DESIGN AND IMPLEMENTATION OF A RAM ON THE XS40 BOARD AND THE BIDIRECCTIONAL INTERFACE WITH A PC.

Miguel Ángel Aguirre Echánove, Jon N. Tombs
Dpto. de Ingeniería Electrónica. Universidad de Sevilla. Escuela Superior de Ingenieros
Avda. Camino de los Descubrimientos s/n. 41927 Sevilla (SPAIN)
aguirre@gte.esi.us.es, jon@gte.esi.us.es

Abstract

We present an implementation exercise of a laboratory project. Our purpose is to build a basic design that can be considered as the seed for more complex projects, which will be performed by students using XeSS XS40 proto-type boards and XilinX Foundation Express software. This equipment is extensively used by Engineering students in the Escuela Superior de Ingenieros de Sevilla (SPAIN).

We have developed two modules:

-A set of routines in C++ that solves the dialogs between the Xess board and the PC in both directions.

-A full VHDL design project that controls the interface with the PC and the FPGA.

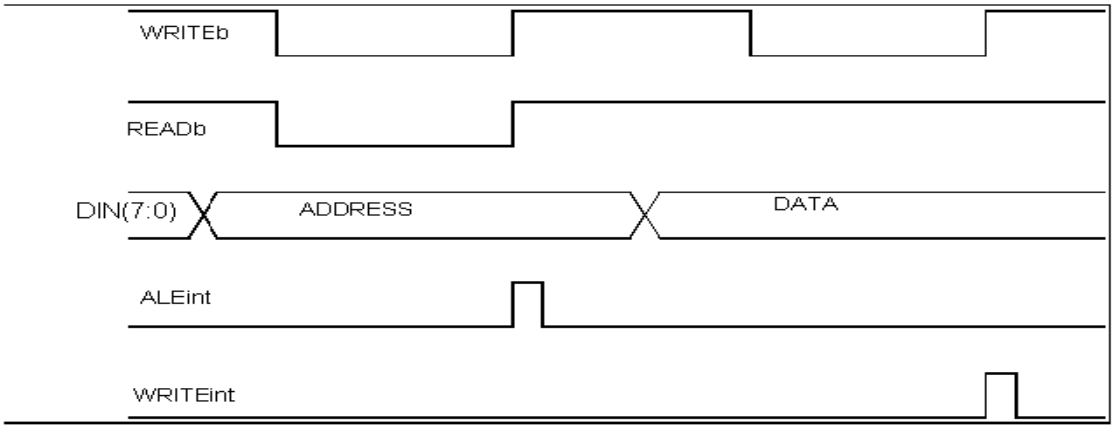
This exercise consists on a 8 bit RAM with 4 address. This memory can be accessed from a host (PC) through the parallel port, in both write and read mode. The board display shows the address that is selected when we are writing or reading.

The parallel port is connected to the programming connector. We do not need any additional hardware. The parallel port must be configured in SPP mode, because in this mode we can control every single control and status line from software routines. If any other mode is selected, the nSTROBE line will toggle when the port is written to, and the nPROGRAM signal will erase the current programming target.

Reading and writing method.

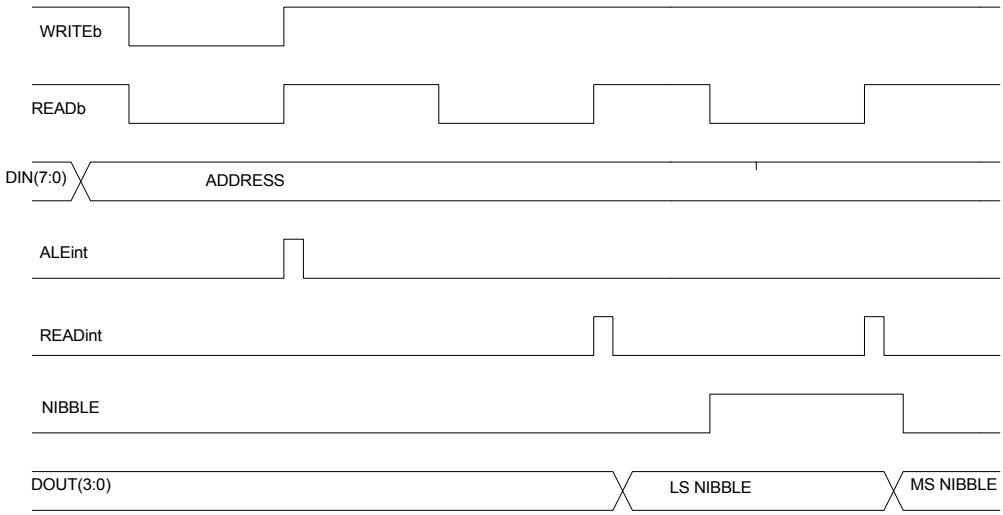
The RAM memory has been designed using a two control wires method, which determines if we are writing an address, writing or reading data. We call theses lines WRITEb and READb. Both are active low.

