Lab 6: Memory

Objective: To be introduced to Quartus II's Mega-Functions wizard and LPM modules, and to create a memory module using the wizard and learn how to use the memory module as part of other designs.

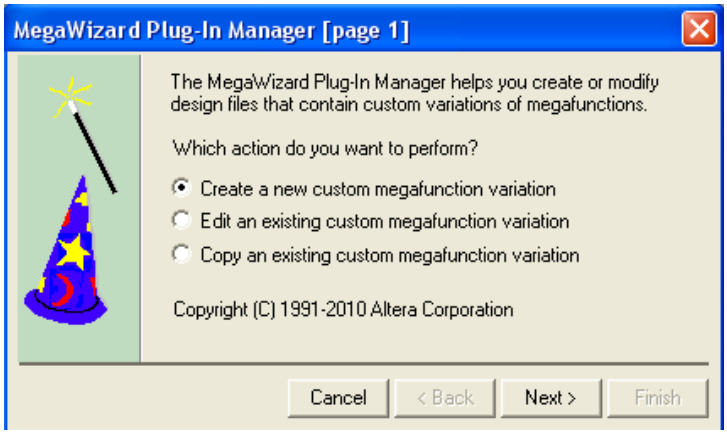
Part I

In computer systems it is necessary to provide a substantial amount of memory. If a system is implemented using FPGA technology it is possible to provide some amount of memory by using the memory resources that exist in the FPGA device.

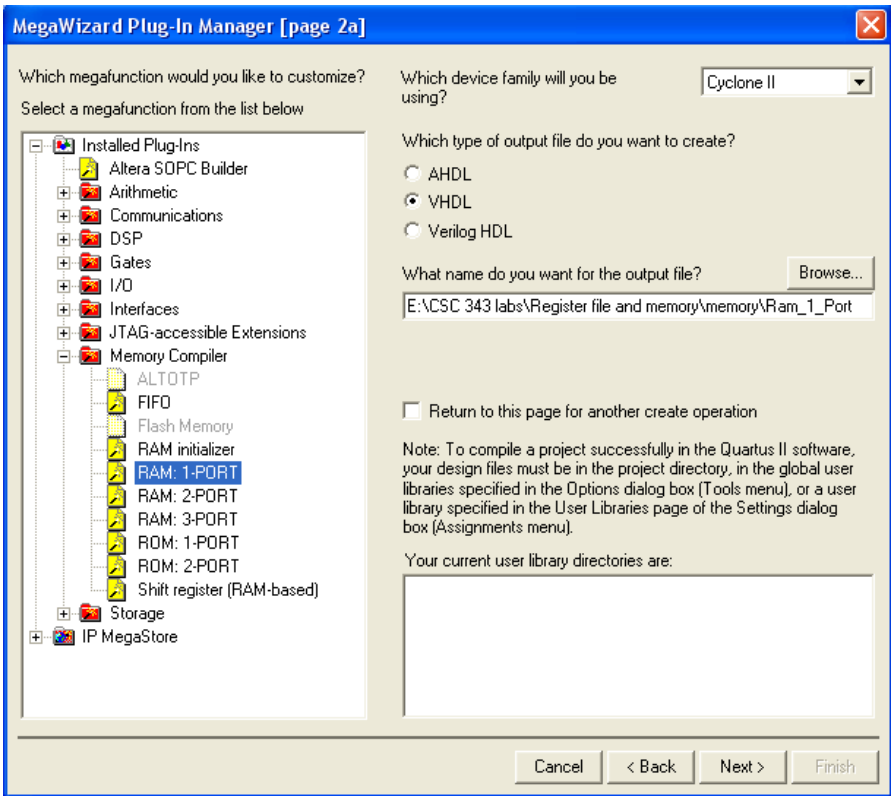
The Cyclone II 2C35 FPGA that is included on the DE2 board provides dedicated memory resources called *M4K blocks*. Each M4K block contains 4096 memory bits, which can be configured to implement memories of various sizes. Altera recommends that a RAM module be implemented by using the *RAM* LPMs. An LPM module is a "black box" representation of a larger circuit. In other words, you can only see the inputs and outputs of the circuit design. What is inside the box would represent *how* it works. However, when using an LPM module, *how* it works is not important. All that matters is that it *does* work.

Follow these steps to implement the LPM memory module.

