

TSEA44: Computer hardware – a system on a chip

Lecture 8: Memories, lab4



Today

- Memories/memory controller
- Lab4, new instruction

Practical info

- Lab closed after 22/12
 - Opens again after new year (probably after 2/1-17)
 - Ask me or Erik to let you in (if we are at the work)
 - Remote login still works
 - Office corridors locked during christmas/new year
 - Hard to get access to people (if they are not on vacation)
 - Lab used for other courses in the spring
 - No access guaranteed after the course end
 - Will try to set up some limited access location
 - Probably a lab location with limited access only on non-scheduled hours
-

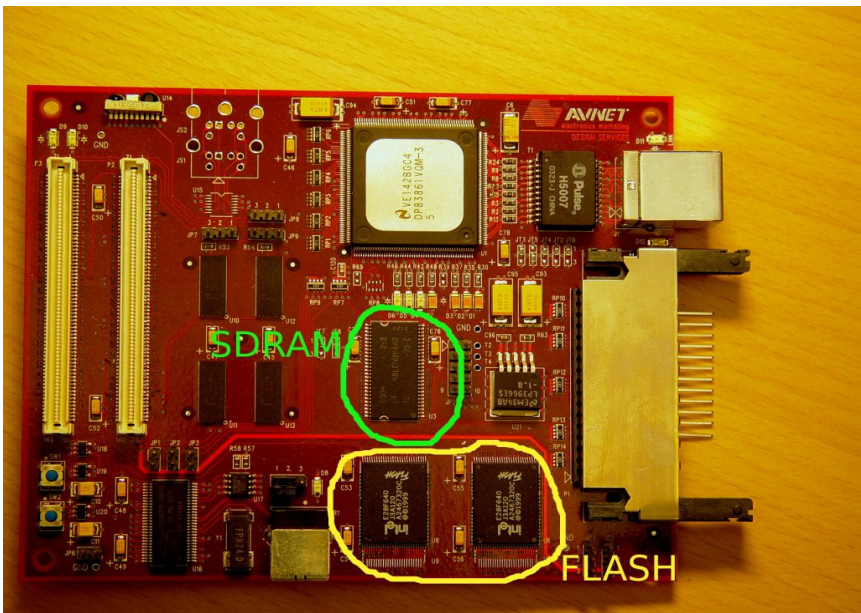
PKMC

Wishbone bus

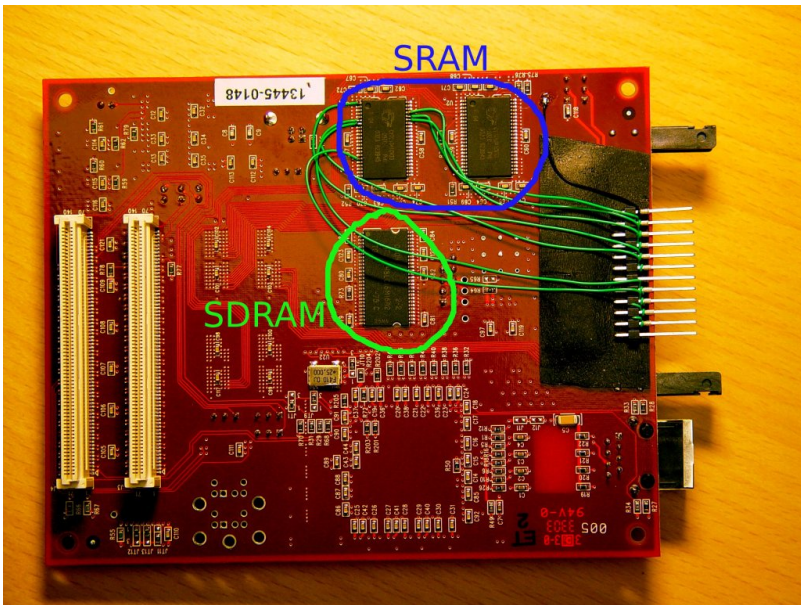
Memory bus



Memory on board



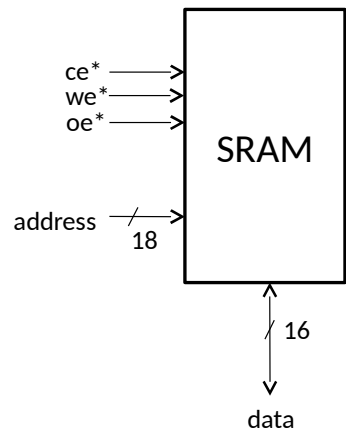
Memory on board, cont.



