Lab Assignment 10
Controlling a Robotic Arm via a WiiMote through the ARM CPU and the FPGA

The goal of this lab is to bring together all of the elements you have developed so far in previous labs, and combine them in a hardware/software co-designed implementation. In this lab you will control the Robotic arm using the FPGA (as you did in Lab 9) and use the ARM processor on the ZedBoard to receive commands from the WiiMote and control the FPGA. You will be writing a C program that receives signals from the WiiMote connected via Bluetooth and sends commands to the FPGA to carry out a specific movement by the robotic arm. The FPGA will generate the necessary PWM signals to control the Robotic arm to carry out a specific action involving multiple servo motors.

Instructions

- Each lab assignment consists of a set of pre-lab questions, the actual time in the lab, and a lab report.
- The pre-lab assignments prepare you for the challenges you will be facing in the actual lab. So that each lab member gets the same benefits, pre-lab assignments have to be solved individually. Feel free to discuss problems with class and lab mates, but complete the assignment on your own.

Pre-Lab Assignment Part A

We have only very limited time for the lab available. To make efficient use of the lab time, you will need to prepare for it. First, go through the assigned reading above and get an overview of the schematics and the manuals mentioned. Second, approach the questions below to apply your knowledge. Please submit the solution to the pre-lab assignments in PDF format via blackboard.

Pre 10.1 Simulation of PWM Generation

Download and open the template design from Blackboard (ServoControl.slx) for generation of the PWM signals for the servos. This design is the essentially same as the design you developed in Lab 9. However, instead of counters for controlling the duty cycle, it has constants (which will be mapped to MMRs (Memory-Mapped Registers) when synthesizing to the ZedBoard). It takes 5 uint8 inputs as the position of 5 servo motors and generates the 5 PWM signals.

Use 5 constants to drive the inputs and a scope to view the outputs. You can change the scope settings to view 5 outputs at the same time. Double click on the scope, then click on the "Parameters" icon. In the "General" tab, change the "Number of axes" to 5. Also, "History" tab, uncheck the option "Limit data points to last". Run the design for at least 2000000 cycles and report the results.

Pre 10.2 Compute Interpolation Steps for Speed Control in Software

As part of this lab, you will use software to control PWM signal generation on the FPGA. The goal of this pre-lab assignment is to write a function that continuously modifies the duty cycle to implement speed control of the servo motor.
The servo motor gets a new PWM signal every 20ms. Write a function which updates the duty cycle every 20ms based on the desired rotational speed. As an example, with a speed of 18 degrees/s, in each second the servo moves by 18 degrees. Consequently, it takes 10 seconds for traveling from 0 angle to 180 angle. The C function should follow the declaration below

```
/**
 * Move servo given a speed.
 * @param servoNr         selected servo number
 * @param from            start position (0-180)
 * @param to              end position (0-180)
 * @param speed           speed (degree/sec) >0
 */
void servoMove(unsigned int servoNr, int from, int to, int speed);
```

Validate the functionality of the C function by printing out the duty cycle every 20 ms. Submit your code, as well as a demo execution for:

```
    servoMove(0, 10, 100, 36);
```

Note: for the purpose of validation you can ignore the servo number. It will be added in the lab assignment.

Pre 10.3 Combining Values in an Integer
Write a line of C code that combines two unsigned char values (*valLow*, *valHigh*) into one unsigned integer (*value*). *valLow* should be stored in the least significant byte (bit 0 - bit 7) of *value*. *valHigh* should be stored in the next more significant byte (bit8 - bit 15). The remaining bits in *value* should be 0 (bit 16 - bit 31).

Pre-Lab Assignment Part B

Pre 10.4) Download the reference code wiimote-skel.c from blackboard. The code currently uses hardcoded values for each button code. Improve the reliability and readability of the code by creating an enumeration for the button code. Please see:

http://crasseux.com/books/ctutorial/enum.html

for a reference and how to use enumerations. Conversely the button code values into an enumeration.

Lab Assignments Part A

In the Lab 9, you developed a Simulink model and downloaded it to the FPGA to control the Robotic arm. In this laboratory, you will be controlling the Robotic arm using the Programmable Logic (PL) side of the ZedBoard, but the FPGA will be under the control of the Processing System (PS) side of the ZedBoard (i.e., the ARM CPU). For this, we will use the AXI lite to connect a memory mapped register which then defines the duty cycle.

This lab has three parts:

1. Generate PWM in hardware, control speed in software.
2. Generate PWM and speed in hardware, controlled by software
3. Control SW in Part 2 with WiiMote.

Lab 10.1 Speed Control in Software

1. Download and open the template design from Blackboard (ServoControl.slx). This design is the essentially same as the design you developed in Lab 9. However, instead of counters for controlling the duty cycle, it has memory-mapped registers that are controllable by the AXI lite. It takes 5 uint8 inputs as the position of 5 servo motors and generates the 5 PWM signals.

