CEE-445 Embedded System Design

FPGA 2 Part 2 Embedded System Design: Last Update: 11/1/2018

Part 1:
UART and Bluetooth Communications
UART – Universal asynchronous receiver/transmitter

Lab Goal

The goal of this lab is to understand how UART communications work, and we will use Bluetooth communication as an example to study the UART protocol. This lab will walk you through to complete the UART module design in Verilog to interface with the standard Bluetooth 2.0 to navigate a robot.

There are two parts of this lab exercises:

Part 1 is to study the UART interface for the robot to connect to an Android application via a Bluetooth module. The second part will discuss how to connect to a Bluetooth module with an Android device and run apk file on a smart phone or a tablet device.

Introduction:

There are a number of communication methods in today's technology, and in particular serial communications which allow fast messages to be sent and received with only a few numbers of available connections, such as UART (universal asynchronous receiver/transmitter, serial peripheral interface (SPI) and inter-integrated circuit (I2C). Each technique requires a protocol which defines the way in which communications are initiated and how the data messages are structured, effectively a code for deciphering all the digital 0s and 1s, in order to understand a message as meaningful information.

Bluetooth (BT) Serial Communication

Bluetooth is a novel method of digital communication, operating at 2.402 – 2.480G Hz radio band. Bluetooth provides wireless data link in a range of up to 100 meters for Class 1 Bluetooth devices and 20

meters for Class 2 Bluetooth devices. Class 1 and 2 Bluetooth devices consume 100mW and 2.5mW, respectively. Bluetooth data rate is up to 3Mbps.

Bluetooth standards, controlled by the Bluetooth Special Internet Group, dictate that when Bluetooth devices detect one another, they determine whether they need to interact with each other. Each Bluetooth device has a Media Access Control (MAC) address which communicating devices can recognize and initialize interaction if required.

Interfacing the RN-42 Bluetooth (BT) Modules

RN-42 Bluetooth Modules from Roving Networks

The Roving Networks RN-41 and RN-42 (shown in picture above) are serial devices that allow replacement of serial wires with a simple Bluetooth interface. The RN-41 and RN-42 modules have identical control and functionality; however, the RN-41 is a Class 1 BT device, whereas the RN-42 is Class 2.

Bluetooth Pin Configuration:

The RN-42 Bluetooth module has the following output pins: RTS, RDX, TXD, CTS, GND, VCC and RST. Power is 5V. TXD serially transmits bits to a receiver which buffers theses bits at the RXD pin. RTS and CTS are 1-bit signals sent and received by the sender and receiver to indicate readiness for transmission.

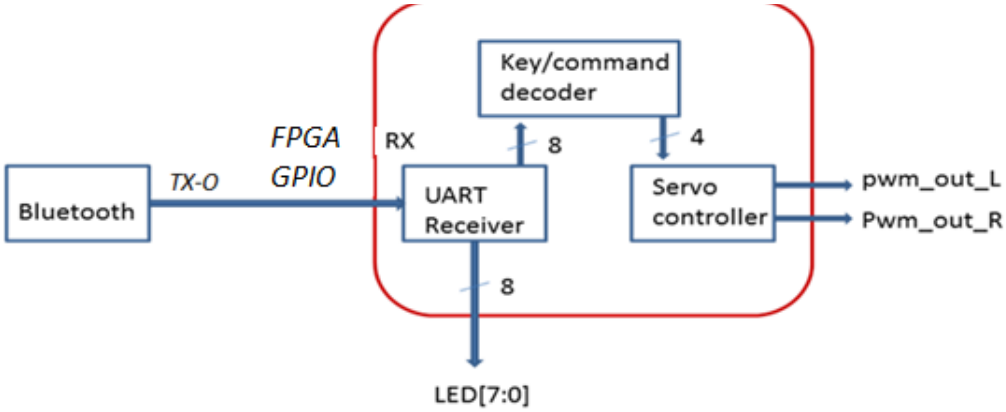
Bluetooth baud Rate:

The RN-41 and RN-42 operate at a baud rate of 1200 bps and 921 kbps and include auto detect and auto connection features. This means that the BT module can be paired for connection with a Bluetooth enabled device (e.g. computer, laptop, phone, tablet etc.). Once a BT module is paired with a device, BT module needs no pairing again. When initializing the connection to the Bluetooth module, it will be necessary to specify a ***passkey, which is set by default to '1234' as specified in the Advanced User Manual for Roving Networks"*** Bluetooth Devices.

