COE4DS4 Lab #5
Debugging Techniques for Embedded Systems

Objective

To learn how to use the three debug techniques (hardware/software co-simulation, embedded logic analysis and software breakpoints) provided by three different tools - ModelSim-Altera, SignalTap and NIOS II software development kit (SDK) - to verify and validate an embedded system.

Preparation

- Read this document and get familiarized with the source code and the in-lab experiments

Experiment 1

The aim of this experiment is to get you familiarized with the co-simulation technique to verify both the software and hardware of an embedded design. Figure 1 shows how the different components from various tools link together for co-simulation. For the hardware side, using Qsys, a Verilog testbench is generated such that the NIOS processor-based system can be simulated using ModelSim-Altera. The testbench is responsible to drive the inputs (i.e., SWITCH_I and PUSH_BUTTON_I) and monitor the outputs (i.e., LED_RED_O and LED_GREEN_O) during simulation. To observe the output of STDIO driven by the NIOS processor through the JTAG UART module, a simulated JTAG terminal is provided in the simulation environment. On the other hand, using the NIOS IDE, the compiled software, which should be loaded into the instruction/data memory, is obtained.

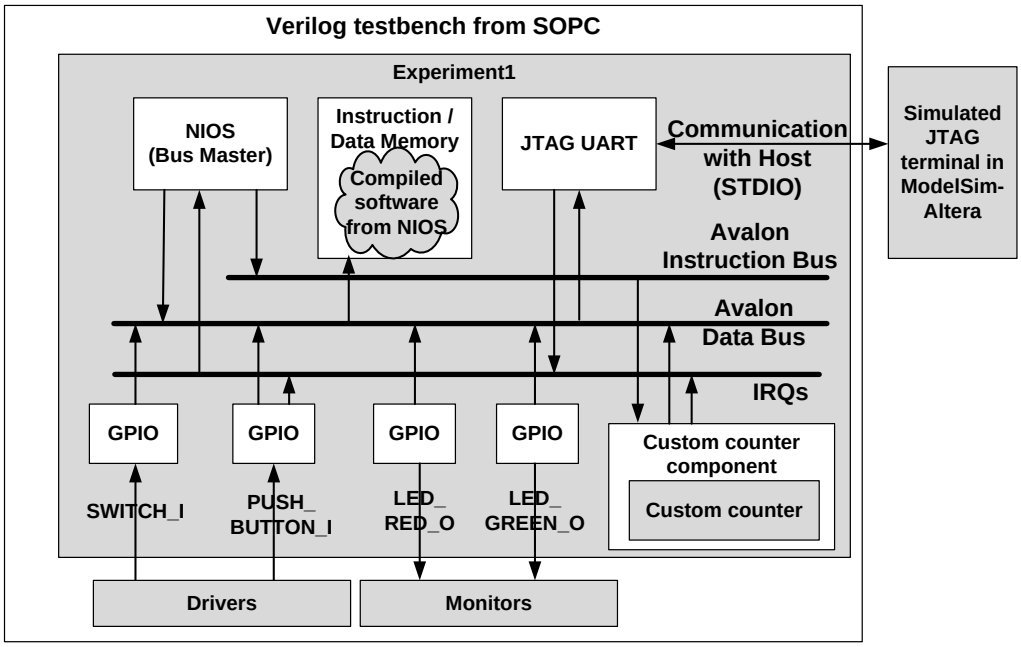

Figure 1: The system setup (both hardware and software) in the co-simulation environment.

The main advantage of being able to functionally simulate a NIOS processor-based embedded design is the ability to observe the internal signals in the entire system under the co-simulation environment. This allows interactions between the processor and the hardware modules to be verified when the software program is executed. However, this cycle-accurate visibility of the design comes at a price of very long verification time. For example, it takes a few minutes to simulate the simple design shown in Figure 1 for just a few milliseconds (ms) of real-time operation.

