Digital Design Lab
EEN 316

Project #2
MAC

Group 4
Nathan Paternoster
(Partners: Andrew O'neil-Smith, Garrett Clausen)

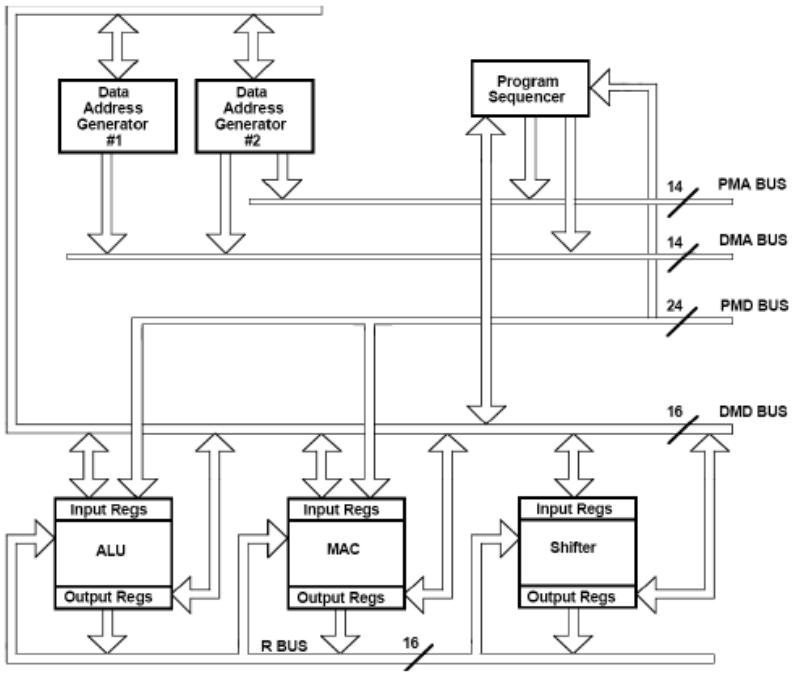
Sayan Maity, TA

University of Miami

4/21/14

Abstract

In this project we build a multiplier accumulator (MAC) circuit. This circuit computes the product of two numbers and adds that product to an accumulator. This operation is called the MAC operation and the piece of hardware that performs this operation is called a MAC unit. This unit is particularly useful in digital signal processes by being able to perform convolution and the FFT.



An example of DSP architecture, including the MAC unit

In addition, the MAC unit is also capable of multiplication with cumulative subtraction and single-cycle multiply. It can work in both integer and fraction modes. The MAC unit contains an extra 8-bits of overflow protection at its output.

Table of Contents

Overview

The design of this MAC unit will begin with the design of its sub-components. The components we will need are:

- 16-bit register, 8-bit register
- 16-bit 2-to-1 MUX, 8-bit 2-to-1 MUX, 16-bit 3-to-1 MUX
- 16-bit tri-state buffer, 8-bit tri-state buffer
- 16-bit multiplier
- 40-bit add/subtract unit

The necessity for 8-bit components comes from the MAC's overflow protection. The final result of the MAC computation will be placed in two 16-bit output registers. A third 8-bit output register will also be provided for the overflow protection. This register and its corresponding MUX will require us to design 8-bit sub-components.

With the components designed, the next step of design will be to construct the top-level, structural entity. This entity will consist of two input paths, X and Y. Each input path will consist of two registers controlled by a single MUX. This allows multiple values to be stored in the X and Y inputs. The inputs will be put into the multiplier which will perform the multiplication and output the product. This product will be fed into the add/subtract unit along with 40 bits from the output. This feedback allows for the accumulation.

After design of the circuit is completed, we will simulate it in ModelSim to verify our code. Once the simulation successfully completes we will write a DO script to save our simulation results and then transfer the design into Quartus. In Quartus we will download our circuit into the DE-2 digital logic board. We will perform a simulation of the two main components (multiplier and add/subtract unit) on the board to visualize our results.

Objectives

To understand the function and implementation of the MAC unit and how to design it. To understand the MAC's practical applications and full capabilities.

Description

The first step to designing the MAC unit is to design the sub-components. The registers, multiplexers, and tri-state buffers will all be designed using the algorithmic behavioral method. The register is the only sequential element, so it will be dependent on the clock. The MUX and TSB will not be. The register will take a load signal that, when set to '1', will store its input to its internal storage signal on the rising edge of the clock. On the falling edge of the clock it will output its storage to its output. The MUX will take a specified select line (in the case of the single 3-to-1 MUX, it will take two input select bits). The TSB will take a specified enable line.