Sending FPGA data over Bluetooth

The RN BT from Roving Networks has a number of configurable features. However, it is very simple to use out of the box as a standard serial interface with a simple TX/RX connection. This lab example set up serial communications with RN-42 and the data are sent from a smart phone or tablet to the RN-42 device over the UART. The data is a continuous count through the ASCII values representing numerical characters 0 to 9. The on-board FPGA light-emitting diodes (LEDs) also configured to represent the ASCII values sent over the UART. Then, we convert the ASCII byte to the relevant numerical value by bits to control the actions of the servo.

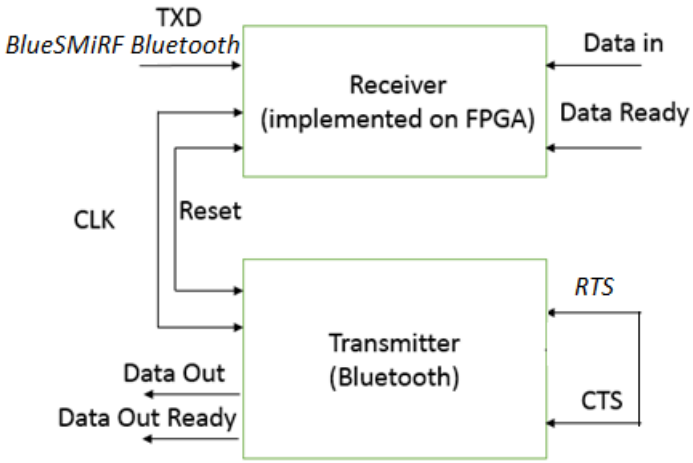
A block diagram to illustrate the relationships between the Verilog modules in this lab exercise



Note: the UART is configured as a receiver for 8-bit ACII code from the Bluetooth module (RN-42). The 8-bit ASCII code is decoded to control the servo motors. The LED[7:0] is used to display the ASCII code to make sure the UART is receiving data from the Bluetooth module.

