22  rgi[.emax_rgi_sk22_expand4k:,] & ld (r0,-1284),r11
//EMAX4A @3,3  mluh (ri,r3),r23  rgi[.emax_rgi_sk20_expand4k:,] & ld (r0,-1280),r12
//EMAX4A @4,1  mluh (r10.l,r21),r13
//EMAX4A @4,2  mluh (r11.l,r22),r14
//EMAX4A @4,3  mluh (r12.l,r23),r15
//EMAX4A @5,0  mluh (r10.h,r21),r13
//EMAX4A @5,1  mauh3 (r13,r14,r15),r16
//EMAX4A @5,2  mluh (r11.h,r22),r14
//EMAX4A @5,3  mluh (r12.h,r23),r15
//EMAX4A @6,0  mauh3 (r13,r14,r15),r17
//EMAX4A @6,1  mluh (ri,r1),r21  rgi[.emax_rgi_kad1_expand4k:,] & ld (r0,   4),r10   lmr[.emax_lmrla_CURR_expand4k:,0,0,0,0,.emax_lmrma_CURR_expand4k:,320]
//EMAX4A @6,2  mluh (ri,r2),r22  rgi[.emax_rgi_kad2_expand4k:,] & ld (r0,  -4),r11
//EMAX4A @6,3  mluh (ri,r3),r23  rgi[.emax_rgi_kad0_expand4k:,] & ld (r0,   0),r12
//EMAX4A @7,1  mluh (r10.l,r21),r13
//EMAX4A @7,2  mluh (r11.l,r22),r14
//EMAX4A @7,3  mluh (r12.l,r23),r15
//EMAX4A @8,0  mluh (r10.h,r21),r13
//EMAX4A @8,1  mauh3 (r13,r14,r15),r18
//EMAX4A @8,2  mluh (r11.h,r22),r14
//EMAX4A @8,3  mluh (r12.h,r23),r15
//EMAX4A @9,0  mauh3 (r13,r14,r15),r19
//EMAX4A @9,1  mluh (ri,r1),r21  rgi[.emax_rgi_sk11_expand4k:,] & ld (r0, 1284),r10  lmr[.emax_lmrla_NEXT_expand4k:,0,0,0,0,.emax_lmrma_NEXT_expand4k:,320]
//EMAX4A @9,2  mluh (ri,r2),r22  rgi[.emax_rgi_sk12_expand4k:,] & ld (r0, 1276),r11
//EMAX4A @9,3  mluh (ri,r3),r23  rgi[.emax_rgi_sk10_expand4k:,] & ld (r0, 1280),r12
//EMAX4A @10,1  mluh (r10.l,r21),r13
//EMAX4A @10,2  mluh (r11.l,r22),r14
//EMAX4A @10,3  mluh (r12.l,r23),r15
//EMAX4A @11,0  mluh (r10.h,r21),r13
//EMAX4A @11,1  mauh3 (r13,r14,r15),r20
//EMAX4A @11,2  mluh (r11.h,r22),r14
//EMAX4A @11,3  mluh (r12.h,r23),r15
//EMAX4A @12,0  mauh3 (r13,r14,r15),r21
//EMAX4A @13,0  mauh3 (r17,r19,r21) | or (-,0)>M8,r21
//EMAX4A @13,1  mauh3 (r16,r18,r20) | or (-,0)>M8,r20
//EMAX4A @14,0  mh2bw (r21,r20)   & st -,(ri+=,4) rgi[.emax_rgi_store_expand4k:,] lmw[.emax_lmwla_store_expand4k:,0,0,0,0,.emax_lmwma_store_expand4k:,1024]
//EMAX4A end .emax_end_expand4k:
```

3.4.6 Unsharp with SIMD and stencil

```
inline unsigned char limitRGB(int c) {
  if (c<0x00) return 0x00;
  if (c>0xff) return 0xff;
  return c;
}

void unsharp(p, r)
     unsigned char *p;
     unsigned char *r;
{
  int t0,t1,t2;
  int j, k;
  int p0 = ((0  )*WD+(1  ))*4;  // p1 p5 p2
  int p1 = ((0-1)*WD+(1-1))*4;  // p6 p0 p7
  int p2 = ((0-1)*WD+(1+1))*4;  // p3 p8 p4
  int p3 = ((0+1)*WD+(1-1))*4;
  int p4 = ((0+1)*WD+(1+1))*4;
  int p5 = ((0-1)*WD+(1  ))*4;
  int p6 = ((0  )*WD+(1-1))*4;
  int p7 = ((0  )*WD+(1+1))*4;
  int p8 = ((0+1)*WD+(1  ))*4;
  for (j=0; j<WD; j++) {
    r[p0+0] = 0;

    t0 = p[p0+1];
    t1 = p[p1+1] + p[p2+1] + p[p3+1] + p[p4+1];
    t2 = p[p5+1] + p[p6+1] + p[p7+1] + p[p8+1];
    r[p0+1] = limitRGB(( t0 * 239 - t1 * 13 - t2 * 15 - t2/4) >> 7);

    t0 = p[p0+2];
    t1 = p[p1+2] + p[p2+2] + p[p3+2] + p[p4+2];
    t2 = p[p5+2] + p[p6+2] + p[p7+2] + p[p8+2];
    r[p0+2] = limitRGB(( t0 * 239 - t1 * 13 - t2 * 15 - t2/4) >> 7);

    t0 = p[p0+3];
    t1 = p[p1+3] + p[p2+3] + p[p3+3] + p[p4+3];
    t2 = p[p5+3] + p[p6+3] + p[p7+3] + p[p8+3];
    r[p0+3] = limitRGB(( t0 * 239 - t1 * 13 - t2 * 15 - t2/4) >> 7);

    p0+=4; p1+=4; p2+=4; p3+=4; p4+=4; p5+=4; p6+=4; p7+=4; p8+=4;
  }
}
```

```
//EMAX4A start .emax_start_unsharp:
//EMAX4A ctl map_dist=1
//EMAX4A @0,0 while (ri+=,-1) rgi[320,]
//EMAX4A @0,1 add  (ri+=,4),r10     rgi[.emax_rgi_p____unsharp:,]
//EMAX4A @1,0                                    & ld (r10,-1276),r1   lmr[.emax_lmrla_PREV_unsharp:,0,0,0,0,.emax_lmrma_PREV_unsharp:,320]
//EMAX4A @1,1                                    & ld (r10,-1284),r2
//EMAX4A @1,2                                    & ld (r10,-1280),r5
//EMAX4A @2,0 mauh  (r1.l,r2.l),r11              & ld (r10,    4),r6   lmr[.emax_lmrla_CURR_unsharp:,0,0,0,0,.emax_lmrma_CURR_unsharp:,320]
//EMAX4A @2,1                                    & ld (r10,   -4),r7
//EMAX4A @2,2                                    & ld (r10,    0),r0
//EMAX4A @2,3 mauh  (r1.h,r2.h),r12
//EMAX4A @3,0 mluh  (r0.l,ri),r20   rgi[,239]    & ld (r10, 1284),r3   lmr[.emax_lmrla_NEXT_unsharp:,0,0,0,0,.emax_lmrma_NEXT_unsharp:,320]
//EMAX4A @3,1 mluh  (r0.h,ri),r21   rgi[,239]    & ld (r10, 1276),r4
//EMAX4A @3,2 mauh  (r5.l,r6.l),r15
//EMAX4A @3,3 mauh  (r5.h,r6.h),r16              & ld (r10, 1280),r8
//EMAX4A @4,0 mauh3 (r11,r3.l,r4.l),r11
//EMAX4A @4,1 mauh3 (r12,r3.h,r4.h),r12
//EMAX4A @5,0 mluh  (r11,ri),r13    rgi[,13]
//EMAX4A @5,1 mluh  (r12,ri),r14    rgi[,13]
//EMAX4A @5,2 mauh3 (r15,r7.l,r8.l),r15
//EMAX4A @5,3 mauh3 (r16,r7.h,r8.h),r16
//EMAX4A @6,0 | or  (r15,0)>M2,r7
//EMAX4A @6,1 mluh  (r15,ri),r17    rgi[,15]
//EMAX4A @6,2 | or  (r16,0)>M2,r8
//EMAX4A @6,3 mluh  (r16,ri),r18    rgi[,15]
//EMAX4A @7,0 msuh3 (r20,r7,r17),r20
//EMAX4A @7,2 msuh3 (r21,r8,r18),r21
//EMAX4A @8,0 msuh (r20,r13) | or (-,0)>M7,r20
//EMAX4A @8,2 msuh (r21,r14) | or (-,0)>M7,r21
//EMAX4A @9,0 mh2bw (r21,r20) & st -,(ri+=,4)   rgi[.emax_rgi_store_unsharp:,] lmw[.emax_lmwla_store_unsharp:,0,0,0,0,.emax_lmwma_store_unsharp:,320]
//EMAX4A end .emax_end_unsharp:
```

3.4.7 Blur with SIMD and stencil

```
void blur(p, r)
     unsigned int *p, *r;
{
  int j, k, l;

  int p0 = (0  )*WD  ;
  int p1 = (0  )*WD  ;
  int p2 = (0  )*WD-1;
  int p3 = (0  )*WD+1;
  int p4 = (0-1)*WD  ;
  int p5 = (0+1)*WD  ;
  int p6 = (0-1)*WD-1;
  int p7 = (0-1)*WD+1;
  int p8 = (0+1)*WD-1;
  int p9 = (0+1)*WD+1;
  for (j=0; j<WD; j++) {
    unsigned int s0,s1,s2,s3,s4,s5,s6,s7,s8;
    unsigned int t0,t1,t2;
    s0=p[p1];s1=p[p2];s2=p[p3];s3=p[p4];s4=p[p5];s5=p[p6];s6=p[p7];s7=p[p8];s8=p[p9];
    /* ┌─┬─┬─┐   ┌─┬─┬─┐   ┌─┬─┬─┐   ┌─┬─┬─┐   ┌─┬─┬─┐   ┌─┬─┬─┐   ┌─┐
       │5│3│6│   │5<3<★│   │5│3│2│   │5<3<★│   │5│  │3│   │5<  <★│   │5│
       ├V┼V┼V┤   ├─┼─┼─┤   ├V┼V┼V┤   ├─┼─┼─┤   ├V┼──┼V┤   ├─┼─┼─┤   ├V┤
       │1│0│2│   │1<0<2│   │-│0│-│   │  0   │   │ │  │0│   │ │ │0│   │0│中間値確定
       ├V┼V┼V┤   ├─┼─┼─┤   ├V┼V┼V┤   ├─┼─┼─┤   ├V┼──┼V┤   ├─┼─┼─┤   ├V┤
       │7│4│8│   │★<4<8│   │1│4│8│   │★<4<8│   │4│  │8│   │★<  <8│   │8│
       └─┴─┴─┘   └─┴─┴─┘   └─┴─┴─┘   └─┴─┴─┘   └─┴─┴─┘   └─┴─┴─┘   └─┘*/
    t0 = pmax3(s5,s1,s7); t1 = pmid3(s5,s1,s7); t2 = pmin3(s5,s1,s7);      s5 = t0; s1 = t1; s7 = t2;
    t0 = pmax3(s3,s0,s4); t1 = pmid3(s3,s0,s4); t2 = pmin3(s3,s0,s4);      s3 = t0; s0 = t1; s4 = t2;
    t0 = pmax3(s6,s2,s8); t1 = pmid3(s6,s2,s8); t2 = pmin3(s6,s2,s8);      s6 = t0; s2 = t1; s8 = t2;

    t0 = pmin3(s5,s3,s6); t1 = pmid3(s5,s3,s6);                            s5 = t0; s3 = t1;
    t0 = pmin3(s1,s0,s2); t1 = pmid3(s1,s0,s2); t2 = pmax3(s1,s0,s2);      s1 = t0; s0 = t1; s2 = t2;
    t0 = pmid3(s7,s4,s8); t1 = pmax3(s7,s4,s8);                            s4 = t0; s8 = t1;

    t0 = pmax2(s5,s1);    t1 = pmin2(s5,s1);                               s5 = t0; s1 = t1;
    t0 = pmax3(s3,s0,s4); t1 = pmid3(s3,s0,s4); t2 = pmin3(s3,s0,s4);      s3 = t0; s0 = t1; s4 = t2;
    t0 = pmax2(s2,s8);    t1 = pmin2(s2,s8);                               s2 = t0; s8 = t1;

    t0 = pmin3(s5,s3,s2); t1 = pmid3(s5,s3,s2);                            s5 = t0; s3 = t1;
    t0 = pmid3(s1,s4,s8); t1 = pmax3(s1,s4,s8);                            s4 = t0; s8 = t1;

    t0 = pmax2(s5,s4);    t1 = pmin2(s5,s4);                               s5 = t0; s4 = t1;
    t0 = pmax3(s3,s0,s8); t1 = pmid3(s3,s0,s8); t2 = pmin3(s3,s0,s8);      s3 = t0; s0 = t1; s8 = t2;

    s5 = pmin2(s5,s3);    s8 = pmax2(s4,s8);

    r[p0] = pmid3(s5,s0,s8);
    p0++; p1++; p2++; p3++; p4++; p5++; p6++; p7++; p8++; p9++;
  }
}
```

