EXPERIMENT NUMBER 4
Extending the 8051 with External Hardware: An Address Latch and Memory-Mapped Port

INTRODUCTION:

In this lab, you play the role of a hardware designer developing components to work with the 8051 and to extend the 8051's capabilities. The software-design team has not yet finished their design, but they have created a software model that implements the most important features of that design. You will use this model to test hardware you develop. By simulating hardware and software early, you have the opportunity to detect and solve problems in your hardware design before that design is set in stone (or set in silicon, as the case may be). Waiting for a hardware prototype to be finished before testing software could add to your troubles. The hardware may be too far along to make changes cost effectively, and some corrections may be impossible to make with software alone. Changing the hardware late in the design could add significant costs to development and could significantly extend the time-to-market, which could be disastrous to your product's success. By testing the design early and often, you can quickly and easily correct the problems that occur. Of course, testing should never replace the process of carefully laying out and developing the design in the first place – a design which is well thought out before any implementation is done is bound to be the best.

You will design hardware to allow the 8031 on the XS40 board to use external memory and to create a memory-mapped port for the seven-segment display. The 8031 included on the XS40 board does not contain internal code memory. An external SRAM is used to store the code. To access this external memory requires an external address latch since the lower address byte and the data are multiplexed on port 0. You will incorporate this external latch in the FPGA on the XS40 board using the eight-bit latch designed in lab 1. You will also create a memory mapped output port for the seven-segment display from another instance of this same eight-bit latch. Note that while a transparent latch would be preferable in these applications, the Xilinx 4000E series FPGA provides only edge triggered flip flops so we must use an edge triggered register instead if a 4000XL series FPGA is not available.

Hardware will be developed in Design Architect and will be simulated using QuickSim Pro. A program for the 8031 will be provided for you. Once your design has been verified through simulation, you will verify your design in hardware.

OBJECTIVES:

1. Improve hardware-software design skills.
2. Learn how to demultiplex the 8051 address/data bus.
3. Observe the timing of signals involved in a code-fetch memory cycle.
4. Introduce a simulation model of the 8051 microcontroller.
5. Learn how to add a memory mapped output port to the 8051.

REFERENCES:

1. Appendix D, XS40 Schematic: http://www.ece.umr.edu/courses/cpe214/schematics.pdf

MATERIALS REQUIRED:

- Mentor Graphics software: Design Architect, Quicksim Pro
- XS40 simulation model: http://www.ece.umr.edu/courses/cpe214/dist/lab4.tar
- 8051 program file: http://www.ece.umr.edu/courses/cpe214/dist/hello.hex
- XS40 board
- Windows-based computer with an unused parallel port
- Ftp program

BACKGROUND:

The 8031 requires external code memory because it contains no internal code space. A latch is needed to latch the lower eight bits of the address from port 0 because this port is multiplexed with both data and addresses. Figure 1(a) shows a block diagram of the 8031 with external memory and an address latch. The schematic shows that port 0 is connected both to the memory address lines

