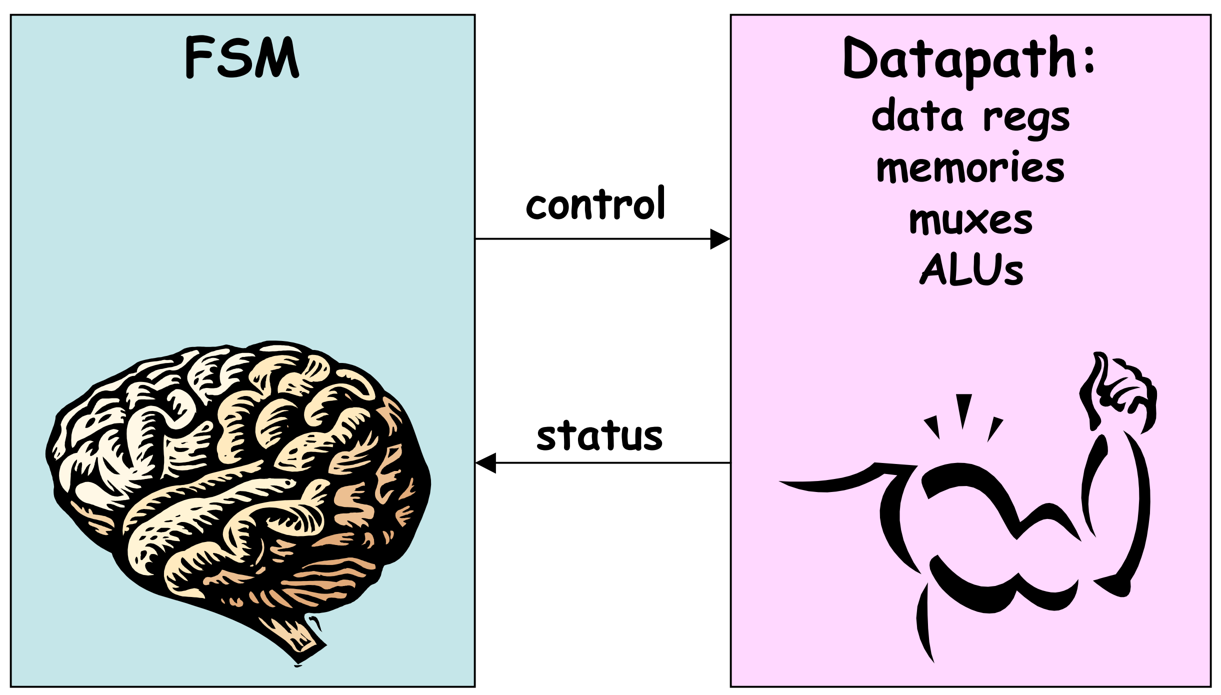
The Last Lecture!

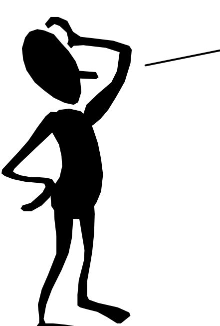
1. Schedule
2. FSM++
3. FGPAs @ Home

Schedule Reminders

- **Fri, 10/26: Lab #5 checkoff by 5pm**
- **Mon, 10/29: upload Project Abstract by 5pm**
- **Wed, 10/31: Quiz, 7:30p - 9:30p, 34-101**
- **Fri, 11/03: mandatory writing workshop, 1p, 34-101**
  complete proposal meeting with mentor
  upload Project Proposal by 5pm
- **Fri, 11/10: upload CI-M final version by 5pm**
  complete block diagram meeting w/ mentor
- **Tu, W, Th: 11/12-15: 15min design presentations**
  schedule TBA (we'll email you!)
  please upload slides to website
- **Fri, 11/16: upload Project Checklist by 5pm**
- **M, Tu, W : 12/10-12: project presentations & videotaping**
  schedule TBA (we'll email you!)
- **Wed, 12/12: upload Final Project Report by 5pm**
  (sorry, no extensions possible!)

Digital Systems = FSMs + Datapath

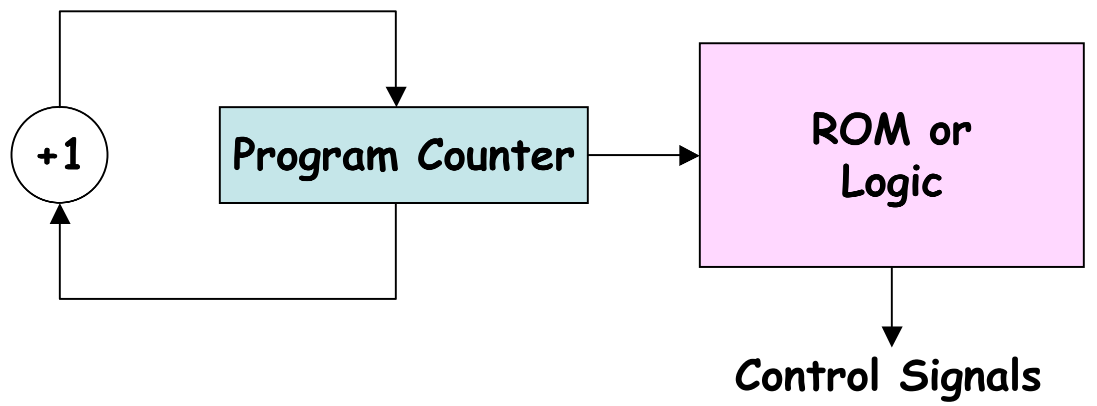




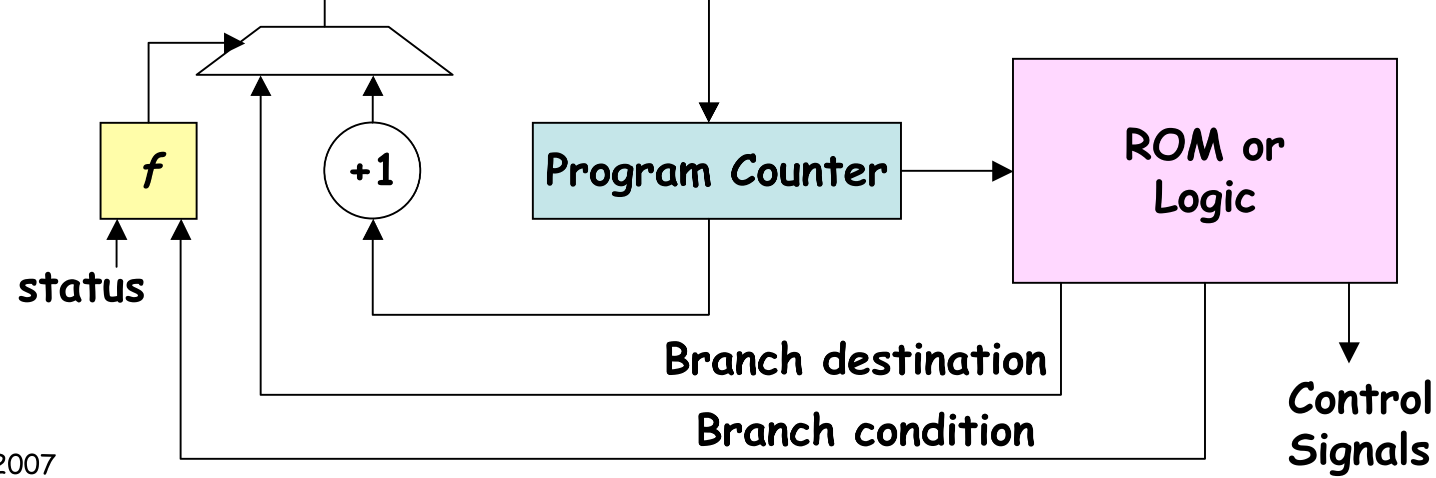
But what if my FSM has hundreds or thousands of states? That's a BIG case statement!

