

Lab 4: Interfacing the 8051 with 16-bit Address-Latched External Devices

CpE 214: Digital Engineering Lab II

Last revised: January 1, 2013 (CAC)

You will design hardware to allow the 8051 on the Altera DE-2 board to use a ROM module that stores your program's instructions. The 8051 module does not contain internal code memory. To access an external I/O (like a display) location that is more than 8 bits in length requires an external address latch. An instruction such as MOVX A, @DPTR will output the high byte (in DPTR) to Port 2, and the lower byte to Port 0. The issue is that since the 8051 doesn't have many ports to work with, chances are you may already be using it for other things. Therefore you only want to send to the display at certain times, even though its always connected to Port 0. This is where using an address decoder and the latch comes in. You will incorporate this external latch on the Altera DE-2 board using the eight-bit latch designed in lab 1. You will also create a memory-mapped output port (via the address decoder) for the seven segment display from another instance of this same eight-bit latch. Schematics will be developed in Quartus-II software and will be simulated using the same. An EDIF module containing the VHDL packages of the 8051 microcontroller will be provided for you. You will then write a simple C program to verify your design in hardware.

0.1 Outline and Concepts

1. Design a 16-bit address decoder
2. Interfacing ROM to the 8051
3. Creating Top Level Design
4. Writing the C program
5. Testing with Cyclone II FPGA

1 Design a 16-bit address decoder

For your address decoder, you want create two 8-bit input buses (one that will take in P0, the other P2), and a wr_bar (inverted write) line input. The address in memory we are going to specify for our display is going to be located at 0xAA55. So we want the output of the address decoder to go high ONLY when the inputs are 0xAA55, and the wr_bar line is low. It will operate as shown in the block diagram of Figure 1.

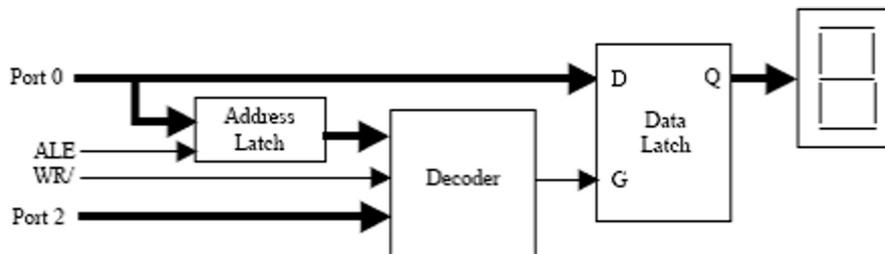


FIG. 1: Block diagram for Address decoder and 8-bit latches

The ALE (address latch enable) line in the figure is used for MOVX A, @DPTR instructions that make use of a full 16 bits (i.e. splitting the address bytes between P2 and P0). For this project we are going to hard code the ALE and wr_bar lines arbitrarily to Port 3 to make life simple. For the address latch, the 8051 will first write an address, which will be stored at this latch. During a data write instruction, the 8051 would then write data to Port 0. Data from P0 is “written” to the display when the WR/ line from the 8051 goes low. Data is latched by the data latch, then, when both the WR/ line is low and the address is 0xAA55, indicating that the 8051 has performed a MOVX to external data memory location 0xAA55. As far as the 8051 is concerned, the seven segment display is just another external memory location.

2 Interfacing ROM to the 8051

Since the original 8051 reference design did not contain any internal code space memory, it is therefore necessary to use a ROM device to hold our instructions. In our project, since our 8051 module has a 12-bit wide ROM address port, this is the type of ROM module we will be creating. The symbol is already provided for you on Blackboard. A timing diagram of how programs are fetched from the ROM utilizing the PSEN (program store enable) line is provided in Figure 2. PSEN is commonly known as the rd (read) line.

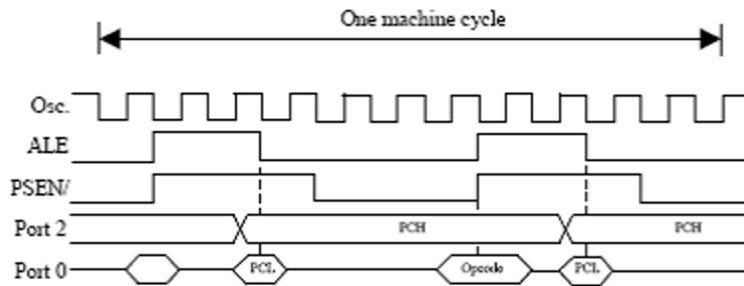


FIG. 2: 8051 timing diagram

3 Creating Top Level Design

The top level design in Quartus is merely combining the provided 8051 module, ROM symbol, your old 8-bit latch, and the newly created address decoder and wiring them together as shown in Figure 3.