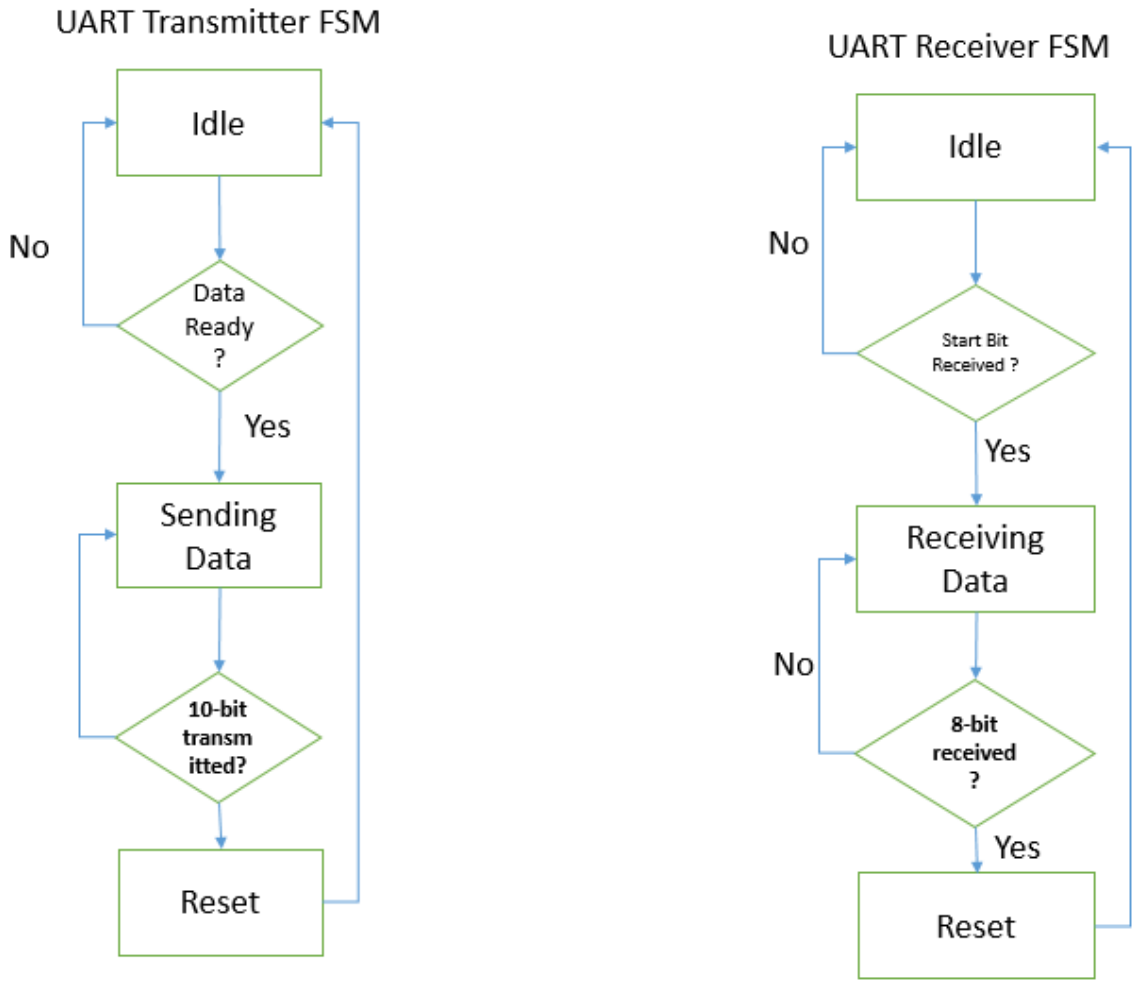
Part 1: UART Receiver Design

The UART receiver plays a key part of this lab exercise. In this exercise, we will implement a UART which consists of communications between receiver and transmitter.



On the Receiver block, TXD will receive the value from the UART interface (Bluetooth) bit by bit then these received bits are stored and sent as a single 8-bit bus data to the FPGA through Data In. Data In Ready bit indicates the data is completely received in a 8-bit format and ready to transmitted to the FPGA.

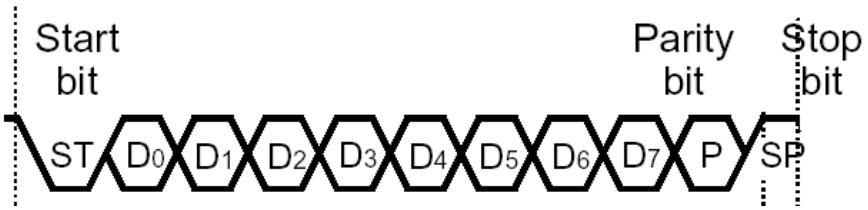
On the Transmitter block, Data Out is an 8-bit bus of the data to be transmitted to the Bluetooth. Data out Ready bit is a 1-bit signal to indicate that there is data to be transmitted by the Bluetooth. The transmitted data will be sent through the RXD signal bit by bit to the Receiver via the Antenna in the Bluetooth module. RTS and CTS are 1-bit signals used to alert to each other for their states and to indicate readiness for transmission. CTS is a clear to send command and RTS is a request to send command and are used in software handshaking.



A **universal asynchronous receiver/transmitter** (UART) is a computer hardware that transmits data between parallel and serial forms. UARTs are commonly used with communication standards such as, RS-232, RS-422 or RS-485. The "*universal"* indicates that the data and transmission speeds are configurable. Standard baud rates include 110, 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 38400, 57600, 115200, 128000 and 256000 bits per second, and it is simply doubling each time. These standard transmission rates are called baud rate.