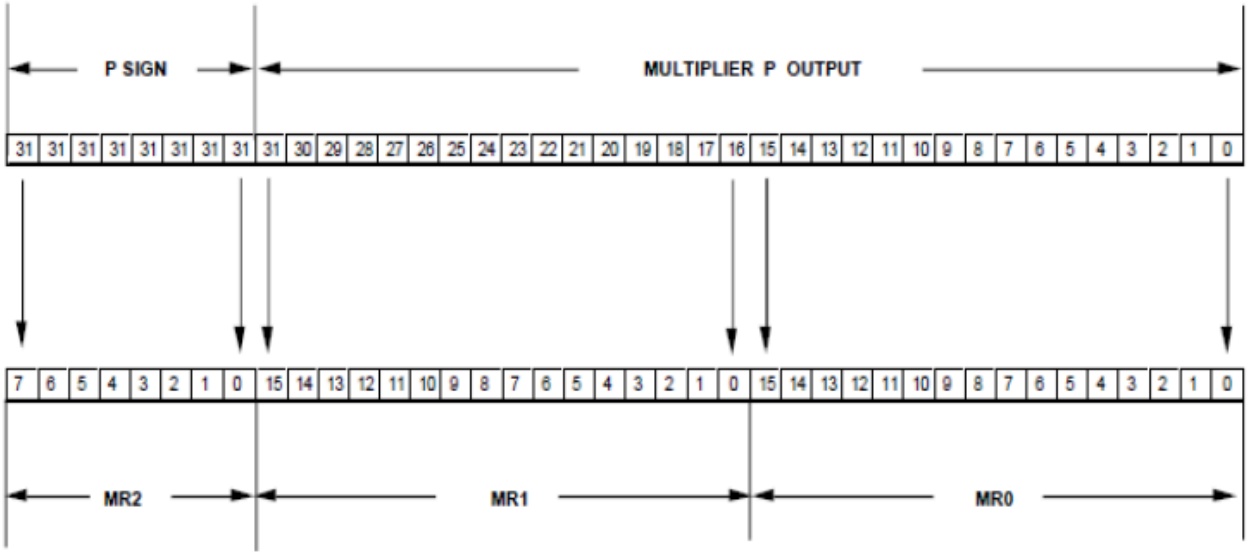
Once the sub-components have been designed, we will design the two main components – the multiplier and add/subtract unit. The multiplier will be a simple component taking two 16-bit inputs and outputting one 32-bit output. It will be designed in the behavioral method and will make use of internal integer signals to perform the multiplication. The inputs will be converted into integers, multiplied, and the product converted back into the std_logic_vector.

The add/subtract unit will also be designed behaviorally. Its inputs are a 32-bit input from the multiplier, a 40-bit input from the output, and a 5-bit operation-code to determine the function to be performed. It will output its result into two 16-bit result vectors, one 8-bit sign-extension result vector (the overflow protection), and an overflow bit. First, the most significant bit of the second result vector will be copied into all 8-bits of the sign-extension vector.

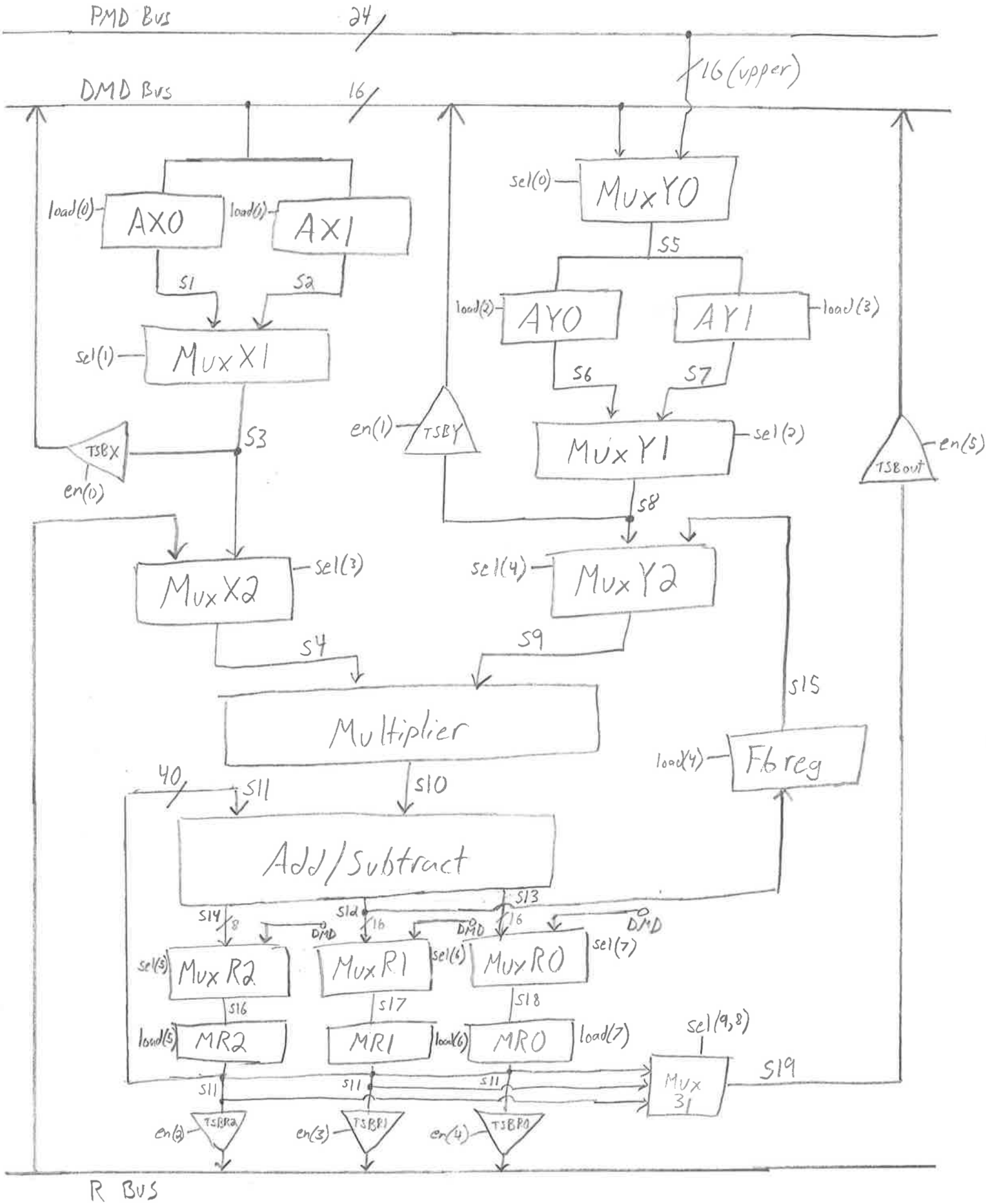
This provides overflow protection by providing a way to set the overflow bit. The signed bit (most significant bit) of the result will be compared to the 8-bits of the sign-extended vector. If they are not equal to each other then the overflow bit will be set to 1. Second, the operation will be performed depending on the op-code. Third, the result will be placed into the appropriate output vectors. Finally, the MSB of the result will be compared to sign-extended vector and the overflow bit will be set if necessary.

