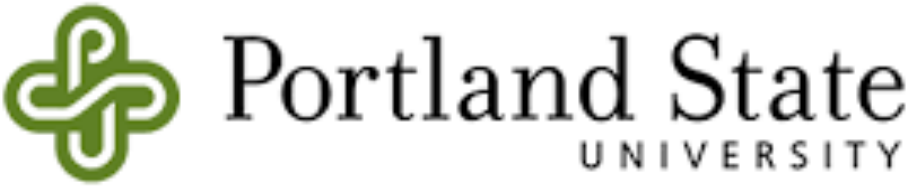

ECE – 540

PORTLAND STATE UNIVERSITY

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

SOC Design with FPGA

SIMPLEBOT Report

Deepen M Parmar
parmar@pdx.edu

Date: 25th January 2020

1. Introduction

The "Virtual" robot is a platform with two wheels, each driven by an independent motor and a third Teflon skid as a third free rotating wheel serve to stabilize the platform. This project models simple virtual robot using push buttons and the seven-segment display. The task is to implement the functionality of the robot using the Digilent Nexys A7 development board. This could be done by implementing the Motion Indicator and the Compass.
The project is divided into two phases

I. Hardware Development
II. Software Development

1.1 Hardware Development

- Design of an AHB-LITE bus peripheral, which drives HCLK, HWRITE, HADDR, HWDATA between the MIPSfpga and seven segments display peripheral. These signals are driven as the input signal to the peripheral.

- The seven-segment display is write only peripheral and not read/write peripheral thus there will be no change to the HRDATA multiplexer (read data from peripheral to CPU).

- The relevant addresses for the seven-segment enable, data point enable, lower display and the upper display are added to the constant header file and then are used directly to pass the data to these addresses to make display perform various operation.

- Then the address decoder will need to be updated to generate the HSEL signal which is used to select the peripheral when the corresponding address range of the seven-segment display is on the HADDR bus.

- The output signals of the seven-segment are brought upwards of the hierarchy by ensuring the proper connection at each level of the MIPSfpga structure to propagate the signal to the top level. In the top level these output signals are directly connected to the ports defined in the constraint file.

1.2 Software Development

- Implementation of the MIPS assembly code that drives the seven-segment display to perform the motion indicator and the compass operation.
- The pushbuttons are used to perform the various motions of the left and right motor.
- The motion indicator is rotated clockwise when the simpleBot is turning right and rotated anticlockwise when the simpleBot is turning left.
- The motion indicator blinks the forward segment (Segment A) when the simpleBot is moving forward and it blinks the reverse segment (Segment D) when the simpleBot is moving reverse
- When the simpleBot is stationary it displays the center segment (Segment G).
- The Compass indicates the degree of the bot by displaying value from 0^0 to 359^0. A right turn increments the value and the left turn decrements the value.

1. Hardware

The hardware portion is mainly about the **mhb_ahb_sevensegment. v file**. The peripheral of the seven-segment display. The peripheral works by driving the value on the AHB-lite bus to display it on the seven-segment display.

In the **mhb_ahb_sevensegment. v file** there was an instantiation of the seven-segment timer which ensures that the display is refreshed at the required rate. It also consists of the seven-segment decoder which is used to convert the input to the equivalent Hexadecimal value which is used to display the values on the seven-segment. Then this module is brought up the hierarchy by instantiating it in the various module.

```
always @(posedge HCLK or negedge HRESETn)
begin
        if (~HRESETn)
        begin
            upper <= 32'H00000000;
            lower <= 32'H00000000;
            en    <= 8'HF0;
            dp    <= 8'hF7;
        end
        else if (we)
        begin
            case (HADDR_d)
                `H_SS_LADDR : lower <= HWDATA[31:0];
                `H_SS_HADDR : upper <= HWDATA[31:0];
                `H_SS_VADDR : en    <= HWDATA[7:0];
                `H_SS_DADDR : dp    <= HWDATA[7:0];
            endcase
        end
end
```

