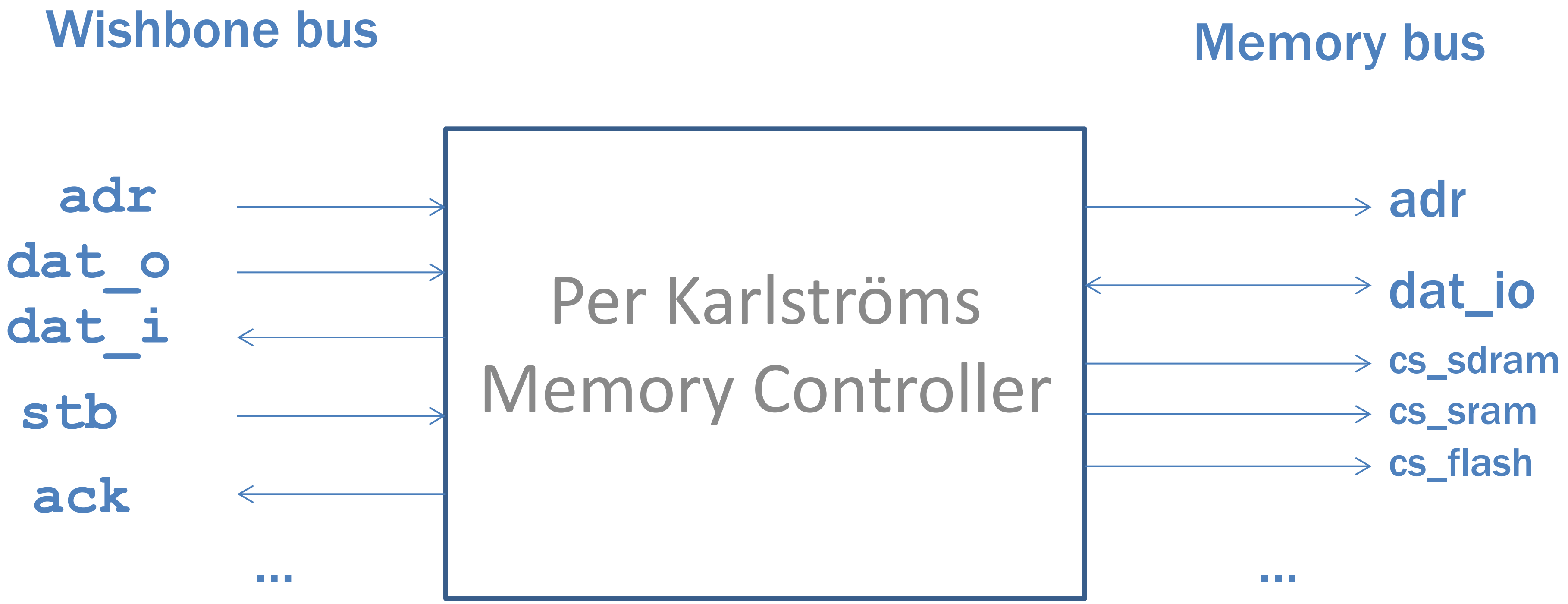
Lecture 8

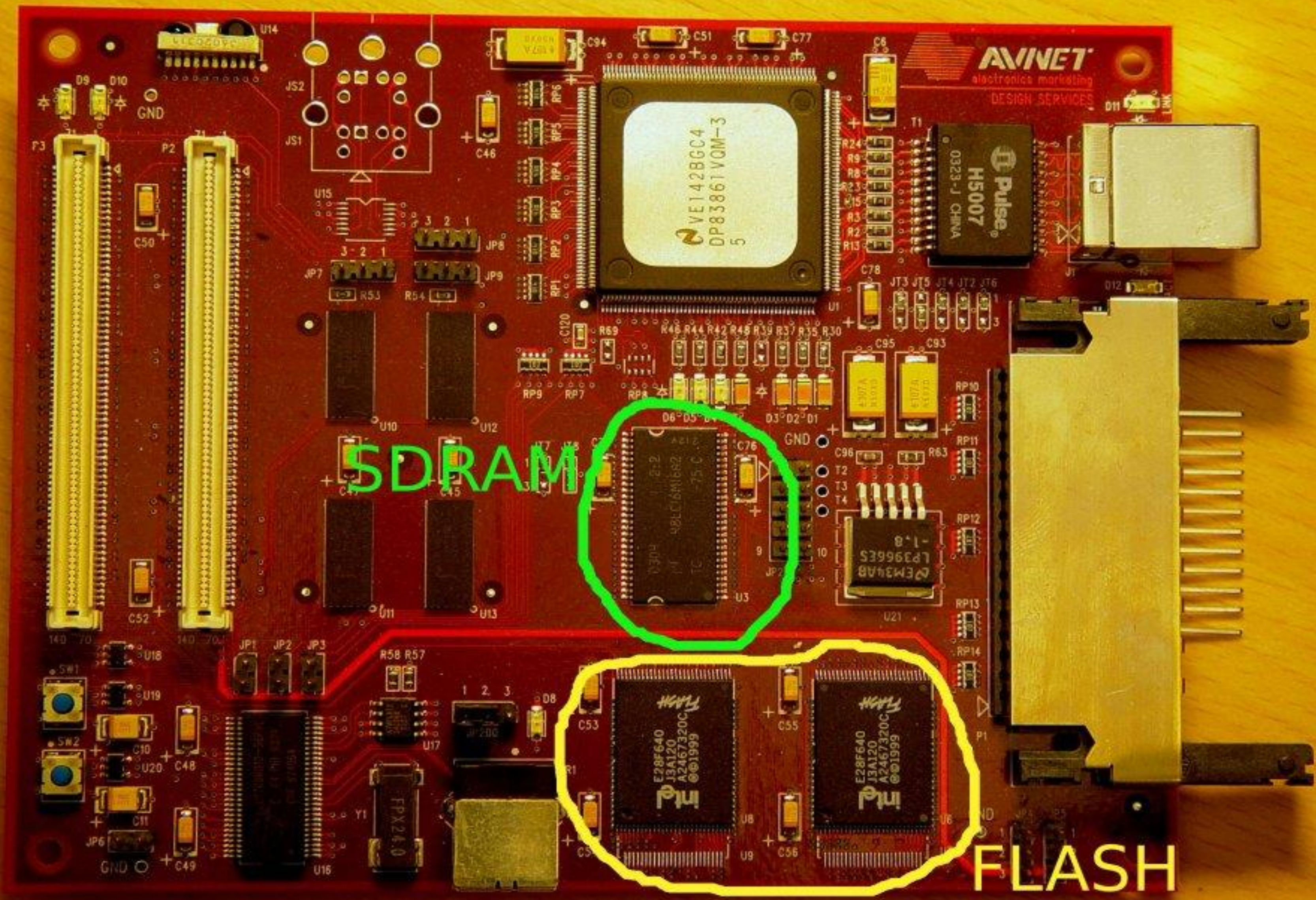
- A memory controller
- Lab4 – a special instruction

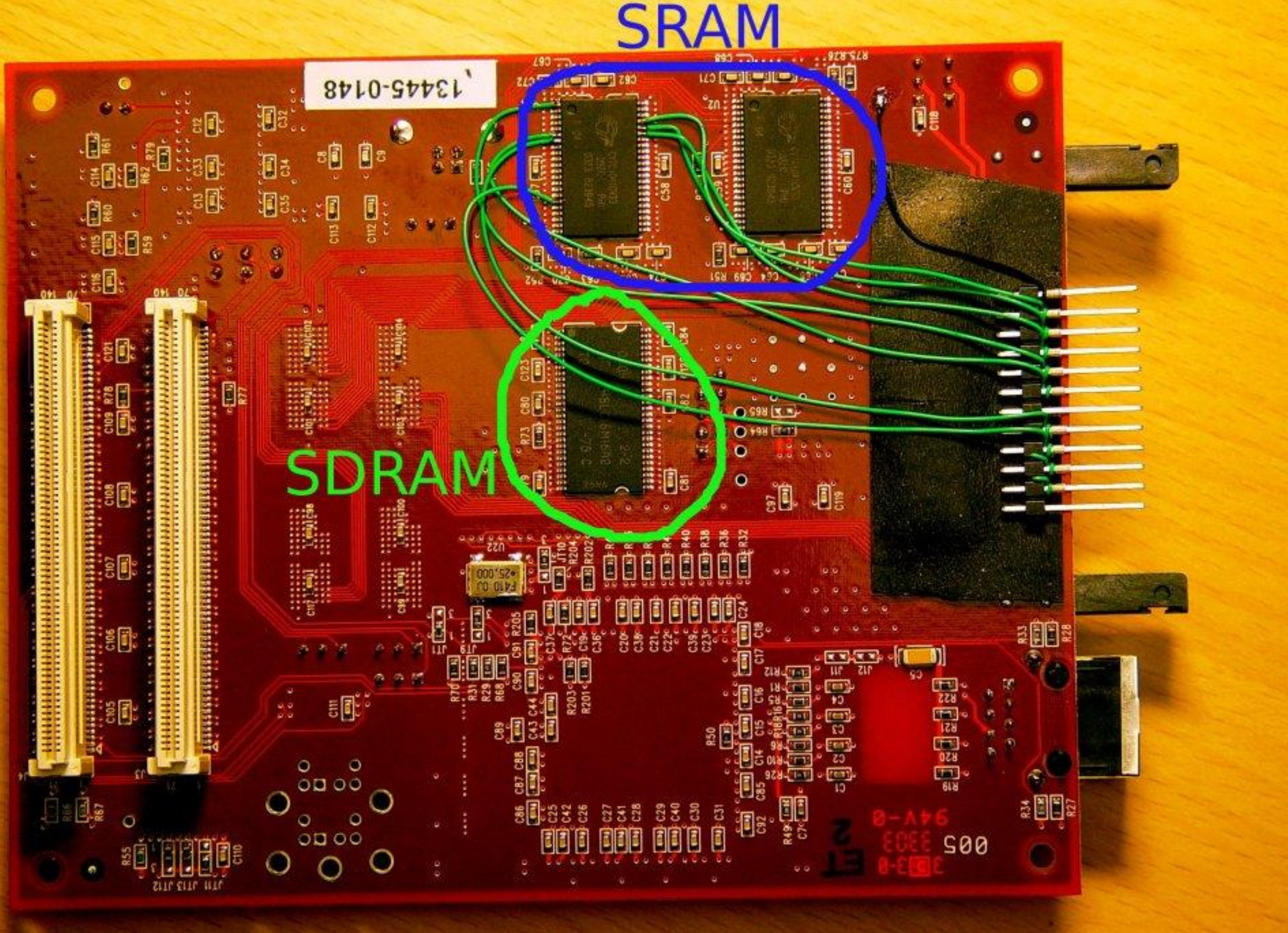
Guest lecture
Time: Friday 5/12 1515-1600.
Place: Nollstället
Image Processing on FPGAs.
Johan Pettersson, Sick IVP

PKMC



- 3 controllers in one
- no registers
  - SDRAM on 0x0,
  - SRAM on 0x20000000 or 0xc0000000
  - Flash on 0xf0000000

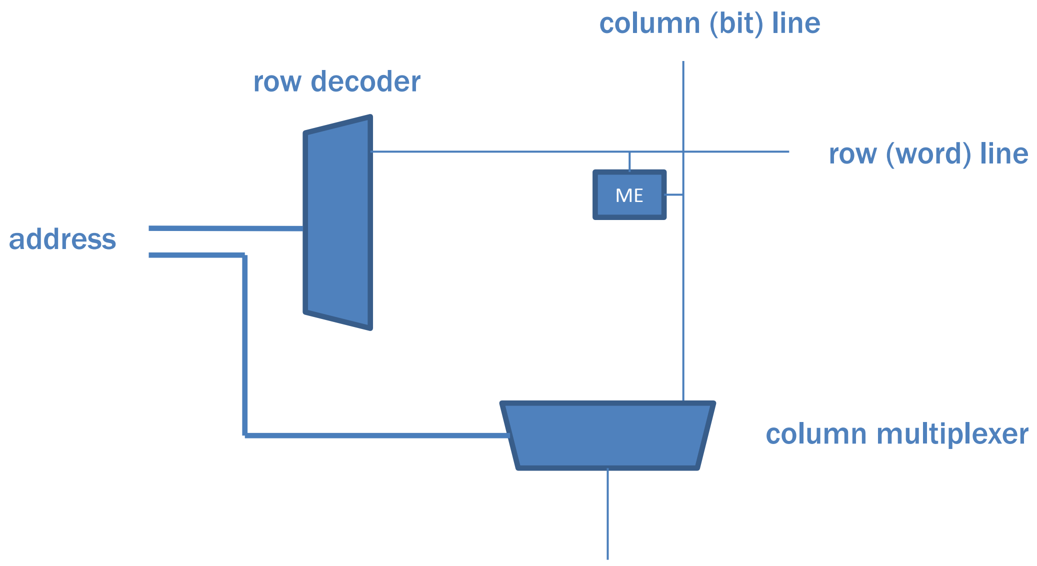

AVNET
electronics marketing
DESIGN SERVICES
SDRAM
FLASH
Pulse H5007
DP83861VQM-3
intel E28F640 J3A120