The final step to designing the MAC circuit is to create a top-level, structural entity that will map all of the previous components into the entire circuit. There will need to be numerous internal signals to accomplish all of the interconnections.

Design Synthesis



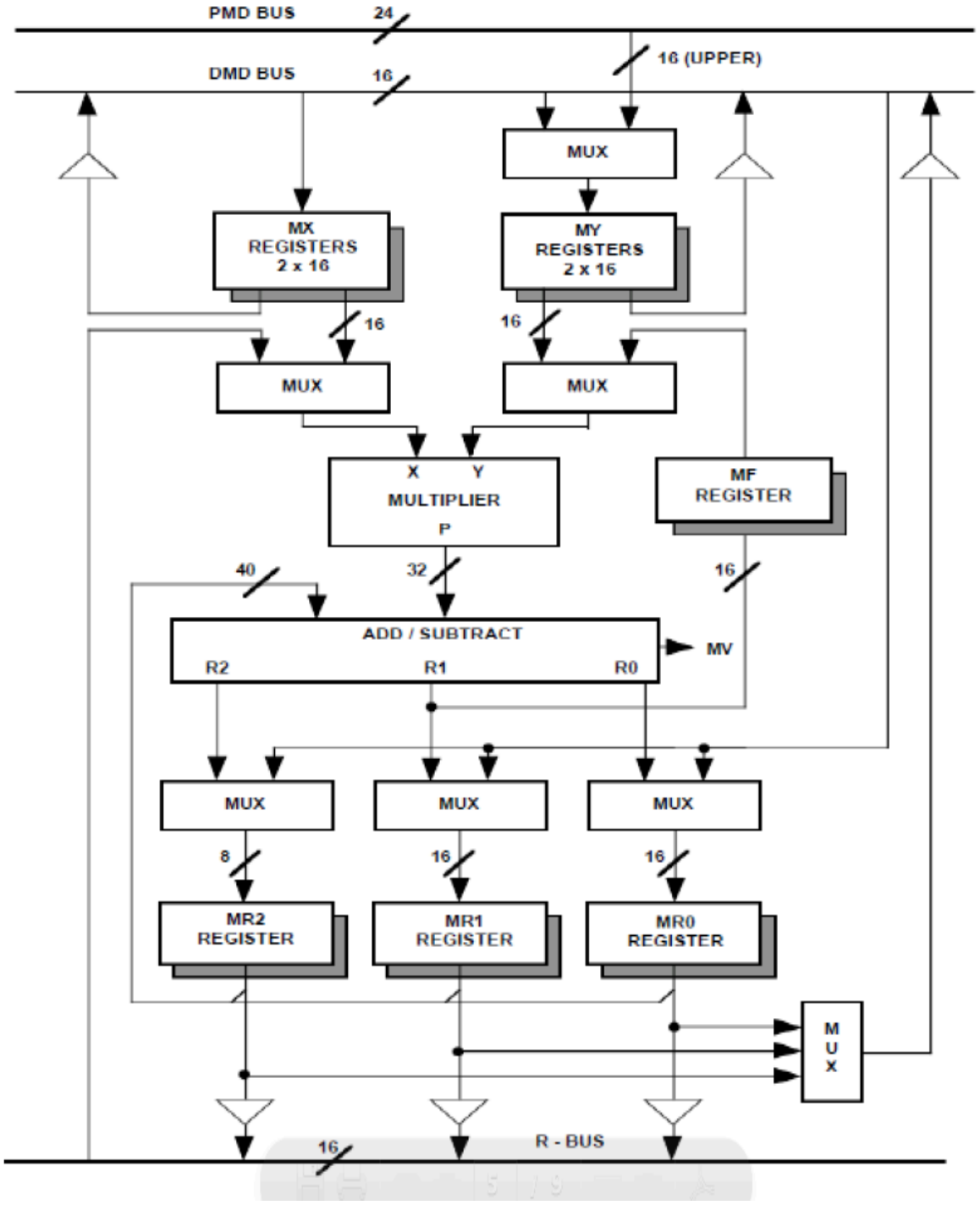
Format of the input to the add/subtract unit (P) and its delegation to the three result registers


PMD Bus
24
DMD Bus
16
16 (upper)
load(0)
AX0
load(1)
AX1
S1
S2
sel(1)
MuxX1
S3
TSBX
en(0)
sel(0)
MuxY0
S5
load(2)
AY0
AY1
load(3)
S6
S7
en(1)
TSBY
MuxY1
sel(2)
TSBout
en(5)
S8
sel(3)
MuxX2
sel(4)
MuxY2
S4
S9
S15
Multiplier
load(4)
Fb reg
40
S11
S10
Add/Subtract
S14
8
S12
16
S13
16
DMD
DMD
DMD
sel(5)
MuxR2
MuxR1
sel(6)
MuxR0
sel(7)
S16
S17
S18
load(5)
MR2
MR1
load(6)
MR0
load(7)
sel(9,8)
S11
S11
S11
Mux 31
S19
en(2)
TSBR2
en(3)
TSBR1
en(4)
TSBR0
R Bus

The previous page shows a diagram of the entire top-level circuit including the internal signal specifications. Below is a list of the operations that our MAC unit will support.