Asynchronous versus **synchronous:** synchronous serial transmission requires that the sender and receiver share a clock with one another. Asynchronous transmission allows data to be transmitted without the sender having to send a clock signal to the receiver. Instead, the sender and receiver must agree on timing parameters in advance and special bits such as "start" and "stop" bits are added to each word which is used

to synchronize the sending and receiving units. The picture below illustrates a start and stop bit being inserted to the data bits.

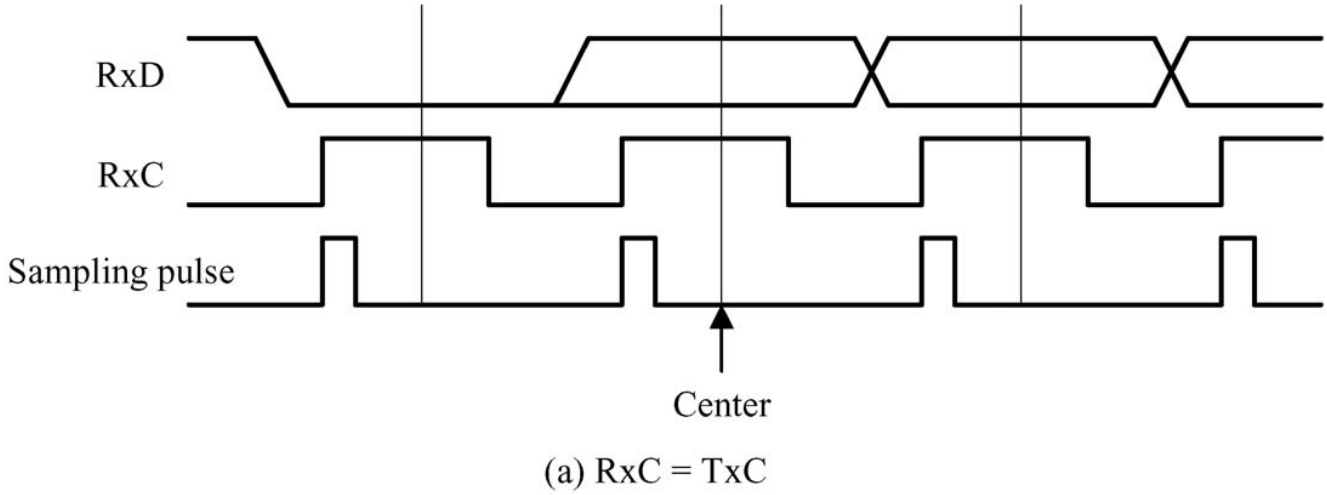


Start bit (ST) has a signal of '0' and the stop bit has a signal of '1'

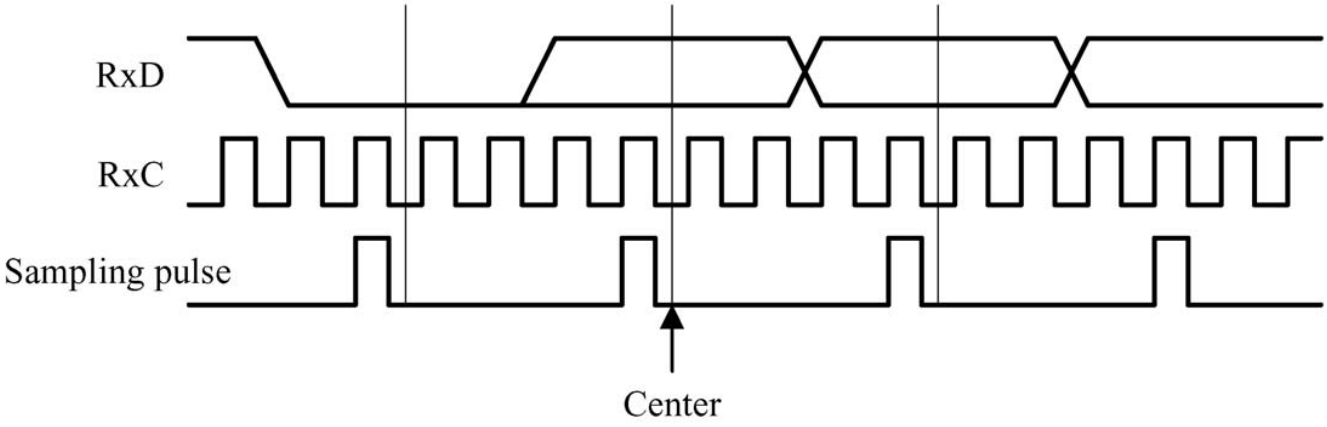
Since no clock information is conveyed from the transmitted signal, the receiver can retrieve the data bits by predetermined parameters. We will use an oversampling technique to estimate the middle points of the transmitted bits and retrieve them at these points accordingly.