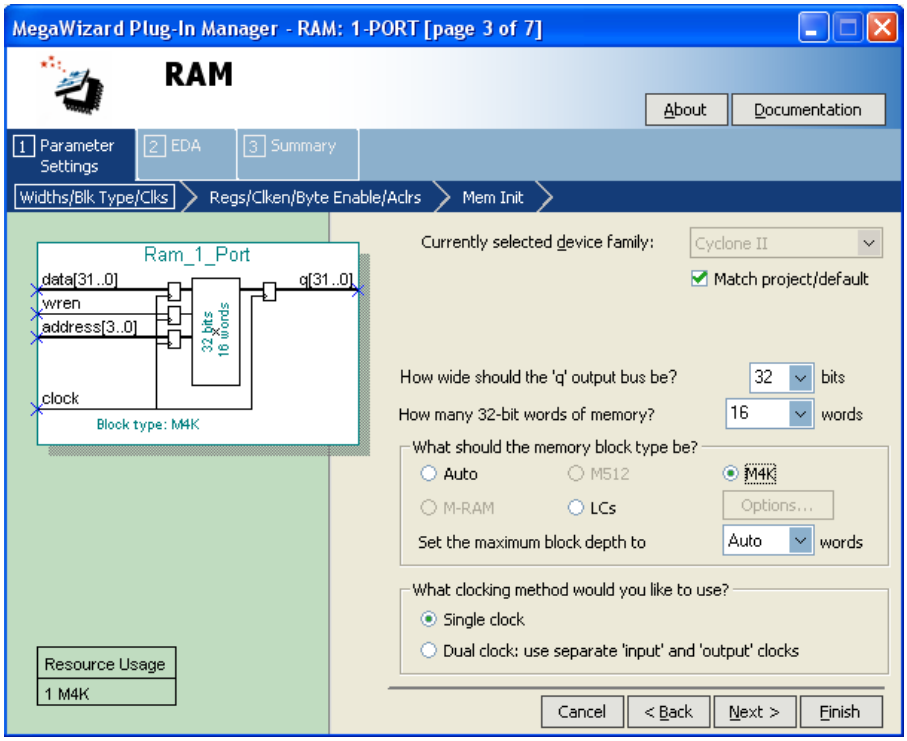
1- Open Quartus II.
2- Click on Tools -> MegaWizard Plug-In Manager.
3- Choose "Create a new custom megafunction variation" and press next



4- From the left side of the window choose "RAM: 1-PORT", set the device family to "Cyclone II", output type to VHDL and choose a name and location to save the output file, and then press Next.



5- On the next page, set the output bus to 32 bits and the number of words to 16 and memory block type to M4K and press “Next”.

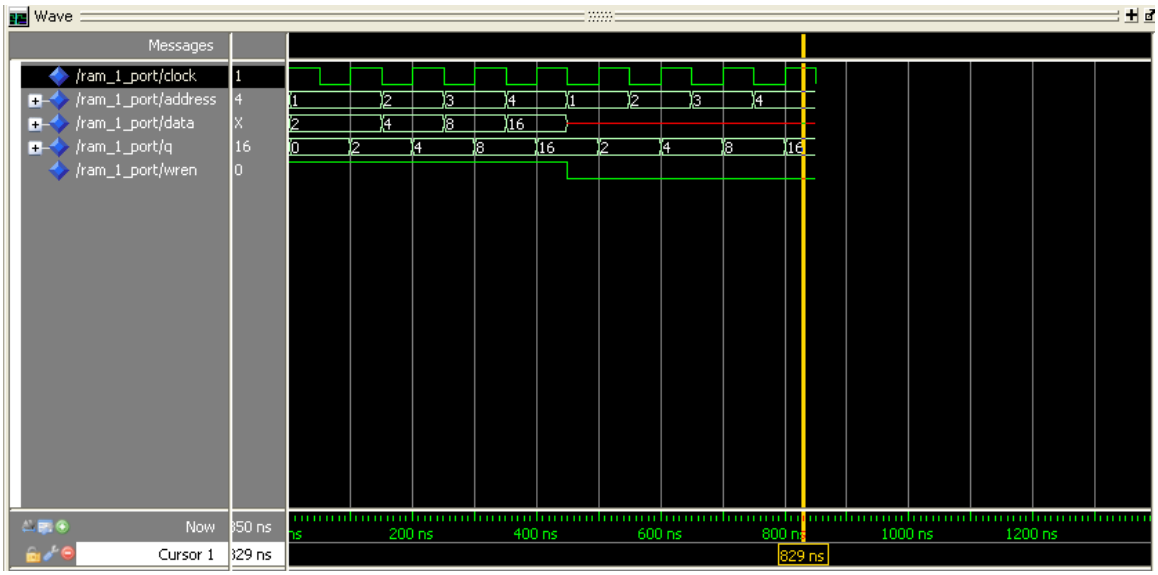


6- Uncheck the “q output port” and press finish twice.