AMF	Operation
0 0 0 0 0	Clear result (MR) to zero
0 0 0 0 1	X * Y
0 0 0 1 0	MR + X * Y
0 1 1 0 0	MR – X * Y

Complete Logic Diagram



Code

Top-Level MAC entity

```
library ieee;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

entity mac_top is
	port(DMD, R : inout std_logic_vector(15 downto 0);
		PMD  : in  std_logic_vector(23 downto 0);
		load : in  std_logic_vector(7 downto 0);
		sel  : in  std_logic_vector(11 downto 0);
		en   : in  std_logic_vector(5 downto 0);
		opc  : in  std_logic_vector(4 downto 0);
		clk  : in  std_logic;
		MV   : out std_logic);
end entity;

architecture structural of mac_top is
	component sixteenbitMUXtwo is
		port(inp1, inp2 : in std_logic_vector(15 downto 0);
			sel : in std_logic;
			 outp : out std_logic_vector(15 downto 0));
	end component;
	component add_sub is
		port(mr : in std_logic_vector(39 downto 0);
			p : in std_logic_vector (31 downto 0);
			amf : in std_logic_vector (4 downto 0);
			r0, r1 : out std_logic_vector(15 downto 0);
			r2 : out std_logic_vector(7 downto 0);
			mv : out std_logic);
	end component;
	component MACtsb is
		port(data : in std_logic_vector(15 downto 0);
			ctr : in std_logic;
			outp : out std_logic_vector(15 downto 0));
	end component;
	component MACregistersixteen is
		port(inp : in  std_logic_vector(15 downto 0);
			load : in std_logic;
			clk : in std_logic;
			outp : out std_logic_vector(15 downto 0));
	end component;
	component sixteenbitMUXthree is
		port(inp1, inp2 : in std_logic_vector(15 downto 0);
			inp3 : in std_logic_vector(7 downto 0);
			sel : in std_logic_vector(1 downto 0);
			outp : out std_logic_vector(15 downto 0));
	end component;
```

```
	component MACtsbeight is
		port(data : in std_logic_vector(7 downto 0);
			ctr : in std_logic;
			outp : out std_logic_vector(15 downto 0));
	end component;
	component MACregistereight is
		port(inp : in  std_logic_vector(7 downto 0);
			load : in std_logic;
			clk : in std_logic;
			outp : out std_logic_vector(7 downto 0));
	end component;
	component mult is
		port(x, y : in std_logic_vector(15 downto 0);
			p : out std_logic_vector(31 downto 0));
	end component;
	component eightbitMUXtwo is
		port(inp1, inp2 : in std_logic_vector(7 downto 0);
			sel : in std_logic;
			outp : out std_logic_vector(7 downto 0));
	end component;

	signal s1, s2, s3, s4, s5, s6, s7, s8, s9, s12, s13, s15, s17, s18, s19 : std_logic_vector(15 downto 0);
	signal s10 : std_logic_vector(31 downto 0);
	signal s11 : std_logic_vector(39 downto 0);
	signal s14, s16 : std_logic_vector(7 downto 0);

begin
	-- X
	ax0   : MACregistersixteen port map (DMD, load(0) ,clk, s1);
	ax1   : MACregistersixteen port map (DMD, load(1) ,clk, s2);
	muxx1 : sixteenbitMUXtwo   port map (s1, s2, sel(1), s3);
	tsbx  : MACtsb             port map (s3, en(0), DMD);
	muxx2 : sixteenbitMUXtwo   port map (R, s3, sel(3), s4);

	-- Y
	muxy0 : sixteenbitMUXtwo   port map (DMD, PMD(23 downto 8), sel(0), s5);
	ay0   : MACregistersixteen port map (s5, load(2), clk, s6);
	ay1   : MACregistersixteen port map (s5, load(3), clk, s7);
	muxy1 : sixteenbitMUXtwo   port map (s6, s7, sel(2), s8);
	tsby  : MACtsb             port map (s8, en(1), DMD);
	muxy2 : sixteenbitMUXtwo   port map (s8, s15, sel(4), s9);

	-- multiplier
	mult1  : mult    port map (s4, s9, s10);
	addsub : add_sub port map (s11, s10, opc, s13, s12, s14, MV);

	-- feedback register
	fbreg : MACregistersixteen port map (s12, load(4), clk, s15);

	-- output MUXs
	muxr2 : eightbitMUXtwo   port map (s14, DMD(7 downto 0), sel(5), s16);
	muxr1 : sixteenbitMUXtwo port map (s12, DMD, sel(6), s17);
	muxr0 : sixteenbitMUXtwo port map (s13, DMD, sel(7), s18);
```

```
	-- output registers
	mr2 : MACregistereight   port map (s16, load(5), clk, s11(39 downto 32));
	mr1 : MACregistersixteen port map (s17, load(6), clk, s11(31 downto 16));
	mr0 : MACregistersixteen port map (s18, load(7), clk, s11(15 downto 0));

	-- output buffers
	tsbr2 : MACtsbeight port map (s11(39 downto 32), en(2), R);
	tsbr1 : MACtsb      port map (s11(31 downto 16), en(3), R);
	tsbr0 : MACtsb      port map (s11(15 downto 0), en(4), R);

	-- output feedback MUX
	mux31  : sixteenbitMUXthree port map (s11(31 downto 16), s11(15 downto 0), s11(39 downto 32), sel(9 downto
8), s19);
	 tsbout : MACtsb           port map (s19, en(5), DMD);

end architecture;
```