```
//EMAX4A start .emax_start_blur:
//EMAX4A ctl map_dist=2
//EMAX4A @0,0 while (ri+=,-1) rgi[320,]
//EMAX4A @0,1 add  (ri+=,4),r10     rgi[.emax_rgi_p____blur:,]
//EMAX4A @1,0                    ld (r10,-1284),r15 & ld (r10,-1276),r7   lmr[.emax_lmrla_PREV_blur:,0,0,0,0,.emax_lmrma_PREV_blur:,320]
//EMAX4A @1,1                    ld (r10,-1284),r25 & ld (r10,-1280),r1
//EMAX4A @1,2                                       & ld (r10,-1284),r5
//EMAX4A @2,0 mmin3 (r7,r1,r15),r7
//EMAX4A @2,1 mmid3 (r7,r1,r25),r1
//EMAX4A @2,2 mmax3 (r7,r1, r5),r5
//EMAX4A @3,0                    ld (r10,   -4),r13 & ld (r10,    4),r4   lmr[.emax_lmrla_CURR_blur:,0,0,0,0,.emax_lmrma_CURR_blur:,320]
//EMAX4A @3,1                    ld (r10,   -4),r23 & ld (r10,    0),r0
//EMAX4A @3,2                                       & ld (r10,   -4),r3
//EMAX4A @4,0 mmin3 (r4,r0,r13),r4
//EMAX4A @4,1 mmid3 (r4,r0,r23),r0
//EMAX4A @4,2 mmax3 (r4,r0, r3),r3
//EMAX4A @5,0                    ld (r10, 1276),r16 & ld (r10, 1284),r8   lmr[.emax_lmrla_NEXT_blur:,0,0,0,0,.emax_lmrma_NEXT_blur:,320]
//EMAX4A @5,1                    ld (r10, 1276),r26 & ld (r10, 1280),r2
//EMAX4A @5,2                                       & ld (r10, 1276),r6
//EMAX4A @6,0 mmin3 (r8,r2,r16),r8
//EMAX4A @6,1 mmid3 (r8,r2,r26),r2
//EMAX4A @6,2 mmax3 (r8,r2, r6),r6
/*step-2*/
//EMAX4A @7,0 mmax3 (r1,r0,r2),r2
//EMAX4A @7,1 mmid3 (r1,r0,r2),r0
//EMAX4A @7,2 mmin3 (r1,r0,r2),r1
//EMAX4A @8,0 mmax3 (r7,r4,r8),r8
//EMAX4A @8,1 mmid3 (r7,r4,r8),r4
//EMAX4A @8,2 mmid3 (r5,r3,r6),r3
//EMAX4A @8,3 mmin3 (r5,r3,r6),r5
/*step-3*/
//EMAX4A @9,0 mmin3 (r3,r0,r4),r4
//EMAX4A @9,1 mmid3 (r3,r0,r4),r0
//EMAX4A @9,2 mmax3 (r3,r0,r4),r3
//EMAX4A @10,0 mmin  (r2,r8),r8
//EMAX4A @10,1 mmax  (r2,r8),r2
//EMAX4A @10,2 mmin  (r5,r1),r1
//EMAX4A @10,3 mmax  (r5,r1),r5
/*step-4*/
//EMAX4A @11,0 mmid3 (r1,r4,r8),r4
//EMAX4A @11,1 mmin3 (r5,r3,r2),r5
/*step-5*/
//EMAX4A @12,0 mmax3 (r1,r4,r8),r8
//EMAX4A @12,1 mmid3 (r5,r3,r2),r3
//EMAX4A @12,2 mmin  (r5,r4),r14
//EMAX4A @12,3 mmax  (r5,r4),r15
//EMAX4A @13,0 mmin3 (r3,r0,r8),r8
//EMAX4A @13,1 mmid3 (r3,r0,r8),r0
//EMAX4A @13,2 mmax3 (r3,r0,r8),r3
/*step-6*/
//EMAX4A @14,0 mmax  (r14,r8),r8
//EMAX4A @14,3 mmin  (r15,r3),r5
/*step-7*/
//EMAX4A @15,0 mmid3 (r5,r0,r8) & st -,(ri+=,4)   rgi[.emax_rgi_store_blur:,] lmw[.emax_lmwla_store_blur:,0,0,0,0,.emax_lmwma_store_blur:,320]
//EMAX4A end .emax_end_blur:
```

3.4.8 Edge with SIMD and stencil

```
void edge(p, r)
     unsigned int *p;
     unsigned char *r;
{
  int j, k;

  int p0 = (0  )*WD  ;
  int p1 = (0-1)*WD-1;
  int p2 = (0+1)*WD+1;
  int p3 = (0-1)*WD  ;
  int p4 = (0+1)*WD  ;
  int p5 = (0-1)*WD+1;
  int p6 = (0+1)*WD-1;
  int p7 = (0  )*WD-1;
  int p8 = (0  )*WD+1;
  for (j=0; j<WD; j++) {
    int d1 = df(p[p1]&MASK,p[p2]&MASK)
           + df(p[p3]&MASK,p[p4]&MASK)
           + df(p[p5]&MASK,p[p6]&MASK)
           + df(p[p7]&MASK,p[p8]&MASK);
    /* 0 < d1(42) < 256*2*4 */
    r[p0] = d1 < EDGEDET ? 0 : PIXMAX;
    p0++; p1++; p2++; p3++; p4++; p5++; p6++; p7++; p8++;
  }
}
```

```
//EMAX4A start .emax_start_edge:
//EMAX4A ctl map_dist=1
//EMAX4A @0,0 while (ri+=,-1) rgi[320,]
//EMAX4A @0,1 add  (ri+=,4),r10     rgi[.emax_rgi_p____edge:,]
//EMAX4A @1,0                             & ld (r10,-1276),r5   lmr[.emax_lmrla_PREV_edge:,0,0,0,0,.emax_lmrma_PREV_edge:,320]
//EMAX4A @1,1                             & ld (r10,-1280),r3
//EMAX4A @1,2                             & ld (r10,-1284),r1
//EMAX4A @2,0                             & ld (r10,    4),r8   lmr[.emax_lmrla_CURR_edge:,0,0,0,0,.emax_lmrma_CURR_edge:,320]
//EMAX4A @2,2                             & ld (r10,   -4),r7
//EMAX4A @3,0 msad  (r7,r8),r7            & ld (r10, 1284),r2   lmr[.emax_lmrla_NEXT_edge:,0,0,0,0,.emax_lmrma_NEXT_edge:,320]
//EMAX4A @3,1                             & ld (r10, 1280),r4
//EMAX4A @3,2                             & ld (r10, 1276),r6
//EMAX4A @4,0 msad  (r1,r2),r1
//EMAX4A @4,1 msad  (r3,r4),r3
//EMAX4A @4,2 msad  (r5,r6),r5
//EMAX4A @5,0 mauh  (r1,r3),r1
//EMAX4A @5,1 mauh  (r5,r7),r5
//EMAX4A @6,0 mauh  (r1,r5) | suml (-),r1
//EMAX4A @7,0 mcas  (r1,ri) rgi[,64]      & stb -,(ri+=,1)   rgi[.emax_rgi_store_edge:,] lmw[.emax_lmwla_store_edge:,0,0,0,0,.emax_lmwma_store_edge:,80]
//EMAX4A end .emax_end_edge:
```

3.4.9 Stereo with SIMD and stencil

```
void wdifline(u1, u2, d, w)
     unsigned int *u1, *u2, *d;
     int w;
{
  int j;

  for (j=0; j<w; j++) { /* one scan-line */
    *d += wdif(WIN*2,u1,u2);
    u1++;
    u2++;
    d++;
  }
}
wdif(w, lp, rp)
     unsigned int w, *lp, *rp;
{
  int j, retval = 0;
  for (j=0; j<w; j++)
    retval += df((*(lp+j))&MASK, (*(rp+j))&MASK);
  return(retval);
}
```

```
//EMAX4A start .emax_start_wdifline:
//EMAX4A ctl map_dist=0
//EMAX4A @0,0 while (ri+=,-1) rgi[320,]
//EMAX4A @0,1 add  (ri+=,4),r0     rgi[.emax_rgiu1_wdifline:,]
//EMAX4A @1,0                     ld (r0, -4),r3      & ld (r0,  0),r2   lmr[.emax_lmrlau1_wdifline:,0,0,0,0,.emax_lmrmau1_wdifline:,320]
//EMAX4A @1,1                     ld (r0,-12),r5      & ld (r0, -8),r4
//EMAX4A @1,2                     ld (r0,-20),r7      & ld (r0,-16),r6
//EMAX4A @1,3                     ld (r0,-28),r9      & ld (r0,-24),r8
//EMAX4A @2,0 add  (ri+=,4),r1     rgi[.emax_rgiu2_wdifline:,]
//EMAX4A @3,0                     ld (r1, -4),r13     & ld (r1,  0),r12  lmr[.emax_lmrlau2_wdifline:,0,0,0,0,.emax_lmrmau2_wdifline:,320]
//EMAX4A @3,1                     ld (r1,-12),r15     & ld (r1, -8),r14
//EMAX4A @3,2                     ld (r1,-20),r17     & ld (r1,-16),r16
//EMAX4A @3,3                     ld (r1,-28),r19     & ld (r1,-24),r18
//EMAX4A @4,0 msad  (r2,r12),r2
//EMAX4A @4,1 msad  (r4,r14),r4
//EMAX4A @4,2 msad  (r6,r16),r6
//EMAX4A @4,3 msad  (r8,r18),r8
//EMAX4A @5,0 msad  (r3,r13),r3
//EMAX4A @5,1 msad  (r5,r15),r5
//EMAX4A @5,2 msad  (r7,r17),r7
//EMAX4A @5,3 msad  (r9,r19),r9
//EMAX4A @6,0 mauh3 (r2,r3,r4),r2 & ld (ri+=,4),r0 rgi[.emax_rgiw0_wdifline:,] lmf[.emax_lmfla_store_wdifline:,0,0,0,0,.emax_lmfma_store_wdifline:,320]
//EMAX4A @6,1 mauh3 (r5,r6,r7),r5
//EMAX4A @6,2 mauh  (r8,r9),   r8
//EMAX4A @7,0 mauh3 (r2,r5,r8) | suml (-),r1
//EMAX4A @8,0 add   (r0,r1)       & st -,(ri+=,4)  rgi[.emax_rgiw1_wdifline:,] lmw[.emax_lmwla_store_wdifline:,0,0,0,0,.emax_lmwma_store_wdifline:,320]
//EMAX4A end .emax_end_wdifline:
```

3.5 Examples (3D-floating-point)

See proj-arm32/sample/stencil-pipe/stencil-emax4.S. Many stencil kernels for floating-point processing are implemented.

3.5.1 Grapes with SIMD and stencil

```
void grapes( b, a, c )
     float *b, *a, *c;
{
  int i;
  float t0, t1;

  for (i=0; i<WD; i++) {
    *(c+i) = *(b+HT*WD  +WD  +i) * *(a+DP*HT*WD*9+HT*WD  +WD  +i)
           + *(b+HT*WD     +1+i) * *(a+DP*HT*WD*8+HT*WD     +1+i)
           + *(b+HT*WD       +i) * *(a+DP*HT*WD*7+HT*WD       +i)
           + *(b+HT*WD     -1+i) * *(a+DP*HT*WD*6+HT*WD     -1+i)
           + *(b+HT*WD  -WD  +i) * *(a+DP*HT*WD*5+HT*WD  -WD  +i)
           + *(b        +WD+1+i) * *(a+DP*HT*WD*4        +WD+1+i)
           + *(b        +WD  +i) * *(a+DP*HT*WD*3        +WD  +i)
           + *(b        +WD-1+i) * *(a+DP*HT*WD*2        +WD-1+i)
           + *(b           +1+i) * *(a+DP*HT*WD*1           +1+i)
           + *(b             +i)
           + *(b           -1+i) * *(a-DP*HT*WD*1           -1+i)
           + *(b        -WD+1+i) * *(a-DP*HT*WD*2        -WD+1+i)
           + *(b        -WD  +i) * *(a-DP*HT*WD*3        -WD  +i)
           + *(b        -WD-1+i) * *(a-DP*HT*WD*4        -WD-1+i)
           + *(b-HT*WD  +WD  +i) * *(a-DP*HT*WD*5-HT*WD  +WD  +i)
           + *(b-HT*WD     +1+i) * *(a-DP*HT*WD*6-HT*WD     +1+i)
           + *(b-HT*WD       +i) * *(a-DP*HT*WD*7-HT*WD       +i)
           + *(b-HT*WD     -1+i) * *(a-DP*HT*WD*8-HT*WD     -1+i)
           + *(b-HT*WD  -WD  +i) * *(a-DP*HT*WD*9-HT*WD  -WD  +i);
  }
}
```

