Green Computing Platforms

Step3. CA1002+1003:FPGA and CGRA

http://archlab.naist.jp/Lectures/ARCH/ca1002_1003/ca100203e.pdf

Copyright © 2024 NAIST Y.Nakashima

Download the template and submit through UNIPA.

http://archlab.naist.jp/Lectures/ARCH/ca1002_1003/ca100203e.docx

in http://archlab.naist.jp/Lectures

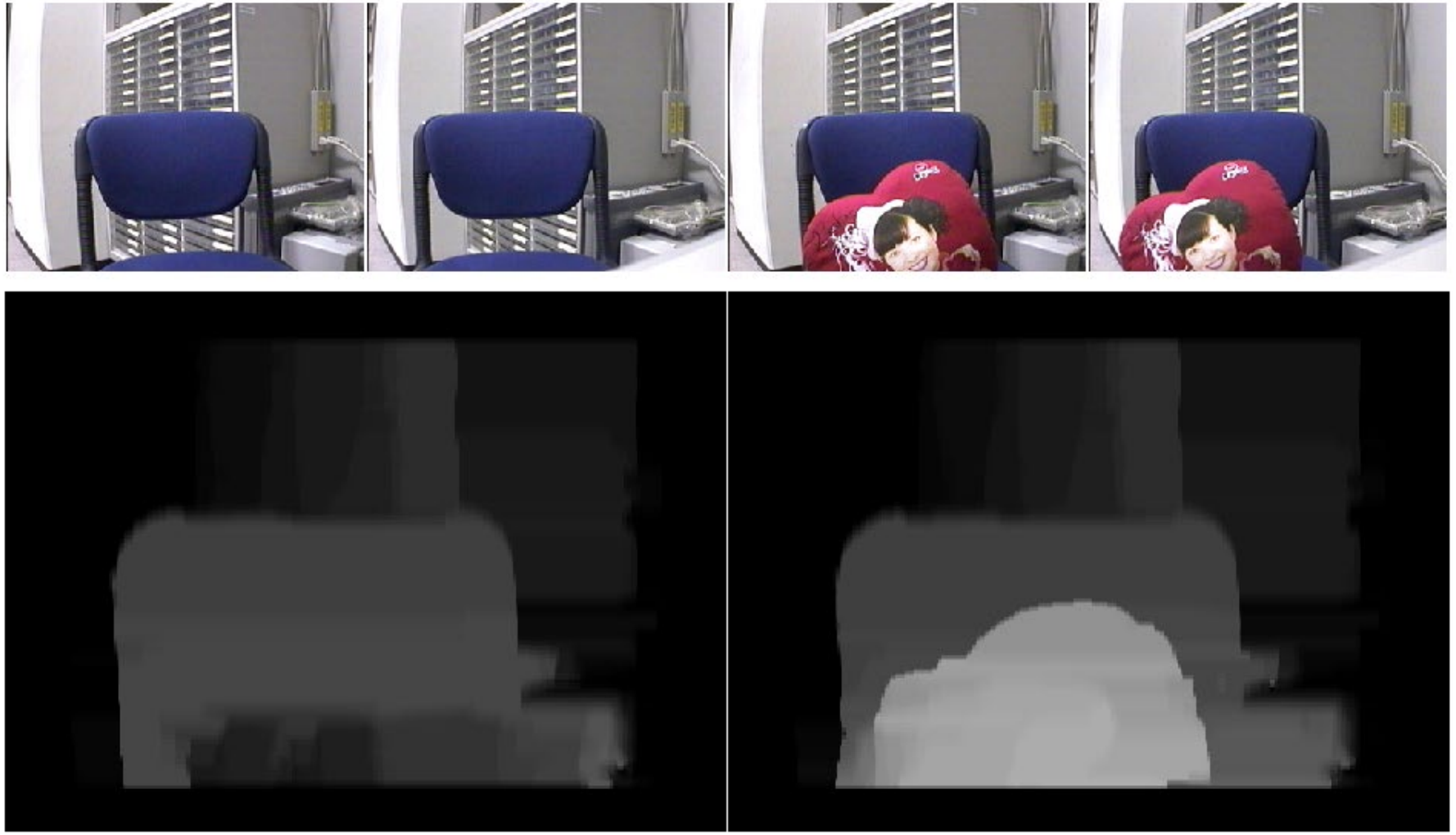
FPGA

Field Programmable Gate Array

(Fine Grained Reconfigurable Array)

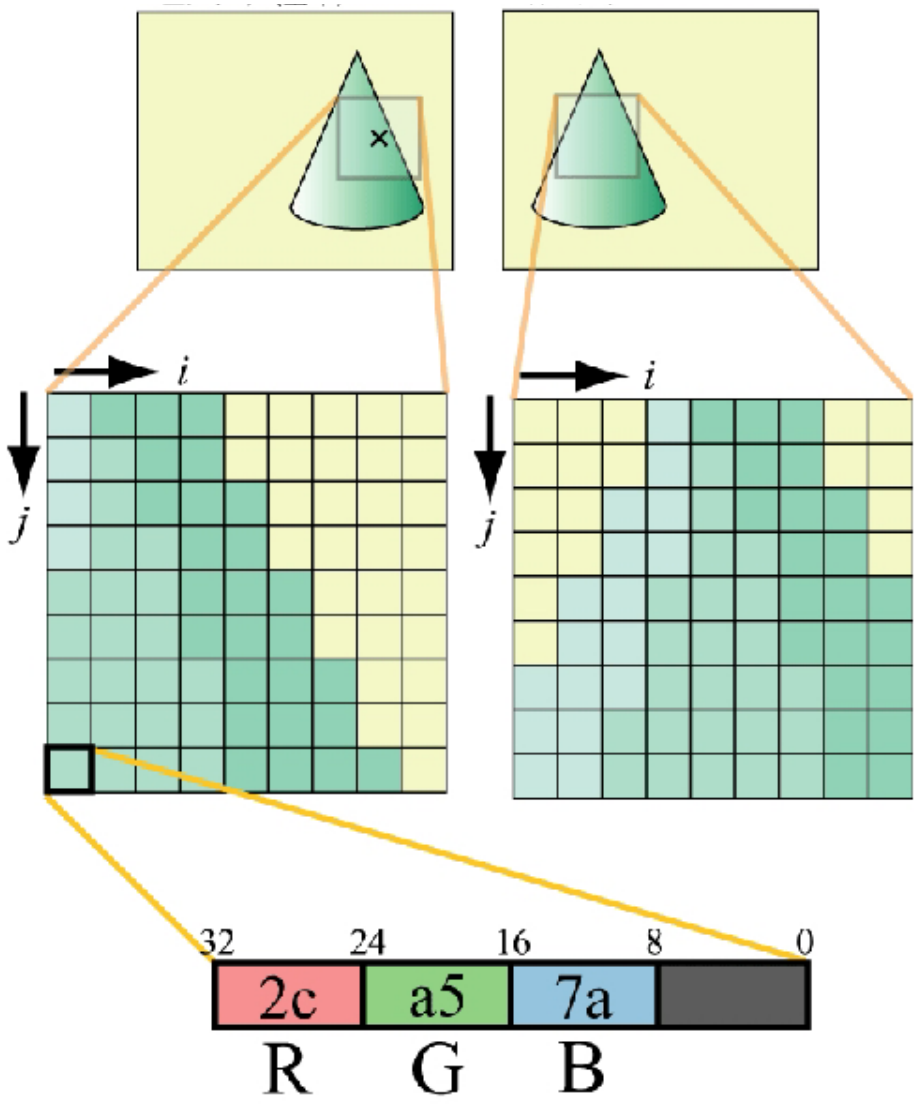
Stereo matching

Object detection



Stereo matching

Search location where small window has minimum difference

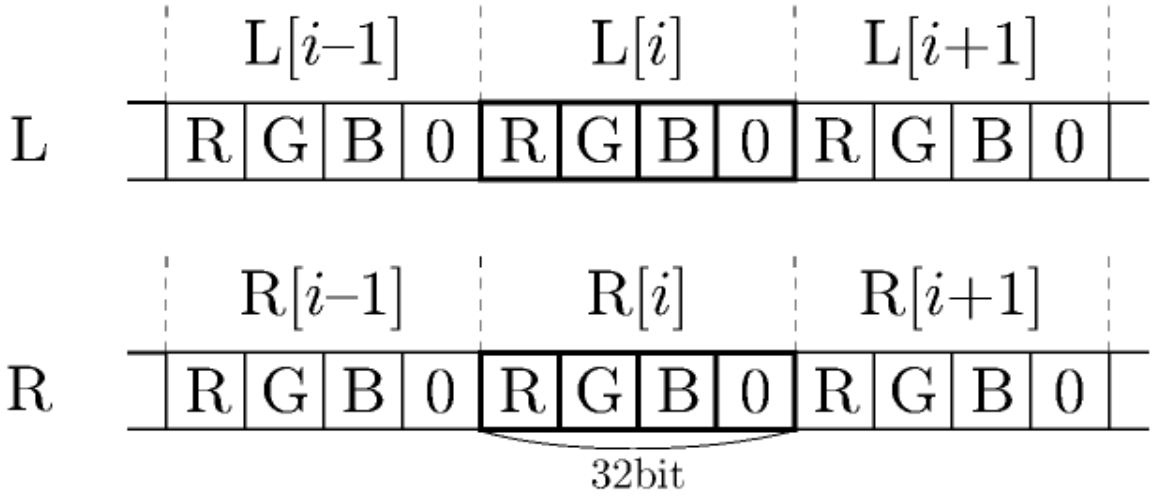


NAIST

How to measure difference

SAD (Sum of Absolute Difference)

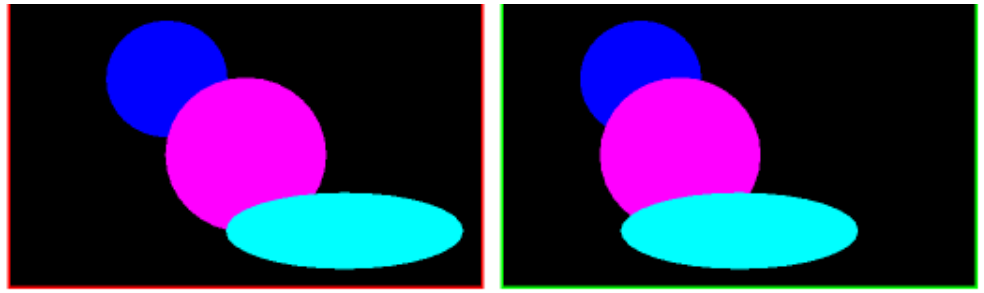
- **around 21x21 pixels in 320x240 image**
- **Each pixel has RGB0 (word-aligned 8bits*4)**



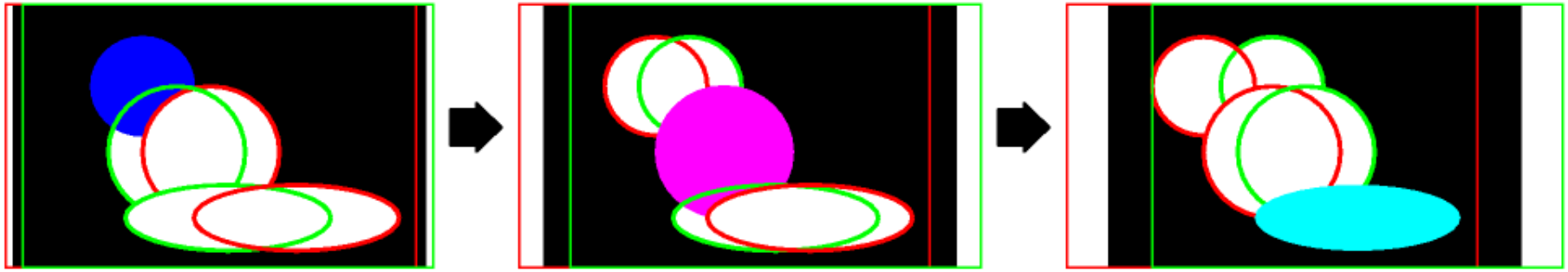
$$DIFF_i = |L[i]_R - R[i]_R| + |L[i]_G - R[i]_G| + |L[i]_B - R[i]_B|$$

Windows with edges are important

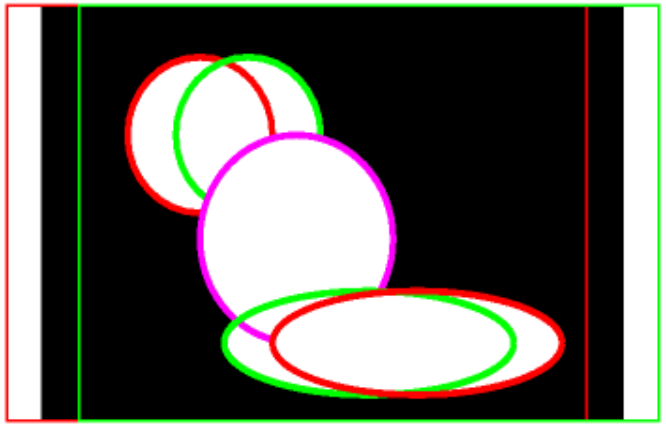
Calculate distance from far-side to near-side

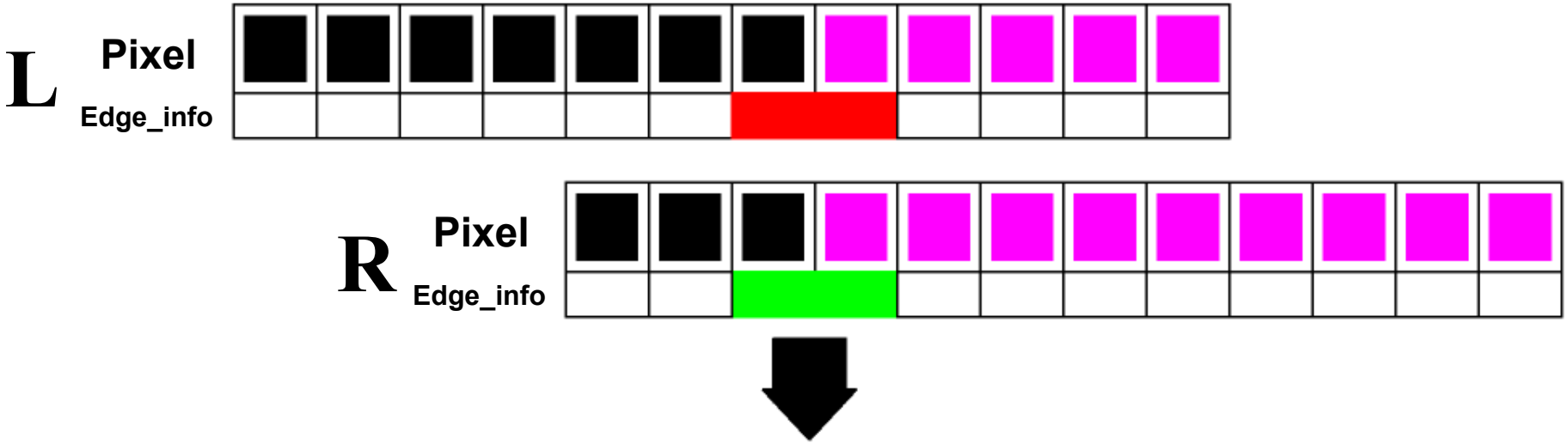


Focus on far-side | Focus on mid-side | Focus on near-side



Windows with edges are important

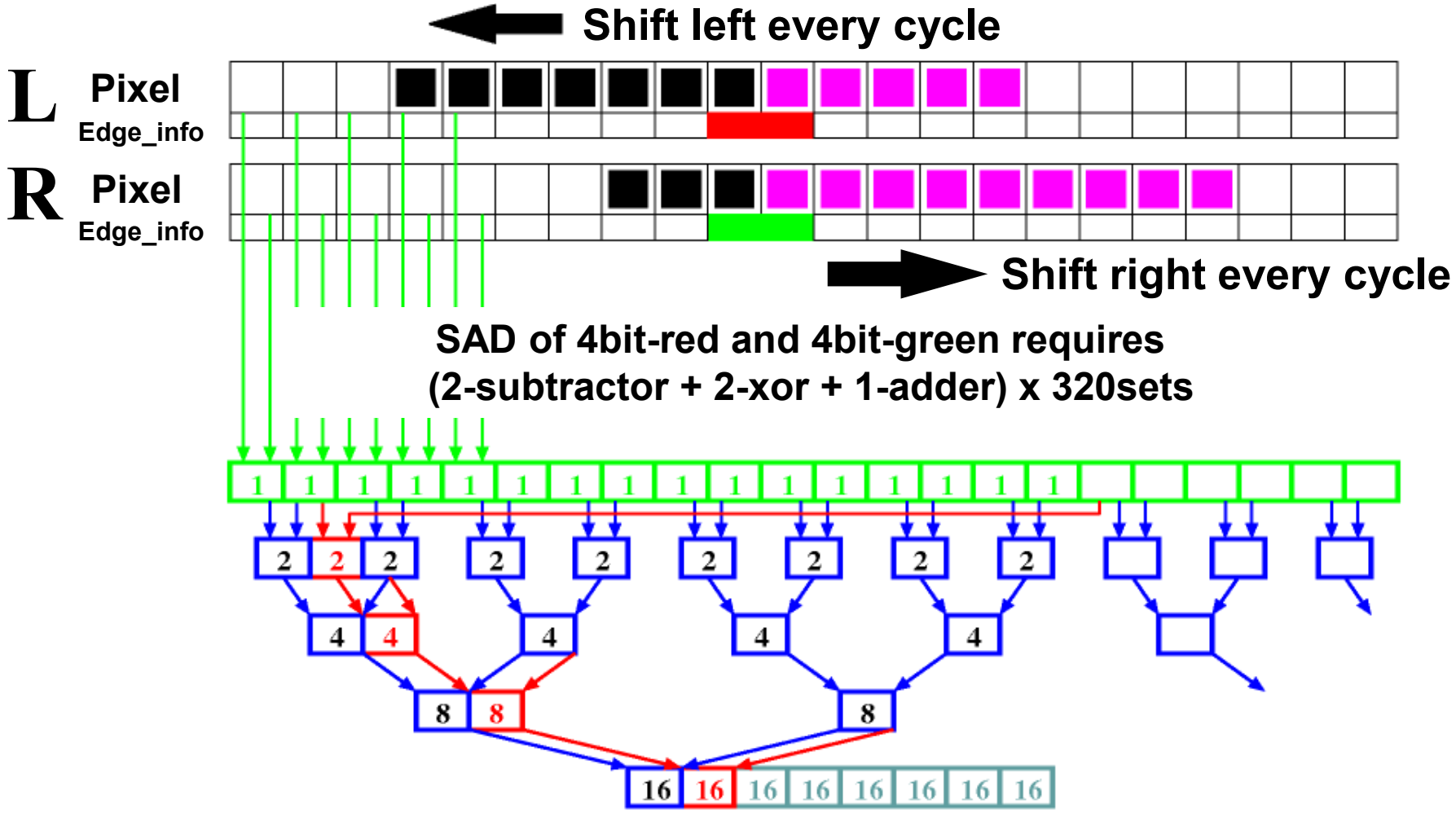




Each window has edge and minimum SAD
⇒ distance found !

Calculation by FPGA

Static allocation of pixels and huge number of ALUs



Calculation by FPGA

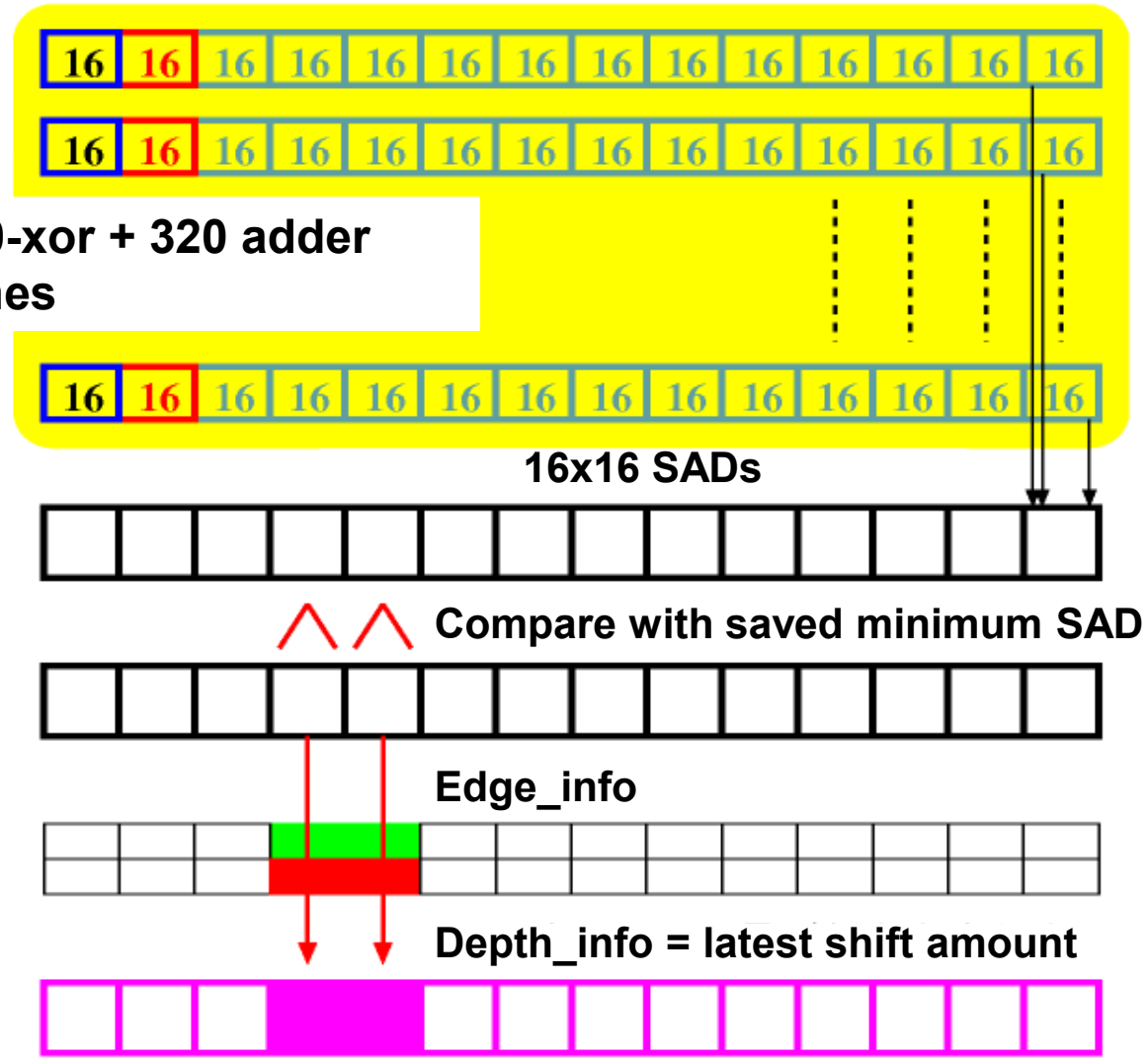
Hardware for simultaneous processing of 16lines

Results from previous page
x 16lines

640-subtractor + 640-xor + 320 adder
x 16lines

Total:

8bit-register	11520
Subtractor	10240
Xor	10240
Adder	5120



Depth_info of one line is
updated every cycle

Estimation of Performance

Let's assume SSE2 can calculate SAD instruction in 1 cycle.