Multiplier Component

```
library ieee;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

entity mult is
	port(x, y : in std_logic_vector(15 downto 0);
		p : out std_logic_vector(31 downto 0));
end entity;

architecture behavioral of mult is
	signal int1, int2 : integer;
begin
	p <= conv_std_logic_vector(conv_integer(x)*conv_integer(y), 32);
end architecture;
```

Add/Subtract Unit Component

```
library ieee;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_1164.all;

entity add_sub is
	port(mr : in std_logic_vector(39 downto 0);
		p : in std_logic_vector (31 downto 0);
		amf : in std_logic_vector (4 downto 0);
		r0, r1 : out std_logic_vector(15 downto 0);
```

```
                r2 : out std_logic_vector(7 downto 0);
                mv : out std_logic);
end entity;

architecture behavioral of add_sub is
begin
        process(mr, p, amf)
                variable p1 : std_logic_vector(39 downto 0);                -- internal variables used to
                variable temp : std_logic_vector(39 downto 0);  -- accomplish the sign extension
        begin
                p1(39) := p(31);            -- sign extension
                p1(38) := p(31);
                p1(37) := p(31);
                p1(36) := p(31);
                p1(35) := p(31);
                p1(34) := p(31);
                p1(33) := p(31);
                p1(32) := p(31);
                p1(31 downto 0) := p(31 downto 0);
                if (amf = "00000") then                         -- performing the operations
                        temp := "0000000000000000000000000000000000000000";
                elsif (amf = "00001") then
                        temp := p1;
                elsif (amf = "00010") then
                        temp := mr + p1;
                elsif (amf = "01100") then
                        temp := mr-p1;
                else
                        temp := "ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ";
                end if;

                r0 <= temp(15 downto 0);                    -- setting the output registers
                r1 <= temp(31 downto 16);
                r2 <= temp(39 downto 32);

                if (p1(39 downto 32) /= temp(39 downto 32)) then                    -- setting overflow
                        mv <= '1';
                else
                        mv <= '0';
                end if;
        end process;
end architecture;
```

16-bit Register Sub-Component

```
library ieee;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

entity MACregistersixteen is
	port(inp : in  std_logic_vector(15 downto 0);
		load : in std_logic;
		clk : in std_logic;
		outp : out std_logic_vector(15 downto 0));
end entity;

architecture behavioral of MACregistersixteen is
	signal store : std_logic_vector(15 downto 0);
begin
	fourbitprocess: process(clk)
	begin
		if (clk'event and clk = '1' and load = '1') then
			store <= inp;
		end if;
		if (clk'event and clk = '0') then
			outp <= store;
		end if;
	end process;
end architecture;
```

8-bit Register Sub-Component

```
library ieee;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

entity MACregistereight is
	port(inp : in  std_logic_vector(7 downto 0);
		load : in std_logic;
		clk : in std_logic;
		outp : out std_logic_vector(7 downto 0));
end entity;

architecture behavioral of MACregistereight is
	signal store : std_logic_vector(7 downto 0);
begin
	fourbitprocess: process(clk)
```

```
	begin
		if (clk'event and clk = '1' and load = '1') then
			store <= inp;
		end if;
		if (clk'event and clk = '0') then
			outp <= store;
		end if;
	end process;
end architecture;
```

16-bit 2-to-1 MUX Sub-Component

```
library ieee;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

entity sixteenbitMUXtwo is
	port(inp1, inp2 : in std_logic_vector(15 downto 0);
		sel : in std_logic;
		outp : out std_logic_vector(15 downto 0));
end entity;

architecture behavioral of sixteenbitMUXtwo is
begin
	process(inp1, inp2, sel)
	begin
		if (sel = '1') then
			outp <= inp2;
		elsif (sel = '0') then
			outp <= inp1;
		end if;
	end process;
end architecture;
```

8-bit 2-to-1 MUX Sub-Component

```
library ieee;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

entity eightbitMUXtwo is
	port(inp1, inp2 : in std_logic_vector(7 downto 0);
		sel : in std_logic;
```

```
		outp : out std_logic_vector(7 downto 0));
end entity;

architecture behavioral of eightbitMUXtwo is
begin
	process(inp1, inp2, sel)
	begin
		if (sel = '1') then
			outp <= inp2;
		elsif (sel = '0') then
			outp <= inp1;
		end if;
	end process;
end architecture;
```

16-bit 3-to-1 MUX Sub-Component

```
library ieee;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

entity sixteenbitMUXthree is
	port(inp1, inp2 : in std_logic_vector(15 downto 0);
		inp3 : in std_logic_vector(7 downto 0);
		sel : in std_logic_vector(1 downto 0);
		outp : out std_logic_vector(15 downto 0));
end entity;

architecture behavioral of sixteenbitMUXthree is
begin
	process(inp1, inp2, inp3, sel)
	begin
		if (sel = "10") then
			outp(15 downto 8) <= "00000000";
			outp(7 downto 0) <= inp3;
		elsif (sel = "01") then
			outp <= inp2;
		elsif (sel = "00") then
			outp <= inp1;
		end if;
	end process;
end architecture;
```