   Note - You can use the ServoControl.slx or your own model that you developed in Lab-9. You will get extra credit if you use your own model instead of the model provided (10pts). If you reuse your own model, replace the *SetCounter* with an input of type unit8. The next steps will guide through how to access the values.

2. Replace constants with Inputs and compile the design using HDL workflow advisor. In the *Set Target Interface*, use AXI4-Lite as your inputs. Make sure to write down the address of each input. Connect the PMOD **A (not E)** connector to the robotic arm, as in Lab 9.

Table 1. Mapping of servos, connections/pins and MMR Addresses

Servo	Connector/pin	Address
Base	JA[0]	0x400D0100
Bicep	JA[1]	0x400D0104
Elbow	JA[2]	0x400D0108
Wrist	JA[3]	0x400D010C
Gripper	JA[4]	0x400D0110

3. Login as root on the Zedboard using mobaxterm.
4. Compile and run the C program (ServoControl_SW.c) provided in Blackboard to control the Robotic arm. The program takes two inputs from the user (servo number, and servo position), and writes these values to the appropriate MMRs. The FPGA then generates the appropriate PWMs based on the settings, effectively controlling the robotic arm.

Note -Make sure that the addresses for memory-mapped registers match those shown in Table 1, otherwise the C program needs to be updated with the correct addresses.

5. Validate that the program works, and report your findings and errors.
6. Modify the C program provided so that it controls the speed of one of the servo motors (e.g., the base). Use the results of Pre 10.2 to guide you. Test out the capabilities of this program using the lowest and the highest speed. Report your findings and errors.

Hint: How to Reprogram the FPGA without HDL Advisor

Going through the HDL Workflow Advisor until "Programming FPGA" performs two steps. It creates a bit file, which is basically the "program" for the FPGA, and programs (downloads and loads) the bit file into the ZedBoard. The FPGA configuration is lost upon power cycling the board.
After creating the bit file once, the FPGA can be configured directly without going through the HDL Workflow Advisor. Here are the steps:

1. Locate the bit file. It should be in the following directory structure:
   `hdl_prj/pa_prj/pa_prj.runs/impl_1/system_stub.bit`
2. Enter the following Matlab command to configure the FPGA:
   `hdlturnkey.tool.EDKTool.downloadBit ('system_stub.bit')`

Lab 10.2 Speed Control in Hardware / Predefined Motion

Controlling the speed in software as done above poses some challenges when multiple servos need to be moved at the same time. Since with the ZedBoard both hardware and software can be programmed, this lab assignment walks you through controlling the speed in hardware. Then, multiple servos may move at the same time and at their own speed.

7. Download and open the template design from Blackboard (ServoControlWSpeed.slx). Compare this design with the previous design (ServoControl.slx) and reason about the difference and functionality.
8. The design template ServoControlWSpeed.slx is designed for simplicity. However, it only allows a very coarse grained speed control. To allow more flexibility, we have an improved version. Please download **ServoControlWSpeed_x10.slx** and continue with this design template.
9. Compile the design using HDL workflow advisor. In the *Set Target Interface*, use the AXI4-Lite as your inputs. Use the following addresses:

Table 2. Mapping of servos, connections/pins and memory-mapped register addresses

Servo	Output connector/pin	AXI4-Lite Input Address Offset
Base	JA[0]	0x100
Bicep	JA[1]	0x104
Elbow	JA[2]	0x108
Wrist	JA[3]	0x10C
Gripper	JA[4]	0x110
ServoStatus	-	0x114

Note that the template has an additional output (ServoStatus), which will not be used for our lab, but it still needs to have an address assigned.

10. Download the template program for HW speed control (**ServoControl_HW.c**). Update the first line in the function servo_move, combining the control of the position and speed into a single integer (see Pre 10.3 for procedure).
11. Compile and run the C program to control the Robotic arm. The program keeps taking 3 inputs from the user (servo_number, servo_position, and speed), and writes these values into the address of the appropriate AXI4-Lite input. The FPGA reads these values and generates the appropriate PWM signals.
12. Write a C program to control the Robotic arm with a predefined motion. The goal is to pick up a box, and throw it.
    a. First, use the provided program to identify the position for each movement
    b. Then, create a set of servo_move statements to move the robotic arm accordingly.

    HINT: It takes the robot a while to move to the desired location. Consider this in the program.
13. Reflect on your findings in your report.

Lab Assignments Part B

This section introduces you to non-blocking and blocking file I/O. With a blocking read, the file read only returns (i.e., execution continues) after all the required bytes are available. However this is problematic if the code should read from multiple sources. If both reads are blocking, input from another source will be missed. The solution is to use non-blocking calls. Using a non-blocking call, the read will terminate (return) even if insufficient data is received. This allows the code to continue execution. However, the code should include error-handling logic to deal with receiving fewer bytes than expected.