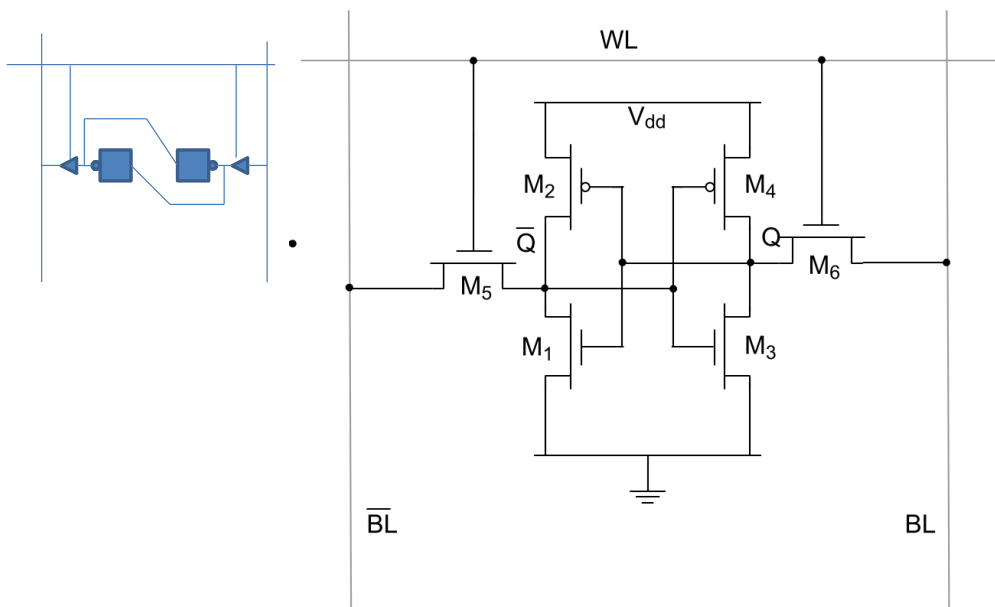
SRAM; Static RAM

- Asynchronous device
- Memory element: latch
- $2 \times (256k \times 16) = 1 \text{ MB}$

* = active low

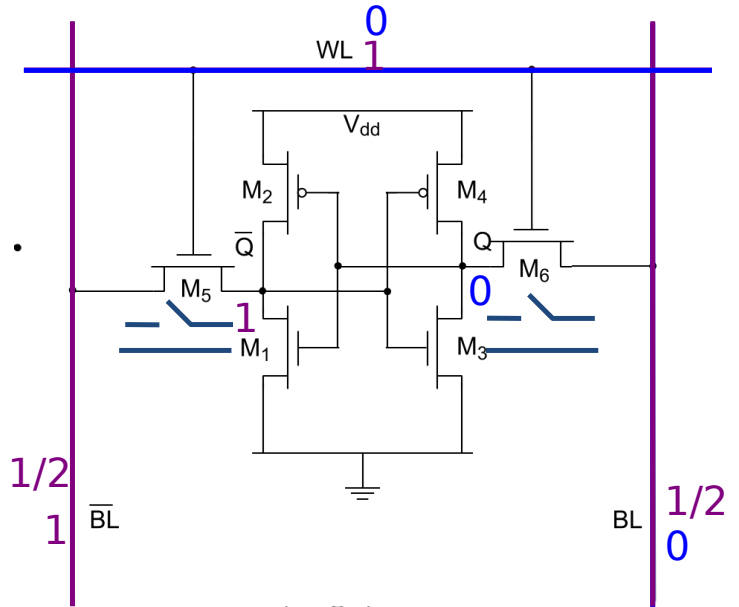


SRAM - Cell



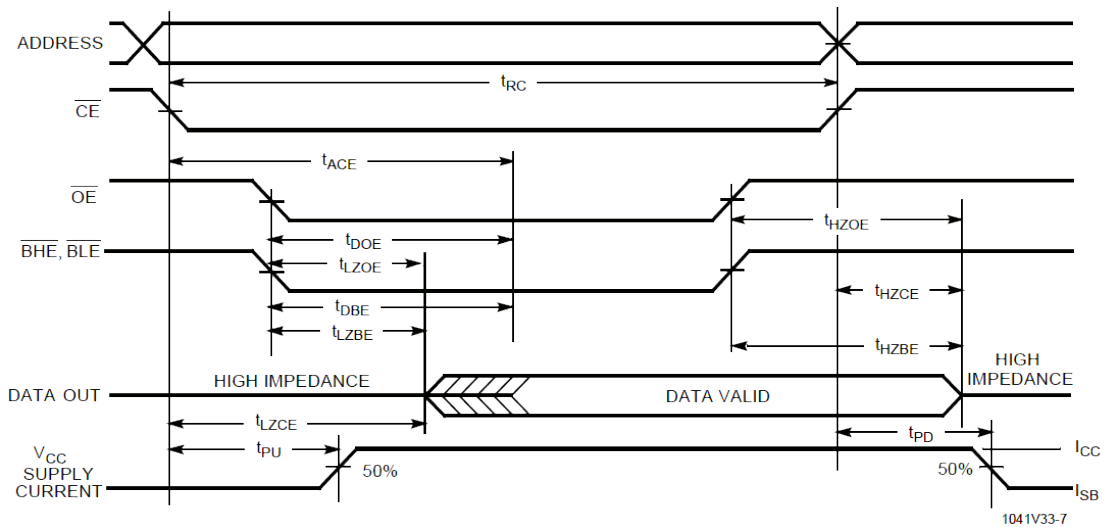
SRAM - Read

- WL=0, Precharge bitlines to $V_{dd}/2$
- WL=1 connects inverters to bitlines
- Bitlines are driven to low and high



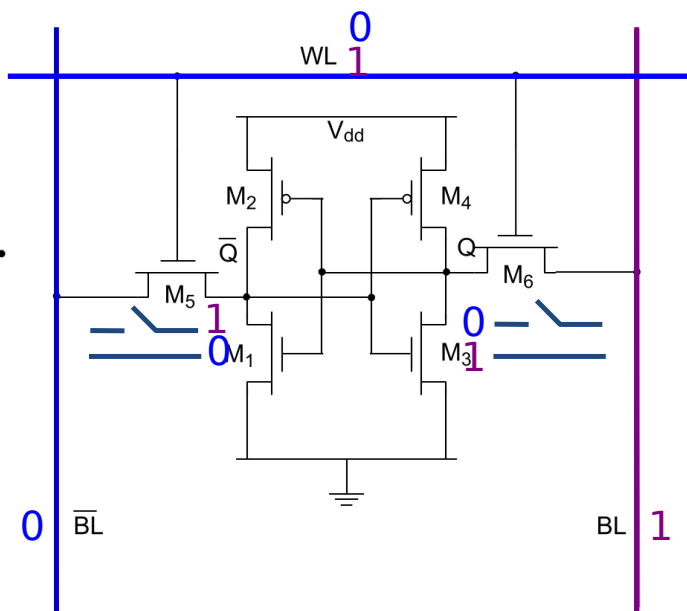
SRAM - Read

- No clocking!



