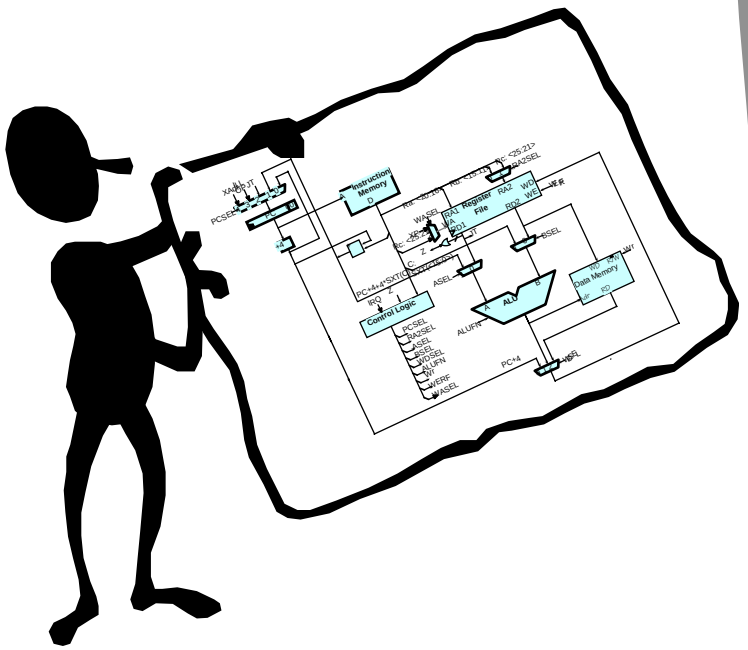
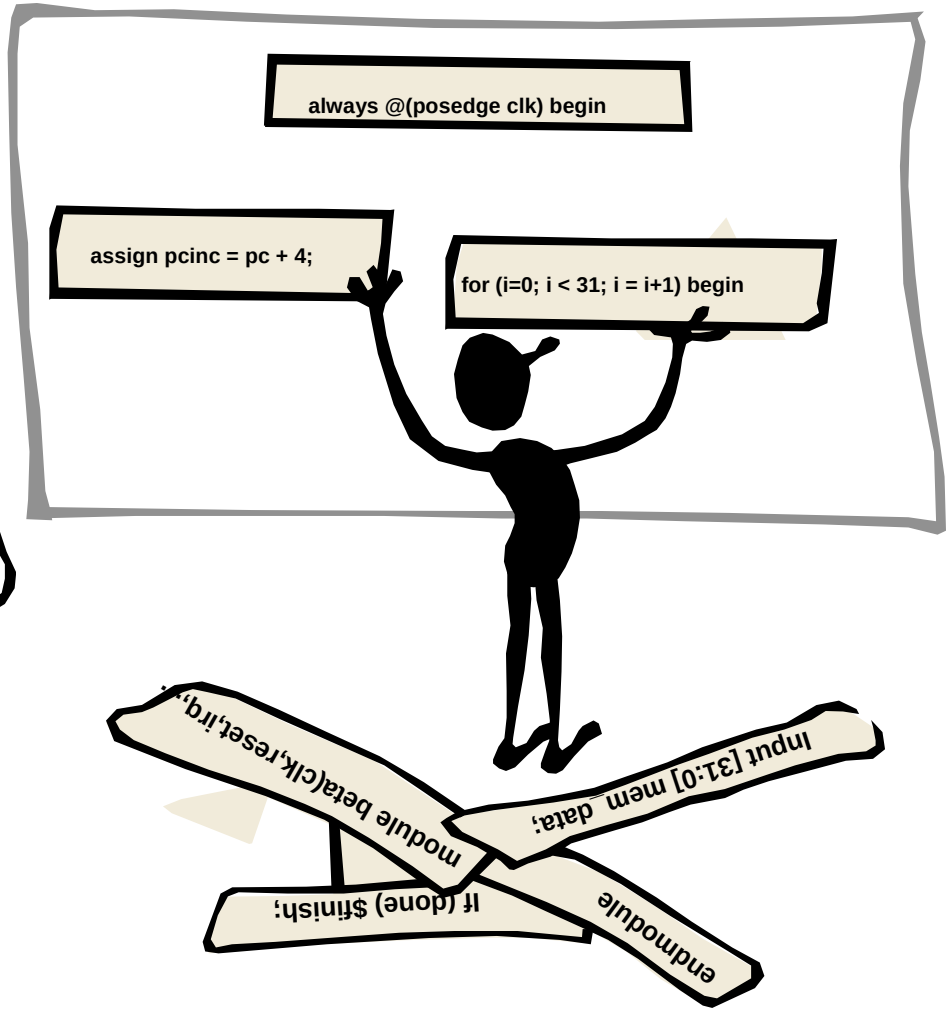
The Last Lecture!

1. Schedule
2. Verilog events
3. FSM++
4. FGPAs @ Home

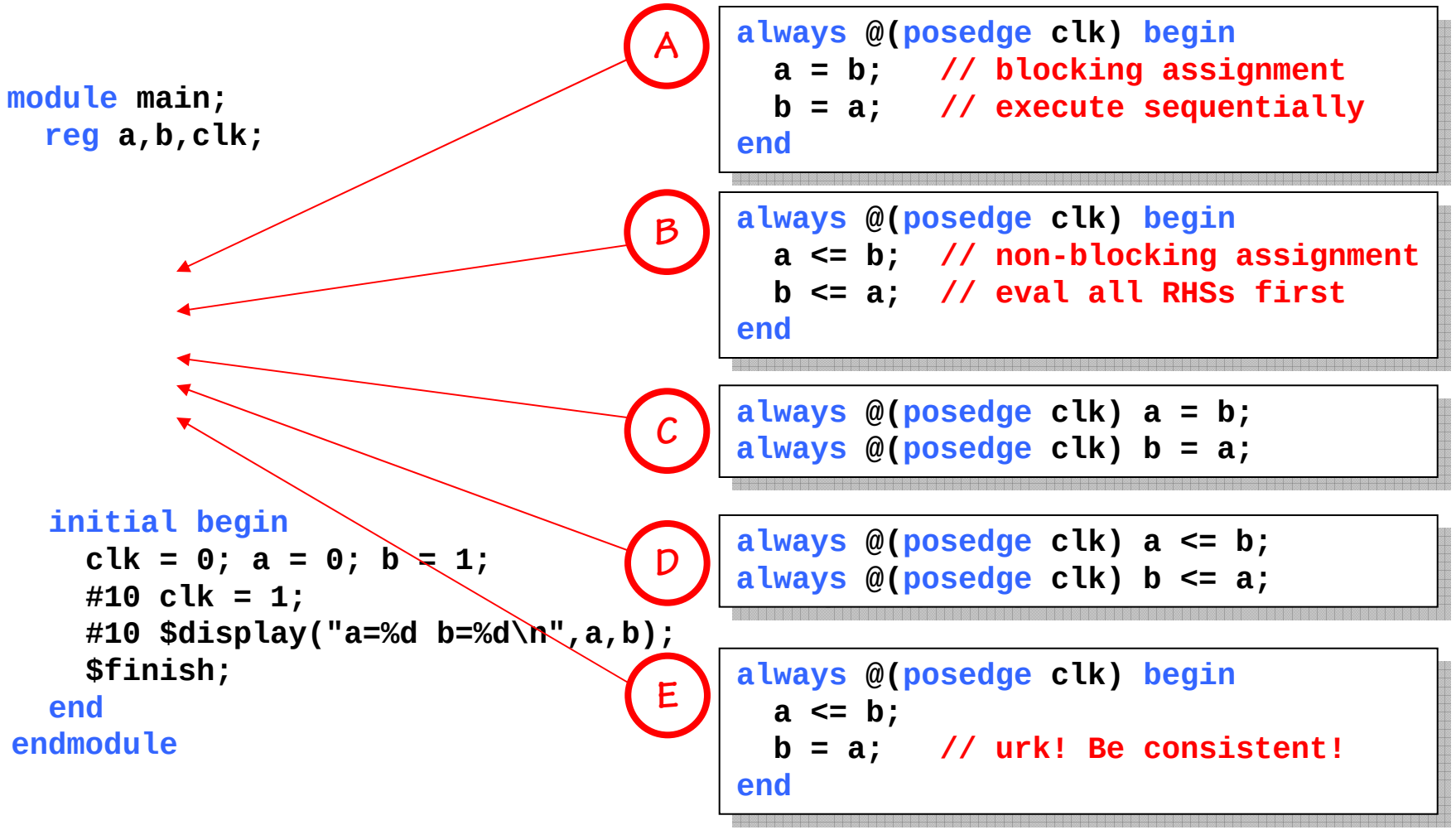




Schedule Reminders

- Wed, 10/26: FSM recap (Willie up in the lab)
- Fri, 10/28: Lab #4 checkoff by 5pm
- Mon, 10/31: upload CI-M Final Version by 5pm
  upload Project Abstract by 5pm
- Wed, 11/02: Quiz, 7:30p – 9:30p, 34-101
- Fri, 11/04: complete proposal meeting with TA
  upload Project Proposal by 5pm
- Fri, 11/11: complete block diagram meeting with TA
- M, Tu, W : 20min design presentations
  11/14–16 schedule TBA (we'll email you!)
  please upload slides to website
- Fri, 11/18: upload Project Checklist by 5pm
- M, Tu, W : project presentations & videotaping
  12/12-14 schedule TBA (we'll email you!)
- Wed, 12/14: upload Final Project Report by 5pm
  (sorry, no extensions possible!)

