J.Natn.Sci.Foundation Sri Lanka 2011 ***39*** *(3): 261-266*

SHORT COMMUNICATION

A simple reconfigurable microprocessor in a 36 macrocell CPLD

W.A.S. Wijesinghe[1], M.K. Jayananda[2] and D.U.J. Sonnadara[2*]

[1] *Department of Electronics, Faculty of Applied Sciences, Wayamba University of Sri Lanka, Makandura.*

[2] *Centre for Instrument Development, Department of Physics, Faculty of Science, University of Colombo, Colombo 03.*

Revised: 14 August 2010 ; Accepted: 17 December 2010

Abstract: This paper describes a simple microprocessor developed using a complex programmable logic device (CPLD), with an instruction set optimized for data acquisition applications. The processor encompasses a tiny instruction set having only the instructions required in data acquisition applications. Due to optimization of the features, it was possible to fit both the CPU and the programme memory in the 36 macrocell Xilinx XC9536XL CPLD. The designing of the CPU was carried out using the hardware description language VHDL. The reconfigurability of the CPLD using VHDL enables the change of features of the CPU, including the instruction set, to suit user requirements. An example data acquisition system implemented using this CPU is also discussed.

Keywords: CPLD, data acquisition, FPGA, microcontrollers, reconfigurable computing, VHDL.

INTRODUCTION

Computerized data acquisition systems are frequently employed in modern research and industrial applications as well as in laboratories of academic institutions. However, most commercially available data acquisition systems are expensive and designed for a specific task and therefore not suited for academic purposes. With the available knowledge in the public domain, low-cost data acquisition systems based on microcontrollers can be developed for academic purposes. This paper describes an attempt to develop low cost data acquisition systems with increased flexibility. By developing a microprocessor with the minimum features required in such applications, the study also provided training in logic design for configurable hardware using hardware description language VHDL to students specializing in electronics or computing.

The development of complex digital circuits such as microprocessors without sophisticated chip manufacturing facilities has been made possible with the advent of programmable logic devices (PLD). Among the various types of PLDs, complex programmable logic devices (CPLD) and field programmable gate arrays (FPGA) with a large number of logic gates are the ones that are most suitable for complex circuits. Methods of implementing data acquisition systems on FPGAs capable of real-time performance are already available in literature (Brown, 1996; Ramanathan *et al.*, 2001).

A programmable logic device (Van Den Bout, 1999) is a device with configurable logic and flip-flops, which can be interconnected by the user. The basic building block of a CPLD is a macrocell. Each macrocell contains a user programmable combinatorial logic function and a flip-flop. A group of macrocells is arranged as a single configurable function block. The outputs of the macrocells are available through I/O pins of the CPLD. In addition they are fed to a global switch matrix, which interconnects the macrocell outputs to the macrocell inputs of all the function blocks. By programming the individual logic function of each macrocell and configuring the interconnections through the switch matrix, the user can build very complex multilevel logic functions.

For example, the XC9536XL used in this work consists of two configurable function blocks, each containing 18 macrocells (Xilinx Data sheet, 1999). A total of 36 external I/O pins and 36 inputs from the two function blocks (i.e. outputs from the 36 macrocells) are connected to the switch matrix while each function block receives 54 inputs from the switch matrix. The macrocells and the switch matrix can be programmed by downloading a bit pattern through the joint test

[*]Corresponding author (upul@phys.cmb.ac.lk)

action group (JTAG) port to the in-system programming controller.

A variety of tools are available for the CPLD design. They can be broadly categorized into two groups as schematic-based tools and hardware description language based tools. Available hardware description languages include VHDL, Verilog and ABEL. It is also possible to design systems by mixing the two approaches.

In the work described in this paper, the VHDL approach was used. The design was developed using (ISE) WebPACK software available free of charge from Xilinx (USA). The configuration of the device was done by downloading the binary file generated by Webpack through a Xilinx parallel port cable connected to the JTAG port of the CPLD.

METHODS AND MATERIALS

In most low-cost data acquisition systems, the main function of the microprocessor or microcontroller is to read data from one or more analog-to-digital converter (ADC) channels and send them to a computer or to store them in an on-board memory for later retrieval. Most of the instructions available in a typical microcontroller are usually not required for such applications. The most important instructions required are memory read and memory write. With memory mapped I/O, the same instructions can be used for accessing ADC and digital-to-analog converter (DAC) channels.

The VHDL implementation of an 8-bit minimal CPU described by Boscke (2002) was adopted as the basis for a very simple processor with a tiny instruction set.

In the initial design, the instructions were encoded using two bits, which made the maximum size of the instruction set to be four. However, only three instructions were actually required, namely, RD: read 8-bit word from the ADC and store it in data register, WR: send the value in data register (DR) to the memory module, and JP: set programme counter (PC) to zero.