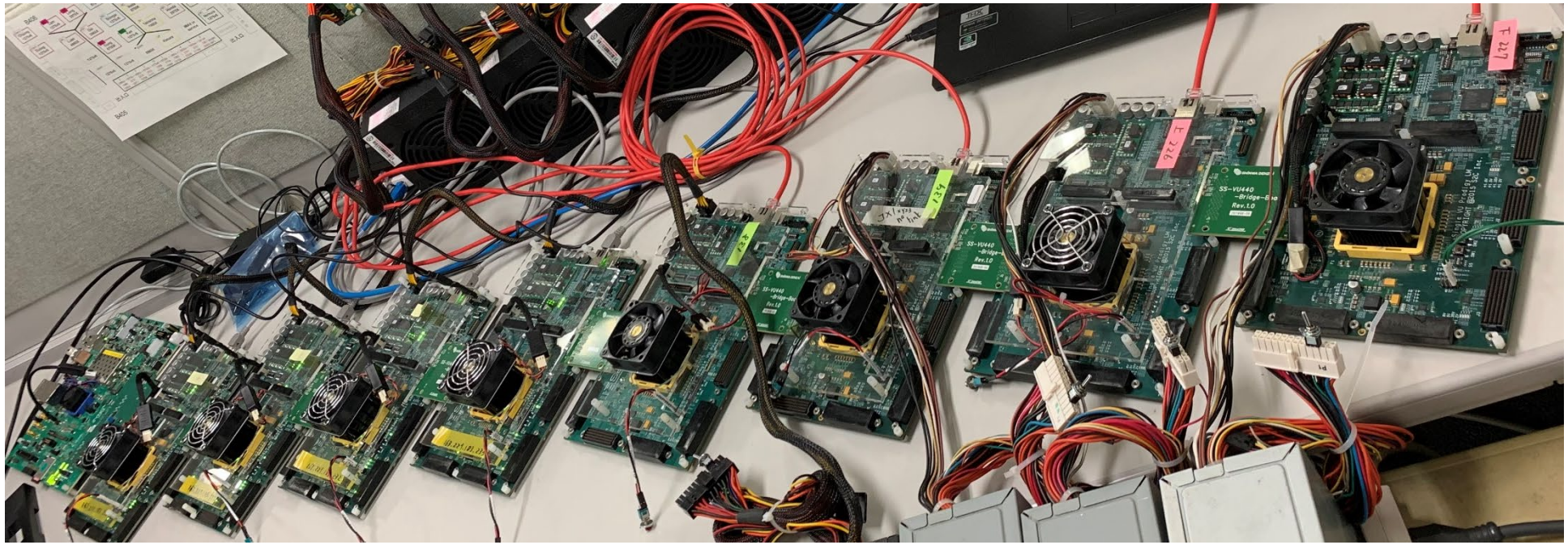
- **8,000,000,000pixels/second (2GHz)**

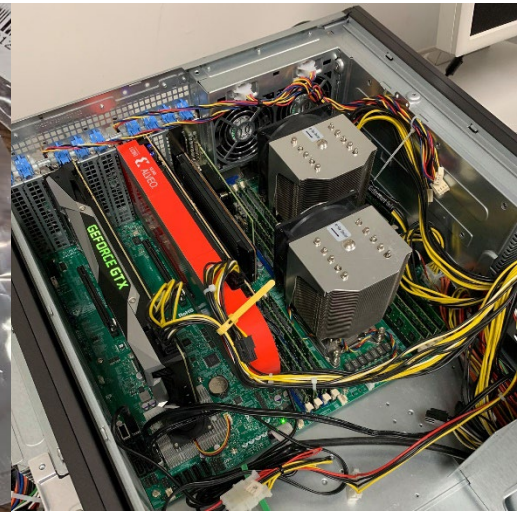
FPGA can calculate 81920 (320x16x16) SAD in 1 cycle.

- **2,700,000,000,000pixels/second (33MHz)**
- **320times faster than SSE2 with 1/60 frequency**

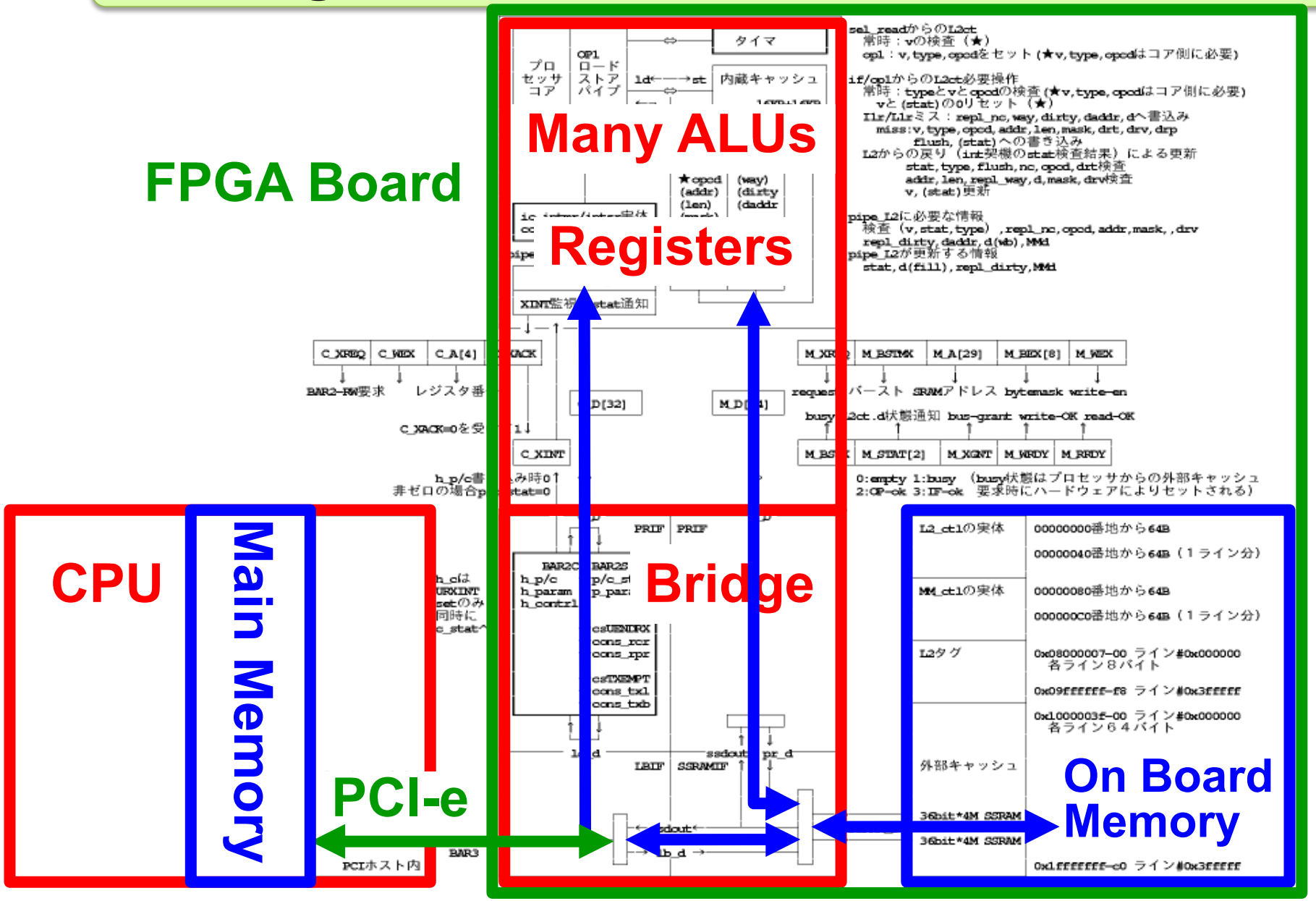
Performance of FPGA: 600frame/seconds

Sample platform





High-end FPGAs have PCI-e interface



How to access memories on FPGA board

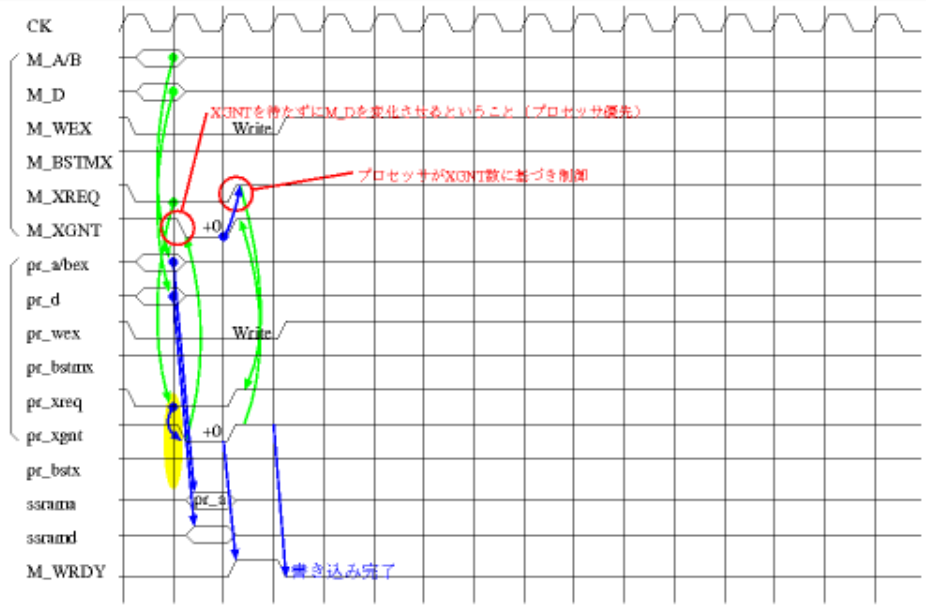
mmap() system call maps on-board memory to user memory space

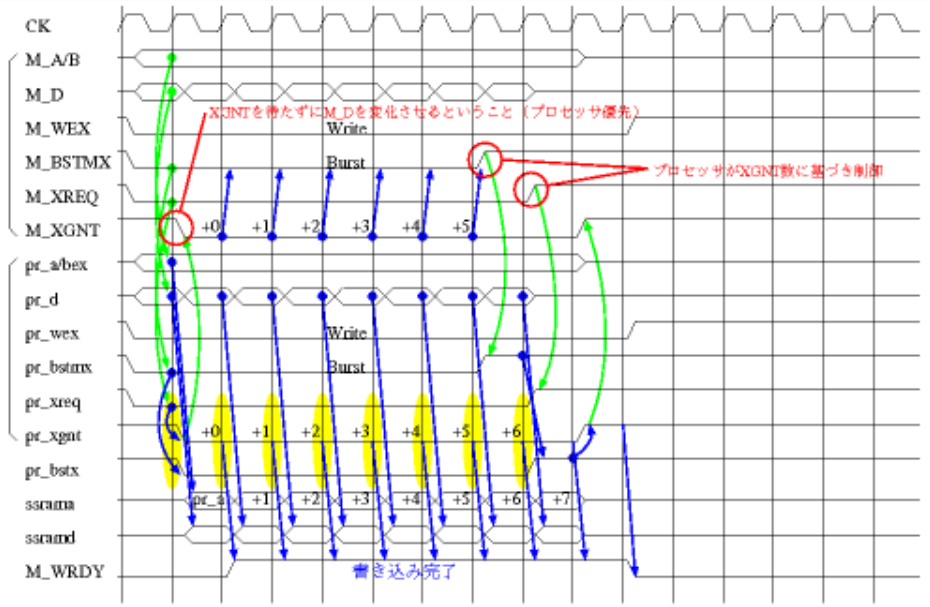
```
if (wait_previous) {
  do { soft_l2ct[BAR3_L2CT_0000] = hard_l2ctr(BAR3_L2CT_0000); /* ★SOFT←HARD★ */
  } while (soft_l2ct[BAR3_L2CT_0000].l2ct_0000.v != 3 || soft_l2ct[BAR3_L2CT_0000].l2ct_0000.stat != 1);
} wait_previous = 1;
/*  Get result Dl,Dr from on-board memory  */
for (i=0; i<HT; i++) {
  for (j=0; j<WD/L2LINE; j++) { /* キャッシュライン=64byte WD=320byte つまり WD1行分=キャッシュ5ライン分 */
    hard_l2lir(0x03000+i*(WD/L2LINE)+j, &Dl[i][j*L2LINE]); /* ★SOFT←HARD★ */
    hard_l2lir(0x03800+i*(WD/L2LINE)+j, &Dr[i][j*L2LINE]); /* ★SOFT←HARD★ */
  }
}
/* Write Bl, Br, El, Er to on-board memory              */
for (i=0; i<HT; i++) {
  for (j=0; j<WD/(L2LINE/2); j++) { /* キャッシュライン=32short WD=320short つまり WD1行分=キャッシュ10ライン分 */
    hard_l2liw(0x00000+i*(WD/(L2LINE/2))+j, &Bl[i*WD+j*L2LINE/2]); /* ★SOFT←HARD★ */
    hard_l2liw(0x01000+i*(WD/(L2LINE/2))+j, &Br[i*WD+j*L2LINE/2]); /* ★SOFT←HARD★ */
    hard_l2lisync(0x00000+i*(WD/(L2LINE/2))+j); hard_l2lisync(0x01000+i*(WD/(L2LINE/2))+j);
  }
}
for (i=0; i<HT; i++) {
  for (j=0; j<WD; j+=8) { /* キャッシュライン=512bit WD=320bit つまり WD1行分=キャッシュ5ライン分 */
    el[i][j/8] = (El[i][j+7]?0x80:0)|(El[i][j+6]?0x40:0)|(El[i][j+5]?0x20:0)|(El[i][j+4]?0x10:0)
                |(El[i][j+3]?0x08:0)|(El[i][j+2]?0x04:0)|(El[i][j+1]?0x02:0)|(El[i][j  ]?0x01:0);
    er[i][j/8] = (Er[i][j+7]?0x80:0)|(Er[i][j+6]?0x40:0)|(Er[i][j+5]?0x20:0)|(Er[i][j+4]?0x10:0)
                |(Er[i][j+3]?0x08:0)|(Er[i][j+2]?0x04:0)|(Er[i][j+1]?0x02:0)|(Er[i][j  ]?0x01:0);
  }
}
for (i=0; i<HT*WD/(L2LINE*8); i++) {
  hard_l2liw(0x02000+i, &el[0][0]+i*L2LINE); /* ★SOFT←HARD★ */
  hard_l2liw(0x02800+i, &er[0][0]+i*L2LINE); /* ★SOFT←HARD★ */
  hard_l2lisync(0x02000+i); hard_l2lisync(0x02800+i);
}
/* Start FPGA     */
soft_l2ct[BAR3_L2CT_0000].l2ct_0000.stat = 3;
hard_l2ctw(BAR3_L2CT_0000, soft_l2ct[BAR3_L2CT_0000]); /* ★HARD←SOFT★ */
hard_l2ctsync();                                       /* ★HARD←SOFT★ */
```

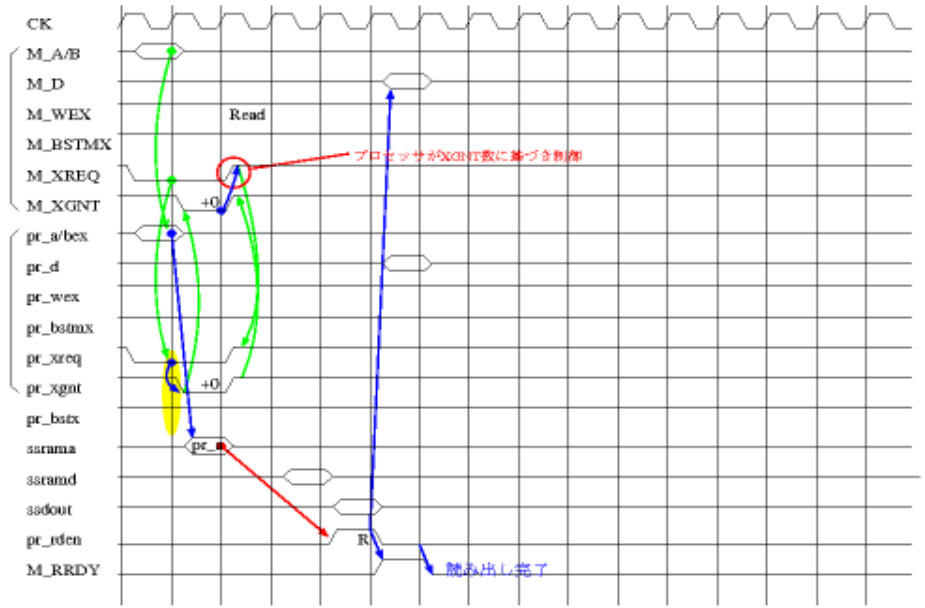
Hard-soft interface from CPU

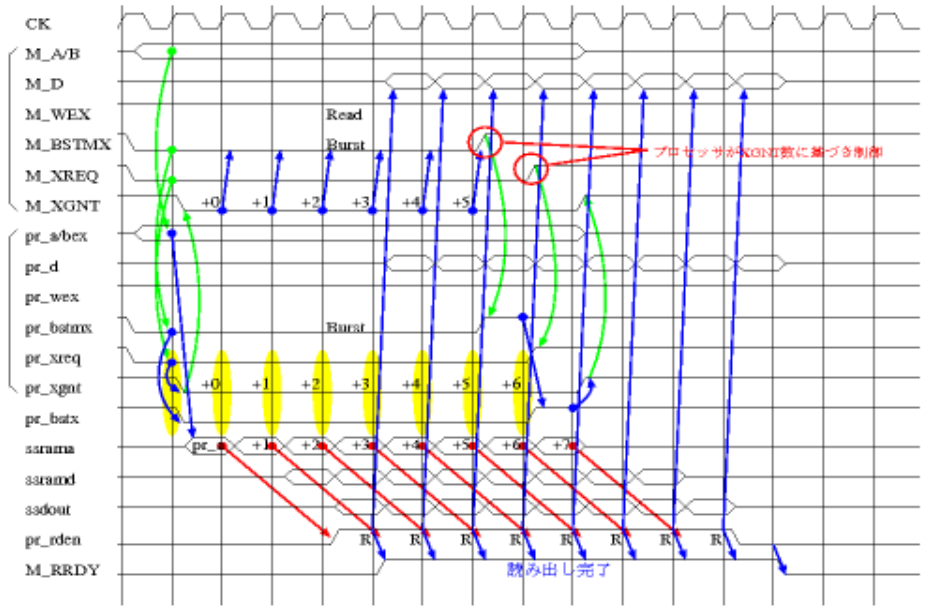
```
/* I/O空間を主記憶へ写像 */ mmap() system call maps on-board memory to user memory space
int gp600m_open()
{ if ((fd=open(GP600M_DEVNAME,O_RDWR)) == -1) {
    printf("gp600m.c: ERROR can not open gp600m\n");
    return (-1);
  }
  if((hard_l2ct=
     (union l2ct*)mmap((caddr_t)0,0x4000000,PROT_READ|PROT_WRITE,MAP_SHARED,fd,(off_t)0))==MAP_FAILED) {
    printf("gp600m.c: ERROR MAP_FAILED on gp600m\n");
    return (-1);
  }
  hard_l2i = (struct l2i*)((char*)hard_l2ct + 0x1000000);
  return (0);
}
/* 制御空間 */ Control registers for FPGA
void hard_l2ctw(i, d) Uint i; union l2ct d;
{ *(hard_l2ct+i) = d;}
union l2ct hard_l2ctr(i) Uint i;
{ return *(hard_l2ct+i);}
void hard_l2ctsync()
{ msync(hard_l2ct, BAR3_L2CT_REGNUM*4, MS_SYNC);}                    Sync
/* 画像空間 */ Pixel memory for FPGA
void hard_l2iw(i, d) Uint i; struct l2i *d;
{ Ull *src, *dst;
  src = d->d;
  dst = (hard_l2i+i)->d;
  dst[0] = src[0];  dst[1] = src[1];  dst[2] = src[2];  dst[3] = src[3];
  dst[4] = src[4];  dst[5] = src[5];  dst[6] = src[6];  dst[7] = src[7];
}
void hard_l2ir(i, d) Uint i; struct l2i *d;
{ Ull *src, *dst;
  src = (hard_l2i+i)->d;
  dst = d->d;
  dst[0] = src[0];  dst[1] = src[1];  dst[2] = src[2];  dst[3] = src[3];
  dst[4] = src[4];  dst[5] = src[5];  dst[6] = src[6];  dst[7] = src[7];
}
void hard_l2isync(i)
{ msync(hard_l2i+i, L2LINE, MS_SYNC);}                               Sync
```

