

Design & Implement a Custom Micro-Controller in One Week or Less

AKA

Hello World for FPGAs

**Jim Brakefield
Brakefield Research
&
On Board Software Inc.**

Spartan-3 Evaluation Board

From Xilinx or www.digilentinc.com for \$99

Spartan-3 XC3S200: 12 18K-bit block RAMs, 3840 LUTs

JTAG programming cable

VGA connector, keyboard connector

Four digit / 7-segment multiplexed display

Four digit enables, active low

Eight segment enables, active low

Switches, LEDs, buttons

256Kx32 SRAM

Serial EPROM for booting FPGA chip

WebPack software

Design Realm

Single Clock Synchronous Design

All FF & registers clocked on rising edge

Block RAM synchronous only

Distributed RAM with asynchronous read and synchronous write

Therefore:

Read Block RAM for instructions at rising clock edge

Decode instruction & forward register address to distributed RAM

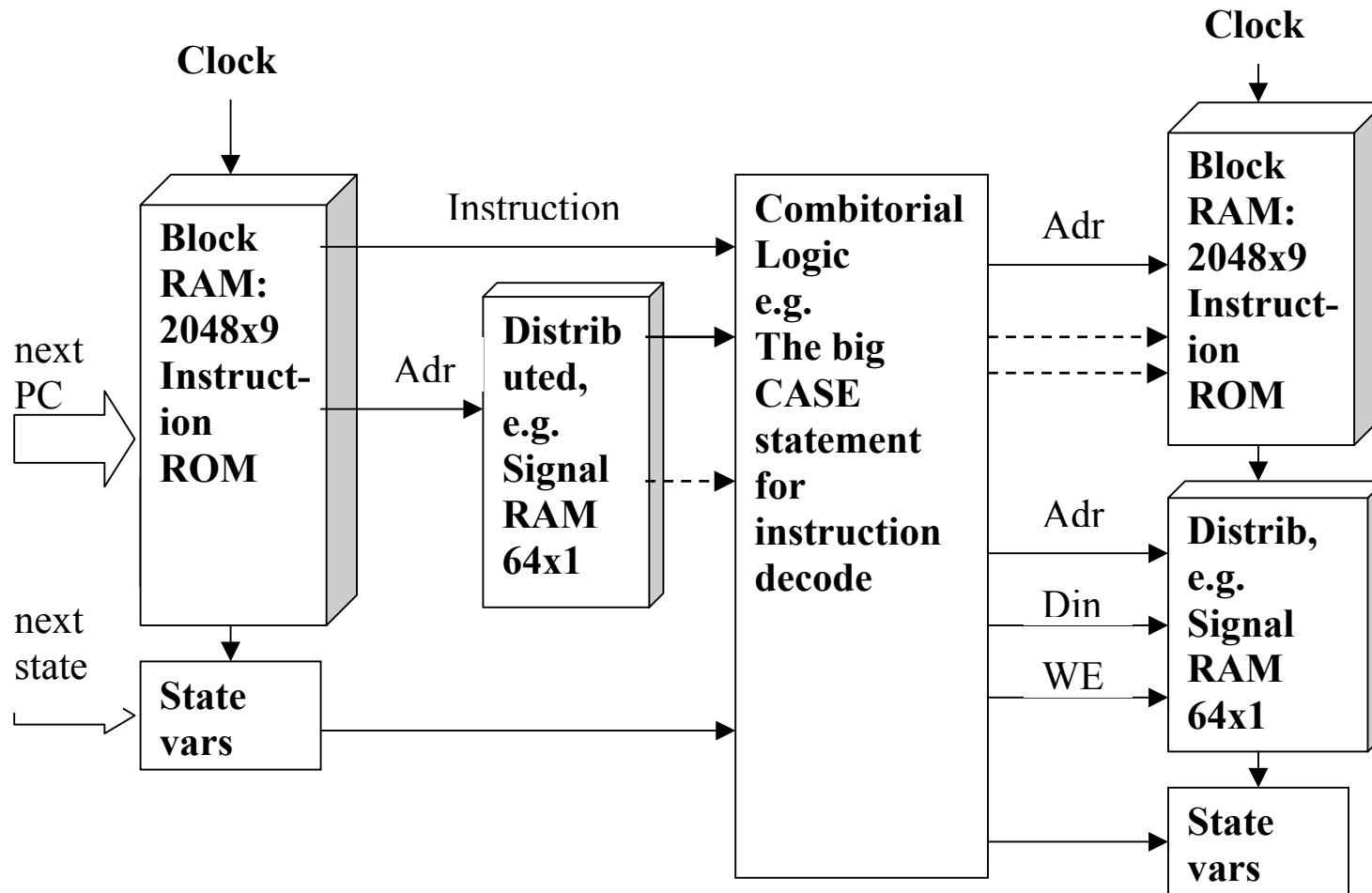
Perform combinatorial function on distributed RAM value