The following figure shows the proposed method for writing a byte. If WRITEb and READb are low, the FPGA consider the byte in data lines as an address. ALEint, READint and WRITEint are internal signals.



When the line rises the protocol FSM reads the address, write or read signal and the byte are internally latched, in the address register or the selected byte register.

The reading method is more complex, because we do not have enough parallel port lines that can be read. We must transfer the data in Nibbles. So we will read a byte in two reading cycles. We generate an internal signal NIBBLE that controls the output multiplexer. We can read it outside but it's not necessary for our purpose. The following figure shows the reading method:



The transfer speed of the SPP port is not very fast. We can read about 50Kbyte per second, but the FPGA is much faster than the parallel port and transfer method will be very robust. However we have an unused status line that can be used as a BUSY signal for obtaining a valid data.

C++ program

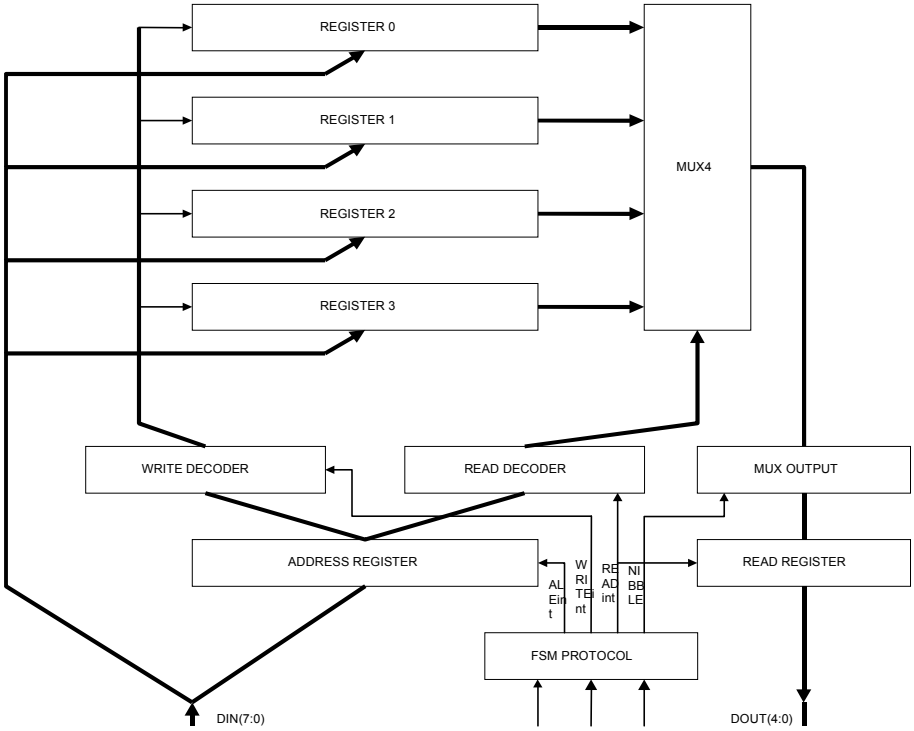
This program has been developed in Borland C++ linked with the port access library DlPortIO, obtained from Scientific Software Tools, Inc. It runs fine in W95/NT. First of all the address and port lines assignment are documented in the source code. We have considered these questions:

1. The base address for the LPT1: port is fixed. It's not very difficult to obtain it automatically or make it variable.
2. We must toggle some lines due to port and on board inversions.
3. Status register transfers through bit 1 to 4. We must shift right in low nibble and left in high nibble.
4. The program solves the transference for our sample, in a dialog window. It's very easy modify the program for other purposes.