```
//EMAX4A start .emax_start_grapes:
//EMAX4A ctl map_dist=1
//EMAX4A @0,0  while (ri+=,-1) rgi[320,]
//EMAX4A @0,1  add (ri+=,4),r9       rgi[.emax_rgi_p0____grapes:,]
//EMAX4A @1,0                                       & ld (r9,-1276),r19 lmr[.emax_lmrla_PREV_B0_grapes:,0,0,0,0,.emax_lmrma_PREV_B0_grapes:,320]
//EMAX4A @1,1                                       & ld (r9,-1280),r20
//EMAX4A @1,2                                       & ld (ri+=,4),r0    rgi[.emax_rgi_A0_grapes:,]    lmr[.emax_lmrla_A0_grapes:,0,0,0,0,.emax_lmrma_A0_grapes:,320]
//EMAX4A @2,0                                       & ld (r9,4),r23     lmr[.emax_lmrla_CURR_B0_grapes:,0,0,0,0,.emax_lmrma_CURR_B0_grapes:,320]
//EMAX4A @2,1                    ld (r9,-4),r21     & ld (r9,0),r22
//EMAX4A @2,2  fmul (r20,r0),r30                    & ld (ri+=,4),r1    rgi[.emax_rgi_A1_grapes:,]    lmr[.emax_lmrla_A1_grapes:,0,0,0,0,.emax_lmrma_A1_grapes:,320]
//EMAX4A @2,3                                       & ld (ri+=,4),r2    rgi[.emax_rgi_A2_grapes:,]    lmr[.emax_lmrla_A2_grapes:,0,0,0,0,.emax_lmrma_A2_grapes:,320]
//EMAX4A @3,0                                       & ld (r9,1284),r19  lmr[.emax_lmrla_NEXT_B0_grapes:,0,0,0,0,.emax_lmrma_NEXT_B0_grapes:,320]
//EMAX4A @3,1                                       & ld (r9,1280),r24
//EMAX4A @3,2  fma3 (r21,r1,r30),r30                & ld (ri+=,4),r3    rgi[.emax_rgi_A3_grapes:,]    lmr[.emax_lmrla_A3_grapes:,0,0,0,0,.emax_lmrma_A3_grapes:,320]
//EMAX4A @3,3  fmul (r22,r2),r31                    & ld (ri+=,4),r4    rgi[.emax_rgi_A4_grapes:,]    lmr[.emax_lmrla_A4_grapes:,0,0,0,0,.emax_lmrma_A4_grapes:,320]
//EMAX4A @4,0  add (ri+=,4),r9       rgi[.emax_rgi_p1____grapes:,]
//EMAX4A @4,2  fma3 (r23,r3,r30),r30                & ld (ri+=,4),r5    rgi[.emax_rgi_A5_grapes:,]    lmr[.emax_lmrla_A5_grapes:,0,0,0,0,.emax_lmrma_A5_grapes:,320]
//EMAX4A @4,3  fma3 (r24,r4,r31),r31                & ld (ri+=,4),r6    rgi[.emax_rgi_A6_grapes:,]    lmr[.emax_lmrla_A6_grapes:,0,0,0,0,.emax_lmrma_A6_grapes:,320]
//EMAX4A @5,0                                       & ld (r9,-1276),r27 lmr[.emax_lmrla_PREV_B1_grapes:,0,0,0,0,.emax_lmrma_PREV_B1_grapes:,320]
//EMAX4A @5,1                    ld (r9,-1284),r25  & ld (r9,-1280),r26
//EMAX4A @5,2                                       & ld (ri+=,4),r7    rgi[.emax_rgi_A7_grapes:,]    lmr[.emax_lmrla_A7_grapes:,0,0,0,0,.emax_lmrma_A7_grapes:,320]
//EMAX4A @5,3                                       & ld (ri+=,4),r8    rgi[.emax_rgi_A8_grapes:,]    lmr[.emax_lmrla_A8_grapes:,0,0,0,0,.emax_lmrma_A8_grapes:,320]
//EMAX4A @6,0                                       & ld (r9,4),r20     lmr[.emax_lmrla_CURR_B1_grapes:,0,0,0,0,.emax_lmrma_CURR_B1_grapes:,320]
//EMAX4A @6,1                    ld (r9,-4),r28     & ld (r9,0),r29
//EMAX4A @6,2  fma3 (r25,r5,r30),r30                & ld (ri+=,4),r10   rgi[.emax_rgi_A10_grapes:,]   lmr[.emax_lmrla_A10_grapes:,0,0,0,0,.emax_lmrma_A10_grapes:,320]
//EMAX4A @6,3  fma3 (r26,r6,r31),r31                & ld (ri+=,4),r11   rgi[.emax_rgi_A11_grapes:,]   lmr[.emax_lmrla_A11_grapes:,0,0,0,0,.emax_lmrma_A11_grapes:,320]
//EMAX4A @7,0                                       & ld (r9,1284),r23  lmr[.emax_lmrla_NEXT_B1_grapes:,0,0,0,0,.emax_lmrma_NEXT_B1_grapes:,320]
//EMAX4A @7,1                    ld (r9,1276),r21   & ld (r9,1280),r22
//EMAX4A @7,2  fma3 (r27,r7,r30),r30                & ld (ri+=,4),r12   rgi[.emax_rgi_A12_grapes:,]   lmr[.emax_lmrla_A12_grapes:,0,0,0,0,.emax_lmrma_A12_grapes:,320]
//EMAX4A @7,3  fma3 (r28,r8,r31),r31                & ld (ri+=,4),r13   rgi[.emax_rgi_A13_grapes:,]   lmr[.emax_lmrla_A13_grapes:,0,0,0,0,.emax_lmrma_A13_grapes:,320]
//EMAX4A @8,1  fadd (r29,r30),r30                   &
//EMAX4A @8,2  fma3 (r20,r10,r31),r31               &
//EMAX4A @8,3  add (ri+=,4),r9       rgi[.emax_rgi_p2____grapes:,]
//EMAX4A @9,1  fma3 (r21,r11,r30),r30               & ld (ri+=,4),r14   rgi[.emax_rgi_A14_grapes:,]   lmr[.emax_lmrla_A14_grapes:,0,0,0,0,.emax_lmrma_A14_grapes:,320]
//EMAX4A @9,2  fma3 (r23,r13,r31),r31               & ld (r9,-1276),r19 lmr[.emax_lmrla_PREV_B2_grapes:,0,0,0,0,.emax_lmrma_PREV_B2_grapes:,320]
//EMAX4A @9,3                                       & ld (r9,-1280),r24
//EMAX4A @10,0                                      & ld (ri+=,4),r15   rgi[.emax_rgi_A15_grapes:,]   lmr[.emax_lmrla_A15_grapes:,0,0,0,0,.emax_lmrma_A15_grapes:,320]
//EMAX4A @10,1 fma3 (r22,r12,r30),r30               & ld (ri+=,4),r16   rgi[.emax_rgi_A16_grapes:,]   lmr[.emax_lmrla_A16_grapes:,0,0,0,0,.emax_lmrma_A16_grapes:,320]
//EMAX4A @10,2 fma3 (r24,r14,r31),r31               & ld (r9,4),r27     lmr[.emax_lmrla_CURR_B2_grapes:,0,0,0,0,.emax_lmrma_CURR_B2_grapes:,320]
//EMAX4A @10,3                   ld (r9,-4),r25     & ld (r9,0),r26
//EMAX4A @11,0 fma3 (r25,r15,r31),r31               & ld (ri+=,4),r17   rgi[.emax_rgi_A17_grapes:,]   lmr[.emax_lmrla_A17_grapes:,0,0,0,0,.emax_lmrma_A17_grapes:,320]
//EMAX4A @11,1                                      & ld (ri+=,4),r18   rgi[.emax_rgi_A18_grapes:,]   lmr[.emax_lmrla_A18_grapes:,0,0,0,0,.emax_lmrma_A18_grapes:,320]
//EMAX4A @11,2                                      & ld (r9,1284),r19  lmr[.emax_lmrla_NEXT_B2_grapes:,0,0,0,0,.emax_lmrma_NEXT_B2_grapes:,320]
//EMAX4A @11,3                                      & ld (r9,1280),r28
//EMAX4A @12,0 fma3 (r27,r17,r31),r31
//EMAX4A @12,3 fma3 (r26,r16,r30),r30
//EMAX4A @13,2 fadd (r30,r31),r31
//EMAX4A @13,3 fmul (r28,r18),r19
//EMAX4A @14,2 fadd (r31,r19),r19
//EMAX4A @15,2                                 & st r19,(ri+=,4)   rgi[.emax_rgi_store_grapes:,] lmw[.emax_lmwla_store_grapes:,0,0,0,0,.emax_lmwma_store_grapes:,320]
//EMAX4A end .emax_end_grapes:
```

3.5.2 Jacobi with SIMD and stencil

```
void jacobi( B, C )
  float *B, *C;
{
    int x;
    float C1 = 0.2;
    float C2 = 0.3;

    for (x=0; x<WD; x++) {
      *(C+x) = C2 * (*(B-HT*WD+x) + *(B-WD+x) + *(B-1+x) + *(B+1+x) + *(B+WD+x) + *(B+HT*WD+x))
             + C1 * *(B+x);
    }
}
```

```
//EMAX4A start .emax_start_jacobe:
//EMAX4A ctl map_dist=1
//EMAX4A @0,0 while (ri+=,-1) rgi[320,]
//EMAX4A @0,1 add (ri+=,4),r0      rgi[.emax_rja:,] & ld (ri+=,4),r1   rgi[.emax_rgi_CURR_A0_ja:,] lmr[.emax_lmrla_CURR_A0_ja:,0,0,0,0,.emax_lmrma_CURR_A0_ja:,320]
//EMAX4A @1,1 fmul (ri,r1),r10     rgi[0x3e99999a,] & ld (r0,-1280),r2 lmr[.emax_lmrla_PREV_A1_ja:,0,0,0,0,.emax_lmrma_PREV_A1_ja:,320]
//EMAX4A @2,1 fma3 (ri,r2,r10),r10 rgi[0x3e99999a,] & ld (r0,4),r5     lmr[.emax_lmrla_CURR_A1_ja:,0,0,0,0,.emax_lmrma_CURR_A1_ja:,320]
//EMAX4A @2,2                                       & ld (r0,0),r4
//EMAX4A @2,3                                       & ld (r0,-4),r3
//EMAX4A @3,1 fma3 (ri,r5,r10),r10 rgi[0x3e99999a,] & ld (r0,1280),r6  lmr[.emax_lmrla_NEXT_A1_ja:,0,0,0,0,.emax_lmrma_NEXT_A1_ja:,320]
//EMAX4A @3,2 fmul (ri,r4),r11     rgi[0x3e4ccccd,] & ld (ri+=,4),r7   rgi[.emax_rgi_CURR_A2_ja:,] lmr[.emax_lmrla_CURR_A2_ja:,0,0,0,0,.emax_lmrma_CURR_A2_ja:,320]
//EMAX4A @3,3 fmul (ri,r3),r12     rgi[0x3e99999a,] &
//EMAX4A @4,1 fma3 (ri,r6,r10),r10 rgi[0x3e99999a,] &
//EMAX4A @4,2 fma3 (ri,r7,r11),r11 rgi[0x3e99999a,] &
//EMAX4A @5,1 fadd (r10,r11),r10                    &
//EMAX4A @6,1 fadd (r10,r12),r10                    &
//EMAX4A @7,1                                       & st r10,(ri+=,4)  rgi[.emax_rgi_store_ja:,] lmw[.emax_lmwla_store_ja:,0,0,0,0,.emax_lmwma_store_ja:,320]
//EMAX4A end .emax_end_jacobe:
```

3.5.3 Fd6 with SIMD and stencil