= vs. <= inside begin … end

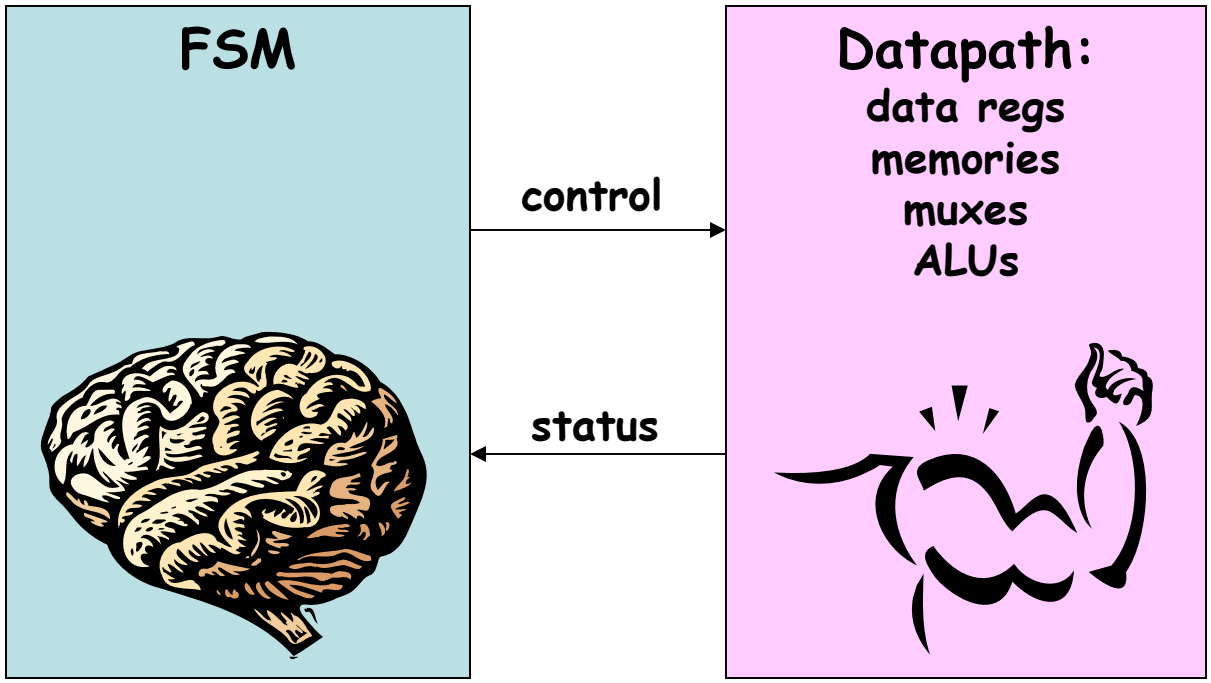


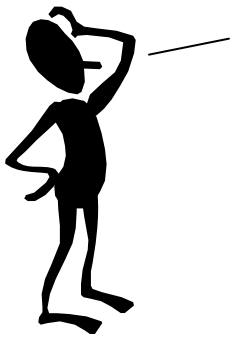
Rule: always change state using <= (e.g., inside `always @(posedge clk)…`)

Verilog Event Processing

- "Active" events
  - Continuous assignments
  - Statements within active `always` blocks
    - Blocking assignments (=)
    - RHS of non-blocking assignments (<=)
- Active events are evaluated in *arbitrary order*
  - Interleaved execution of statements in different active `always` blocks or continuous assignments is possible
  - Statements are executed sequentially only with respect to other statements within the same `always` block
- Assignments to LHS of non-blocking assignments happens after all active events have been processed
- Because of interleaved execution, blocking assignments can lead to *nondeterministic behavior* (this is bad!).

Digital Systems = FSMs + Datapath

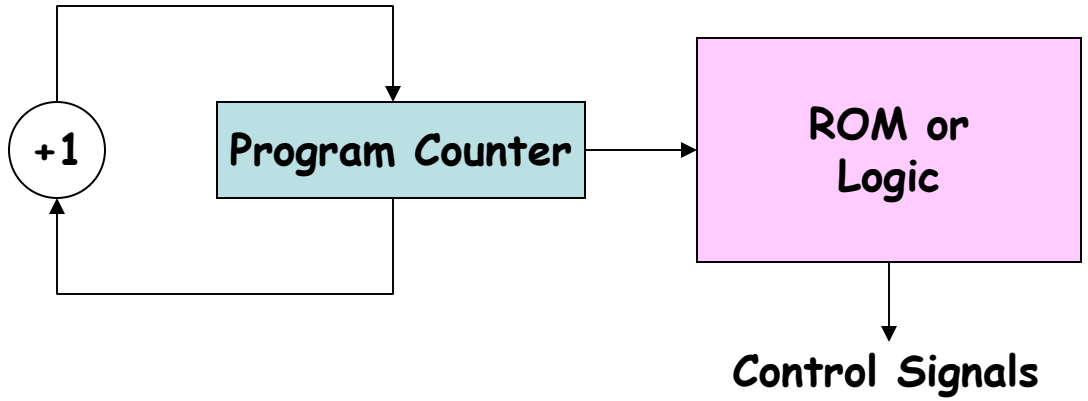




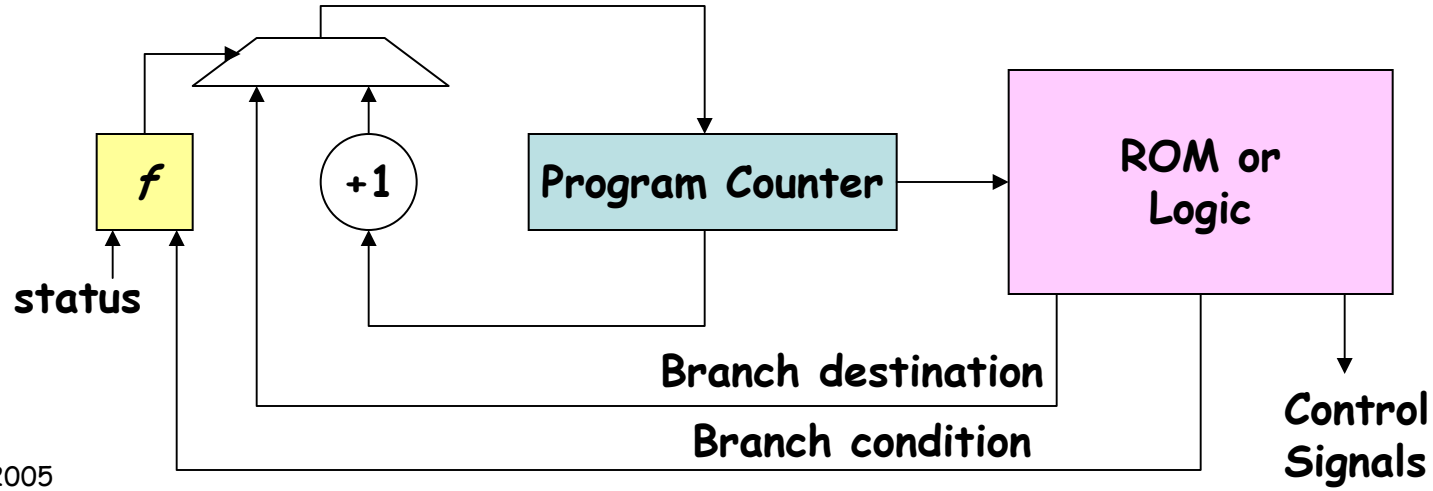
But what if my FSM has hundreds or thousands of states? That's a BIG case statement!