Lab 10.3 Establish Bluetooth Connection to WiiMote

In Lab4 we read the WiiMote buttons, and the WiiMote acceleration separately. In this assignment, we will try to read the acceleration values continuously, while the button is pressed. As long as the button minus is pressed, the X acceleration should be displayed. While pressing the plus button, the Y acceleration should be displayed. If both plus and minus buttons are pushed, both X and Y should be displayed. The challenge is that the events come from different device files (buttons from event2, acceleration from event0). With blocking reads, it is impossible to read both at the same time. In the first part of this assignment you will see the effect of blocking calls, and then will make changes to turn the call into a non-blocking call.

14. Connect the WiiMote
    a. Connect the USB Bluetooth dongle (through the standard USB-to-micro USB adapter) to the ZedBoard in the micro USB port on the ZedBoard named "USB OTG" (the label is on the board).
    b. Log into the ZedBoard
    c. Start the Bluetooth adapter:
       `sudo hciconfig hci0 up`

15. Identify Your WiiMote via Bluetooth
    a. Place the Wiimote in pairing mode so that it can be found
       i. Remove the battery cover and push the red sync button.
       ii. The four blue LEDs on the Wiimote are blinking to indicate pairing mode.
       iii. Pairing mode is active for ~20 seconds (push red button again if you miss the window)
    b. Issue:
       `hcitool scan`
       The Wiimote should show up as "Nintendo RVL-CNT-01" and a MAC address (e.g., XX:XX:XX:XX:XX:XX). Note, you will see any Wiimote that is trying to sync at the same time. So, make sure you found yours, by checking the MAC address on the box.
    c. Store the WiiMote MAC address in a variable in the shell (you will need it multiple times in the future). Issue:
       `export WII_MAC=XX:XX:XX:XX:XX:XX`
       Replace XX:XX:XX:XX:XX:XX with the MAC address of your Wiimote. The variable is available as long as you stay in the same shell.

16. Start the Bluetooth service which provides standard operations between Bluetooth devices. Issue:
    `sudo /usr/sbin/bluetoothd`
17. Put the WiiMote into pairing mode again. To establish the connection, issue:
    `sudo bluez-test-input connect $WII_MAC`
18. The LED 1 on WiiMote should turn on.

Lab 10.4 Reading Button Events and Acceleration Simultaneously

1. Compile and run improved reference code (see Pre-Lab 10.3) wiimote-skel.c
2. Analyze the behavior. Can both buttons and acceleration values be read? What are the limitations? Please record your experience.
3. Change the file open for the buttons from:
   `gWiiMote.fileEvt2 = open(WIIMOTE_EVT2_FNAME , O_RDONLY);`
   to:
   `gWiiMote.fileEvt2 = open(WIIMOTE_EVT2_FNAME , O_RDONLY | O_NONBLOCK);`
   By adding the option O_NONBLOCK, a read call will return, even if we have not received the expected number of bytes.
4. Compile and run the modified example with a non-blocking read. Observe and describe the difference.
5. Describe the difference in operation (especially the error handling, if insufficient data is received) in your report. How does the code deal with the fact that no button press event has been received?
   Hint: To better understand the working principle, start looking at this line of code:
   `if (WIIMOTE_EVT2_PKT_SIZE == read(gWiiMote.fileEvt2, buf, WIIMOTE_EVT2_PKT_SIZE)) {`
   It requests to read `WIIMOTE_EVT2_PKT_SIZE` number of bytes from the file. The read call returns the number of bytes actually read. In case fewer bytes were read, the "if condition" follow the false branch. To indicate that fewer than requested number of bytes are received, the errno (system error number) is set to the constant EWOULDBLOCK.

Lab 9.5 Control Robot through Wiimote

This is an open-ended assignment. Your task is to design and realize how to control the robotic arm through the WiiMote. Use the code wiimote-skel.c for reading acceleration and buttons, and use the hardware speed control for controlling the robotic arm.

1. Design a mechanism to control the robotic arm. You can use buttons and acceleration as inputs. You can use all buttons and accelerometer values to control 5 servo motors. For example, you can use the 'X' value of the accelerometer to control the base, or '↑' and '↓' buttons to control the wrist.
2. Record design decisions of how you control the robot in your report.
3. Realize your design decisions by implementing them. Validate your design decisions by executing on the robotic arm. Please also record mistakes/improvements of your design decision.
4. Record a video of your robotic. This will earn extra credit (10 Pts).
5. Describe your findings in the report.

Laboratory Report

You should follow the lab report outline provided on Blackboard. Your report should be developed on a wordprocessor (e.g., OpenOffice or LaTeX), and should include graphics when trying to present a large amount of data. Include the output of compiling and running your programs. Upload the lab report on blackboard. Include each program that you wrote in the appendix of your report.