



Laboratory Stage #1

Interface RAM and a Latch to the TSK80 CPU

1 Introduction

This document provides details for the first stage of the microprocessor-based digital stop watch laboratory project. An overview of the project is described in the *Laboratory Project* hand out and the project was also covered during the Week 8 lecture.

Apart from the CPU, reset circuit and the JTAG connector, the first stage requires the design and implementation of an:

- Address decoder to select between the RAM and the latch,
- Data-bus ORing function for CPU input data,
- RAM interface (includes the RAM and simple logic), and
- The latch to be used for debugging purposes.

Most of the logic design will be done in VHDL, except where directed otherwise, such as where simple logical components could be used (it is interesting to compare the benefits and disadvantages of using VHDL over a schematic description). An FPGA project for the hardware design and an embedded project for the software will need to be created. Refer to *Using the TSK80x Embedded Tools* manual for information regarding writing software for the TSK80 in the C programming language. This document is available as part of the help facility or on-line:

- GU0109 Using the TSK80x Embedded Tools
(http://www.altium.com/files/AltiumDesigner6/LearningGuides/GU0109_Using_the_TSK80x_Embedded_Tools.pdf)

The top-level functional decomposition is shown in Figure 1. Highlighted is that part that will be implemented in this laboratory. Note that it is not very detailed and does not include modules for address decoding nor the data bus ORing function.

2 Top-Level Schematic

The top-level schematic of the design used for this stage of the laboratory is shown in Figure 5. The data-bus ORing function, address decoder and latch are all VHDL files. The RAM is implemented in another schematic with the RAM and some logic. The reset circuit is the same as that used in the tutorials. The JTAG circuit has to be included to allow downloading code to the TSK80 and for debugging. You may also want to experiment with virtual instruments for debugging your design.

3 Address Decoder

The address decoder is used to select which device on the bus will be accessed. All devices should be placed in the memory address space (that is, do not use the I/O address space). The memory should be located at 0×0000 and the address for the latch is up to you. However, you should consider what other devices will be added in the future (LCD interface, keypad interface and timer). The address decoder should be written in VHDL.

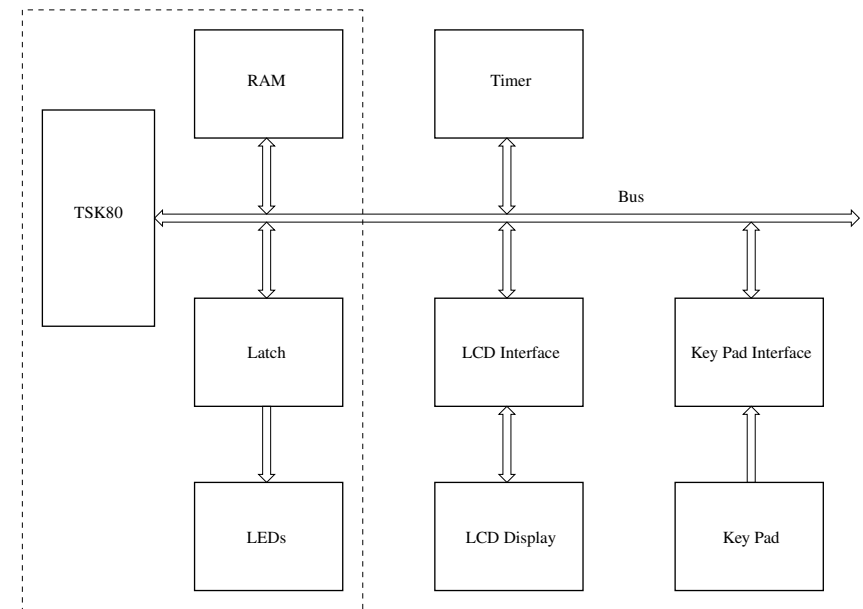


Figure 1: Top level functional decomposition with modules required for Stage 1 highlighted.

4 Data-Bus ORing Function

In multi-chip designs, the data bus going to devices has the ability to go into a high-impedance state when they are not being accessed by the CPU. This allows the data lines from different devices to be connected together. However, when the whole system is contained within an FPGA, a so called System-on-Chip (SoC) design, there are benefits in doing this a different way.

The CPU, RAM and other microprocessor-interface library parts have two sets of data lines, one used as an input and one used as an output. There is no high-impedance state. A bus master (such as a CPU) has its data-output lines connected to all the data-input lines of devices on the bus. The opposite direction is more complicated as there are a number of devices that want to send data back to the CPU during a read operation.

Routing data back to the CPU can be done a couple of different ways. The first is to implement a multiplexer which directs data back to the CPU depending on which device is selected. This method was used in Tutorial 5. Another method is to ensure that devices which are not selected output a logical low and only the selected device outputs valid data. Thus, all data lines can be ORed together with the result that only the data from the selected device will be routed back to the CPU. A comparison of these two methods is shown in Figure 2. It is interesting to note that these two methods are actually performing the exact same function and will use the same number of FPGA resources. The only thing that is different is how the functionality of the design has been decomposed.

The data-bus ORing function has the data lines from all the devices on the bus as inputs (the RAM and latch in this case) and the output goes to the data-line inputs of the CPU. It could have been implemented using a schematic part but instead it should be implemented in VHDL. You will find that this makes it easier to add extra data lines from other devices in future stages of the laboratory.

As the design for the ORing function is rather trivial, there is no need to simulate it.