```
void fd6( B, C )
  float *B, *C;
{
    int x;
    float t0, t1, t2;
    float C1 = 0.1;
    float C2 = 0.2;
    float C3 = 0.3;
    float C4 = 0.4;

    for (x = 0; x < WD; x++) {
      *(C+x) = C4 * (*(B-3*HT*WD+x) + *(B-3*WD+x) + *(B-3+x)
                   + *(B+3+x)       + *(B+3*WD+x) + *(B+3*HT*WD+x))
             + C3 * (*(B-2*HT*WD+x) + *(B-2*WD+x) + *(B-2+x)
                   + *(B+2+x)       + *(B+2*WD+x) + *(B+2*HT*WD+x))
             + C2 * (*(B-1*HT*WD+x) + *(B-1*WD+x) + *(B-1+x)
                   + *(B+1+x)       + *(B+1*WD+x) + *(B+1*HT*WD+x))
             + C1 * *(B+x);
    }
}
```

```
//EMAX4A start .emax_start_fd6:
//EMAX4A ctl map_dist=1
//EMAX4A @0,0  while (ri+=,-1) rgi[320,]                    & ld (ri+=,4),r0    rgi[.emax_rgi_C_A0_fd6:,] lmr[.emax_lmrla_C_A0_fd6:,0,0,0,0,.emax_lmrma_C_A0_fd6:,320]
//EMAX4A @0,1  add (ri+=,4),r10     rgi[.emax_rgi_p0_fd6:,] & ld (ri+=,4),r1    rgi[.emax_rgi_C_A1_fd6:,] lmr[.emax_lmrla_C_A1_fd6:,0,0,0,0,.emax_lmrma_C_A1_fd6:,320]
//EMAX4A @0,2                                               & ld (ri+=,4),r2    rgi[.emax_rgi_C_A2_fd6:,] lmr[.emax_lmrla_C_A2_fd6:,0,0,0,0,.emax_lmrma_C_A2_fd6:,320]
//EMAX4A @1,0  fmul(ri,r0),r20      rgi[0x3ecccccd,]        & ld (r10,-3840),r3 lmr[.emax_lmrla_P3_A3_fd6:,0,0,0,0,.emax_lmrma_P3_A3_fd6:,320]
//EMAX4A @1,1  fmul(ri,r1),r21      rgi[0x3e99999a,]        &
//EMAX4A @1,2  fmul(ri,r2),r22      rgi[0x3e4ccccd,]        &
//EMAX4A @2,0  fma3(ri,r3,r20),r20  rgi[0x3ecccccd,]        & ld (r10,-2560),r4 lmr[.emax_lmrla_P2_A3_fd6:,0,0,0,0,.emax_lmrma_P2_A3_fd6:,320]
//EMAX4A @2,1  fadd(r21,r22),r21                            &
//EMAX4A @3,0  fma3(ri,r4,r20),r20  rgi[0x3e99999a,]        & ld (r10,-1280),r5 lmr[.emax_lmrla_P1_A3_fd6:,0,0,0,0,.emax_lmrma_P1_A3_fd6:,320]
//EMAX4A @3,1                                               &
//EMAX4A @4,0  add (ri+=,4),r10     rgi[.emax_rgi_p1_fd6:,] & ld (r10,12),r12   lmr[.emax_lmrla_C_A3_fd6:,0,0,0,0,.emax_lmrma_C_A3_fd6:,320]
//EMAX4A @4,1                                 ld (r10,4),r10  & ld (r10,8),r11
//EMAX4A @4,2                                 ld (r10,-4),r8  & ld (r10,0),r9
//EMAX4A @4,3                                 ld (r10,-12),r6 & ld (r10,-8),r7
//EMAX4A @5,0  fma3(ri,r5,r20),r20  rgi[0x3e4ccccd,]    & ld (r10,1280),r13  lmr[.emax_lmrla_NEXT1_A3_fd6:,0,0,0,0,.emax_lmrma_NEXT1_A3_fd6:,320]
//EMAX4A @5,1  fma3(ri,r10,r21),r21 rgi[0x3e4ccccd,]    &
//EMAX4A @5,2  fmul(ri,r8),r22      rgi[0x3e4ccccd,]    &
//EMAX4A @5,3  fmul(ri,r6),r23      rgi[0x3ecccccd,]    &
//EMAX4A @6,0  fma3(ri,r12,r20),r20 rgi[0x3ecccccd,]    & ld (r10,2560),r14  lmr[.emax_lmrla_NEXT2_A3_fd6:,0,0,0,0,.emax_lmrma_NEXT2_A3_fd6:,320]
//EMAX4A @6,1  fma3(ri,r11,r21),r21 rgi[0x3e99999a,]    &
//EMAX4A @6,2  fma3(ri,r9,r22),r22  rgi[0x3dcccccd,]    &
//EMAX4A @6,3  fma3(ri,r7,r23),r23  rgi[0x3e99999a,]    &
//EMAX4A @7,0                                           & ld (r10,3840),r15  lmr[.emax_lmrla_NEXT3_A3_fd6:,0,0,0,0,.emax_lmrma_NEXT3_A3_fd6:,320]
//EMAX4A @7,1                                           & ld (ri+=,4),r16    rgi[.emax_rgi_C_A4_fd6:,] lmr[.emax_lmrla_C_A4_fd6:,0,0,0,0,.emax_lmrma_C_A4_fd6:,320]
//EMAX4A @7,2  fma3(ri,r13,r22),r22 rgi[0x3e4ccccd,]    & ld (ri+=,4),r17    rgi[.emax_rgi_C_A5_fd6:,] lmr[.emax_lmrla_C_A5_fd6:,0,0,0,0,.emax_lmrma_C_A5_fd6:,320]
//EMAX4A @7,3  fma3(ri,r14,r23),r23 rgi[0x3e99999a,]    & ld (ri+=,4),r18    rgi[.emax_rgi_C_A6_fd6:,] lmr[.emax_lmrla_C_A6_fd6:,0,0,0,0,.emax_lmrma_C_A6_fd6:,320]
//EMAX4A @8,0  fma3(ri,r15,r20),r20 rgi[0x3ecccccd,]    &
//EMAX4A @8,1  fma3(ri,r16,r21),r21 rgi[0x3e4ccccd,]    &
//EMAX4A @8,2  fma3(ri,r17,r22),r22 rgi[0x3e99999a,]    &
//EMAX4A @8,3  fma3(ri,r18,r23),r23 rgi[0x3ecccccd,]    &
//EMAX4A @9,1  fadd(r20,r21),r21                        &
//EMAX4A @9,2  fadd(r22,r23),r22                        &
//EMAX4A @10,2 fadd(r21,r22),r22                        &
//EMAX4A @11,0                                          & st r22,(ri+=,4)    rgi[.emax_rgi_store_fd6:,] lmw[.emax_lmwla_store_fd6:,0,0,0,0,.emax_lmwma_store_fd6:,320]
//EMAX4A end .emax_end_fd6:
```

3.5.4 Resid with SIMD and stencil

```
void resid( B, C, D )
      float *B, *C, *D;
{
    int x;

    float A0 = 0.1;
    float A1 = 0.2;
    float A2 = 0.3;
    float A3 = 0.4;

    for (x=0; x<WD; x++) {
        *(D+x) = *(C+x)
                - A0 * *(B+x)
                - A1 * ( *(B-HT*WD     +x) + *(B+HT*WD   +x)
                       + *(B      -WD  +x) + *(B      +WD+x)
                       + *(B         -1+x) + *(B       +1+x) )
                - A2 * ( *(B-HT*WD-WD  +x) + *(B+HT*WD-WD+x) + *(B-HT*WD+WD+x) + *(B+HT*WD+WD+x)
                       + *(B      -WD-1+x) + *(B    +WD-1+x) + *(B    -WD+1+x) + *(B    +WD+1+x)
                       + *(B-HT*WD   -1+x) + *(B-HT*WD +1+x) + *(B+HT*WD -1+x) + *(B+HT*WD +1+x) )
                - A3 * ( *(B-HT*WD-WD-1+x) + *(B+HT*WD-WD-1+x)
                       + *(B-HT*WD+WD-1+x) + *(B-HT*WD-WD+1+x)
                       + *(B+HT*WD+WD-1+x) + *(B+HT*WD-WD+1+x)
                       + *(B-HT*WD+WD+1+x) + *(B+HT*WD+WD+1+x) );
    }
}
```

```
//EMAX4A start .emax_start_resid:
//EMAX4A ctl map_dist=1
//EMAX4A @0,0  while (ri+=,-1) rgi[320,]
//EMAX4A @0,1  add (ri+=,4),r0      rgi[.emax_rgi_p0____resid:,]  &
//EMAX4A @1,1                                         & ld (r0,-1276),r3  lmr[.emax_lmrla_P_B0_resid:,0,0,0,0,.emax_lmrma_P_B0_resid:,320]
//EMAX4A @1,2                                         & ld (r0,-1280),r2
//EMAX4A @1,3                                         & ld (r0,-1284),r1
//EMAX4A @2,1  fmul (ri,r3),r29     rgi[0xbecccccd,]  & ld (r0,4),r6      lmr[.emax_lmrla_C_B0_resid:,0,0,0,0,.emax_lmrma_C_B0_resid:,320]
//EMAX4A @2,2  fmul (ri,r2),r30     rgi[0xbe99999a,]  & ld (r0,0),r5
//EMAX4A @2,3  fmul (ri,r1),r31     rgi[0xbecccccd,]  & ld (r0,-4),r4
//EMAX4A @3,0  add (ri+=,4),r0      rgi[.emax_rgi_p1____resid:,]  &
//EMAX4A @3,1  fma3 (ri,r6,r29),r29 rgi[0xbe99999a,]  & ld (r0,1284),r9   lmr[.emax_lmrla_NEXT_B0_resid:,0,0,0,0,.emax_lmrma_NEXT_B0_resid:,320]
//EMAX4A @3,2  fma3 (ri,r5,r30),r30 rgi[0xbe4ccccd,]  & ld (r0,1280),r8
//EMAX4A @3,3  fma3 (ri,r4,r31),r31 rgi[0xbe99999a,]  & ld (r0,1276),r7
//EMAX4A @4,1  fma3 (ri,r9,r29),r29 rgi[0xbecccccd,]  & ld (r0,-1276),r12 lmr[.emax_lmrla_P_B1_resid:,0,0,0,0,.emax_lmrma_P_B1_resid:,320]
//EMAX4A @4,2  fma3 (ri,r8,r30),r30 rgi[0xbe99999a,]  & ld (r0,-1280),r11
//EMAX4A @4,3  fma3 (ri,r7,r31),r31 rgi[0xbecccccd,]  & ld (r0,-1284),r10
//EMAX4A @5,1  fma3 (ri,r12,r29),r29 rgi[0xbe99999a,] & ld (r0,4),r15     lmr[.emax_lmrla_C_B1_resid:,0,0,0,0,.emax_lmrma_C_B1_resid:,320]
//EMAX4A @5,2  fma3 (ri,r11,r30),r30 rgi[0xbe4ccccd,] & ld (r0,0),r14
//EMAX4A @5,3  fma3 (ri,r10,r31),r31 rgi[0xbe99999a,] & ld (r0,-4),r13
//EMAX4A @6,0  add (ri+=,4),r0      rgi[.emax_rgi_p2____resid:,]  &
//EMAX4A @6,1  fma3 (ri,r15,r29),r29 rgi[0xbe4ccccd,] & ld (r0,1284),r18  lmr[.emax_lmrla_NEXT_B1_resid:,0,0,0,0,.emax_lmrma_NEXT_B1_resid:,320]
//EMAX4A @6,2  fma3 (ri,r14,r30),r30 rgi[0xbdcccccd,] & ld (r0,1280),r17
//EMAX4A @6,3  fma3 (ri,r13,r31),r31 rgi[0xbe4ccccd,] & ld (r0,1276),r16
//EMAX4A @7,1  fma3 (ri,r18,r29),r29 rgi[0xbe99999a,] & ld (r0,-1276),r21 lmr[.emax_lmrla_P_B2_resid:,0,0,0,0,.emax_lmrma_P_B2_resid:,320]
//EMAX4A @7,2  fma3 (ri,r17,r30),r30 rgi[0xbe4ccccd,] & ld (r0,-1280),r20
//EMAX4A @7,3  fma3 (ri,r16,r31),r31 rgi[0xbe99999a,] & ld (r0,-1284),r19
//EMAX4A @8,1  fma3 (ri,r21,r29),r29 rgi[0xbecccccd,] & ld (r0,4),r24     lmr[.emax_lmrla_C_B2_resid:,0,0,0,0,.emax_lmrma_C_B2_resid:,320]
//EMAX4A @8,2  fma3 (ri,r20,r30),r30 rgi[0xbe99999a,] & ld (r0,0),r23
//EMAX4A @8,3  fma3 (ri,r19,r31),r31 rgi[0xbecccccd,] & ld (r0,-4),r22
//EMAX4A @9,1  fma3 (ri,r24,r29),r29 rgi[0xbe99999a,] & ld (r0,1284),r27  lmr[.emax_lmrla_NEXT_B2_resid:,0,0,0,0,.emax_lmrma_NEXT_B2_resid:,320]
//EMAX4A @9,2  fma3 (ri,r23,r30),r30 rgi[0xbe4ccccd,] & ld (r0,1280),r26
//EMAX4A @9,3  fma3 (ri,r22,r31),r31 rgi[0xbe99999a,] & ld (r0,1276),r25
//EMAX4A @10,1 fma3 (ri,r27,r29),r29 rgi[0xbecccccd,] & ld (ri+=,4),r28   rgi[.emax_rgi_C_resid:,] lmr[.emax_lmrla_C_resid:,0,0,0,0,.emax_lmrma_C_resid:,320]
//EMAX4A @10,2 fma3 (ri,r26,r30),r30 rgi[0xbe99999a,] &
//EMAX4A @10,3 fma3 (ri,r25,r31),r31 rgi[0xbecccccd,] &
//EMAX4A @11,1 fadd (r29,r28),r28                    &
//EMAX4A @11,2 fadd (r30,r31),r31                    &
//EMAX4A @12,1 fadd (r31,r28),r28                    &
//EMAX4A @13,3                                        & st r28,(ri+=,4)   rgi[.emax_rgi_st_resid:,] lmw[.emax_lmwla_st_resid:,0,0,0,0,.emax_lmwma_st_resid:,320]
//EMAX4A end .emax_end_resid:
```

3.5.5 Wave2d with SIMD and stencil