Microsequencers

Step 1: use a counter for the state

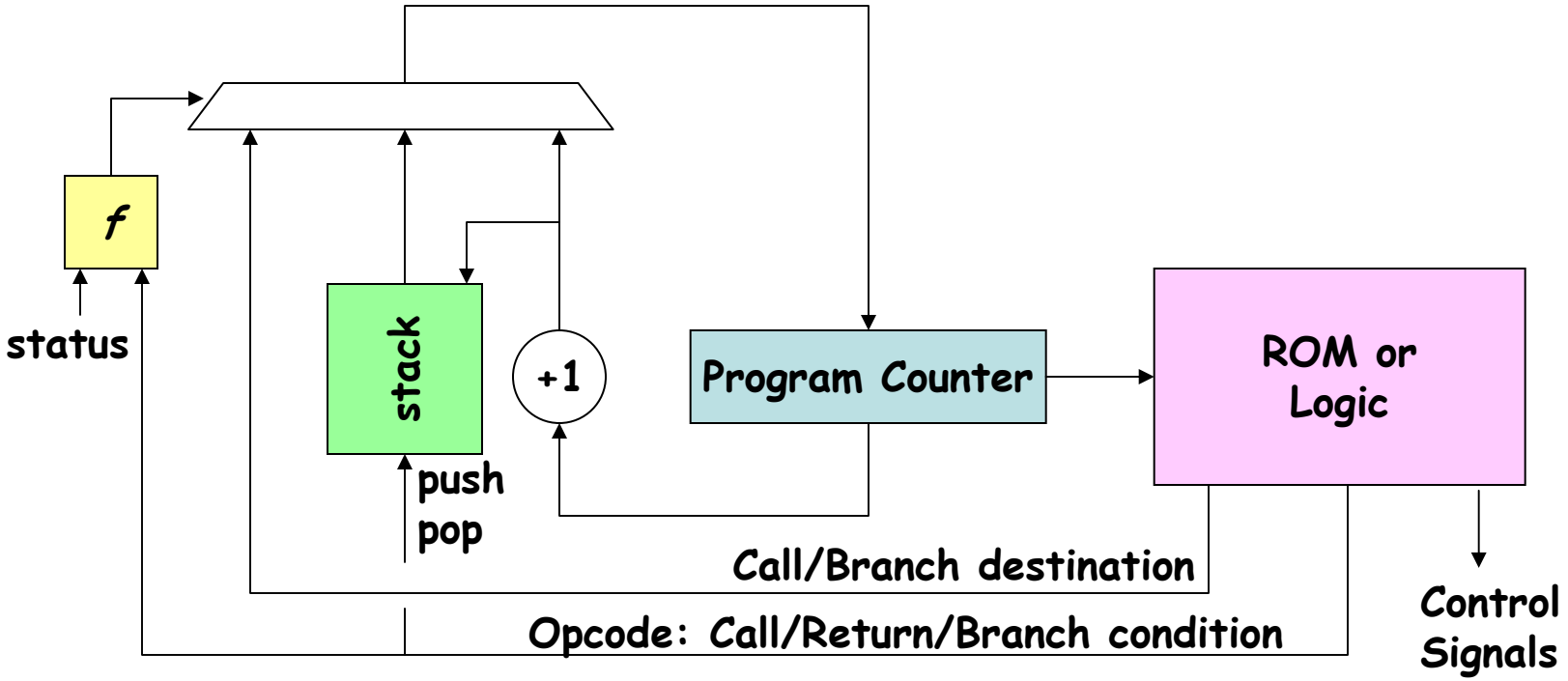


Step 2: add a conditional branch mechanism



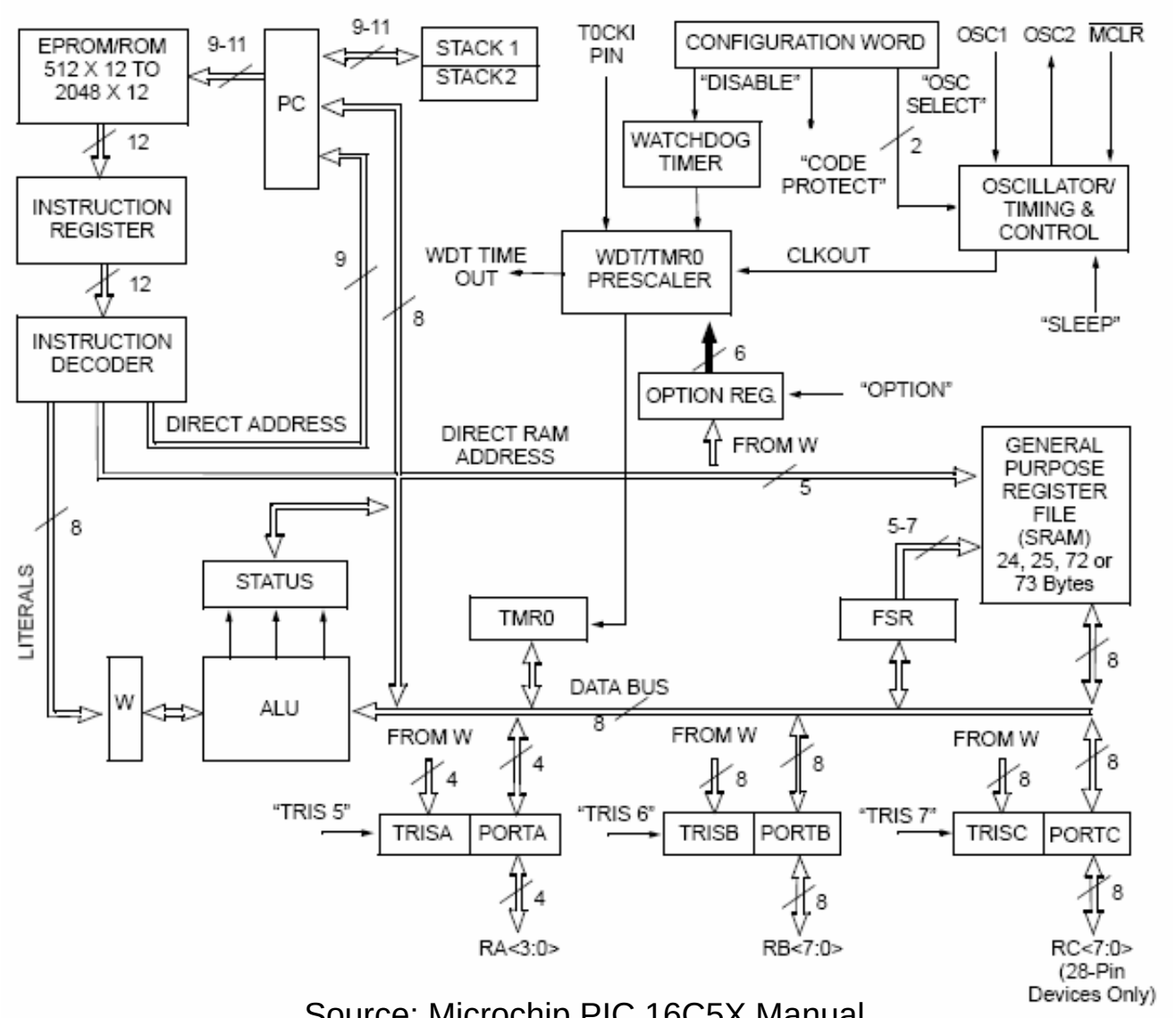
Microsequencers (cont'd.)

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

PIC Processor: Microsequencer + general-purpose datapath



Source: Microchip PIC 16C5X Manual

PIC Instruction Set

Mnemonic, Operands		Description	Cycles	12-Bit Opcode MSb		LSb	Status Affected
ADDWF	f,d	Add W and f	1	0001	11df	ffff	C,DC,Z
ANDWF	f,d	AND W with f	1	0001	01df	ffff	Z
CLRF	f	Clear f	1	0000	011f	ffff	Z
CLRW	–	Clear W	1	0000	0100	0000	Z
COMF	f, d	Complement f	1	0010	01df	ffff	Z
DECF	f, d	Decrement f	1	0000	11df	ffff	Z
DECFSZ	f, d	Decrement f, Skip if 0	1(2)	0010	11df	ffff	None
INCF	f, d	Increment f	1	0010	10df	ffff	Z
INCFSZ	f, d	Increment f, Skip if 0	1(2)	0011	11df	ffff	None
IORWF	f, d	Inclusive OR W with f	1	0001	00df	ffff	Z
MOVF	f, d	Move f	1	0010	00df	ffff	Z
MOVWF	f	Move W to f	1	0000	001f	ffff	None
NOP	–	No Operation	1	0000	0000	0000	None
RLF	f, d	Rotate left f through Carry	1	0011	01df	ffff	C
RRF	f, d	Rotate right f through Carry	1	0011	00df	ffff	C
SUBWF	f, d	Subtract W from f	1	0000	10df	ffff	C,DC,Z
SWAPF	f, d	Swap f	1	0011	10df	ffff	None
XORWF	f, d	Exclusive OR W with f	1	0001	10df	ffff	Z
BIT-ORIENTED FILE REGISTER OPERATIONS							
BCF	f, b	Bit Clear f	1	0100	bbbf	ffff	None
BSF	f, b	Bit Set f	1	0101	bbbf	ffff	None
BTFSC	f, b	Bit Test f, Skip if Clear	1 (2)	0110	bbbf	ffff	None
BTFSS	f, b	Bit Test f, Skip if Set	1 (2)	0111	bbbf	ffff	None
LITERAL AND CONTROL OPERATIONS							
ANDLW	k	AND literal with W	1	1110	kkkk	kkkk	Z
CALL	k	Call subroutine	2	1001	kkkk	kkkk	None
CLRWDT	k	Clear Watchdog Timer	1	0000	0000	0100	$\overline{TO}$, $\overline{PD}$
GOTO	k	Unconditional branch	2	101k	kkkk	kkkk	None
IORLW	k	Inclusive OR Literal with W	1	1101	kkkk	kkkk	Z
MOVLW	k	Move Literal to W	1	1100	kkkk	kkkk	None
OPTION	k	Load OPTION register	1	0000	0000	0010	None
RETLW	k	Return, place Literal in W	2	1000	kkkk	kkkk	None
SLEEP	–	Go into standby mode	1	0000	0000	0011	$\overline{TO}$, $\overline{PD}$
TRIS	f	Load TRIS register	1	0000	0000	0fff	None
XORLW	k	Exclusive OR Literal to W	1	1111	kkkk	kkkk	Z

Source: Microchip PIC 16C5X Manual

PIC Example: Verilog

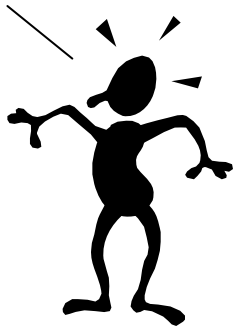
```
 // PIC core
 wire [9:0] pc;
 wire [11:0] idata;
 wire [3:0] porta;
 wire [7:0] portb;
 pic16c5x pic(.clk(clk),.reset(reset),
              .pc(pc),.idata(idata),     // instruction memory
              .port_a_out(an[3:0]),      // digit select
              .port_b_out(segment[7:0]), // display segments
              .port_c_in(sw[7:0])        // slide switches
              );
defparam pic.PCMSB = 9;   // 1024 inst locations

// PIC program memory (1024 x 12): test.pic
test pgm({1'b0,pc},~clk,idata);
```