You have to perform the following tasks in the lab:

- Follow the instructions provided in the *readme.cosimulation.txt* to create a design to be simulated
- To monitor in the simulator when each character is sent to the JTAG UART, the `Table_Ascii_codes.pdf` file is provided in the "doc" sub-folder for your reference
- From studying the simulation waveform, in which any signals from the design can be monitored, understand the following:
    - How the testbench is used to drive input signals and monitor output signals
    - How the NIOS processor communicates with all the hardware modules through the bus
    - How does the execution of software on the NIOS processor interact with the hardware modules (specifically, the behavior of the JTAG UART module when messages are printed onto the terminal, and the behavior of the custom counter when driven by NIOS)

Experiment 2

Due to the long runtime of co-simulation, an emerging technology called ***embedded logic analysis*** can be used to monitor an embedded system in real-time. The embedded logic analysis exploits the fact that in programmable logic devices, such as field-programmable gate arrays (FPGAs), portions of embedded memory may not be used for the application at hand. Thus, this "spare" memory can be used for acquiring data on-the-chip. Acquisition is achieved when a "trigger" event occurs. This trigger event can be initiated by the user from the PC, or it can be detected internally based on the sequencing logic that is compiled into the design (see Figure 2 for the conceptual interface between a PC and the device under debug). After the internal "trace" buffers (i.e., the embedded memory blocks not used by the application) have been filled, the acquired data is off-loaded to the PC where it is displayed into a waveform window that resembles simulation. Altera devices can use embedded logic analysis through a tool called SignalTap.

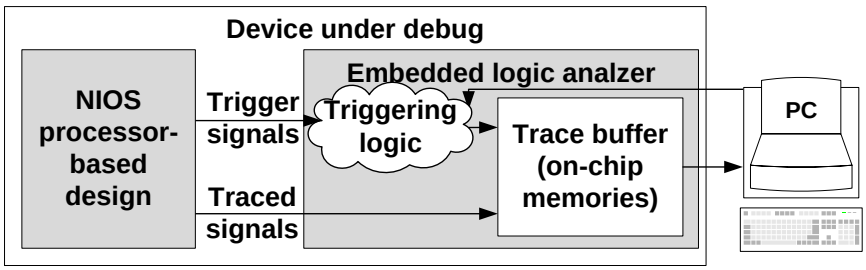

Figure 2: Validation and debugging of an embedded system using an embedded logic analyzer.

The main advantage of employing embedded logic analysis for debug is the capability to acquire data when the circuit is operating in real-time. However, it is obvious that because of the limited size of the on-chip memories in the FPGAs, the number of signals that can be monitored is constrained. This is why when building the embedded logic analyzer, a set of trigger signals are selected and programmed at runtime with the appropriate trigger condition to control *when* data should be acquired. Some examples of trigger conditions are: when a trigger signal has a positive edge transition, when a set of signals reaches a specific value, or a positive edge transition on one signal combined with a negative edge transition on another trigger signal. Also, the signals that are of interest are selected as trace signals. These signals are the ones that are sampled into the trace buffer when the programmed trigger conditions occur.

You have to perform the following tasks in the lab:

- Follow the instructions provided in the *readme.signaltap.txt* to insert an embedded logic analyzer using the SignalTap environment (check `SignalTap_userguide.pdf` for detailed features).
- Get familiarized with the SignalTap environment and understand the following:
    - How to select different sets of trigger signals and trace signals
    - How to setup different trigger conditions for acquiring different sets of data (e.g., trigger on interrupts or specific counter values)
    - How different features are beneficial during debug. These features include:
        - Changing the depth of the trace buffer
        - Changing the types of the trace buffer (i.e. continuous, pre-trigger, post-trigger)

Experiment 3

There are two main objectives for this lab experiment. First, you will be introduced to a new peripheral component used to write/read data to/from a secure digital (SD) card. Second, you will become familiar with the basic software debug techniques for embedded systems.

We use as a starting point the setup from the previous labs for streaming data from the digital camera to the synchronous dynamic random access memory (SDRAM) and from the SDRAM to the liquid crystal display module (LCD). The streaming of video can be started/resumed using the keys on the touchpanel (the commands are the same as in the previous labs and for the specific behavior you should check `LCD_Camera_TouchPanel.c`); the writing/reading of raw images in PPM format to/from the SD card is done using push buttons (for the specific behavior of each push button, you should check `PB_button.c`).