16-bit TSB Sub-Component

```
library ieee;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

entity MACtsb is
	port(data : in std_logic_vector(15 downto 0);
		ctr : in std_logic;
		outp : out std_logic_vector(15 downto 0));
end entity;

architecture behavioral of MACtsb is
begin
	outp <= data when (ctr = '1') else "ZZZZZZZZZZZZZZZZ";
end architecture;
```

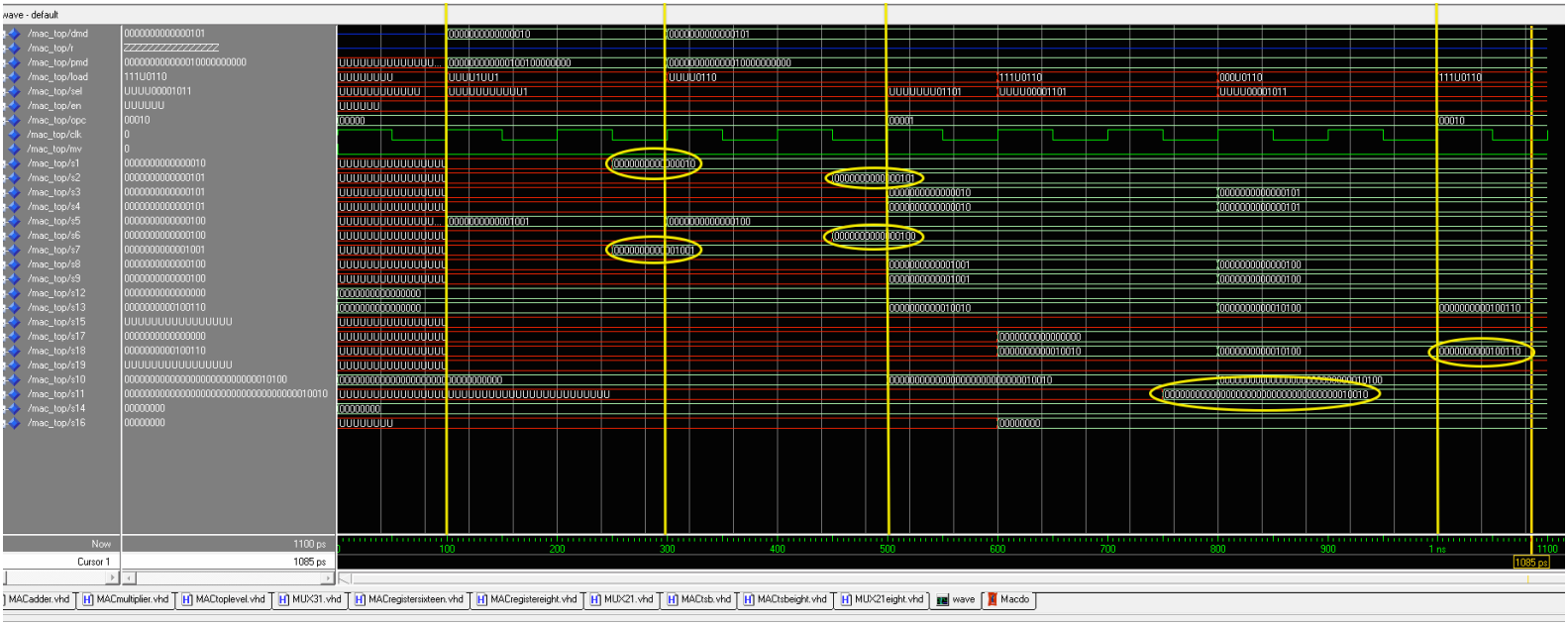
8-bit TSB Sub-Component

```
library ieee;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

entity MACtsbeight is
	port(data : in std_logic_vector(7 downto 0);
		ctr : in std_logic;
		outp : out std_logic_vector(15 downto 0));	-- output will be 16-bit
end entity;

architecture behavioral of MACtsbeight is
begin
	process(data, ctr)
	begin
		if (ctr = '1') then
			outp(15 downto 8) <= "00000000";		-- pad 0's onto the front of the input
			outp(7 downto 0) <= data;
		else
			outp <= "ZZZZZZZZZZZZZZZZ";
		end if;
	end process;
end architecture;
```

Results and Simulations

ModelSim

```
MR = 0;  (Clear MR to Zero)
MX0 = 2; MY1 = 9;
MX1 = 5; MY0 = 4;
MR = MX0 * MY1;
MR = MR + MX1 * MY0;
```