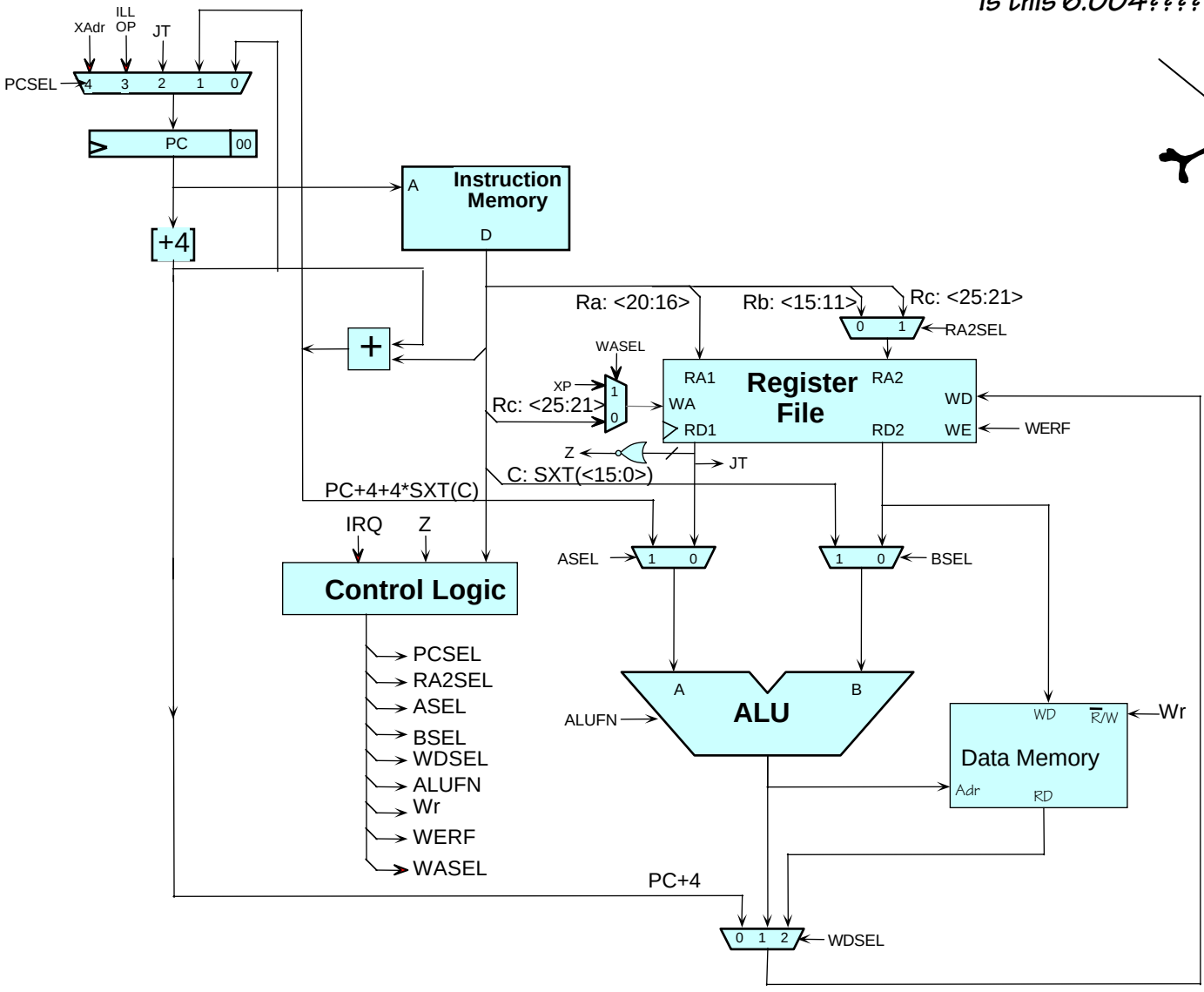
PIC Example: test.pic

```
sw    equ h'0018'
digit equ h'0019'
count equ h'001A'
Start   movlw  0
        tris   PORTA        ; ports A and B are outputs
        tris   PORTB
        movlw  h'ff'
        tris   PORTC        ; port C are inputs
        movlw  h'f'
        movwf  PORTA
        clrf   count
Loop
        movlw  h'f'
        movwf  PORTA
        movf   PORTC,0      ; read switches into W
        movwf  sw
        andlw  h'f'         ; keep low 4 bits
        call   Segments
        movwf  PORTB        ; set up segments
        movlw  h'e'
        movwf  PORTA        ; turn on AN[0]
buzz1
        decfsz count,1     ; display this digit for a while
        goto   buzz1
        …
```

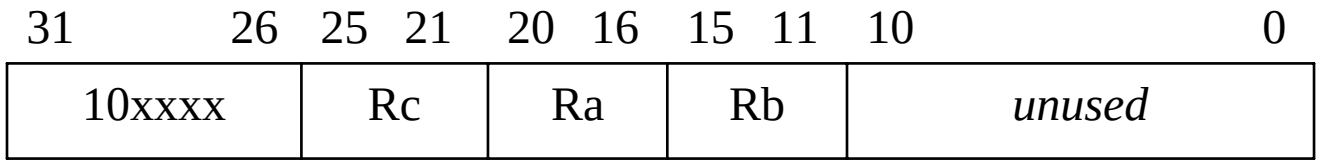
A "Real" Processor: the Beta!

Is this 6.004?????



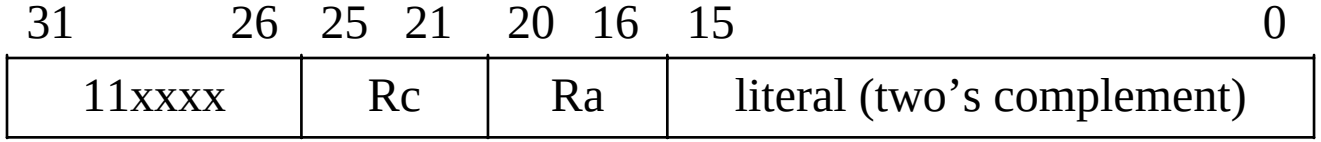


Beta Instructions - I



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

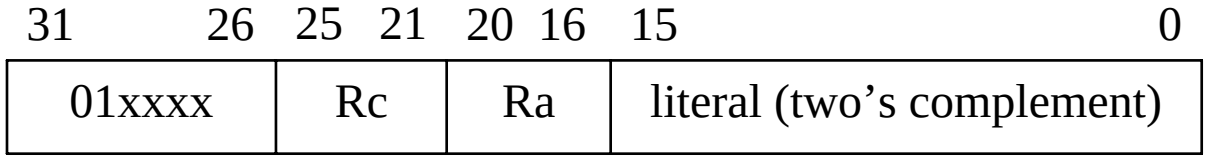
Opcodes:
ADD (plus), SUB (minus), **MUL (multiply)**, **DIV (**divided by),
AND (bitwise and), **OR (**bitwise or), **XOR (**bitwise exclusive or)
CMPEQ (equal), CMPLT (less than), **CMPLE (**less than or equal) [result = 1 if true, 0 if false]
SHL (left shift), **SHR (**right shift w/o sign extension),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



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



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)

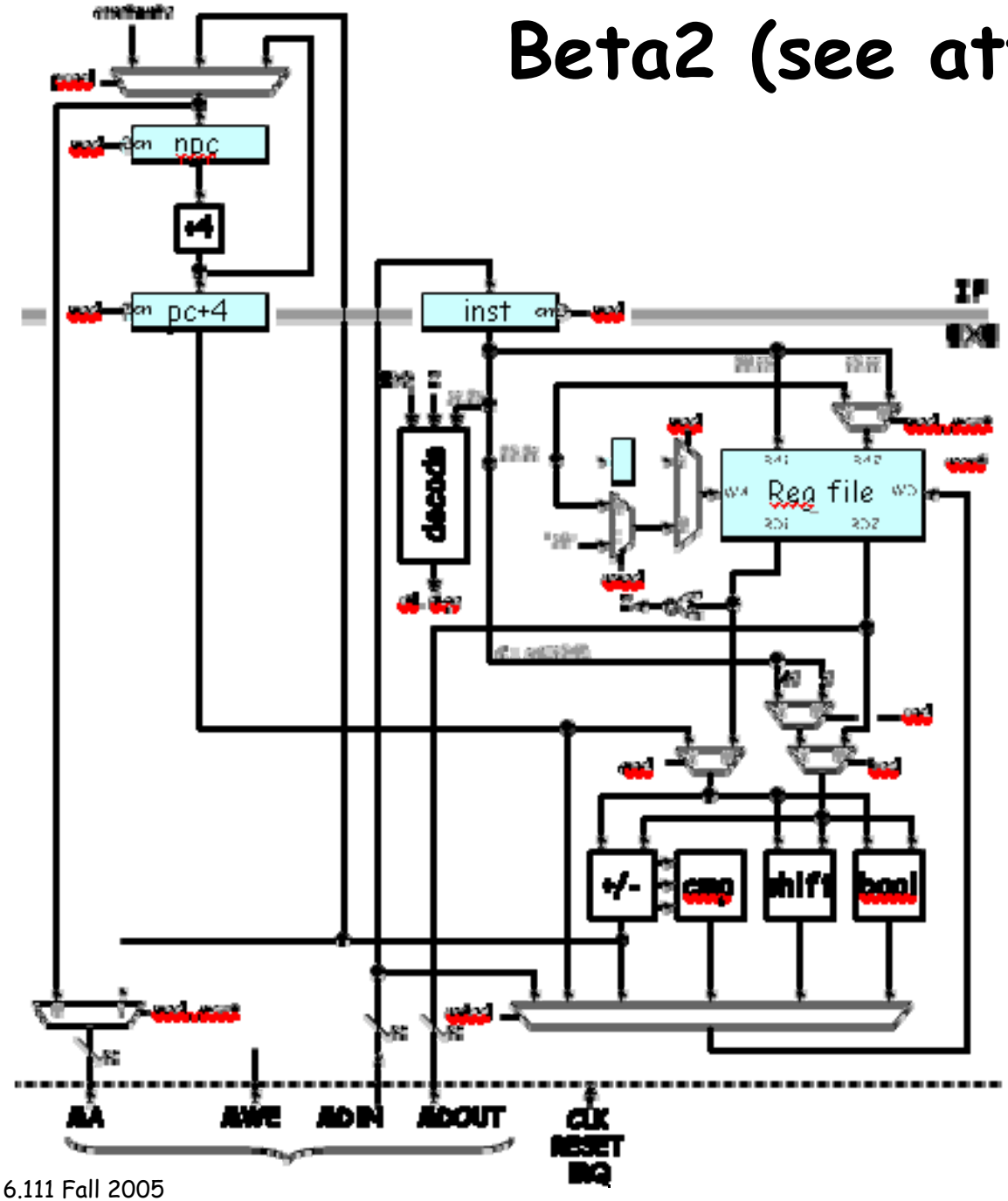
BNE/BT(Ra,label,Rc): Reg[Rc] ← PC + 4;
if Reg[Ra] ≠ 0 then PC ← PC + 4 + 4*SEXT(literal)