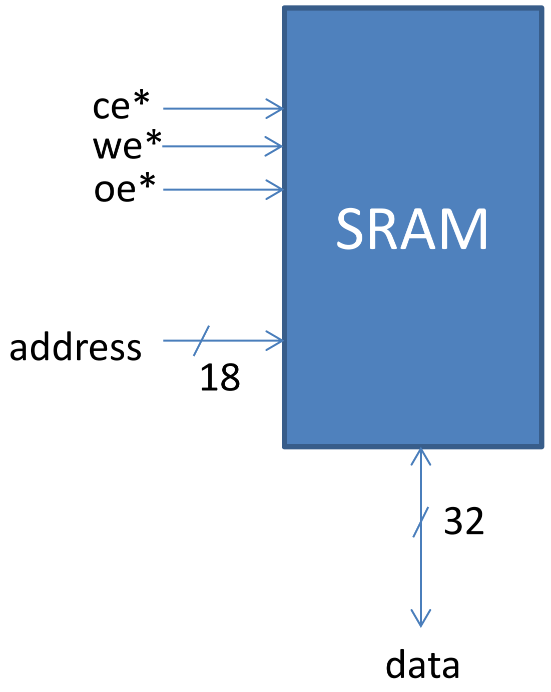

SRAM
SDRAM
13445-0148

SRAM

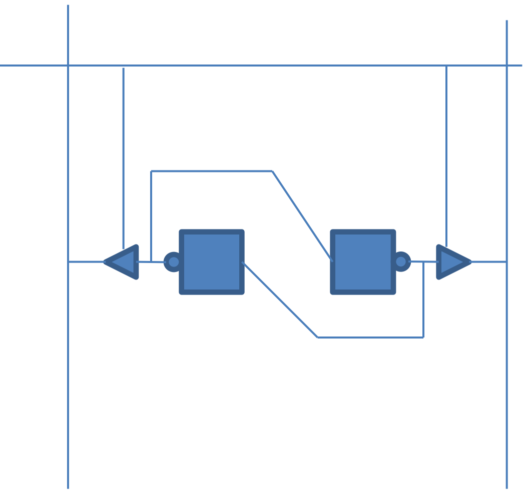
Static RAM

- Asynchronous device
- Memory element: Latch
- 2 x (256k x 16) = 1MB

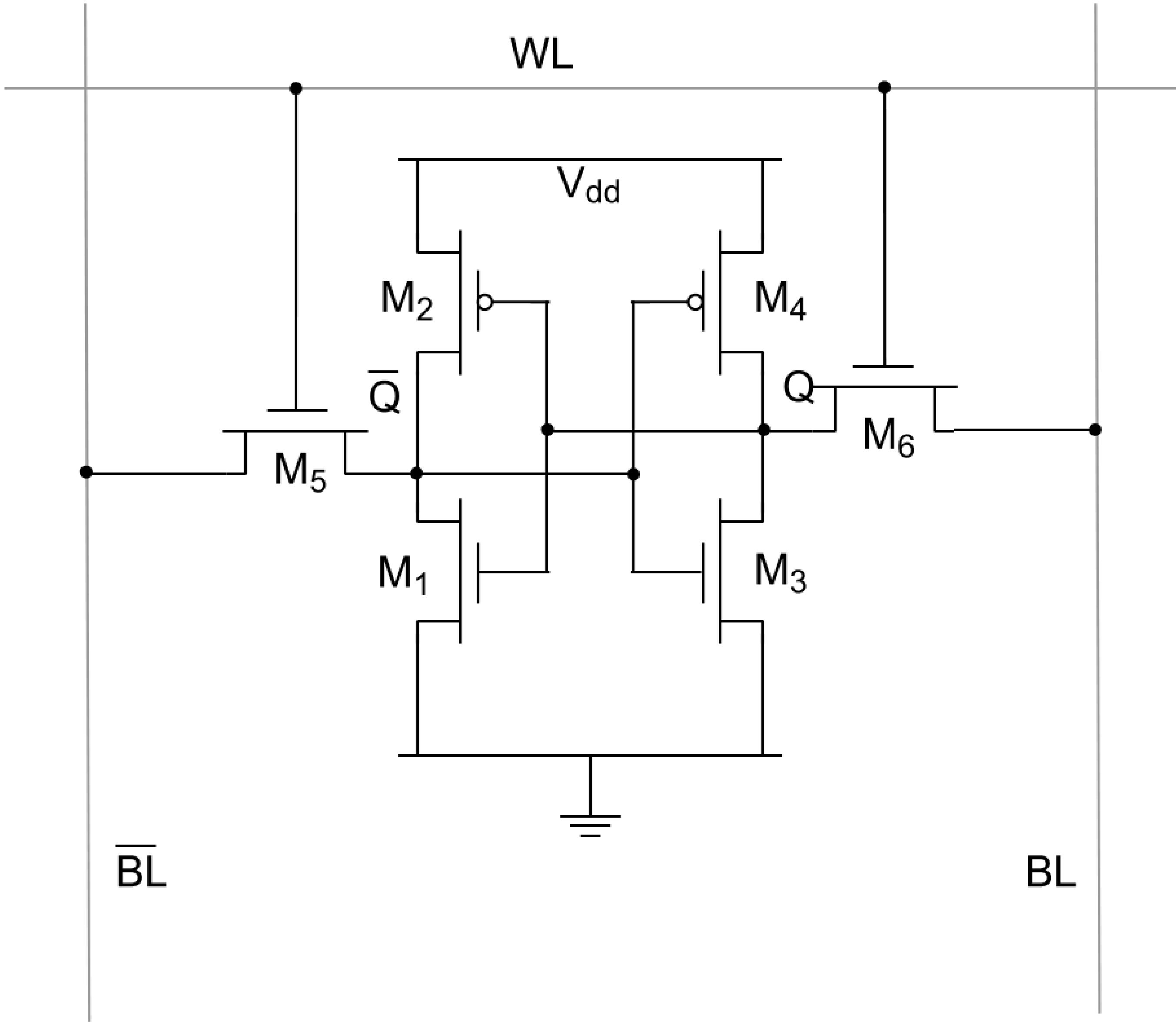




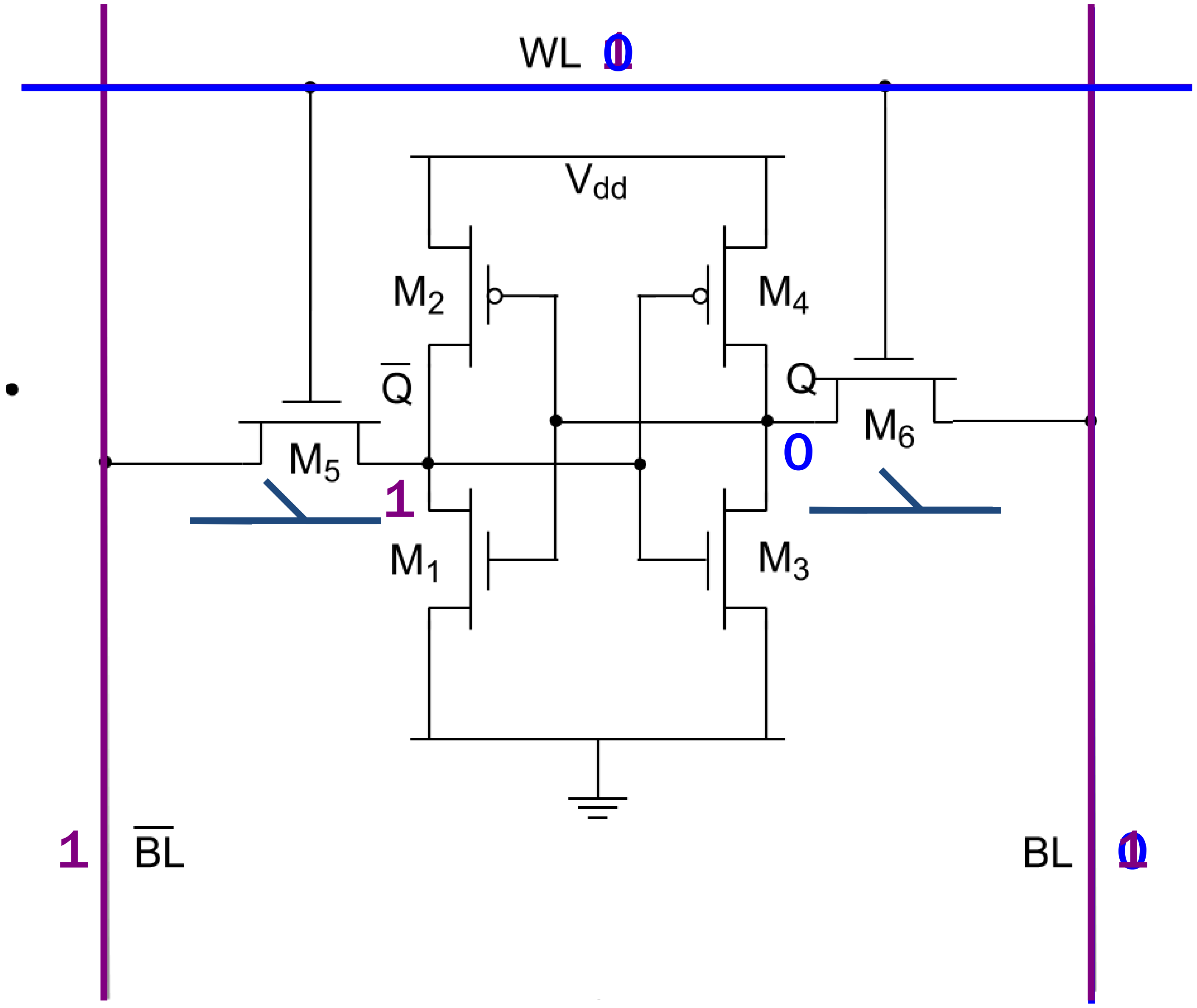
SRAM - 6 transistor cell



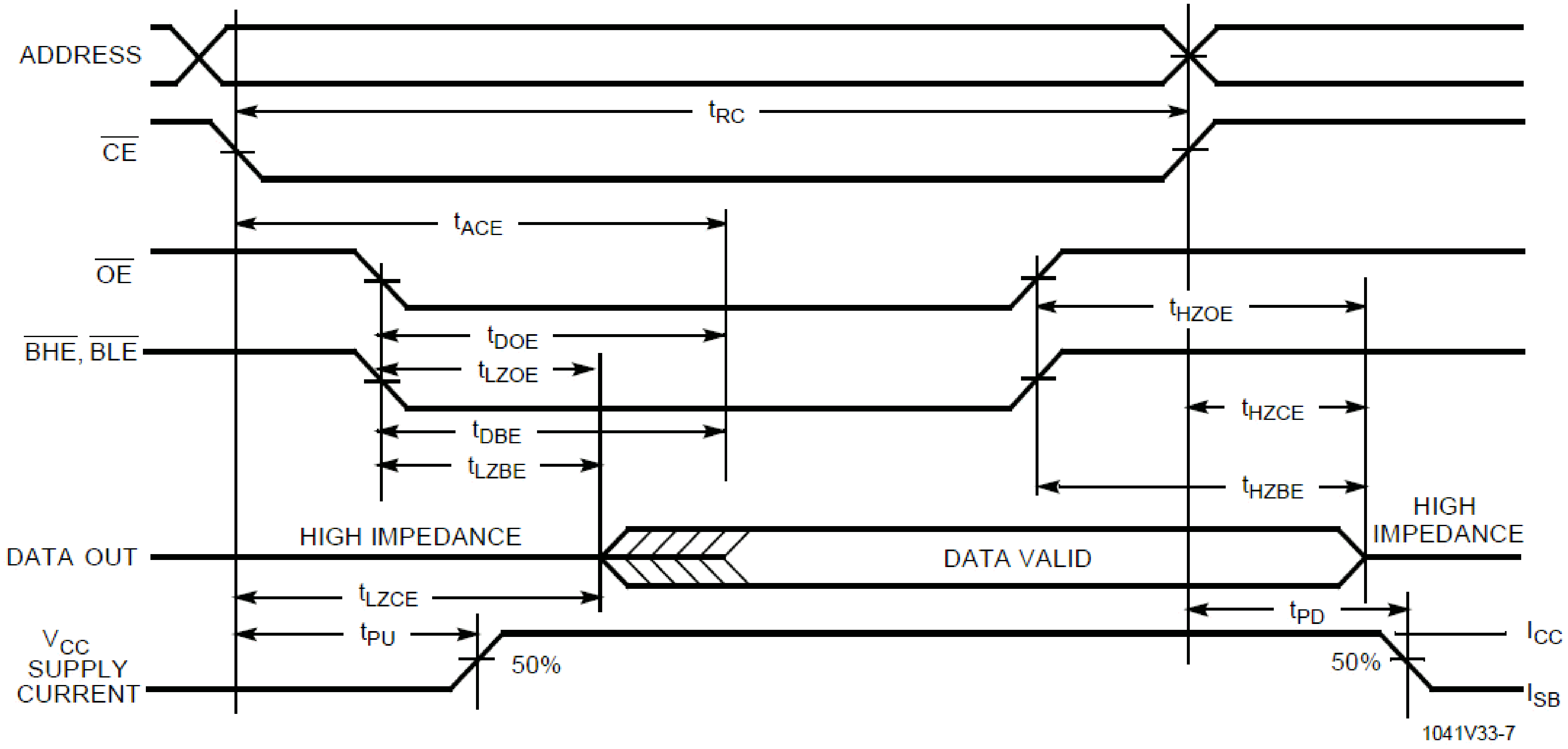
•



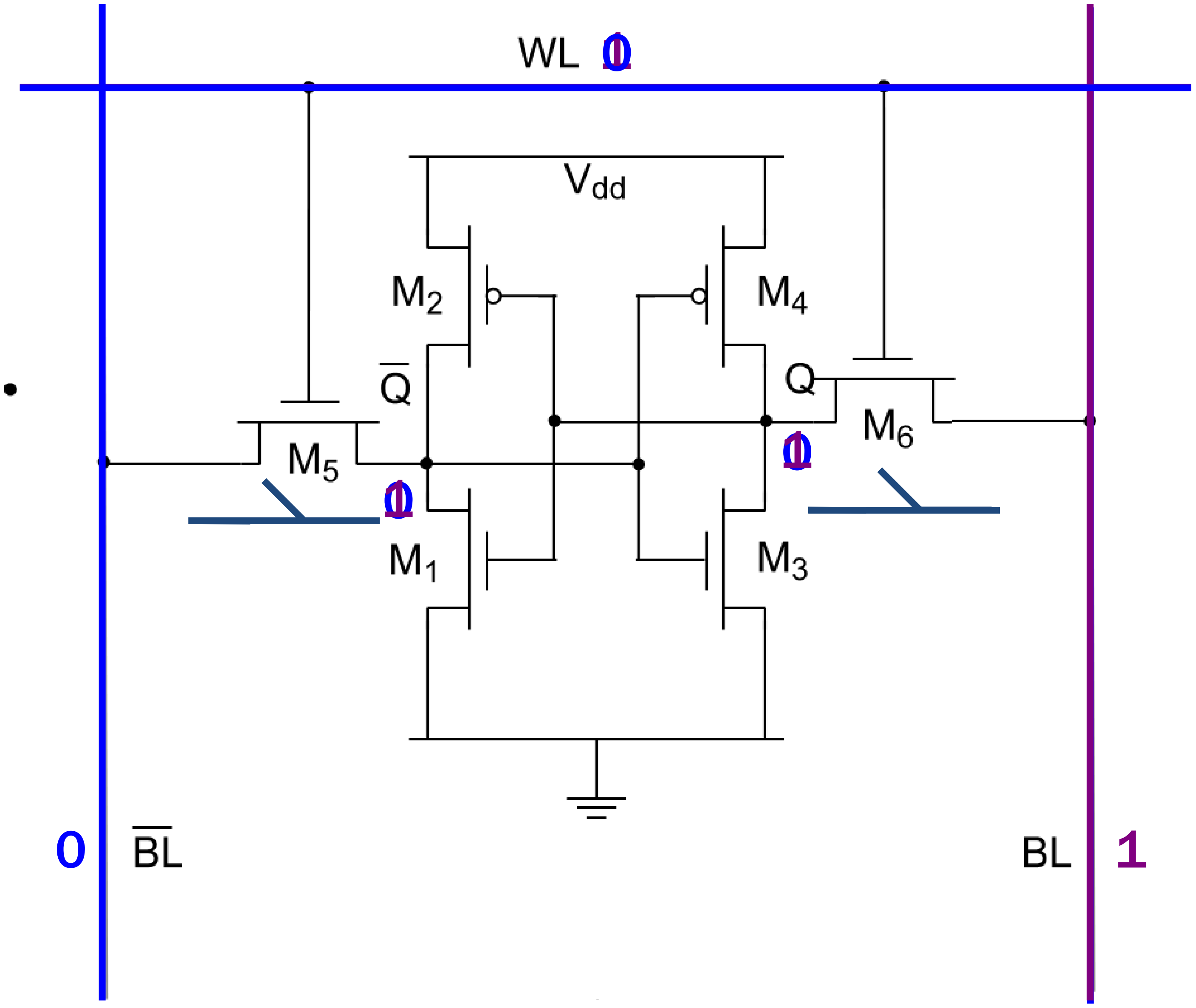
SRAM - Read



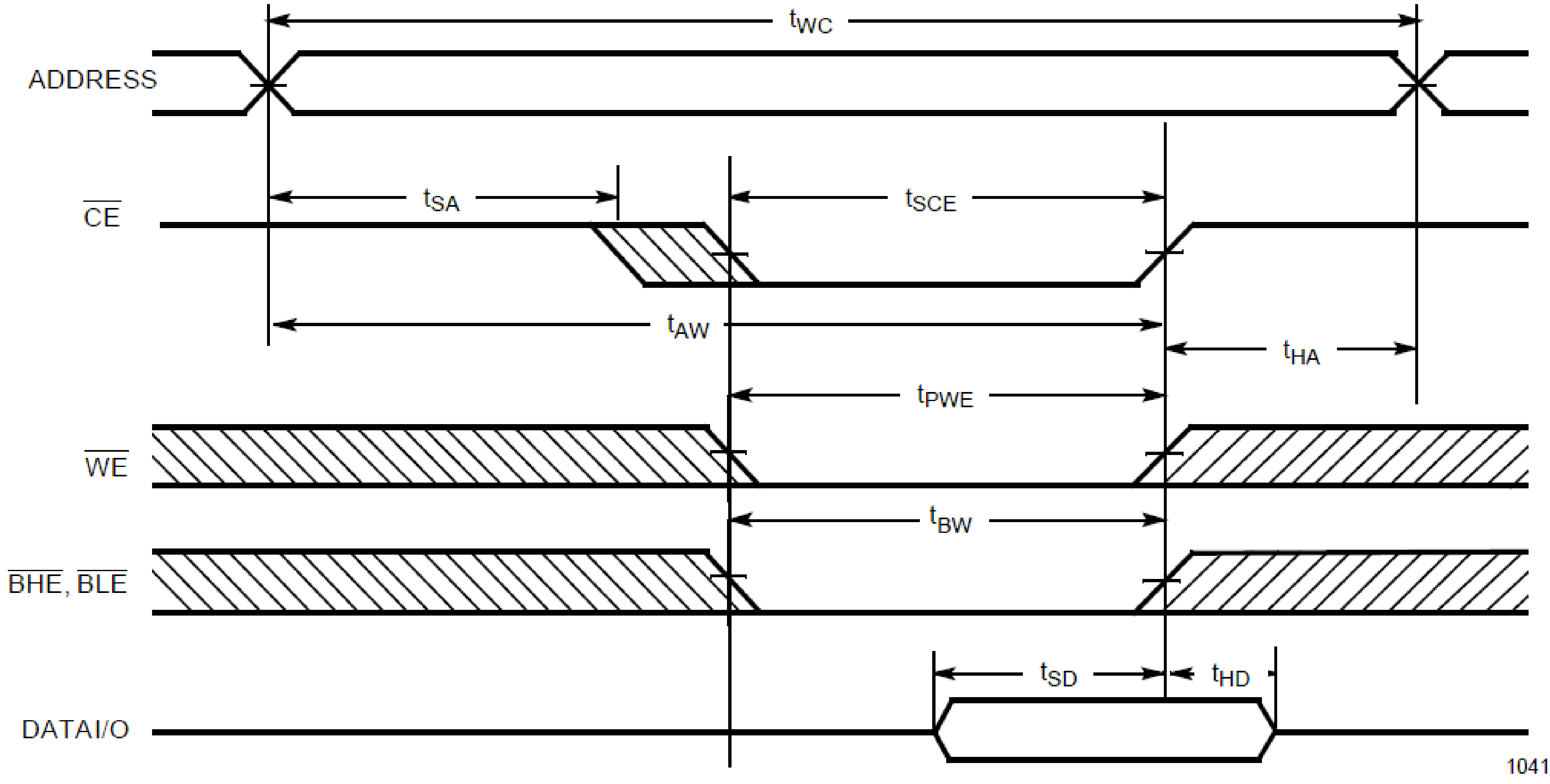
SRAM - Read


ADDRESS
$\overline{CE}$
$\overline{OE}$
$\overline{BHE}$, $\overline{BLE}$
DATA OUT
V_{CC} SUPPLY CURRENT
t_{RC}
t_{ACE}
t_{DOE}
t_{LZOE}
t_{DBE}
t_{LZBE}
t_{HZOE}
t_{HZCE}
t_{HZBE}