Microsequencers

Step 1: use a counter for the state

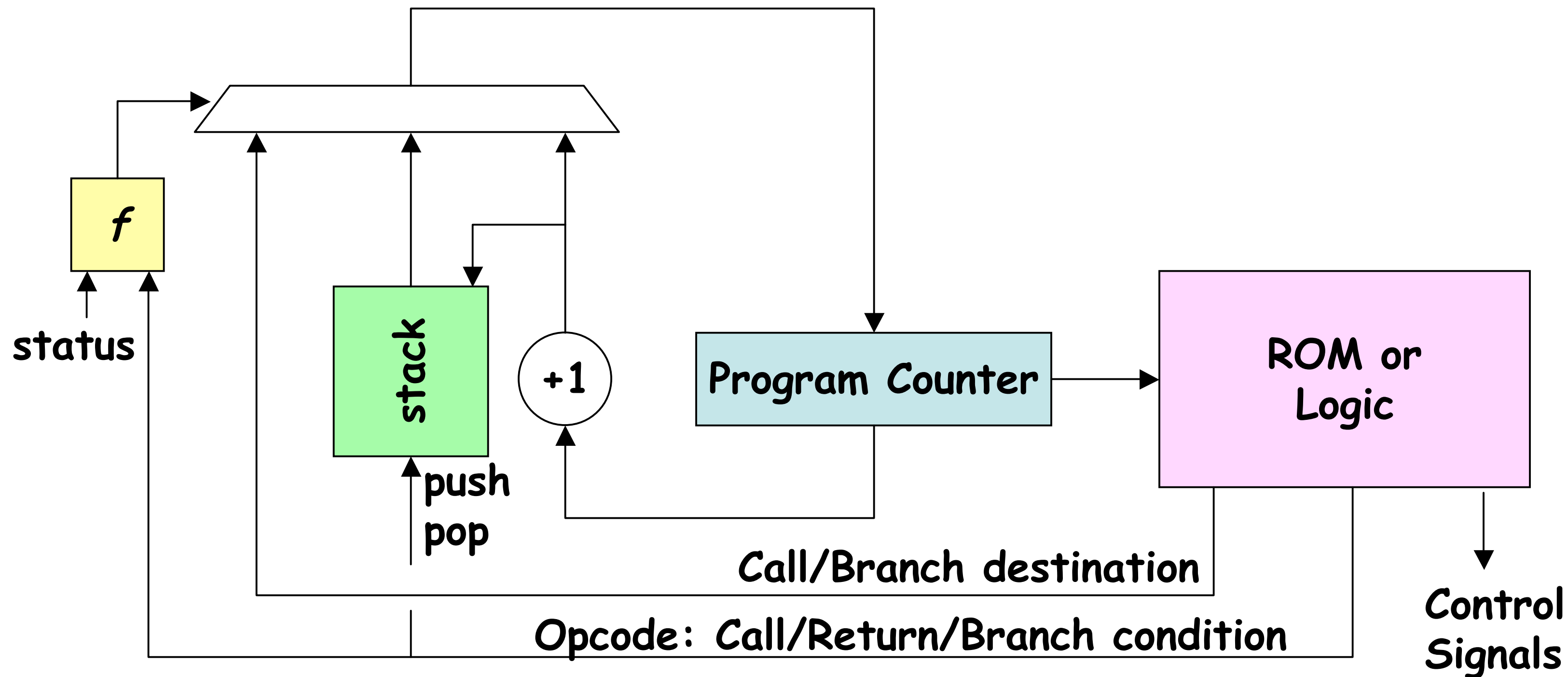


Step 2: add a conditional branch mechanism



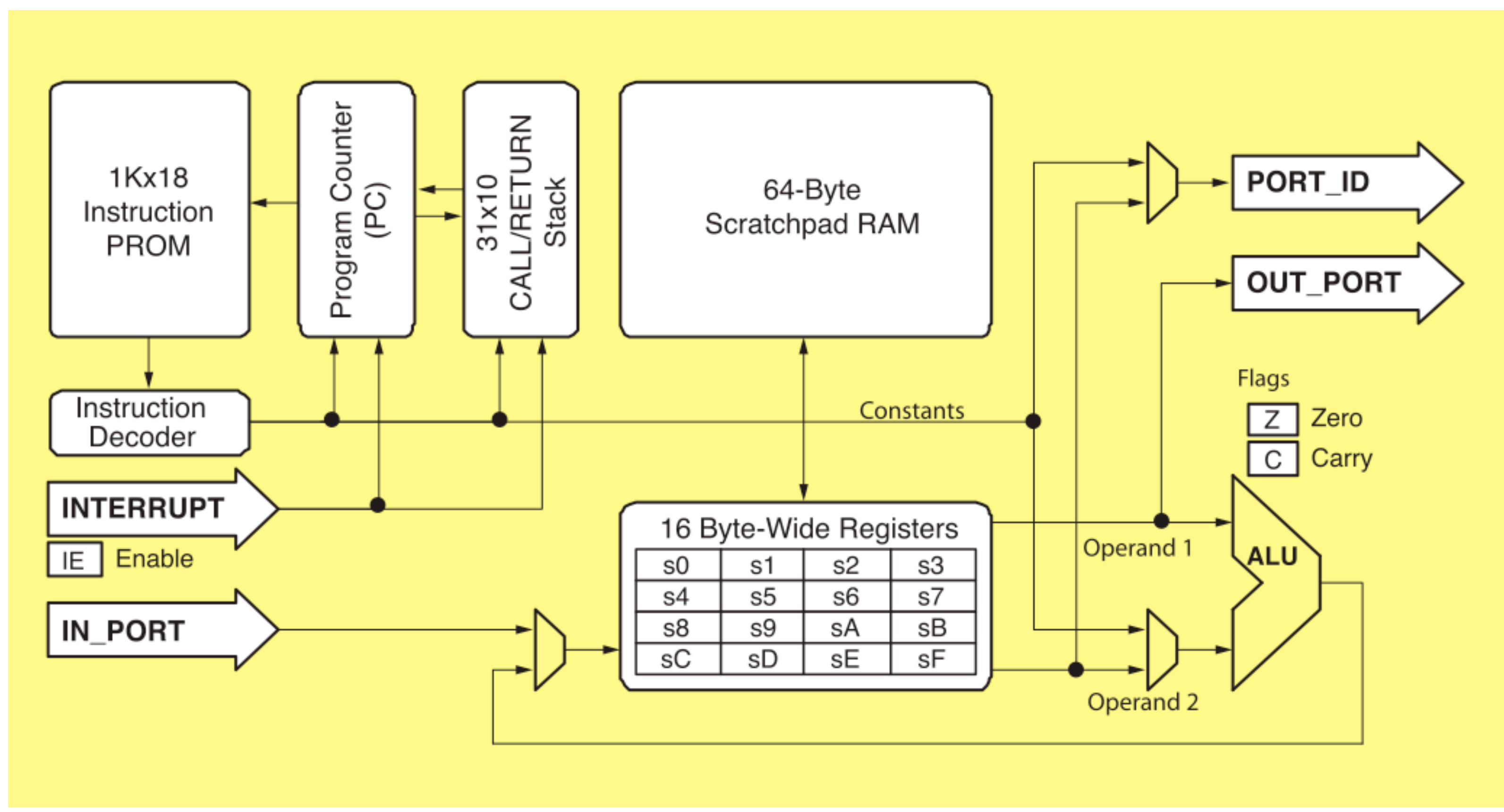
Microsequencers (cont'd.)

Step 3: add a (small) call/return stack to support "subroutines"



Subroutine call: select destination as new PC, push PC+1 onto stack
Subroutine return: select top of stack as new PC, pop stack