Burst data transmission for high-performance









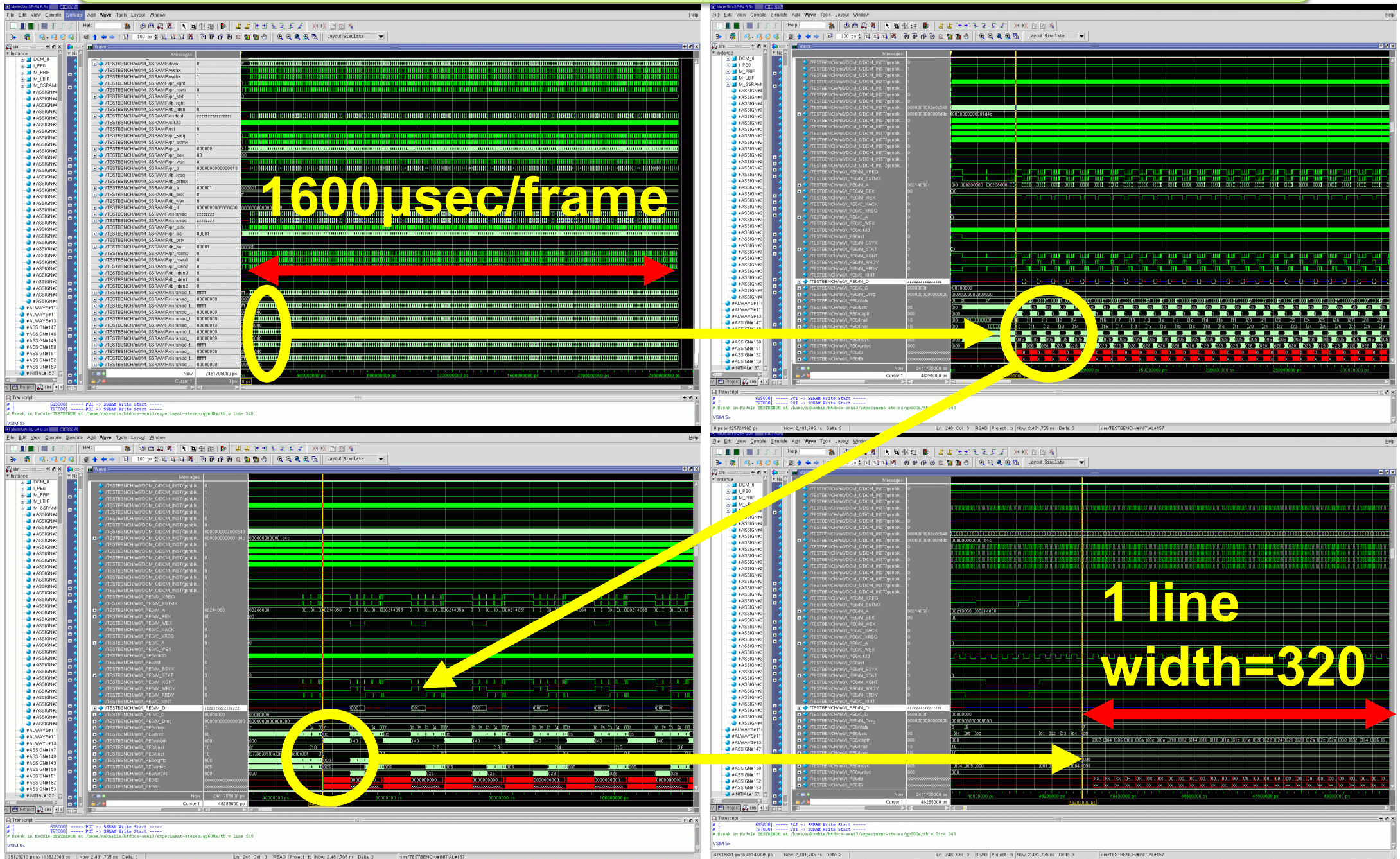
Hard-soft interface from FPGA


```
reg [15:0]      Bl0[319:0],...Blf[319:0]; /* 16bit×320画素×16行分のバッファ */
reg [15:0]      Br0[319:0],...Brf[319:0]; /* 16bit×320画素×16行分のバッファ */
reg [319:0]     El,Er;                    /*  1bit×320画素×  1行分のバッファ */
reg [7:0]       Dl[319:0], Dr[319:0];     /*  8bit×320画素×  1行分のバッファ */

/* Reset hardware */
/* Get command from CPU */
else if ((state==STATE_INIT) & (M_STAT[1:0]==2'b11)) begin
  state <= STATE_PFBL; /* start SSRAM -> internal_Bl */
  linel <= 0; /* 240回カウント */
  liner <= 0; /* 240回カウント */
  xgntc <= 0;
end
/* Read Bl, Br, El, Er */
else if ((state==STATE_PFBL) | (state==STATE_PFBR) | (state==STATE_RDBL) | (state==STATE_RDBR)) (省略)
else if ((state==STATE_RDEL) | (state==STATE_RDER)) (省略)
/* SAD body */
else if (state==STATE_EXEC) begin
/******** refer next page ********/
end
/* Write to Dl Dr */
else if ((state==STATE_WRDL) | (state==STATE_WRDR)) (省略)
/* Report to CPU */
else if (state==STATE_TERM) begin
  if (M_XREQ & xgntc==0) begin
    M_A[31:3]    <= 29'b00000000_00000000_00000000_00000___;
    M_Dreg[63:0] <= 64'h00000000_00000013; /* host_request */
    M_WEX        <= 1'b0;
    M_BSTMX      <= 1'b1;
    M_XREQ       <= 1'b0;
  end
  if (~M_XGNT) begin
    xgntc <= xgntc + 1;
    M_XREQ <= 1'b1;
  end
  if (M_WRDY) state <= STATE_INIT;
end
```

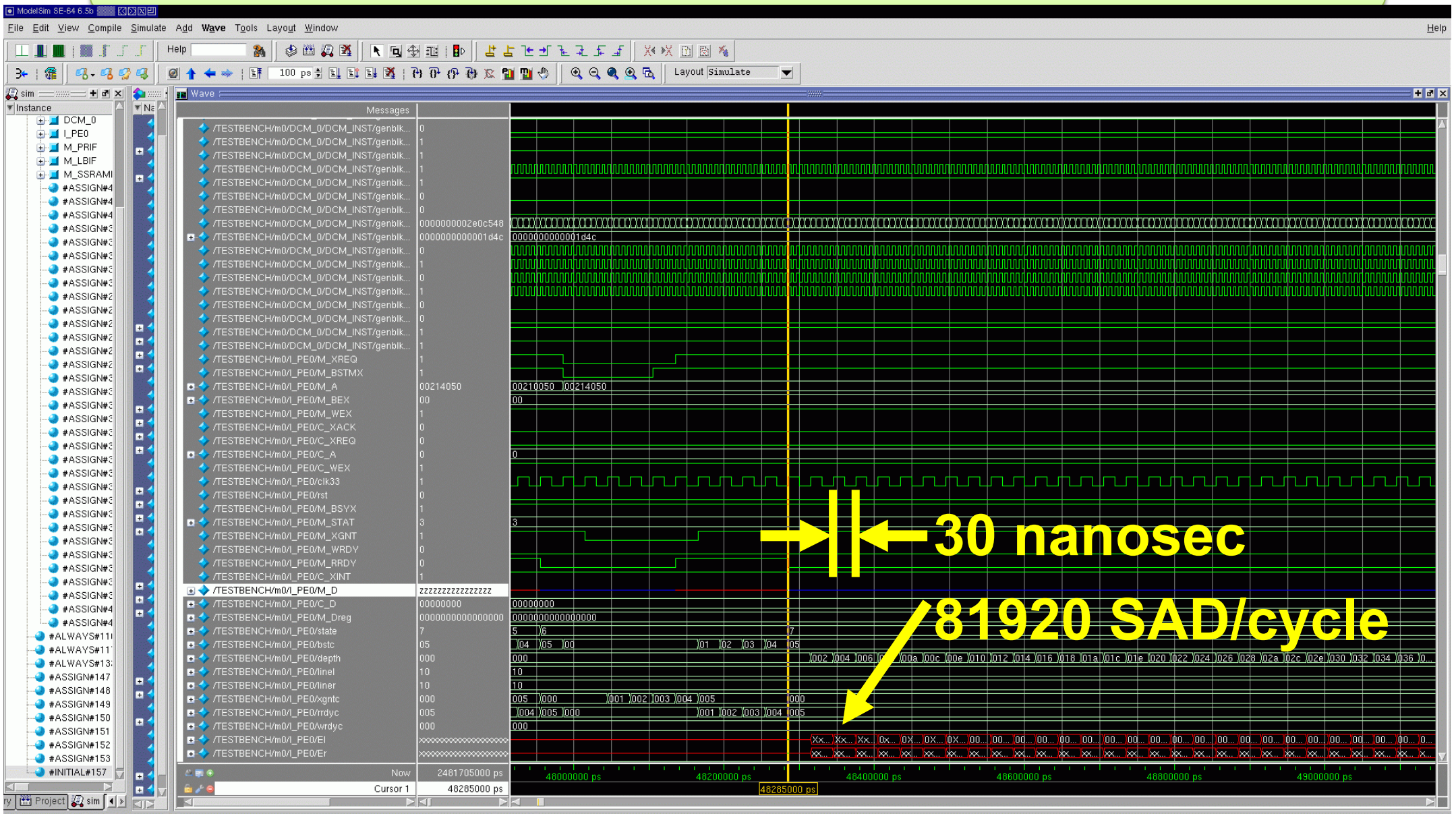
Kernel of SAD in FPGA

```
if (depth < DMAX*4) begin /* バーストモードのためにDMAXの4倍分実行する */
  for (i=48; i<=352; i=i+1) begin /* shift L/R pixels */
    /* Calculate Dl */
    if (~El[i-40] & (Dl[i-40] < Dl[i-41])) begin
      Dl[i-40] <= Dl[i-41]; /* if El=0, copy neighbor Dl */
    end
    /* 本来は16行分だがFPGAの容量制限により現在は2行分 */
    if (depth < DMAX && Sminl[i-40]>(sa0[i]+sa1[i])) begin
      /* if current SAD is minimum */
      if (~El[i-40] | Er[i-40]) begin  /* if El=0 or El and Er have edge, update minimum SAD */
        Sminl[i-40] <=(sa0[i]+sa1[i]);
      end
      if ( El[i-40] & Er[i-40]) begin  /* if El and Er have edge, store current "depth" to Dl */
        Dl[i-40] <= depth;
      end
    end
    /* Calculate Dr */
    if (~Er[i-40] & (Dr[i-40] < Dr[i-39])) begin
      Dr[i-40] <= Dr[i-39]; /* if Er=0, copy neighbor Dl */
    end
    /* 本来は16行分だがFPGAの容量制限により現在は2行分 */
    if (depth < DMAX && Sminr[i-40]>(sa0[i]+sa1[i])) begin
      /* if current SAD is minimum */
      if (~Er[i-40] | El[i-40]) begin  /* if Er=0 or El and Er have edge, update minimum SAD */
        Sminr[i-40] <=(sa0[i]+sa1[i]);
      end
      if ( Er[i-40] & El[i-40]) begin  /* if El and Er have edge, store current "depth" to Dr */
        Dr[i-40] <= depth;
      end
    end
  end
end
/* Finalize */
if (depth < DMAX*4) begin
  depth <= depth+2;
  for (i=0; i<319; i=i+1) El[i] <= El[i+1]; El[319] <= 1'b0;
  for (i=0; i<319; i=i+1) Er[i+1] <= Er[i]; Er[0]   <= 1'b0;
end
else if (depth == DMAX*4) state <= STATE_WRDL;
```

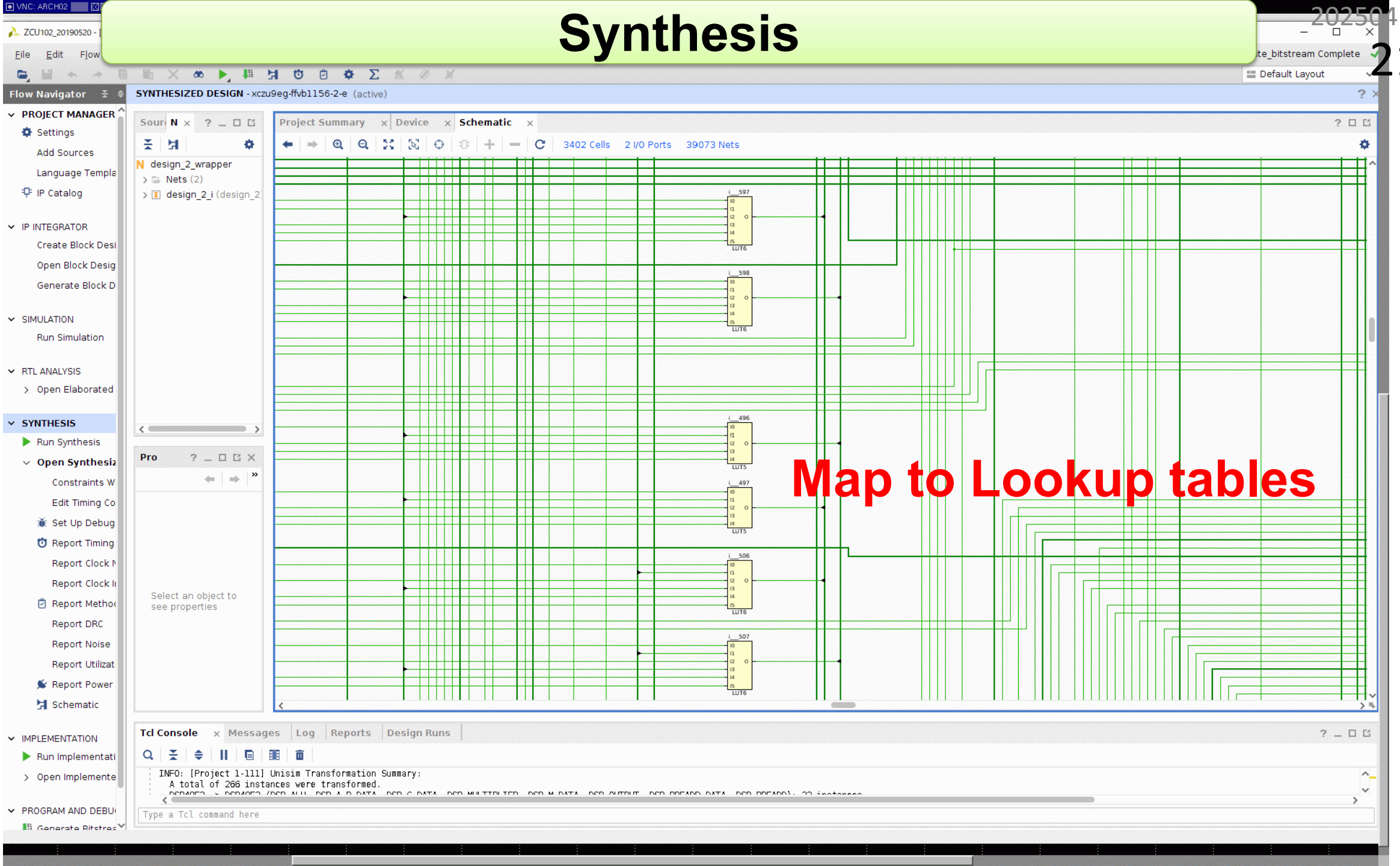
RTL Simulation



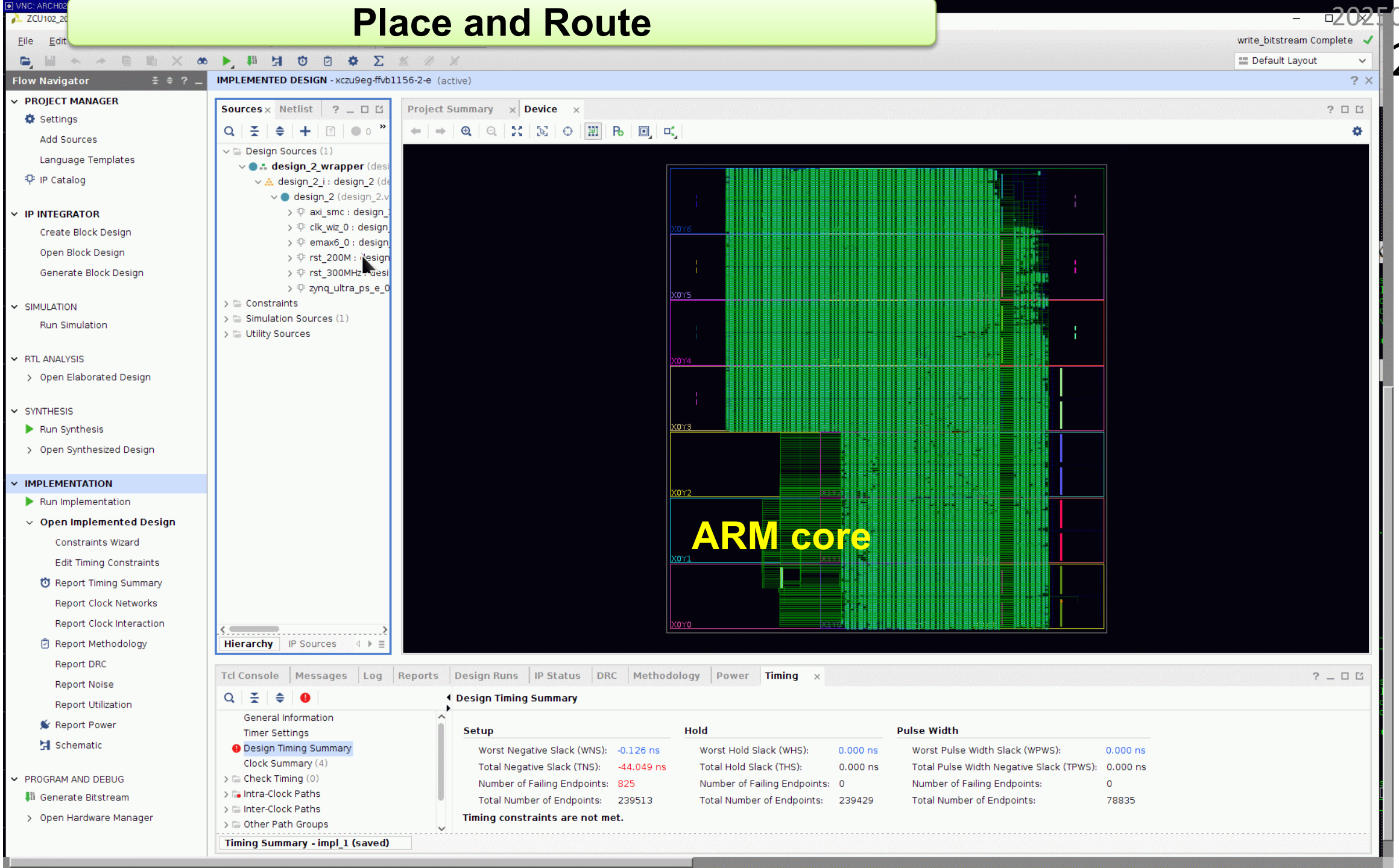
RTL Simulation



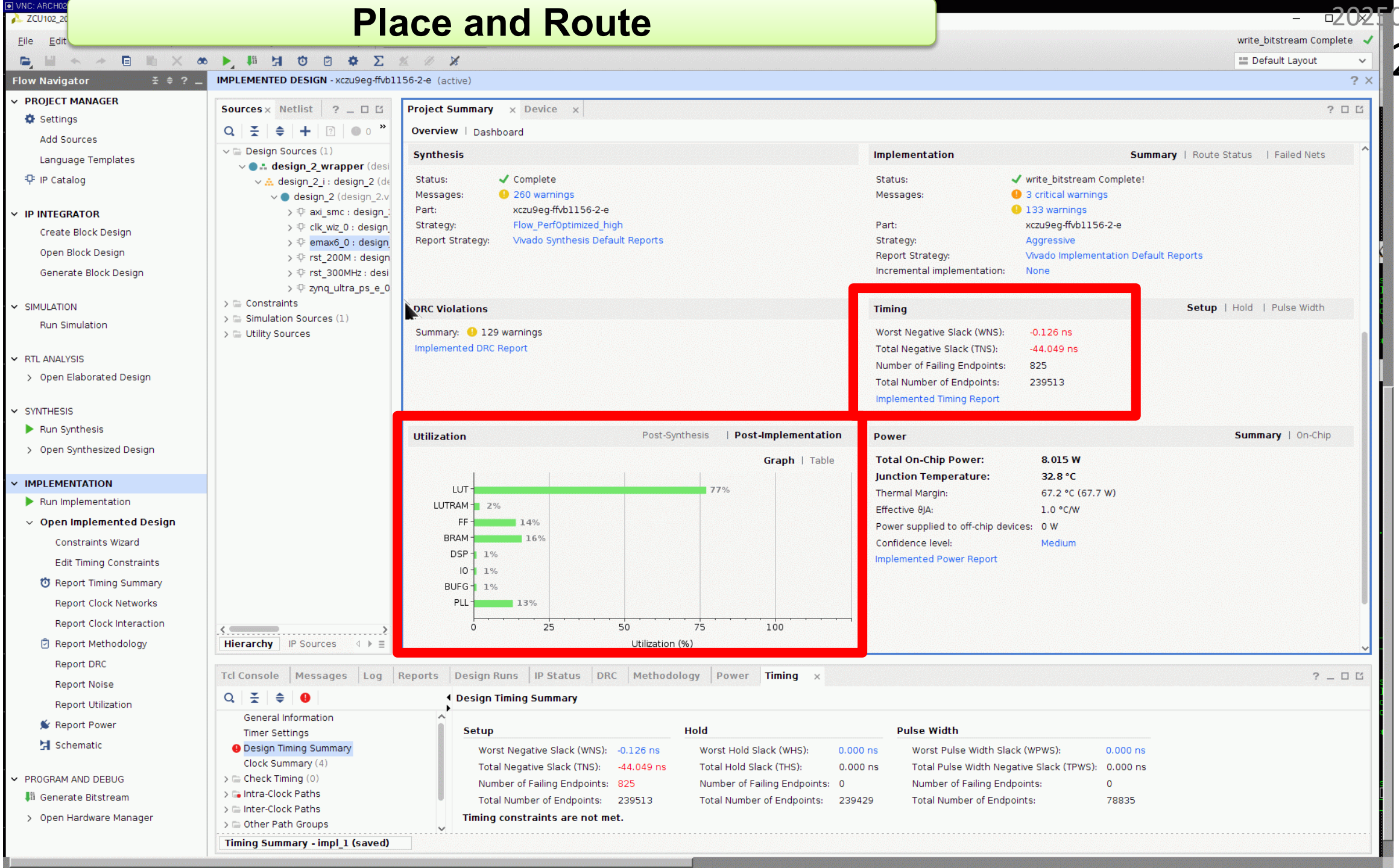
Synthesis


Flow Navigator
PROJECT MANAGER
Settings
Add Sources
Language Templa
IP Catalog
IP INTEGRATOR
Create Block Desi
Open Block Desig
Generate Block D
SIMULATION
Run Simulation
RTL ANALYSIS
Open Elaborated
SYNTHESIS
Run Synthesis
Open Synthesiz
Constraints W
Edit Timing Co
Set Up Debug
Report Timing
Report Clock N
Report Clock I
Report Metho
Report DRC
Report Noise
Report Utilizat
Report Power
Schematic
IMPLEMENTATION
Run Implementati
Open Implemente
PROGRAM AND DEBU
SYNTHESIZED DESIGN - xczu9eg-ffvb1156-2-e (active)
Project Summary
Device
Schematic
3402 Cells
2 I/O Ports
39073 Nets
design_2_wrapper
Nets (2)
design_2_i
Select an object to see properties
Map to Lookup tables
Tcl Console
Messages
Log
Reports
Design Runs
INFO: [Project 1-111] Unisim Transformation Summary:
A total of 266 instances were transformed.
Type a Tcl command here