HIGH IMPEDANCE
DATA VALID
HIGH IMPEDANCE
t_{LZCE}
t_{PU}
t_{PD}
50%
50%
I_{CC}
I_{SB}
1041V33-7

SRAM - Write

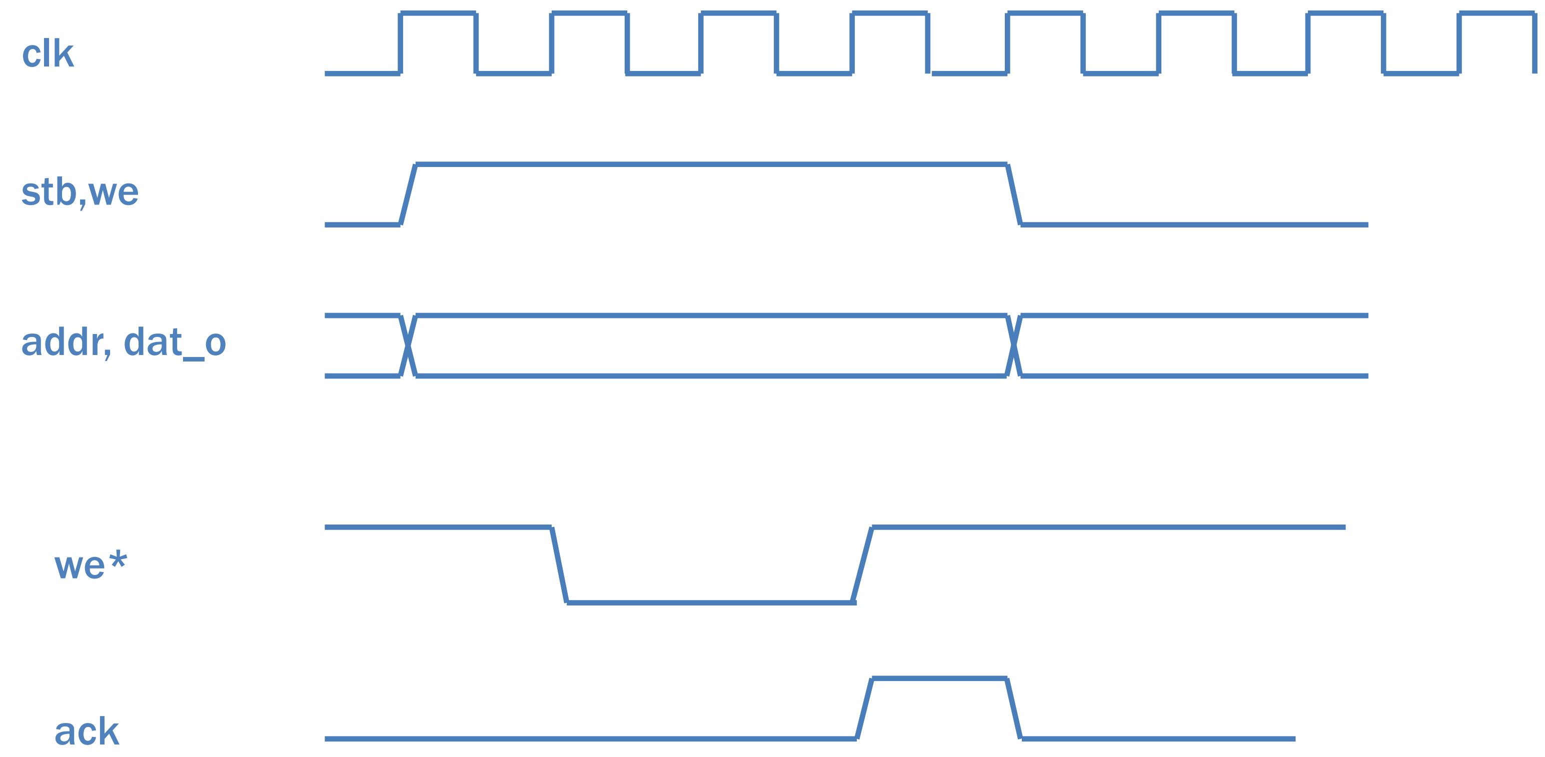


SRAM - Write



t_{WC}
ADDRESS
$\overline{CE}$
t_{SA}
t_{SCE}
t_{AW}
t_{HA}
t_{PWE}
$\overline{WE}$
t_{BW}
$\overline{BHE}$, $\overline{BLE}$
t_{SD}
t_{HD}
DATAI/O
1041

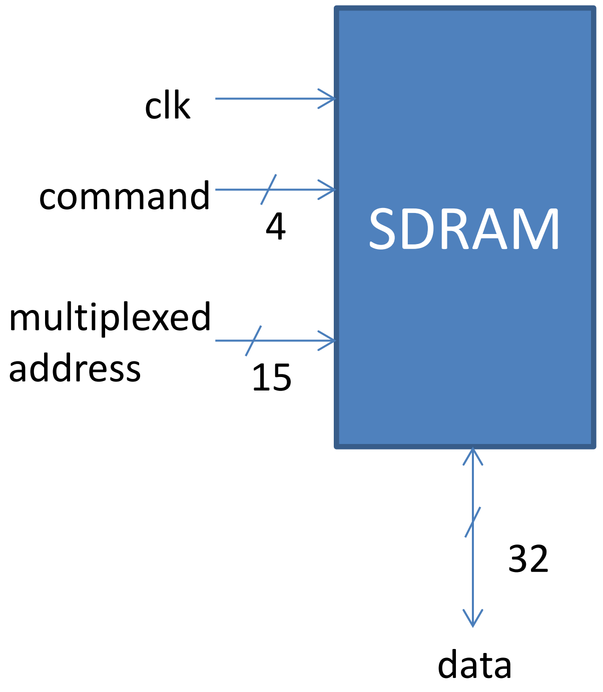
Some signals from PKMC



SDRAM

Synchronous Dynamic RAM

- Clocked device
- Memory element: Capacitance
- Needs periodic refreshing
- Pipelined operation
  - Needs a series of commands
  - No loss using muxed address
- Burst oriented
  - One address many data
- 2 x (16M x 16) = 64MB