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

Beta Control Logic

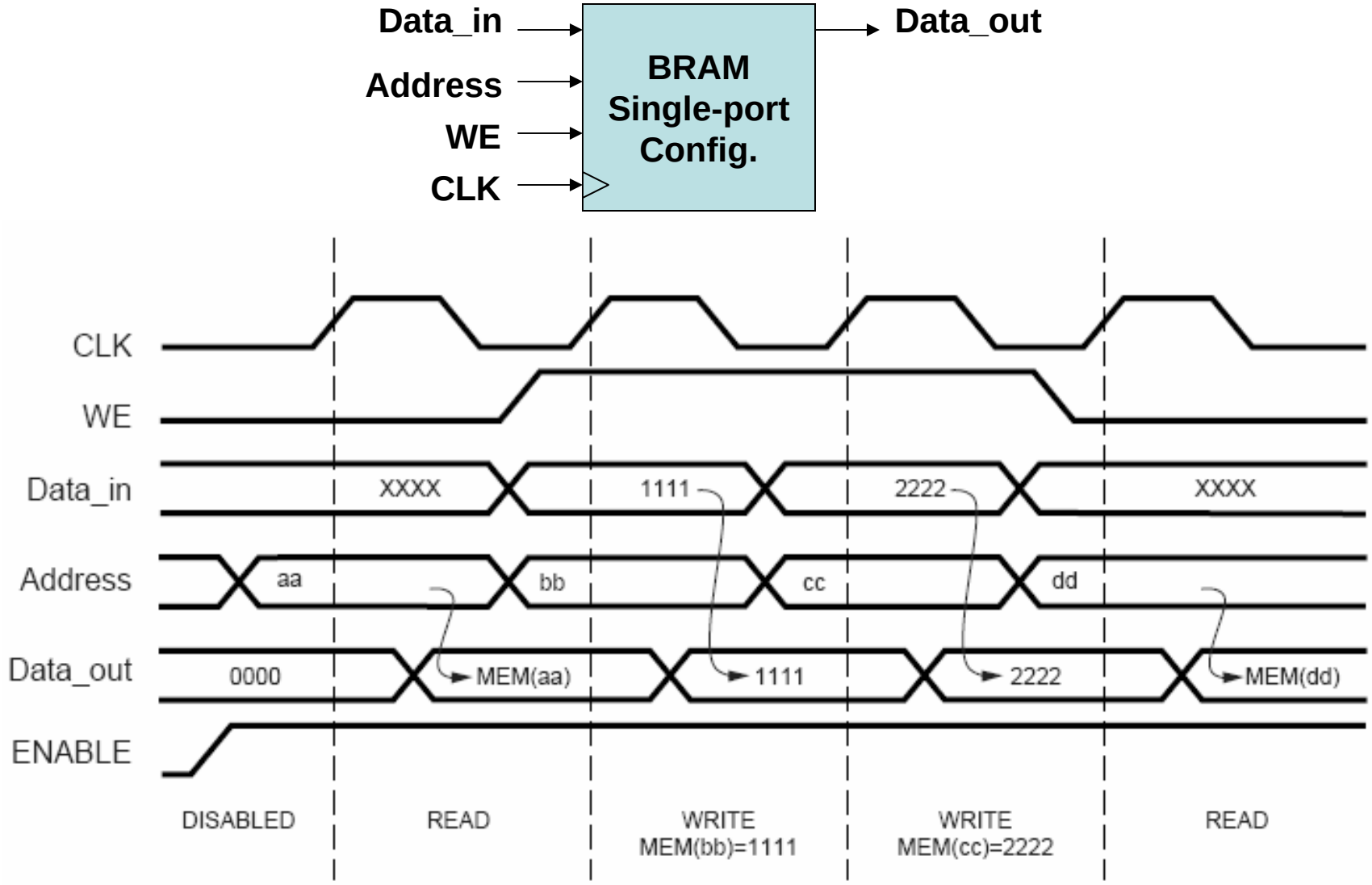
	OP	OPC	LD	ST	JMP	BEQ	BNE	LDR	Illop	trap
ALUFN	F(op)	F(op)	"+"	"+"	—	—	—	"A"	—	—
WERF	1	1	1	0	1	1	1	1	1	1
BSEL	0	1	1	1	—	—	—	—	—	—
WDSEL	1	1	2	—	0	0	0	2	0	0
WR	0	0	0	1	0	0	0	0	0	0
RA2SEL	0	—	—	1	—	—	—	—	—	—
PCSEL	0	0	0	0	2	Z ? 1 : 0	Z ? 0 : 1	0	3	4
ASEL	0	0	0	0	—	—	—	1	—	—
WASEL	0	0	0	—	0	0	0	0	1	1

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

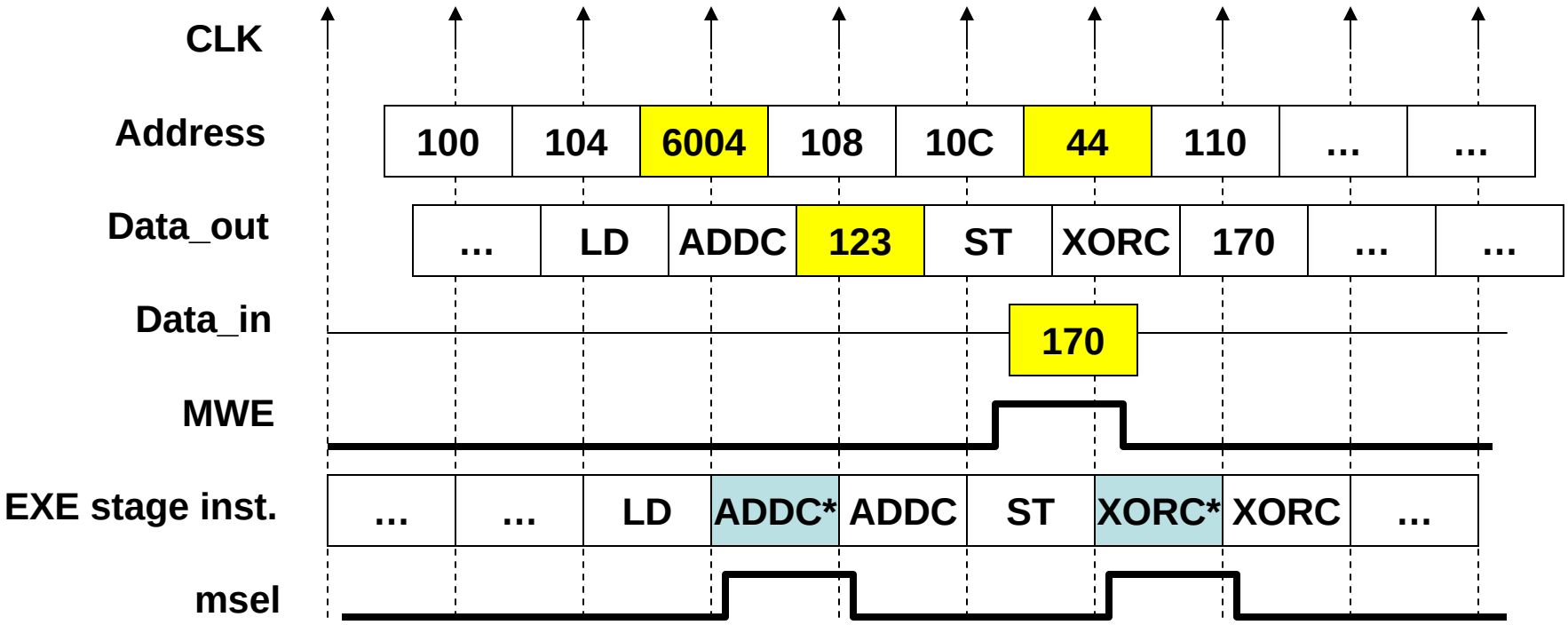
Xilinx Synchronous Block Memory



Source: Xilinx App Note 463

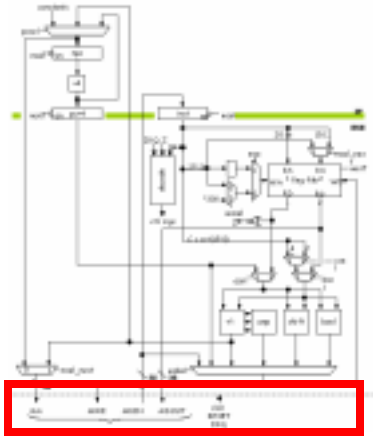
```
100: LD(R31,6004,R2)
104: ADDC(R2,47,R2)
108: ST(R2,44,R31)
10C: XORC(R2,-1,R2)
110: …
…
6004: 123
```