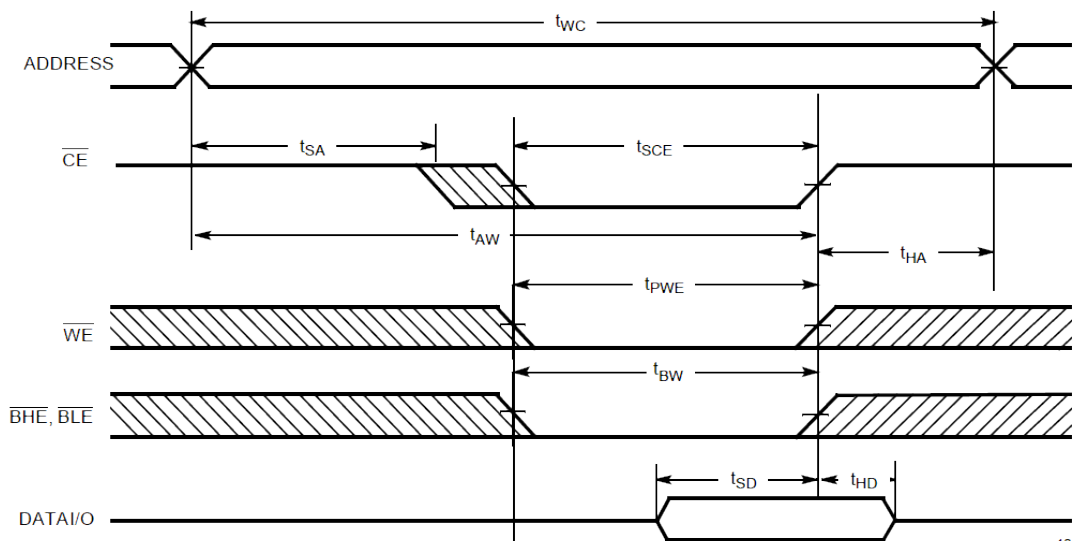
SRAM - Write

- $WL=0$,
Set logic values
on bitlines
- $WL=1$
connects
inverters
to bitlines
- Bitline values
override
internal
value



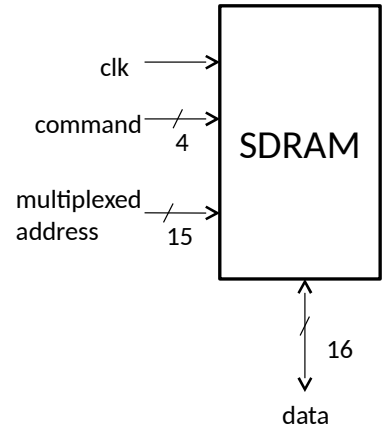
SRAM - Write

- Timing important (avoid writing to wrong address)



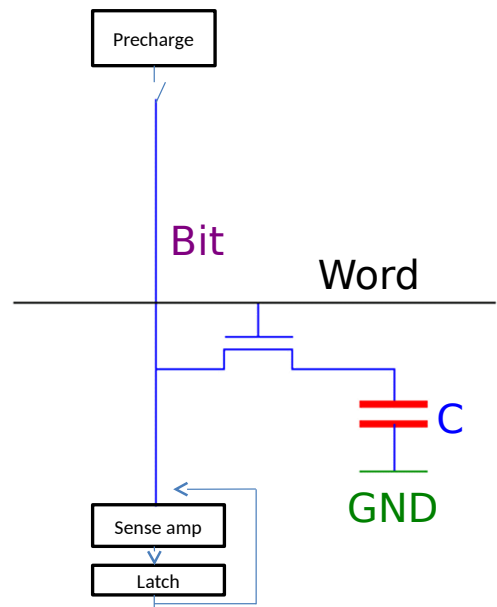
SDRAM; Synchronous Dynamic RAM

- Clocked device
- Memory element: Capacitance
- Needs periodic refreshing
- Pipelined operation
- Burst oriented
 - Single burst in our design
- $2 \times (16\text{M} \times 16) = 64\text{MB}$