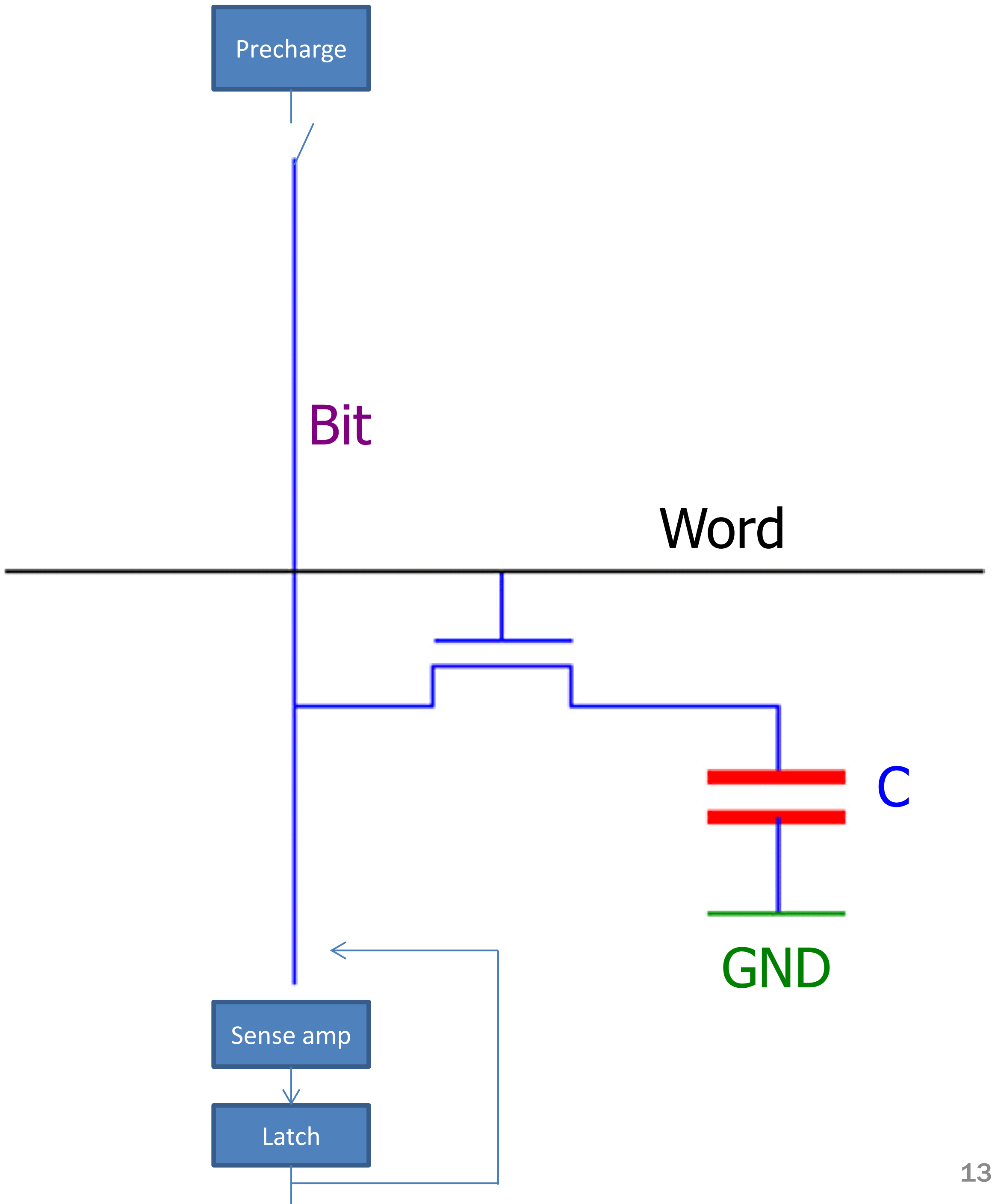
SDRAM

Sketchy read cycle

1. Precharge bit line to V/2
2. Let bit line float
3. Connect sense amp to bit line
4. Connect C to bit line
5. Hold value in latch
6. Write value back to C