```
void wave2d( Z0, Z1, Z2 )
  float *Z0, *Z1, *Z2;
{
    int x;

    float value = 0.0025;

    for (x=0; x<WD; x++) {
      *(Z2+x) = 2.0 * *(Z1+x)
        - *(Z0+x)
        + value * ( *(Z1+WD+x) + *(Z1-WD+x) + *(Z1-1+x) + *(Z1+1+x) - 4.0 * *(Z1+x) );
    }
}
```

```
//EMAX4A start .emax_start_wave2d:
//EMAX4A ctl map_dist=1
//EMAX4A @0,0 while (ri+=,-1) rgi[320,]
//EMAX4A @0,1 add (ri+=,4),r0       rgi[.emax_rgi_p0:,] & ld (ri+=,4),r10 rgi[.emax_rgi_Z0:,] lmr[.emax_lmrla_Z0:,0,0,0,0,.emax_lmrma_Z0:,320]
//EMAX4A @1,1 fmul (ri,r10),r31     rgi[0xbf800000,]    & ld (r0,-1280),r1  lmr[.emax_lmrla_PREV_Z1:,0,0,0,0,.emax_lmrma_PREV_Z1:,320]
//EMAX4A @2,1 fma3 (ri,r1,r31),r31  rgi[0x3b23d70a,]    & ld (r0,4),r4      lmr[.emax_lmrla_CURR_Z1:,0,0,0,0,.emax_lmrma_CURR_Z1:,320]
//EMAX4A @2,2                                           & ld (r0,0),r3
//EMAX4A @2,3                                           & ld (r0,-4),r2
//EMAX4A @3,1 fma3 (ri,r4,r31),r31  rgi[0x3b23d70a,]    & ld (r0,1280),r5   lmr[.emax_lmrla_NEXT_Z1:,0,0,0,0,.emax_lmrma_NEXT_Z1:,320]
//EMAX4A @3,2 fmul (ri,r3),r30      rgi[0x3b23d70a,]    &
//EMAX4A @3,3 fmul (ri,r2),r29      rgi[0x3b23d70a,]    &
//EMAX4A @4,1 fma3 (ri,r5,r31),r31  rgi[0x3b23d70a,]    &
//EMAX4A @4,2 fmul (ri,r3),r29      rgi[0x40000000,]    &
//EMAX4A @4,3 fma3 (ri,r30,r29),r30 rgi[0xc0800000,]    &
//EMAX4A @5,1 fadd (r31,r29),r31                        &
//EMAX4A @6,1 fadd (r31,r30),r31                        &
//EMAX4A @7,1                                           & st r31,(ri+=,4)   rgi[.emax_rgi_store:,] lmw[.emax_lmwla_store:,0,0,0,0,.emax_lmwma_store:,320]
//EMAX4A end .emax_end_wave2d:
```

3.6 Examples (4D-imaging)

See proj-arm32/sample/4dimage/gather-emax4.S and gdepth-emax4.S. 4D-imaging with CGRA is implemented.

3.6.1 Gather with SIMD and stencil

```
gather_x1(int yin, int yout)
{
  int x, dx, dy, w, pix;
  int cvalR, cvalG, cvalB;

  for (x=36; x<FWD-36; x++) {
\#ifdef PRECISE_SCALE
    int image_center = yin+((x/coresize*WINSIZE+(offset*(coresize-(x%coresize)))/coresize+shift_x)*1021/1024);
\#else
    int image_center = (x>>4)*WINSIZE + (((~x&15)*offset)>>4) + shift_x + yin;
\#endif
    cvalR=0;
    cvalG=0;
    cvalB=0;
    for (dy=-1; dy<=1; dy++) {
      for (dx=-1; dx<=1; dx++) {
        Uint pix = ACCI[image_center+smallwin_offset_y*dy+smallwin_offset_x*dx];
        w = weight[WBASE+dy*MAXDELTA*2+dx];
        cvalR += ((pix>>24)&255)*w;
        cvalG += ((pix>>16)&255)*w;
        cvalB += ((pix>> 8)&255)*w;
      }
    }
    ACCO[(yout+x)] = ((cvalR>>8)<<24) | ((cvalG>>8)<<16) | ((cvalB>>8)<<8);
  }
```

```
//EMAX4A start .start_gather_x1:
//EMAX4A ctl map_dist=0
//EMAX4A @0,0  while (ri+=,-1)             rgi[1528,]
//EMAX4A @0,1  add (ri+=,1),r0             rgi[35,]
//EMAX4A @1,0  sub (-1,r0)|and (-,15),r1
//EMAX4A @1,1  |or (r0,0)>>4,r2
//EMAX4A @2,0  mluh (r1,ri)|or (-,0)>>4,r3  rgi[,.x1_offset:]
//EMAX4A @2,1  mluh (r2,ri),r4             rgi[,75]
//EMAX4A @2,2  add (ri,ri),r5              rgi[.x1_shift_x:,.x1_yin:]
//EMAX4A @3,0  add3 (r3,r4,r5)|or (-,0)<<2,r7
//EMAX4A @4,0                                   & ld (r7,ri),r10    rgi[,.x1_ym1_xm1:] lmr[.x1_la_acci_ym1_0:,0,0,0,0,.x1_ma_acci_ym1_0:,7240]
//EMAX4A @4,1                                   & ld (r7,ri),r11    rgi[,.x1_ym1_xe0:] lmr[.x1_la_acci_ym1_1:,0,0,0,0,.x1_ma_acci_ym1_1:,7240]
//EMAX4A @4,2                                   & ld (r7,ri),r12    rgi[,.x1_ym1_xp1:] lmr[.x1_la_acci_ym1_2:,0,0,0,0,.x1_ma_acci_ym1_2:,7240]
//EMAX4A @5,0  mluh (r10.l,ri),r20   rgi[,16]   & ld (r7,ri),r13    rgi[,.x1_ye0_xm1:] lmr[.x1_la_acci_ye0_0:,0,0,0,0,.x1_ma_acci_ye0_0:,7240]
//EMAX4A @5,1  mluh (r11.l,ri),r22   rgi[,32]   & ld (r7,ri),r14    rgi[,.x1_ye0_xe0:] lmr[.x1_la_acci_ye0_1:,0,0,0,0,.x1_ma_acci_ye0_1:,7240]
//EMAX4A @5,2  mluh (r10.h,ri),r21   rgi[,16]   & ld (r7,ri),r15    rgi[,.x1_ye0_xp1:] lmr[.x1_la_acci_ye0_2:,0,0,0,0,.x1_ma_acci_ye0_2:,7240]
//EMAX4A @5,3  mluh (r11.h,ri),r23   rgi[,32]
//EMAX4A @6,0  mluh (r12.l,ri),r30   rgi[,16]
//EMAX4A @6,1  mluh (r13.l,ri),r24   rgi[,32]
//EMAX4A @6,2  mluh (r12.h,ri),r31   rgi[,16]
//EMAX4A @6,3  mluh (r13.h,ri),r25   rgi[,32]
//EMAX4A @7,0  mauh3 (r20,r22,r30),r30
//EMAX4A @7,1  mluh (r14.l,ri),r26   rgi[,64]
//EMAX4A @7,2  mauh3 (r21,r23,r31),r31
//EMAX4A @7,3  mluh (r14.h,ri),r27   rgi[,64]
//EMAX4A @8,0  mluh (r15.l,ri),r20   rgi[,32]   & ld (r7,ri),r16    rgi[,.x1_yp1_xm1:] lmr[.x1_la_acci_yp1_0:,0,0,0,0,.x1_ma_acci_yp1_0:,7240]
//EMAX4A @8,1  mauh3 (r24,r26,r30),r30          & ld (r7,ri),r17    rgi[,.x1_yp1_xe0:] lmr[.x1_la_acci_yp1_1:,0,0,0,0,.x1_ma_acci_yp1_1:,7240]
//EMAX4A @8,2  mluh (r15.h,ri),r21   rgi[,32]   & ld (r7,ri),r18    rgi[,.x1_yp1_xp1:] lmr[.x1_la_acci_yp1_2:,0,0,0,0,.x1_ma_acci_yp1_2:,7240]
//EMAX4A @8,3  mauh3 (r25,r27,r31),r31
//EMAX4A @9,0  mluh (r16.l,ri),r22   rgi[,16]
//EMAX4A @9,2  mluh (r16.h,ri),r23   rgi[,16]
//EMAX4A @10,0 mauh3 (r20,r22,r30),r30
//EMAX4A @10,1 mluh (r17.l,ri),r20   rgi[,32]
//EMAX4A @10,2 mauh3 (r21,r23,r31),r31
//EMAX4A @10,3 mluh (r17.h,ri),r21   rgi[,32]
//EMAX4A @11,0 mluh (r18.l,ri),r22   rgi[,16]
//EMAX4A @11,2 mluh (r18.h,ri),r23   rgi[,16]
//EMAX4A @12,0 mauh3 (r20,r22,r30)|or (-,0)>M8,r30
//EMAX4A @12,2 mauh3 (r21,r23,r31)|or (-,0)>M8,r31
//EMAX4A @13,0 mh2bw (r31,r30)                  & st -,(ri+=,4)     rgi[.x1_acco:,] lmw[.x1_la_acco:,0,0,0,0,.x1_ma_acco:,1528]
//EMAX4A end .end_gather_x1:
```

3.6.2 Gdepth with SIMD and stencil

```
gdepth_x1(int yin, int yout)
{
  int i, j;
  int x, dx, dy;
  int cvalR, cvalG, cvalB;
  Uint pix00, pix01, pix02, pix03, pix04, pix05, pix10, pix11, pix20, pix21;

  for (x=36; x<FWD-36; x++) {
#ifdef PRECISE_SCALE
    int image_center = yin+((x/coresize*WINSIZE+(offset*(coresize-(x%coresize)))/coresize+shift_x)*1021/1024);
#else
    int image_center = (x>>4)*WINSIZE + (((~x&15)*offset)>>4) + shift_x + yin;
#endif
    Uint sad = 0;
    cvalR=0;
    cvalG=0;
    cvalB=0;
    for (dy=-1; dy<=1; dy++) {
      for (dx=-1; dx<=1; dx++) {
        if (dy == 0 && dx == 0)
          continue;
        for (i=-1; i<=1; i++) {
          for (j=-1; j<=1; j++) {
            if (j == 0)
              continue;
            pix00 = ACCI[image_center                     +(i*image_WD)                     +j];
            pix10 = ACCI[image_center+smallwin_offset_y*dy+(i*image_WD)+smallwin_offset_x*dx+j];
            sad += sad(pix00, pix10);
          }
        }
      }
    }
    if (SAD[yout+x] > (255*6*3*6)/200 && sad < SAD[yout+x]) {
      SAD[yout+x]  = sad;
      ACCO[yout+x] = offset;
    }
  }
}
```