Instruction Pipeline Diagram



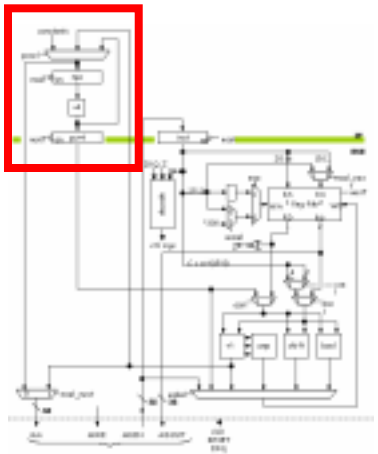
*** Stalled in pipeline**

beta2.v



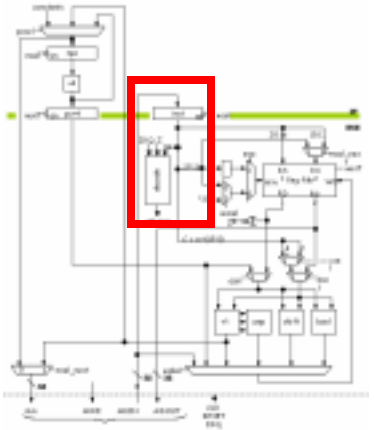
```
module beta2(clk,reset,irq,ma,mdin,mdout,mwe);
  input clk,reset,irq;
  output [31:0] ma,mdout;
  input [31:0] mdin;
  output mwe;

  // 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



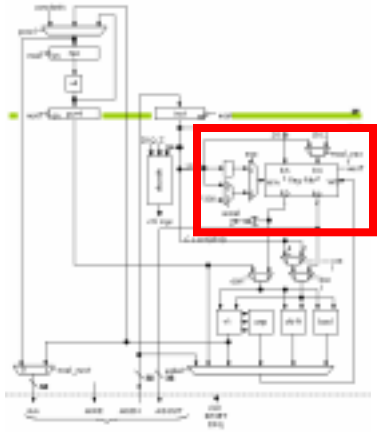
```
// pc
wire [31:0] npc_inc,npc_next;

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



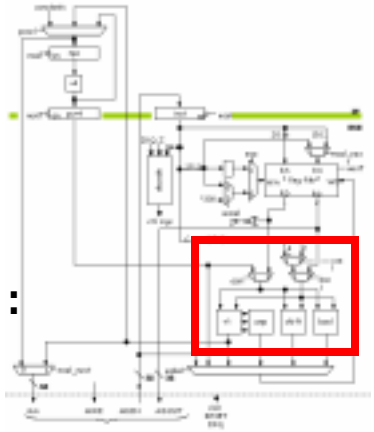
```
// instruction reg
always @ (posedge clk) if (!msel) inst <= mdin;

// 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



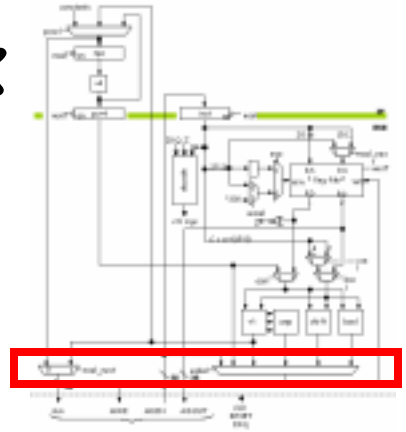
```
// register file
wire [4:0] ra1,ra2,wa;
always @ (posedge clk)
  if (!msel) rc_save <= inst[25:21];

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



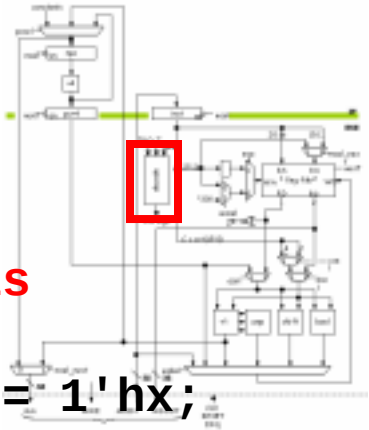
```
// alu
assign a = asel ? pc_inc : rd1;
assign b = bsel ? c : rd2;
assign c = csel ? {{14{inst[15]}},inst[15:0],2'b00} :
                  {{16{inst[15]}},inst[15:0]};

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)

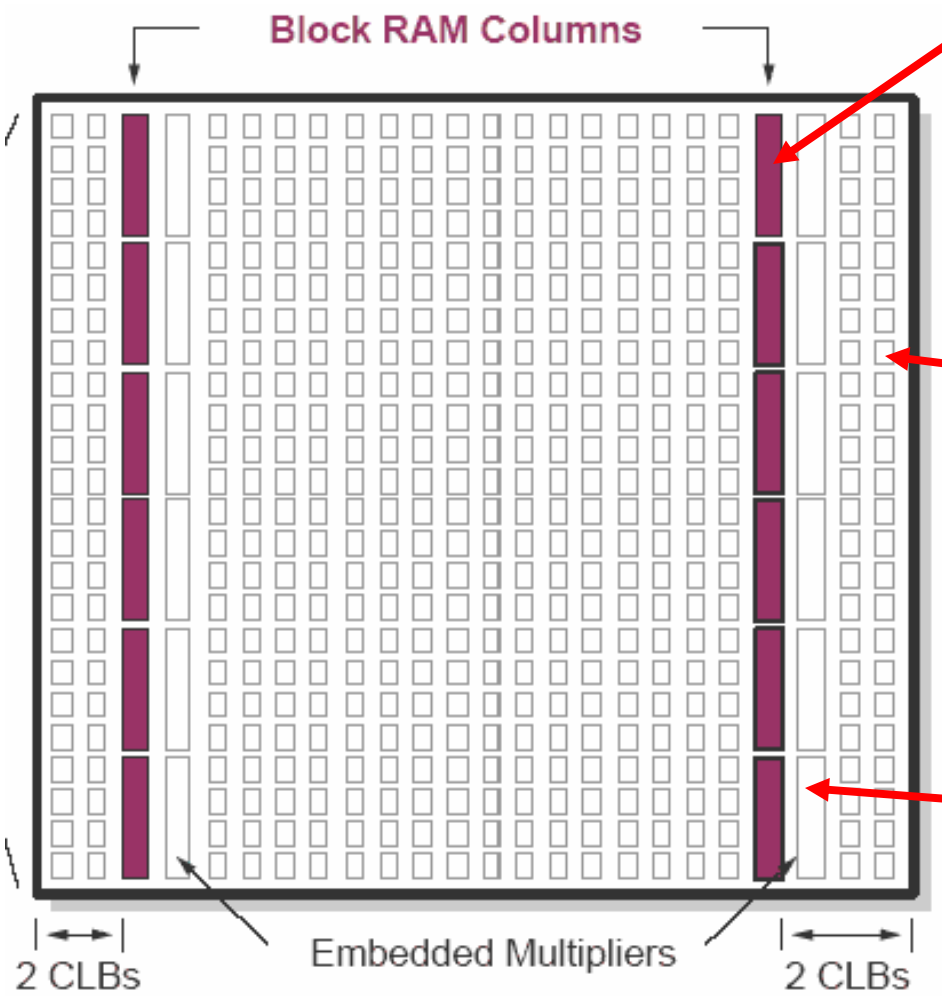


```
always @ (opcode or z or annul or irq or reset)
 begin
   // initial assignments for all control signals
   asel = 1'hx; bsel = 1'hx; csel = 1'hx;
   addsub_op = 1'hx; shift_op = 1'hx; shift_sxt = 1'hx;
   cmp_lt = 1'hx; cmp_eq = 1'hx;
   boole_and = 1'hx; boole_or = 1'hx;
   wasel = 0; mem_next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



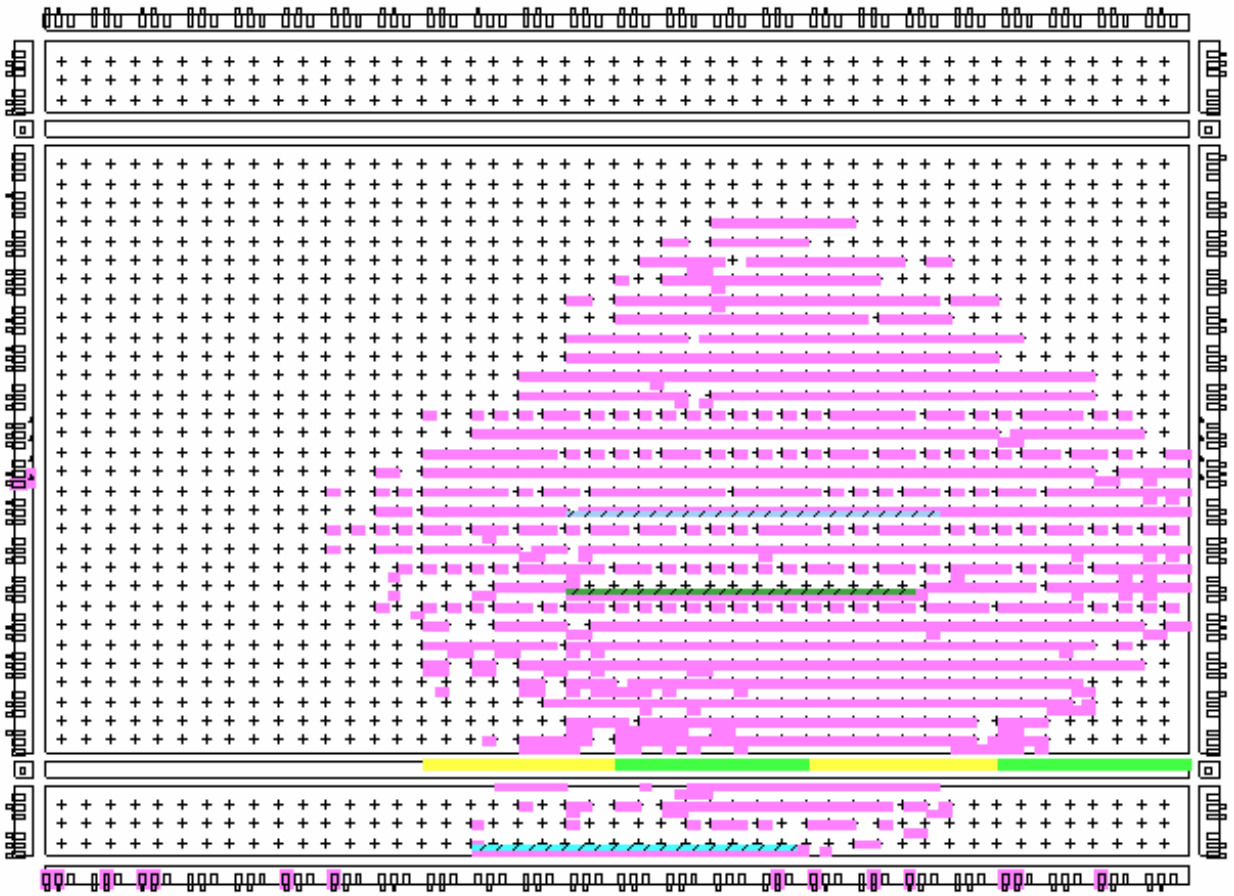
Memories:
ROM/RAM, Single/dual port
16Kx1 … 512x32
Quantity: 12