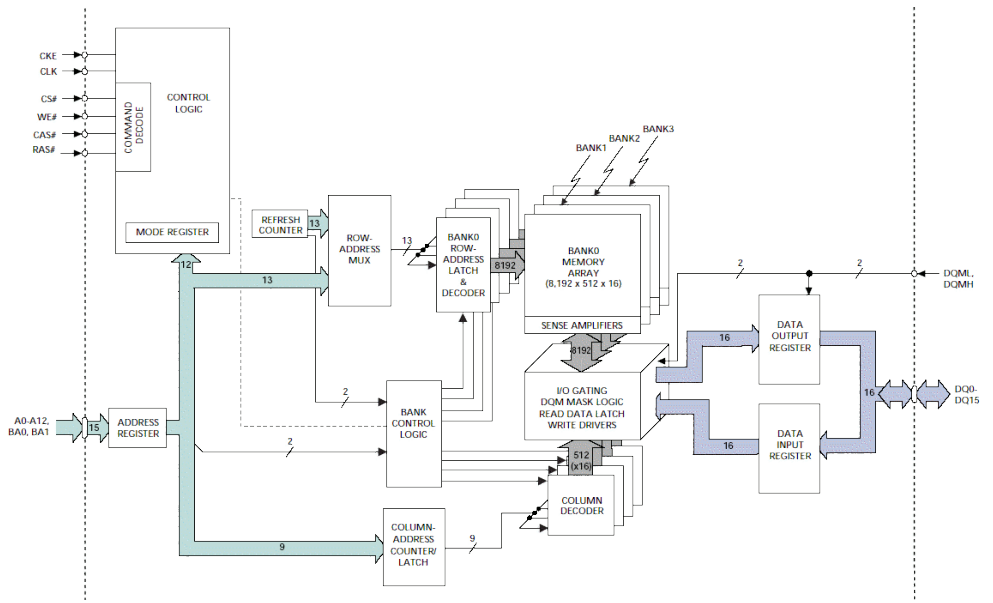


A measurement: make sim_jpeg

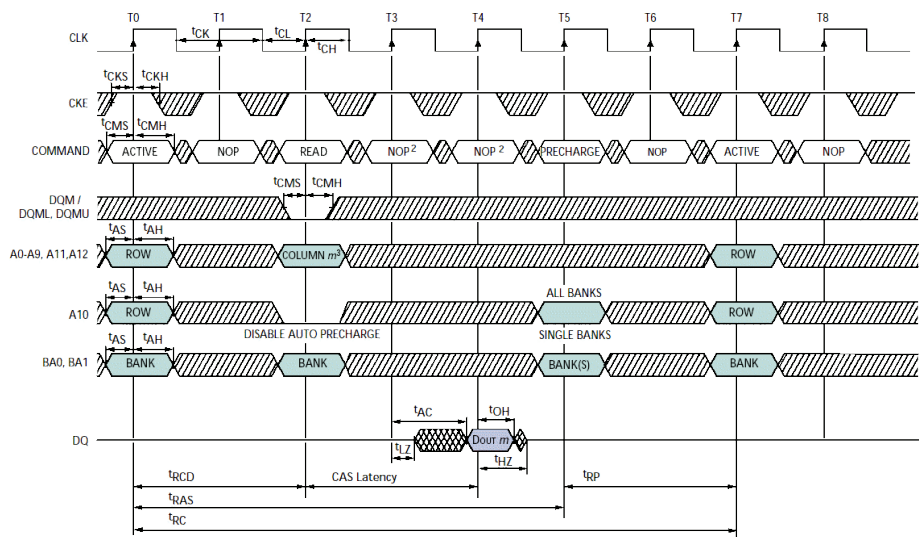
- Sketchy read cycle
 - Precharge bit line to $V_{dd}/2$
 - Let bit line float
 - Connect sense amp to bit line
 - Connect C to bit line
 - Hold value in latch
 - Write value back to C
- Refresh cycle
 - Dummy read cycle