Xilinx PicoBlaze™

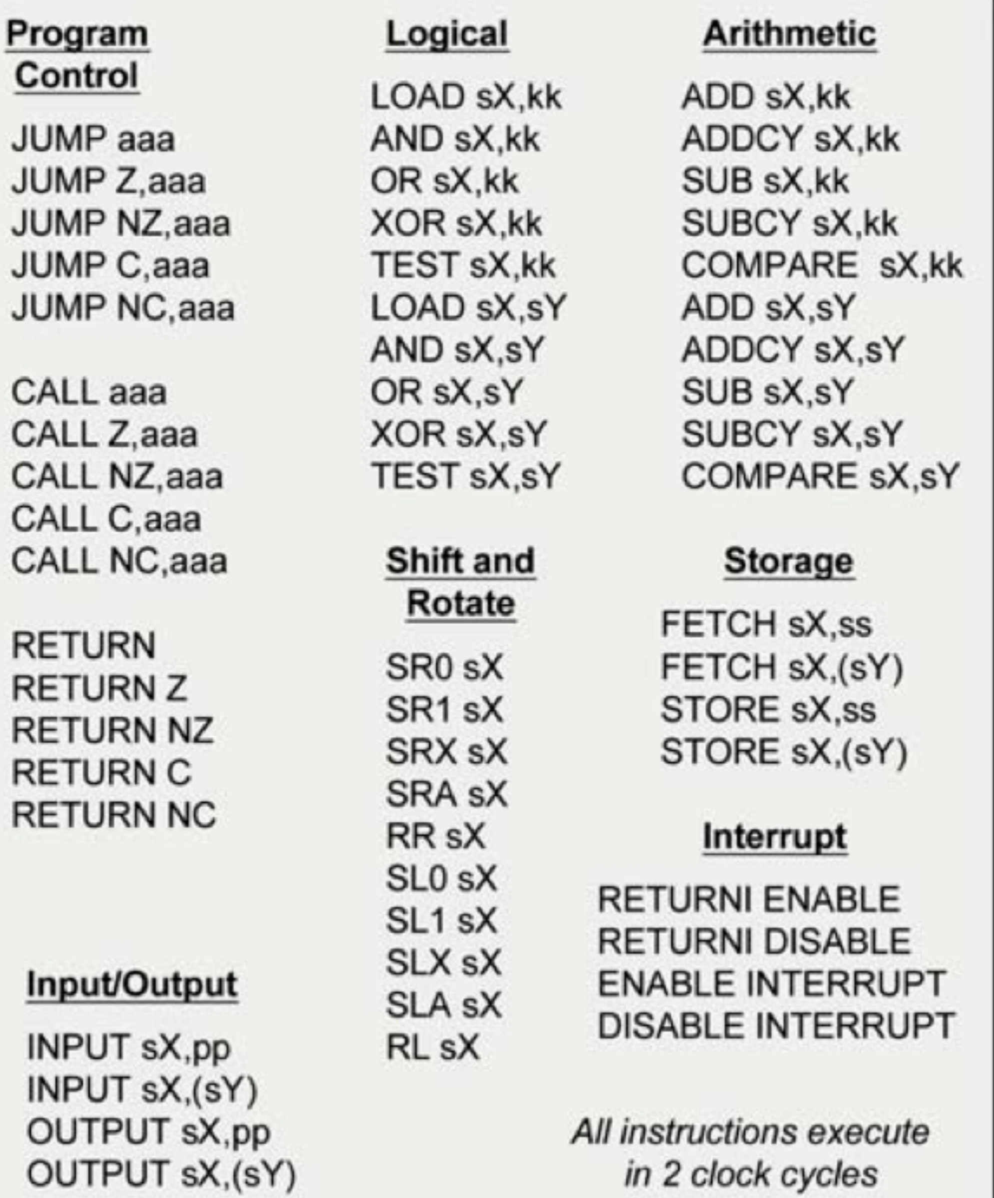


- 8-bit data path
- internal memory
  - 1K 18-bit insts
  - 31-locn stack
  - 16 8-bit registers
  - 64-locn local mem
- external 8-bit ports
  - 256 in, 256 out
- small: only 96 slices + inst mem
- fast: 2 cycles/inst [IF→EXE]
  200MHz (100MIPS) on labkit

PicoBlaze Instructions

Conditional and unconditional flow of control

Access to external devices



Program Control

JUMP aaa
JUMP Z,aaa
JUMP NZ,aaa
JUMP C,aaa
JUMP NC,aaa

CALL aaa
CALL Z,aaa
CALL NZ,aaa
CALL C,aaa
CALL NC,aaa

RETURN
RETURN Z
RETURN NZ
RETURN C
RETURN NC

Input/Output

INPUT sX,pp
INPUT sX,(sY)
OUTPUT sX,pp
OUTPUT sX,(sY)

Logical

LOAD sX,kk
AND sX,kk
OR sX,kk
XOR sX,kk
TEST sX,kk
LOAD sX,sY
AND sX,sY
OR sX,sY
XOR sX,sY
TEST sX,sY

Shift and Rotate

SR0 sX
SR1 sX
SRX sX
SRA sX
RR sX
SL0 sX
SL1 sX
SLX sX
SLA sX
RL sX

Arithmetic

ADD sX,kk
ADDCY sX,kk
SUB sX,kk
SUBCY sX,kk
COMPARE sX,kk
ADD sX,sY
ADDCY sX,sY
SUB sX,sY
SUBCY sX,sY
COMPARE sX,sY

Storage

FETCH sX,ss
FETCH sX,(sY)
STORE sX,ss
STORE sX,(sY)

Interrupt

RETURNI ENABLE
RETURNI DISABLE
ENABLE INTERRUPT
DISABLE INTERRUPT

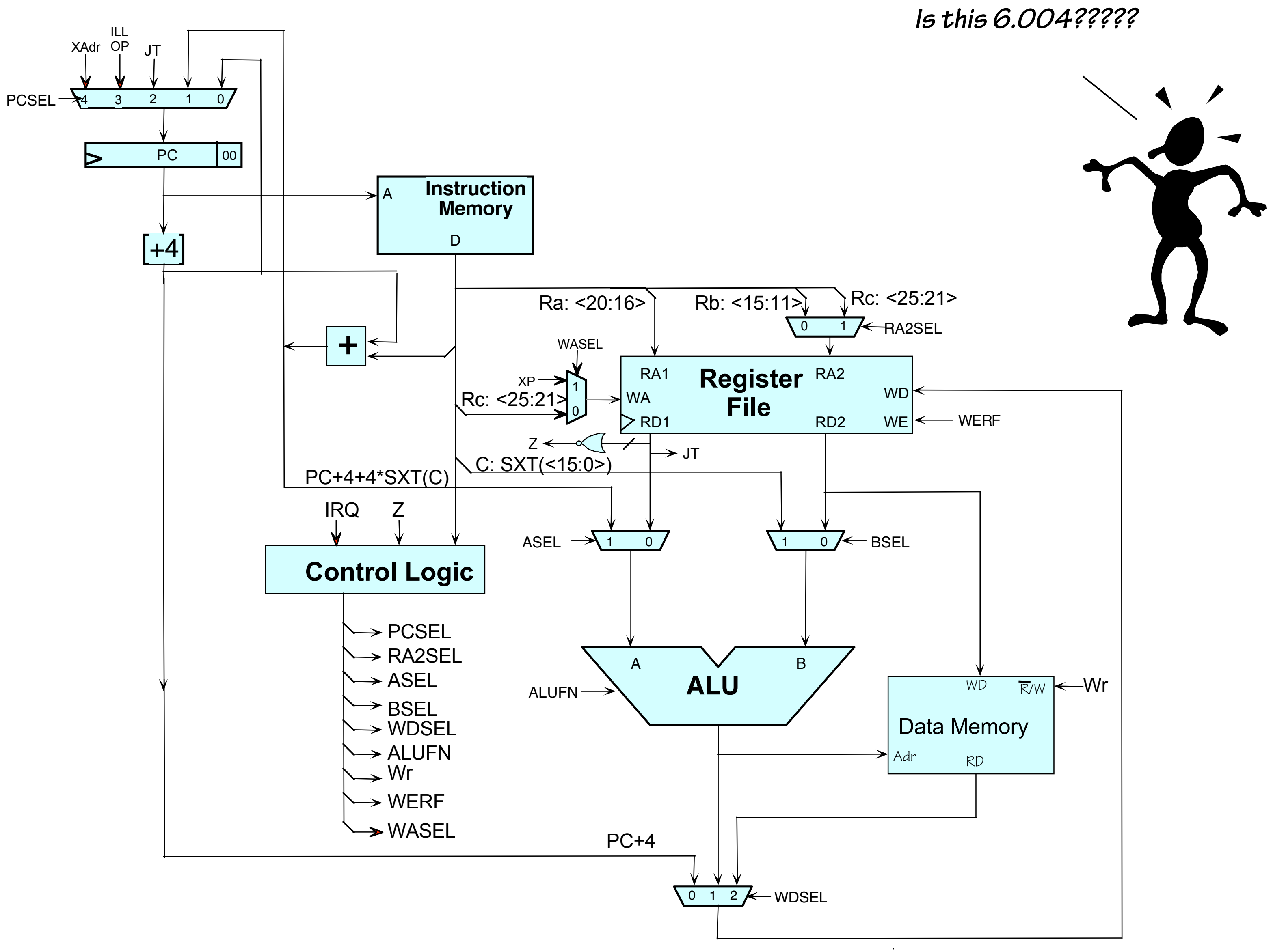
All instructions execute in 2 clock cycles

8-bit ALU:
<r1> <op>= <r2>
<r1> <op>= const

Access to 64-locn local mem

Interrupt management

A “Real” Processor: the Beta!



Beta Instructions - I

31 – 26	25 – 21	20 – 16	15 – 11	10 – 0
10xxxx	Rc	Ra	Rb	*unused*

OP(Ra,Rb,Rc): Reg[Rc] ← Reg[Ra] op Reg[Rb]

Opcodes:
ADD (plus), SUB (minus), **MUL (multiply)**, **DIV (**divided by),
AND (bitwise and), **OR (**bitwise or), **XOR (**bitwise exclusive or)
CMPEQ (equal), CMPLT (less than), **CMPLE (**less than or equal) [result = 1 if true, 0 if false]
SHL (left shift), **SHR (**right shift w/o sign extension), **SRA (**right shift w/ sign extension)

31 – 26	25 – 21	20 – 16	15 – 0
11xxxx	Rc	Ra	literal (two's complement)

OPC(Ra,literal,Rc): Reg[Rc] ← Reg[Ra] op SEXT(literal)

Opcodes:
ADDC (plus), SUBC (minus), **MULC (multiply)**, **DIVC (**divided by)
ANDC (bitwise and), **ORC (**bitwise or), **XORC (**bitwise exclusive or)
CMPEQC (equal), CMPLTC (less than), **CMPLEC (**less than or equal) [result = 1 if true, 0 if false]
SHLC (left shift), **SHRC (**right shift w/o sign extension), **SRAC (**right shift w/ sign extension)

Beta Instructions - II

31–26	25–21	20–16	15–0
01xxxx	Rc	Ra	literal (two's complement)

LD(Ra,literal,Rc): Reg[Rc] ← Mem[Reg[Ra] + SEXT(literal)]

ST(Rc,literal,Ra): Mem[Reg[Ra] + SEXT(literal)] ← Reg[Rc]

JMP(Ra,Rc): Reg[Rc] ← PC + 4; PC ← Reg[Ra]