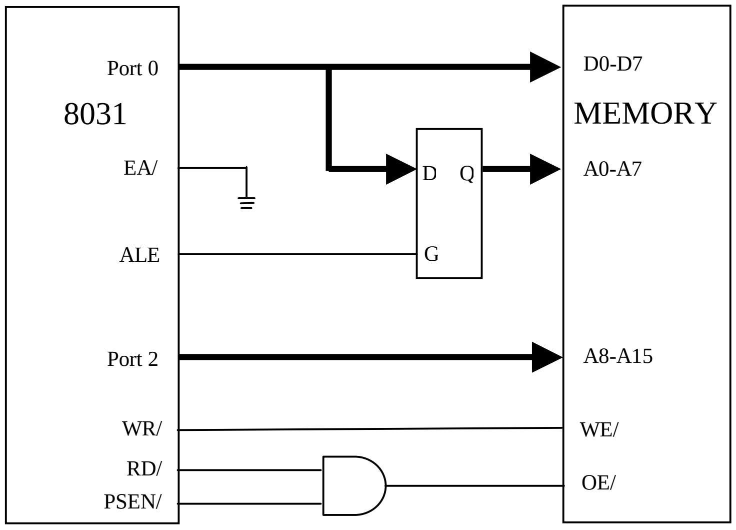


Figure 1(a) 8031 code space expansion

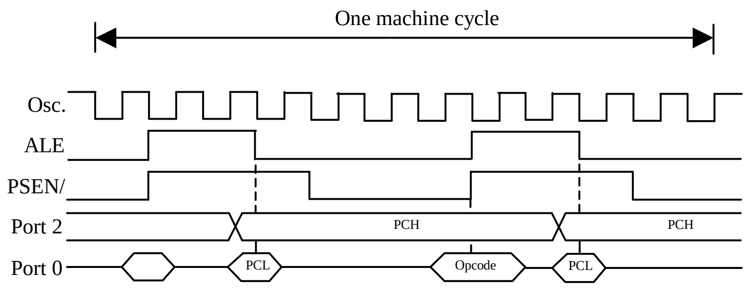


Figure 1(b) 8051 machine cycle

(through the latch) and to the memory data lines. Figure 1(b) is a timing diagram for an external memory access. A memory access proceeds as follows: 1) The 8031 writes the lower and upper address byte out port 0 and port 2, respectively; 2) On the first falling edge of ALE (address latch enable), the low byte (PCL) from Port 0 is latched. Once latched, port 0 is available to receive data from memory; 3) PSEN (program store enable) goes low, enabling memory to send data back to the 8031. 4) Data – the opcode of the next instruction – is received by the 8031. This process is illustrated in the timing diagram in Figure 1(b). Notice that Port 2 does not need a latch because its value does not change during a single bus cycle.

To free up ports on the 8051, external devices can be mapped within the microcontroller's data memory address space. In this case, you can read or write the device by using a MOVX instruction. In this lab, we would like to map the seven-segment display to memory location 0xAA55. We will put a latch there so that data we write will be shown on the seven-segment display, even after the microcontroller has gone on to other tasks. The block diagram is illustrated in Figure 2.

The address latch is the same latch seen in Figure 1(a). Just as before, the 8031 will first write an ad-

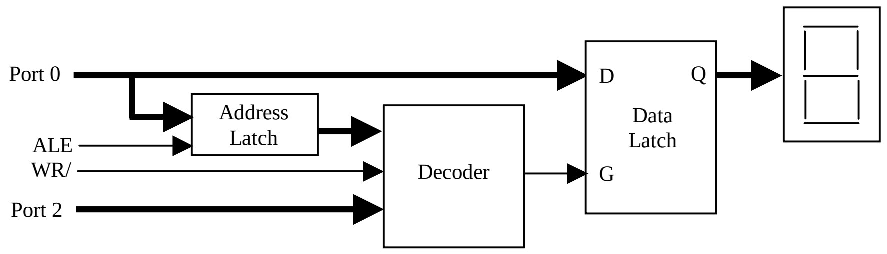


Figure 2 Memory mapped output port

dress, which will be stored at this latch. During a data write instruction, the 8031 would then write data, which would be stored by the data latch if the address was 0xAA55. Data is "written" when the WR/ line from the 8031 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 8031 has performed a MOVX to external data memory location 0xAA55. As far as the 8031 is concerned, the seven segment display is just another external memory location.

The steps needed to write a value to the seven segment display are illustrated in the following example. Say the 8031 wants to write a number, say 0x42, to the seven segment display. It would do so using the assembly code:

```
MOV A, #42H
MOV DPTR, #0AA55H
MOVX @DPTR,A
```

The timing diagram resulting from the fetch and execution of the MOVX instruction is shown in Figure 3. This timing diagram was adapted from a diagram in the Philips 8051 Family Hardware Guide. After the instruction is fetched, the MOVX command writes the address (0xAA55) out P0 and P2. This address is latched on the falling edge of ALE. After a short while, the data (0x42) is written out Port 0 to the data bus. The WR/ line goes low, writing this value to the external latch.

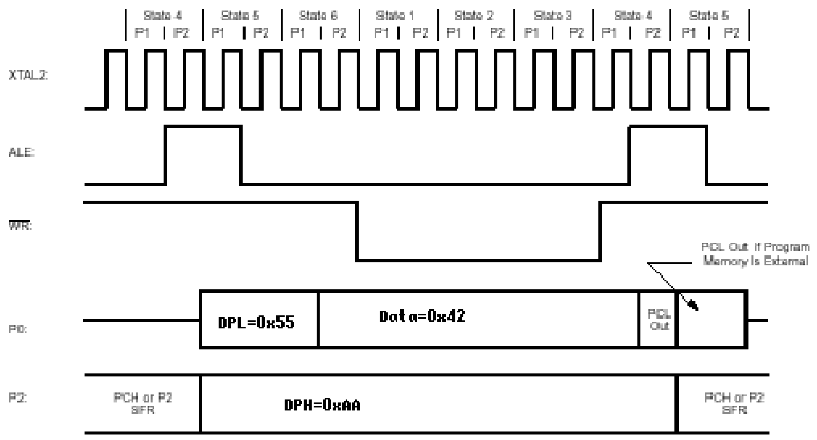


Figure 3 MOVX instruction timing

PRELIMINARY:

- Review the design and simulation of the eight-bit register from lab 1.