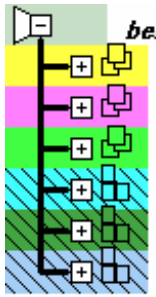
CLBs:
24 rows x 20 cols = 480
= 4320 logic cells
= 200K gates
Beta2 takes ~30%

18x18 Multipliers:
Sync/async
Quantity: 12

Source: Xilinx App Note 463

Beta2 Floorplan





beta2demo "beta2demo" [15 IOBs, 4 BLKRAMs, 128 DPRAMs, 1051 FGs, 60 FG5s, 90 CYs, 174 DFFs, 1 BUFG]

imem "rom_1k_32" [2 BLKRAMs]

beta2demo "Primitives" [15 IOBs, 128 DPRAMs, 1051 FGs, 60 FG5s, 3 CYs, 174 DFFs, 1 BUFG]

dmem "ram_1k_32" [2 BLKRAMs]

CarryChain_5 "CarryChain" [27 CYs] ← **Clock divider for 7-seg display**

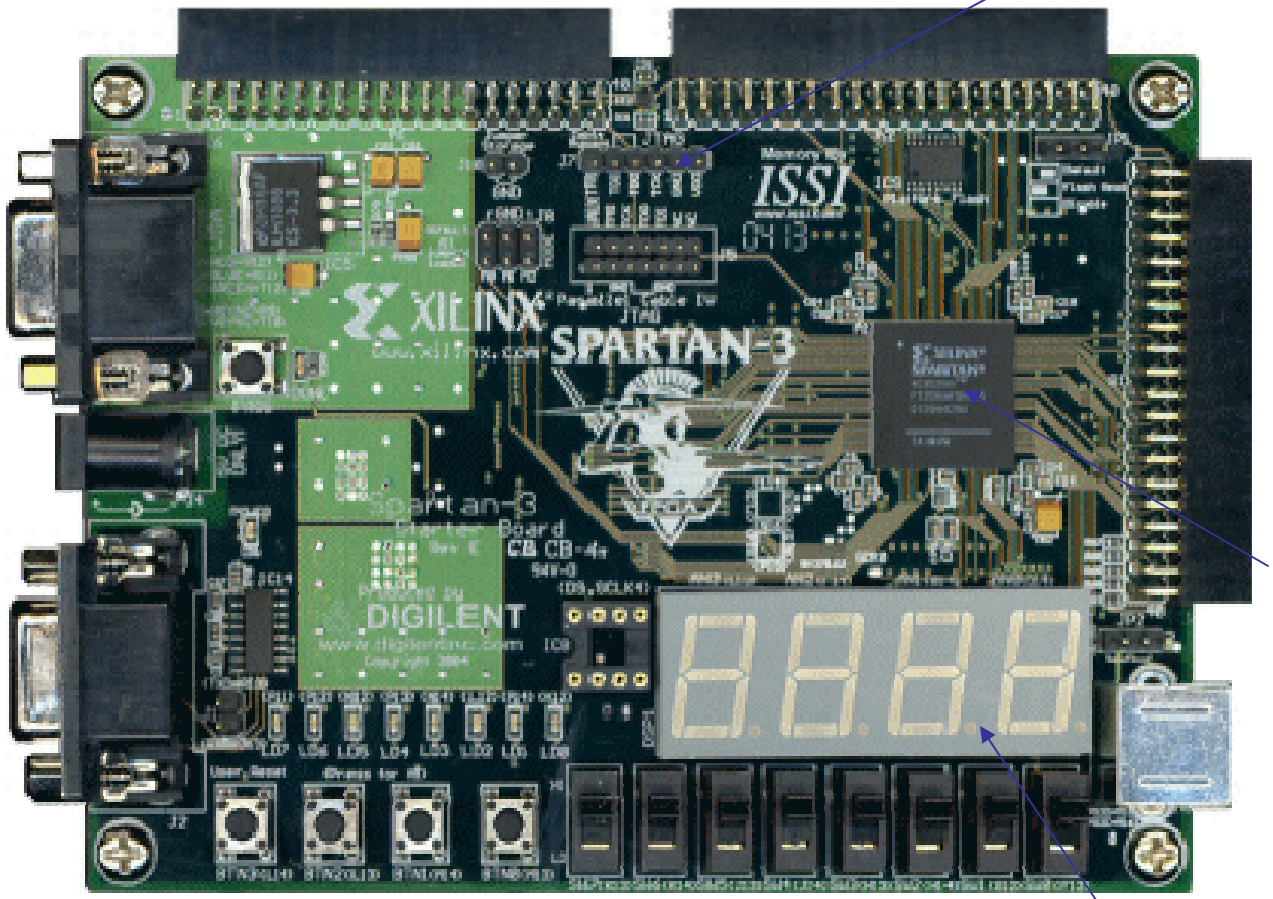
CarryChain_3 "CarryChain" [29 CYs] ← **npc + 4**