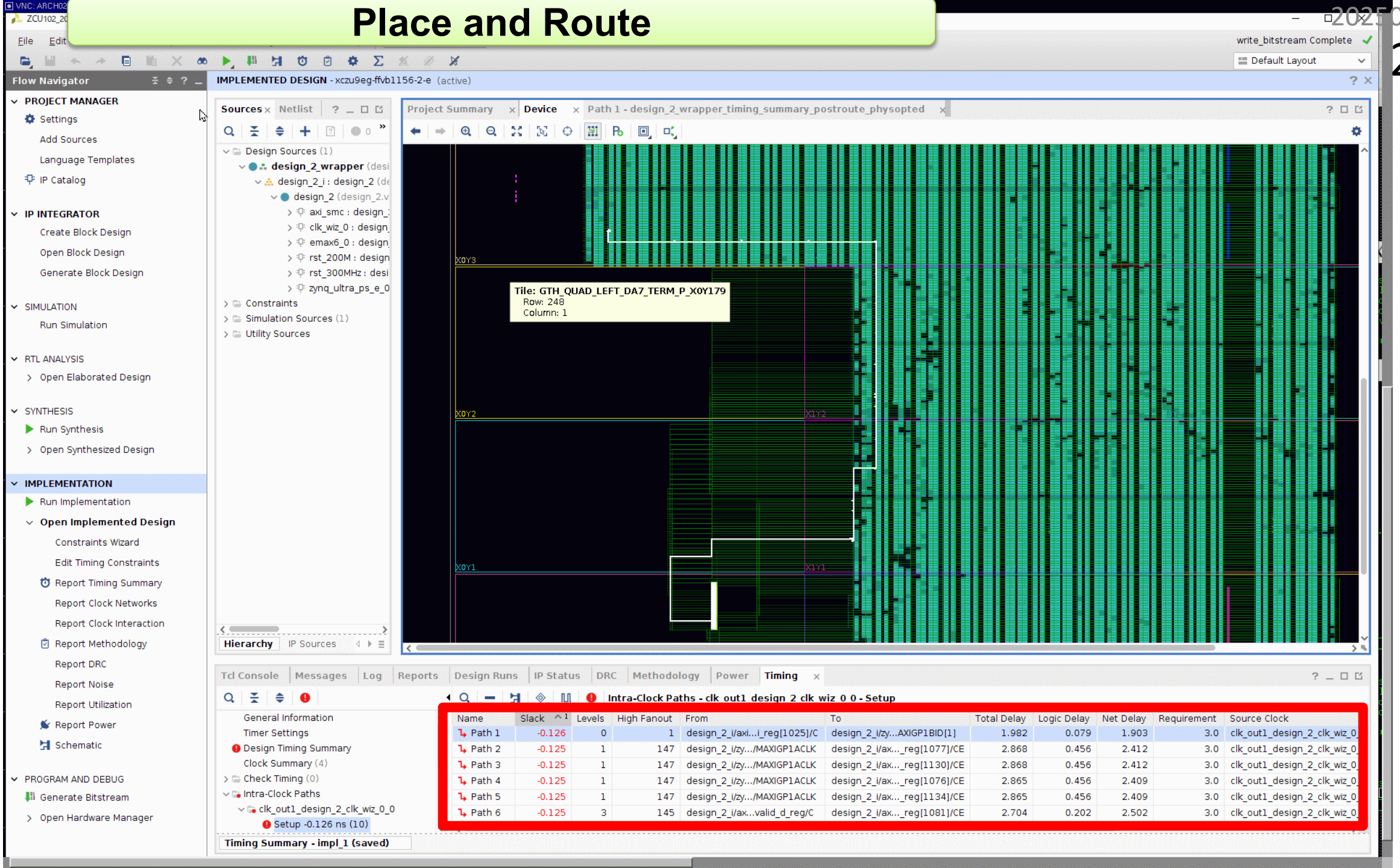
Place and Route



Place and Route

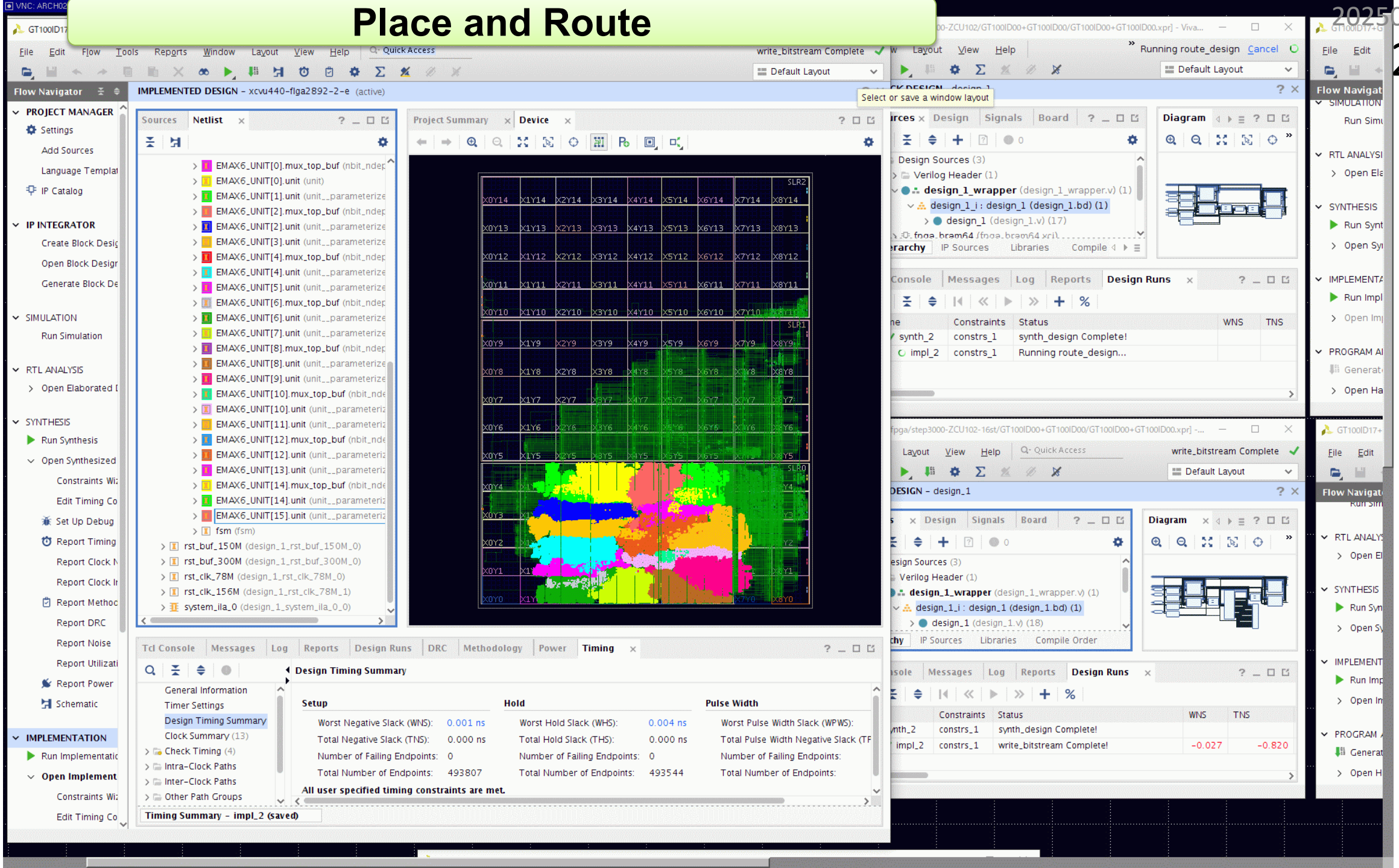


Place and Route



Name	Slack	Levels	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock
Path 1	-0.126	0	1	design_2_i/axi...i_reg[1025]/C	design_2_i/zy...AXIGP1BID[1]	1.982	0.079	1.903	3.0	clk_out1_design_2_clk_wiz_0
Path 2	-0.125	1	147	design_2_i/zy.../MAXIGP1ACLK	design_2_i/ax..._reg[1077]/CE	2.868	0.456	2.412	3.0	clk_out1_design_2_clk_wiz_0
Path 3	-0.125	1	147	design_2_i/zy.../MAXIGP1ACLK	design_2_i/ax..._reg[1130]/CE	2.868	0.456	2.412	3.0	clk_out1_design_2_clk_wiz_0
Path 4	-0.125	1	147	design_2_i/zy.../MAXIGP1ACLK	design_2_i/ax..._reg[1076]/CE	2.865	0.456	2.409	3.0	clk_out1_design_2_clk_wiz_0
Path 5	-0.125	1	147	design_2_i/zy.../MAXIGP1ACLK	design_2_i/ax..._reg[1134]/CE	2.865	0.456	2.409	3.0	clk_out1_design_2_clk_wiz_0
Path 6	-0.125	3	145	design_2_i/ax...valid_d_reg/C	design_2_i/ax..._reg[1081]/CE	2.704	0.202	2.502	3.0	clk_out1_design_2_clk_wiz_0

Place and Route



CGRA

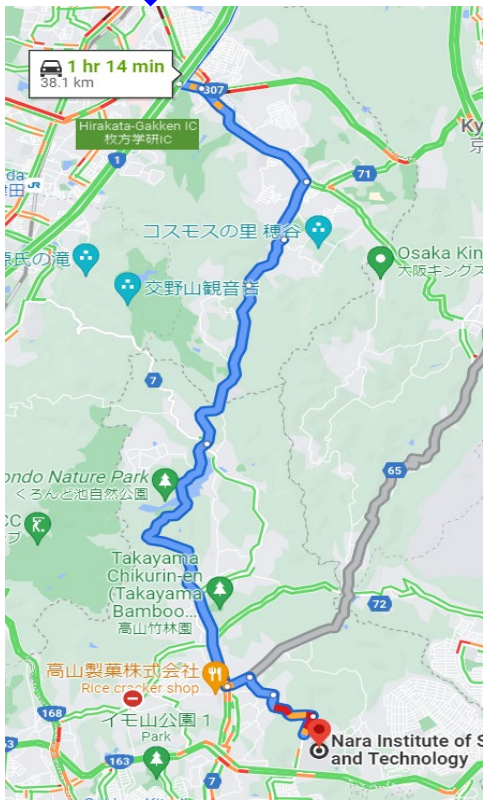
Coarse Grained Reconfigurable Array

NAIST

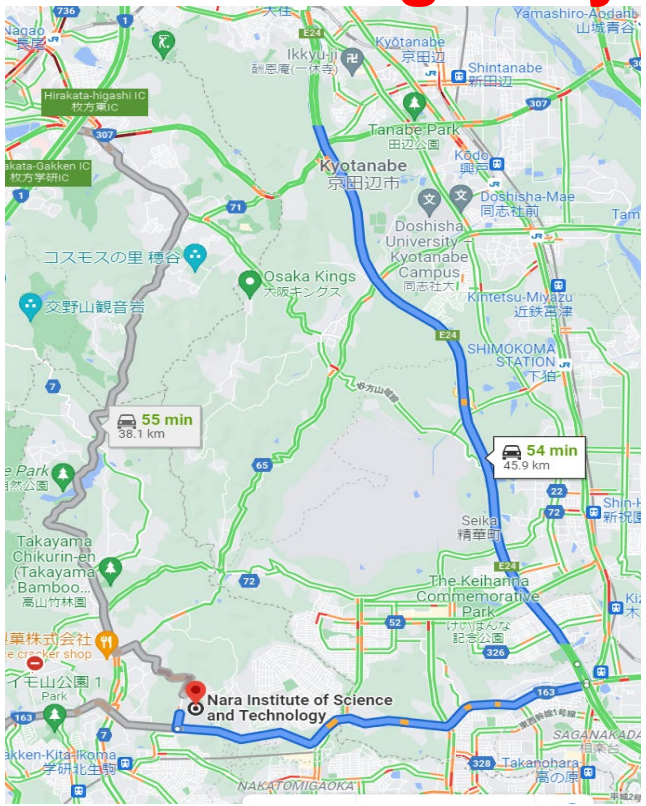
CGRA is similar to stream of water

Coarse Grained Reconfigurable Array
Other name: Reconfigurable Data Path (RDP)

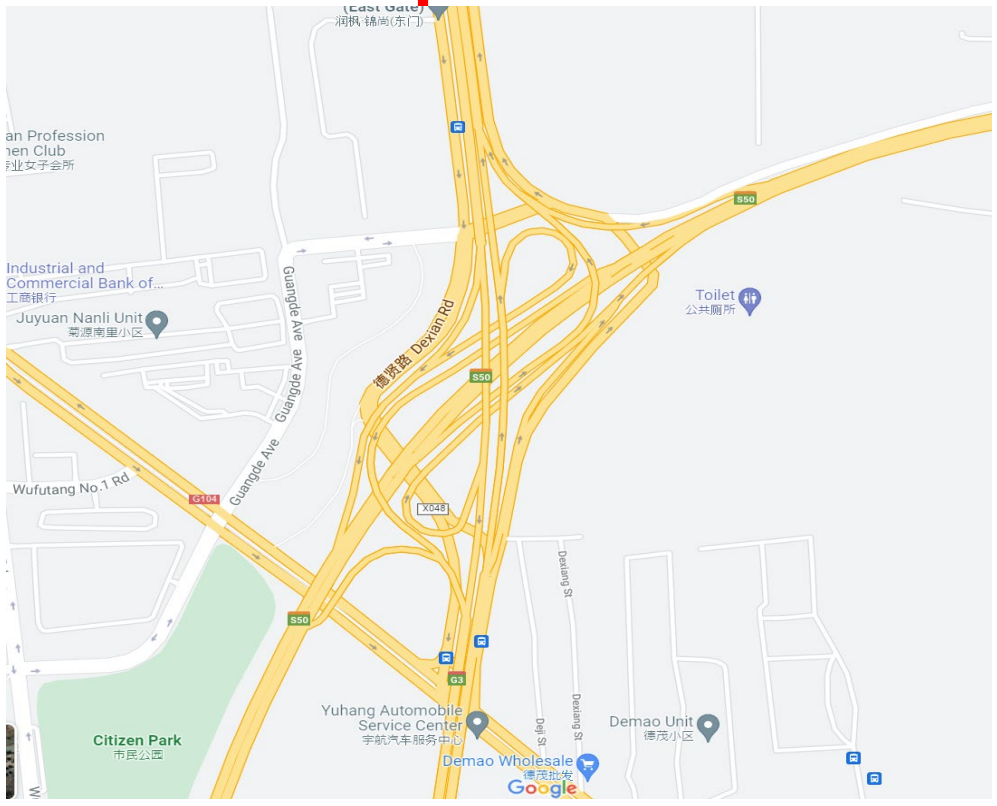
Von Neumann computers have many traffic signals



CGRA is highway

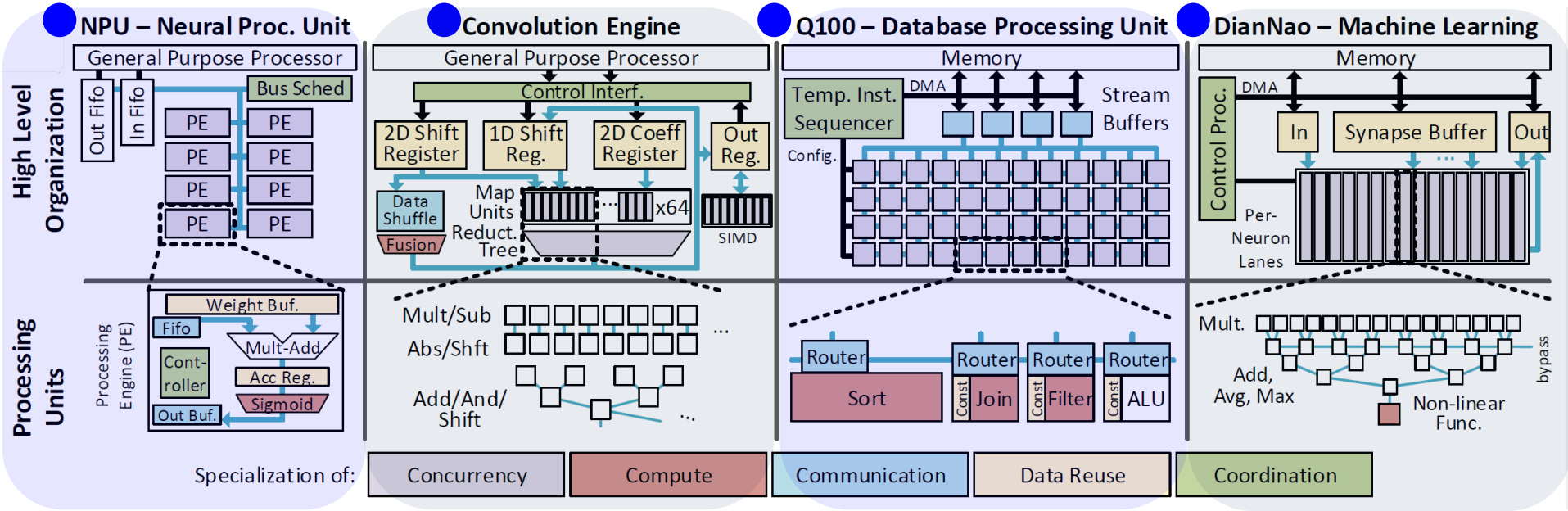


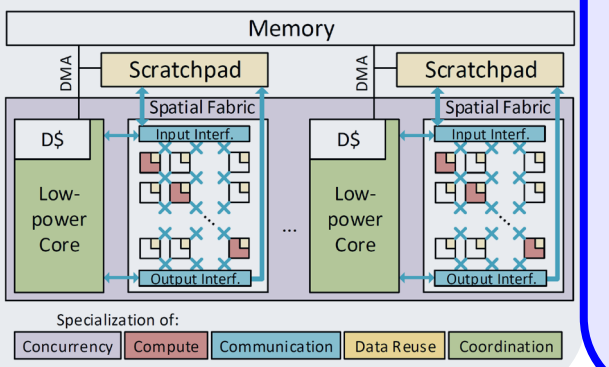
Non-stop execution

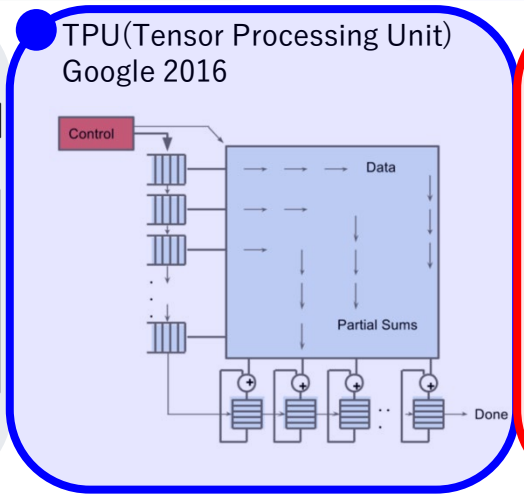


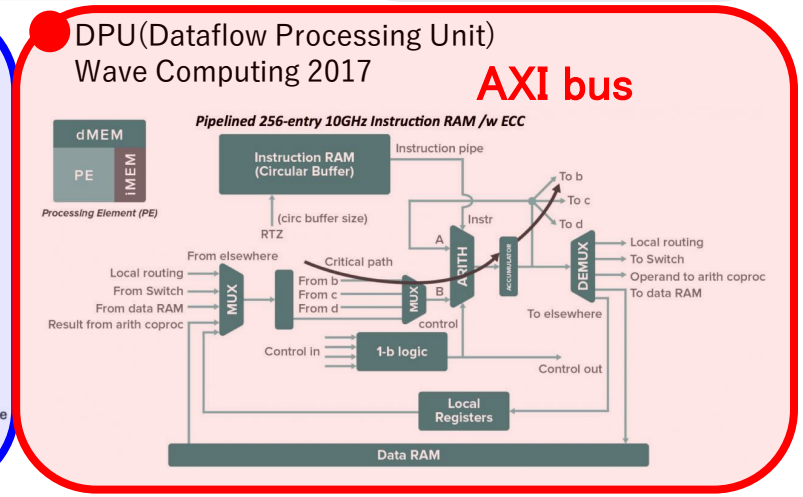
Systolic type DSA(Domain Specific Architecture) is emerging

- 2nd generation: ALU + external memory … wide memory bus
- 3rd generation: Network of near-memory ALUs … narrow memory bus