Refresh cycle

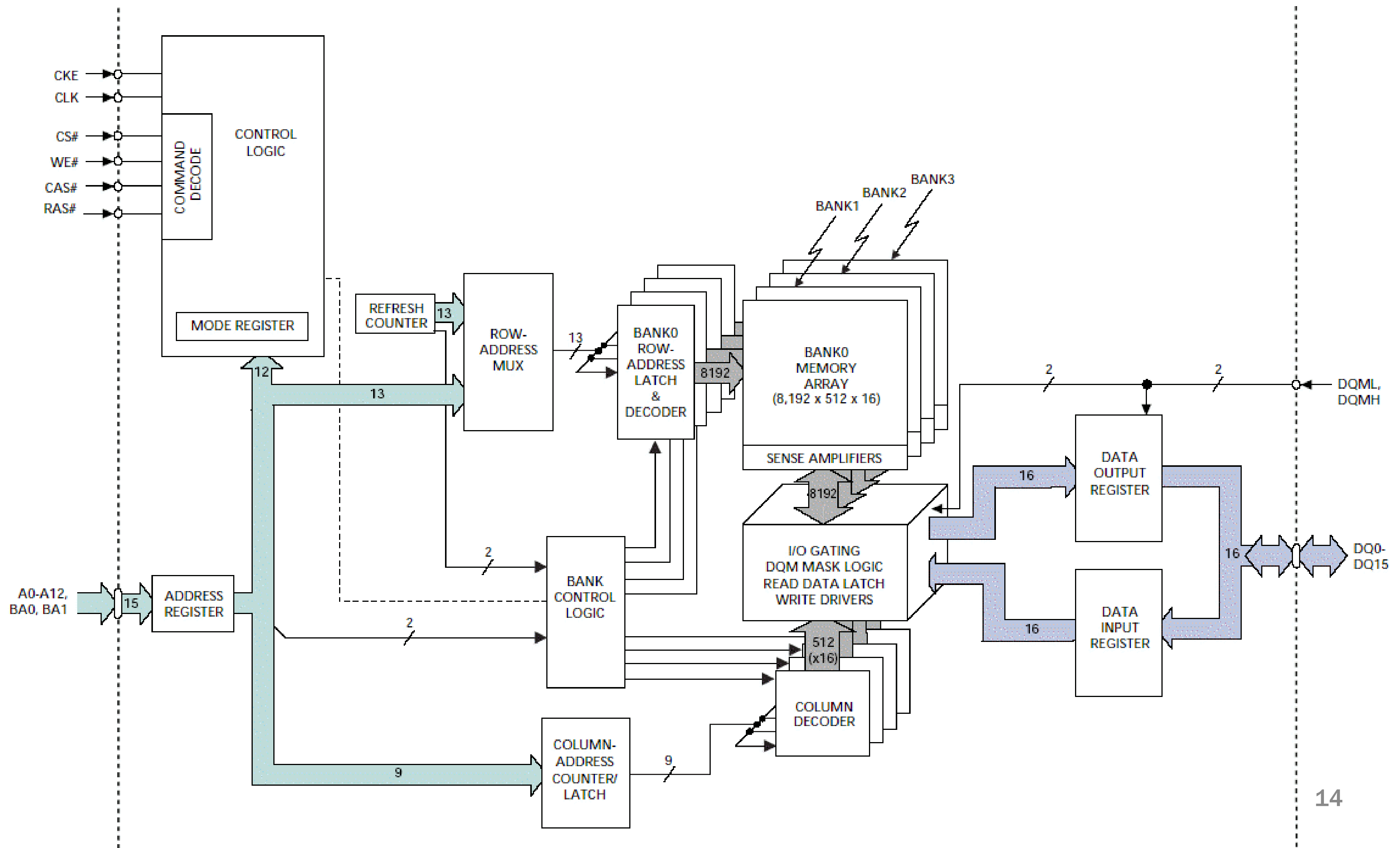
Dummy read cycle for a whole row



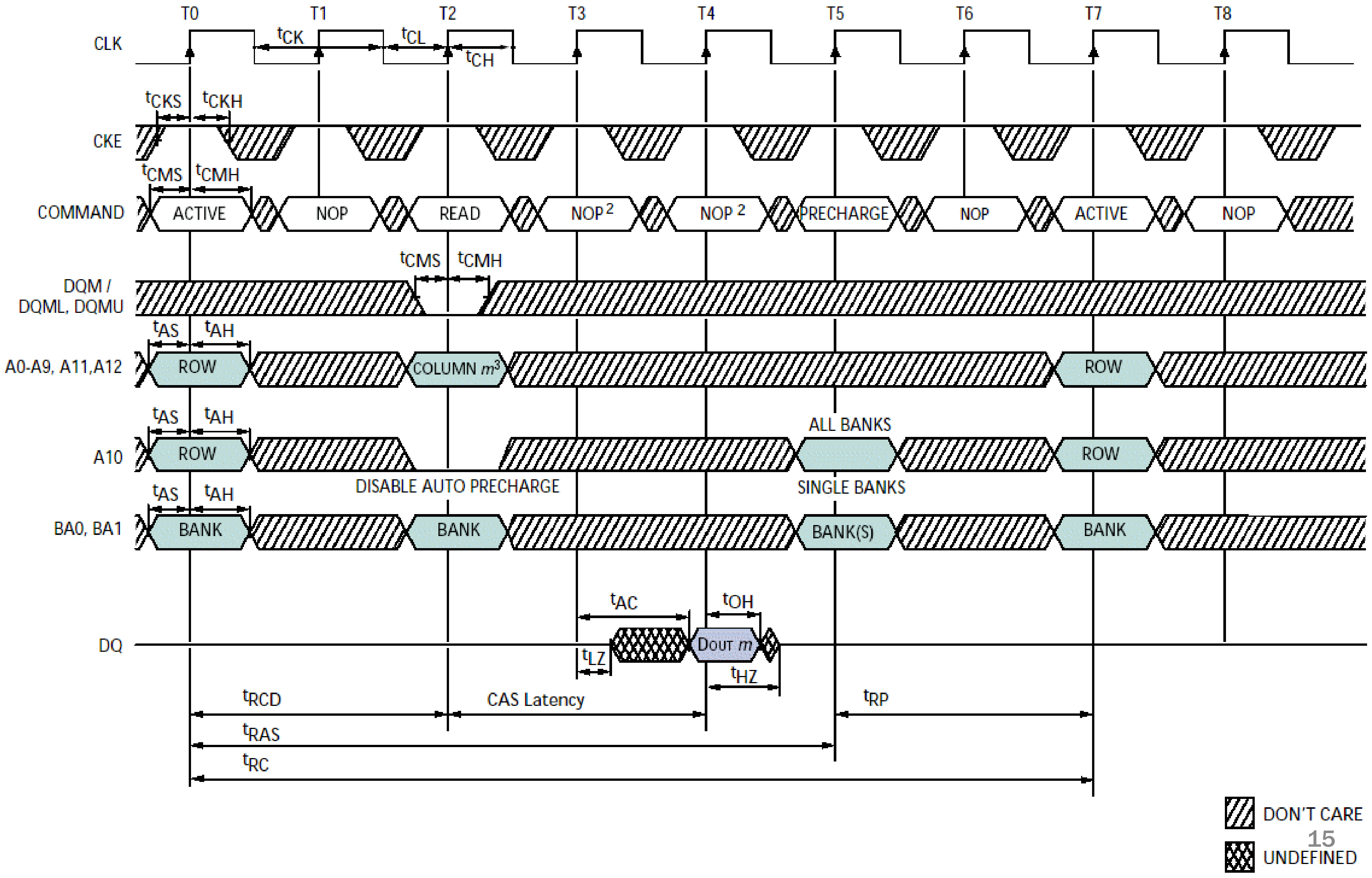
SDRAM Architecture

32 MB = 16 MW per chip, address bits = 24 = 13+11

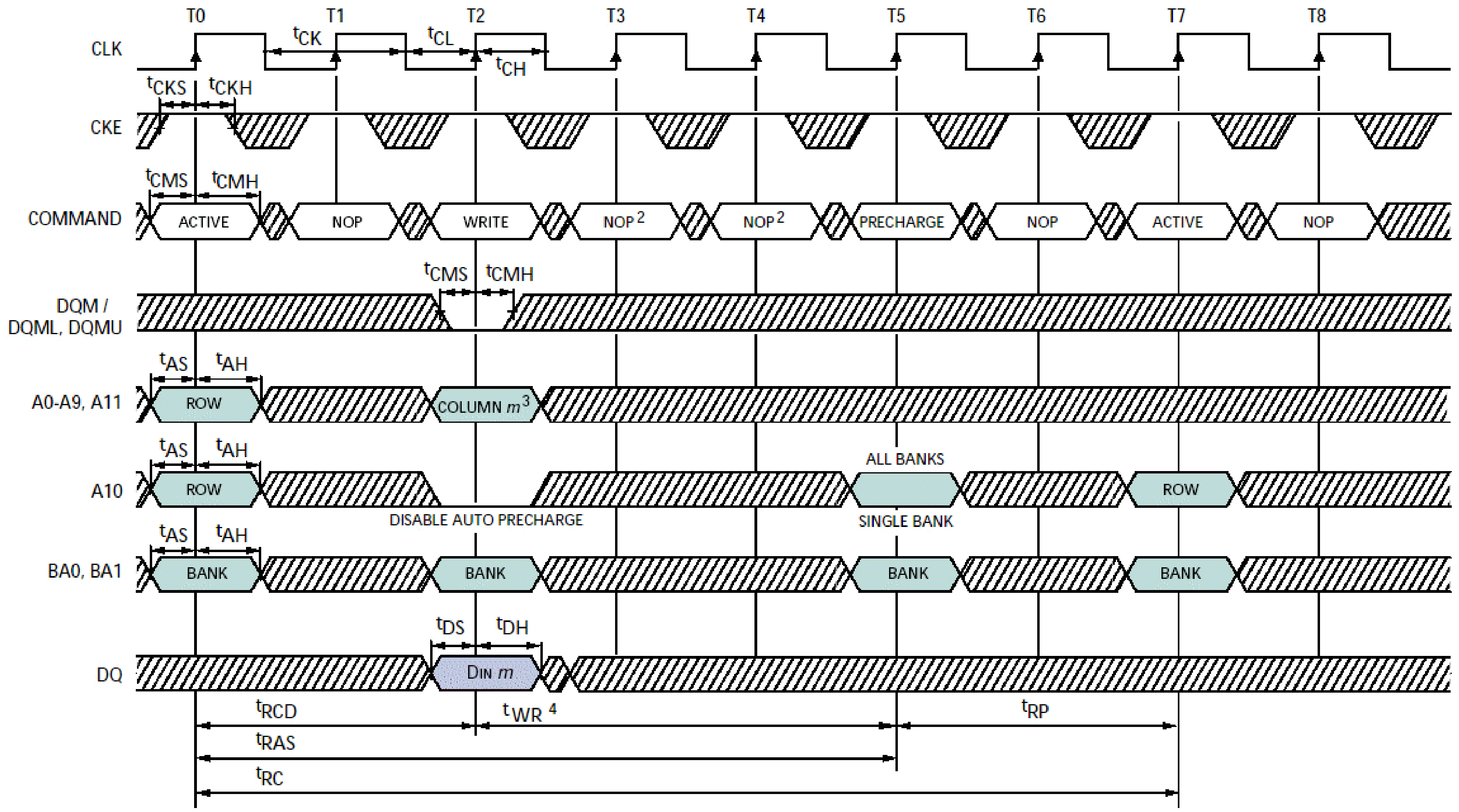
burst oriented



SDRAM Read


T0
T1
T2
T3
T4
T5
T6
T7
T8
CLK
tCK
tCL
tCH
tCKS
tCKH
CKE
tCMS
tCMH
COMMAND
ACTIVE
NOP
READ
NOP 2
NOP 2
PRECHARGE
NOP
ACTIVE
NOP
tCMS
tCMH
DQM / DQML, DQMU
tAS
tAH
A0-A9, A11,A12
ROW
COLUMN m3
ROW
tAS
tAH
A10
ROW
ALL BANKS
ROW
DISABLE AUTO PRECHARGE
SINGLE BANKS
tAS
tAH
BA0, BA1
BANK
BANK
BANK(S)
BANK
tAC
tOH
DQ
Dout m
tLZ
tHZ
tRCD
CAS Latency
tRP
tRAS
tRC
DON'T CARE
UNDEFINED

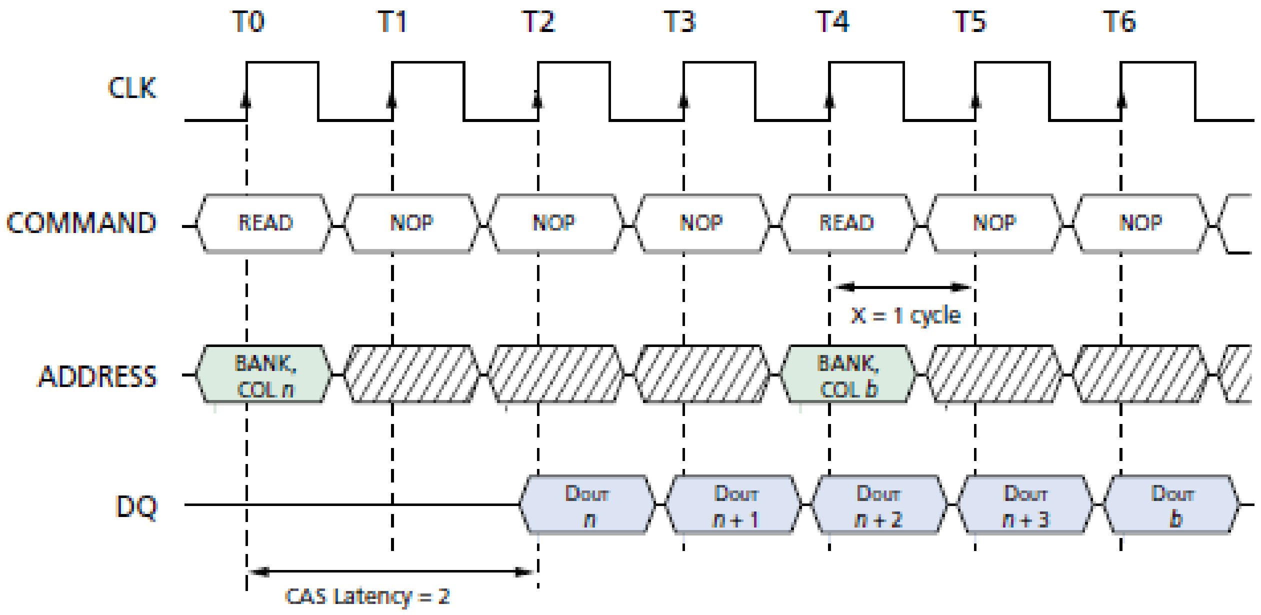
SDRAM Write



T0
T1
T2
T3
T4
T5
T6
T7
T8
CLK
tCK
tCL
tCH
tCKS
tCKH
CKE
tCMS
tCMH
COMMAND
ACTIVE
NOP
WRITE
NOP 2
NOP 2
PRECHARGE
NOP
ACTIVE
NOP
DQM / DQML, DQMU
A0-A9, A11
tAS
tAH
ROW
COLUMN m 3
A10
ROW
ALL BANKS
ROW
DISABLE AUTO PRECHARGE
SINGLE BANK
BA0, BA1
BANK
BANK
BANK
BANK
tDS
tDH
DQ
DIN m
tRCD
tWR 4
tRP
tRAS
tRC