The most commonly used oversampling rate is 16 times the baud rate, which means that each serial bit is sampled 16 times. Assume that we have communication that uses N data bits and M stop bits. The oversampling technique works as follows:

1. Wait until incoming "Start bit" to become '0', then at the beginning of the start bit, start the sampling tick counter.
2. When the bit counter reaches 7, the incoming signal reaches the next start bits with a signal of '0', clear the counter and restart.
3. When the same bit counter reaches 15, the incoming signal progresses for one bit and reaches the next start bit. Retrieve its value at the next bit, shift it into a register.
4. Repeat the step 3 to retrieve the remaining data bits.
5. Obtain the stop bit and reset the counter.



(a) RxC = TxC



(b) RxC = TxC * 4

Baud rate generator:

We need to divide the clock frequency down to a desired baud rate with a scaling factor or an oversampling factor.

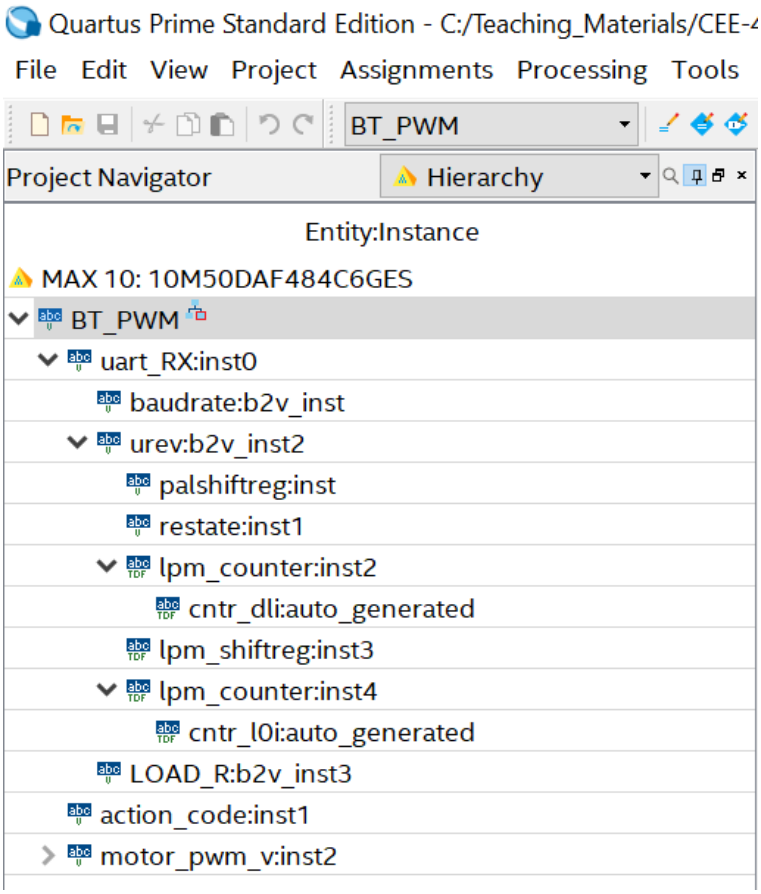
50MHz clock is provided from the DE10-Lite board, we need a baud rate of 9600 bits per second with a 16x oversampling rate. That is:

Division factor = 50 MHz / (9600 bits per second*16) = 325.5

We will use the integer value 325 with a slight frequency error.