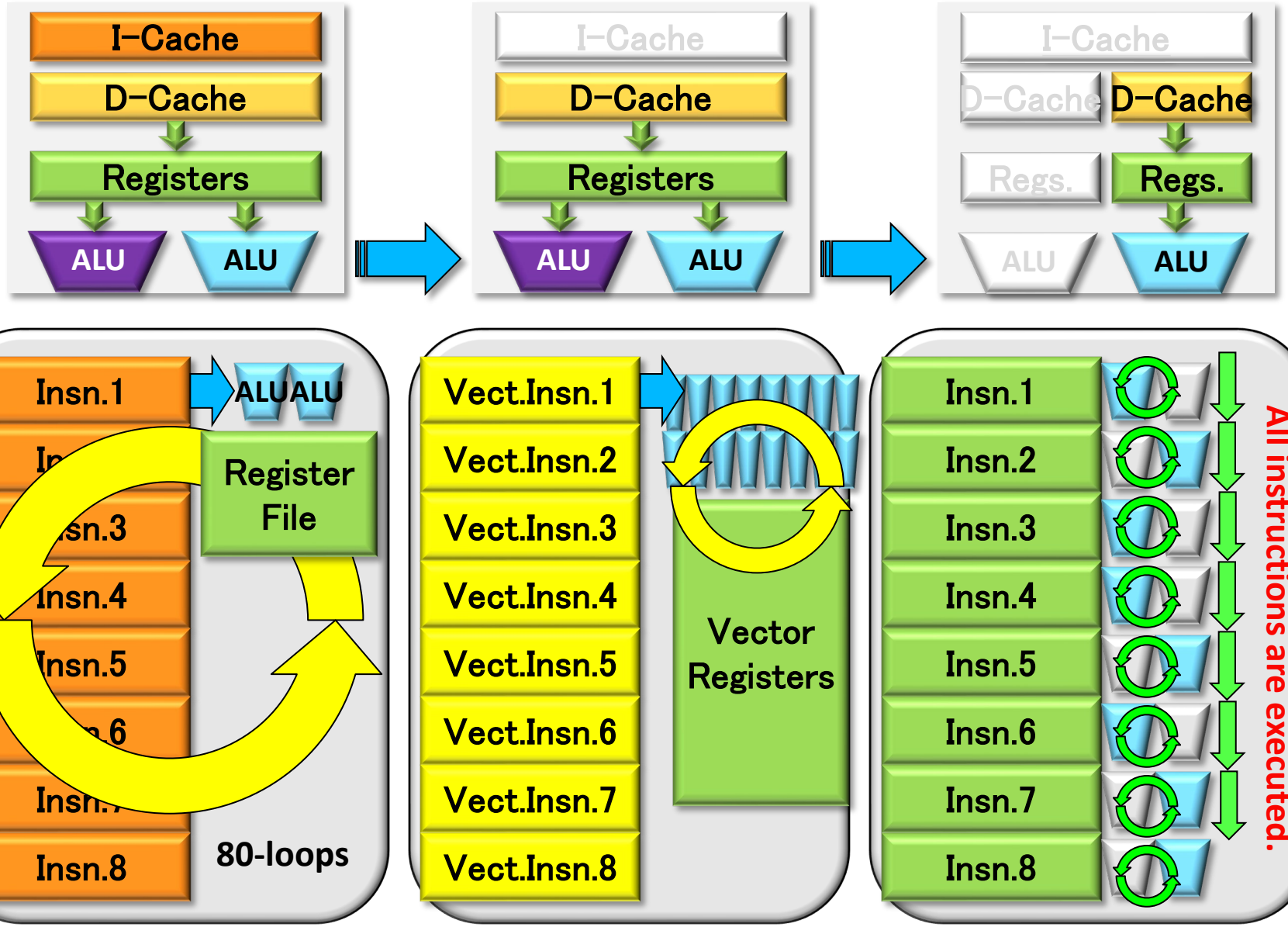




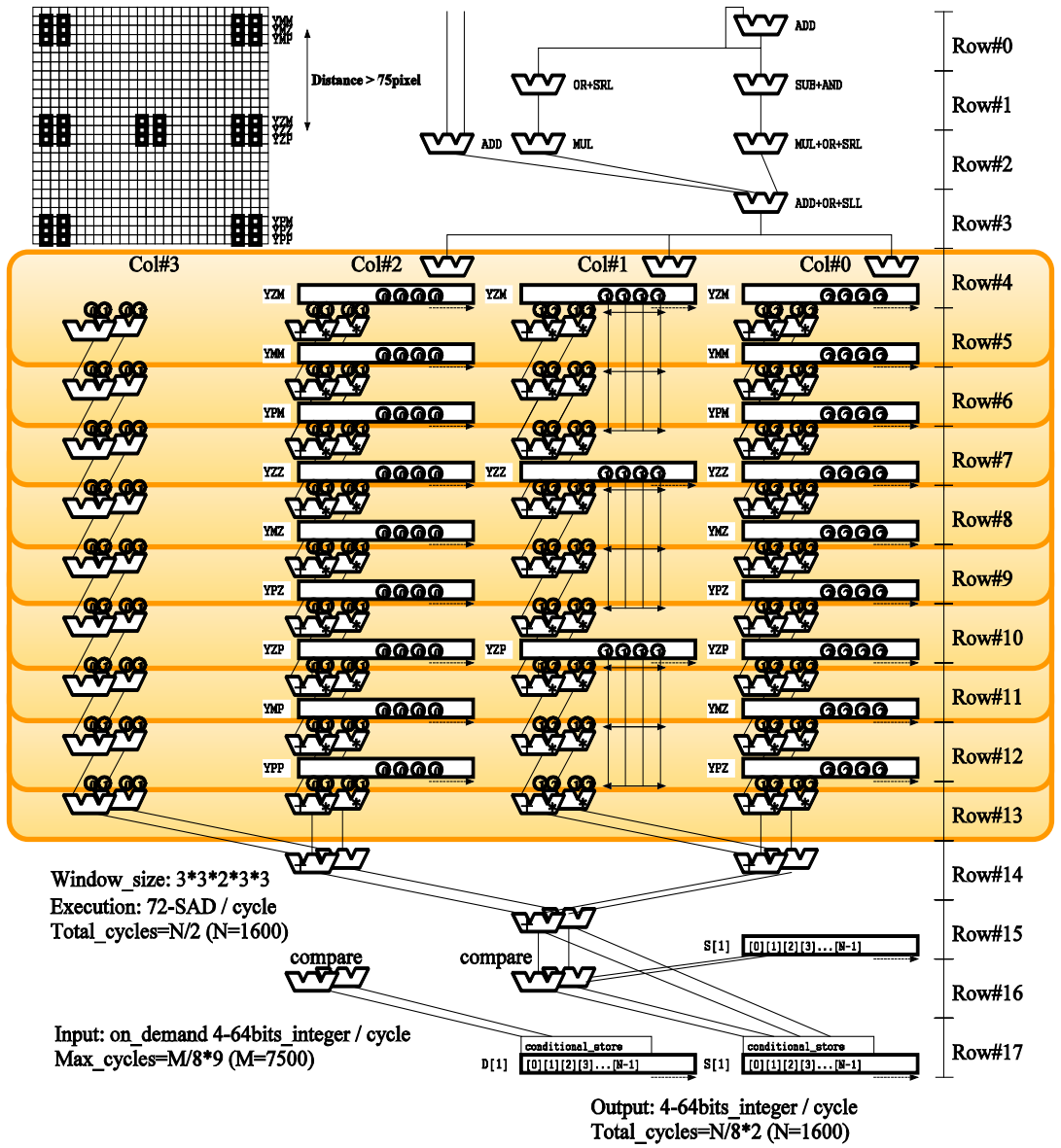


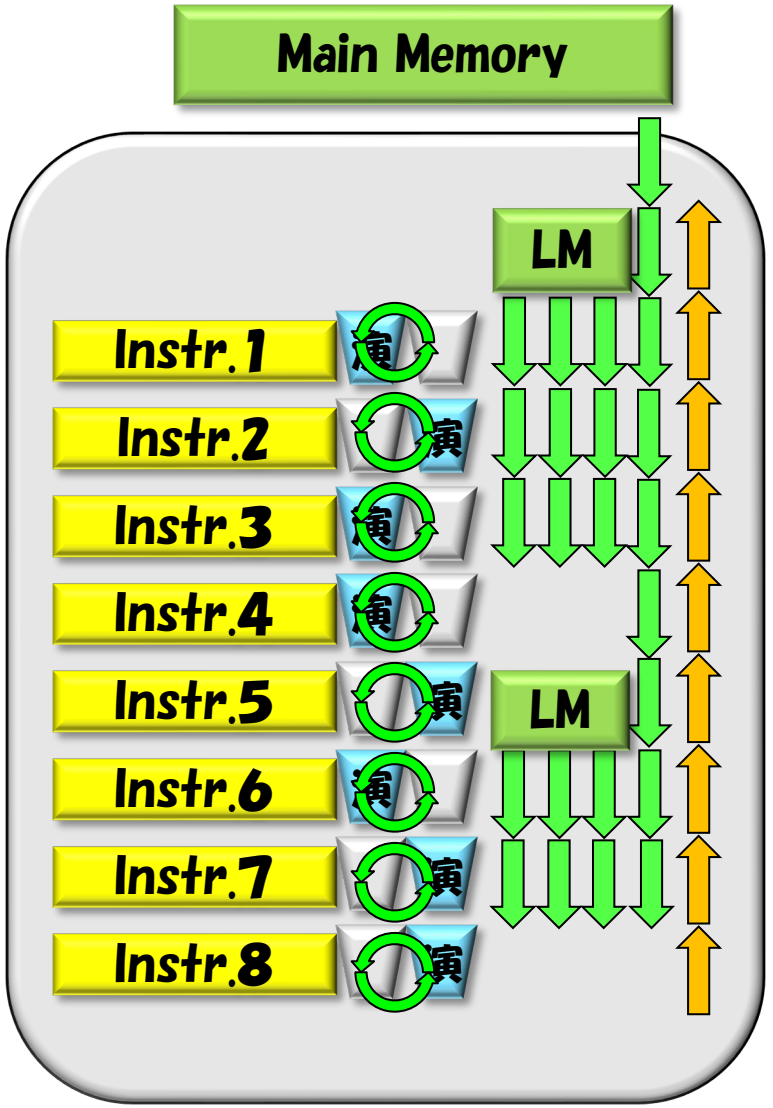


Basic Idea of CGRA

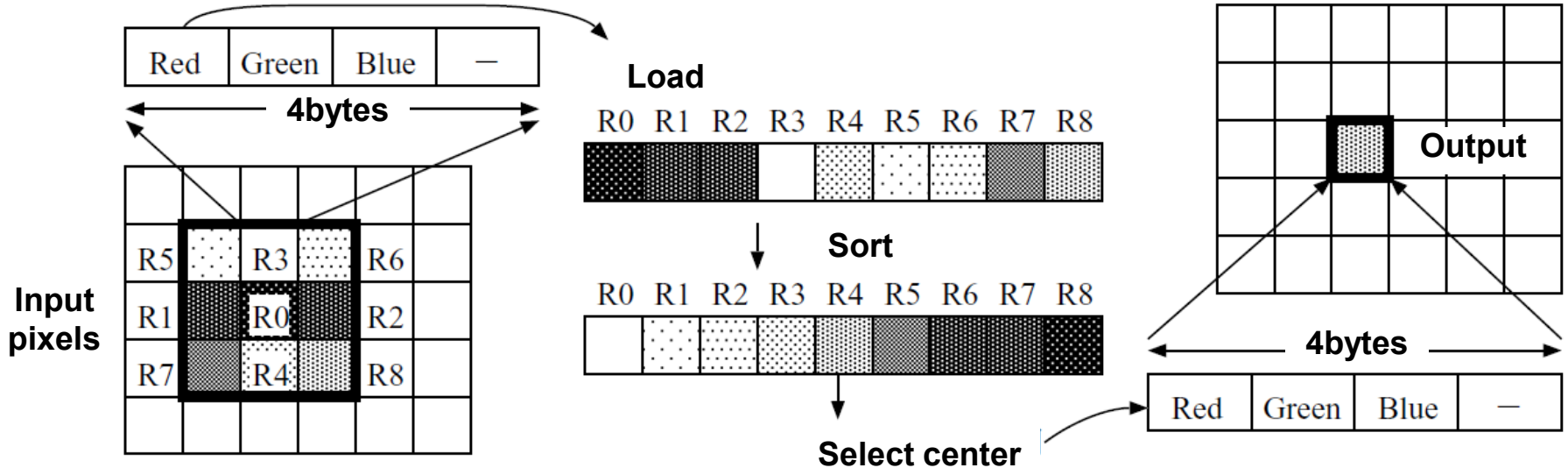


Basic structure of CGRA



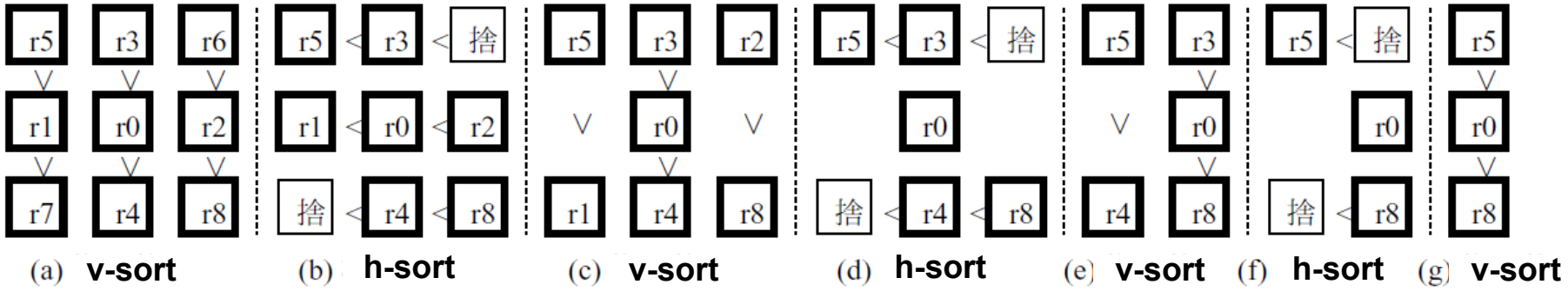


Median filter with CGRA




```
1. unsigned int *in, *out;
2. unsigned char R[9], G[9], B[9], t;
3. R[0] = *(in)>>24; R[1] = *(in−1)>>24; R[2] = *(in+1)>>24; ...; R[8] = ...;
4. G[0] = *(in)>>16; G[1] = *(in−1)>>16; G[2] = *(in+1)>>16; ...; G[8] = ...;
5. B[0] = *(in)>> 8; B[1] = *(in−1)>> 8; B[2] = *(in+1)>> 8; ...; B[8] = ...;
6. for (i=8; i≥4; i−−) /* i を 8 から 4 まで変化させる */
7.  for (j=0; j<i; j++) { /* j を 0 から i-1 まで変化させる */
8.   if (R[j] > R[j+1]) { t = R[j]; R[j] = R[j+1]; R[j+1] = t;} /* R[j] と R[j+1] を交換 */
9.   if (G[j] > G[j+1]) { t = G[j]; G[j] = G[j+1]; G[j+1] = t;} /* G[j] と G[j+1] を交換 */
10.   if (B[j] > B[j+1]) { t = B[j]; B[j] = B[j+1]; B[j+1] = t;} /* B[j] と B[j+1] を交換 */
11.  } /* バブルソートによる RGB 毎の整列が完了 */
12. *out = (R[4]≪24) | (G[4]≪16) | (B[4]≪8); /* 1 画素を出力 */
```

Special instruction for median filter



`max3(A,B,C),mid3(A,B,C),min3(A,B,C)` … Select max/mid/min value from three 8bits

`max2(A,B,C),min2(A,B,C)` … Select max/min value from two 8bits

```
 1. unsigned int *in, *out;
 2. unsigned int r0, r1, r2, r3, r4, r5, r6, r7, r8;
 3. r0 = *(in); r1 = *(in-1); r2 = *(in+1); ...; r8 = ...;
 4. t0 = max3(r5,r1,r7); t1 = mid3(r5,r1,r7); t2 = min3(r5,r1,r7); r5=t0; r1=t1; r7=t2;
 5. t0 = max3(r3,r0,r4); t1 = mid3(r3,r0,r4); t2 = min3(r3,r0,r4); r3=t0; r0=t1; r4=t2;
 6. t0 = max3(r6,r2,r8); t1 = mid3(r6,r2,r8); t2 = min3(r6,r2,r8); r6=t0; r2=t1; r8=t2;
 7. t0 = min3(r5,r3,r6); t1 = mid3(r5,r3,r6); r5=t0; r3=t1;
 8. t0 = min3(r1,r0,r2); t1 = mid3(r1,r0,r2); t2 = max3(r1,r0,r2); r1=t0; r0=t1; r2=t2;
 9. t0 = mid3(r7,r4,r8); t1 = max3(r7,r4,r8); r4=t0; r8=t1;
10. t0 = max2(r5,r1);    t1 = min2(r5,r1);    r5=t0; r1=t1;
11. t0 = max3(r3,r0,r4); t1 = mid3(r3,r0,r4); t2 = min3(r3,r0,r4); r3=t0; r0=t1; r4=t2;
12. t0 = max2(r2,r8);    t1 = min2(r2,r8);    r2=t0; r8=t1;
13. t0 = min3(r5,r3,r2); t1 = mid3(r5,r3,r2); r5=t0; r3=t1;
14. t0 = mid3(r1,r4,r8); t1 = max3(r1,r4,r8); r4=t0; r8=t1;
15. t0 = max2(r5,r4);    t1 = min2(r5,r4);    r5=t0; r4=t1;
16. t0 = max3(r3,r0,r8); t1 = mid3(r3,r0,r8); t2 = min3(r3,r0,r8); r3=t0; r0=t1; r8=t2;
17. r5 = min2(r5,r3);    r8 = max2(r4,r8);
18. *out = mid3(r5,r0,r8); /* 1 画素を出力 */
```


NAIST

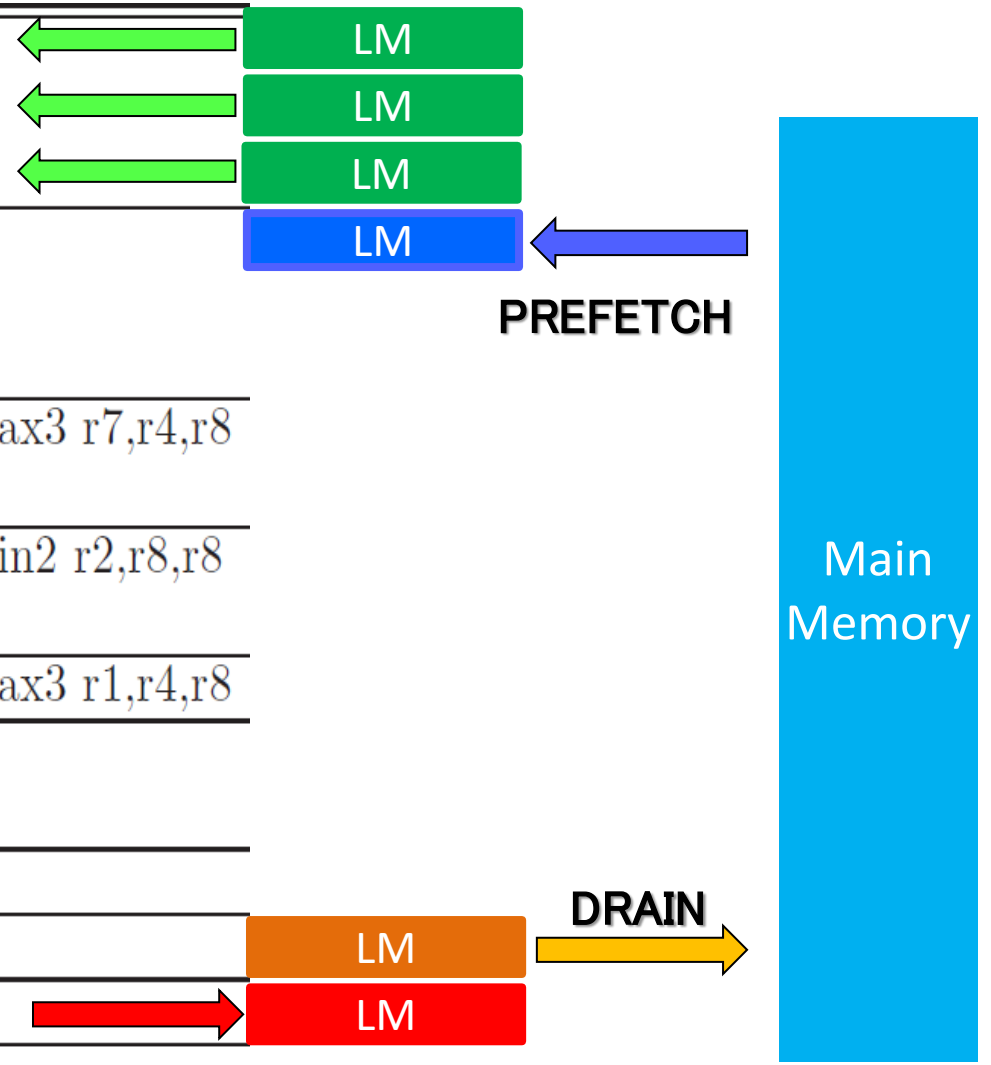
VLIW format instructions

Unlike VLIW, all instructions are executed simultaneously

	1.	load (in),r0;	load (in-4),r1;	load (in+4),r2	
	2.	load (..),r3;	load (....),r4;	load (....),r5	
	3.	load (..),r6;	load (....),r7;	load (....),r8	
(a)	4.	max3 r1,r7,r5;	mid3 r7,r5,r1;	min3 r5,r1,r7	
	5.	max3 r0,r4,r3;	mid3 r4,r3,r0;	min3 r3,r0,r4	
	6.	max3 r2,r8,r6;	mid3 r8,r6,r2;	min3 r6,r2,r8	
(b)	7.	min3 r3,r6,r5;	mid3 r6,r5,r3;	mid3 r8,r7,r4;	max3 r7,r4,r8
	8.	min3 r0,r2,r1;	mid3 r2,r1,r0;	max3 r1,r0,r2	
(c)	9.	max2 r5,r1,r5;	min2 r5,r1,r1;	max2 r2,r8,r2;	min2 r2,r8,r8
	10.	max3 r0,r4,r3;	mid3 r4,r3,r0;	min3 r3,r0,r4	
(d)	11.	min3 r3,r2,r5;	mid3 r2,r5,r3;	mid3 r8,r1,r4;	max3 r1,r4,r8
(e)	12.	max2 r5,r4,r5;	min2 r5,r4,r4		
	13.	max3 r0,r8,r3;	mid3 r8,r3,r0;	min3 r3,r0,r8	
(f)	14.	min2 r5,r3,r5;	max2 r4,r8,r8		
(g)	15.	mid3 r8,r5,r0			
	16.	store (out),r0			

Connecting operations to local memory

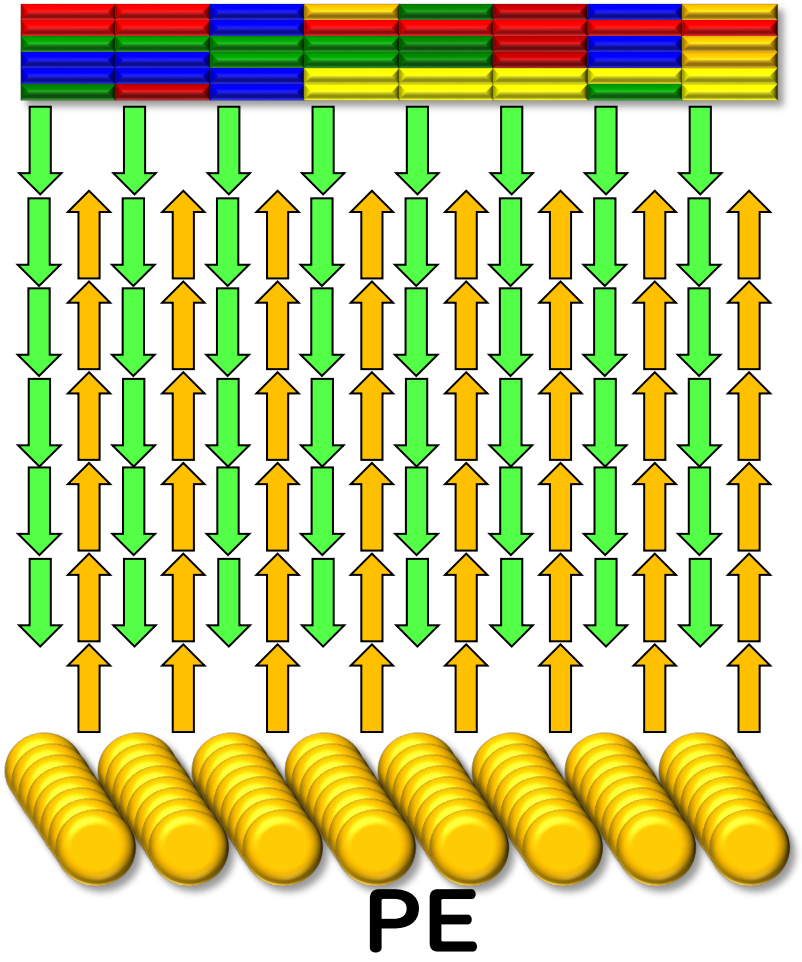
1.	load (in),r0;	load (in-4),r1;	load (in+4),r2	
2.	load (..),r3;	load (....),r4;	load (....),r5	
3.	load (..),r6;	load (....),r7;	load (....),r8	
4.	max3 r1,r7,r5;	mid3 r7,r5,r1;	min3 r5,r1,r7	
5.	max3 r0,r4,r3;	mid3 r4,r3,r0;	min3 r3,r0,r4	
6.	max3 r2,r8,r6;	mid3 r8,r6,r2;	min3 r6,r2,r8	
7.	min3 r3,r6,r5;	mid3 r6,r5,r3;	mid3 r8,r7,r4;	max3 r7,r4,r8
8.	min3 r0,r2,r1;	mid3 r2,r1,r0;	max3 r1,r0,r2	
9.	max2 r5,r1,r5;	min2 r5,r1,r1;	max2 r2,r8,r2;	min2 r2,r8,r8
10.	max3 r0,r4,r3;	mid3 r4,r3,r0;	min3 r3,r0,r4	
11.	min3 r3,r2,r5;	mid3 r2,r5,r3;	mid3 r8,r1,r4;	max3 r1,r4,r8
12.	max2 r5,r4,r5;	min2 r5,r4,r4		
13.	max3 r0,r8,r3;	mid3 r8,r3,r0;	min3 r3,r0,r8	
14.	min2 r5,r3,r5;	max2 r4,r8,r8		
15.	mid3 r8,r5,r0			
16.	store (out),r0			