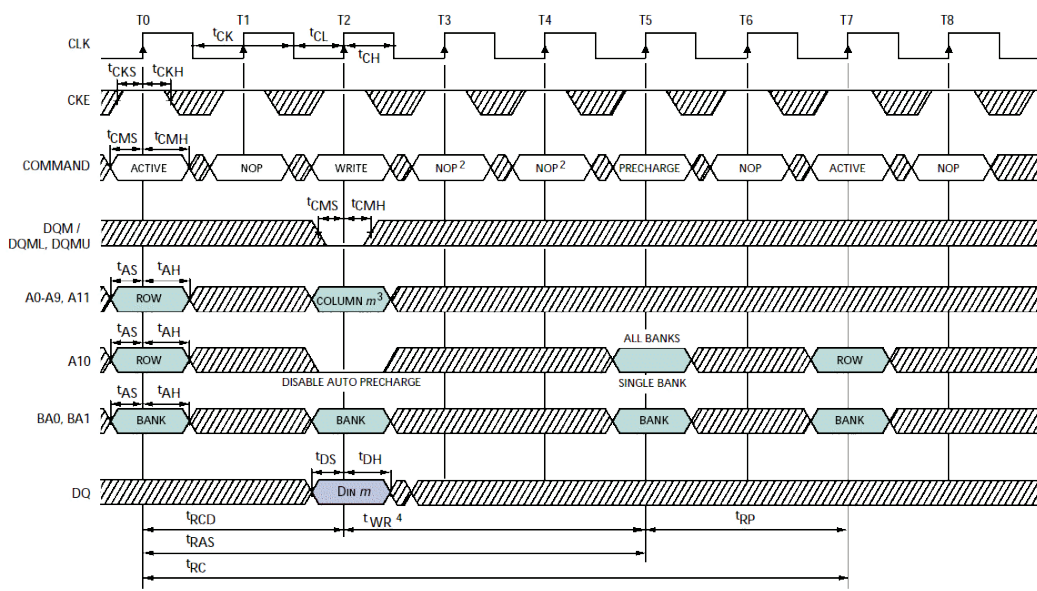
SDRAM Architecture



SDRAM Read

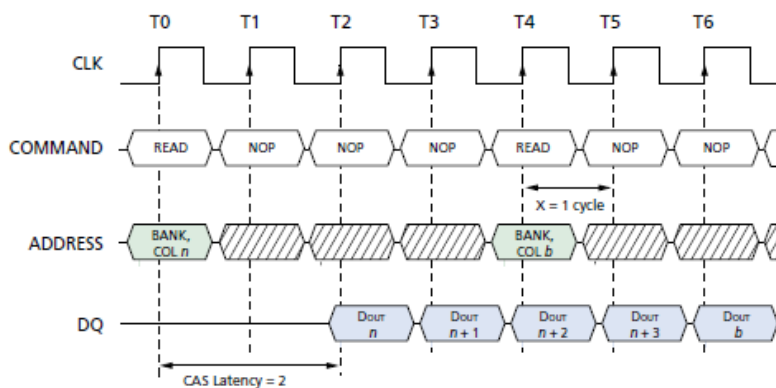


SDRAM Write



Consecutive read bursts

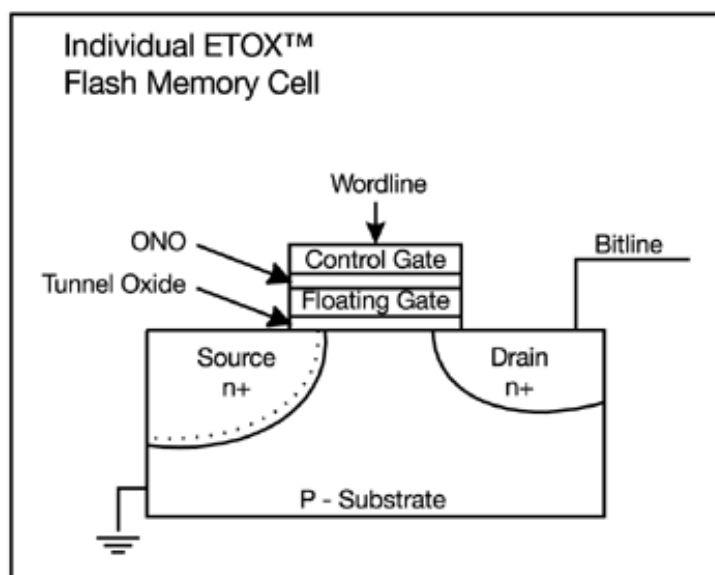
- Mode register must be programmed



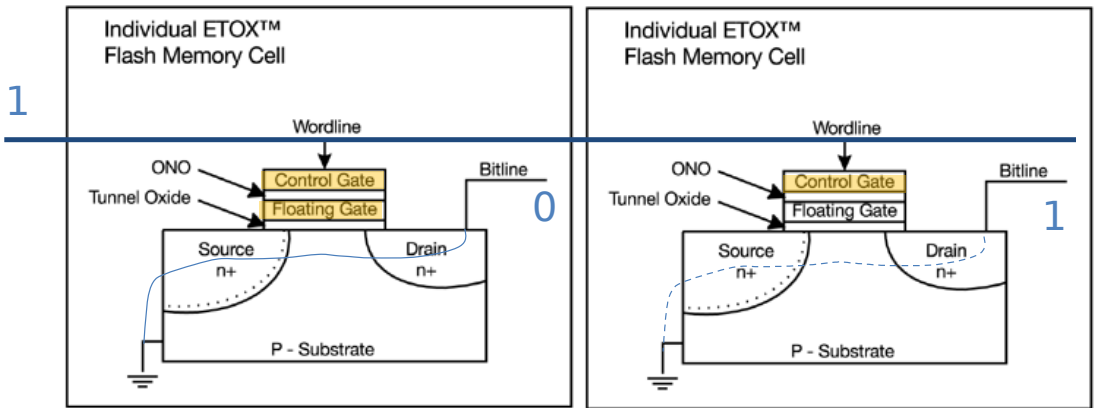
FLASH - Interface

- Looks like SRAM
 - Read
 - Write commands
- Erase is done in blocks
- Contains uCLinux kernel + file system

FLASH - Cell

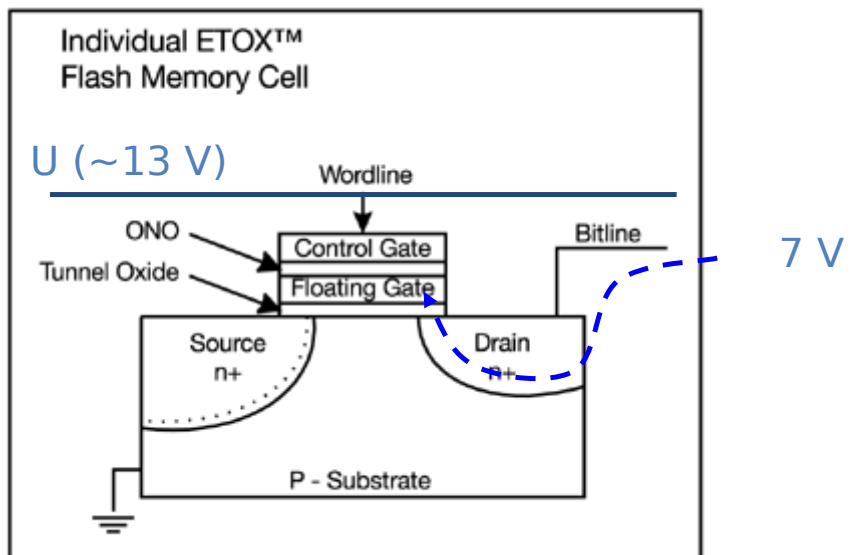


FLASH - Read (NOR)