CarryChain_1 "CarryChain" [31 CYs] ← **alu adder**

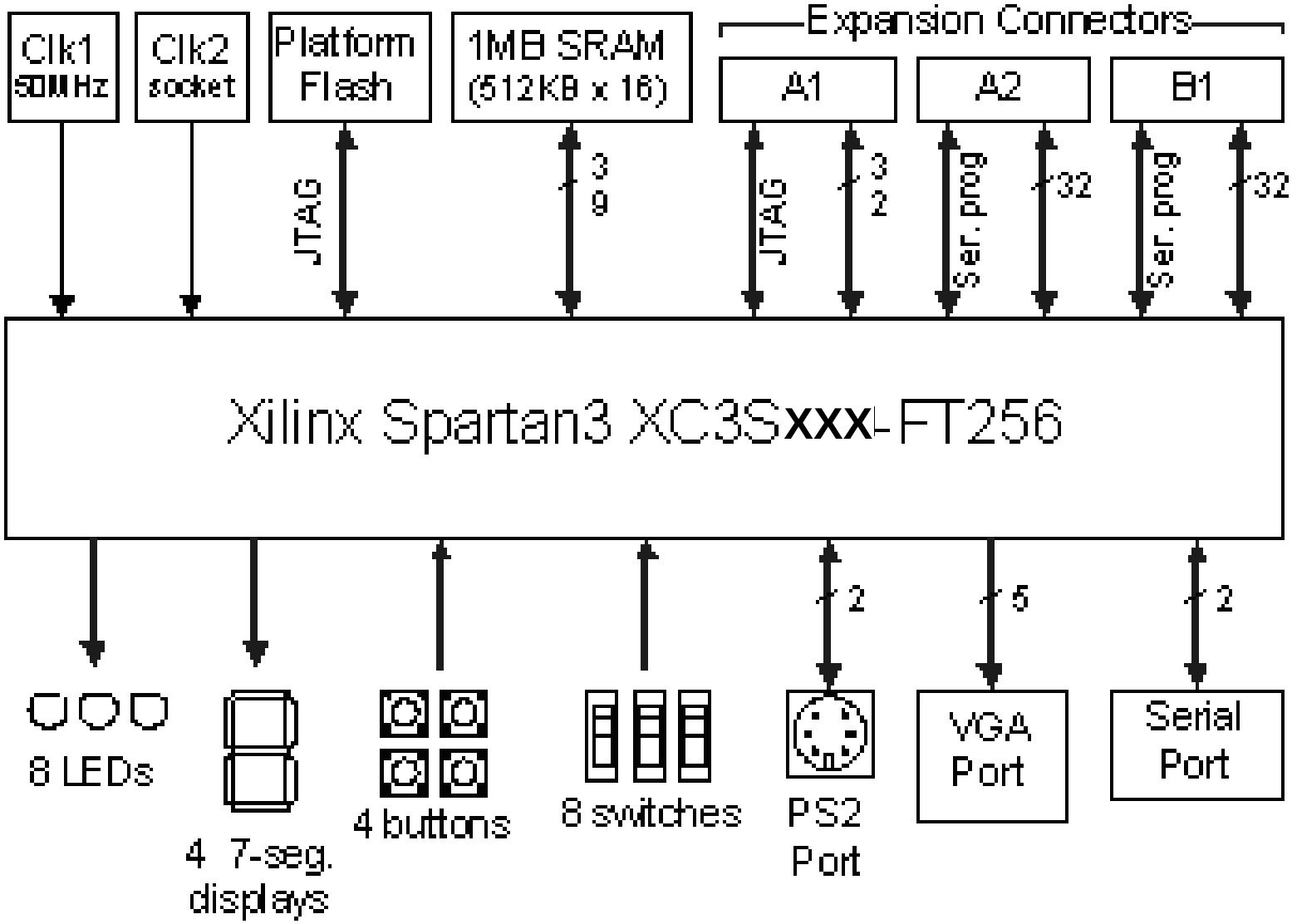
FPGAs @ Home

- 6.111 labkit: the Lexus of FPGA protoboards
  - XC2V6000 (76032 logic cells, 2.5 Mbits BRAM)
- Two affordable alternatives (lots more out there)
  - Spartan-3 Starter Board (www.digilentinc.com)
    - $99 = Spartan XC3S200 (4320 logic cells, 216 Kbits BRAM)
    - $119 = Spartan XC3S400 (8064 logic cells, 288 Kbits BRAM)
    - $149 = Spartan XC3S1000 (17480 logic cells, 432 Kbits BRAM)
    - Switches, buttons, leds, 4-digit display
    - 1Mbyte 10ns SRAM
    - PS2, serial port, 8-color VGA, 3 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)

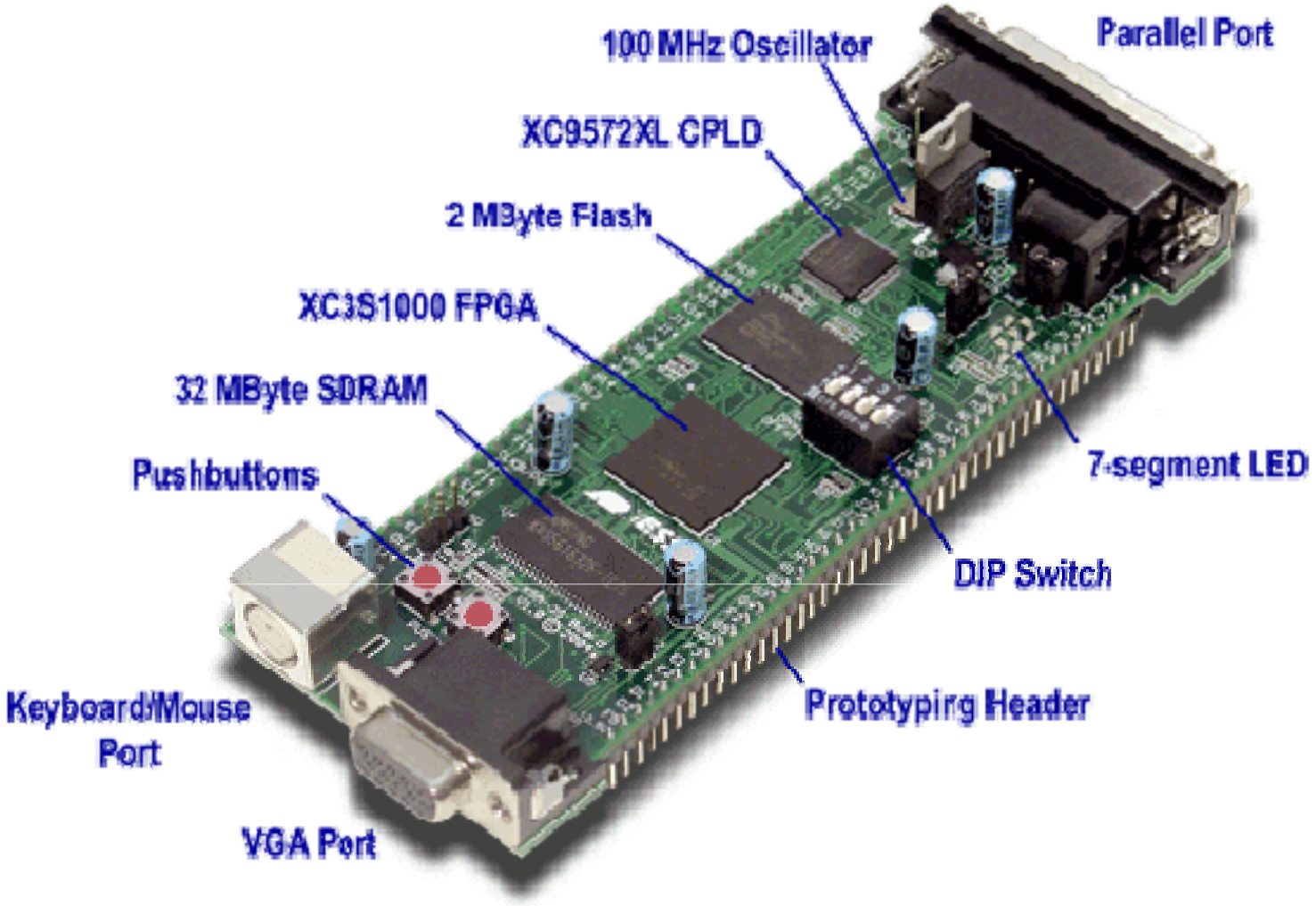
Xilinx Spartan-3 Starter Board



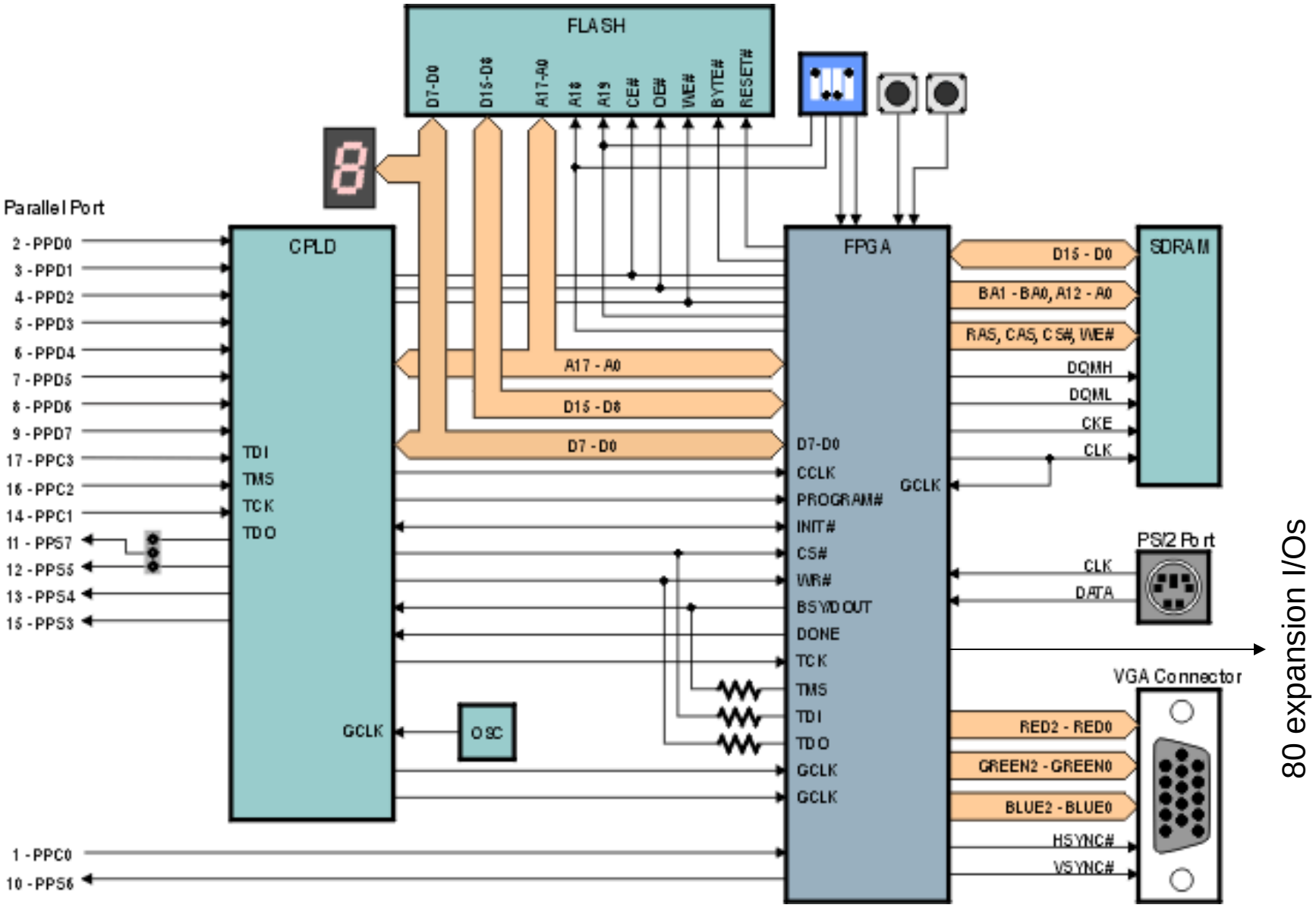
Spartan-3 Starter Board Diagram



XSA-3S1000


100 MHz Oscillator
Parallel Port
XC9572XL CPLD
2 MByte Flash
XC3S1000 FPGA
32 MByte SDRAM
Pushbuttons
7-segment LED
DIP Switch
Keyboard/Mouse Port
Prototyping Header
VGA Port

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)
  - Wimpy simulator
  - Need computer with parallel port to connect programming cable for boards listed on the previous slide