

Counter with ROM

Counter with ROM Lab

Introduction

ROM is effectively just another way of defining the lookup tables in the XC4000 family. It can be very convenient to use for ROM-like transfer functions. In this example, the outputs of a 4-bit counter will be used to address a 16-word ROM. The 16-word ROM in turn will drive the row of 8 LEDs on the demo board. An advantage of using ROM is that you can define the light pattern directly.

This lab was originally designed for XACTstep v6.0. This lab has been converted to use LogiBLOX and M1 rather than an older tool called MemGen. This lab will demonstrate the use of LogiBLOX and the ability to modify a text file to control an LED light sequence stored in a ROM table.

Objective

- To understand how to integrate ROM into an XC4000 design.
- To practice using the LogiBLOX utility.

Procedure

This design replaces the 8-bit counters we've used with a 4-bit counter, since we will only use 16 words of ROM. The ROM effectively converts the binary counter output into any desired sequence. It would be more effective, in most cases, to put this decoding logic in front of the flip-flops, allowing the flip-flops themselves to generate the desired output function.

- 1) Invoke the **Foundation Project Manager**.
- 2) Select Project → **Open Project...**
- 3) Go to the **C:\F15_labs\Memory** directory and select the **COUNT** project.
- 4) Since this design was originally created to target an XC4003E device and we want to target the XC4005XL, let's change this. Most **Unified library** macros and primitives are common across several product families, so converting device targets is easy. Switching designs between various XC4000 sub families is almost always quite simple. In other cases, minor changes might need to be made, and the online Libraries Guide provides good information about what families all of the Unified Library macros support..

In **Foundation Project Manager**, click the **Design Info** box.

Under Project Info, change the **Family**, **Part**, and **Speed** settings to **XC4000XL**, **XC4005XLPC84**, and **3**, respectively. Click **<OK>**.

5) Select the **Schematic Editor**.

Note that there are pin locations shown as attributes on the several I/O Pads. However, the pin assignments in the **COUNT.UCF** file, which are specific to the XS40-005XL and XSTEND boards, will take precedence over the settings in the schematic.

6) Note the symbol for the ROM function following the counter. Select this symbol so that a red selection rectangle surrounds it. Click the **Disconnect Symbol** icon.



Hit the <**Delete**> key to delete this previously created ROM module.

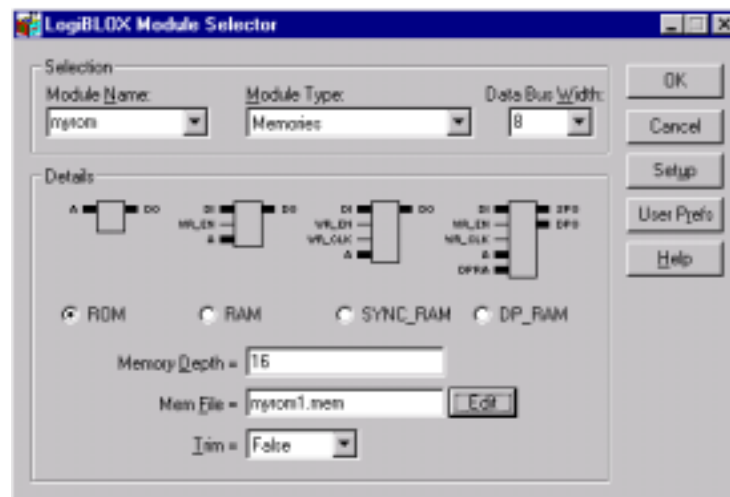
7) Select **Options** → **LogiBLOX...** Your LogiBLOX Module Selector window comes up. We will create a 16x8 ROM module called “**MYROM1**”.

8) Under Module Name, type **MYROM1**. Under **M**odule **T**ype, select **M**emories.

Choose the **ROM** radio box.

Type Memory Depth = **16**. Data Bus Width = **8**.

Type Mem File = **MYROM1.MEM**. This will create a text file for you which contains the contents of your ROM table.



9) Edit your ROM file by clicking on the **E**dit button.