Figure 1: Verilog Seven-Segment Code

This the main Verilog code snippet I wrote in the seven-segment module. In this snippet it shows that whenever reset button is pressed it will display the hardcoded value which I am passing as the default value to the decimal point, Enable display, upper segment and the lower segment. If the **we (Write enable signal)** is asserted high it will pass the specific data value which comes on the HWDATA signal to the specific lower, upper, en and dp register which are defined under the module. These signals pass the value to the address which are defined under the label next to it. Then these signals are connected all the way to the top module. The label corresponds to the particular address which are defined under the **mfp_ahb_const. vh**

```
`define H_SS_VADDR          (4'h0)          // address for digit enable
`define H_SS_LADDR          (4'h8)          // address for lower bit
`define H_SS_HADDR          (4'h4)          // address for upper bit
`define H_SS_DADDR          (4'hc)          // address for decimal point
```

Figure 2: Address label for Seven-segment

3. Software

The software part consists of the assembly code which was written using the Codescape IDE. It is divided into two parts.

- Motion indicator
- Compass

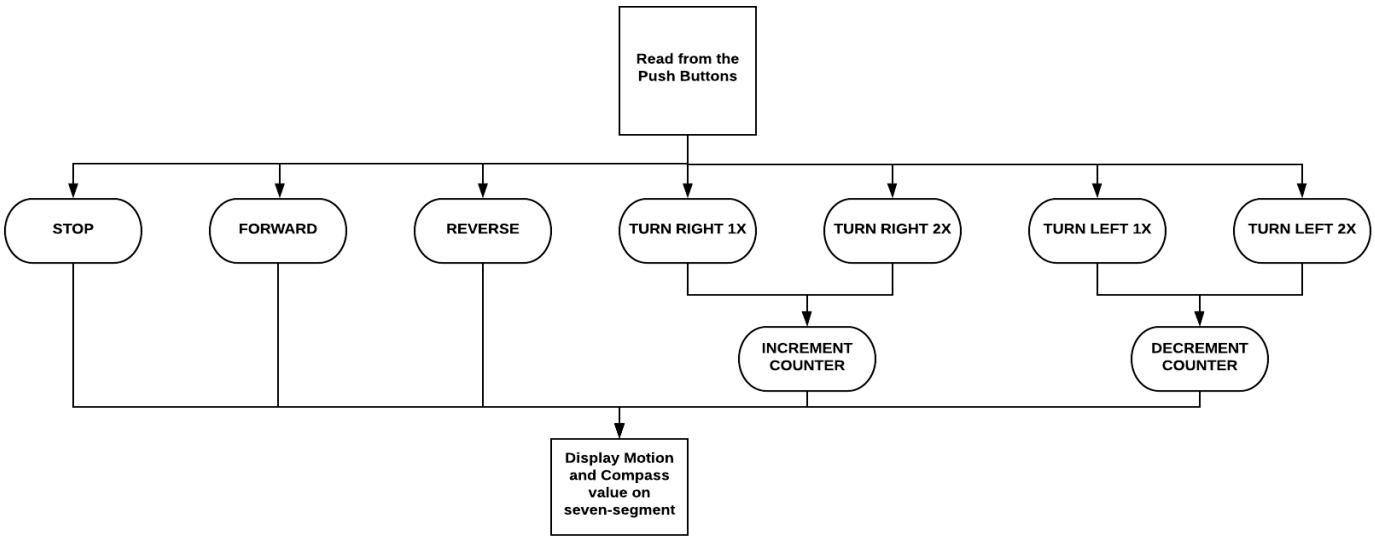

Figure 3: Generic flow of MIPS Code

Given above is the generic block diagram of all the possible combinations of the push buttons and what is the generic flow of the motion indicator in MIPS code. Thus, depending upon the different combination of the pushbuttons the particular functions are made in the code. Then as the particular combination is pressed the particular function call is made.