BEQ/BF(Ra,label,Rc): Reg[Rc] ← PC + 4;
if Reg[Ra] = 0 then PC ← PC + 4 + 4*SEXT(literal)

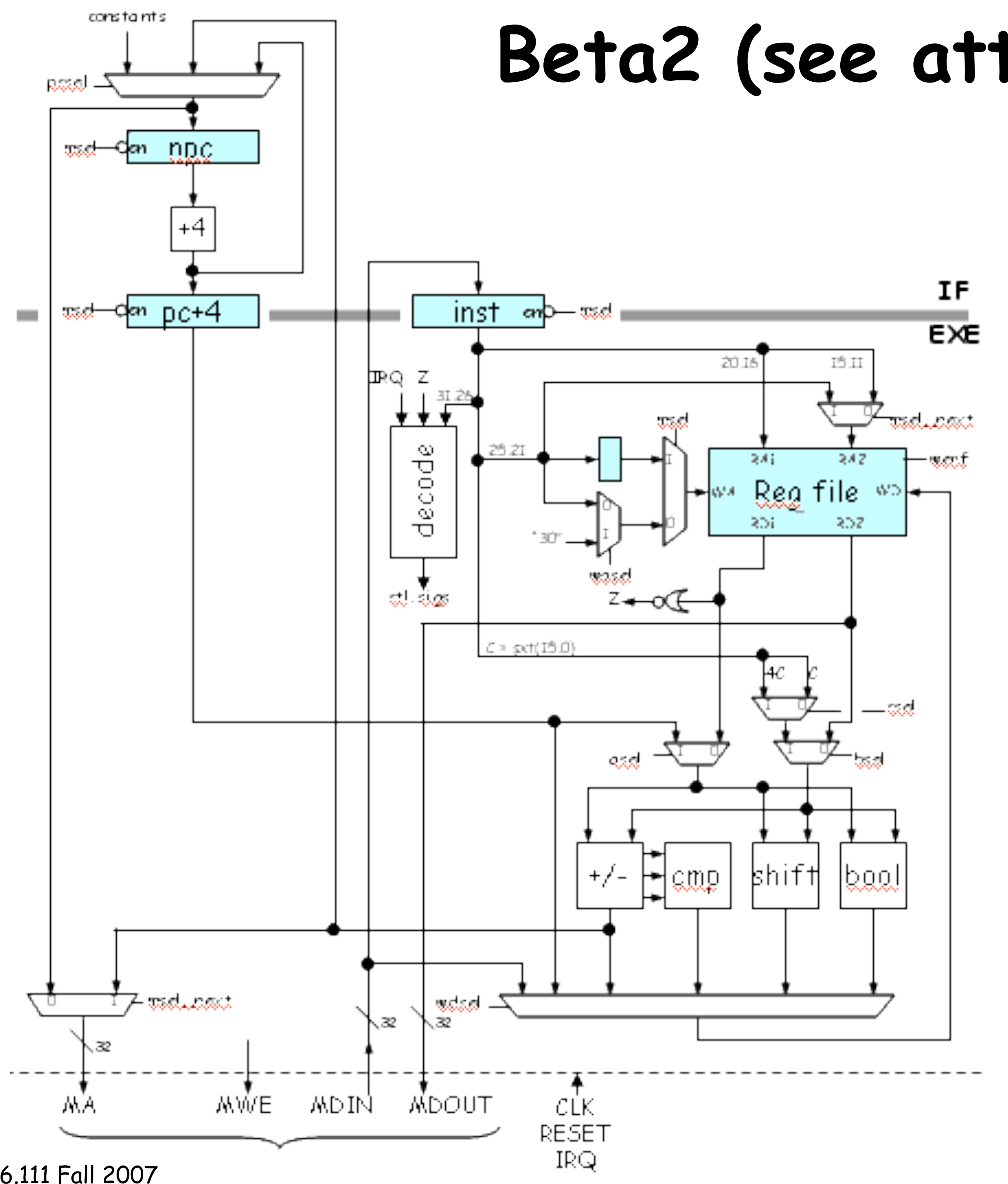
BNE/BT(Ra,label,Rc): Reg[Rc] ← PC + 4;
if Reg[Ra] ≠ 0 then PC ← PC + 4 + 4*SEXT(literal)

LDR(label,Rc): Reg[Rc] ← Mem[PC + 4 + 4*SEXT(literal)]

Beta Control Logic

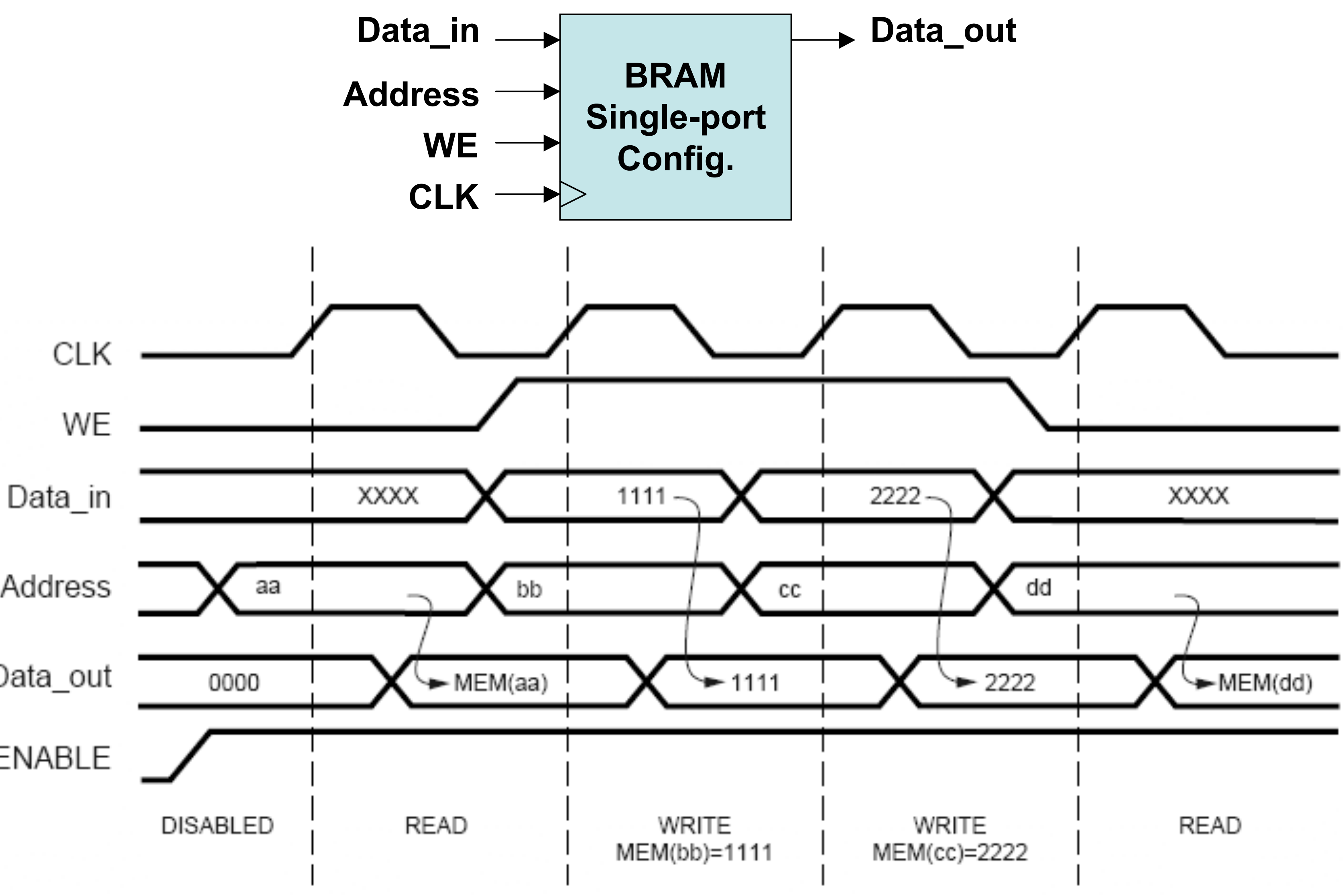
	OP	OPC	LD	ST	JMP	BEQ	BNE	LDR	Illop	trap
ALUFN	F(op)	F(op)	"+"	"+"	—	—	—	"A"	—	—
WERF	1	1	1	0	1	1	1	1	1	1
BSEL	0	1	1	1	—	—	—	—	—	—
WDSEL	1	1	2	—	0	0	0	2	0	0
WR	0	0	0	1	0	0	0	0	0	0
RA2SEL	0	—	—	1	—	—	—	—	—	—
PCSEL	0	0	0	0	2	Z ? 1 : 0	Z ? 0 : 1	0	3	4
ASEL	0	0	0	0	—	—	—	1	—	—
WASEL	0	0	0	—	0	0	0	0	1	1

Beta2 (see attached sheet)



- 2-stage pipeline, 1 annuled branch delay slot
- Memory ops (LD, LDR, ST) take two cycle in EXE stage: addr computed in 1st cycle, memory access made in 2nd
- Branch and LDR address arithmetic performed in ALU
- JMP routed thru ALU
- Single memory port shared by inst. fetch and memory access