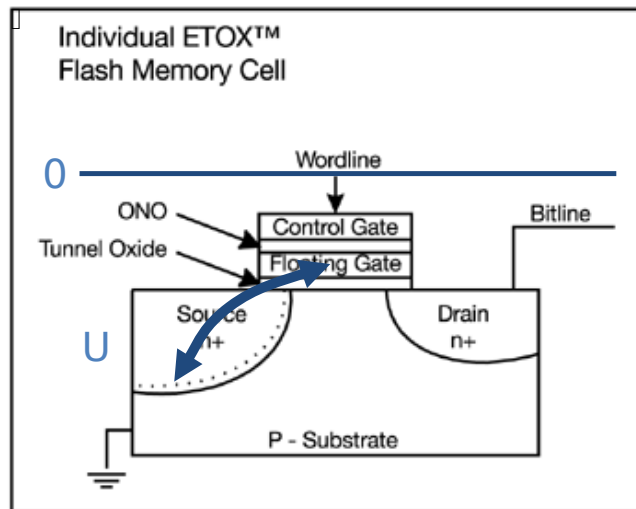
1

FLASH - Program (to 0)

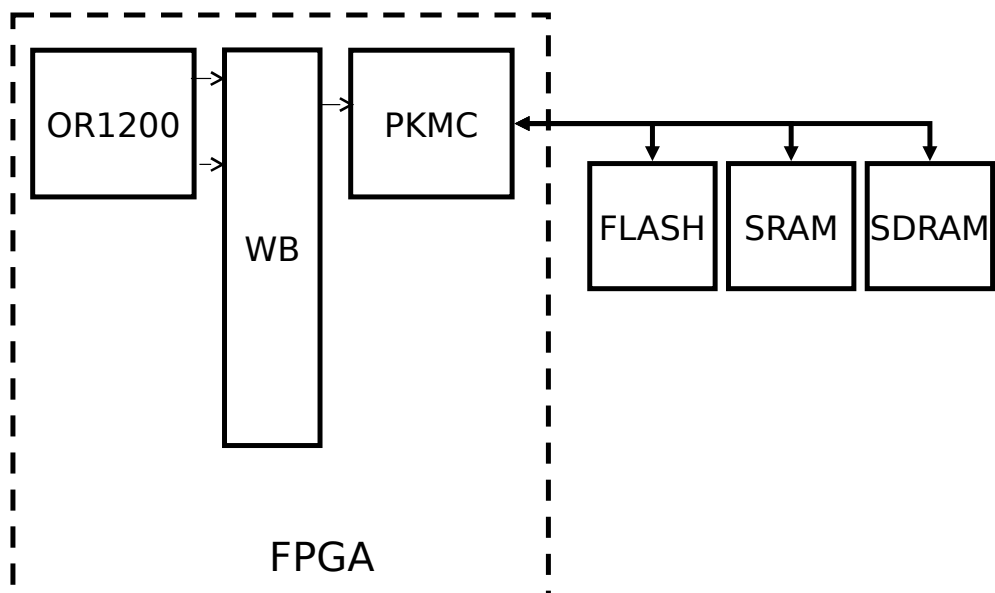


FLASH - Erase (to 1)