Procedure:

Create a new project in Quartus. The project tree should be similar to the one as illustrated below.



Example top module:

```
1
2   //module Bluetooth_Enabled_Cart_control
3
4   module BT_PWM ( CLOCK_50, reset_n,  RX, LED, pwm_out_left, pwm_out_right );
5   parameter divvalue =     ;//9600 bps
6
7   input CLOCK_50;
8   input  reset_n;
9   input    RX;
10  output [7:0] LED;
11  output pwm_out_left;
12  output pwm_out_right;
13
14  wire [3:0] action;
15
16  uart_RX inst0(.clk(CLOCK_50),.rdata(RX),.rst(reset_n) ,.R_out(LED));
17  defparam  inst0.divvalue =         ; //9600
18
19  action_code inst1(.ASCII_code(LED), .action(action) );
20
21  motor_pwm_v inst2( .reset_n(reset_n),. clk(CLOCK_50),  .SW(      )
22                     ,.pwm_out_left(pwm_out_left), .pwm_out_right(pwm_out_right));
23
24  endmodule
25
```

This example has three sub modules.

1. **Baud rate generator**

```
1
2   //The baudrate module is to generate the sampling ticks for the UART receiver
3
4   module baudrate(clk, baud8clk);
5
6      // Input Port(s)
7
8      input clk;
9
10     // Output Port(s)
11
12     output baud8clk;
13
14     // divvalue not important here; will be overriden on top module
15      parameter  divvalue = 5 ;
16      reg     [26:0]   cnt;
17      reg     divcout;
18
19     //Create a baudrate counter using a rising edge of the clk signal
20     //The counter value (cnt) was reset to zero when the divvalue is zero
21     //Otherwise increment 1 to the counter value 'cnt'
22
23
24
25      //Create the baud rate sampling ticks at every rising edge of
26      // cnt value from the baud rate counter above;
27      // name the output as divcout
28
29
30
31      //At every rising edge of clk, invert each sample ticks using
32      // the xor function; name baud8clk as the parameter for the output
33
34
35  endmodule
```

2. **UART receiver module**: it receives signals/commands from a Bluetooth module connected to the FPGA board. The UART receiver receives the following commands “Stop”, “Start”, Turn Left”, “Turn Right”, Turn Back” from the smart phone.

```
1
2   //This module is a UART receiver, which makes up of
3   //standard modules provided from the Quantus II library
4   //shift registers, counters, flip-flops are used for the
5   //UART communication
6
7   //This UART receiver receives button signals "forward"
8   //"backward", "turn left", and "turn right" commands
9   //from the application run on the smartphone/tablet.
10
11  module urev(clk,rst,uartdata,uartredone,uartredata);
12  input clk,rst,uartdata;
13  output   uartredone;
14  output   [7:0] uartredata;
15  wire  [3:0] bitcount;
16  wire  shEN,bitcountEN,clkcountEN,redone,rstbitcount;
17  wire  [2:0] clkcount;
18  wire  [1:0] redata;
19
20  palshiftreg inst(.clrn(rst),.clk(clk),.Ldn(1'b0),.Sh(shEN),
21                   .Di(redata[0]),.Q(uartredata));
22
23  defparam inst.width = 8;
24
25  restate  inst1(.clk(clk),.redata(redata[0]),.rst(rst),
26                .bitcount(bitcount),.clkcount(clkcount),
27                .clkcountEN(clkcountEN),.shEN(shEN),
28                .bitcountEN(bitcountEN),.rstbitcount(rstbitcount),
29                .redone(redone));
30
31  lpm_counter inst2(.sclr(rstbitcount),.clock(clk),
32                    .cnt_en(bitcountEN),.aclr(~rst),.q(bitcount));
33  defparam inst2.lpm_width = 4,inst2.lpm_type = "LPM_COUNTER",
34         inst2.lpm_direction = "UP";
35
36  lpm_shiftreg inst3 (.clock(clk),.shiftin(uartdata),.aset(~rst),
37                  .q(redata));
38
39  defparam inst3.lpm_type = "LPM_SHIFTREG",inst3.lpm_width = 2,
40         inst3.lpm_direction = "RIGHT";
41
42  lpm_counter inst4 (.sclr(~clkcountEN),.clock(clk),.aclr(~rst),
43                     .q(clkcount));
44
45  defparam inst4.lpm_width = 3,inst4.lpm_type = "LPM_COUNTER",
46         inst4.lpm_direction = "UP";
47
48  dff inst5(.D(redone), .CLK(clk), .CLRN(rst), .Q(uartredone) );
49
50  endmodule
51
```