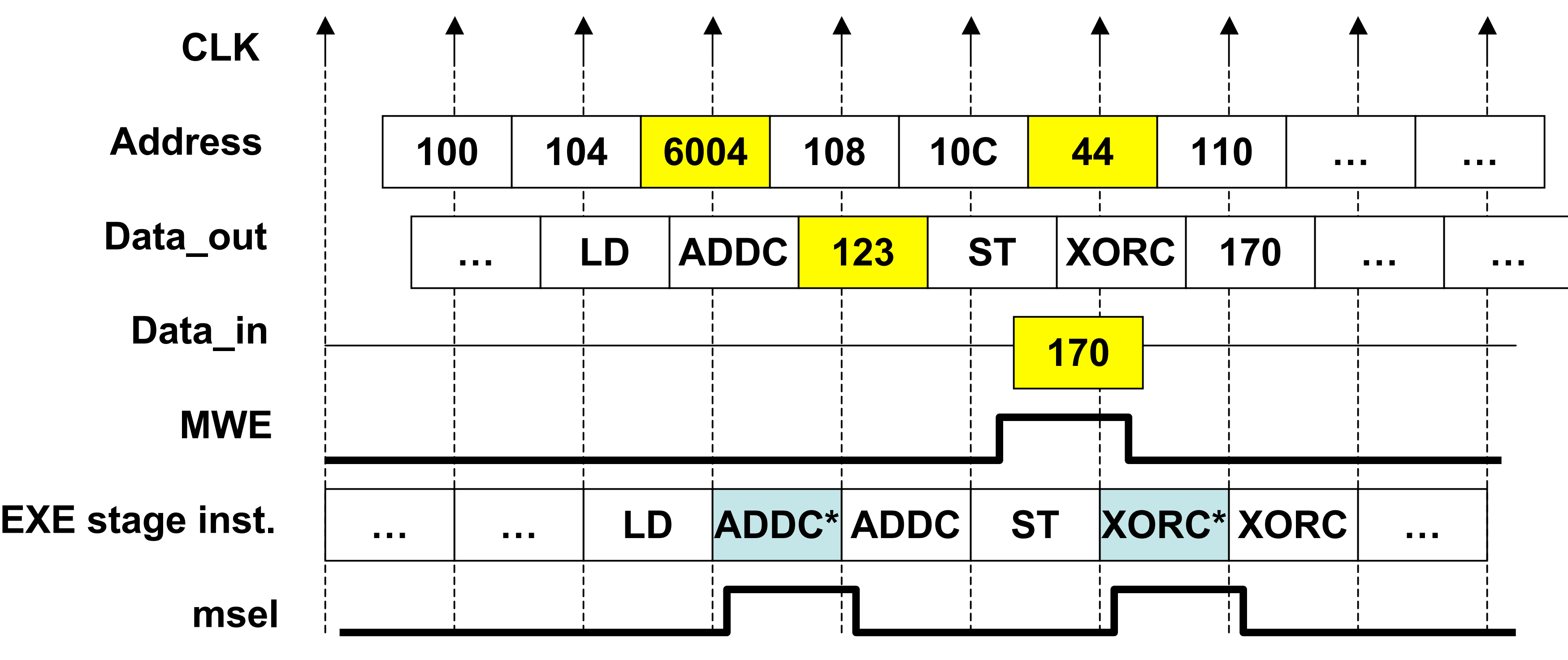
Xilinx Synchronous Block Memory



Source: Xilinx App Note 463

Instruction Pipeline Diagram

```
100: LD(R31,6004,R2)
104: ADDC(R2,47,R2)
108: ST(R2,44,R31)
10C: XORC(R2,-1,R2)
110: …
…
6004: 123
```



* Stalled in pipeline

beta2.v

```
module beta2(clk,reset,irq,ma,mdin,mdout,mwe);
  input clk,reset,irq;
  output [31:0] ma,mdout;
  input [31:0] mdin;
  output mwe;

  // beta2 registers
  reg [31:0] regfile[31:0];
  reg [31:0] npc,pc_inc,inst;
  reg [4:0] rc_save;  // needed for second cycle on LD,LDR

  // internal buses
  wire [31:0] rd1,rd2,wd,a,b,xb,c,addsub,cmp,shift,boole;

  // control signals
  wire wasel,werf,z,asel,bsel,csel;
  wire addsub_op,cmp_lt,cmp_eq,shift_op
  wire shift_sxt,boole_and,boole_or;
  …
endmodule
```

PC Logic

```
// pc
wire [31:0] npc_inc,npc_next;

assign npc_inc = npc + 4;
assign npc_next = reset ? 32'h80000000 :
                  msel ? npc :
                  branch ? {npc[31]&addsub[31],
                            addsub[30:2],2'b00} :
                  trap ? 32'h80000004 :
                  interrupt ? 32'h80000008 :
                  npc_inc;

always @ (posedge clk) begin
  npc <= npc_next;   // stall on msel handled above
  if (!msel) pc_inc <= npc_inc;
end
```

Instruction Register & Decode

```
// instruction reg
always @ (posedge clk) if (!msel) inst <= mdin;

// control logic
decode ctl(.clk(clk),.reset(reset),.irq(irq & !npc[31]),
           .z(z),.opcode(inst[31:26]),
           .asel(asel),.bsel(bsel),.csel(csel),
           .wasel(wasel),.werf(werf),.msel(msel),
           .msel_next(msel_next),.mwe(mwe),
           .addsub_op(addsub_op),.cmp_lt(cmp_lt),
           .cmp_eq(cmp_eq),
           .shift_op(shift_op),.shift_sxt(shift_sxt),
           .boole_and(boole_and),.boole_or(boole_or),
           .wd_addsub(wd_addsub),.wd_cmp(wd_cmp),
           .wd_shift(wd_shift),.wd_boole(wd_boole),
           .branch(branch),.trap(trap),
           .interrupt(interrupt));
```

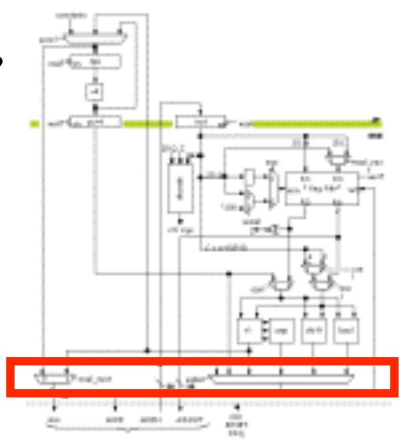
Register File

```
// register file
wire [4:0] ra1,ra2,wa;
always @ (posedge clk)
  if (!msel) rc_save <= inst[25:21];

assign ra1 = inst[20:16];
assign ra2 = msel_next ? inst[25:21] : inst[15:11];
assign wa = msel ? rc_save :
            wasel ? 5'd30 : inst[25:21];
assign rd1 = (ra1 == 31) ? 0 : regfile[ra1];
assign rd2 = (ra2 == 31) ? 0 : regfile[ra2];
always @ (posedge clk)
  if (werf) regfile[wa] <= wd;

assign z = ~| rd1;   // used in BEQ/BNE instructions
```

ALU

```
// alu
assign a = asel ? pc_inc : rd1;
assign b = bsel ? c : rd2;
assign c = csel ? {{14{inst[15]}},inst[15:0],2'b00} :
                  {{16{inst[15]}},inst[15:0]};

wire addsub_n,addsub_v,addsub_z;
assign xb = {32{addsub_op}} ^ b;
assign addsub = a + xb + addsub_op;
assign addsub_n = addsub[31];
assign addsub_v = (addsub[31] & ~a[31] & ~xb[31]) |
                  (~addsub[31] & a[31] & xb[31]);
assign addsub_z = ~| addsub;

assign cmp[31:1] = 0;
assign cmp[0] = (cmp_lt & (addsub_n ^ addsub_v)) |
                (cmp_eq & addsub_z);

wire [31:0] shift_right;
shift_right sr(shift_sxt,a,b[4:0],shift_right);
assign shift = shift_op ? shift_right : a << b[4:0];

assign boole = boole_and ? (a & b) :
               boole_or ? (a | b) : a ^ b;
```

Result Mux, Address Mux



```
// result mux, listed in order of speed (slowest first)
assign wd = msel ? mdin :
            wd_cmp ? cmp :
            wd_addsub ? addsub :
            wd_shift ? shift :
            wd_boole ? boole :
            pc_inc;

// assume synchronous external memory
assign ma = msel_next ? addsub : npc_next;
assign mdout = rd2;
```