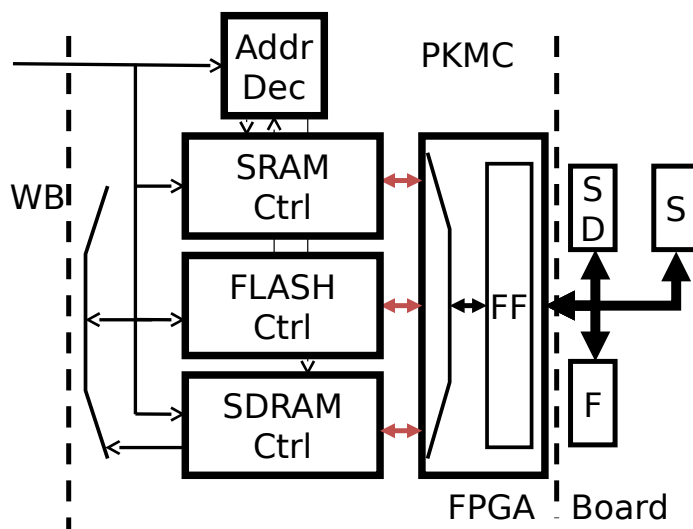
- Only blockwise



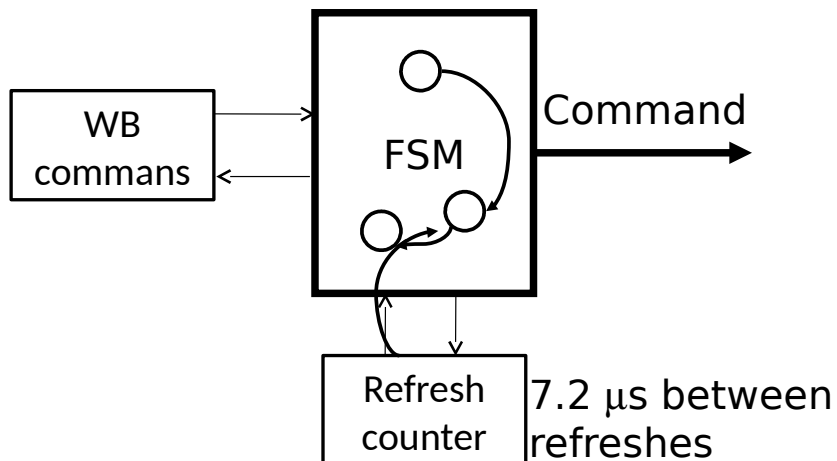
System Overview



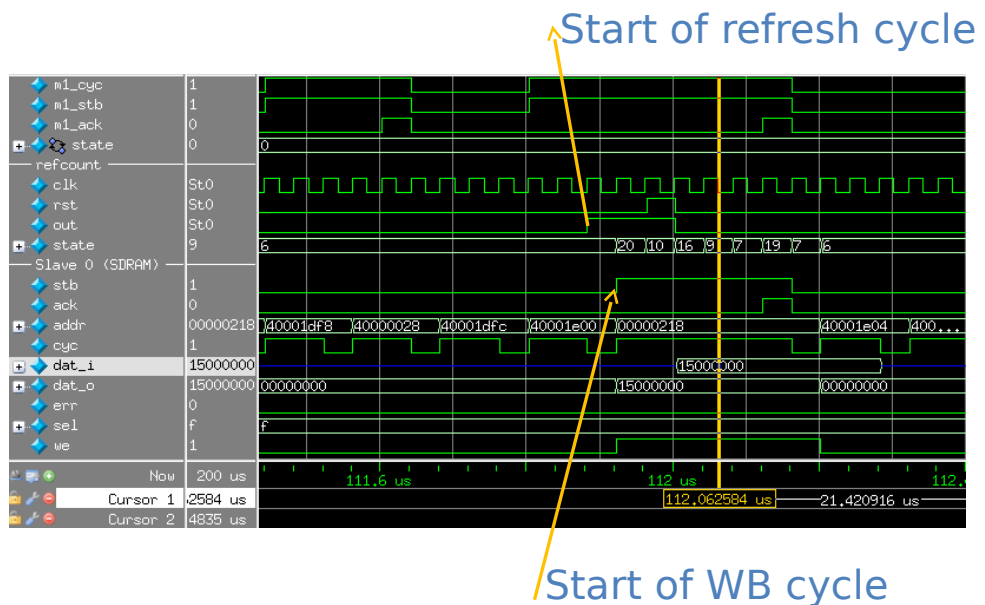
PKMC internals



SDRAM Controller Internals



Refresh cycle



Lab 4, Custom instruction

- Increase performance by adjusting instruction set
- Specific for application domain
 - General purpose processor is general purpose
 - Not exceptionally good at anything
- Use profiling to find out the most timeconsuming part of the application code

Huffman Encoding/Decoding

1) After Q

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

2) After zig-zag

```

22
12
04
00 -12
-8
0000000000000000
0000000000000000
0000000000000001

```

3) After RLE

Value	raw bits (amplitude value)	
05	10110	
04	1100	-12 =>
13	100	12-1, force MSB=0
24	0011	=> 0011
04	0111	
F0		-8 =>
F0		8-1, force MSB=0
D1	1	=> 0111
00		

Run of 0:s Magnitude

4) Huffman coding

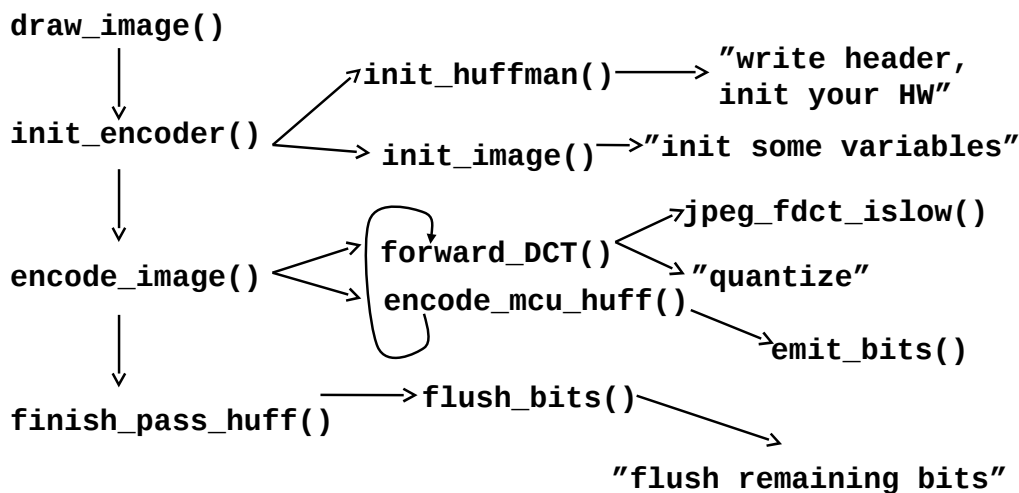
- Value are HC (variable length)
- using table lookup
- raw bits are left untouched

Huffman in JFIF

- Output: 1 - 16 bits
- Encodes bytes
- 2 tables used
 - Y DC
 - Y AC

jpegfiles

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



Emit_bits()

```

/* Only the right 24 bits of put_buffer are used; the valid bits are left-justified in
 * this part. At most 16 bits can be passed to emit_bits in one call, and we never retain
 * more than 7 bits in put_buffer between calls, so 24 bits are sufficient.
 */
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 */
}