7- The VHDL file is now stored in the folder you specified earlier. Now to verify and simulate your design in Modelsim, open Modelsim, change the directory and create a work library.

8- Now compile into the work library the VHDL file you have created using the Quartus II's "MegaWizard Plug-In Manager". (Note: You need Modelsim Altera edition to compile VHDL files you create using the wizard in Quartus, otherwise you need to go through some additional steps. If you are using Model's Modelsim, download the Modelsim Altera starter edition from Altera's website for free.)

9- Simulate the memory and verify that it works properly. Here is a sample timing diagram.

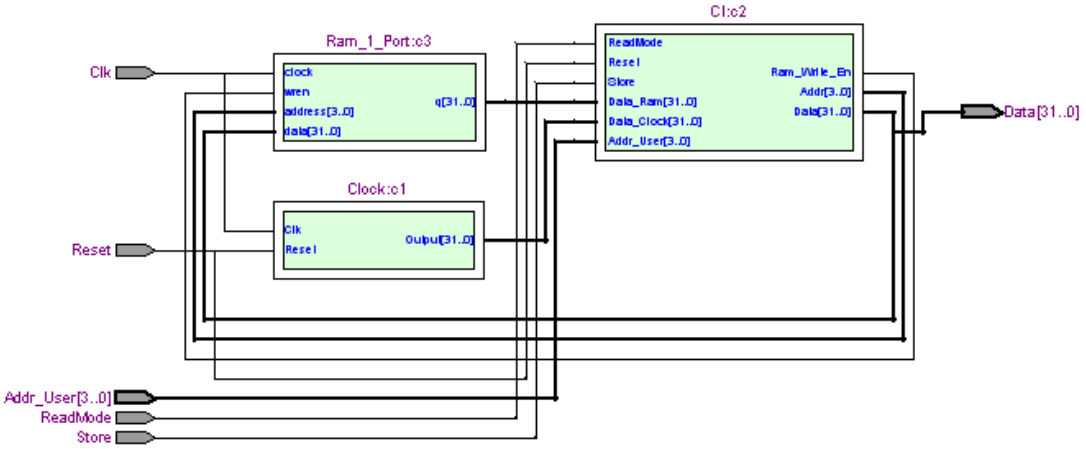


(To see numbers in decimal, right click on a signal and click on Radix -> Decimal).

Part II

In this part you are to design a stopwatch that can measure time to the ten thousandth of a second. The time must be shown on the seven segment displays on the board. The first two displays from left must show the minutes passed, the seconds two have to show the seconds past and the other four will be used to display time past to the ten thousandth of a second. The stopwatch has two modes, the watch mode and the read mode. In the watch mode, the displays must accurately show the time that has past since the stopwatch has been reset. In this mode by pressing a pushbutton for the first time, the time displayed at that moment should be stored in memory at address zero as a 32-bit string; however the timer will not stop. Now by pressing the pushbutton again the current time displayed should be stored in the next memory address, and so on.

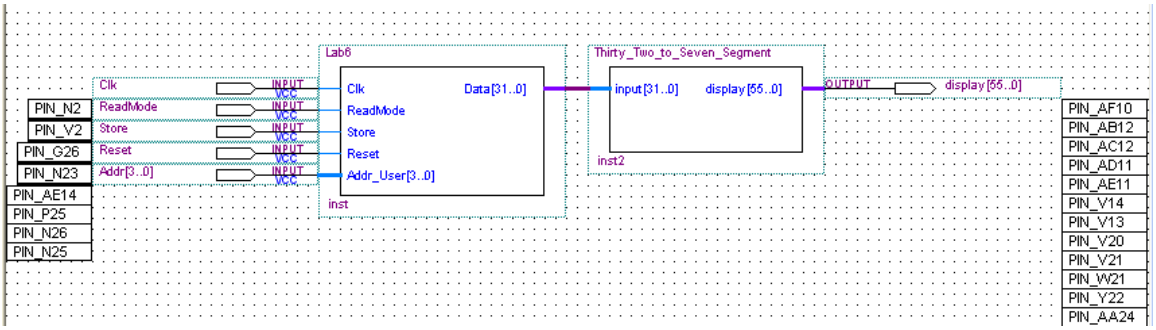
In read mode, switches 0 to 3 will be used to indicate the address where the data we want to display is stored. For example if we set the address to zero the exact time when we pressed the pushbutton for the first time should be displayed on the board.
You must use the memory unit from the previous part in your design. Here a suggested way to achieve this.



The first component is a timer. The timer is a counter that outputs a 32-bit string. The first and the second 4-bits of the string represent the minutes and so on. The second component and the brain of the design is the control unit. It manages the memory and the output of the design.