Write **combinatorial** result back to distributed RAM, block RAM,
and registers/FF on next rising clock edge

Scheme works as block & distributed RAM very fast (~2 & ~1 ns)

And distributed RAM write occurs during block RAM read

Design Realm



And now for a really simple micro-controller

3-bit op-code, 6-bit address

Block RAM organized as 2048x9

Bit serial arithmetic

Memory organized as 64x1

Execute / Wait cycle

Execute until halt instruction, wait for next start signal

Designed for logic emulation

And really low LUT counts

Instruction Set

LD	load memory bit to accumulator
LDC	load complement memory bit to accumulator
ST	store accumulator to memory
AND	and memory bit with accumulator
OR	or memory bit with accumulator
XOR	exclusive-or memory bit with accumulator
ADC	add memory bit to accumulator and update carry
HLT	zero program counter and wait for start signal
Misc	variety of functions on accumulator and carry (clear, set, complement, copy, swap)

VHDL Code

Processor (d3_1em9_1min.vhd):

- RAM definitions

- Interface description

- Signal declarations

- RAM Port maps, Initialize Block RAM with assembler output

- Combinatorial section

- Register update section

Test Fixture(d3_1em9_1min_hw.vhd):

- Define interface to circuit board

- Watch distributed RAM writes

Constraint File(d3_1em9_1min_hw.ucf): pin #s and clock speed

Register Update Section

```
update: process(clk,reset) begin if reset='1' -- master reset
then    dly <= hlt;
        acc <= '0'; ..... mem_rd <= (others => '0');
elsif (clk'event and clk='1') then
    dly    <= nxdly;                -- state variable update
    if xacc = '1'    then acc <= nxacc; end if; -- accum update
    if xcry = '1'    then cry <= nxcry; end if; -- update carry bit
    if xpc = '1'     then pc  <= nxpc; end if; -- PC update
----    monitoring signals
        acc_cpy    <= acc; ..... mem_rd    <= inst & nxadr;
end if;    end process update;
```


Combinatorial Section: default values

-- instruction processing

decode: **process**(dly,start,memrd,acc,cry,inst,pc,ir) **begin**

-- default values for update enables & "nx" signals

nxdlyv <= hlt; xpc <= '-'; nxpc <= (others => '-');

nxwe <= '-'; nxmem <= '-'; xacc <= '-';

nxacc <= '-'; xcry <= '-'; nxcry <= '-';

Combinatorial Section: state machine

```
-- state dispatch
case dly is      -- “dly” is the state variable
when hlt =>
    if start = '1' then nxdly <= run; else nxdly <= hlt; end if;
    xacc <= '1';    nxacc <= '0';
    xpc <= '1';    nxpc <= (others => '0');    -- keep PC reset
    xcry <= '1';   nxcry <= '0';    nxwe <= '0';
when run =>
case inst is    -- op-code dispatch
when opMSC =>
    case ir(5 downto 4) is when opHLT =>        nxdly <= hlt;
```