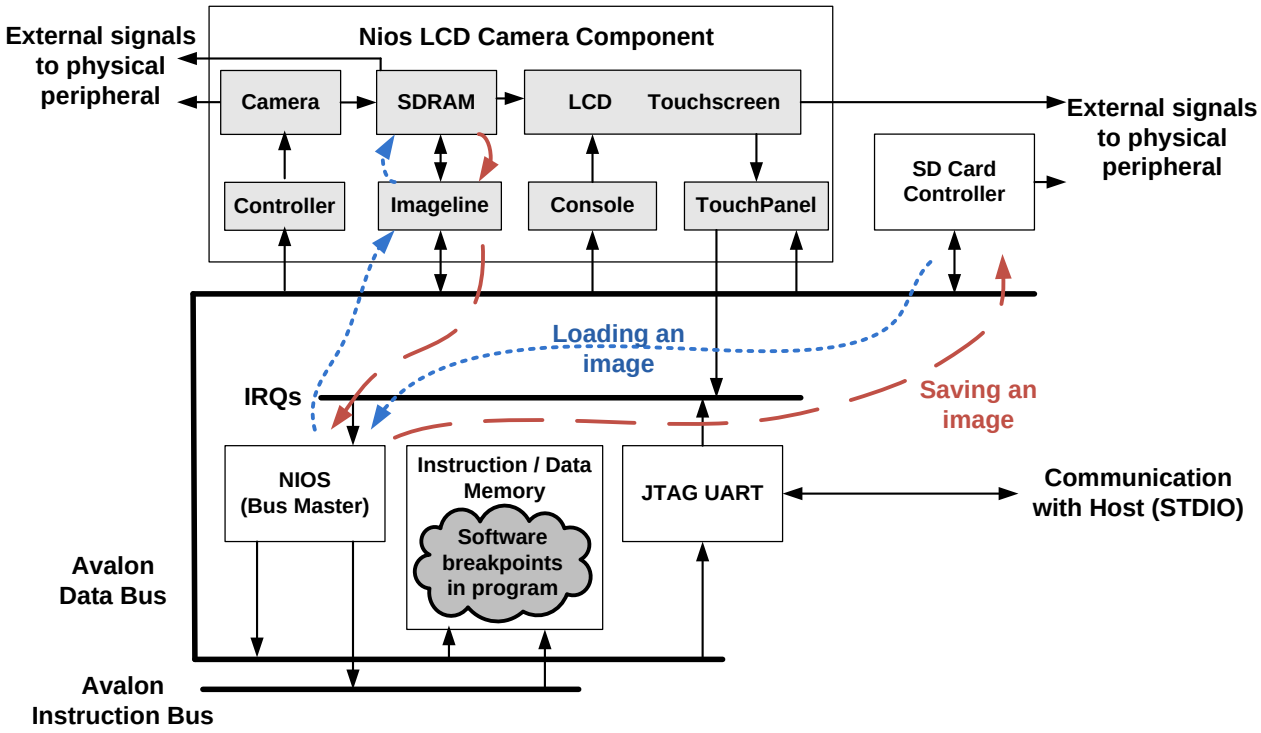

Figure 3: The embedded system setup to save/load images to/from the SD card.

The SD card controller is a pre-defined component provided by Altera and it is written in VHDL. Its physical interface (given as a reference in `Spec_SD.pdf`) is abstracted from the user who is provided with a set of functions, such as `sd_card_read` and `sd_card_write` for accessing the data on the SD card (for a complete list refer to `SD_card_controller.h`). The data on the SD card is organized according to the FAT16 file system format (for the full spec, again, abstracted from the user, refer to `Spec_FAT.pdf`).

You have to perform the following tasks in the lab:

- Follow the instructions provided in the *readme.software_debug.txt* to get familiarized with the NIOS IDE debug environment (check `NIOS_debugger_userguide.pdf` for detailed features)
- As you get familiar with the design, understand the following:
  - How to capture an image and write it to the SD card in PPM format
  - How to read the captured image from the SD card and display it on the LCD
  - How to insert breakpoints in the source code and use the stepping features in the debugger to step through/over/out of functions
  - As a case study, you should set the breakpoint such that the program execution is suspended each time a row of image data is written into the SDRAM

Write-up Template

COE4DS4 – Lab #5 Report
Group Number
Student names and IDs
McMaster email addresses
Date

There are 3 take-home exercises that you have to complete within one week.