The register set of the CPU and the data flow within the CPU is shown in Figure 1. The PC holds the address of the next instruction to be fetched. Just before an instruction is fetched, the content of the PC is transferred to the address register (AR) and the PC is incremented by one. After fetching, the instruction is loaded to the instruction register (IR). Data read from the memory (by RD instruction) or the data to be written to memory (by WR instruction) is stored in DR. The register MA contains the address of the next free location of data memory.

The operation of the CPU is controlled by a simple state machine consisting of three states (see Figure 2). The three states represent the fetching, decoding and execution phases of the instructions.

The implementation of the CPU required only a small programme memory. In the initial design, the size of the memory was limited to four two-bit words. The incorporation of memory inside reduces the external device count and increases the speed of programme execution. Although this constitutes to hard-wiring the

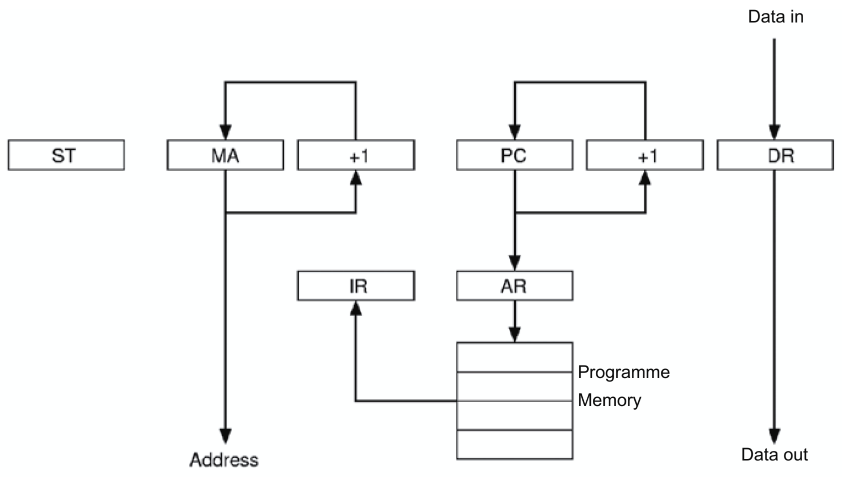

Figure 1: The register set of the CPU. AR: 2-bit address register, DR: 8-bit data register, IR: 2-bit instruction register, PC: 2-bit programme counter, ST: 2-bit to hold the state of the process, MA: 6-bit to hold the address of the next data memory location

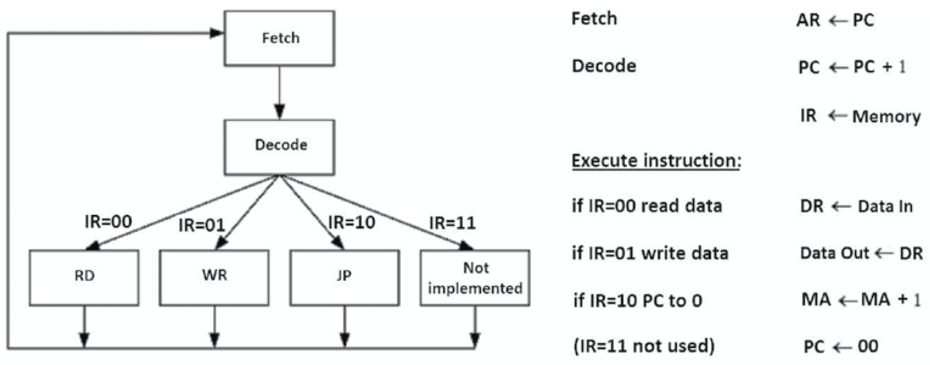

Figure 2: The operation of the CPU as a state diagram

programme into the CPU, there is no loss of flexibility because the CPLD can be easily reconfigured.

RESULTS AND DISCUSSION

The CPU described above was incorporated in a prototype data acquisition system to evaluate the performance. The data acquisition system comprises of three basic components - the CPU, a memory module and an analog-to-digital converter (ADC). A block diagram of the data acquisition system is shown in Figure 3. The programme stored in the programme memory of the CPU repeatedly reads the ADC and stores the data in the memory module. The ADC is configured so that each read operation initiates a new conversion. Each time a data word has been stored in the memory module, the memory address register is incremented by one. The programme used for this process consisted of just three instructions: RD, WR and JP.

A 6116 SRAM chip, which contains a 2K×8 memory arrangement was used for the memory module. The 6116 has eight I/O lines and 11 address lines. As shown in Figure 3, all 8 data lines and 5 of the address lines are connected to the CPU. The chip select (CS), write enable (WR), and output enable (OE) pins are also connected to the CPU.