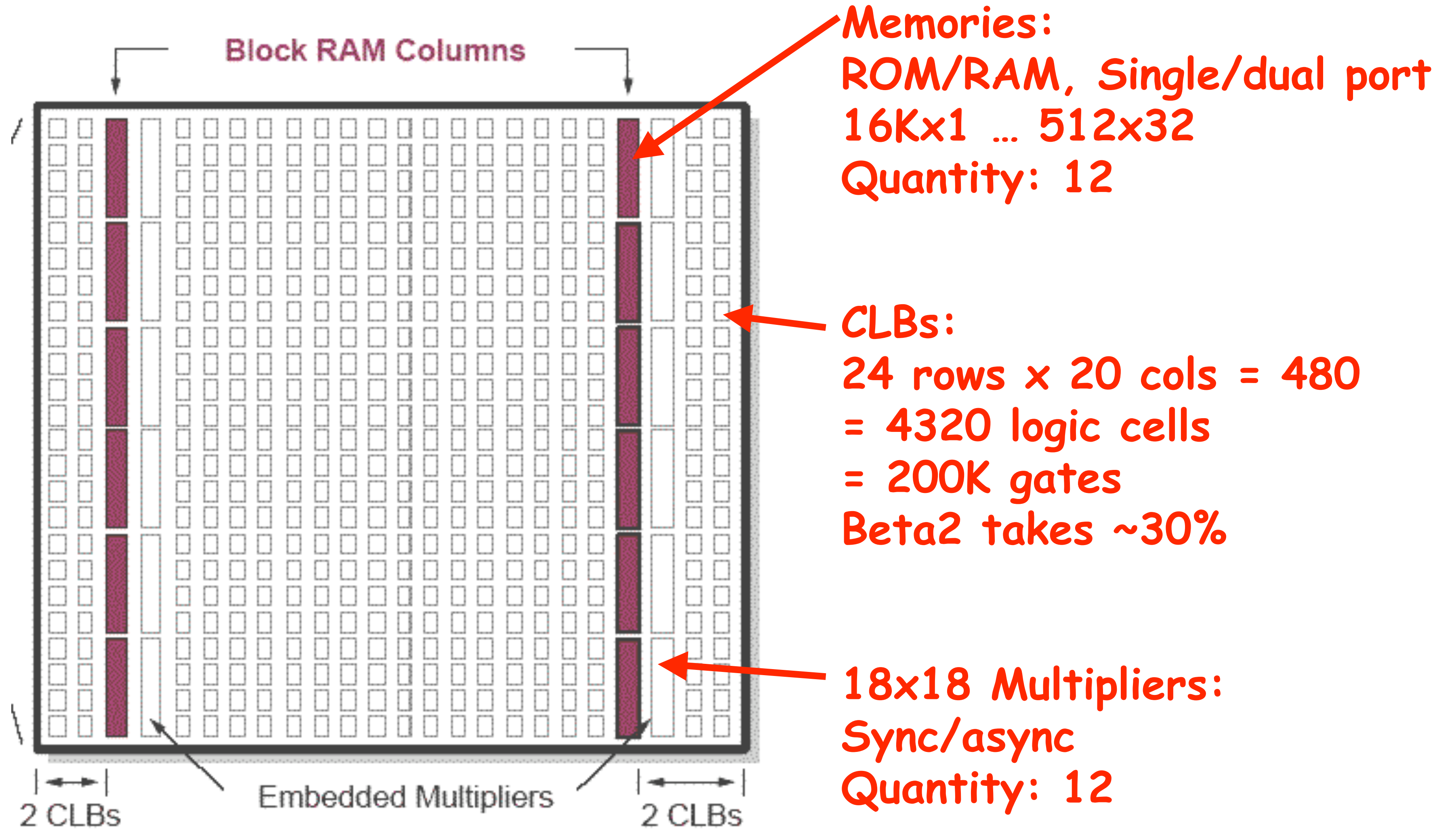
Control Logic (decode.v)

```
always @ (opcode or z or annul or irq or reset)
 begin
   // initial assignments for all control signals
   asel = 1'hx; bsel = 1'hx; csel = 1'hx;
   addsub_op = 1'hx; shift_op = 1'hx; shift_sxt = 1'hx;
   cmp_lt = 1'hx; cmp_eq = 1'hx;
   boole_and = 1'hx; boole_or = 1'hx;
   wasel = 0; mem_next = 0;
   wd_addsub = 0; wd_cmp = 0; wd_shift = 0; wd_boole = 0;
   branch = 0; trap = 0; interrupt = 0;

   if (irq && !reset && !annul) begin
     interrupt = 1;
     wasel = 1;
   end else casez (opcode)
     6'b011000: begin    // LD
                  asel = 0; bsel = 1; csel = 0;
                  addsub_op = 0;
                  mem_next = 1;
                end
     ...
```

Control Logic (cont'd.)

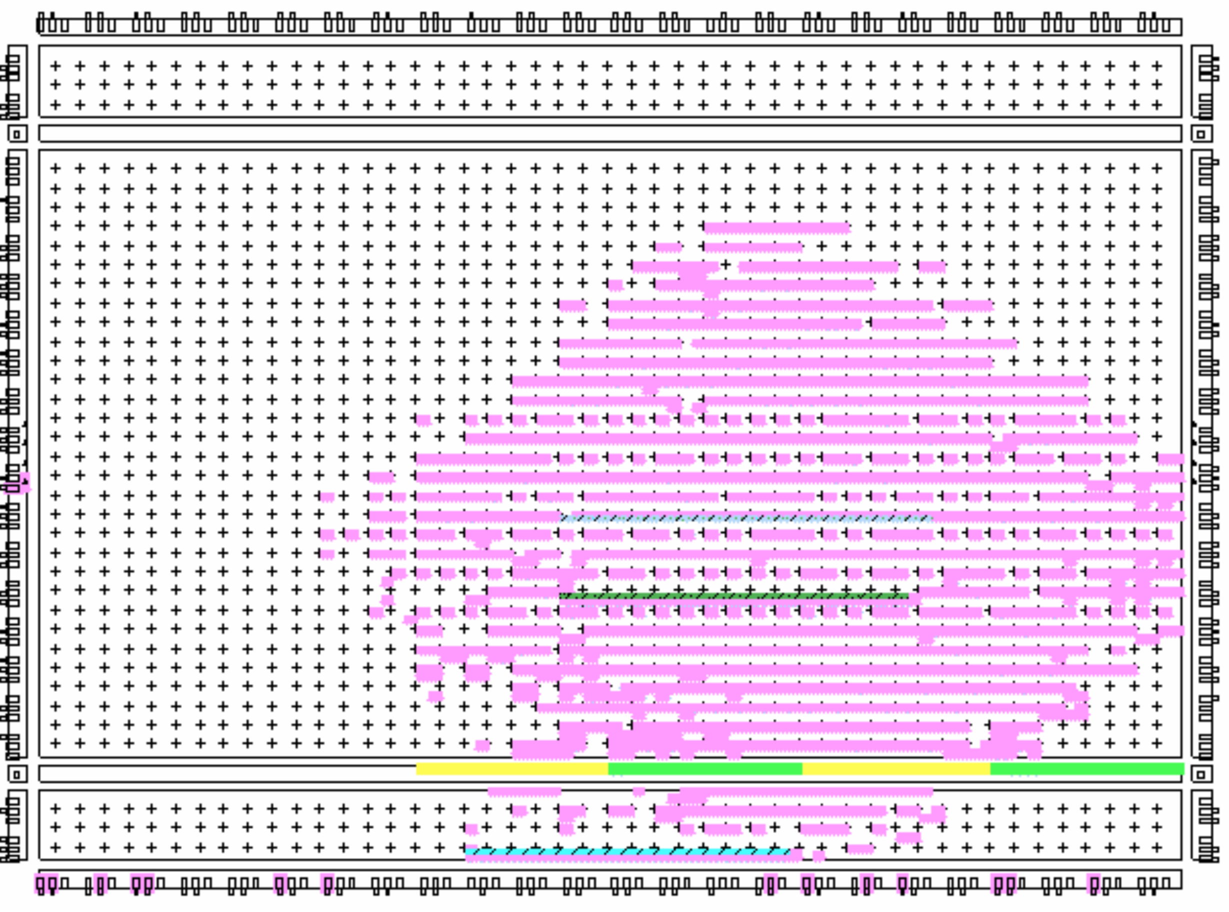
```
…
    6'b1?1100: begin   // SHL, SHLC
                 asel = 0; bsel = opcode[4]; csel = 0;
                 shift_op = 0;
                 wd_shift = 1;
               end
    6'b1?1101: begin   // SHR, SHRC
                 asel = 0; bsel = opcode[4]; csel = 0;
                 shift_op = 1; shift_sxt = 0;
                 wd_shift = 1;
               end
    6'b1?1110: begin   // SRA, SRAC
                 asel = 0; bsel = opcode[4]; csel = 0;
                 shift_op = 1; shift_sxt = 1;
                 wd_shift = 1;
               end
    default:   begin   // illegal opcode
                 trap = !annul; wasel = 1;
               end
  endcase
end  // always @ (opcode or …)
```