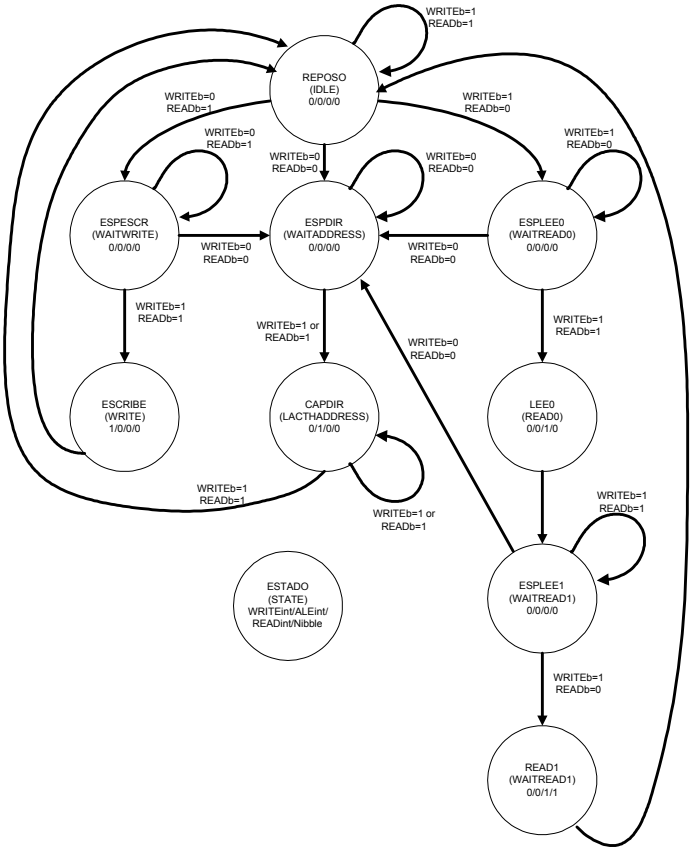
VHDL Program

The VHDL design is very simple. Figure X shows a block diagram with the basic elements. The most interesting block is the Finite State Machine that controls the dialogs with the parallel port. Before explaining it, here are some comments about other blocks:

1. The Xsess boards have special purpose pins that must be instanced in a special way (TDI, TMS, TCK, ...)
2. The on board 7 segment DISPLAY shows the selected address.
3. Two flip-flops are included at the input of the FPGA for READb and WRITEb. We don't know if the PC will produce glitches in these signals.



The FSM that controls the designed protocol is depicted in the figure. When READb or WRITEb is lowered the machine acts to read or write and waits for the rising edge. If the READb and WRITEb signals are lowed the machine waits for an address.



Application note

We have heard about at least five versions of the XS40 board. Version 1.2 or lower has an 12MHz oscillator and it works fine with our software. Version 1.3 has a programmable oscillator that must be programmed less that 40MHz. We work with 20MHz.

XS40 1.4 Version has new important changes in the FPGA-PORT pin assignment due to the memory of 128K and the FPGA programming method. Our design has been adapted to the new architecture and we can work with the new version too. The user must comment some lines in the VHDL source code and uncomment others. If the line is marked for the v1.3 must be active if we are using this board or older. Also we must comment the lines for 1.4. On the contrary is we have a v1.4 the active code for this version must be uncommented. The user constraints file has also two related board line.

New Capabilities (v. 2.3 and newer of the spp.exe program)

We have included five new capabilities in spp program for easy working with ascii files.

1. *ReadFileAddress*. Reads a file named < *filename*..rd> contents of the internal address from the FPGA, and writes the results in other one <*filename*.rdw>. The input file format is an integer number representing the address per file line.
2. *ReadFilePoll*. Reads a file named < *filename*.rd> the number of data from the same address at each of the FPGA and writes it in another file named < *filename*.rdw>. The input file format is two integers per line. Firs represents the number of data to be read and second the actual internal address. Output file shows the read values in a sequential way.
3. *WriteFileData*. Writes data read from < *filename*.rdw> into the FPGA. File format is one integer per line.
4. *WriteFileAddrData*. Reads the file named < *filename*.rdw> a pair of integers address and data and sends them to the FPGA. File format is two integers per line, firs address and second.
5. *RandomReadWrite*. Reads or write from or to an internal address. A file named <*filename*.rd> (<*filename*.ird> for version 2.4) is read. It has three fields per line: address, data and command. The field command is 0 when a data has to be written in a particular address and a 1 if a data has to be read from a particular address. In this case the field data from the input file is ignored, but required. Output file is named < *filename*.rdw>, and its a copy from the input file where the read data substitutes the ignored data from the input file. In version 2.4 comment lines are accepted. Comment lines must start with the character '#'. In version 2.5 the third field has been redefined. In this case we'll use the character *w* for writing and the character *r* for reading

Example:
#write 5 in address 233
233 5 w
#read from address 138
138 0 r
#in the output file we obtain 138 35 r

Acknowledgements

To Sergio Cuenca Asensi from Universidad de Alicante (SPAIN) who pointed me towards the way of reading from the board. Xilinx and XESS and their many kind people who are always helping us, specially to Dave Vanden Bout and Anna Acevedo.

Sevilla, July, 2000