```
//EMAX4A start .start_gdepth_x1:
//EMAX4A ctl map_dist=9
//EMAX4A @0,0  while (ri+=,-1)              rgi[1528,]
//EMAX4A @0,1  add (ri+=,1),r0              rgi[35,]
//EMAX4A @1,0  sub (-1,r0)|and (-,15),r1
//EMAX4A @1,1  |or (r0,0)>>4,r2
//EMAX4A @2,0  mluh (r1,ri)|or (-,0)>>4,r3  rgi[,.x1_offset:]
//EMAX4A @2,1  mluh (r2,ri),r4              rgi[,75]
//EMAX4A @2,2  add (ri,ri),r5               rgi[.x1_shift_x:,.x1_yin:]
//EMAX4A @3,0  add3 (r3,r4,r5)|or (-,0)<<2,r0
//EMAX4A @4,1  |or (r0,0),r1
//EMAX4A @4,2  |or (r0,0),r2
//EMAX4A @4,3  |or (r0,0),r3
//EMAX4A @5,0  ld (r0,ri),r21 rgi[,yzm_xm_p4_0:] & ld (r0,ri),r20 rgi[,yzm_xm_m4_0:] lmr[.x1_la_acci_yzm_0:,0,0,0,0,.x1_ma_acci_yzm_0:,7240]
//EMAX4A @5,1  ld (r1,ri),r9  rgi[,yzm_xz_p4_1:] & ld (r1,ri),r8  rgi[,yzm_xz_m4_1:] lmr[.x1_la_acci_yzm_1:,0,0,0,0,.x1_ma_acci_yzm_1:,7240]
//EMAX4A @5,2  ld (r2,ri),r23 rgi[,yzm_xp_p4_2:] & ld (r2,ri),r22 rgi[,yzm_xp_m4_2:] lmr[.x1_la_acci_yzm_2:,0,0,0,0,.x1_ma_acci_yzm_2:,7240]
//EMAX4A @5,3  ld (r3,ri),r11 rgi[,yzm_xz_p4_3:] & ld (r3,ri),r10 rgi[,yzm_xz_m4_3:] lmr[.x1_la_acci_yzm_3:,0,0,0,0,.x1_ma_acci_yzm_3:,7240]
//EMAX4A @6,0  msad  (r8 ,r20),r24
//EMAX4A @6,1  msad  (r9 ,r21),r25
//EMAX4A @6,2  msad  (r10,r22),r26
//EMAX4A @6,3  msad  (r11,r23),r27
//EMAX4A @7,0  mauh  (ri,r24),r28 rgi[0,]
//EMAX4A @7,1  mauh  (ri,r25),r29 rgi[0,]
//EMAX4A @7,2  mauh  (ri,r26),r30 rgi[0,]
//EMAX4A @7,3  mauh  (ri,r27),r31 rgi[0,]
//EMAX4A @8,0  ld (r0,ri),r21 rgi[,ymm_xm_p4:] & ld (r0,ri),r20 rgi[,ymm_xm_m4:] lmr[.x1_la_acci_ymm_0:,0,0,0,0,.x1_ma_acci_ymm_0:,7240]
//EMAX4A @8,2  ld (r2,ri),r23 rgi[,ymm_xp_p4:] & ld (r2,ri),r22 rgi[,ymm_xp_m4:] lmr[.x1_la_acci_ymm_2:,0,0,0,0,.x1_ma_acci_ymm_2:,7240]
//EMAX4A @9,0  msad  (r8 ,r20),r24
//EMAX4A @9,1  msad  (r9 ,r21),r25
//EMAX4A @9,2  msad  (r10,r22),r26
//EMAX4A @9,3  msad  (r11,r23),r27
//EMAX4A @10,0 mauh  (r28,r24),r28
//EMAX4A @10,1 mauh  (r29,r25),r29
//EMAX4A @10,2 mauh  (r30,r26),r30
//EMAX4A @10,3 mauh  (r31,r27),r31
//EMAX4A @11,0 ld (r0,ri),r21 rgi[,ypm_xm_p4:] & ld (r0,ri),r20 rgi[,ypm_xm_m4:] lmr[.x1_la_acci_ypm_0:,0,0,0,0,.x1_ma_acci_ypm_0:,7240]
//EMAX4A @11,2 ld (r2,ri),r23 rgi[,ypm_xp_p4:] & ld (r2,ri),r22 rgi[,ypm_xp_m4:] lmr[.x1_la_acci_ypm_2:,0,0,0,0,.x1_ma_acci_ypm_2:,7240]
//EMAX4A @12,0 msad  (r8 ,r20),r24
//EMAX4A @12,1 msad  (r9 ,r21),r25
//EMAX4A @12,2 msad  (r10,r22),r26
//EMAX4A @12,3 msad  (r11,r23),r27
//EMAX4A @13,0 mauh  (r28,r24),r28
//EMAX4A @13,1 mauh  (r29,r25),r29
//EMAX4A @13,2 mauh  (r30,r26),r30
//EMAX4A @13,3 mauh  (r31,r27),r31
//EMAX4A @14,0 ld (r0,ri),r21 rgi[,yzz_xm_p4_0:] & ld (r0,ri),r20 rgi[,yzz_xm_m4_0:] lmr[.x1_la_acci_yzz_0:,0,0,0,0,.x1_ma_acci_yzz_0:,7240]
//EMAX4A @14,1 ld (r1,ri),r13 rgi[,yzz_xz_p4_1:] & ld (r1,ri),r12 rgi[,yzz_xz_m4_1:] lmr[.x1_la_acci_yzz_1:,0,0,0,0,.x1_ma_acci_yzz_1:,7240]
//EMAX4A @14,2 ld (r2,ri),r23 rgi[,yzz_xp_p4_2:] & ld (r2,ri),r22 rgi[,yzz_xp_m4_2:] lmr[.x1_la_acci_yzz_2:,0,0,0,0,.x1_ma_acci_yzz_2:,7240]
//EMAX4A @14,3 ld (r3,ri),r15 rgi[,yzz_xz_p4_3:] & ld (r3,ri),r14 rgi[,yzz_xz_m4_3:] lmr[.x1_la_acci_yzz_3:,0,0,0,0,.x1_ma_acci_yzz_3:,7240]
//EMAX4A @15,0 msad  (r12,r20),r24
//EMAX4A @15,1 msad  (r13,r21),r25
//EMAX4A @15,2 msad  (r14,r22),r26
//EMAX4A @15,3 msad  (r15,r23),r27
//EMAX4A @16,0 mauh  (r28,r24),r28
//EMAX4A @16,1 mauh  (r29,r25),r29
//EMAX4A @16,2 mauh  (r30,r26),r30
//EMAX4A @16,3 mauh  (r31,r27),r31
//EMAX4A @17,0 ld (r0,ri),r21 rgi[,ymz_xm_p4:] & ld (r0,ri),r20 rgi[,ymz_xm_m4:] lmr[.x1_la_acci_ymz_0:,0,0,0,0,.x1_ma_acci_ymz_0:,7240]
//EMAX4A @17,2 ld (r2,ri),r23 rgi[,ymz_xp_p4:] & ld (r2,ri),r22 rgi[,ymz_xp_m4:] lmr[.x1_la_acci_ymz_2:,0,0,0,0,.x1_ma_acci_ymz_2:,7240]
//EMAX4A @18,0 msad  (r12,r20),r24
//EMAX4A @18,1 msad  (r13,r21),r25
//EMAX4A @18,2 msad  (r14,r22),r26
//EMAX4A @18,3 msad  (r15,r23),r27
//EMAX4A @19,0 mauh  (r28,r24),r28
//EMAX4A @19,1 mauh  (r29,r25),r29
//EMAX4A @19,2 mauh  (r30,r26),r30
//EMAX4A @19,3 mauh  (r31,r27),r31
//EMAX4A @20,0 ld (r0,ri),r21 rgi[,ypz_xm_p4:] & ld (r0,ri),r20 rgi[,ypz_xm_m4:] lmr[.x1_la_acci_ypz_0:,0,0,0,0,.x1_ma_acci_ypz_0:,7240]
//EMAX4A @20,2 ld (r2,ri),r23 rgi[,ypz_xp_p4:] & ld (r2,ri),r22 rgi[,ypz_xp_m4:] lmr[.x1_la_acci_ypz_2:,0,0,0,0,.x1_ma_acci_ypz_2:,7240]
//EMAX4A @21,0 msad  (r12,r20),r24
//EMAX4A @21,1 msad  (r13,r21),r25
//EMAX4A @21,2 msad  (r14,r22),r26
//EMAX4A @21,3 msad  (r15,r23),r27
//EMAX4A @22,0 mauh  (r28,r24),r28
//EMAX4A @22,1 mauh  (r29,r25),r29
//EMAX4A @22,2 mauh  (r30,r26),r30
//EMAX4A @22,3 mauh  (r31,r27),r31
//EMAX4A @23,0 ld (r0,ri),r21 rgi[,yzp_xm_p4_0:] & ld (r0,ri),r20 rgi[,yzp_xm_m4_0:] lmr[.x1_la_acci_yzp_0:,0,0,0,0,.x1_ma_acci_yzp_0:,7240]
//EMAX4A @23,1 ld (r1,ri),r17 rgi[,yzp_xz_p4_1:] & ld (r1,ri),r16 rgi[,yzp_xz_m4_1:] lmr[.x1_la_acci_yzp_1:,0,0,0,0,.x1_ma_acci_yzp_1:,7240]
//EMAX4A @23,2 ld (r2,ri),r23 rgi[,yzp_xp_p4_2:] & ld (r2,ri),r22 rgi[,yzp_xp_m4_2:] lmr[.x1_la_acci_yzp_2:,0,0,0,0,.x1_ma_acci_yzp_2:,7240]
//EMAX4A @23,3 ld (r3,ri),r19 rgi[,yzp_xz_p4_3:] & ld (r3,ri),r18 rgi[,yzp_xz_m4_3:] lmr[.x1_la_acci_yzp_3:,0,0,0,0,.x1_ma_acci_yzp_3:,7240]
//EMAX4A @24,0 msad  (r16,r20),r24
//EMAX4A @24,1 msad  (r17,r21),r25
//EMAX4A @24,2 msad  (r18,r22),r26
//EMAX4A @24,3 msad  (r19,r23),r27
//EMAX4A @25,0 mauh  (r28,r24),r28
//EMAX4A @25,1 mauh  (r29,r25),r29
//EMAX4A @25,2 mauh  (r30,r26),r30
//EMAX4A @25,3 mauh  (r31,r27),r31
//EMAX4A @26,0 ld (r0,ri),r21 rgi[,ymp_xm_p4:] & ld (r0,ri),r20 rgi[,ymp_xm_m4:] lmr[.x1_la_acci_ymp_0:,0,0,0,0,.x1_ma_acci_ymp_0:,7240]
//EMAX4A @26,2 ld (r2,ri),r23 rgi[,ymp_xp_p4:] & ld (r2,ri),r22 rgi[,ymp_xp_m4:] lmr[.x1_la_acci_ymp_2:,0,0,0,0,.x1_ma_acci_ymp_2:,7240]
//EMAX4A @27,0 msad  (r16,r20),r24
//EMAX4A @27,1 msad  (r17,r21),r25
//EMAX4A @27,2 msad  (r18,r22),r26
//EMAX4A @27,3 msad  (r19,r23),r27
//EMAX4A @28,0 mauh  (r28,r24),r28
//EMAX4A @28,1 mauh  (r29,r25),r29
//EMAX4A @28,2 mauh  (r30,r26),r30
//EMAX4A @28,3 mauh  (r31,r27),r31
//EMAX4A @29,0 ld (r0,ri),r21 rgi[,ypp_xm_p4:] & ld (r0,ri),r20 rgi[,ypp_xm_m4:] lmr[.x1_la_acci_ypp_0:,0,0,0,0,.x1_ma_acci_ypp_0:,7240]
//EMAX4A @29,2 ld (r2,ri),r23 rgi[,ypp_xp_p4:] & ld (r2,ri),r22 rgi[,ypp_xp_m4:] lmr[.x1_la_acci_ypp_2:,0,0,0,0,.x1_ma_acci_ypp_2:,7240]
//EMAX4A @30,0 msad  (r16,r20),r24
//EMAX4A @30,1 msad  (r17,r21),r25
//EMAX4A @30,2 msad  (r18,r22),r26
//EMAX4A @30,3 msad  (r19,r23),r27
//EMAX4A @31,0 mauh3 (r28,r24,r25),r28
//EMAX4A @31,2 mauh3 (r30,r26,r27),r30
//EMAX4A @32,0 mauh  (r28,r30) | suml (-),r31
//EMAX4A @32,1                                 & ld (ri+=,4),r28 rgi[.x1_sadi:,] lmf[.x1_la_sadi:,0,0,0,0,.x1_ma_sadi:,1528]
//EMAX4A @33,0 cmp.lt (r31,r28),c0
//EMAX4A @33,1 cmp.gt (r28,ri),c1 rgi[,137]
//EMAX4A @33,2 |or (ri,0),r30 rgi[.x1_ofso:,]
//EMAX4A @34,0 cexe (,,c1,c0,0x8888) & st r31,(ri+=,4) rgi[.x1_sado:,] lmx[.x1_la_sado:,0,0,0,0,.x1_ma_sado:,1528]
//EMAX4A @34,1 cexe (,,c1,c0,0x8888) & st r30,(ri+=,4) rgi[.x1_acco:,] lmx[.x1_la_acco:,0,0,0,0,.x1_ma_acco:,1528]
//EMAX4A end .end_gdepth_x1:
```

3.7 Examples (graph processing)

See proj-arm32/sample/tricount8/tricount-emax4.S. Triangle counting with CGRA and transactions is implemented. See proj-arm32/sample/dijkstra3-shimizu/dijkstra-emax4.S. Dijkstra with CGRA and transactions is implemented.

3.7.1 Triangle counting kernel0 with TCU