An 8-bit AD7569 ADC was used in an infinite sample-and-hold configuration. The sample signal is sent by the CPU. It drives the CS, OE and inputs of the AD7569. Although the CPU was capable of reading data at a few MHz, the sampling rate of the AD7569 is limited to about 500 kHz. Therefore, to ensure that the ADC

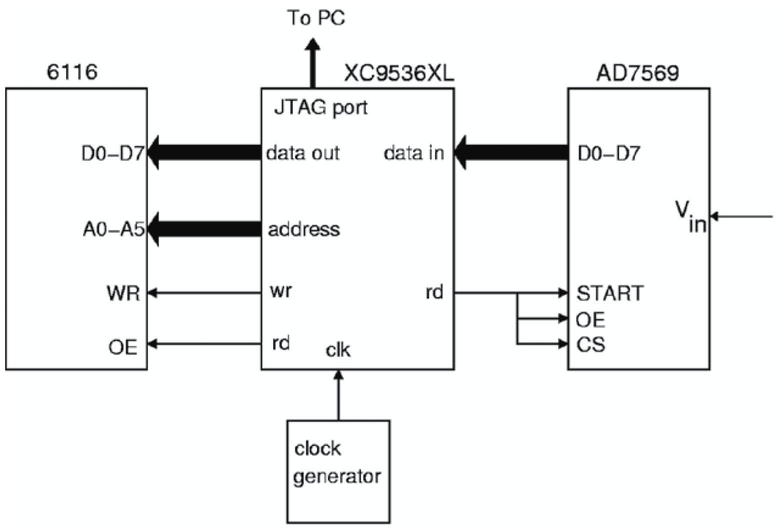

Figure 3: Block diagram of the data acquisition system

has completed before reading data, a delay of 3 ms was incorporated into the RD instruction. This brought the sampling rate of the ADC to approximately 333 kHz.

This data acquisition system can function as a standalone device and record data until the memory is full. The JTAG boundary scan (IEEE standard 1149) available in the XC9536XL was used to transfer the saved data to a PC. The main purpose of the JTAG boundary scan is to provide a convenient method of diagnosing problems in complex circuits. Chips that support this standard allow the isolation of the core of the chip from its pins, the setting of values to output pins and the reading of the status of input pins using commands sent to the chip through the JTAG port.

Making use of this facility, the data transfer to the PC was carried out by connecting the JTAG port to the parallel port of a PC using the same cable utilized for configuring the CPLD. Each word stored in the 6116 SRAM was read to the PC using the boundary scan commands.

The major advantage of CPLD based design is the possibility of reconfiguring the CPU to suit user requirements. Features such as the number of registers, their size and the size of the programme memory can easily be changed to suit different applications. The implementation of the CPU described in this paper did not require all the resources available in the XC9536XL. The remaining resources can be used to add more features to the CPU. If more than the remaining amount of resources are necessary to add a new feature, the total gate count can be doubled by removing the XC9536XL from its socket and inserting a XC9572XL, without any other hardware change.

CONCLUSION

The CPU described in this paper provides a low cost alternative to general purpose microcontrollers of data acquisition systems used in academic institutions for training purposes. Even the simplest microcontrollers contain many features that are not used in simple data acquisition applications. The CPU described above has only the bare minimum features required in such applications. This project was initiated in order to introduce hardware level concepts and training of logic design for configurable hardware using VHDL for students specializing in electronics and computing.

The incorporation of the programme memory into the CPU was an important feature of this work. The stored programme concept introduced in the very early stages of the development of computers vastly increased the versatility of computers. Although, at first glance, hardwiring of the programme into the CPU would appear as a retrograde step, abandoning the stored programme concept does not affect the versatility of CPLD based CPUs because they can be reconfigured easily by downloading a new bit pattern. The reconfiguration of a CPLD is almost identical to the process of downloading a programme to a microcontroller based system.

In the prototype data acquisition system described here, advantage was taken of the JTAG boundary scan support available in the XC9536XL for transferring data to a PC. The alternative to this would have been the implementation of a serial port, resulting in much higher resource requirements for the CPLD.

In addition to the training in logic designs for configurable hardware, the CPU described in this work was found to be very useful to train students on simple data acquisition concepts. The lack of complex features makes it easier to understand the basic operation of a CPU. Ease of reconfiguration allows students to learn computer architecture by changing various features of the CPU.

Acknowledgement

Financial assistance by the National Science Foundation of Sri Lanka (research grant RG/2007/FR/03) and the Wayamba University of Sri Lanka (research grant RG/2005/07) are acknowledged.