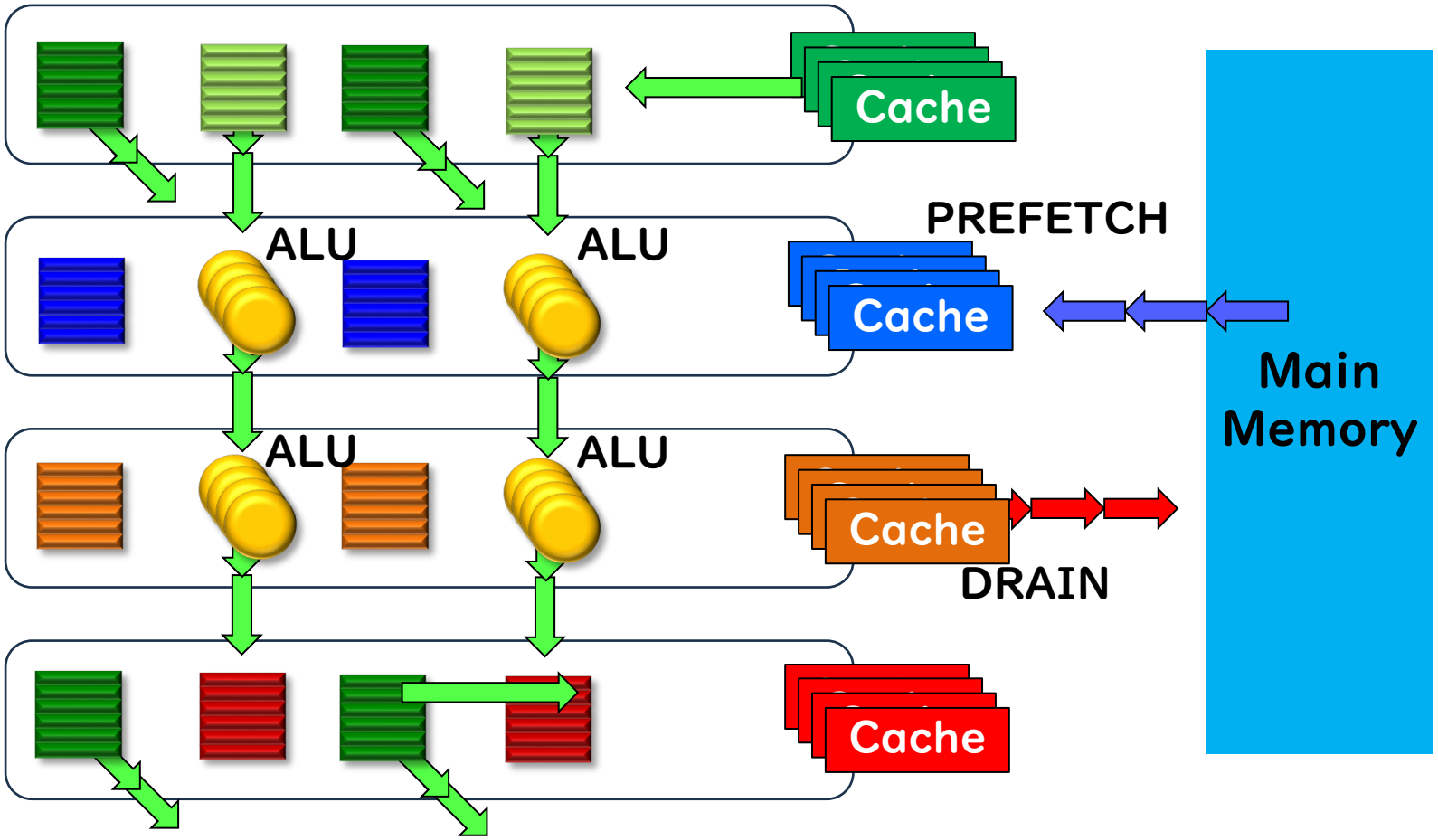
2008 LAPP: VLIW compatible (no compiler for CGRA)

Pipeline processing is performed like running water.

SIMD/SIMT

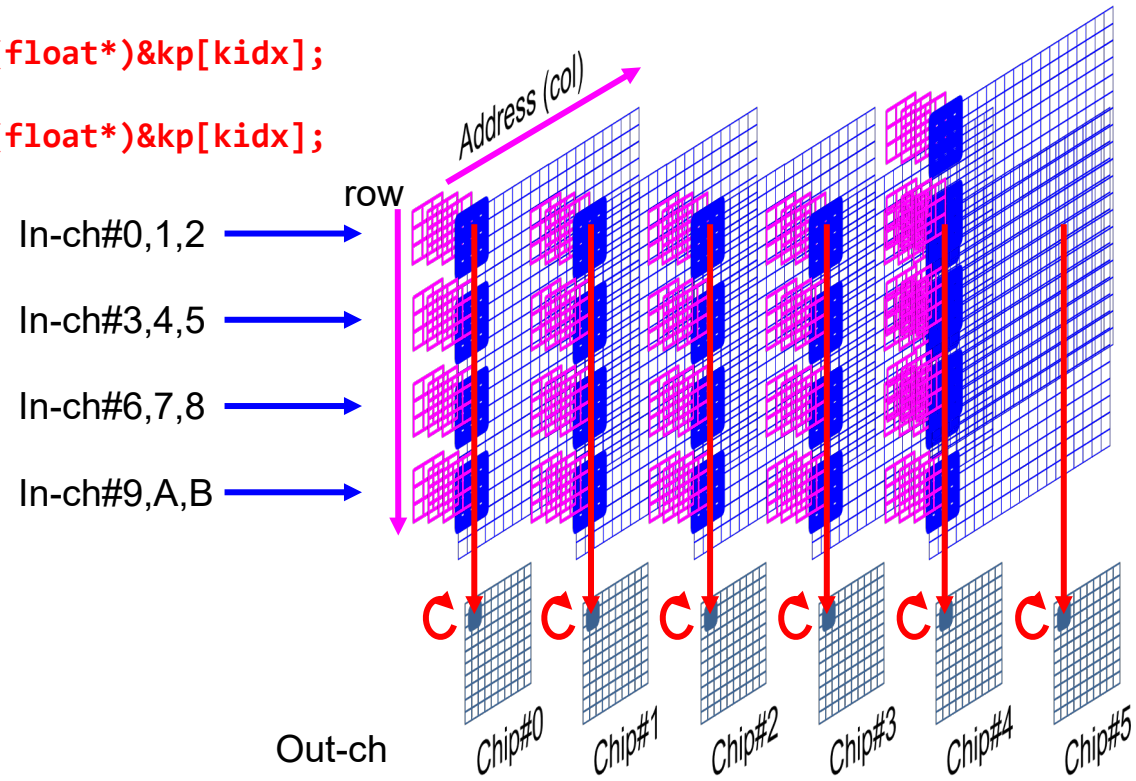


CGRA

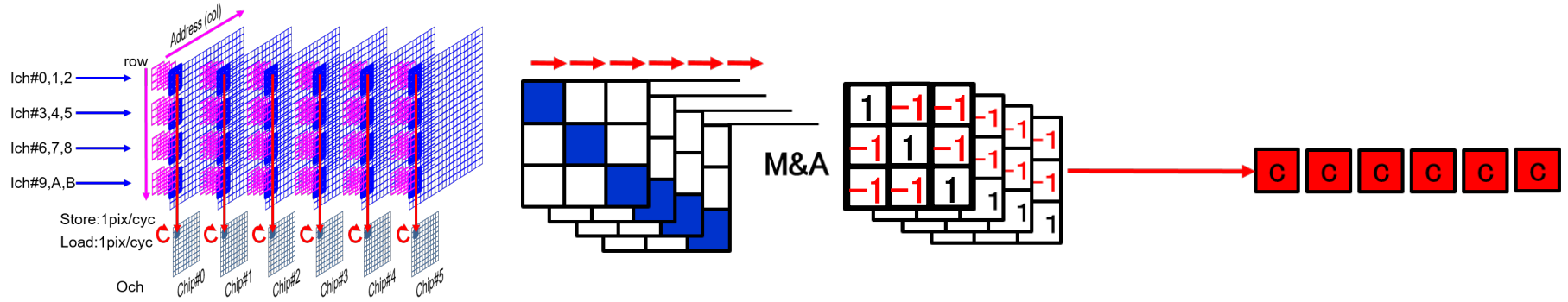


Example: Typical Convolutional Neural Network

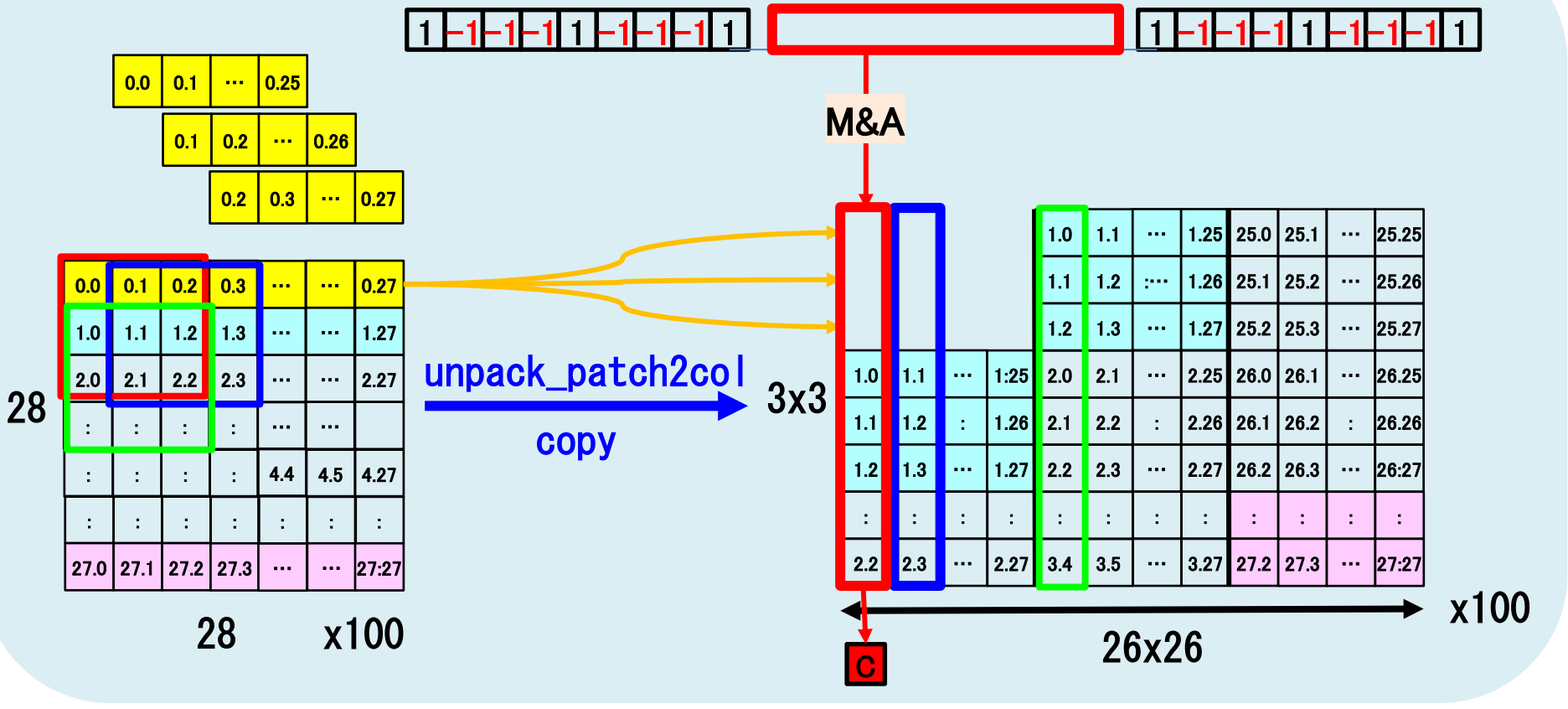
```
for (ic=0; ic<IC; ic++) { /* set input channel */
  ip0 = &in[ic*M*M]; /* top of input */
  for (row=1; row<M-1; row++) { /* image loop */
    for (col=1; col<M-1; col++) {
      for (oc=0; oc<OC; oc++) { /* set output channel */
        op = &out0[oc*M*M+row*M+col]; /* top of output */
        kp = &ker[(oc*IC+ic)*K*K];
        kidx = 0;
        for (y=-((K-1)/2); y<=(K-1)/2; y++) { /* kernel loop */
          for (x=-((K-1)/2); x<=(K-1)/2; x++) {
            if (ic == 0 && kidx == 0)
              *(float*)&*op  = *(float*)&ip0[(row+y)*M+col+x] * *(float*)&kp[kidx];
            else
              *(float*)&*op += *(float*)&ip0[(row+y)*M+col+x] * *(float*)&kp[kidx];
            kidx++;
} } } } } }
```



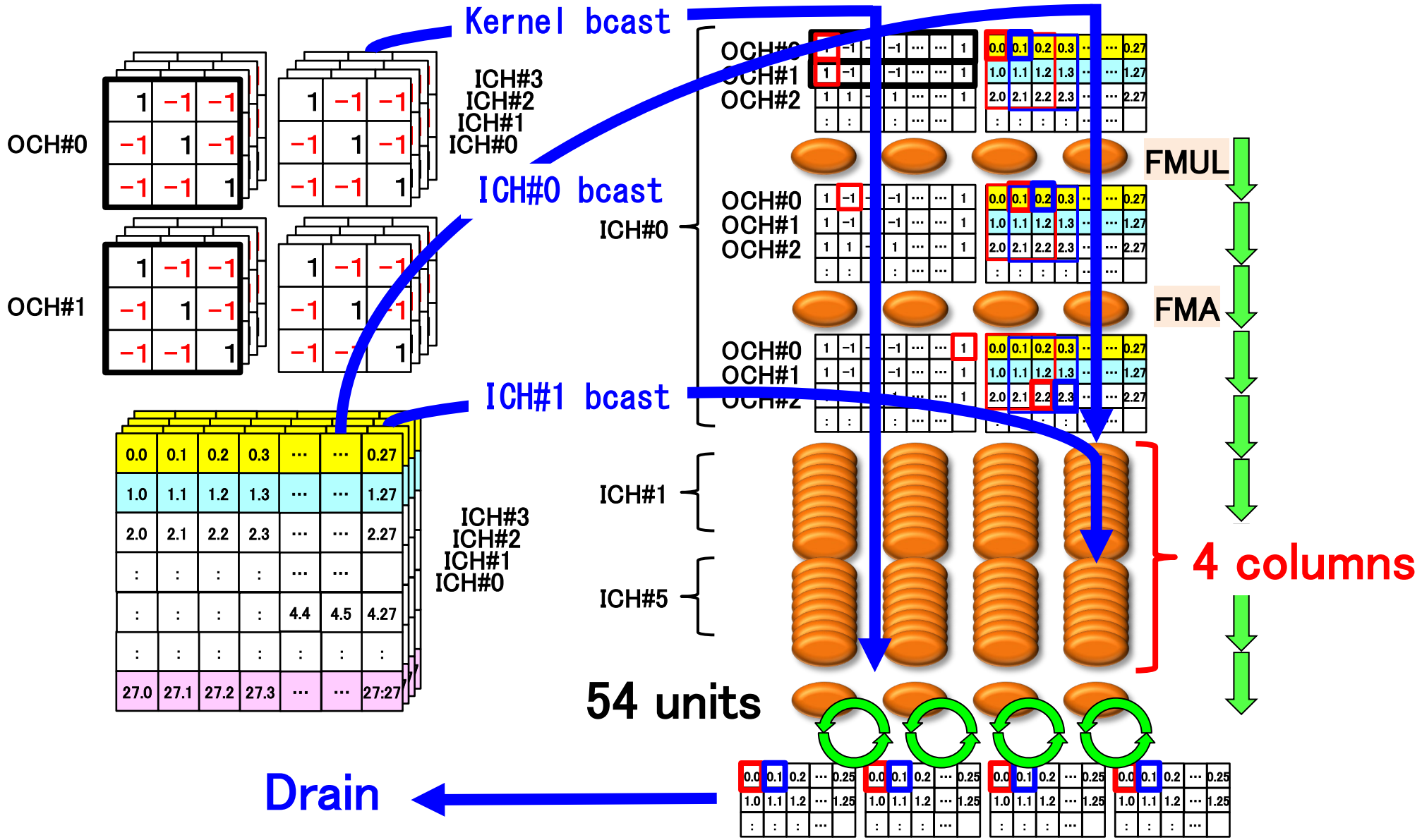
Convolution on CPU/GPU



GPU: unpack and matrix-multiplication (x9 memory space)

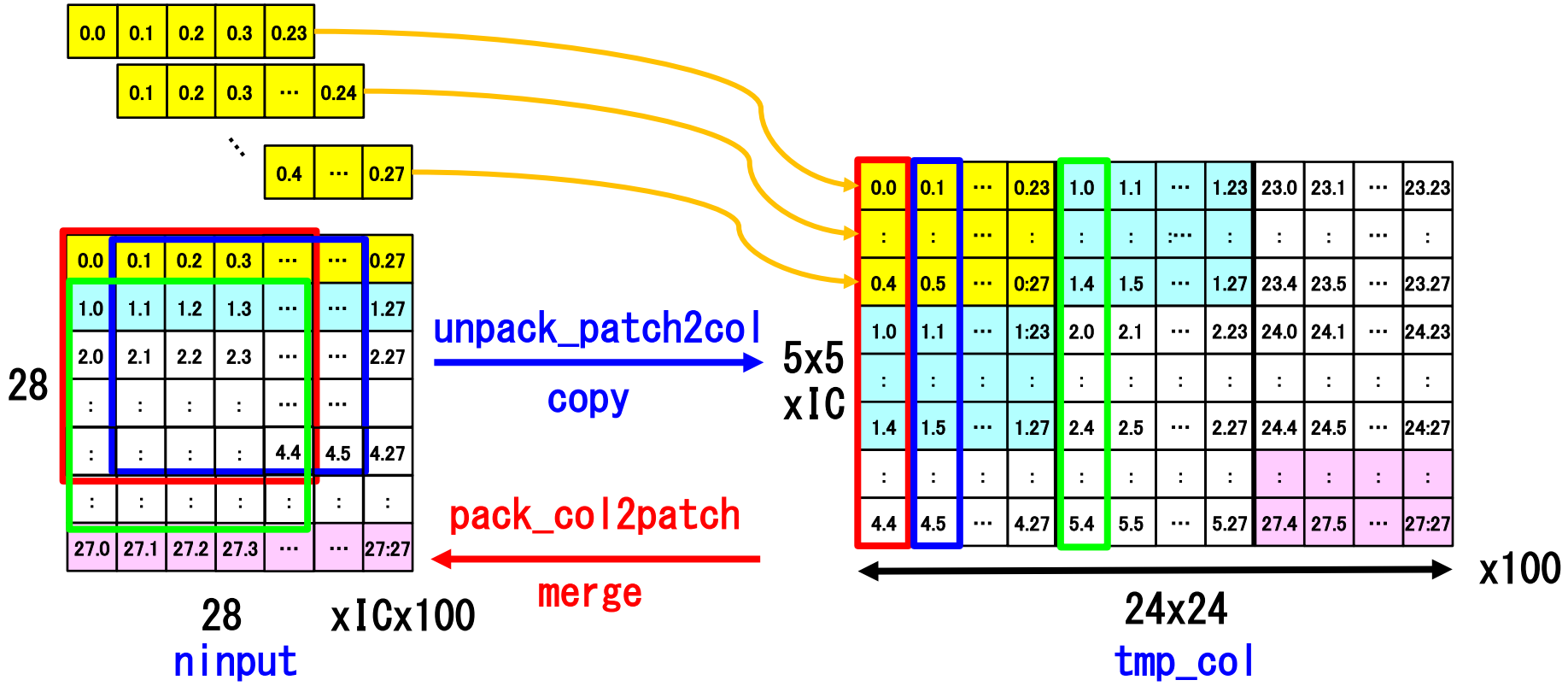


Convolution on CGRA (hardware unpacking)