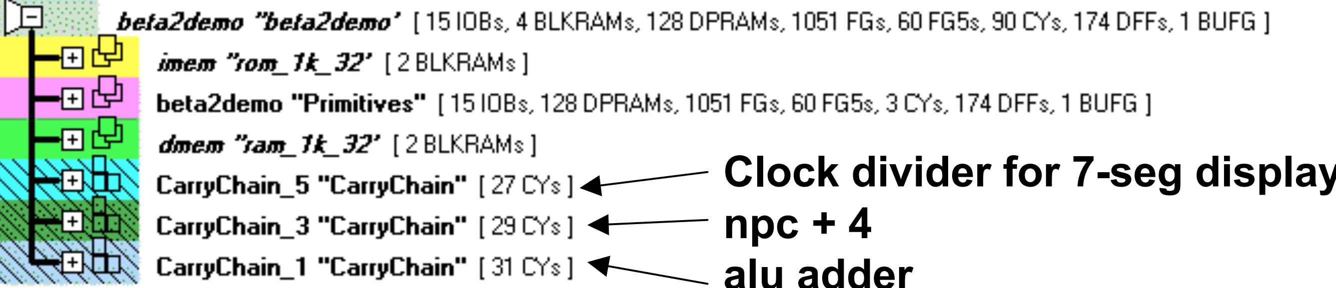
Xilinx XC3S200 FPGA



Source: Xilinx App Note 463

Beta2 Floorplan





Clock divider for 7-seg display

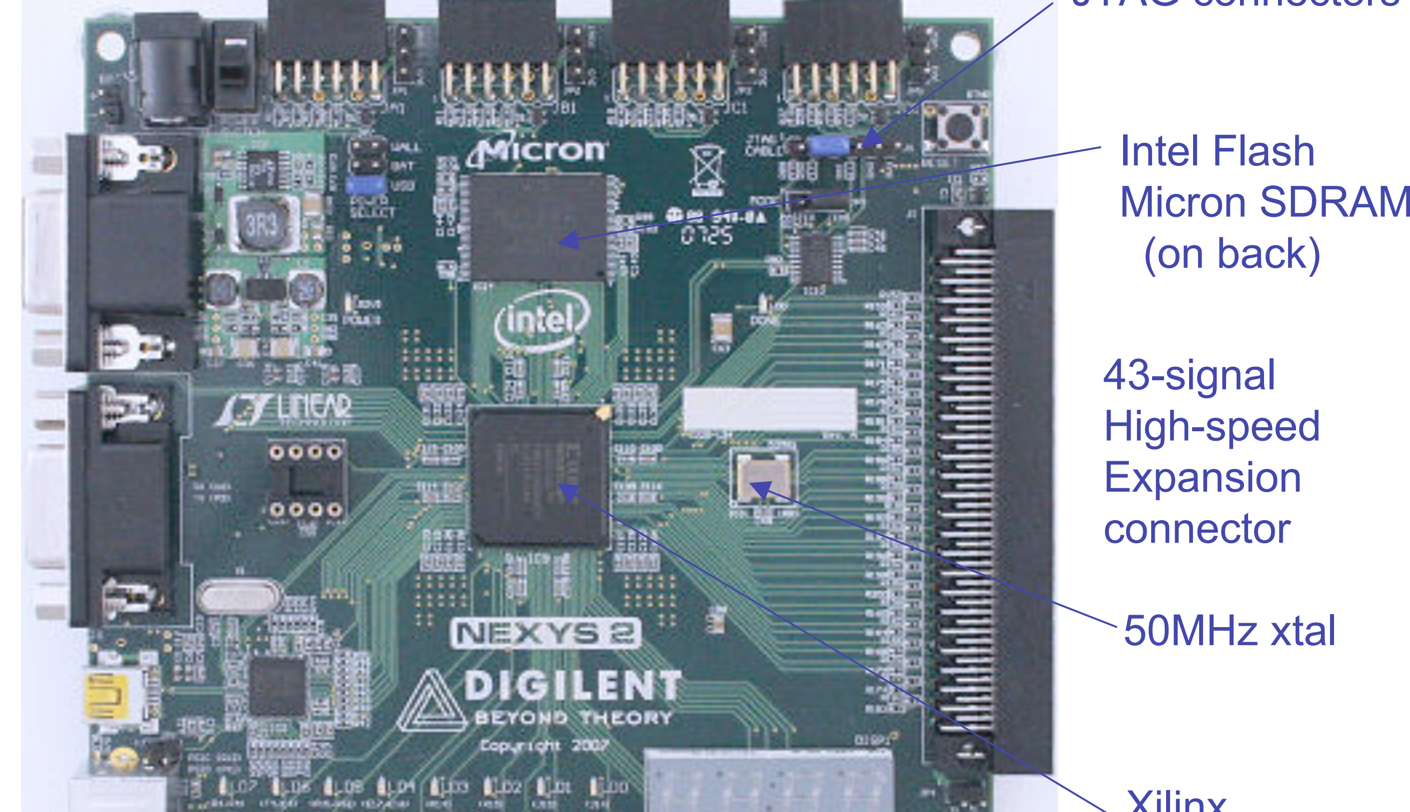
npc + 4

alu adder

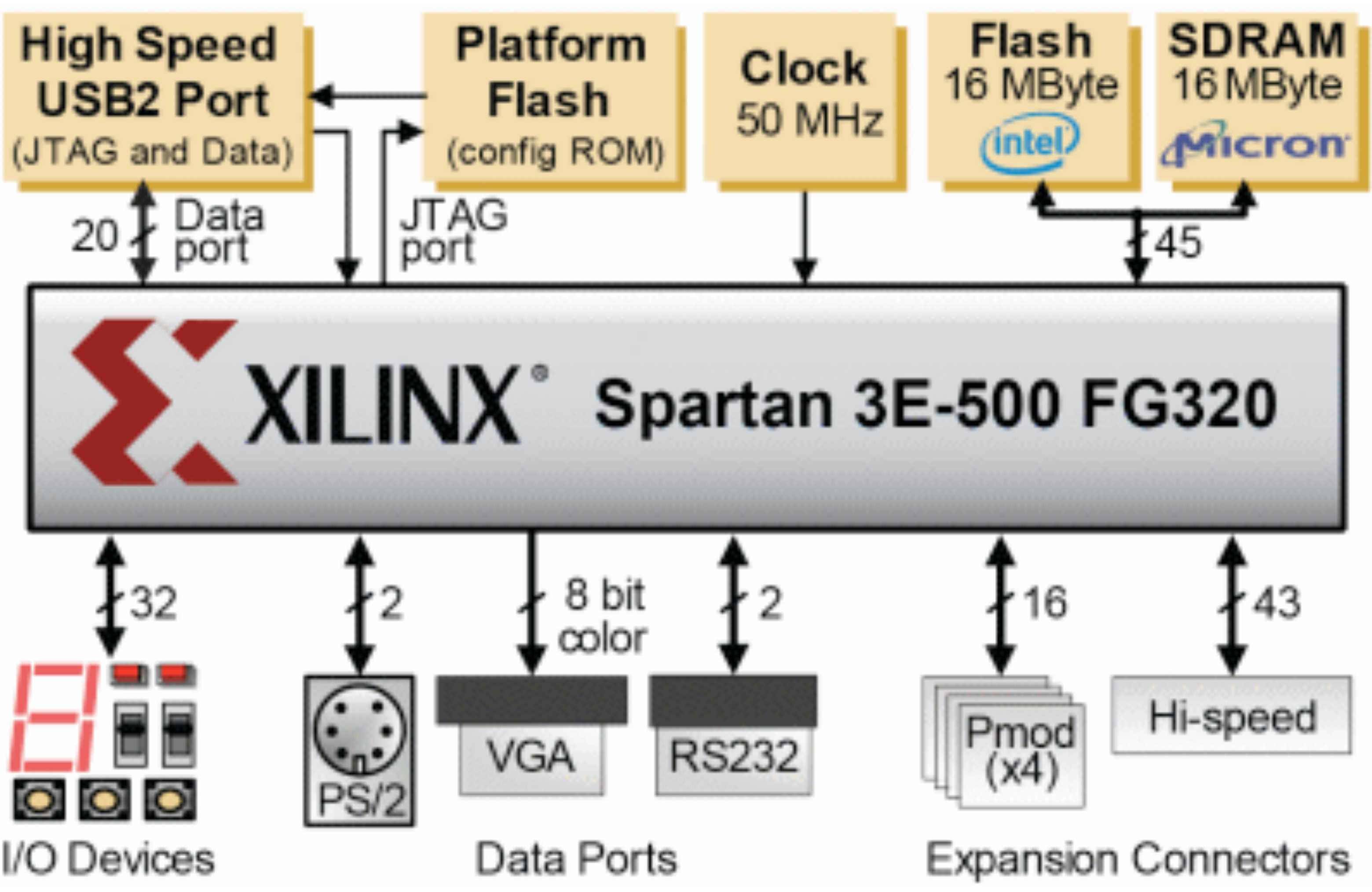
FPGAs @ Home

- **6.111 labkit: the Lexus of FPGA protoboards**
  - XC2V6000 (76032 logic cells, 2.5 Mbits BRAM)
- **Two affordable alternatives (lots more out there)**
  - Nexys2 Board (www.digilentinc.com)
    - $99 = Spartan 3E-500 (10476 logic cells, 360 Kbits BRAM)
    - Switches, buttons, leds, 4-digit display
    - 16Mbyte flash, 16Mbyte SDRAM
    - USB2 slave (power, programming, 8-bit host data stream)
    - PS2, serial port, 256-color VGA, 4 expansion connectors
  - XSA-3S1000 @ $199 (www.xess.com)
    - Spartan XC3S1000 (17480 logic cells, 432 Kbits BRAM)
    - Switches, buttons, 1-digit display
    - 32Mbyte SDRAM, 2Mbyte Flash
    - PS2, 512-color VGA
    - 80-pin expansion connector (protoboard friendly)