- Trace the connection from the oscillator circuit to the FPGA on the XS40 schematic and note which pin is used to connect the oscillator to the FPGA. The FPGA pins are labeled on the schematic as XCBUSxx. Repeat for the XTAL1 connection on the 8031. Repeat for the CS/ pin of the SRAM. Trace the connections between the low order addresses from the 8031's Port 0 and the high order addresses from Port 2 to the SRAM. Note the position of the FPGA and which pins are used. Trace the connections between the SRAM I/O lines, the FPGA, and the 8031's Port 0. Note in particular that the 8031's Port 0 and Port 2 pins are wired to the SRAM in a peculiar 'scrambled' order. For example, note that P2.6 (address bit 14) does not connect to SRAM's address bit 14 but to address bit 6. This was no doubt done to make the printed wiring board layout more efficient and it makes no difference in the operation of the SRAM but can be a bit confusing at first. What it means is that if you want to look at the address bus in 'normal' order, you'll need to look at P0 and P2 coming out of the processor and not the address(14:0) pins on the SRAM. The SRAM's data bus pins are similarly scrambled. If you open down on the SRAM schematic while in da, you'll see this address mapping a bit more clearly.

- Sketch a design for the address decoder and latch for the seven-segment display in your lab notebook. Keep in mind that you are using a level sensitive latch and that addresses are latched on the trailing edge of ALE, not the leading edge. Only use components that are available in the Xilinx library. Label the inputs and outputs of your design to show how they should be connected to the rest of the circuit (for example, the output from your latch should be labeled to show it is connected to the pins controlling the seven segment display). Pay special attention to the decoder input signals (the address and WR/) and the decoder output signals (output register clock input), to make sure your design will function as desired. You do not need to show the internal logic of the 8-bit register, since you already designed this component in lab 1.

PROCEDURE:

Summary: Create an address latch, address decoder, and output port in Design Architect. Connect them up as in Figures 1a and 2. Connect necessary control signals to the 8031 through the FPGA. Cosimulate your hardware with "demo" software in QuickSim Pro. Download the design to the XS40 board and verify in hardware.

1. Download the simulation model of the XS40 board from http://www.ece.umr.edu/courses/cpe214/dist/lab4.tar and untar it in the directory used for lab 1. This file contains a top level model of the XS40 (xess40_schematic) as well as models of each major component on the XS40 board. Untarring the file in this directory will overwrite the old simulation model, but will save your 8-bit register design. Run *build_simulation* to compile the 8051 VHDL model.

2. Create the address decoder for the seven-segment display. Begin Design Architect (da) and open a blank sheet. Draw the logic for your decoder within this sheet. Connect the inputs to and outputs from the decoder to portin and portout ports. Using ports will allow you to create a decoder "symbol". When you have finished creating your logic, check and save the *component* as "mydecoder". Create a symbol for this decoder and check and save it as "mydecoder".

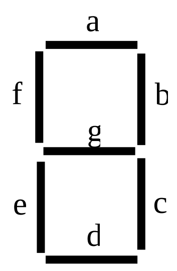
3. Add the registers and decoder to the design within the FPGA. Open the xc4005 sheet. Place your decoder (the decoder symbol) and two instantiations of your register within the sheet. You will be wiring them up similar to Figure 2, so you may want to place them in a similar manner. Wire the latches and decoder to each other and to the seven-segment display and to the address and control lines from the 8031. Use the circuit sketch you made in the preliminary assignment as a guide. The example software (hello.hex) makes the assumption that segment g (P20) is wired to the least significant bit of the data byte (P0.0) so wire the display accordingly. The connections to the seven-segment display on the xc4005 component schematic sheet are not in order so be careful how you wire them up. It's helpful to relate the xc4005 pins to those shown on the xess40_schematic sheet. Table 1 shows the FPGA/7Seg connections.