Combinatorial Section: misc. instructions

```
when opAnC => nxdly <= run;
```

```
case ir(3 downto 0) is
```

```
when "0000" => xacc <= '1'; nxacc <= '0'; xcry <= '1'; nxcry <= '0'; -- A,C = 0,0
```

```
when "1001" => xacc <= '1'; nxacc <= not acc; xcry <= '0'; -- A = not A
```

```
when "1100" => xacc <= '0'; xcry <= '1'; nxcry <= acc AND cry; -- C = A & C
```

```
when "1101" => xacc <= '1'; nxacc <= cry; xcry <= '0'; -- A = C
```

```
end case;
```

```
xpc <= '1'; nxpc <= pc + 1; nxwe <= '0';
```

```
when others => null;
```

```
end case;
```

Combinatorial Section: normal instructions

when opST =>

nxdly <= run;

xacc <= '0';

xpc <= '1'; nxpc <= pc + 1;

xcry <= '0';

nxwe <= '1'; nxmem<= acc;

when opADC =>

nxdly <= run; xpc <= '1'; nxpc <= pc + 1; nxwe <= '0';

xacc <= '1'; nxacc <= acc xor memrd xor cry;

xcry <= '1'; nxcry <= (acc and cry) or (acc and memrd) or
(cry and memrd);

Assembler: op-code functions

```
int[] mem = new int[2048];  
//opcodes write binary to instruction buffer  
void HLT ()      {mem[IP++] = 0; }  
void NA ()      {mem[IP++] = 0x19; }  
void AND2C ()   {mem[IP++] = 0x1c; }  
void C2A ()     {mem[IP++] = 0x1d; }  
  
void ST(int x)  {mem[IP++] = 0x040 | x; }  
void ADC(int x) {mem[IP++] = 0x1c0 | x; }
```

Assembler: 24-bit counter program

```
void toggle() // toggle Accum & Carry
  {CACC(); SASC(); HLT();}

void INC(int loc) // inc memory bit "macro"
  {ADC(loc); ST(loc); CA();}

void CNTR24() // 24 bit inc counter "macro"
  {CASC(); for(int i=23; i>0; i--) INC(i);
  ADC(0); ST(0);}

void cntr24() {CNTR24(); HLT();} // 24 bit
incrementing counter program
```

Assembler: Hello World program

```
void HELlo_UJorld() // sliding 7-segment "hello
world" with w via reversed & forward "j"
{
CNTR24 (); // lsb at location 23, msb at location 0

// add segment position ("00"+14..15) to counter
position (4..7)
const int LT0=56, LT1=57, LT2=58, LT3=59, CT14=14,
CT15=15, CT7=5, CT6=4, CT5=3, CT4=2;

CACC (); LD (CT15); ADC (CT7); ST (LT0);
LD (CT14); ADC (CT6); ST (LT1);
CA (); ADC (CT5); ST (LT2);
CA (); ADC (CT4); ST (LT3);
```


Assembler: Hello World program

```
// digit select decode, from 14..15, active low  
const int DIG3 = 60, DIG2 = 61, DIG1 = 62, DIG0 = 63,  
BT0 = 15, BT1 = 14;
```

```
LD (BT1) ; OR (BT0) ; ST (DIG0) ;  
LDC (BT1) ; OR (BT0) ; ST (DIG2) ;  
XOR (BT1) ; ST (DIG3) ;  
XOR (BT0) ; ST (DIG1) ;
```

Assembler: Hello World program

```
// segment logic, segments: 0:top, 1:top right,  
2:bottom right; 3:bottom, 4:bottom left, 5:top left,  
6:middle, 7:decimal point  
const int SEG0=55, SEG1=54, SEG2=53, SEG3=52,  
SEG4=51, SEG5=50, SEG6=49, DP=48;  
const int A0=LT3, B0=LT2, C0=LT1, D0=LT0;  
  
//LD(LT0); OR(LT1); OR(LT2); OR(LT3); ST(DP); //  
rotating decimal point  
SA(); ST(DP); // no decimal point
```