REFERENCES

1. Boscke T. (2002). A minimal 8Bit CPU in a 32 macrocell CPLD. Available at *http://www.csie.ntu.edu.tw/~b92029/data/EXP/mcpu-doc.pdf* , Accessed June 2007.
2. Brown G. (1996). User-configurable data acquisition systems. *Proceedings of the International Society for Optical Engineering (SPIE): High-Speed Computing, Digital Signal Processing, and Filtering using Reconfigurable Logic* (eds. J. Schewel, P.M. Athanas, V.M. Bove & J. Watson), volume 2914, Boston, USA, 20-22 November. The International Society for Optical Engineering (SPIE), Bellingham, USA, pp. 54-64.
3. Ramanathan A., Tessier R., McLaughlin D., Carswell J. & Frasier S. (2001). Acquisition of sensing data on a reconfigurable platform. *Proceedings of the IEEE International Symposium on Geosciences and Remote Sensing (IGARSS),* volume 1, Sydney, Australia, 9-13 July, pp. 222-223.
4. Van Den Bout D.E. (1999). *The Practical Xilinx Designer Lab Book*. Prentice-Hall Publishers, New York, USA.

5. Xilinx Data Sheet (1999). FastFLASH XC9500XL high performance CPLD family. Available at *www.xilinx.com/support/documentation/data_sheets/ds054.pdf,* Accessed June 2007.

ANNEX

The complete VHDL code of the CPU:

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use IEEE.std_logic_arith.all;

entity DAQ_CPU is
port (
   DataIn: in STD_LOGIC_VECTOR (7 downto 0);
   clk: in STD_LOGIC;
   reset: in STD_LOGIC;
   DataOut: out STD_LOGIC_VECTOR (7 downto 0);
   Address: out STD_LOGIC_VECTOR (5 downto 0);
-- Address line
   ADC_Sample:out STD_LOGIC;
            -- Sampling signal to the ADC
   OE_SRAM:out STD_LOGIC;
            -- SRAM output enable
   WR_SRAM: out STD_LOGIC
            -- Memory write signal to the SRAM chip
);

attribute pin_assign: string;
attribute pin_assign of clk: signal is "33";
attribute pin_assign of reset: signal is "34";
attribute pin_assign of DataIn: signal is "22,20,18,14,
12,13,11,9";
attribute pin_assign of DataOut: signal is "37,38,39,
43,1,3,8,7";
attribute pin_assign of WR_SRAM: signal is "29";
attribute pin_assign of ADC_Sample: signal is "35";
attribute pin_assign of OE_SRAM: signal is "25";
attribute pin_assign of Address: signal is "40,42,44,
2,4,6";

end DAQ_CPU;

architecture Archi_DAQ_CPU of DAQ_CPU is

signal DR: std_logic_vector(7 downto 0);
signal TmpOut: std_logic_vector(7 downto 0);
signal PC: std_logic_vector(1 downto 0);
signal AR: std_logic_vector(1 downto 0);
signal IR: std_logic_vector(1 downto 0);
signal states: std_logic_vector(1 downto 0);
signal AddCnt: std_logic_vector(5 downto 0);
            -- Counts the address value
signal SPulse: std_logic;
            -- ADC sampling pulse

--ROM
type ROM_ARRAY is array (0 to 3)of std_logic_vector
(1 downto 0);
constant ROM:ROM_ARRAY:=(
      0 => "00",
      1 => "01",
      2 => "10",
      3 => "00");
begin

process(clk,reset)
begin
  if (reset='0')then
     states<="00";
     AR<=(others=>'0');
     IR<="00";
     DR<=(others=>'0');
     PC<=(others=>'0');
     AddCnt<=(others=>'0');
     Spulse<='1';
  elsif rising_edge(clk)then
     if (states="00")then -- Fetch 1
        AR<=PC;
        SPulse<='1';
       states<="01";
     elsif (states="01")then -- Fetching an instruction
       PC<=PC+1;
       states<="10";
     elsif (states="10")then -- Executing the instruction
       states<="00";
       case IR is
        when "00"=>SPulse<='0';DR<=DataIn after 2 ms;
        --Read the input port
        when "01"=>TmpOut<=DR(7 downto 0);-- Writing
        to the output port
        when "10"=> PC<="00";AddCnt<=AddCnt+1;
        when others=>null;
       end case;
     end if;
  end if;-- end rising edge
  if( states="01")then
     IR<=ROM(conv_integer(AR));
  end if;

end process;
DataOut<=TmpOut;
```

```
Address<=AddCnt;
OE_SRAM<='1';
ADC_Sample<=SPulse;
WR_SRAM<='0' when(clk='0'and reset='1' and states="00" and IR="01")else '1'; -- Memory write pulse

end Archi_DAQ_CPU;
```