Phase 1: MR = 0

Phase 2: MX0 (s1) is loaded with 2, MY1 (s7) is loaded with 9

Phase 3: MX1 (s2) is loaded with 5, MY0 (s6) is loaded with 4

Phase 4: MR (s11) is loaded with MX0*MY1 = 18

Phase 5: MR (s13 here) is loaded with MR + MX1*MY0 = 36

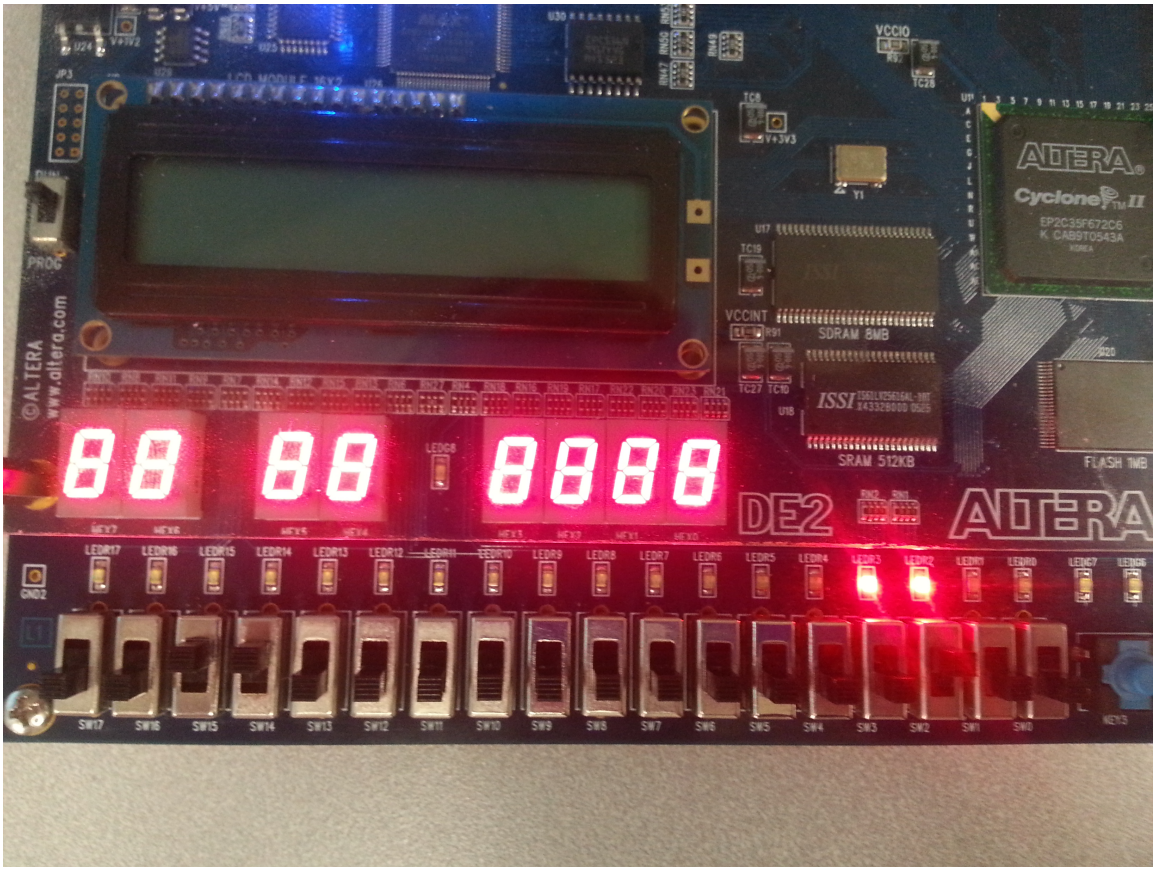
Do script

```
vsim work.mac_top(struct)
add wave sim:/mac_top/*

# clear MR to zero
force -freeze sim:/mac_top/clk 1 0, 0 {50 ps} -r 100
force -freeze sim:/mac_top/opc 00000 0
run

# load MX0 = 2 and MY1 = 9
force -freeze sim:/mac_top/dmd 0000000000000010 0
force -freeze sim:/mac_top/pmd 000000000000100100000000 0
force -freeze sim:/mac_top/load(0) 1 0
force -freeze sim:/mac_top/load(3) 1 0
force -freeze sim:/mac_top/sel(0) 1 0
run
run

# load MX1 = 5 and MY0 = 4
force -freeze sim:/mac_top/dmd 0000000000000101 0
force -freeze sim:/mac_top/pmd 000000000000010000000000 0
force -freeze sim:/mac_top/load(3) 0 0
force -freeze sim:/mac_top/load(0) 0 0
force -freeze sim:/mac_top/load(2) 1 0
force -freeze sim:/mac_top/load(1) 1 0
run
run

# multiply MR = MX0 * MY1
force -freeze sim:/mac_top/opc 00001 0
force -freeze sim:/mac_top/sel(1) 0 0
force -freeze sim:/mac_top/sel(2) 1 0
force -freeze sim:/mac_top/sel(3) 1 0
force -freeze sim:/mac_top/sel(4) 0 0
run

# multiply MR = MR + MX1 * MY0

# 1. Load S11 (first input to add/subtract component) with MR
force -freeze sim:/mac_top/sel(5) 0 0
force -freeze sim:/mac_top/sel(6) 0 0
force -freeze sim:/mac_top/sel(7) 0 0
force -freeze sim:/mac_top/load(5) 1 0
force -freeze sim:/mac_top/load(6) 1 0
force -freeze sim:/mac_top/load(7) 1 0
run
run
```

```
# 2. Perform the multiply and add
force -freeze sim:/mac_top/sel(1) 1 0
force -freeze sim:/mac_top/sel(2) 0 0
force -freeze sim:/mac_top/load(5) 0 0
force -freeze sim:/mac_top/load(6) 0 0
force -freeze sim:/mac_top/load(7) 0 0
run
run
force -freeze sim:/mac_top/opc 00010 0
force -freeze sim:/mac_top/load(5) 1 0
force -freeze sim:/mac_top/load(6) 1 0
force -freeze sim:/mac_top/load(7) 1 0
run
```