To test the stopwatch on the DE2 board you also need a unit that takes a 32-bit number and displays it on the seven segment displays. The code for this is available in the appendix along with the pin assignment file.

(Note: The pushbuttons on the DE2 board are inverted, in other words “Each switch provides a high logic level when it is not pressed, and provides a low logic level when depressed.” Have that in mind when writing the code for reset and start, since they are assigned to pushbuttons.)



Appendix

Thirty_Two_to_Seven_Segment.vhd

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.all;
USE IEEE.STD_LOGIC_ARITH.all;
USE IEEE.STD_LOGIC_UNSIGNED.all;

ENTITY Thirty_Two_to_Seven_Segment IS
PORT(
input : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
display : OUT std_logic_vector(55 downto 0)
);
END Thirty_Two_to_Seven_Segment;
ARCHITECTURE a OF Thirty_Two_to_Seven_Segment IS

component dec_7seg_hex
PORT(
hex_digit : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
en : in std_logic;
segment_a, segment_b, segment_c, segment_d, segment_e, segment_f,
segment_g : OUT std_logic);
END component;

signal en : std_logic := '1';
signal HEX0, HEX1, HEX2, HEX3, HEX4, HEX5, HEX6, HEX7 :
std_logic_vector(6 downto 0);
BEGIN

en <= '1';
display <= HEX7 & HEX6 & HEX5 & HEX4 & HEX3 & HEX2 & HEX1 & HEX0;


d0: dec_7seg_hex port map
(
hex_digit(0) => input(0),
hex_digit(1) => input(1),
hex_digit(2) => input(2),
hex_digit(3) => input(3),
en => en,
segment_a => HEX0(0),
segment_b => HEX0(1),
segment_c => HEX0(2),
segment_d => HEX0(3),
segment_e => HEX0(4),
segment_f => HEX0(5),
segment_g => HEX0(6)
);

d1:dec_7seg_hex port map
(
hex_digit(0) =>input(4),
```

```
hex_digit(1) =>input(5),
hex_digit(2) =>input(6),
hex_digit(3) =>input(7),
en => en,
segment_a => HEX1(0),
segment_b => HEX1(1),
segment_c => HEX1(2),
segment_d => HEX1(3),
segment_e => HEX1(4),
segment_f => HEX1(5),
segment_g  => HEX1(6)
);

d2:dec_7seg_hex port map
(
hex_digit(0) =>input(8),
hex_digit(1) =>input(9),
hex_digit(2) =>input(10),
hex_digit(3) =>input(11),
en => en,
segment_a => HEX2(0),
segment_b => HEX2(1),
segment_c => HEX2(2),
segment_d => HEX2(3),
segment_e => HEX2(4),
segment_f => HEX2(5),
segment_g  => HEX2(6)
);

d3:dec_7seg_hex port map
(
hex_digit(0) =>input(12),
hex_digit(1) =>input(13),
hex_digit(2) =>input(14),
hex_digit(3) =>input(15),
en => en,
segment_a => HEX3(0),
segment_b => HEX3(1),
segment_c => HEX3(2),
segment_d => HEX3(3),
segment_e => HEX3(4),
segment_f => HEX3(5),
segment_g => HEX3(6)
);

d4:dec_7seg_hex port map
(
hex_digit(0) =>input(16),
hex_digit(1) =>input(17),
hex_digit(2) =>input(18),
hex_digit(3) =>input(19),
en => en,
segment_a => HEX4(0),
segment_b => HEX4(1),
segment_c => HEX4(2),
segment_d => HEX4(3),
segment_e => HEX4(4),
```

```
segment_f => HEX4(5),
segment_g  => HEX4(6)
);

d5:dec_7seg_hex port map
(
hex_digit(0) =>input(20),
hex_digit(1) =>input(21),
hex_digit(2) =>input(22),
hex_digit(3) =>input(23),
en => en,
segment_a => HEX5(0),
segment_b => HEX5(1),
segment_c => HEX5(2),
segment_d => HEX5(3),
segment_e => HEX5(4),
segment_f => HEX5(5),
segment_g => HEX5(6)
);

d6:dec_7seg_hex port map
(
hex_digit(0) =>input(24),
hex_digit(1) =>input(25),
hex_digit(2) =>input(26),
hex_digit(3) =>input(27),
en => en,
segment_a => HEX6(0),
segment_b => HEX6(1),
segment_c => HEX6(2),
segment_d => HEX6(3),
segment_e => HEX6(4),
segment_f => HEX6(5),
segment_g => HEX6(6)
);

d7:dec_7seg_hex port map
(
hex_digit(0) =>input(28),
hex_digit(1) =>input(29),
hex_digit(2) =>input(30),
hex_digit(3) =>input(31),
en => en,
segment_a => HEX7(0),
segment_b => HEX7(1),
segment_c => HEX7(2),
segment_d => HEX7(3),
segment_e => HEX7(4),
segment_f => HEX7(5),
segment_g => HEX7(6)
);