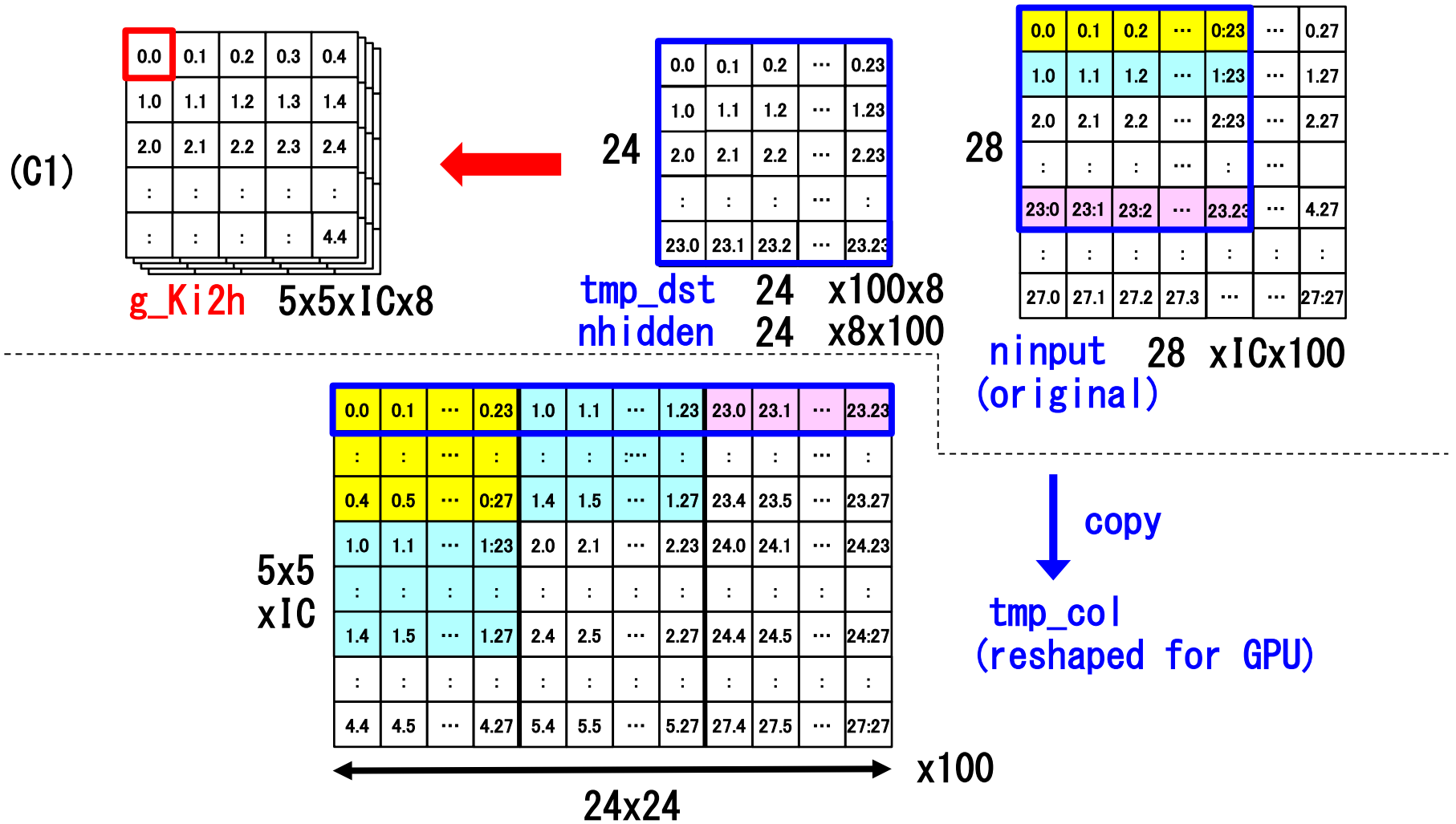
Back propagation on CPU/GPU

```
(conv_backward(nhidden, Ki2h, g_Ki2h, ninput, ksize, kstride, tmp_col, tmp_dst))
 unpack_patch2col(tmp_col:25x[100x24x24] <-- ninput:■100x28x28, ksize, kstride)
 reshape              nhidden:100x[8x24x24] --> tmp_dst:8x[100x24x24]
 multiply_float2D(▲g_Ki2h:8x25              <-- tmp_dst:8x[100x24x24], tmp_col:25x[100x24x24]^T)
 multiply_float2D(tmp_col:25x[100x24x24] <-- Ki2h:8x25^T, tmp_dst:8x[100x24x24])
 pack_col2patch(ninput:■100x28x28        <-- tmp_col:25x[100x24x24], 5, 1)
```



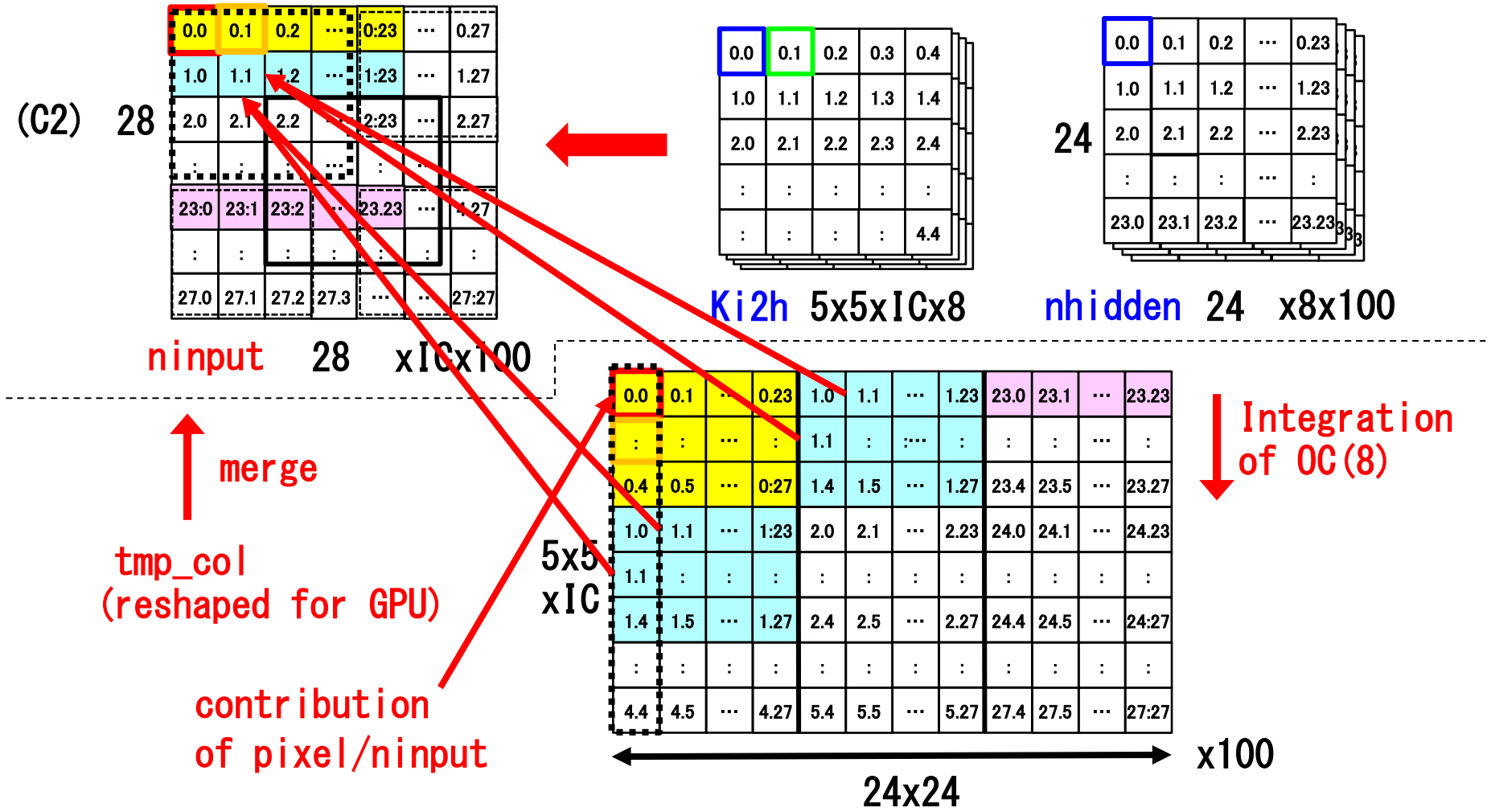
g_Ki2h on CGRA

unpack_patch2col(tmp_col:25x[100x24x24] <-- ninput:■100x28x28, ksize, kstride)
reshape nhidden:100x[8x24x24] --> tmp_dst:8x[100x24x24]
(C1)multiply_float2D(▲g_Ki2h:8x25 <-- tmp_dst:8x[100x24x24], tmp_col:25x[100x24x24]^T)
(C2)multiply_float2D(tmp_col:25x[100x24x24] <-- Ki2h:8x25^T, tmp_dst:8x[100x24x24])
pack_col2patch(ninput:■100x28x28 <-- tmp_col:25x[100x24x24], 5, 1)



ninput on CGRA

unpack_patch2col(tmp_col:25x[100x24x24] <-- ninput:■100x28x28, ksize, kstride)
reshape nhidden:100x[8x24x24] --> tmp_dst:8x[100x24x24]
(C1)multiply_float2D(▲g_Ki2h:8x25 <-- tmp_dst:8x[100x24x24], tmp_col:25x[100x24x24]^T)
(C2)multiply_float2D(tmp_col:25x[100x24x24] <-- Ki2h:8x25^T, tmp_dst:8x[100x24x24])
pack_col2patch(ninput:■100x28x28 <-- tmp_col:25x[100x24x24], 5, 1)

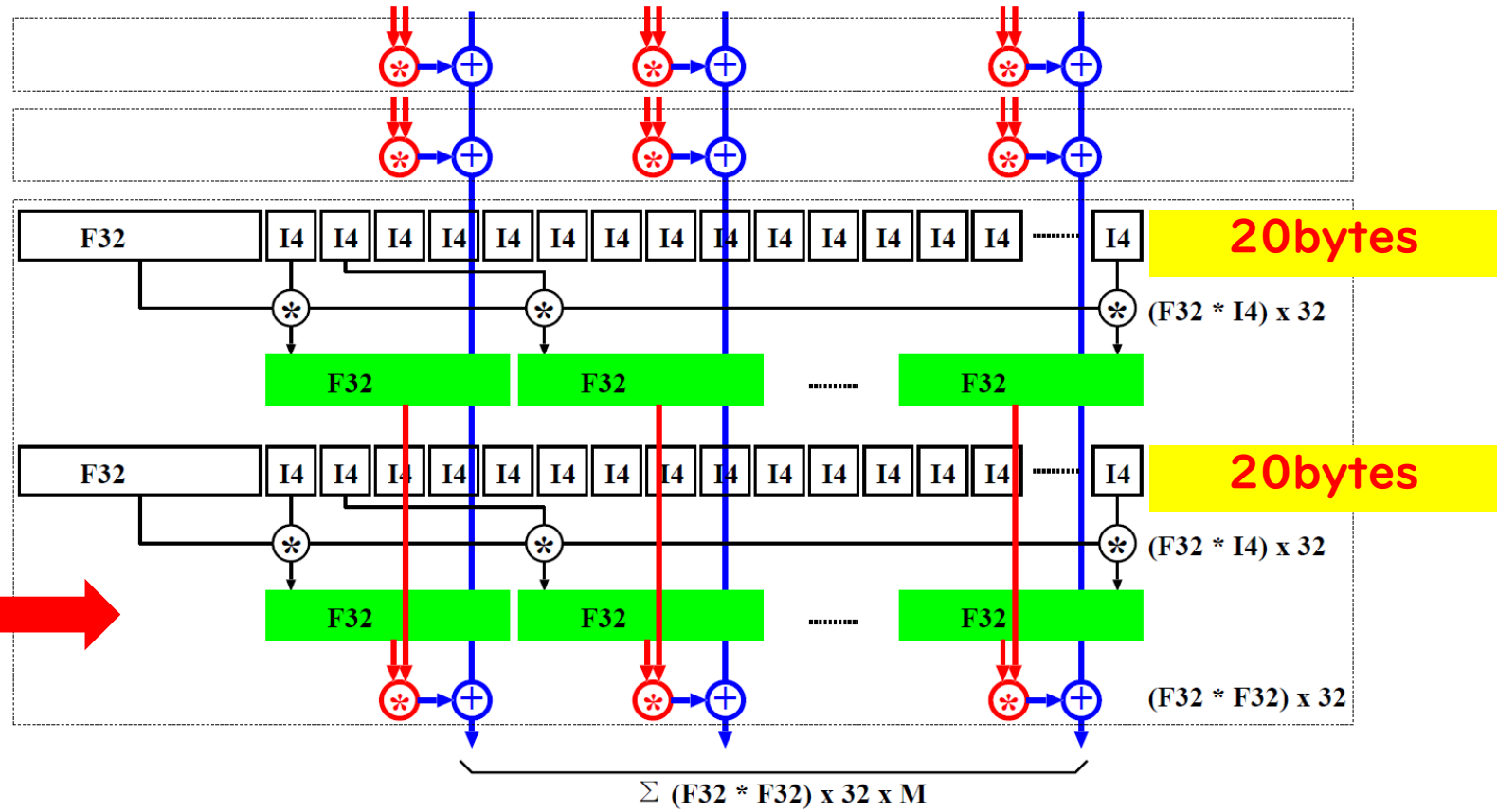


Case of Ggml

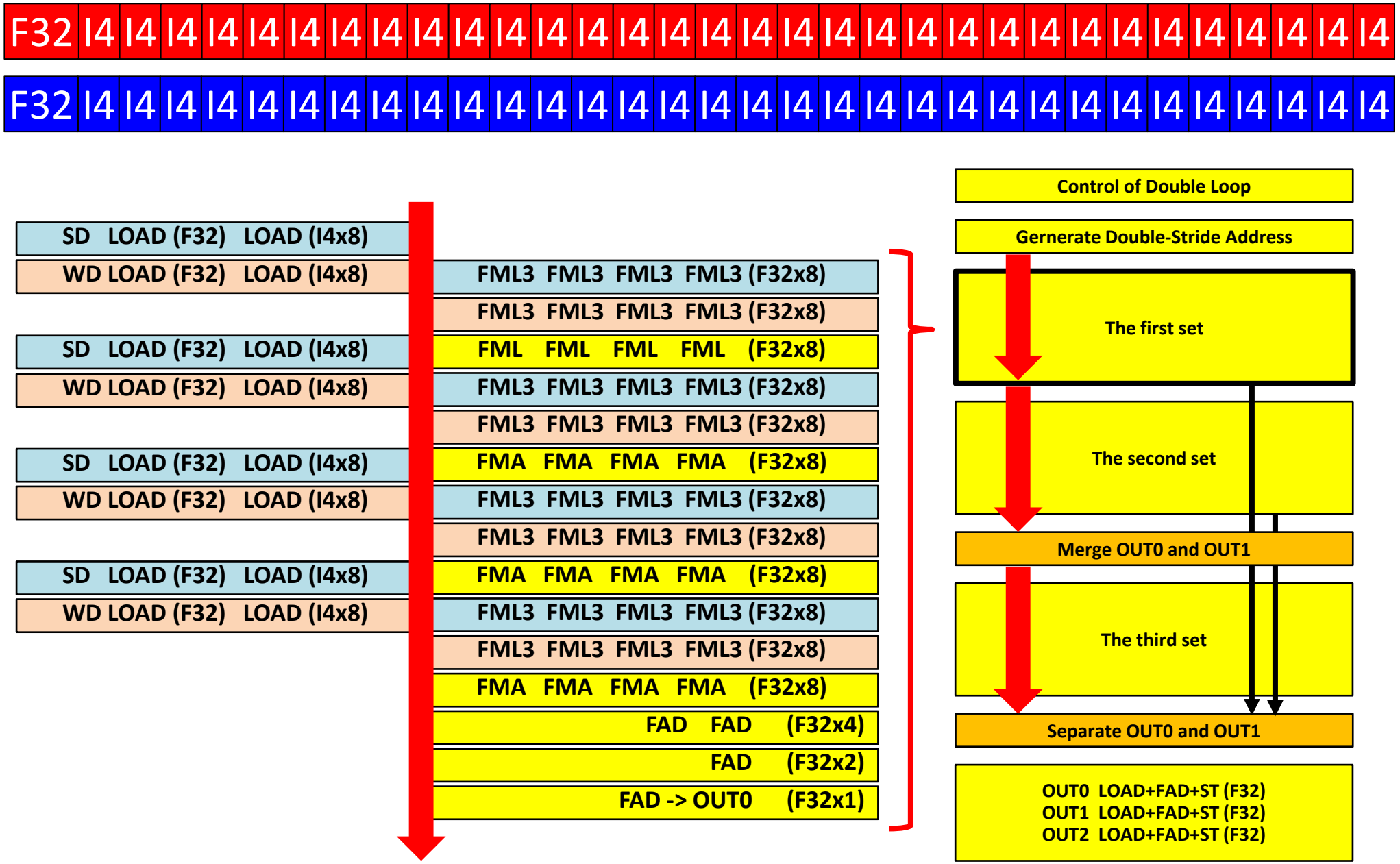
```
python3 chat.py ->interface.py:main_file="../vsim-acap.emax7+dma" ->main.cpp:main_gptneox()

% ./vsim-ubuntu gptneox -m /xxx/.cformers/models/OpenAssistant/oasst-sft-1-pythia-12b/int4_fixed_zero
  --prompt "50278 12092 2 0 50281" --seed 42 --threads 1 --n_predict 100 --top_k 20 --top_p 0.95
  --temp 0.85 --repeat_last_n 64 --repeat_penalty 1.3
main: seed = 42
model_type: gptneox
gptneox_model_load: loading model from '/xxx/.cformers/models/OpenAssistant/oasst-sft-1-pythia-12b/
                    int4_fixed_zero' - please wait ...
gptneox_model_load: n_vocab = 50288
gptneox_model_load: n_ctx   = 512
gptneox_model_load: n_embd  = 5120
gptneox_model_load: n_head  = 40
gptneox_model_load: n_layer = 36
gptneox_model_load: n_rot   = 32
gptneox_model_load: use_parallel_residual = 1
gptneox_model_load: f16     = 2
gptneox_model_load: ggml ctx size = 7786.26 MB
gptneox_model_load: memory_size =   720.00 MB, n_mem = 18432
gptneox_model_load: ..................................................................... done
gptneox_model_load: model size =  7066.11 MB / num tensors = 580
main_gptneox: prompt: '50278 12092 2 0 50281'
main_gptneox: number of tokens in prompt = 5
sampling parameters: temp=0.850000, top_k=20, top_p=0.950000, repeat_last_n=64, repeat_penalty=1.300000
membase: 09e6b600
 embd.size()=0 embd_inp.size()=5 params.n_predict=100
(Hey there!)                    <|BEGIN> 50278 12092 2 0 50281  8262 627  2 <END|>
(Hi there, how are you doing?..) <|BEGIN> 50278 12092 2 0 50281 12764 627 13 849 403 368 2509 32..<END|>
(Hi there!)                     <|BEGIN> 50278 12092 2 0 50281 12764 627  2 <END|>
T_MAIN_GPTNEOX                          : 101.6s(100.0%)
 T_LOAD                                 :   2.7s(  2.7%)
 T_EVAL                                 :  41.2s( 40.5%)
 T_PREDICT                              :  57.7s( 56.8%)
  T_GGML_INIT_GELU                      :   0.0s(  0.0%)
  T_GGML_INIT_GSTAT                     :   0.0s(  0.0%)
  T_COMPUTE_INIT                        :   0.0s(  0.0%)
  T_COMPUTE_NODES                       :  98.9s( 97.3%)
   T_COMPUTE_FORWARD                    :  98.9s( 97.3%)
    T_COMPUTE_FORWARD_ADD               :   0.0s(  0.0%)
    T_COMPUTE_FORWARD_SUB               :   0.0s(  0.0%)
    T_COMPUTE_FORWARD_MUL               :   0.0s(  0.0%)
    T_COMPUTE_FORWARD_DIV               :   0.0s(  0.0%)
    :
    T_COMPUTE_FORWARD_MUL_MAT           :  98.8s( 97.2%)
    T_COMPUTE_FORWARD_MUL_MAT_Q4_0_F32  :  98.8s( 97.2%)
    T_COMPUTE_FORWARD_MUL_MAT_Q4_0_F32_ACC :   0.0s(  0.0%)
    T_COMPUTE_FORWARD_MUL_MAT_Q4_0_F32_INI :   0.1s(  0.1%)
    T_COMPUTE_FORWARD_MUL_MAT_Q4_0_F32_FIN :   0.0s(  0.0%)
    T_COMPUTE_FORWARD_MUL_MAT_Q4_0_F32_NB01_GE_NB00 :   0.0s(  0.0%)
    T_GGML_VEC_DOT_Q4_0                 :   0.0s(  0.0%)
    IMAX_COMPUTE_FORWARD_MUL_MAT_Q4_0_F32_NB01_GE_NB00:  98.7s( 97.1%)
    IMAX_CPYIN                          :   0.0s(  0.0%)
    IMAX_CPYOUT                         :   0.0s(  0.0%)
    T_COMPUTE_FORWARD_MUL_MAT_Q4_0_F32_NB01_LT_NB00 :   0.0s(  0.0%)
    T_COMPUTE_FORWARD_MUL_MAT_Q4_1_F32  :   0.0s(  0.0%)
    T_COMPUTE_FORWARD_MUL_MAT_Q4_F16_F32 :   0.0s(  0.0%)
    T_COMPUTE_FORWARD_MUL_MAT_F32       :   0.0s(  0.0%)
    :
    T_COMPUTE_FORWARD_FLASH_ATTN        :   0.0s(  0.0%)
    T_COMPUTE_FORWARD_FLASH_FF          :   0.0s(  0.0%)
  T_GGML_OPT                            :   0.0s(  0.0%)
 T_SAMPLE                               :   0.0s(  0.0%)
```

Heavy function

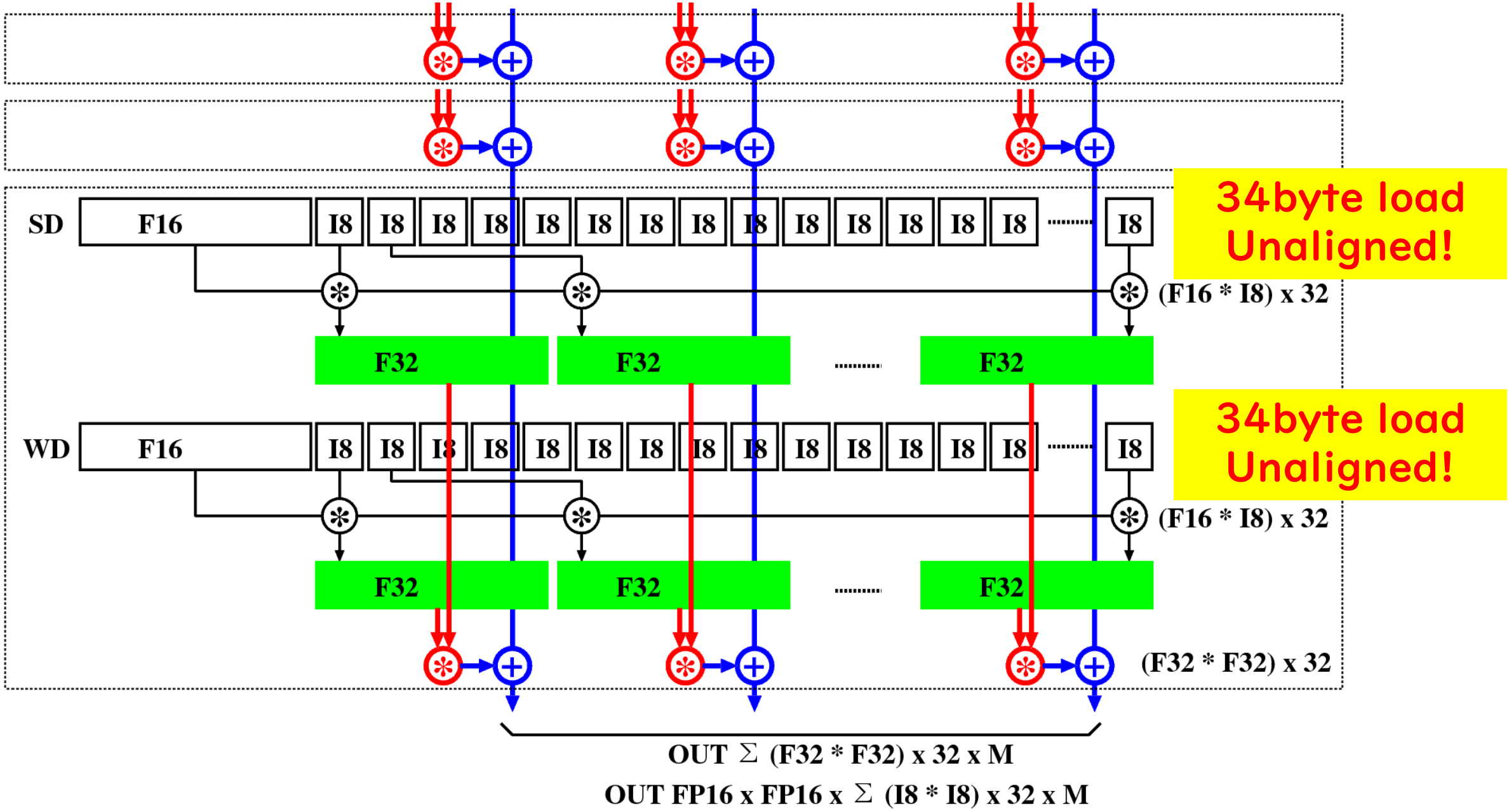
```
imax_compute_forward_mul_mat_q4_0_f32_nb01_ge_nb00() {
  for (int ir=0; ir<nr; ir++) { /* 5120, 20480, 50288■ */
    for (int ic=0; ic<ne11; ic++) { /* 1,5,8,9 */
      for (int i=0; i<nb; i++) { /* 160, 640 */
        for (int j=0; j<QK/2; j++) { /* 16 */
          const float slo=*sdf32 * ((int8_t)(s0[j]&0xf)-8); const float shi=*sdf32*((int8_t)(s0[j]>>4)-8);
          const float wlo=*wdf32 * ((int8_t)(w0[j]&0xf)-8); const float whi=*wdf32*((int8_t)(w0[j]>>4)-8);
          sumf += slo*wlo + shi*whi;
      } }
      dst_col[ic*ne0]=sumf; /* ic毎に, ne0:5120W, 20480W, 50288W飛び■ 最外irで4B毎にストア */
} } }
```