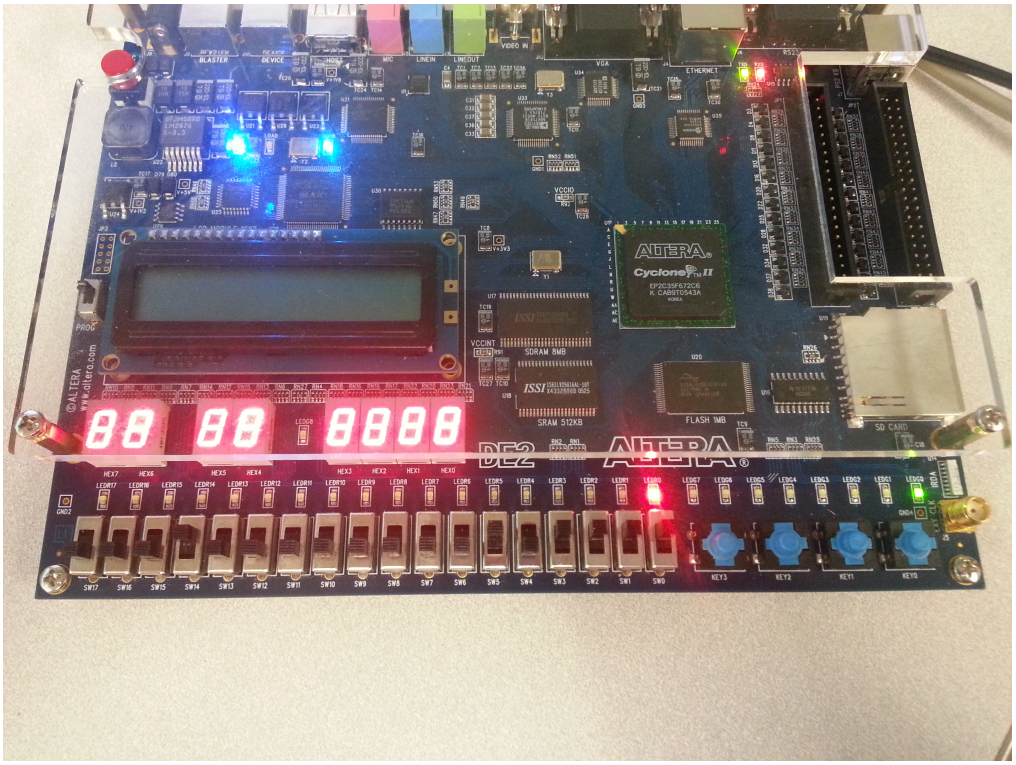
Quartus

Multiplier



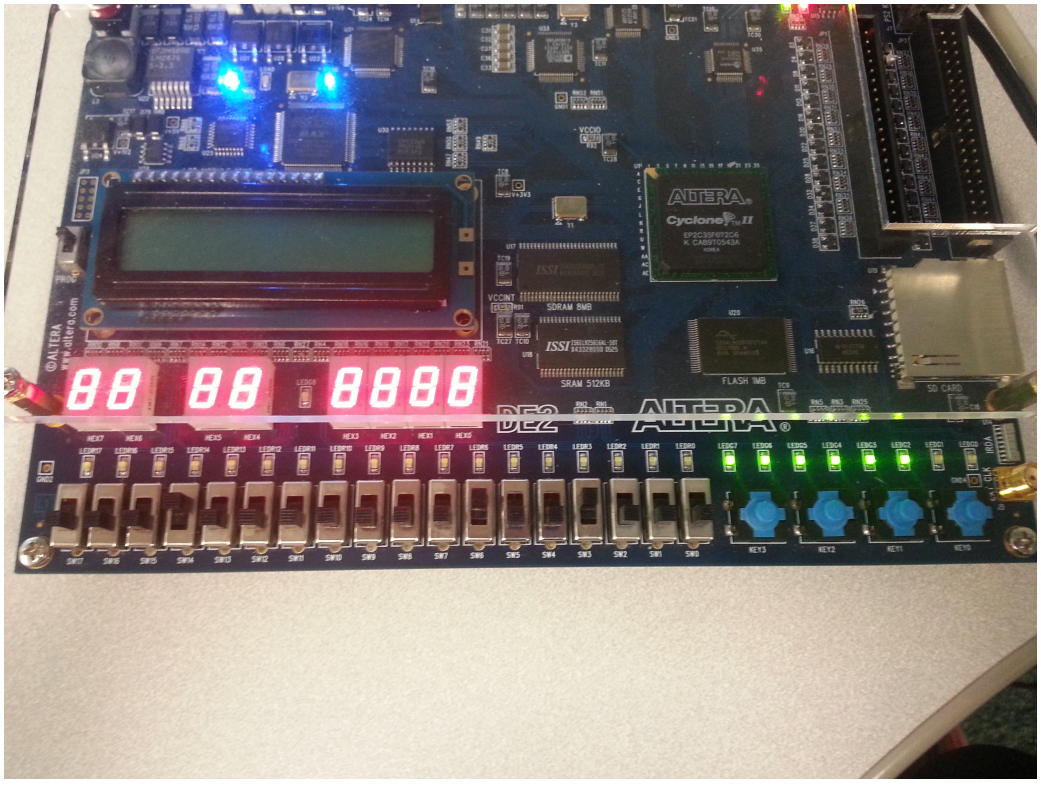
4 x 3 = 12 (LED)

Add/Subtract Unit



15 + 2 = 17 (red LED = overflow, green LED = result)

8 + 4 = 12 (right four green LEDs = result, left four green LEDs = sign extension)



Conclusion

We successfully designed the MAC unit to perform the multiply accumulate function. Our ModelSim simulation shows us the correct result from the example calculation. We had some difficulty downloading the circuit into the DE-2 board for visual simulation. We had to edit the code a little bit to scale down the input and output lengths from 16-bit and 40-bit to 4-bit and 8-bit respectively in order to fit the simulation on the board's limited controls. We also decided to only simulate the two main components (multiplier and add/subtract unit) separately on the board to verify that the components worked rather than the MAC as a whole entity.

Signed OFF

[Included]