Assembler: Hello World program

```
// rotating HELLO UJOrLd
const int AB=47, AD=46, BD=45, NCD=44, BNCD=43,
NANB=42, CND=41, t=40;
// common factor sub-expression evaluation
LD(A0); AND(B0); ST(AB); //ab
LD(A0); AND(D0); ST(AD); //ad
LD(B0); AND(D0); ST(BD); //bd
LDC(C0); AND(D0); ST(NCD); //ncd
AND(B0); ST(BNCD); // & bncd
LD(A0); OR(B0); NA(); ST(NANB); //nanb
LDC(D0); AND(C0); ST(CND); //cnd
```

Assembler: Hello World program

```
LDC (D0) ; AND (NANB) ; OR (C0) ; OR (BD) ; OR (AD) ; OR (AB) ;  
ST (SEG0) ; //a' b' d' +c+bd+ad+ab
```

```
LD (NANB) ; AND (D0) ; OR (NCD) ; OR (CND) ; OR (AB) ;  
ST (SEG1) ; // c' d+cd' +ab+a' b' d
```

```
LD (NANB) ; AND (C0) ; ST (t) ; LDC (B0) ; AND (CND) ; OR (t) ;  
OR (NCD) ; OR (AB) ; ST (SEG2) ; //c' d+ab+a' b' c+b' cd'
```

```
LD (C0) ; OR (D0) ; NA () ; AND (NANB) ; ST (t) ; LD (A0) ;  
AND (NCD) ; OR (t) ; OR (BNCD) ; OR (AB) ; ST (SEG3) ;//  
a' b' c' d' +bc' d+ac' d+ab
```

```
LD (BNCD) ; OR (AB) ; ST (SEG4) ; // bc' d+ab  
LD (BD) ; OR (AD) ; OR (AB) ; ST (SEG5) ; // bd+ad+ab  
LDC (A0) ; AND (C0) ; ST (t) ; LDC (D0) ; AND (A0) ; OR (B0) ;  
OR (t) ; ST (SEG6) ; //a' c+b+ad' HLT () ;
```

Results:

No pipelines

And 50+ Mhz (the 166 instructions in “hello world” in 3.32us)

Design will run at 121 Mhz (need to do a clock generator)

Which uses a “slow” speed grade Spartan-3

100-200 Mhz ? for Virtex-4 & Stratix-2

Instruction set easily edited

CASE construct

Each instruction entered separately

Enables for register updates

And then optimized by VHDL compiler

Altera does not have distributed RAM

Requires a 2nd clock phase to emulate asynchronous read

After FPGA Hello World

To get a realistic micro-controller:

- Expand instruction set

- Expand instruction word size

- Expand memory word size

- Can add dual port distributed RAM (for MIPS & RISC)

Design Process Supports:

- Classical accumulator designs

- Register to register designs

- Stack machines

- De-coupled pipelines

- High levels of parallelism:

 - 24 u-controllers on XC3S200 (via 12 shared block RAMs)

After FPGA Hello World cont'd

LEM1_9MIN expansion via unused op-code space:

- Repeat loops

 - Hold loop instructions in shift registers

 - Auto-increment memory addresses each loop

- Branches & calls using a PC stack

- Lookup table instructions with tables in program memory

 - Requires 2nd address register and block RAM address mux

“Hello World” should be implemented as

- 24-bit memory increment

- 4-bit add

- Digit-select table lookup (four entries of 4-bits each)

- Seven-segment table lookup (16 entries of 7-bits each)

After FPGA Hello World cont'd

Uses:

Interrupt controllers, one for each interrupt

Parallel processing research

Motor/motion control:

At 50M instructions/sec one can do

5K instructions 10K times per second.

Other tools:

JHDL: unified environment & simulation

Confluence: 3X denser than VHDL or Verilog

Custom microprocessor environments: Tensilica, etc.

TBD: new class of FPGAs for processor design