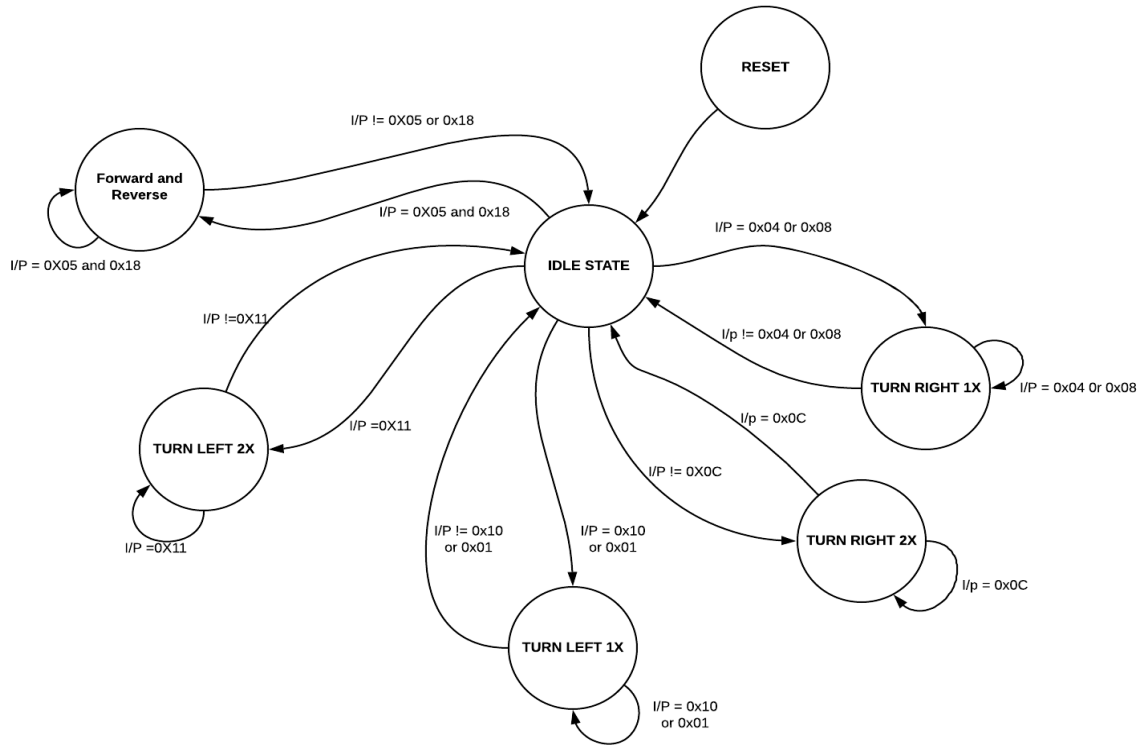

Figure 4: State Transition Diagram

The above state diagram describes the conditions when the particular function will be called depending upon the data from the pushbuttons. It will remain in the particular state until those buttons are pressed and when they are left it comes to the idle state again. This way in the above diagram I have covered all the possible combination of pushbutton and defined the action which should be executed when the specific combination takes place.

```
beq    $20,0x10,L3   # if pushbutton input is 0x10 turn left 1x speed
beq    $20,0x04,L2   # if pushbutton input is 0x04 turn right 1x speed
beq    $20,0x08,L2   # if pushbutton input is 0x08 turn right 1x speed
beq    $20,0x0c,L6   # if pushbutton input is 0x0C turn right 2x speed
beq    $20,0x01,L3   # if pushbutton input is 0x01 turn left 1x speed
beq    $20,0x05,L4   # if pushbutton input is 0x05 go forward
beq    $20,0x11,L7   # if pushbutton input is 0x11 turn left 2x speed
beq    $20,0x18,L5   # if pushbutton input is 0x18 go reverse
        j      idle  # if none of the switch case is pressed go to the idle state
```