Consecutive read bursts

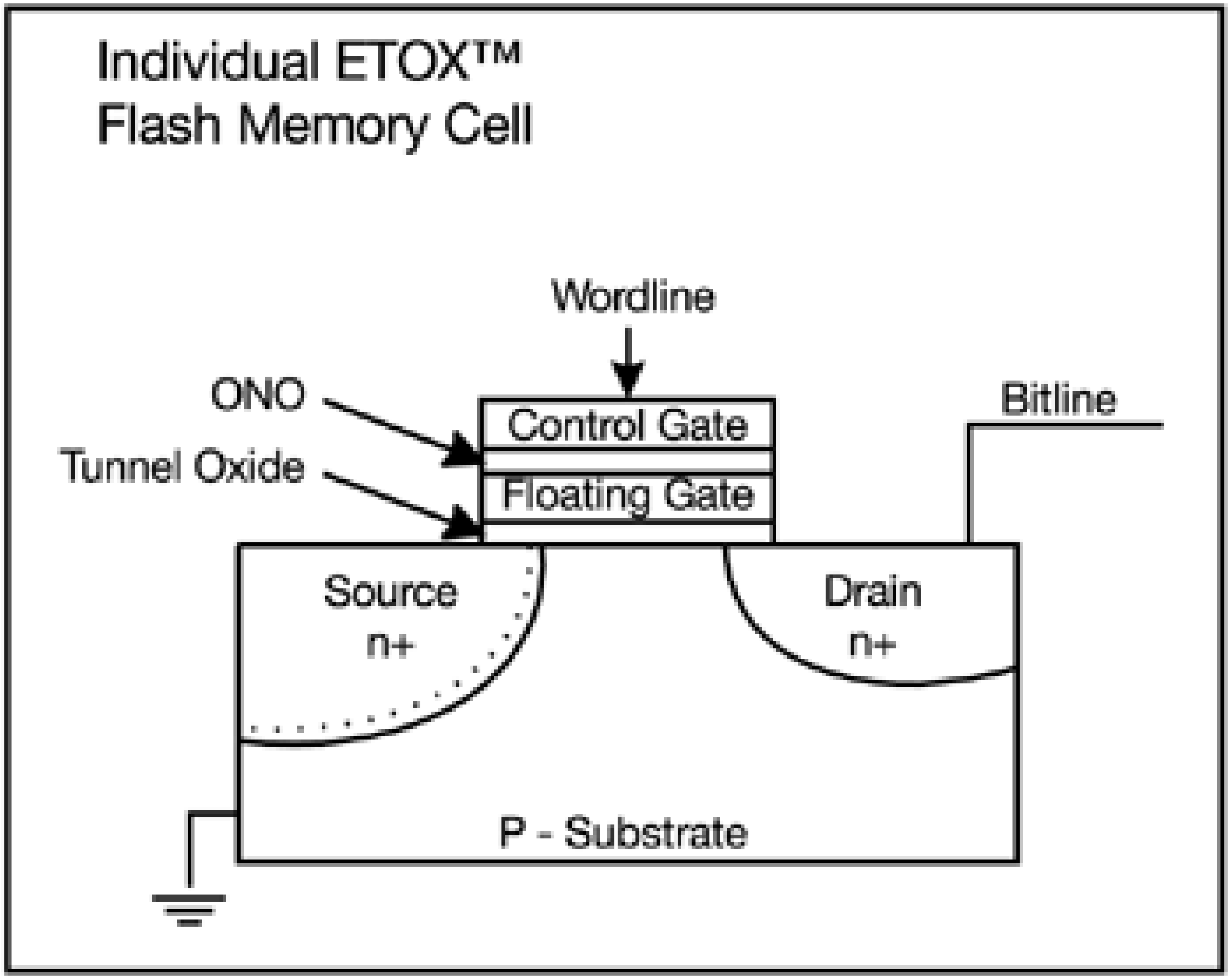
Mode register must be programmed



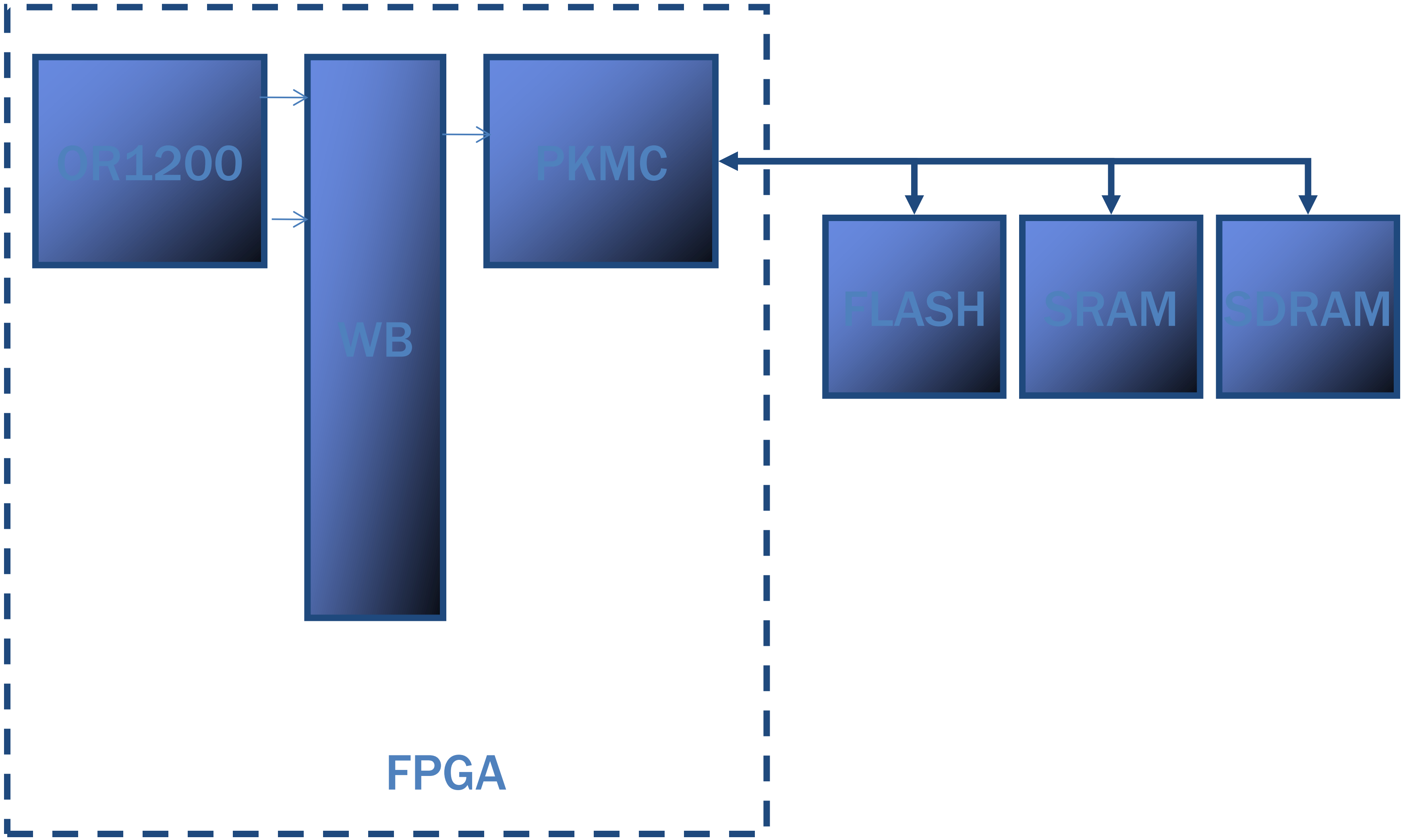
FLASH - Interface

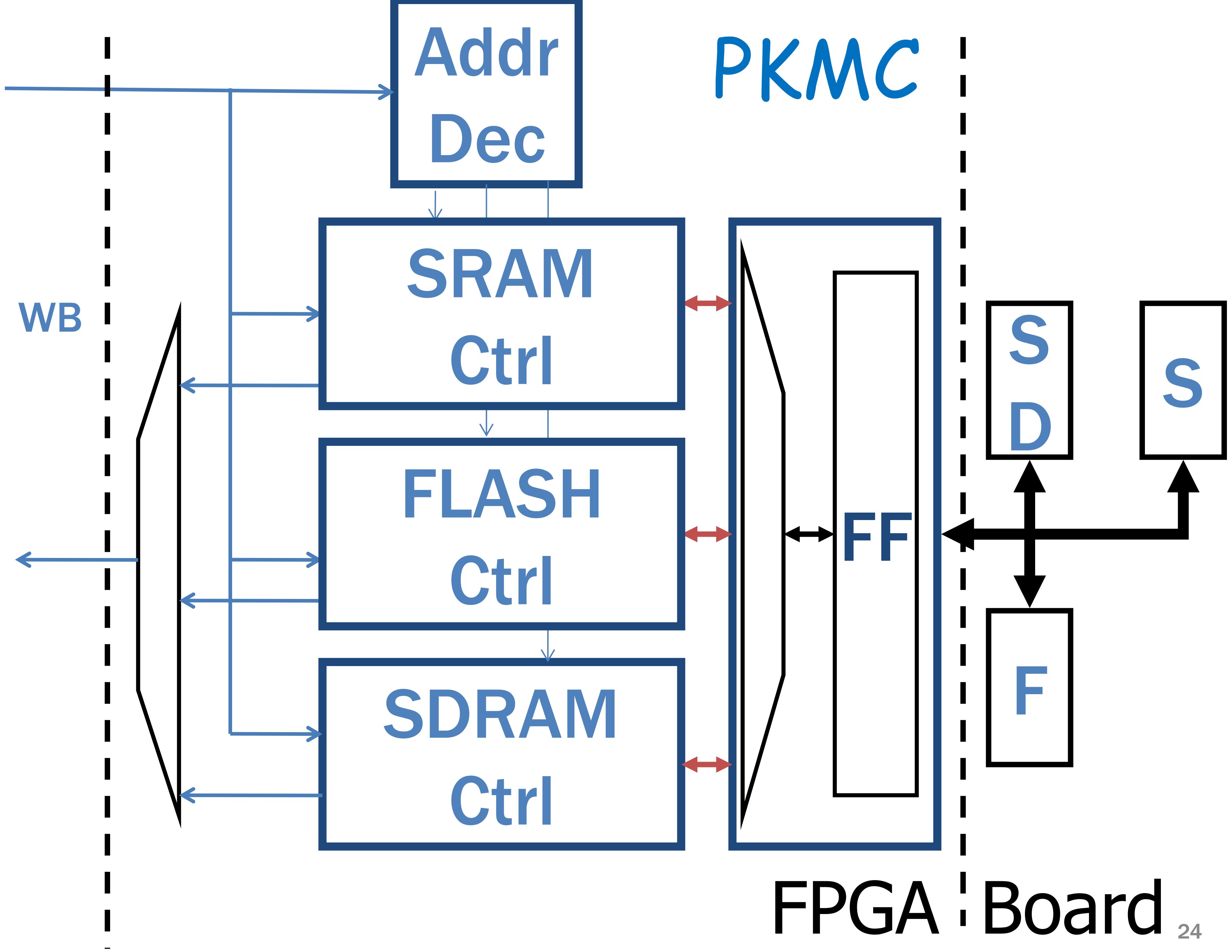
- Looks like SRAM
  - Read
- Write commands
  - Unlock a block
  - Erase a block
  - Program a block
- Contains uCLinux kernel +file system

FLASH - Cell

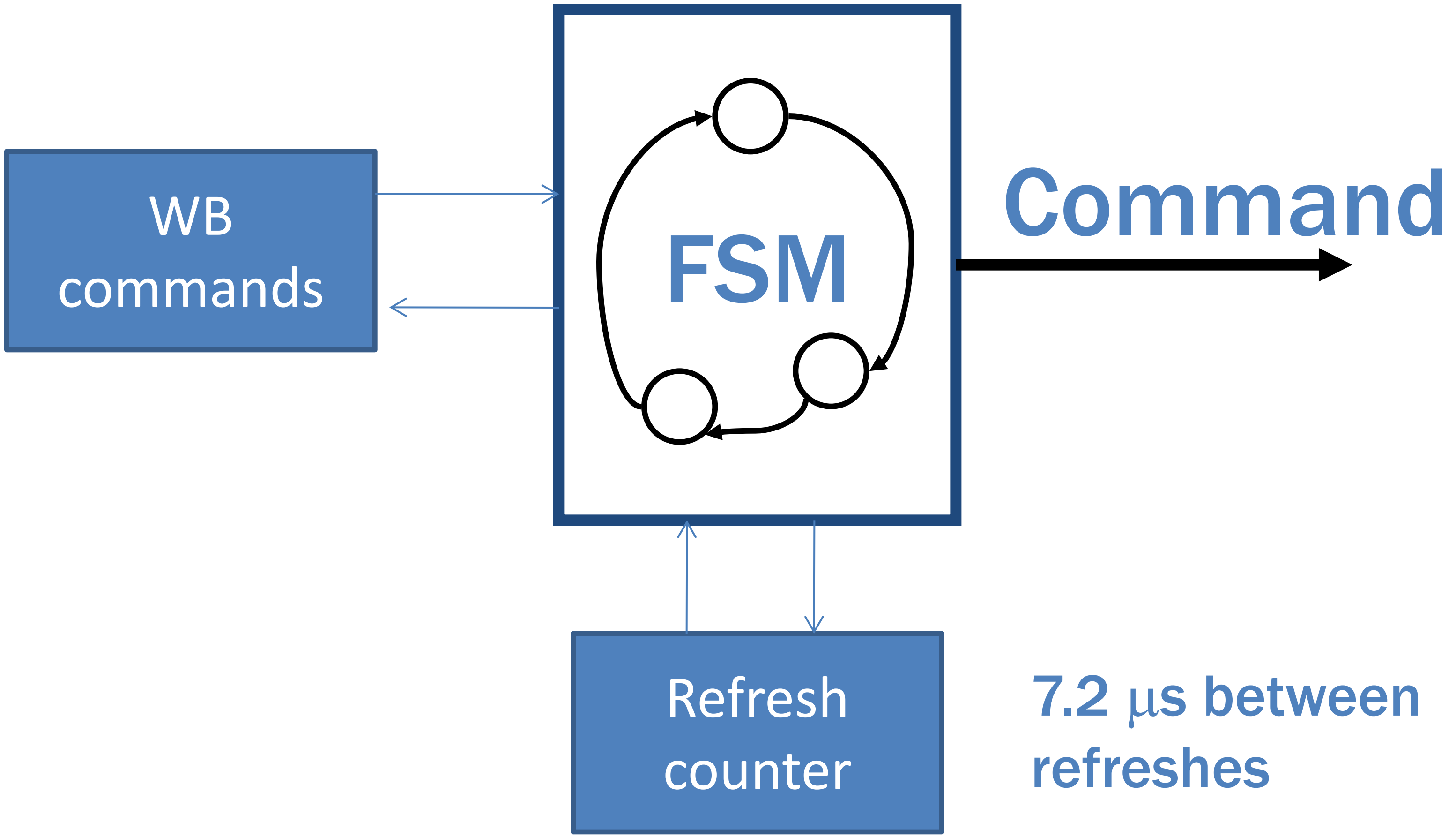

Individual ETOX™
Flash Memory Cell
Wordline
ONO
Tunnel Oxide
Control Gate
Floating Gate
Bitline
Source
n+
Drain
n+
P - Substrate

System Overwiev



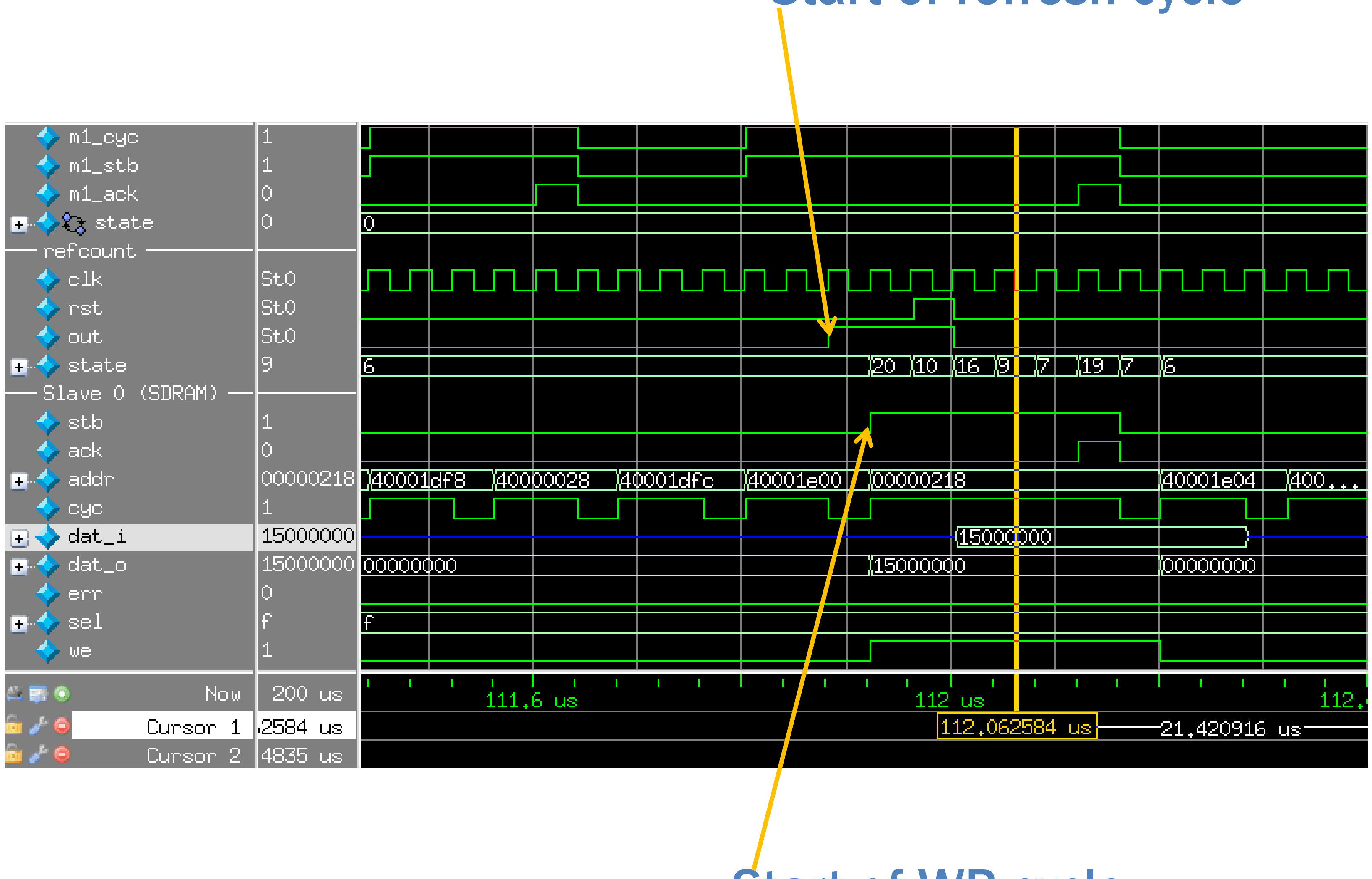

Addr Dec
PKMC
WB
SRAM Ctrl
FLASH Ctrl
SDRAM Ctrl
FF
SD
S
F
FPGA
Board

SDRAM Controller Internals



7.2 μs between refreshes

Refresh cycle



Signal	Value
m1_cyc	1
m1_stb	1
m1_ack	0
state	0
refcount	
clk	St0
rst	St0
out	St0
state	9
Slave 0 (SDRAM)	
stb	1
ack	0
addr	00000218
cyc	1
dat_i	15000000
dat_o	15000000
err	0
sel	f
we	1
Now	200 us
Cursor 1	.2584 us
Cursor 2	4835 us

Lab 4

A Custom Instruction

Huffman Encoding/Decoding

1) After Q (for instance):

$$\begin{bmatrix} 22 & 12 & 0 & -12 & 0 & 0 & 0 & 0 \\ 0 & 0 & -8 & 0 & 0 & 0 & 0 & 0 \\ 4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

2) After zig-zag:

22
12
0 4
0 0 -12
-8
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 1
0 … 0