4. To complete the circuit, you must also make the following connections:

   a) The clock signal is not directly connected to the 8031 on the XS40 board, rather you must route it to the 8031 through the FPGA. To route the clock, connect pin 13 of the FPGA (which is connected to the on-board clock oscillator) to pin 37 (which is connected to the XTAL1 clock input of the 8031). Verify these pin numbers from your preliminary notes in your lab notebook.

Table 1 FPGA - Seven Segment Connections

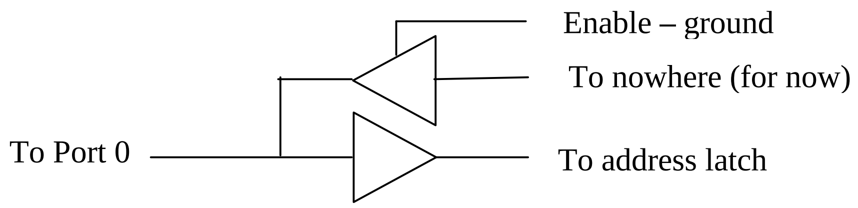
FPGA	7Seg
P19	a
P23	b
P26	c
P25	d
P24	e
P18	f
P20	g



b) Address line 15 should be connected to the CS\ pin of the SRAM. Use your preliminary notes to verify the FPGA pin number to use.

c) Connect the address and data lines from the 8031 to SRAM, as shown in Figure 1a. Port 2 of the 8031, which writes out the high address byte, is already connected directly to the SRAM on the XS40 board, so you don’t have to make any connections there (See schematic, appendix D). You will need to connect your address latch between port 0 and SRAM. Port 0 is connected to the FPGA as are the low-order address and data lines of the SRAM. Connect the wires to port 0 to the input of your address latch and connect the output of the latch to the pins running to SRAM.

The schematic has two drivers (buffers) connected up to port 0, an input buffer and a tri-state driver (see below). The input buffer boosts the input signal to be used within the FPGA. The tri-state driver is used to write data from the FPGA back out to port 0. The tri-state driver is used so that the FPGA does not write something to port 0 while something else is driving those signal lines (i.e. the 8031 or SRAM). Normally, the E (enable) line of the tri-state driver would be connected to logic such that the FPGA would only drive signals out to port 0 on a read. For this exercise, the FPGA never needs to write data back to port 0, so we can disable the tri-state drivers simply by grounding the enable lines (E=0). Go ahead and ground these lines if they haven’t already been grounded for you.



d) An 8051 microcontroller typically uses two separate chips for data and for code memory. The XS40 board, however, uses a single SRAM chip for both program and data memory. The extra logic is required so that the SRAM’s OE/ pin will go low when *either* PSEN/ (code fetch) or RD/ (data fetch) go low. The logic is a simple AND gate, as shown in Figure 1a. In this case, OE/ = RD/ & PSEN/, so that OE/ is high only when both RD/ and PSEN/ are high (e.g. not active) and are low otherwise. To implement this logic, tie the pins RD/ and PSEN/ from the 8031 to the OE/ of the SRAM using a nor2b2 (the DeMorgan equivalent of an AND gate). Also tie WR/ from the 8031 to WE/ of the SRAM. Recall that pins P3.7 and P3.6 of port 3 double as the RD/ and WR/ lines.

e) Connect the reset pin of the 8031 (XCBUS36) to jumper J1-2, pin p44, which is part of the parallel port jack. By connecting reset to the PC parallel port, we can reset the 8031 directly from the PC using the GXPORT program.

5. Check and save your design and exit Design Architect. (If you are unsure of your design, you might minimize Design Architect instead. That way, if you have to make changes to your design you only have to recall the window instead of restarting da.

6. The subdirectory *program_files* contains two dummy files that hold the contents of the 8031's program memory. Code.hex contains data for the 8031 model's internal code space while sram.hex contains data for the SRAM model. Since the XS40 uses external code space, you will need to either make sram.hex the same as your program file or point sram.hex to your hex file using a symbolic link. If hello.hex is not already in program_files, download a copy of it from http://www.ece.umr.edu/courses/cpe214/dist/hello.hex and copy it over the file sram.hex located in the sub-directory: program_files. This program, hello.hex, will be run by the 8031 to help verify your design. QuickSim automatically loads the file sram.hex and runs it on the 8031 model during simulation.