Figure 5: Snippet of the Switch Cases in MIPS

As shown in the code, if we pressed the pushbutton in the particular way that it gives input as the Hexadecimal number 0x04 then the function to turn the bot left will be called where the segments rotate in the clockwise direction. Likewise, in the above code snippet there is the switch case statement in the assembly code where it covers all the possible cases the push buttons can be pressed to display the particular motion. If neither of the case is not called it indicates that the bot is in the idle condition and is in the stop state where it is not performing any operation.

```
Reverse:    li $23,0x13000000

            and   $5,$5,0x00ffffff   # perform the and operation between the data in the $5 and the 0x00FFFFFF and store value in $5
            or    $5,$5,$23          # perform the and operation between the data in the $5 and the 0x00FFFFFF and store value in $5
            sw    $5, 0($16)         # write pushbutton values to lower 7 segment digits

            jal delayblink           # function call to generate delay

            li    $23,0x1c000000     # load the specific value in the $23 to display 'D' segment
            and   $5,$5,0x00ffffff   # perform the and operation between the data in the $5 and the 0x00FFFFFF and store value in $5
            or    $5,$5,$23          # perform the and operation between the data in the $5 and the 0x00FFFFFF and store value in $5
            sw    $5, 0($16)         # write pushbutton values to lower 7 segment digits

            jal delayblink           # function call to generate delay
            jal display_compass      # funcation call to display compass value
            jal read                 # function call to read pushbutton
            j idle
```

Figure 6: Snippet of the Reverse State

For indicating the reverse motion in the seven-segment display, I need to blink the ‘D’ segment of the seven-segment. Thus, for this reason the above function is called when specific up and down pushbutton are pressed together. If the above given function is called it will pass the value 0x1c to the seven-segment which will turn on the ‘D’ segment and after that passing the small amount of delay for the specific time I am passing the value 0xFF which indicates that the display is off and after that again passing the delay for the specific interval. This way using different logic each and every function mention above is called and implemented in the code.

```
             null: li   $7,0000                  # loading 0 into $7
        increment: addi $7, $7, 1                # incrementing the value of $7
                   beq  $7,$21,overflow_detected # comparing the value of $7 with $21 (359) if equal will jump to specific location
                   j skip_overflow_detected
overflow_detected:  li   $7,0000
skip_overflow_detected:
                       j $ra
                        nop
```

Figure 7: Snippet of the Increment Counter

The above snippet is of the incrementing counter which is called each and every time when the robot takes the right turn with 1x or 2x speed. This function is called when the segment of the seven-segment rotates in the clockwise direction thus it counts the number of time segment changes the number to display on the seven-segment

The same goes for the decrementing function but here instead of incrementing the count I decrement it to indicate that the robot is moving in the opposite direction. Hence, this function is called whenever the bot is turning left at 1x or 2x speed.

```
display_compass: move  $22,$7                  # move the counter value in to $22
                 li $8,3                       # load value three in $8
         display: addi $8,$8,-1                # decrement the value of the $8
                 divu $22,10                   # divide the value to get the single digit number
                 mfhi $4                       # remainder of the division store in $4
                 mflo $22                      # quotient of the division sotore in $22
                 sll  $4,$4,24                 # left shift the value in $4 by 24 bit
                 or $9,$9,$4                   # or the value of $4 and the $9 result store in $9
                 srl $9,$9,8                   # right shift the value of the $9 by 8 bits
                 bne $8,$0, display            # if the value of the $8 not equal to 0 branch
                 or  $23,$9,$23                # or the content of the $23 and the $9
                 sw $23,0($16)                 # display thr value on the segment
                 move $5,$23                   # move the value to $5
                 j $ra                         # unconditional jump
                 nop

                 beq   $0, $0, readIO          # repeat
                 nop                    # branch delay slot
```

Figure 8: Snippet of the Display Function

The above function is of the displaying the content of the compass. In this function I am first passing the number which I got from the counter. Then dividing that number by 10 to get the individual number after getting the individual number I am shifting it right by 24 bits and then oring it with the next single digit number which I get from the counter and then left shift it by 8 bits to get the actual counter number back and after that I display it on the seven-segment display.

4. References

- The getting stated project, and the provided documentation with the project.
- The MIPS Instruction Sheet
  https://courses.cs.washington.edu/courses/cse378/09au/MIPS_Green_Sheet.pdf
- The MIPS reference sheet
  https://inst.eecs.berkeley.edu/~cs61c/resources/MIPS_help.html