Programming environment for ARM using FPGA

Chetan Mittal, 2016CS10343

1 Introduction

The objective of the project is to design and implement a complete ARM CPU based computer. While some work has already been done, the project focuses on completing the ARM system with a keyboard and monitor.

2 Modules Completed Previously

Overview
The modules are designed for a Processor for the ARM language which is written in VHDL. It works on 32 bit instructions and can implement the following instructions:

- $Branch- < b|bl > \{cond\} < destination >$
- $Add/Sub- < add|sub|rsb|adc|sbc|rsc > \{cond\}\{s\}Rn, Rd < Op2 >$
- $Logical- < and|orr|eor|bic > \{cond\}\{s\}Rn, Rd < Op2 >$
- $Test- < cmp|cmn|teq|tst > \{cond\}Rn < Op2 >$
- $Move- < mov|mvn > \{cond\}\{s\}Rd < Op2 >$
- $Multiply- < mul|mla > \{cond\}\{s\}Rd, Rm, Rs$
- $Load/Store- < ldr|str > \{cond\}\{b\}Rd, \# < Imm12 >$

Components
The section describes the details of the various components :

- Computer : This provides an interface between the Switches, Seven Segment Display (SSD), LED's of the FPGA board and the processor. It has the components of Processor, Data Memory, LED interface, SSD interface and the Switch Interface connected via the AHB Lite bus.
- Processor : This is the main processor for the ARM instruction set. It consists of the Datapath and the Controller.

- Datapath : The components required for the elementary operations (ALU, Register File, Shifter, Multiplier, Processor Memory Path (PMP)) are connected in the Datapath with the Instruction Memory with control signals coming from the Controller. Datapath also has the registers to store the flag values.
- Controller : It connects the components of Instruction Decoder, ALU controller, Flag Checker, State Controller and sends out the control signals for the Datapath depending upon the current state.
- Instruction Decoder : It gives the type (DP, DT, MUL/MLA, B/BL), subtype (DP : test/cmp, arith/logical; DT : ldr, ldrh, ldrb, ldrsh, ldrsb, str, strh, strb; MUL/MLA : mul, mla; B/BL : b, bl) and the variant (imm, imm specified shift, reg specified shift) of the intruction.
- State Controller : It specifies the next state depending on the current state and the instruction type.
- ALU controller : It gives out the control signals for the ALU.
- Flag Checker : It specifies if the condition is satisfied by the flag values or not.

3 Interfacing with PC

Interfacing with PC is required to enable usage of keyboard and monitor via UART.

3.1 Features

The following features are proposed to be implemented :

- Allowing standard input from the keyboard and standard output to the monitor, with ARM instructions to handle the same.
- Ability to write ARM code using the keyboard, store it in the Instruction Memory and execute the code, with its output on the monitor.
- Basic Operating System to guide the user and allow switching between the environment to write the ARM code and the environment for the execution of the ARM code.

3.2 Required Modules

- **UART Interface** : Interface to take input from the keyboard and store it in a buffer on the FPGA board, another buffer on the FPGA to get data from the processor and display it on the PC monitor (Hyperterminal).

- **SWI instructions** : Instructions to read data from the input buffer and transfer them to the register file of the processor, Transfer data from the register file to the output buffer. The following instructions will be implemented :
  1. To display a character on the monitor
  2. To clear the Hyperterminal
  3. To take character input from the keyboard

  Additional instructions may be implemented if required

- **Basic Operating System** : It is required for the following purposes :
  1. Startup code to display initial text on the Hyperterminal
  2. Provide two different working environments, one for writing ARM code, another for executing the written code.
  3. Switching between the two environments.

  The code for the OS will be stored in a predefined portion of the Instruction Memory and the execution of the processor will begin from the startup code.

4 Implementation Details

4.1 UART

A UART is one method of talking to the FPGA. It can be used to send commands from a computer to an FPGA and vice versa. A UART is an interface that sends out usually a byte at a time over a single wire. The serial line is '1' when it is idle. The transmission starts with a start-bit, which is '0', followed by data-bits and an optional parity-bit, and ends with stop-bits, which are '1'.

Most UART implementations use a serial terminal to transmit the ASCII code of the characters input into the terminal. It is proposed to store the 8 bit ASCII code in a register (input buffer) on the FPGA board, and refresh the register every time a new character is input into the terminal. Also, a refresh bit needs to be stored which toggles every time the register is refreshed. The processor is supposed to read the data from the buffer using polling, and if the polling is frequent enough, no data bits would be missed out. This polling is proposed to be carried out in software. Similarly, a character can be displayed on the terminal by sending the ASCII code of the character from the processor to the PC. The data to be output is to be stored in a register (output buffer) by the processor before transmission.

A UART interface needs to be implemented to provide a simpler abstraction of data input and data output to the processor.

4.2 SWI Instructions

The Processor needs to allow instructions to input and output using the PC. Atleast two instructions are required to be implemented, one to receive data from PC and another to transmit data to PC.

The read instruction should read the data from the input buffer only when the refresh bit has toggled (It needs to maintain a copy of the refresh bit) and store it in a register specified in the instruction. Since the data is of 8 bits while the register is of 32 bits, 0's may be appended to the beginning of the 8 data bits.

The write instruction would store the least significant 8 bits from a specified register to the output buffer and initiate the transmission of data from FPGA to PC. Special output instructions may be implemented (eg : clear the terminal) by giving special meanings to some patterns of the ouput data bits (i.e., the allowed ouput data bits are limited and must be specified).

4.3 OS

Operating System is required to provide a user interface and handle the input and output operations. Two working environments are proposed to be implemented, one for writing code, and another for program execution.

Upon startup, guiding text would be output on the terminal and the user should be in the coding environment. In this environment, the OS should repeatedly poll for character inputs and store the valid inputs in the memory. Also, since the ARM instructions are encoded in 32 bits, whereas the input is in the form of 8 bit ASCII codes, the OS should convert a valid sequence of characters to the 32 bit ARM instruction. The OS should also notify the user of an invalid instruction (Sequence of characters before an escape character is received). The OS should also switch the environment to program execution mode when a special input sequence is recieved.

The OS should begin the execution of the instructions written by the user. In this mode, polling for an input must be done by the user program. Also, switching back from the execution mode to the coding environment must be specified by an instruction in the user program.

5 Proposed Workflow

- Testing and debugging the already implemented instructions
- Implementing the UART interface
- Adding SWI instructions to the processor
- Adding OS (System code) and testing

6 References

[1] $http://www.dejazzer.com/eece4740/lectures/example\_uart\_echo.pdf$