- 10) As shown in the comments section of this text file, you need to enter 16 words of 8-bit wide data. Note that the default is HEX, Radix 16, and that any space, tab, comma, or return will be treated as the division between each of the 16 words. Try to come up with an interesting light pattern. If you like, you can try the following:

```
. . .  
RADIX 16  
DATA  
  
0:00, 1:11, 2:22, 3:43, 4:84, 5:45, 6:26, 7:17  
8:08, 9:19, A:2A, B:4B, C:8C, D:4D, E:2E, F:1F  
. . .
```

(You can also cheat by copying the completed **myrom1.mem** file at **c:\F15_labs\shortcut\memory\count\myrom1.mem** into the current project directory of **c:\F15_labs\memory\count\myrom1.mem**.)

- 11) Exit your text editor and save this new file as a **Text** document.
- 12) Return (<Alt>+<Tab>) to your **LogiBLOX Module Selector**. It is also helpful to put the LogiBLOX GUI Messages window in view now also so that you can watch LogiBLOX compile this module into an EDIF netlist.

Note: We will not do this in this lab, but if you wanted to instantiate this custom LogiBLOX macro into a VHDL design instead of into a schematic, you could click <Setup> under the **LogiBLOX Module Selector** window, click the **Options** tab, and check the VHDL options to create a **Behavioral VHDL Netlist** and a **VHDL Template**.

- 13) Select <OK> in the **LogiBLOX Module Selector** window. Your module, **MYROM**, has now been created. When done, a message should come on screen saying, “LogiBLOX symbol myrom1 successfully put into library.”

If LogiBLOX did not compile, you should first check in your Windows Explorer to make sure that the text editor did not append a suffix to the memory contents file name when it was saved. It should be named **MYROM1.MEM** and not **MYROM1.MEM.txt**.

- 14) Go to the **Foundation Schematic Capture** window. We will now insert this module into the design.

- 15) Click on the **Symbols Toolbox** icon to pull up a symbol.

- 16) Type **MYROM**. Once **MYROM** is highlighted in the SC Symbols window, you can place the symbol in the middle of your schematic *above* where the previous ROM block had been located.

Hint: Place the **MYROM** symbol so that the bottom of the symbol is above the top of the **CB4CLED** outputs, as shown in the diagram, Figure 1. This will make it easiest to connect wires to the bus taps in the next step.

If the **MYROM** symbol does not appear in the Symbols Toolbox, these are the most likely causes:

- Your project library's Access Mode is set as Read Only (R/O). This can be changed in the Library Manager utility.
- An error or incorrect directory was used when creating MYROM
- You need to update your project libraries to include the LogiBLOX module by selecting **File → Update Libraries.**)

17) Delete any hanging nets from the **CB4CLED** symbol.

18) Create an address bus: Click on the **Draw Busses** icon. Start at the top left and draw a bus called **ADDR[3:0]**, with I/O marker of **None**.

19) Connect the **Q0..Q3** output taps of **CB4CLED** to this bus.

Hint: Select the “Draw Bus Taps” icon, click on the **ADDR[3:0]** bus **name** once. Then notice the message on the bottom of the screen saying “**Expand Bus Tap: ADDR3**”. Then click once on each output tap **Q0** through **Q3**, i.e. from LSB to MSB. Tap to Bus connections are made automatically.

20) Connect the output bus of your ROM symbol to bus **O[7:0]** directly. Delete individual wires, or bus taps, hanging from the bus.

21) Name your ROM symbol by double-clicking on it and setting the **Reference** to some name such as **ROM_16**. Click <OK>. Your final schematic should look similar to Figure 1, below:

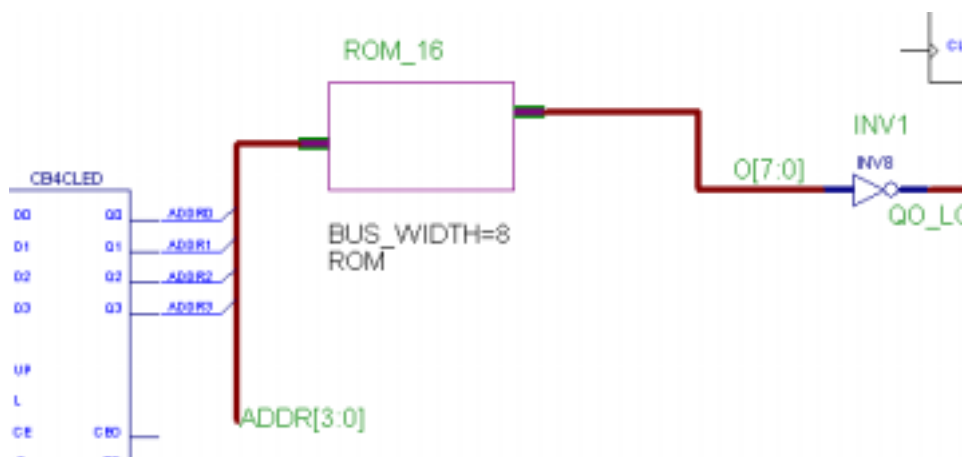


Figure 1

22) Save your schematic and return (**<Alt>+<Tab>**) back to **Foundation Project Manager**.

23) Click on **Implement M1**.

24) **Select Design → Implement... → Run** to compile your design. Make sure the **Produce Configuration Data** box is checked to be able to test your design.

25) When finished, download the design. Does it operate as you expected?

Note that in the User Constraints File (c:\F15_labs\memory\count\count.ucf) that these switches control the operation of the ROM counter:

```
NET UP_P      LOC=P7;      # DIPSW1 → Up (1)
NET CLKEN_P   LOC=P9;      # DIPSW3 → Up (1)
NET NOTCLR_P  LOC=P6;      # DIPSW4 → Up then Down (1 -> 0) to clear
```

If you have time, you can create a new memory file, or just modify your existing file and re-implement with the new pattern.

26) Based on the size of the ROM, how many CLBs should this ROM require? Examine the **Placement Report**. How many CLBs were required in the entire design?

.....

Optional – Timing Analyzer

27) Now we will look at the performance of the ROM, including specific net delays by filtering out everything but the inputs and outputs of the ROM block.

In the **Design Manager**, highlight the implemented revision. Then select **Tools → Timing Analyzer**, or click the corresponding icon instead.

28) **Select Path Filters → Custom Filters → Select Sources...** . Choose **Selected Sources** of type **Nets**. Highlight **ADDR<0>** to **ADDR<3>**, and click the move button '>' to select these sources. Click **<OK>**.

29) Similarly, select **Path Filters → Custom Filters → Select Destinations...** and select only **O<0>** to **O<7>** as the destinations.

30) Note that under **Options → Report Options...**, that the maximum **Paths per Timing Constraint** is set as '1'.

31) Now select **Analyze → Custom**. This will do a timing analysis on only the sources and destinations we have chosen.

What is the worst case address access time of the ROM (T_{ILO})?

What is the total worst case delay including nets?

Answers

How many CLBs should this ROM require?

Each CLB can hold 16x2, so 16x8 requires 4 CLBs. The entire design required 7 out of 196 available CLBs.

What is the worst case address access time of the ROM (T_{ILO})?

1.59ns

It is one level deep, so the max delay is the same as the block delay, 1.6ns in the XC4000XL-3, according to the 1998 Xilinx databook, p. 4-74 – “F/G inputs to X/Y outputs” ratings. A ‘09’ speedgrade would improve the time to 1.2ns.

What is the total worst case delay including nets?

Approximately 14.4ns. This will vary since the actual routing affects this value. See the Timing Analyzer report, as shown below.

Timing Analyzer M1.5.19

Copyright (c) 1995-1998 Xilinx, Inc. All rights reserved.

Design file: c:\f15_labs\memory\count\xproj\ver2\rev1\count.ncd

Device,speed: xc4003e,-4 (x1_0.95 PRELIMINARY)

Report level: verbose report, limited to 1 item per constraint

=====

Timing constraint: PATH "PATHFILTERS" = FROM TIMEGRP "SOURCES" TO TIMEGRP "DESTINATIONS" ;
32 items analyzed, 0 timing errors detected.
Maximum delay is 14.393ns.

Delay: 14.393ns ADDR<0> to OUT<5>

Path ADDR<0> to OUT<5> contains 1 level of logic:
Path starting from Comp: CLB_R6C1.YQ

To	Delay type	Delay(ns)	Physical Resource Logical Resource(s)
CLB_R7C1.G3	net (fanout=11)	2.555R	ADDR<1>
CLB_R7C1.Y	Tilo	2.700R	O<4> L1/MEM5_0
P81.0	net (fanout=1)	9.138R	O<5>
Total (2.700ns logic, 11.693ns route)		14.393ns	
(18.8% logic, 81.2% route)			