7. Start QuickSim Pro using the command:

```
qspro -as hdl xess40_schematic
```

8. Simulate and verify your design. Set QuickSim to trace: a) WR/, PSEN/, ALE, P0, and P2 from the 8031, b) the latched address from P0, and c) the seven output lines to the seven-segment display. The steps needed to add these traces were shown in lab 1. Add the GLOBALSETRESET line as an editable waveform and pulse it as in lab 1, so that the flip flops will be in a known state. Also keep the RST input to the 8031 low by forcing J1-2 high. You may also want to look at the SRAM control lines. Be careful about which lines you select to display or you'll get the addresses and data in XS40-scrambled order!

9. Run the simulation for about 100,000 nanoseconds, which is equivalent to 1200 clock cycles (with a 12MHz clock), or about 45 to 80 instruction cycles. During simulation, a QSPro(HDL) window will show the output to the seven-segment display. You should see the message "hello world" appear in this window, each character displayed one after the other. The character displayed should change on the trailing edge of WR/ when the address lines contain 0xAA55. If your design does not work at first, you should carefully check the address, data, and control signals to solve the problem. Compare what you expect to happen when signals change on these lines to what you actually see happening. If needed, look at signals between parts inside your FPGA. The ability to look at signals inside chips is a big advantage of simulation. You could not do this if you were using actual hardware chips or at least it would be rather difficult to do so. Fix any problems in the original design using Design Architect and recompile and reload your simulation. Indicate what problems occurred and what you did to solve the problems in your lab notebook.

10. Once software simulation is complete and you can simulate the correct message on the seven-segment display, print out a copy of the trace that contains the WR/ signal, address bus signals, and the signals sent to the seven-segment display. Exit QuickSim (and Design architect, if necessary).

11. Create a bit file of your design, which will be downloaded to the FPGA on the XS40 board. This can be accomplished using the ‘xmake’ command as shown in lab 1.

12. Download the hex program file (hello.hex) and your bit file to the XS40 board using GXSLOAD program (see lab 1). Use the digital oscilloscope’s digital inputs to display the WR/ signal and the seven signals going to the seven-segment display. Clear the scope screen with the Erase button. Set it to trigger on the WR/ signal rising edge, capture a single trace, and set the time division to 5 μs/div. Reset the 8031 by toggling bit D0 of the parallel port using GXSPORT. Toggle the reset line and capture a screen full of information. Print these results and compare them with those obtained in QuickSim. Mark and explain any discrepancies.

13. If all went well, the trace from your scope should match the trace from your simulation, so the program and your design work, right? Look at the seven-segment display. It should be showing the sequence “hello world”. Does it? Why doesn’t the seven-segment display show the correct values? Remember, we already know from simulation that several aspects work: the 8031 is connected correctly to the address latch and to SRAM; the address latch works; the decoder and data latch work, the data latch is connected correctly to the seven-segment display, the program causes the correct values to be sent to the seven-segment display. What else is left? Write your solution in your notebook.

    A significant advantage of simulation before implementation is that you can eliminate most problems ahead of time. Problems are much easier to solve in a simulation environment, but you won’t always eliminate them all. When you see a new problem when you implement the design in hardware, you have a measure of confidence in your design and can immediately eliminate several possible causes that were already tested through simulation. The problem can be solved much more quickly as a result. It’s much easier to change a connection or a component in a schematic capture program than it is to change a trace on a printed circuit board or an ASIC.

14. Your TA should have a “working” version of the hex file which corrects the problem seen in step 13 above. Get this hex file from your TA and run the program again. Does the seven segment display show the correct message now?

QUESTIONS:

1. In part 4b of the procedure, we connected address line 15 to the CS/ pin of the SRAM. As a result, at what addresses (locations in the address space) can the 8031 access the SRAM? Will the 8031 write data to the SRAM when it performs an external memory write to address 0xFFFF? Why/why not?

2. Name some advantages of simulating hardware before creating a hardware prototype. If you did not have software (or, at least, a model of the complete software), why would it still be advantageous to simulate your hardware?

3. In step 13 of the procedure, we found one error with our design that was not uncovered (easily) with simulation. Describe three other possible errors that might not be discovered with simulation.

4. Given that some errors are not easily found through simulation, can you list some reasons why we continue to use simulation, rather than skipping simulation entirely and going straight to the final hardware test?

5. Take your oscilloscope printout from step 13 and mark each external memory write and the value written. Indicate what letters on the seven-segment display these values represent.

6. Explain why the seven-segment display is a “write-only” memory location. How could we extend the hardware to allow reading from the seven-segment display? Sketch the necessary hardware modifications in your lab notebook.