END a;
```

dec_7seg_hex.vhd

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.all;
USE IEEE.STD_LOGIC_ARITH.all;
USE IEEE.STD_LOGIC_UNSIGNED.all;
-- Hexadecimal to 7 Segment Decoder for LED Display
ENTITY dec_7seg_hex IS
PORT(
hex_digit : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
en : in std_logic;
segment_a, segment_b, segment_c, segment_d, segment_e, segment_f,
segment_g : OUT std_logic);
END dec_7seg_hex;
ARCHITECTURE a OF dec_7seg_hex IS
SIGNAL segment_data : STD_LOGIC_VECTOR(6 DOWNTO 0);
BEGIN
PROCESS (Hex_digit)
-- HEX to 7 Segment Decoder for LED Display
BEGIN -- Hex-digit is the four bit binary value to display
if(en = '1') then
CASE Hex_digit IS
WHEN "0000" =>
segment_data <= "1111110";
WHEN "0001" =>
segment_data <= "0110000";
WHEN "0010" =>
segment_data <= "1101101";
WHEN "0011" =>
segment_data <= "1111001";
WHEN "0100" =>
segment_data <= "0110011";
WHEN "0101" =>
segment_data <= "1011011";
WHEN "0110" =>
segment_data <= "1011111";
When "0111" =>
segment_data <= "1110000";
When "1000" =>
segment_data <= "1111111";
When "1001" =>
segment_data <= "1111011";
When "1010"  =>
segment_data <= "1110111";
When "1011" =>
segment_data <= "0011111";
When "1100" =>
segment_data <= "1001110";
When "1101" =>
segment_data <= "0111101";
When "1110" =>
segment_data <= "1001111";
When "1111" =>
segment_data <= "1000111";
WHEN OTHERS =>
```

```
segment_data <= "0111110";
END CASE;
end if;
END PROCESS;
-- extract segment data bits and invert
-- LED driver circuit is inverted
segment_a
<=
NOT
segment_data(6);
segment_b
<=
NOT
segment_data(5);
segment_c
<=
NOT
segment_data(4);
segment_d
<=
NOT
segment_data(3);
segment_e
<=
NOT
segment_data(2);
segment_f
<=
NOT
segment_data(1);
segment_g
<=
NOT
segment_data(0);
END a;
```

Lab_6_Pin_Assignment.txt

```
To, Location
Store, PIN_G26
Reset, PIN_N23
Clk, PIN_N2
ReadMode, PIN_V2
Addr[3], PIN_AE14
Addr[2],PIN_P25
Addr[1],PIN_N26
Addr[0],PIN_N25
display[0], PIN_AF10
display[1], PIN_AB12
display[2], PIN_AC12
display[3], PIN_AD11
display[4], PIN_AE11
display[5], PIN_V14
```

display[6], PIN_V13
display[7], PIN_V20
display[8], PIN_V21
display[9], PIN_W21
display[10], PIN_Y22
display[11], PIN_AA24
display[12], PIN_AA23
display[13], PIN_AB24
display[14], PIN_AB23
display[15], PIN_V22
display[16], PIN_AC25
display[17], PIN_AC26
display[18], PIN_AB26
display[19], PIN_AB25
display[20], PIN_Y24
display[21], PIN_Y23
display[22], PIN_AA25
display[23], PIN_AA26
display[24], PIN_Y26
display[25], PIN_Y25
display[26], PIN_U22
display[27], PIN_W24
display[28], PIN_U9
display[29], PIN_U1
display[30], PIN_U2
display[31], PIN_T4
display[32], PIN_R7
display[33], PIN_R6
display[34], PIN_T3
display[35], PIN_T2
display[36], PIN_P6
display[37], PIN_P7
display[38], PIN_T9
display[39], PIN_R5
display[40], PIN_R4
display[41], PIN_R3
display[42], PIN_R2
display[43], PIN_P4
display[44], PIN_P3
display[45], PIN_M2
display[46], PIN_M3
display[47], PIN_M5
display[48], PIN_M4
display[49], PIN_L3
display[50], PIN_L2
display[51], PIN_L9
display[52], PIN_L6
display[53], PIN_L7
display[54], PIN_P9
display[55], PIN_N9