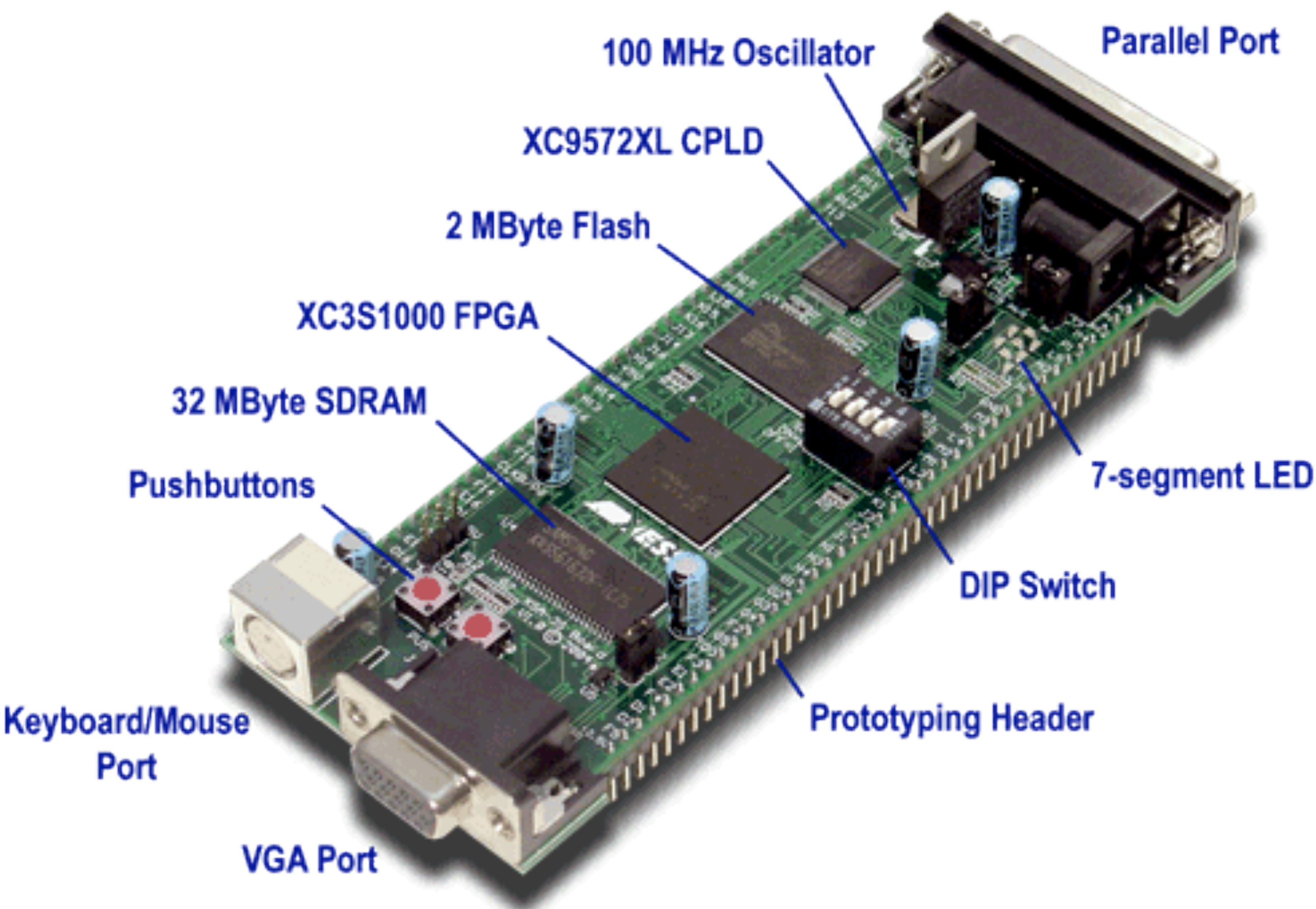
Digilent Nexys2 Board



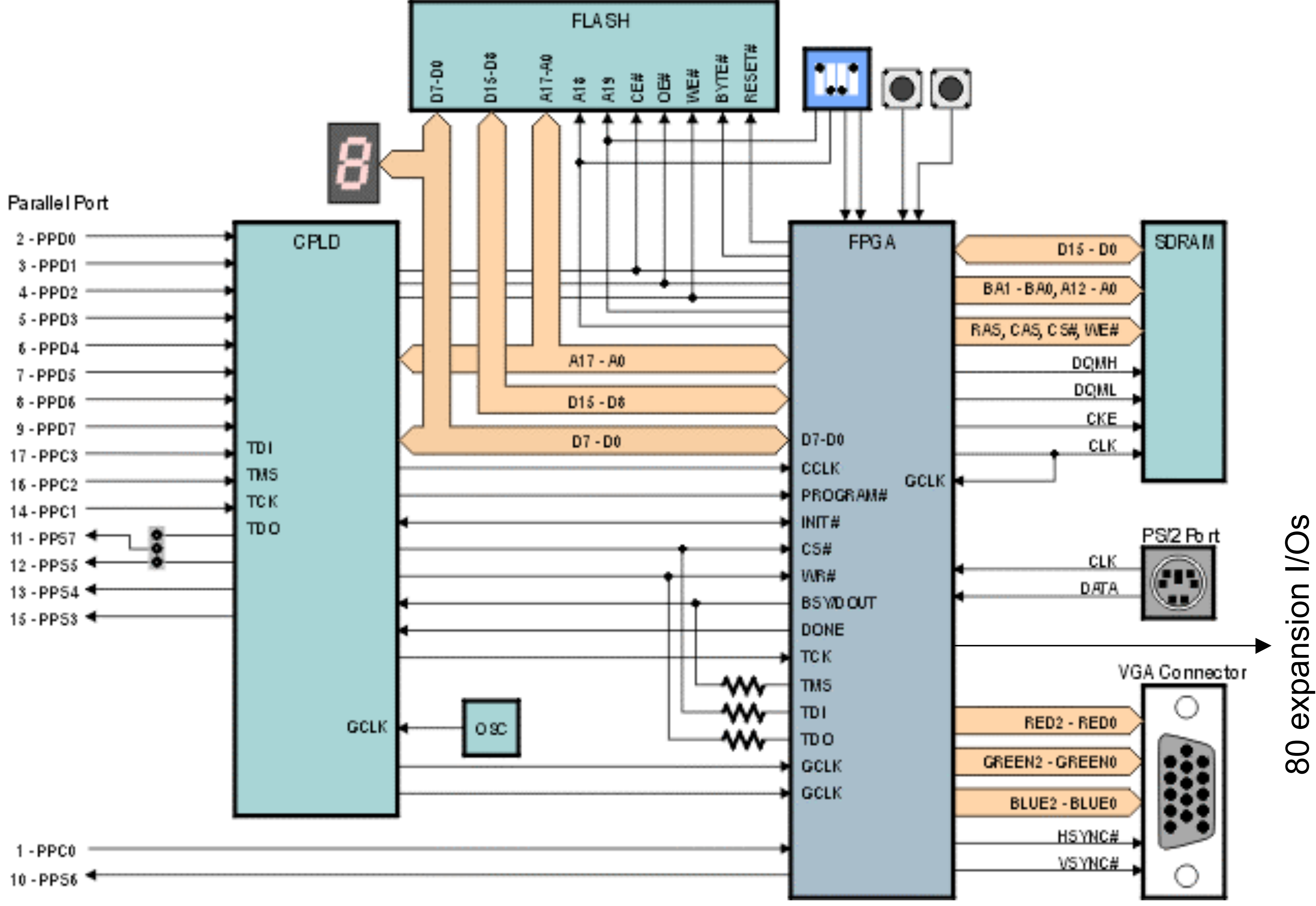
Nexys2 Diagram



XSA-3S1000



XSA-3S1000 Block Diagram



FPGA Software

- Xilinx ISE Web-pack
  - Free!
  - Windows 2000/XP, Red Hat Enterprise Linux 3
  - Supports subset of Xilinx FPGAs (but covers the chips used in the boards listed on the previous slide)
  - No IP Wizard, but
    - You can build memories, logic "by hand" using available components (eg, RAMB16_Sxx) and appropriate defparams or attribute assignments – see Xilinx documentation
    - A *lot* of very good design info in Xilinx App Notes (on-line)
  - Built-in simulator
  - Need computer with parallel port to connect programming cable for boards listed on the previous slide
OR use Digilent USB cable for Digilent Boards