3) After RLE

value	raw bits
05	10110
04	1100
13	100
24	0011
04	0111
F0	
F0	
D1	1
00	

magnitude

Run of 0:s

4) After Huffman

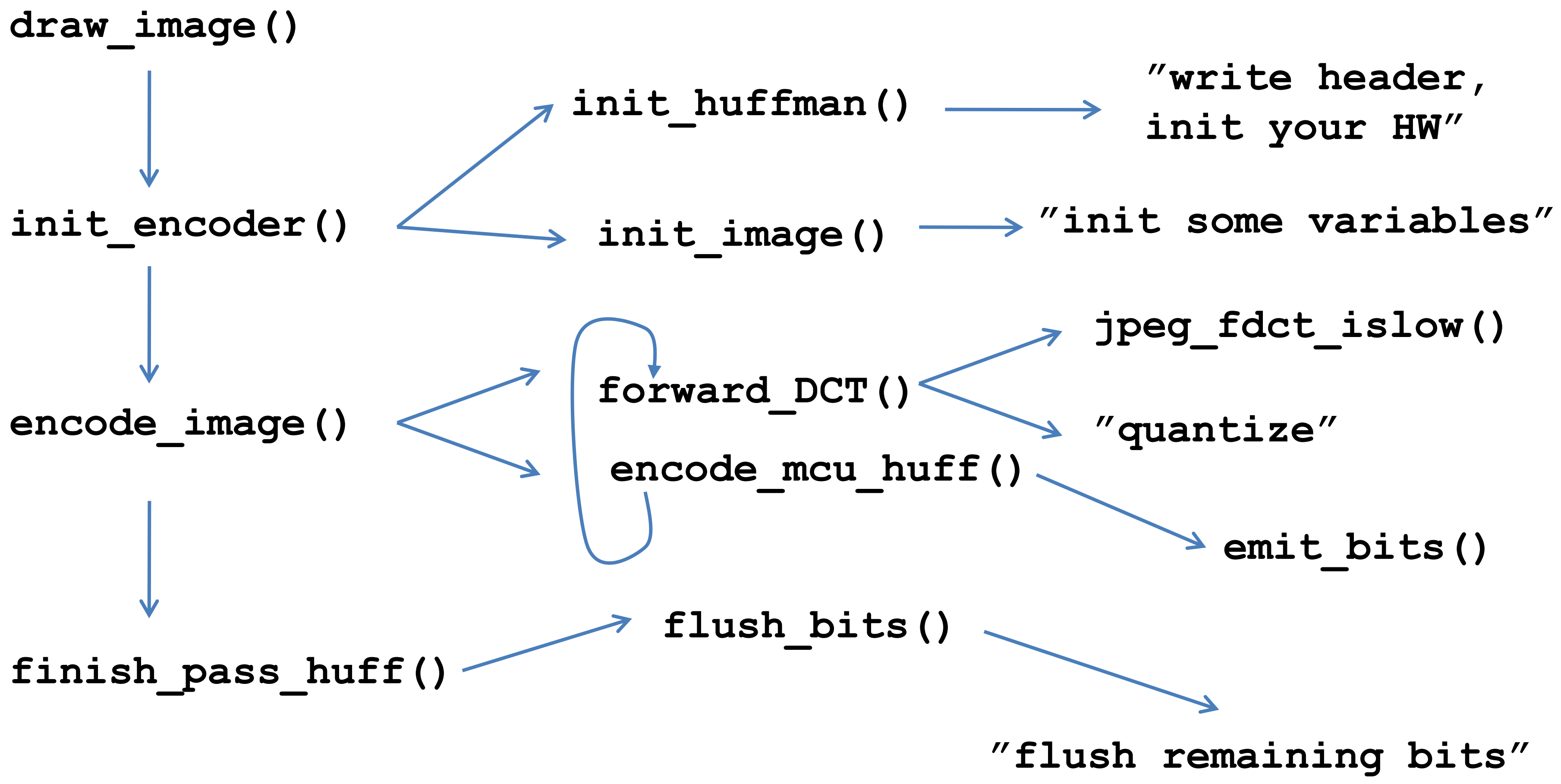
The values (bytes) are encoded by table lookup

Huffman in JFIF

- Output: 1 – 16 bits
- Encodes bytes
- 2 tables used
  - Y DC (differential)
  - Y AC

jpegfiles

jpegtest.c, jcdctmgr.c, jdct.c, jchuff.c



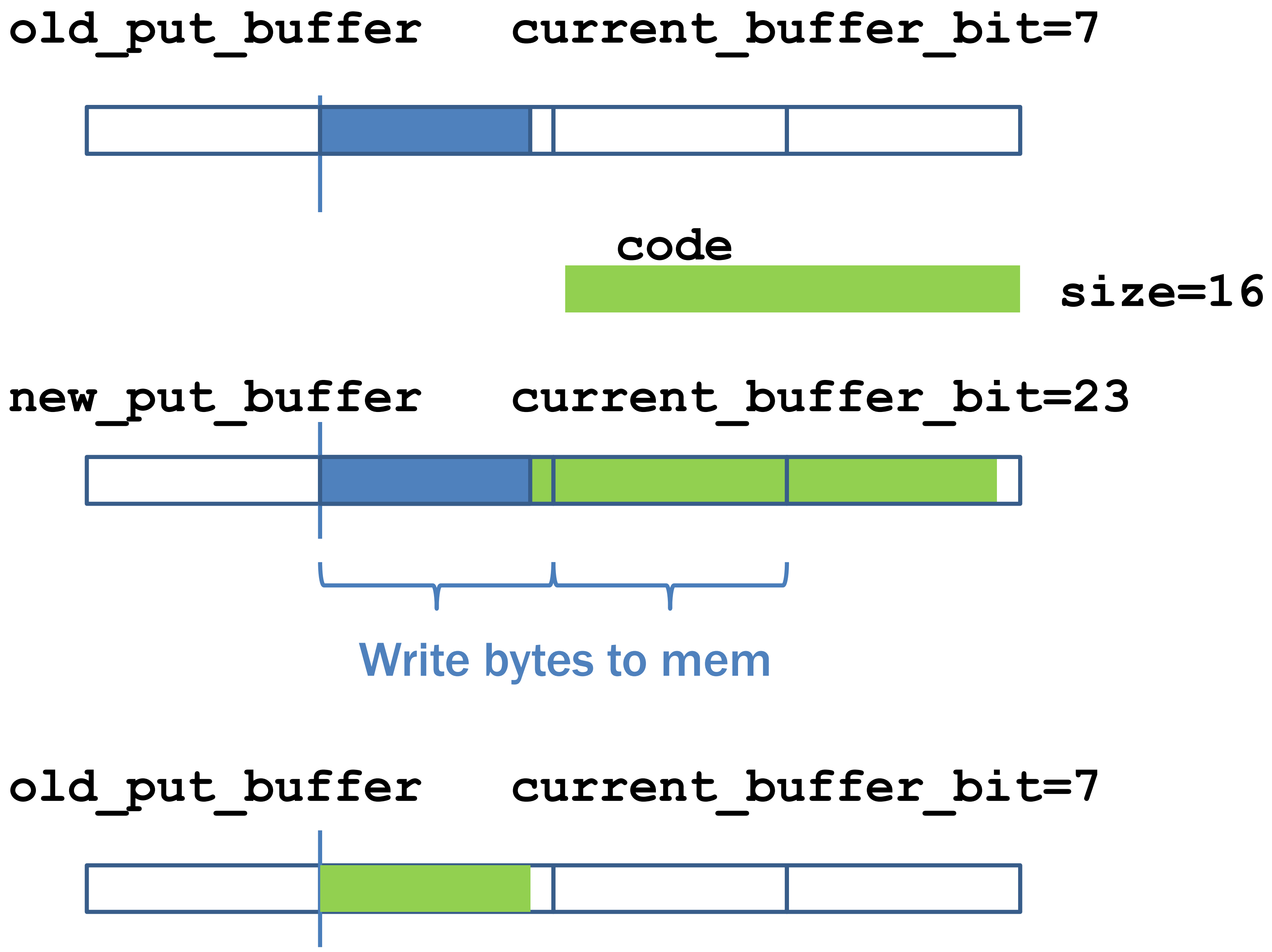
Emit_bits()

Max 16

```
static void emit_bits (unsigned int code, int size)
{
   unsigned int startcycle;

   new_put_buffer = (int) code;

// Add new bits to old bits. If at least 8 bits then write a char to buffer,
// save the rest until we get more bits.

   new_put_buffer &= (1<<size) - 1;                /* mask off any extra bits in code */
   current_buffer_bit += size;                      /* new number of bits in buffer */
   new_put_buffer = new_put_buffer << (24 - current_buffer_bit);     /* align incoming bits */
   new_put_buffer = new_put_buffer | old_put_buffer;    /* and merge with old buffer contents */

   while (current_buffer_bit >= 8) {
     int c = ((new_put_buffer >> 16) & 0xFF); // Mask out the 8 bits we want
     buffer[next_buffer] = (char) c;
     next_buffer++;
     if (c == 0xFF) {          // 0xFF is a reserved code for tags, if we get image data
        buffer[next_buffer] = 0x00;      // with an FF value it has to be followed by 0x00.
        next_buffer++;
    }
    new_put_buffer <<= 8;
    current_buffer_bit -= 8;
  }
  old_put_buffer = new_put_buffer; /* update state variables */
}
```

When **buffer** is complete, it is written to file

Emit_bits()


old_put_buffer
current_buffer_bit=7
code
size=16
new_put_buffer
current_buffer_bit=23
Write bytes to mem
old_put_buffer
current_buffer_bit=7

Adding an Instruction

1. Instruction Selection
2. Hardware modification
3. (Assembler modification)
4. (Compiler modification)

Instruction Selection

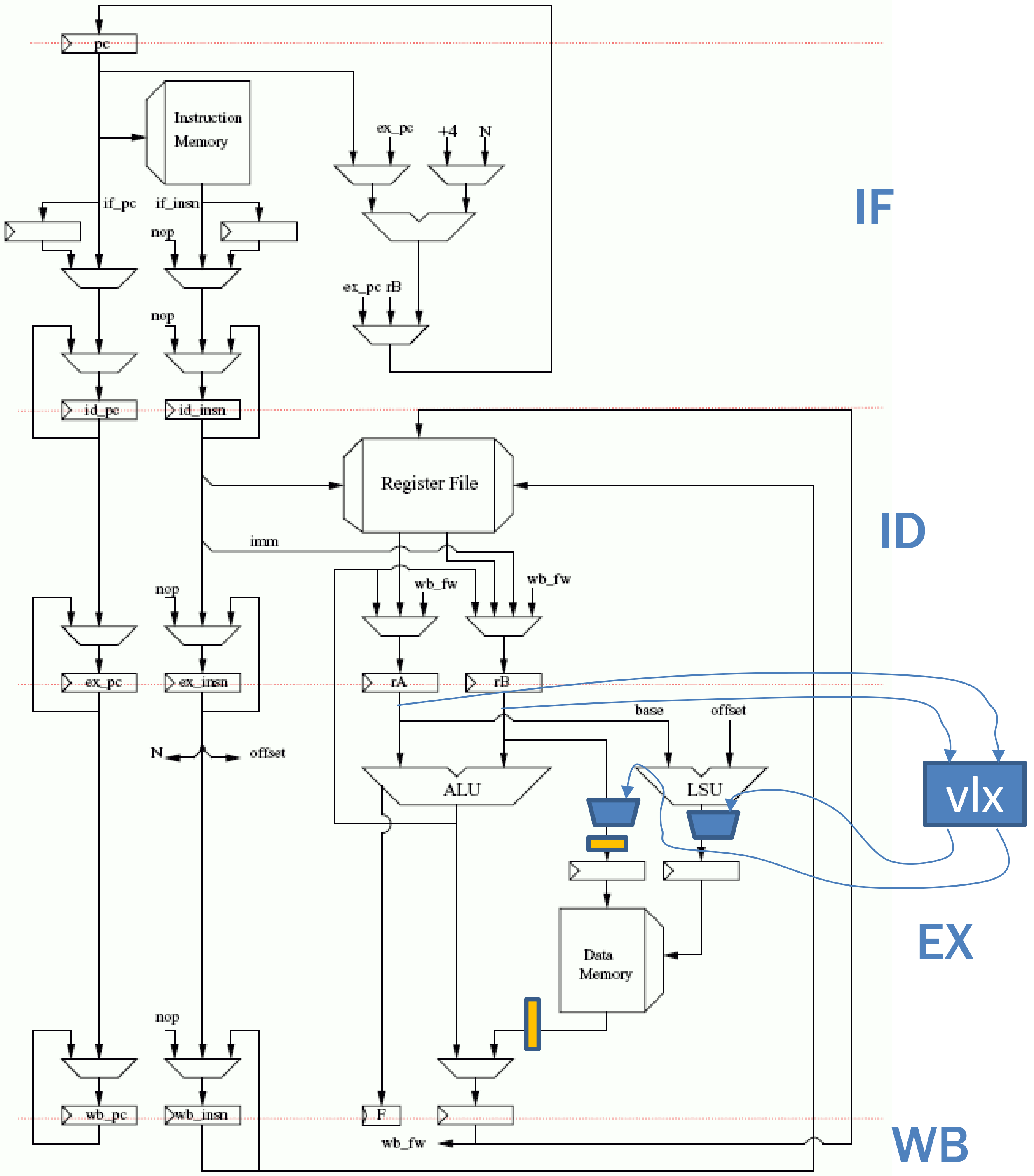
- **`l.custx`**
  - No operands
- Instructions for 64 bit
  - Not used
  - Assembler can understand
  - **`l.sd I(rA),rB`**