```
void tri_kernel0(struct param_bfs *param)
{
  volatile int i, j, pid, qid, MVL, MEL;
  volatile struct vertex *p, *np, *q;
  volatile struct neighborvertex *n;

  i = param->i;
  p = param->p;
  np = param->nextp;
  MVL = param->maxvlist;
  MEL = param->maxelist;
  pid = p->id;

    for (j=0; j<p->nedges; j++) {                    /* R 0段:最内ループ 256 回転程度 */
      n = p->npage[j/MAXNV_PPAGE]+(j%MAXNV_PPAGE);
      q = n->vp;                                       /* R 0段:neighborvertex 全体を配置 pointer を使い参照 */
      qid = n->id;                                     /* R 0段:同上 */
      if (!q->parent) {                                /* R 1段:vertex 全体を配置 pointer->pointer を使い参照 */
        /************************/
        while (cmpxchg(&Sem0, -1, param->th) != -1);
        /************************/
        if (!q->parent) {                              /* R 1段:同上 */
          if (nnextfrontiers >= MVL) {
            printf("vlist[%d] exhausted\n", MVL);
            exit(1);
          }
          q->parent = pid;                             /* W 2段:verex 更新 */
          q->depth  = depth;                           /* W 2段:同上 */
          q->findex = nnextfrontiers;                  /* W 2段:同上 */
          nextfrontier[nnextfrontiers] = q;            /* W 2段:next_frontier[] 更新 */
          nnextfrontiers++;                            /* W 2段:同上 */
          nnextfrontiers__neighbors+=q->nedges;
        }
        /************************/
        /*cmpxchg(&Sem0, param->th, -1);*/
        release(&Sem0, -1);
        /************************/
      }
      else if (q->depth==depth-1 && q->findex<i) {     /* R 1段:vertex 全体を配置 pointer->pointer を使い参照 */
        /************************/
        while (cmpxchg(&Sem1, -1, param->th) != -1);
        /************************/
        if (nfrontier_edges >= MEL) {
          printf("elist[%d] exhausted\n", MEL);
          exit(1);
        }
        frontier_edge[nfrontier_edges].src = (pid<qid)?p:q; /* W 2段:frontier_edge[] 更新 */
        frontier_edge[nfrontier_edges].dst = (pid<qid)?q:p; /* W 2段:同上 */
        nfrontier_edges++;                                  /* W 2段:同上 */
        nfrontier_edges__neighbors+=((pid<qid)?p:q)->nedges;
        /************************/
        /*cmpxchg(&Sem1, param->th, -1);*/
        release(&Sem1, -1);
        /************************/
      }
    }
}
```

```
//EMAX4A start .emax_start_tri_kernel0:
//EMAX4A ctl map_dist=1
//EMAX4A @0,0 while (ri+=,-1) rgi[.emax_count_tri_kernel0:,] & ld (ri+=,4),-  rgi[-4,] lmr[0,0,2,2,0,.emax_lmrma0_tri_kernel0:,.emax_lmrl0_tri_kernel0:]
//EMAX4A @0,1                                               & ld (ri+=,4),-  rgi[-4,] ! prefetch 済なら ld 実行 (-)(-)(id)(vp), なければ gather 後 ld 開始
//EMAX4A @0,2                                               & ld (ri+=,4),r0 rgi[-4,] ! rI(r0)
//EMAX4A @0,3                                               & ld (ri+=,4),r1 rgi[-4,] ! rV(r1)
//EMAX4A @1,0                                               &                        lmp[0,0,2,2,0,.emax_lmpma0_tri_kernel0:,.emax_lmpl0_tri_kernel0:]
//                                                                                   ! 次の vertex 周辺 neighborvertex の prefetch
//EMAX4A @2,0 cmp.lt (ri,r0),c0 rgi[.emax_rgi04pid_tri_kernel0:,] & ld (r1,8),r2 mmr[,0,2,0,0,,1] ! unit0<-(nedges) addr →  ← data0 rE(r2)
//EMAX4A @2,1                                                     & ld (,),r3                   ! unit1<-(parent)        ← data1 rP(r3)
//EMAX4A @2,2                                                     & ld (,),r4                   ! unit2<-(depth)         ← data2 rD(r4)
//EMAX4A @2,3                                                     & ld (,),r5                   ! unit3<-(findex)        ← data3 rF(r5)
//EMAX4A @3,0 cexe (,,,c0,0xaaaa) cmov (ri,r1),r6 rgi[.emax_rgi05p_tri_kernel0:,]  !  cond ? const(p):q          -> rSRC(r6)
//EMAX4A @3,1 cexe (,,,c0,0xaaaa) cmov (r1,ri),r7 rgi[,.emax_rgi06p_tri_kernel0:]  !  cond ? q:const(p)          -> rDST(r7)
//EMAX4A @3,2 cexe (,,,c0,0xaaaa) cmov (ri,r2),r8 rgi[.emax_rgi07ne_tri_kernel0:,] ! (cond ? const(p):q)->nedges) -> rNEN(r8)
//EMAX4A @3,3 cmp.eq (r3,0),c0
//EMAX4A @4,0 cexe (,,,c0,0xaaaa) & ld (r1,12),- mmtr[0,.trans0_start_tri_kernel0,.trans0_end_tri_kernel0] ! mem_bank tr_top tr_end
//                                          ^^^^自 fsm 経由で，他 MEM からトランザクションコード本体を取ってくる仕組み（2 回目以降は当然再利用）を仮定
//                                              当分は，トランザクションコードは MUX に設定済とし，fsm.memi の mem_top で特定できる.
//                  ^c3c2c1c0 の組合せ: 1111,1110,1101,1100,....,0011,0010,0001,0000 の各々に 0/1 を割り当てた 16bit を指定
//                   c0 の場合は, 1010101010101010=0xaaaa
//EMAX4A @4,1                     & ld (r1,0),-  ! (q)      word#1 →
//EMAX4A @4,2                     & ld (0,r2),-  ! (nedges) word#2 → @3.0.t1_v が conflict するので (r2,0) ではなく (0,r2) にして@3.0.t2_v を使用
//EMAX4A @5,0 cmp.ne (r3,0),c0
//EMAX4A @5,1 cmp.eq (r4,ri),c1   rgi[,.emax_rgi08de_tri_kernel0:] ! const(depth-1)
//EMAX4A @5,2 cmp.lt (r5,ri),c2   rgi[,.emax_rgi09i_tri_kernel0:]  ! const(i)
//EMAX4A @6,0 cexe (,c2,c1,c0,0x8080) & ld (r6,0),- mmtr[0,.trans1_start_tri_kernel0,.trans1_end_tri_kernel0] ! mem_bank tr_top tr_end
//                  ^c3c2c1c0 の組合せ: 1111,1110,1101,1100,....,0011,0010,0001,0000 の各々に 0/1 を割り当てた 16bit を指定
//                   c2&c1&c0 の場合は, 1000000010000000=0x8080
//EMAX4A @6,1                         & ld (r7,0),-  ! (dstp)   word#1 →
//EMAX4A @6,2                         & ld (r8,0),-  ! (nen)    word#2 →
//EMAX4A end .emax_end_tri_kernel0:

//EMAX4T start .trans0_start_tri_kernel0:
//EMAX4T @0 read  base=r0                      ofs=0     ?ne(0) term                          dst=r4 ! reg#4 は実際には再利用しない
//EMAX4T @1 read  base=.trans0_nnf_tri_kernel0: ofs=0     ?ge(.trans0_MVL_tri_kernel0:) error dst=r5 ! nnf->r5
//EMAX4T @2 write base=r0                      ofs=0                                          src=.trans0_pid_tri_kernel0: ! pid
//EMAX4T @3 write base=r0                      ofs=4                                          src=.trans0_dep_tri_kernel0: ! depth
//EMAX4T @4 write base=r0                      ofs=8                                          src=r5 ! nnf
//EMAX4T @5 write base=.trans0_nfp_tri_kernel0: ofs=r5<<2                                     src=r1 ! q
//EMAX4T @6 read  regv=r5                                +1                                   dst=r5 ! nnf increment
//EMAX4T @7 read  base=.trans0_nfn_tri_kernel0: ofs=0     +r2                                  dst=r6 ! nnf_n->tmp#2 初回のみ mem-read
//EMAX4T @8 write base=.trans0_nn2_tri_kernel0: ofs=0                                          src=r5 ! reg(nnf) writeback 最終的には EMAX4A 終了時のみ動作
//EMAX4T @9 write base=.trans0_nf2_tri_kernel0: ofs=0     term                                 src=r6 ! reg(nnf_n) writeback 最終的には EMAX4A 終了時のみ動作
//EMAX4T end .trans0_end_tri_kernel0:

//EMAX4T start .trans1_start_tri_kernel0:
//EMAX4T @0 read  base=.trans1_nfe_tri_kernel0: ofs=0     ?ge(.trans1_MEL_tri_kernel0:) error dst=r5 ! nfe->reg#5
//EMAX4T @1 write base=.trans1_fre_tri_kernel0: ofs=r5<<3                                     src=r0
//EMAX4T @2 write base=.trans1_fr4_tri_kernel0: ofs=r5<<3                                     src=r1
//EMAX4T @3 read  regv=r5                                +1                                   dst=r5 ! nfe increment
//EMAX4T @4 read  base=.trans1_nen_tri_kernel0: ofs=0     +r2                                  dst=r6 ! nfe_n->reg#6 初回のみ mem-read
//EMAX4T @5 write base=.trans1_nf2_tri_kernel0: ofs=0                                          src=r5 ! reg(nfe) writeback 最終的には EMAX4A 終了時のみ動作
//EMAX4T @6 write base=.trans1_ne2_tri_kernel0: ofs=0     term                                 src=r6 ! reg(nfe_n) writeback 最終的には EMAX4A 終了時のみ動作
//EMAX4T end .trans1_end_tri_kernel0:
```

3.7.2 Triangle counting kernel1 with TCU

```
void tri_kernel1(struct param_tricount *param)
{
  /* search triangle in {frontier,next} */
  /* case 1: e ∈ frontier, v ∈ prev     */
  /* case 2: e ∈ frontier, v ∈ frontier */
  /* case 3: e ∈ frontier, v ∈ next     */
  int i, j, pid, qid, sdepth, tdepth, tricount;
  struct vertex *p, *np, *q, *t;
  struct neighborvertex *n;

  p = param->p;
  np = param->nextp;
  t = param->t;
  pid = p->id;
  sdepth = p->depth;

    tricount = 0;
    for (j=0; j<p->nedges; j++) {                     /* R 0段:最内ループ 256 回転程度 */
      n = p->npage[j/MAXNV_PPAGE]+(j%MAXNV_PPAGE);
      q = n->vp;                                      /* R 0段:neighborvertex 全体を配置 pointer を使い参照 */
      qid = n->id;                                    /* R 0段:同上 */
      tdepth = q->depth;                              /* R 1段:vertex 全体を配置 pointer->pointer を使い参照 */
      if ((tdepth==sdepth-1)||(tdepth==sdepth+1)||(tdepth==sdepth && qid<pid)) { /* R 2段:比較 */
        if (search_nvertex(t->nhashtbl, qid))         /* R 3段:HASH-SEARCH/CAM-SEARCH */
          tricount++;                                 /* W 4段:カウンタ更新 */
      }
    }
    param->tricount += tricount;
}
```