```


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

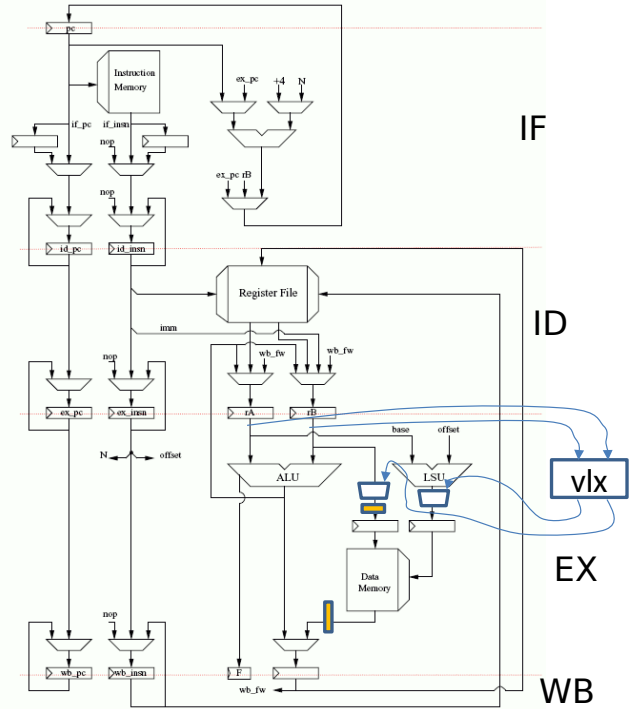
Hardware Modifications

- 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
 ↓ ↓
 1.sd (rA), rB

 = align

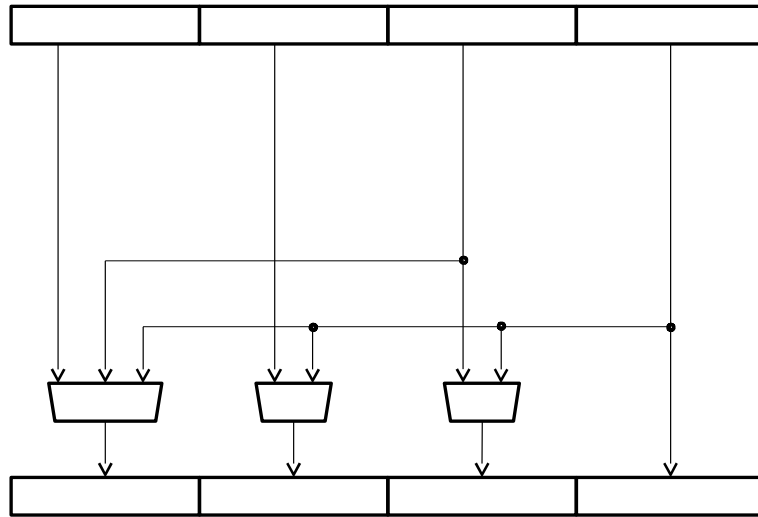


Or1200 Pipeline

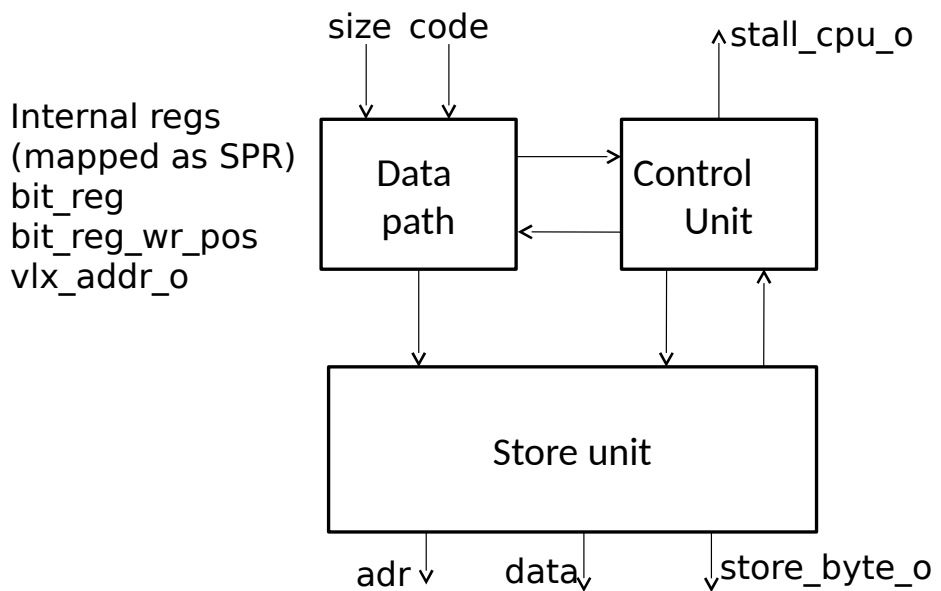
- Remember stall

	1	2	3	4	5	6	7
IF	ld	add	sub	-			
ID/RR		ld	add	-	sub		
EX/M			ld	ld	add	sub	
W				-	ld	add	sub

Align reg2mem



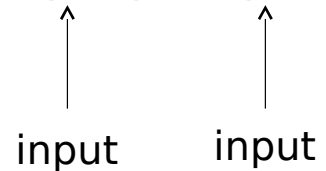
Proposed Architecture



Inline asm

In jpegfiles insert: ↙ template

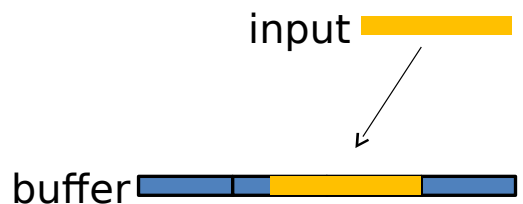
```
asm volatile("l.sd 0x0(%0),%1" : : "r"(code), "r"(size));
```



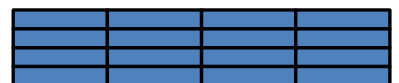
=> code and size will show up at your vlx

Data Path

- Fill buffer alternatives
 - One bit/clock cycle
 - All bits at once
- Write to mem alternatives
 - One byte
 - One 32 bit word, must be on word boundaries



mem



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

output
↓

```
00000250 <_sb_get_buff_pos>:
250: 9c 21 ff fc    l.addi r1,r1,0xffffffffc
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
```

www.liu.se

li.u LINKÖPING
UNIVERSITY