Exercise 1 (1 mark) – Modify the *experiment1* as follows. The message to be displayed in the interrupt service routine that is executed when the custom counter expires should be changed to print the name of your favorite "Harry Potter Character" (or alternatively, name of your favorite Sci-Fi movie). In your simulation you should monitor and display in the transcript window the exact times when the first and last characters of the Soccer team are displayed. In the write-up you should explain how much time it takes to service the interrupt. Submit your sources and in your report write a quarter-of-a-page paragraph describing your reasoning.

Exercise 2 (5 marks) – Extend the *experiment2* as follows. You should implement a custom component for finding the largest and the second largest values in an array of 256 elements represented on 16-bits as signed numbers. The circuit for finding the largest and the second largest values, together with a 256 x 16 bit dual-port embedded memory, should be enclosed in a hardware component (with the following *offset* registers), which is connected to NIOS using Qsys:

- *offset* 0 – is used to provide the write data (bits [15:0]) and the address for the embedded memory (bits [23:16]) where the write data is written if bit [24] is a 1; if bit [24] is 0, the data from embedded memory location given by bits [23:16] can be obtained by doing a subsequent read from *offset* 1
- *offset* 1 – provides the read data (16-bit value) from the memory at the location defined at *offset* 0
- *offset* 2 – holds the largest value which must be read after the interrupt was generated
- *offset* 3 – holds the second largest value which must be read after the interrupt was generated
- *offset* 4 – clear the interrupt which is asserted when the two largest values are found
- *offset* 5 – when changing from 0 to 1 on its LSB, hardware circuitry starts searching for largest and the second largest values

NIOS is responsible for randomly generating the data in the array and this data should be transferred to the embedded memory in your custom component before searching is initiated. NIOS will need to access only one port of the dual-port memory. The custom hardware for searching the largest and the second largest values should access the other port. After searching is initiated by NIOS, the custom circuitry (described by you in Verilog) will have control over the embedded memory until the largest and the second largest values are found. NIOS must be notified through an interrupt when searching is complete, so it can read back the largest and second largest values. These two values should be printed by NIOS in the host terminal. Submit your sources and in your report write a half-a-page paragraph describing your reasoning.

Exercise 3 (3 marks) – Modify the *experiment3* as follows. In the lab you are manipulating images in the PPM format. Using the information provided in the `Spec_BMP.pdf` file, you should write/read files to/from the SD card in the BMP format, which is supported by most image viewers, including Microsoft Paint. Note, only images of size 320 x 240 need to be supported by your implementation. When displaying a 320x240 image from the SD card, you should do so in the "center" of the 640x480 viewing area of the LCD screen.

When storing an image on the SD card, the 640x480 image buffered in the SDRAM should downsampled by simply removing every odd row and column. Note, the multi-byte fields in the BMP files (both header and the data region fields) are written in the little-endian format. For example, if the width of the image is 320 in decimal, this will amount to 140 in hexadecimal. This hexadecimal number written on 4-bytes in little-endian will be 40 01 00 00 (i.e., the least significant byte is first). Another key point about BMP files is that if the height is positive than the last row from the image will be stored first in the data region of the BMP file. However, if the height is set as negative, i.e. -240 in the description from `Spec_BMP.pdf`, then the raw data is written starting from the first line (as it is common for the raster image formats). Hence, because -240 in decimal is F10 in hexadecimal, when written in little-endian on 4-bytes, the value 10 FF FF FF is written in the BMP file header. Submit your sources and in your report write a quarter-of-a-page paragraph describing your reasoning.

VERY IMPORTANT NOTE:

This lab has a weight of 9% of your final grade. The report has no value without the source files, where requested. Your report must be in "pdf" format and together with the requested source files it should be included in a directory called "coe4ds4_group_xx_takehome5" (where xx is your group number). Archive this directory (in "zip" format) and upload it through Avenue to Learn before noon on the day you are scheduled for lab 6. Late submissions will be penalized.