Case of Ggml


F32 I4 I4 I4 I4 I4 I4 I4 I4 I4 I4 I4 I4 I4 I4 I4 I4 I4 I4 I4 I4 I4 I4 I4 I4 I4 I4 I4 I4 I4 I4 I4 I4
F32 I4 I4 I4 I4 I4 I4 I4 I4 I4 I4 I4 I4 I4 I4 I4 I4 I4 I4 I4 I4 I4 I4 I4 I4 I4 I4 I4 I4 I4 I4 I4 I4
SD LOAD (F32) LOAD (I4x8)
WD LOAD (F32) LOAD (I4x8)
SD LOAD (F32) LOAD (I4x8)
WD LOAD (F32) LOAD (I4x8)
SD LOAD (F32) LOAD (I4x8)
WD LOAD (F32) LOAD (I4x8)
SD LOAD (F32) LOAD (I4x8)
WD LOAD (F32) LOAD (I4x8)
FML3 FML3 FML3 FML3 (F32x8)
FML3 FML3 FML3 FML3 (F32x8)
FML FML FML FML (F32x8)
FML3 FML3 FML3 FML3 (F32x8)
FML3 FML3 FML3 FML3 (F32x8)
FMA FMA FMA FMA (F32x8)
FML3 FML3 FML3 FML3 (F32x8)
FML3 FML3 FML3 FML3 (F32x8)
FMA FMA FMA FMA (F32x8)
FML3 FML3 FML3 FML3 (F32x8)
FML3 FML3 FML3 FML3 (F32x8)
FMA FMA FMA FMA (F32x8)
FAD FAD (F32x4)
FAD (F32x2)
FAD -> OUT0 (F32x1)
Control of Double Loop
Gernerate Double-Stride Address
The first set
The second set
Merge OUT0 and OUT1
The third set
Separate OUT0 and OUT1
OUT0 LOAD+FAD+ST (F32)
OUT1 LOAD+FAD+ST (F32)
OUT2 LOAD+FAD+ST (F32)

Case of llama (ggml+LLM)

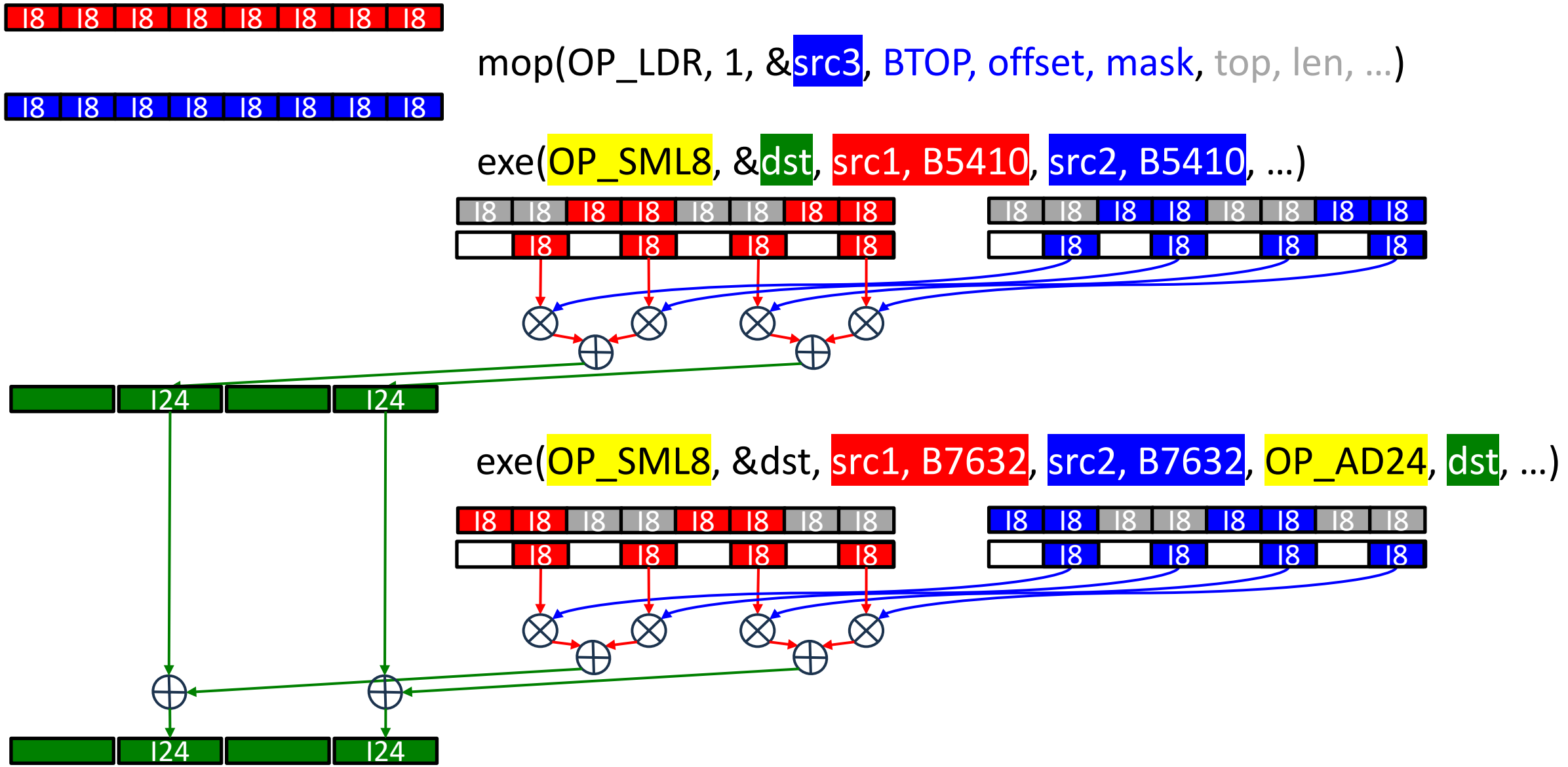


SML8/AD24: New operation for quad I8*I8

mop(OP_LDR, 1, &src2, ATOP, offset, mask, top, len, …)

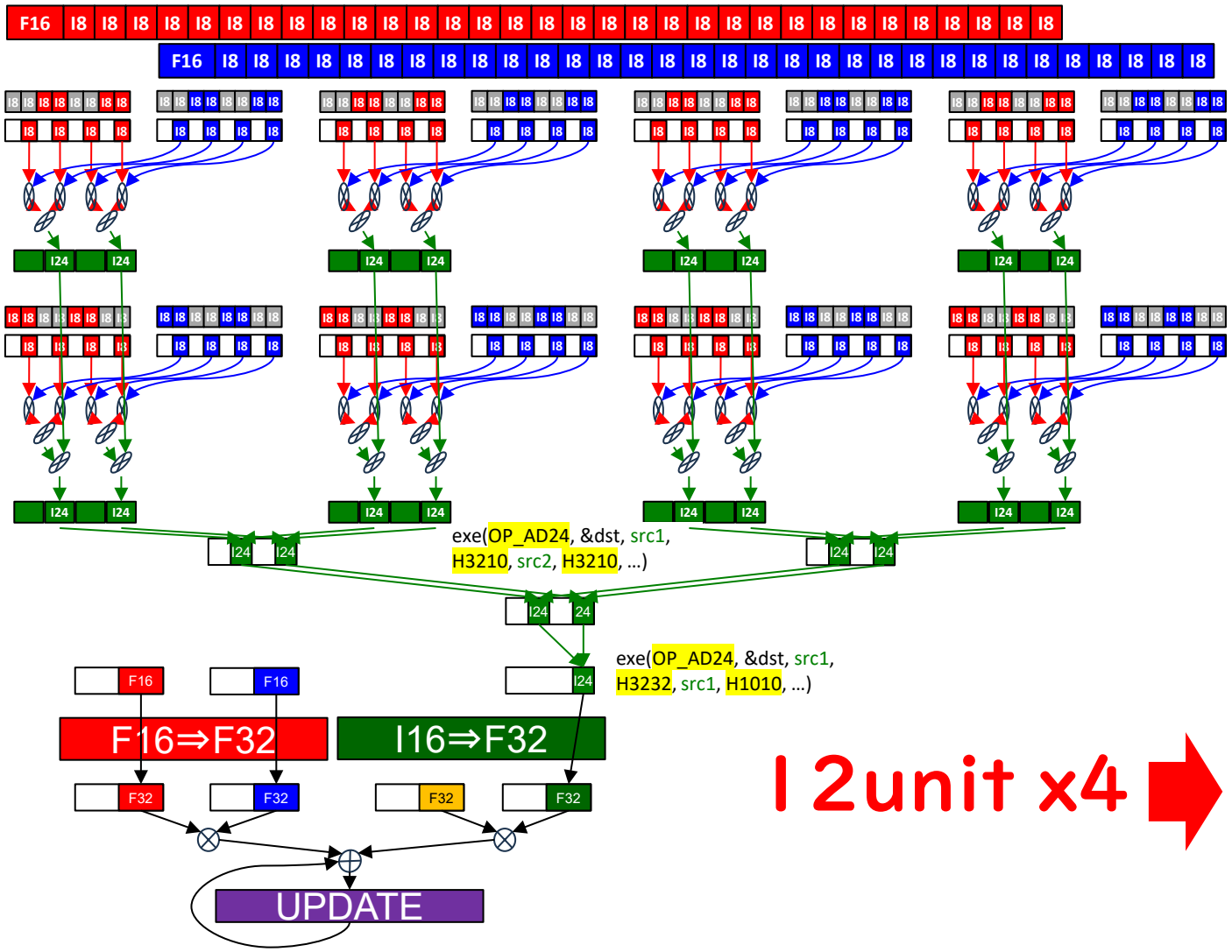
mop(OP_LDR, 1, &src3, BTOP, offset, mask, top, len, …)

exe(OP_SML8, &dst, src1, B5410, src2, B5410, …)

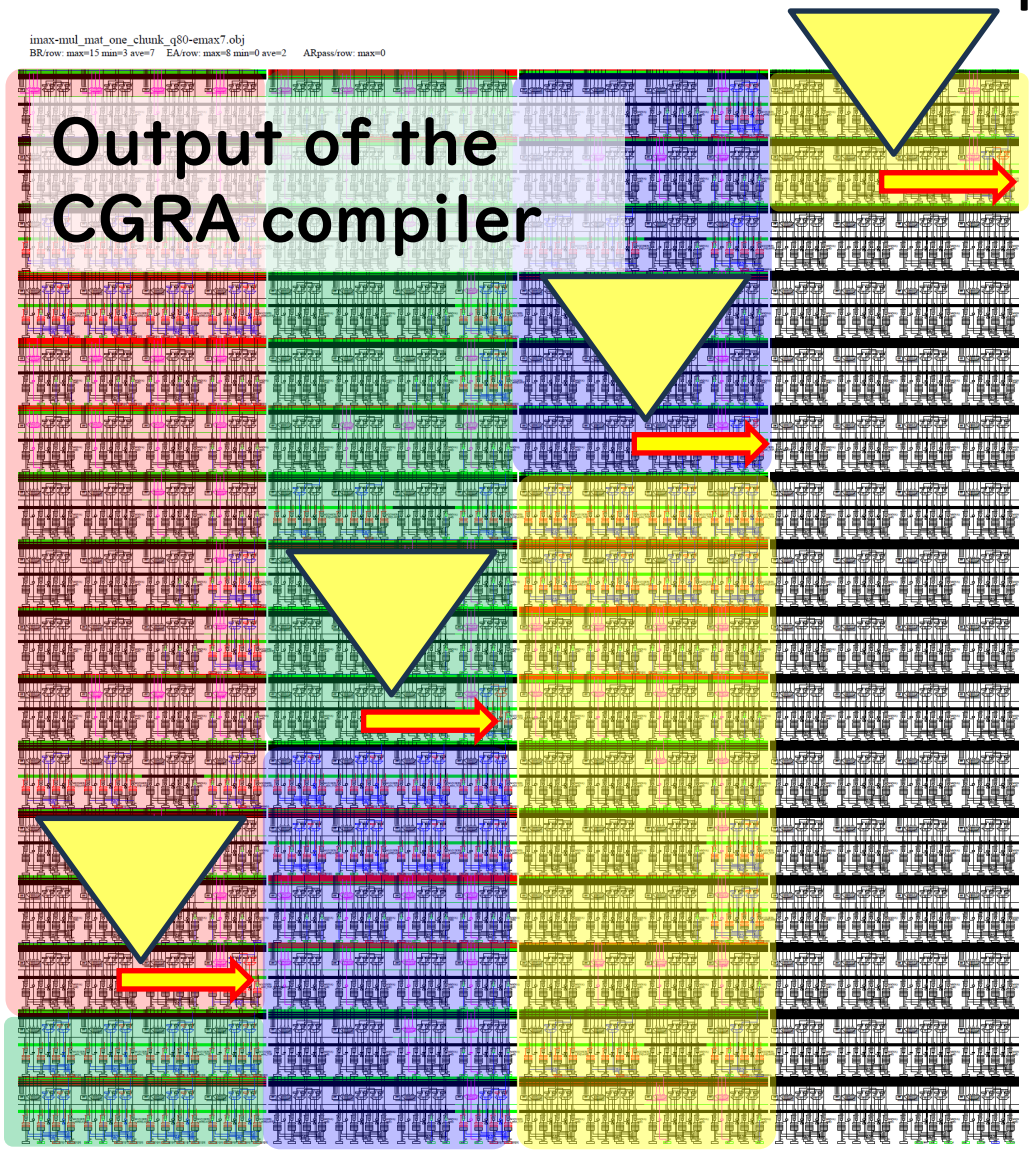


exe(OP_SML8, &dst, src1, B7632, src2, B7632, OP_AD24, dst, …)

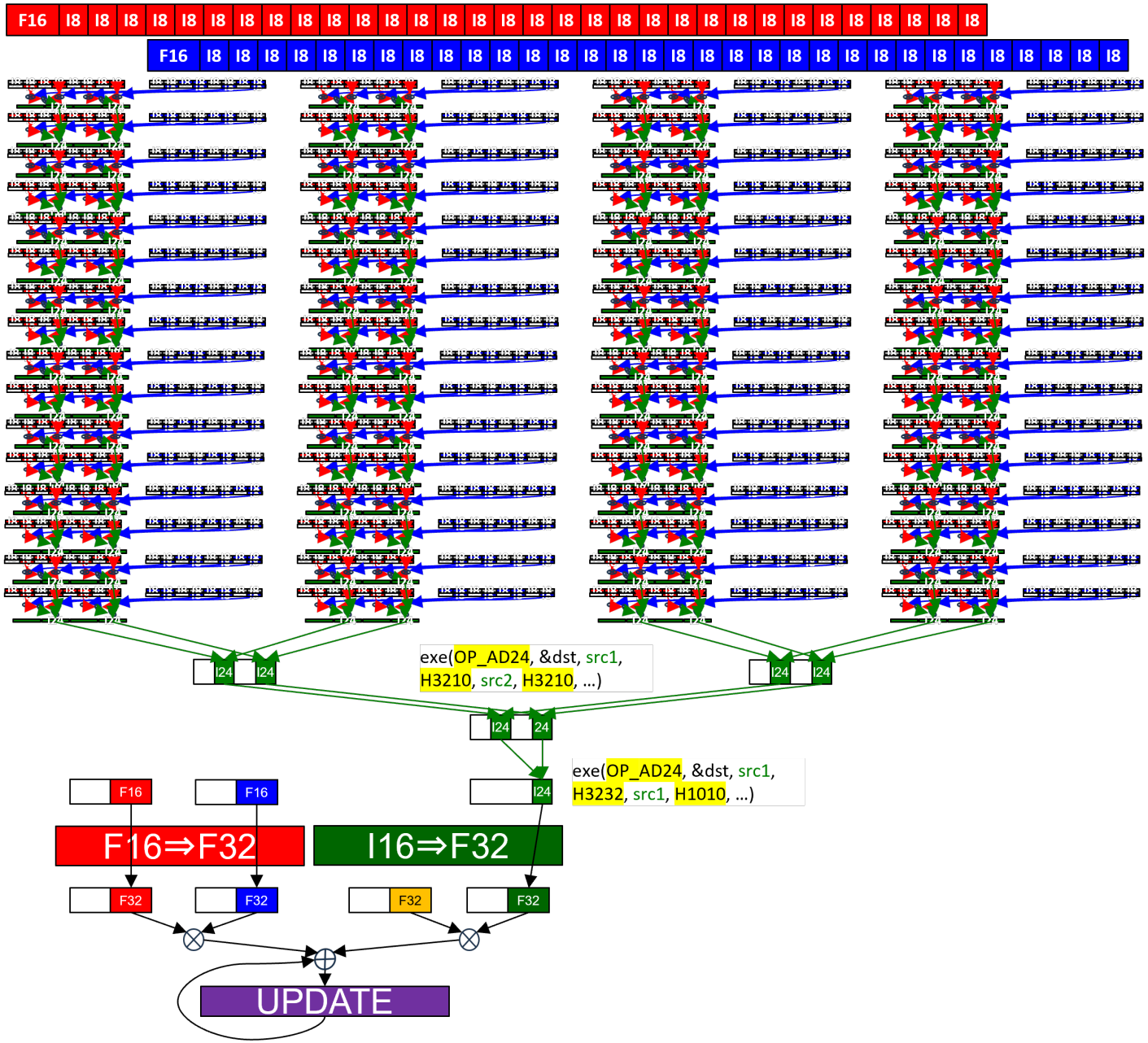
Llama mul_mat_Q80 (8bit MAC)

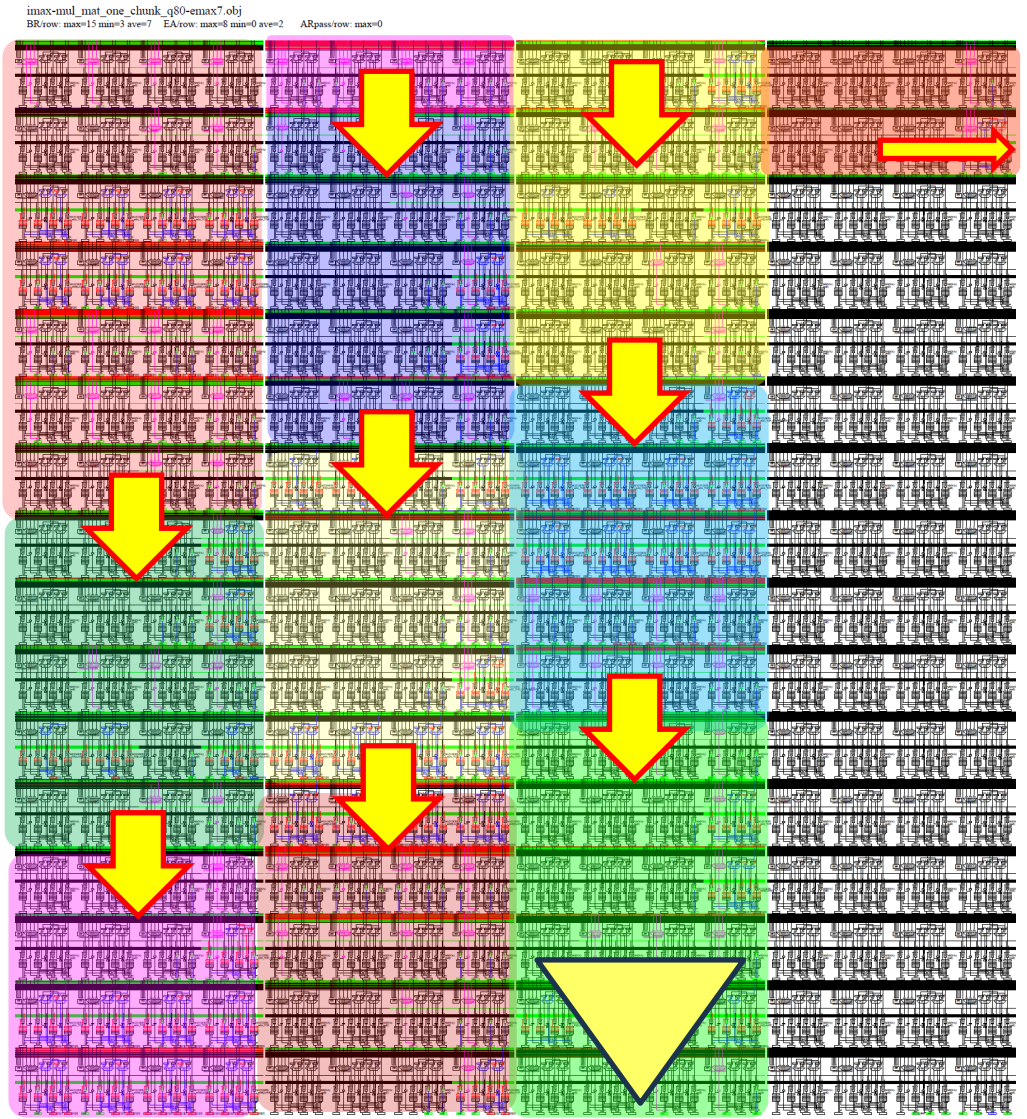


I 2unit x4



Llama mul_mat_Q3K(I+2,2+4,8bit MAC)





Summary

Important feature for large-scale data processing is:
memory bandwidth (not peak performance of ALU)

The only way to improve performance
with limited bandwidth is:
to keep reusable data in local memory as many as possible

That’s all for today