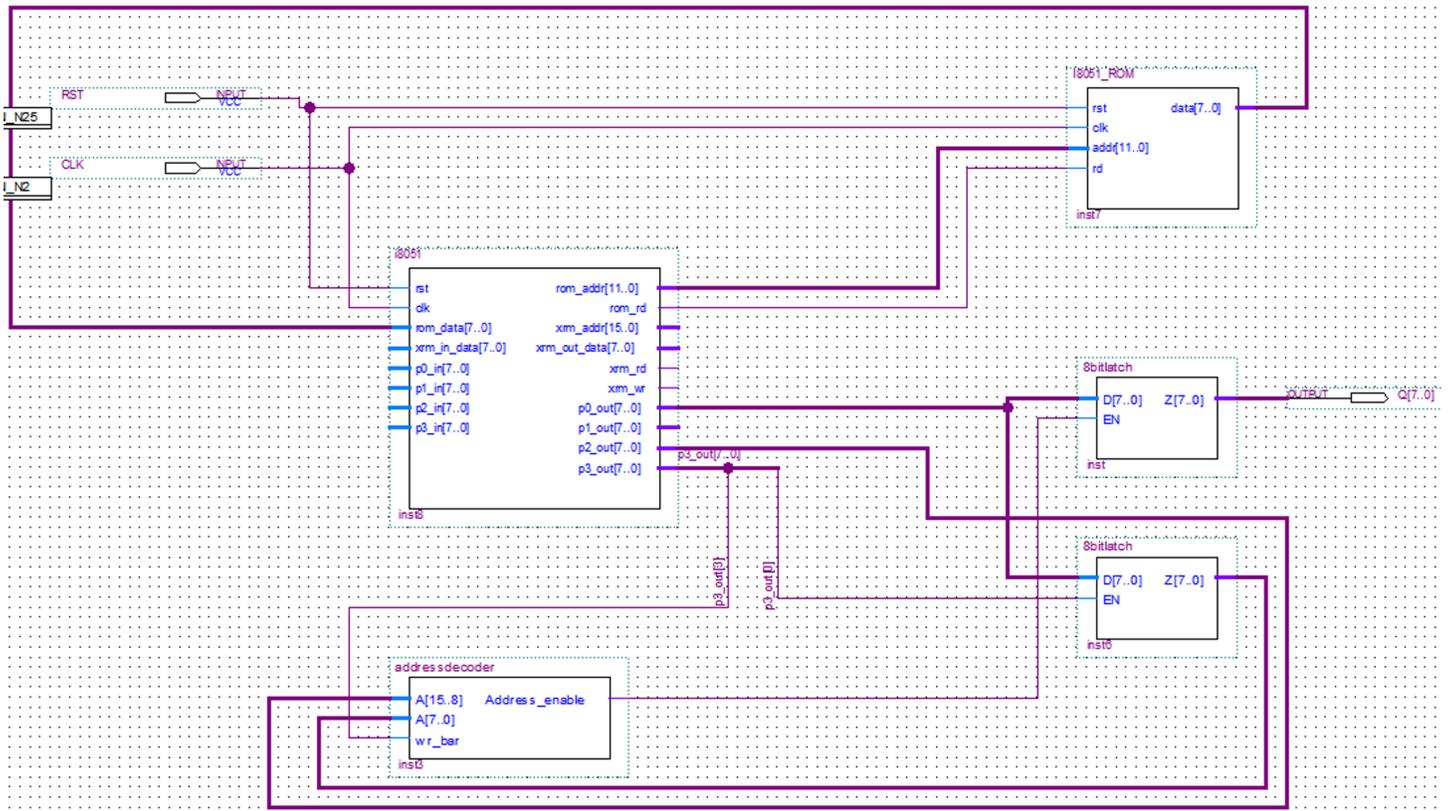


FIG. 3: Top level design to recreate

When you finish, you should compile your design to check for any errors. *If the only error you receive is about I8051_ROM, then everything is GOOD and you can proceed to writing your code.*

4 Writing the C program

The software you will write is going to demonstrate and test if our circuit works properly. Since this is your first C code project, you will be provided a template to get you started off correctly. This will also serve as a refresher for those who have not had a programming class recently. Just like in the C you are used to writing for Linux/Unix platforms, or the Visual .NET frameworks, you need to define function prototypes and you need a void main() at a minimum for your code to work properly. *Additionally, in the embedded C world, compiler specifics really matter!* Fancy tricks you are used to using in the C++ world, along with using a plethora of libraries you may be used to are not going to cut it in embedded systems. We use the industry standard Keil C51 compiler platform, and most all of these things are not supported. In fact, the simpler you can make your software, either through hard-coding, or otherwise is always your best bet.

Some general notes to be aware of:

- Ints and floats are VERY hard to work with. Choose char variable types whenever possible. This means decimal/fractions are out.
- Global variables tend to make your life easy. If you choose local variables within a single function, you *MUST DEFINE THEM AT THE BEGINNING OF THAT FUNCTION*, or Keil will throw compiler errors.

On to the instructions for how your code needs to work:

1. The data we will write to the address specified (0xAA55) on the 7 segment display is the word HELLO. You will write code to sequentially write each letter. The code for the letter “H” to be displayed should be when the corresponding segments 1,2,4,5,6 are held 0 (they are active low), with only segments 0 and 3 needing to be turned off (assigned a 1). Since the data is 8 bits instead of 7, just hold the 8th bit as a 0 in all cases. Thus the data code for displaying “H” will be 00001001, or 0x09. In the following manner each of the subsequent data codes can be formulated corresponding to the other letters. A diagram for how the segments are physically indexed on the board are defined is shown in Figure 4.
2. The next step is to convert your C code into the corresponding vhd format to act as your ROM symbol in the workspace. Have Keil generate a hex output file, and use the Hex to ROM converter on Blackboard to create the i8051_rom.vhd which you can import into your Quartus project.

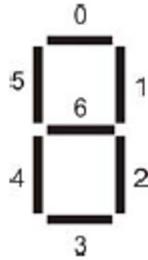


FIG. 4: FPGA’s 7-seg indices

5 Testing with Cyclone II FPGA

Before you test on the FPGA, you should compile your design first after you import your VHDL code back to Quartus. This way we can fix any errors first before you get to the board and something doesn’t work.

1. Assign input pins N25 as your reset, N2 as your clock, and lastly your output bus to the 7 segment display. Compile once more. Then download the design onto the Altera DE-2 FPGA board using the Programmer tool in Quartus II.
2. Observe the output and show the hardware implementation to your lab instructor.

6 Questions (attach at end of your report)

1. Explain how the address/data multiplexing takes place in your design.
2. Why is P0 called a “memory mapped port”?