```
//EMAX4A start .emax_start_tri_kernel1:
//EMAX4A ctl map_dist=1
//EMAX4A @0,0 while (ri+=,-1) rgi[.emax_count_tri_kernel1:,] & ld (ri+=,4),-  rgi[-4,] lmr[0,0,2,2,0,.emax_lmrma0_tri_kernel1:,.emax_lmrl0_tri_kernel1:]
//EMAX4A @0,1                                                & ld (ri+=,4),-  rgi[-4,] ! prefetch 済なら ld 実行 (-)(-)(id)(vp), なければ gather 後 ld 開始
//EMAX4A @0,2                                                & ld (ri+=,4),r0 rgi[-4,] ! rI(r0)
//EMAX4A @0,3                                                & ld (ri+=,4),r1 rgi[-4,] ! rV(r1)
//EMAX4A @1,0                                                &                         lmp[0,0,2,2,0,.emax_lmpma0_tri_kernel1:,.emax_lmpl0_tri_kernel1:]
//                                                                                     ! 次の vertex 周辺 neighborvertex の prefetch
//EMAX4A @2,0                                                & ld (r1,8),r2   mmr[,0,2,0,0,,1] ! unit0<-(nedges) addr →  ← data0 rE(r2)
//EMAX4A @2,1                                                & ld (,),r3                       ! unit1<-(parent)         ← data1 rP(r3)
//EMAX4A @2,2                                                & ld (,),r4                       ! unit2<-(depth)          ← data2 rD(r4)
//EMAX4A @2,3                                                & ld (,),r5                       ! unit3<-(findex)         ← data3 rF(r5)
//EMAX4A @3,0 cmp.eq (r4,ri),c0 rgi[,.emax_rgi04sdm1_tri_kernel1:] ! const(sdepth-1)
//EMAX4A @3,1 cmp.eq (r4,ri),c1 rgi[,.emax_rgi05sdp1_tri_kernel1:] ! const(sdepth+1)
//EMAX4A @3,2 cmp.eq (r4,ri),c2 rgi[,.emax_rgi06sd_tri_kernel1:]   ! const(sdepth)
//EMAX4A @3,3 cmp.lt (r0,ri),c3 rgi[,.emax_rgi07pid_tri_kernel1:]  ! const(pid)
//                                                                 ! t->nhashtbl を LMM に prefetch してよい. 通常 1 回で HIT と考えても hash 検索回数は頂点数あり高速化可能
//                                                                 ! ただし, seqlink と異なり, hash 値が同じ V を繋ぐリンクは離散. next_ptr の変換が必要. addr->TLB->LMM
//EMAX4A @4,0 cexe (c3,c2,c1,c0,0xfeee) & ld (r0,0),- mmtr[0,.trans0_start_tri_kernel1,.trans0_end_tri_kernel1] ! mem_bank tr_top tr_end
//                  ^c3c2c1c0 の組合せ: 1111,1110,1101,1100,....,0011,0010,0001,0000 の各々に 0/1 を割り当てた 16bit を指定
//                   c0|c1|(c2&c3) の場合は, 1111111011101110=0xfeee
//EMAX4A end .emax_end_tri_kernel1:

//EMAX4T start .trans0_start_tri_kernel1:
//EMAX4T @0 read  base=.trans0_nht_tri_kernel1: ofs=r0.0<<2 ?eq(0)  term    dst=r5 ! vp->reg#5
//EMAX4T @1 read  base=r5                       ofs=8       ?eq(r0) @3      dst=r4 ! hash 探索の表現 tmp#0 は実際には再利用しない
//EMAX4T @2 read  base=r5                       ofs=0       ?eq(0)  term @1 dst=r5 ! repeat
//EMAX4T @3 read  base=.trans0_tr0_tri_kernel1: ofs=0       +1              dst=r6 ! increment 初回のみ mem-read tricount->reg#6
//EMAX4T @4 write base=.trans0_tr1_tri_kernel1: ofs=0       term            src=r6 ! writeback 最終的には EMAX4A 終了時のみ動作
//EMAX4T end .trans0_end_tri_kernel1:
```

3.7.3 Dijkstra kernel with TCU

```
void *dij_kernel(param) struct vertex *param;
{
  //puts("ker1");
  int j, MFL, min_dist, new_dist;
  struct frontier *pf;
  struct frontier *f;
  struct vertex *p, *q;
  struct neighborvertex *n;

  p        = param;
  //MFL      = param->maxflist;

  for (j=0; j<p->nedges; j++) {
    n = p->npage[j/MAXNV_PPAGE]+(j%MAXNV_PPAGE);
    q = n->vp;
    new_dist = p->total_distance + n->distance;
    if (q->total_distance > new_dist) {
      /************************/
      //while (cmpxchg(&Sem, -1, ) != -1);
      /************************/
      if (q->total_distance > new_dist) {
        if (!freelist) {
          printf("frontier[%d] exhausted\n", MFL);
          exit(1);
        }
        f = freelist;
        freelist = freelist->fp;
        f->vp = q;

        //nfrontiers++;

        q->parent = p;
        q->total_distance = new_dist;

        f->fp = dclass[new_dist%MAXDCLASS];
        dclass[new_dist%MAXDCLASS] = f;
        //f->pfp = dclass[new_dist%MAXDCLASS];
      }
      /************************/
      //cmpxchg(&Sem, param->th, -1);
      /************************/
    }
  }
}
```

```
//EMAX4A start .emax_start_dij_kernel:
//EMAX4A ctl map_dist=1
//EMAX4A @0,0 while (ri+=,-1) rgi[.emax_count_dij_kernel:,] & ld (ri+=,4),-  rgi[-4,] lmr[0,0,2,2,0,.emax_lmrma0_dij_kernel:,.emax_lmrl0_dij_kernel:]
//EMAX4A @0,1                                                & ld (ri+=,4),r0 rgi[-4,] ! distance(r0)
//EMAX4A @0,2                                                & ld (ri+=,4),-  rgi[-4,] !
//EMAX4A @0,3                                                & ld (ri+=,4),r1 rgi[-4,] ! *vp(r1)
//EMAX4A @1,0 add (ri,r0),r10 rgi[.emax_t_dist_dij_kernel:,] ! new_dist(r10) = ri(t_dist) + distance(r0)
//EMAX4A @2,0                                                & ld (r1,0),r2 mmr[,0,2,0,0,,1] ! unit0<-(nedges) addr                                        → ←
data0 rE(r2) ! (lmm_top) mem_bank width block dist (top) len
//EMAX4A @2,1                                                & ld (,),r3                     ! unit1<-(parent)         ← data1 rP(r3)
//EMAX4A @2,2                                                & ld (,),r4                     ! unit2<-(total_distance)    ← data2 rTD(r4)
//EMAX4A @2,3                                                & ld (,),-                      ! unit3<-(min_neighbor_dist) ← data3 rMND(r5)
//EMAX4A @3,0 cmp.lt (r10,r4),c0 ! c0 = new_dist(r10) < total_distance
//EMAX4A @3,1 cmp.eq (r3, 0),c1
//EMAX4A @3,2 | and (r10,0x7f),r11 !
//EMAX4A @4,0 cexe (,,,c0,0xaaaa)       & ld (r10,0),-  mmtr[0,.trans0_start_dij_kernel,.trans0_end_dij_kernel] ! mem_bank tr_top tr_end
//EMAX4A @4,1                           & ld (r1,0),-   ! arg(r10, r1, r11) = (new_dist, q, new_dist%MAXDCLASS)
//EMAX4A @4,2                           & ld (r11,0),-
//EMAX4A end .emax_end_dij_kernel:

//EMAX4T start .trans0_start_dij_kernel: !registration to dclass and swith freelist r0=new_dist r1=q r2=dclass_index=new_dist%MAXDCLASS r3=p
//EMAX4T @0 read  base=r1                               ofs=8 ?ge(r0)@1 term dst=r4     ! if (total_distance < new_distance) then term
//EMAX4T @1 read  base=.trans0_freelist_kernel:         ofs=0 dst=r4            ! r4 = f = freelist
//EMAX4T @2 read  base=r4                               ofs=0 dst=r5            ! r5 = f->fp
//EMAX4T @3 write base=.trans0_fl2_kernel:              ofs=0 src=r5            ! freelist = freelist->fp
//EMAX4T @4 write base=r4                               ofs=4 src=r1            ! freelist->vp = q
//EMAX4T @5 read  base=.trans0_dclass_kernel:           ofs=r2<<2 dst=r6        ! r6 = dclass[r2]
//EMAX4T @6 write base=r4                               ofs=0 src=r6            ! f(r4)->fp = r6(dlass[r2])
//EMAX4T @7 write base=.trans0_dcl1_kernel:             ofs=r2<<2 src=r4        ! dclass[r2] = f(r4)
//EMAX4T @8 write base=r1                               ofs=4 src=.trans0_p_kernel:     ! q(r1)->parent = p
//EMAX4T @9 write base=r1                               ofs=8 term src=r0               ! q(r1)->total_distance = new_distance
//EMAX4T end .trans0_end_dij_kernel:
```

3.8 Compiling application programs

See "all:" tag in each Makefile-arm.emax4.

3.9 Executing application programs on simulator

See "run:" tag in each Makefile-arm.emax4.

Appendix A

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，ツールチェインを列挙する．

A.1 EMAX2asic/ZYNQ

- EMAX2 基本特許 proj-emax/doc/pat33.tgz
- EMAX2 科研基盤 A proj-emax/doc/kaken2012.tgz
- EMAX2/intel 仕様書 proj-emax/doc/emax2/emax2.pdf
- EMAX2asic 設計データ proj-emax/fpga/step4008-GP6X-fpu/RTL/pe0/
- EMAX2asic/ZYNQ 仕様書 proj-arm32/doc/emax4/emax4.pdf
- ZYNQ-7000 Technical Reference Manual proj-zynq706/doc/xilinx/ug585-Zynq-7000-TRM.pdf
- AMBA4 AXI and ACE Protocol Specification
  .... proj-zynq706/doc/IHI0022E_amba_axi_and_ace_protocol_spec.pdf
- AMBA4 AXI4-Stream Protocol
  .... proj-zynq706/doc/IHI0051A_amba4_axi4_stream_v1_0_protocol_spec.pdf
- ARM アーキテクチャ仕様書 .... proj-arm32/doc/arm/DDI0100E_arm_arm.pdf
- ARM システムコール仕様書 proj-arm32/doc/arm/swi.pdf
- EMAX2asic ZYNQ インタフェース proj-arm32/fpga/step4008-ZYNQ-fpu-rohm18/
- コンパイル環境は ZYNQ-Linux .... arch04(主記憶 512MB),arch05(主記憶 1GB)
- EMAX2asic ディレクティブ変換 proj-arm32/src/conv-a2b/conv-a2b
- EMAX2asic シミュレータ proj-arm32/sample/stencil-pipe/emax2.c
- プログラム例（浮動小数点ステンシル） proj-arm32/sample/stencil-pipe/stencil-zynq.emax2

A.2 EMAX4/bsim

- ARM アーキテクチャ仕様書 .... proj-arm32/doc/arm/DDI0100E_arm_arm.pdf
- ARM システムコール仕様書 proj-arm32/doc/arm/swi.pdf
- EMAX4/bsim 仕様書 proj-arm32/doc/emax4/emax4.pdf
- コンパイル環境は Intel-Linux cad101-102,111-120,605-606
- EMAX4 C コンパイラ proj-arm32/bin/arm-uclinux-eabi-gcc
- EMAX4 アセンブラ proj-arm32/bin/arm-uclinux-eabi-as
- EMAX4 リンカ proj-arm32/bin/arm-uclinux-eabi-ld
- EMAX4 逆アセンブラ proj-arm32/bin/arm-uclinux-eabi-objdump
- EMAX4 標準ライブラリ proj-arm32/arm-uclinux-eabi/lib/*.a
- EMAX4 GCC ライブラリ proj-arm32/lib/gcc/arm-uclinux-eabi/4.8.2/*.a
- EMAX4 ディレクティブ変換 proj-arm32/src/conv-a2c/conv-a2c

- EMAX4 シミュレータ proj-arm32/src/bsim/bsim
- プログラム例（画像フィルタ） proj-arm32/sample/filter/filter-arm.emax4
- プログラム例（浮動小数点ステンシル） proj-arm32/sample/stencil-pipe/stencil-arm.emax4
- プログラム例（4D 画像処理. 再生） proj-arm32/sample/4dimage/gather-arm.emax4
- プログラム例（4D 画像処理. 距離画像） proj-arm32/sample/4dimage/gdepth-arm.emax4
- プログラム例（グラフ処理） proj-arm32/sample/tricount8/tricount-arm.emax4