5 RAM

As was found in the tutorial sessions that included a CPU in the design, the RAM used has a clock input. That is, this clock input, which is the same clock as used by the CPU, is used to latch the data for a write cycle. Normally data is latched by a RAM on the rising edge of the write signal (negative logic) with the data guaranteed to be valid for a sufficient

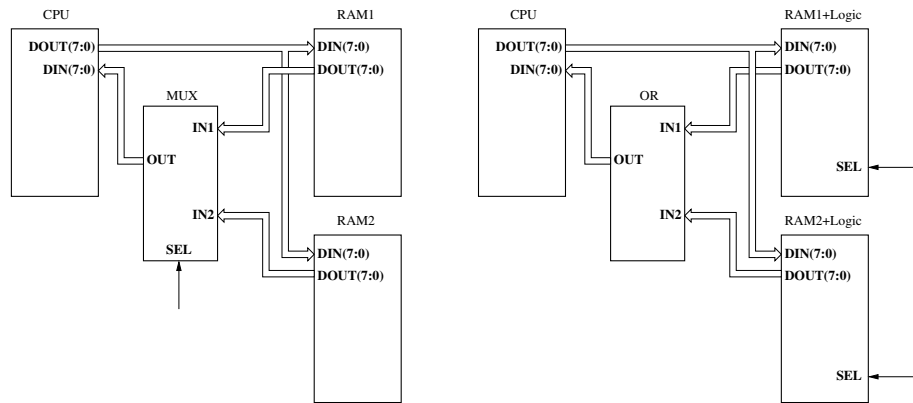


Figure 2: Comparison of a data bus implemented using a multiplexer (left) and using an ORing function (right).

period of time.

When the whole design is implemented in an FPGA, the change in state of the write signal cannot be used since the change in state of the write signal occurs at the same time as the change in state of the data and address signals which are all dependent on the system clock. Thus, the write signal is used as an enable signal to cause the RAM to latch the data on the next rising-edge of the clock. The timing diagram for a write to a RAM which has a clock input is shown in Figure 3. This type of RAM, used in the FPGA, also has separate data input and output signals. There is no extra signal required

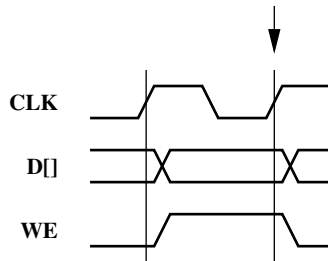


Figure 3: Timing diagram for a write to a RAM which has a clock input with the data being latched on the rising edge of the clock signal when the write-enable signal is asserted.

for a read operation as the data lines for the current address are always being driven.

The symbol shown in the schematic of Figure 5 for the RAM is actually another schematic with the RAM and some very simple logic that does not require any VHDL code to be written. The schematic for the RAM is shown in Figure 4. Note that the multiplexer is used to ensure that the data lines going back to the CPU are logically low when the RAM is not being read.

There is no need to simulate this part of the design.

6 Latch

The latch is used for debugging purposes to drive the LEDs on the NanoBoard. This should be written in VHDL and it should also use the clock signal to latch the data from the CPU when the latch is selected and it is a write cycle. It should also be possible to read the current value of the latch. Remember that the data output from the latch back to the CPU must be logically low if the latch is not being read by the CPU.

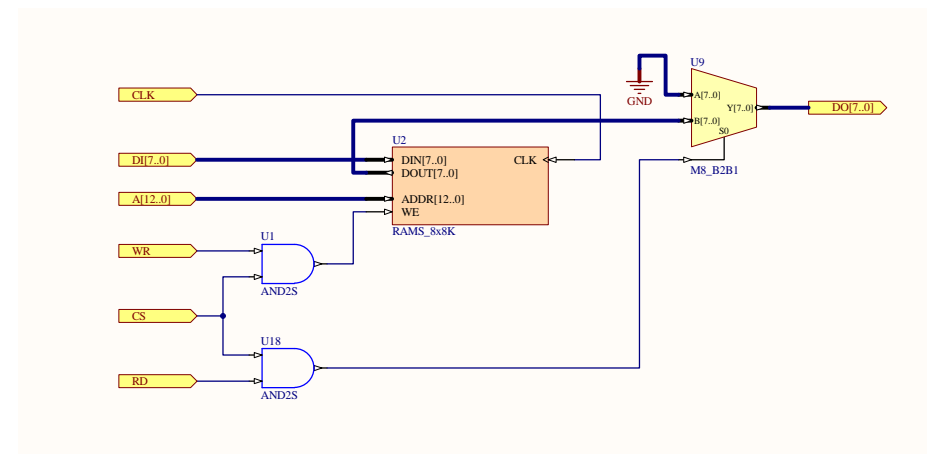


Figure 4: Schematic showing RAM and some logic including a multiplexer.

7 Test Program

The embedded project should be linked to the FPGA project as was done in the tutorials. The address space for the code and data both reside in the RAM that you have implemented in your design. These are labeled as `xrom` and `xram` sections by the compiler. The start address and length of these sections are set as a project option. Set the start of the `xrom` section to be `0x0000` and the length to be `0x1400`. Set the start of the `xram` section to be `0x1400` and the length to be `0x0c00`. These can be adjusted depending on the program you write. You should also set the stack and heap size to be `1k`.

You should write a test program to check that the RAM and latch interface are working correctly. The code should write and read from the RAM and latch to ensure that data is being stored and retrieved correctly. Be careful of compiler optimization when writing the test code. Pointers can be used to access the LEDs at a fixed memory address.