We hijack this instruction

Hardware Modifications

- Instruction decoder modifications
  - Legal instruction
  - or1200_ctrl.v
- Special purpose register
  - New group
  - or1200_sprs.v
- Data path
  - New hardware
  - or1200_lsu.v
  - or1200_vlx_top.v

or1200 Pipeline



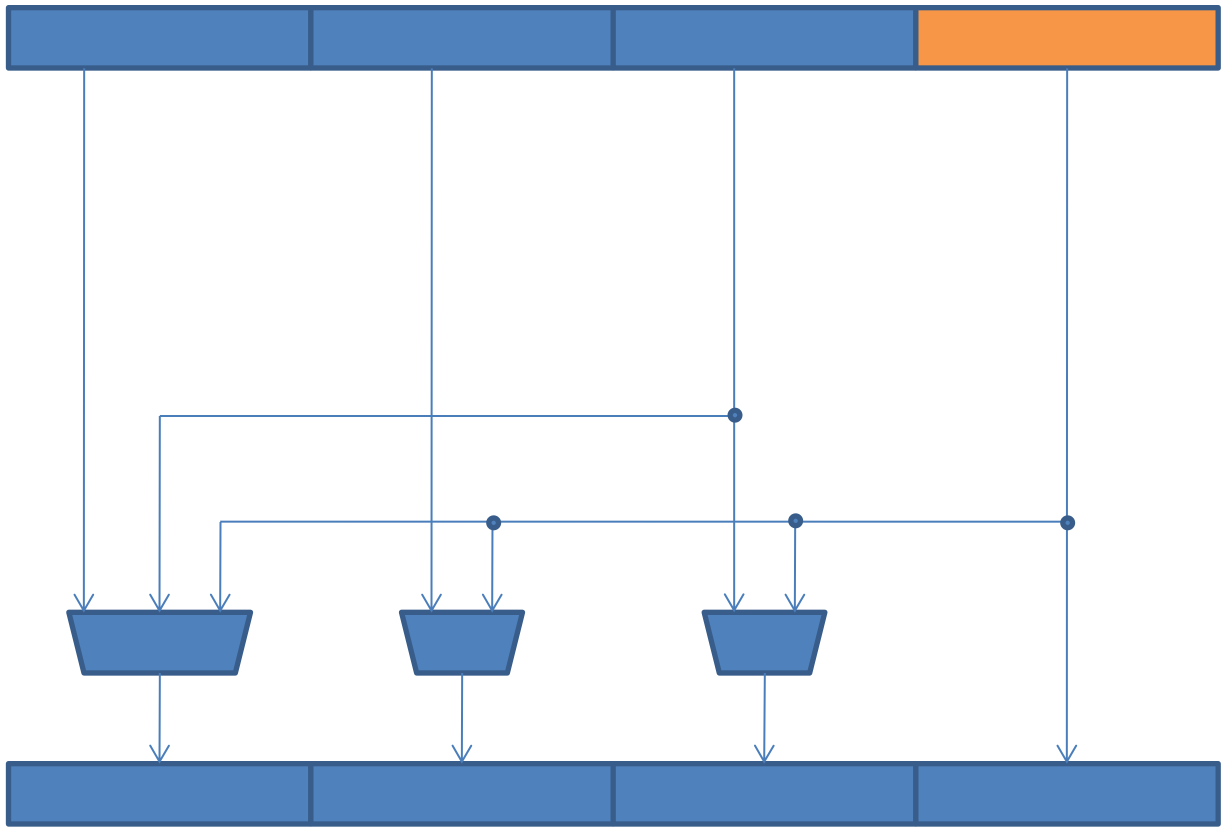
code size

l.sd (rA),rB

▬ align, reg2mem

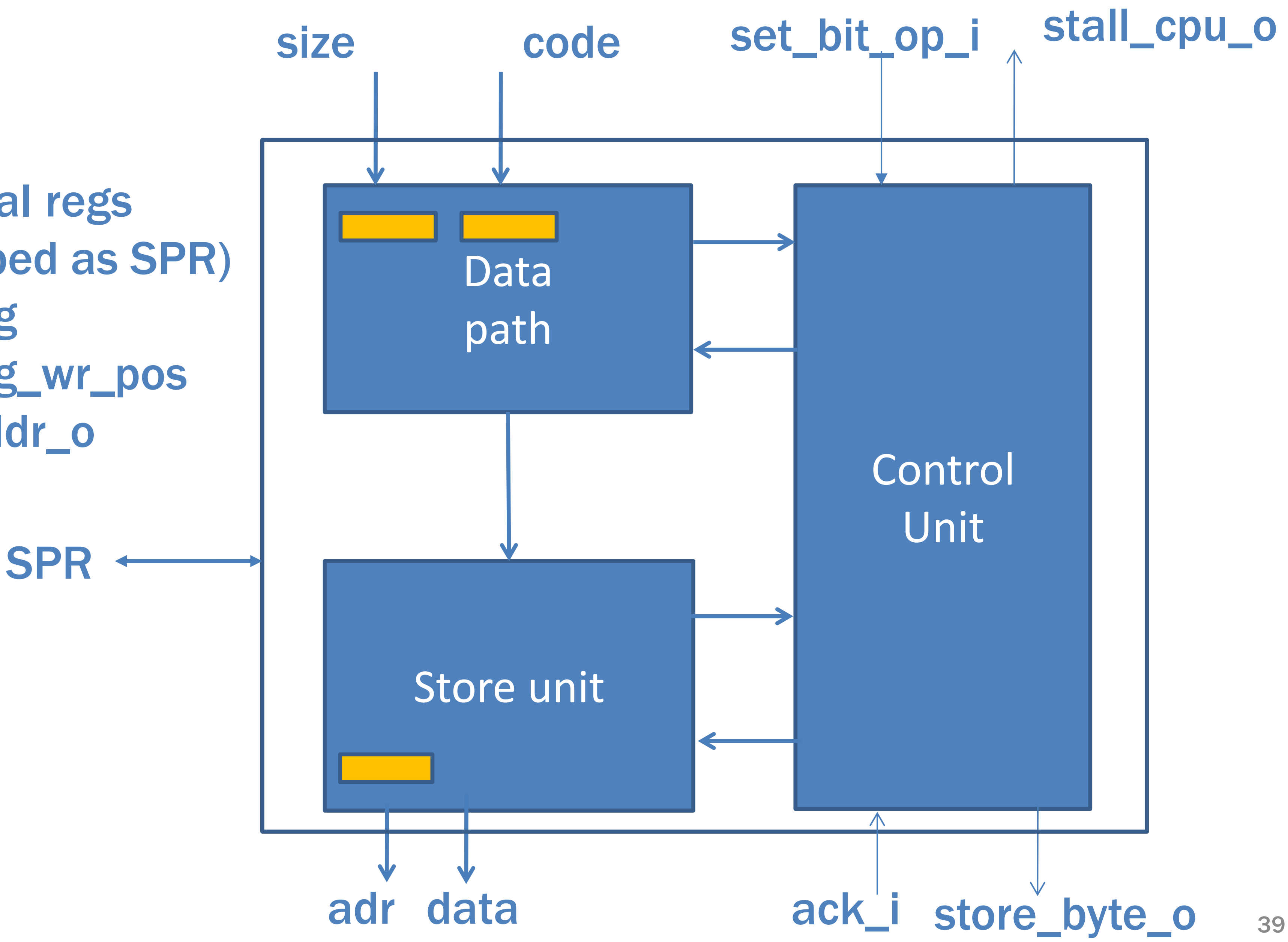
▮ align, mem2reg

Align reg2mem
(for byte write)

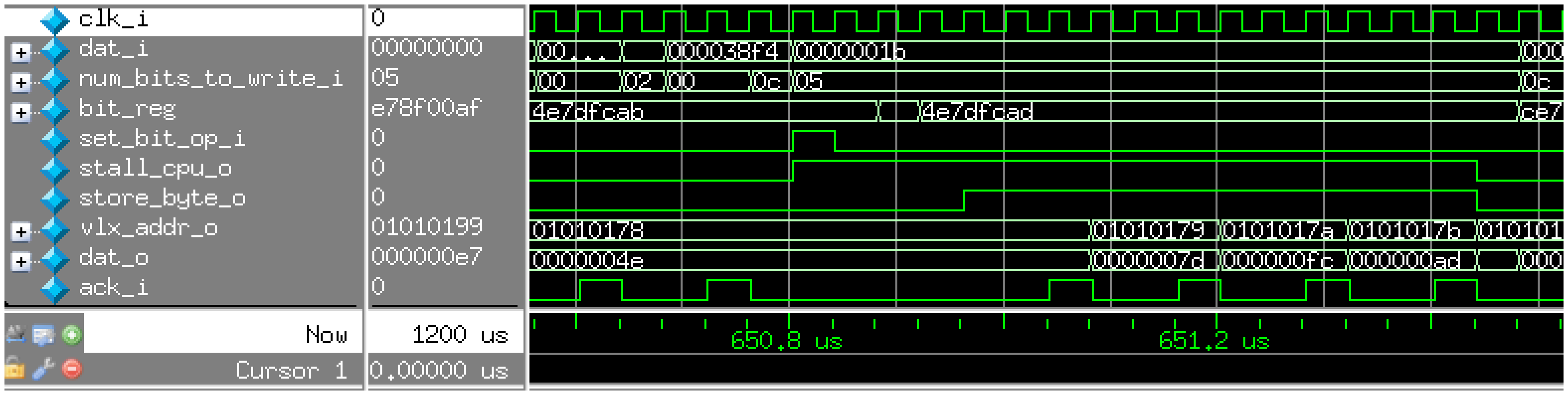


Proposed Architecture

Internal regs
(mapped as SPR)
bit_reg
bit_reg_wr_pos
vlx_addr_o



An example: PK:s implementation



```
clk_i                 0
dat_i                 00000000
num_bits_to_write_i   05
bit_reg               e78f00af
set_bit_op_i          0
stall_cpu_o           0
store_byte_o          0
vlx_addr_o            01010199
dat_o                 000000e7
ack_i                 0
                Now   1200 us
           Cursor 1   0.00000 us
```

Inline asm

In jpegfiles insert:

template

```
asm volatile("l.sd 0x0(%0),%1" : : "r"(code), "r"(size));
```

input

input

=> `code` and `size` will show up at your vlx

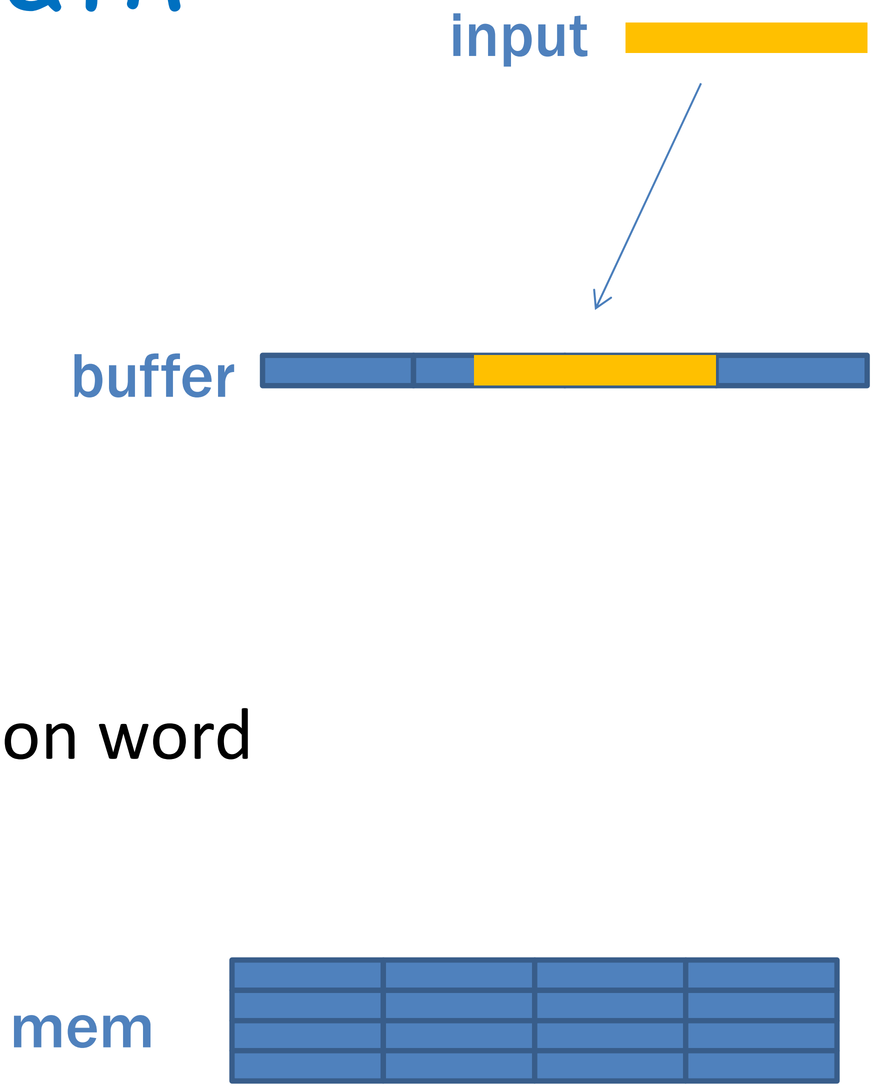
SPR = special purpose registers

- is an internal bus in the CPU for control and status
- two instructions
  - `mtspr K(rD),rA`
  - `mfspr rD,K(rA)`
- simple bus `spr_cs, spr_write, spr_addr, spr_dat_i, spr_dat_o`
- `Connect 3 registers to SPR bus!`

```
// copy from address register to variable pos
asm volatile("l.mfspr %0,%1,0x2":"=r"(pos):"r"(0xc000));

// zero bit counter
asm volatile("l.mtspr %0,%1,0x1":"=r"(0xc000):"r"(0x0));
```

Data Path

- Fill buffer
  - One bit / clock cycle
  - All bits at once
- Write to mem
  - One byte
  - One 32 bit word, must be on word boundaries (multiple of 4)



Control

- May not be needed
- May be an FSM

Store Unit

- Stores the data
- 0xFF stored as 0xFF00
  - Jpeg markers
- Only byte alignment!
  - Parallel stores faster

Software

- New Assembler
  - Easy
- New Compiler
  - Hard problem for complex instructions
  - Compiler knows functions
- C
  - Inline Assembler

Instruction Usage

```
unsigned char* sb_get_buff_pos(void)
{
                                                    output
    unsigned char* pos;
    asm volatile("l.mfspr %0,%1,0x2":"=r"(pos):"r"(0xc000));
    return pos;
}
```

```
    00000250 <_sb_get_buff_pos>:
     250:9c 21 ff fc     l.addi r1,r1,0xfffffffc
     254:d4 01 10 00     l.sw 0x0(r1),r2
     258:9c 41 00 04     l.addi r2,r1,0x4
     25c:a9 60 c0 00     l.ori r11,r0,0xc000
     260:b5 6b 00 02     l.mfspr r11,r11,0x2
     264:84 41 00 00     l.lwz r2,0x0(r1)
     268:44 00 48 00     l.jr r9
     26c:9c 21 00 04     l.addi r1,r1,0x4
```