3. Load data module:

```
1
2   //This module checks if data is ready from the UART receiver
3   //for the bluetooth module.
4   //If data is ready, the bluetooth module will switch the red LED
5   //to Green LED.
6
7   module LOAD_R(clk,RDR, R_out,ready_data);
8   input ready_data,clk;
9   input [7:0]RDR;
10  output reg  [7:0]R_out;
11
12  always @(posedge clk)
13  begin
14     if(ready_data)
15     begin
16
17       R_out<=RDR;
18
19
20     end
21  end
22
23
24  endmodule
```

Action code module:

The action code module below is a key decoder which processes the commands “Stop”, “Start”, Turn Left”, “Turn Right”, Turn Back”. This decoder also translates these commands in ASCII codes into hex and binary numbers for the sub module motor_pwm_v.v.

A sample code to process key commands:

```
module  action_code(ASCII_code, action );
input [7:0] ASCII_code;
output [3:0]action;
reg [3:0] action;

always @ (ASCII_code)
begin
case(ASCII_code)
8'b00110001: action =4'b0001; //if ASCII_code =8'b00110001,action =4'b0001
8'b00110010: action =4'b0100; //if ASCII_code =8'b00110010,action =4'b0100
8'b00110011: action =4'b1000; //if ASCII_code =8'b00110100,action =4'b1000
8'b00110100: action =4'b0010; //if ASCII_code =8'b00110101,action =4'b0010
default: action =4'b0000;
endcase
end
endmodule
```

motor_pwm_v.v:

This module receives outputs from the action code module and control servo motors and LEDs according to the commands from the android app. These command keys are summarized below.

Command Key	Key code	ASCII, decimal	ASCII, hex	LED[7:0]*
Stop	0	48	30	00110000
Forward	1	49	31	00110001
Backward	2	50	32	00110010
Turn Left	3	51	33	00110011
Turn Right	4	52	34	00110100

Note: * ‘0’ indicates a LED is OFF, and ‘1’ indicates “ON”

Pin assignment:

Named: * Edit:

Node Name	Direction	Location	I/O Bank	/REF Group	tter Locatic	'O Standar	Reserved	rent Stren	Slew Rate
CLOCK_50	Input	PIN_P11	3	B3_N0	PIN_P11	2.5 V		12mA...ult)	
LED[7]	Output	PIN_D14	7	B7_N0	PIN_D14	2.5 V		12mA...ult)	2 (default)
LED[6]	Output	PIN_E14	7	B7_N0	PIN_E14	2.5 V		12mA...ult)	2 (default)
LED[5]	Output	PIN_C13	7	B7_N0	PIN_C13	2.5 V		12mA...ult)	2 (default)
LED[4]	Output	PIN_D13	7	B7_N0	PIN_D13	2.5 V		12mA...ult)	2 (default)
LED[3]	Output	PIN_B10	7	B7_N0	PIN_B10	2.5 V		12mA...ult)	2 (default)
LED[2]	Output	PIN_A10	7	B7_N0	PIN_A10	2.5 V		12mA...ult)	2 (default)
LED[1]	Output	PIN_A9	7	B7_N0	PIN_A9	2.5 V		12mA...ult)	2 (default)
LED[0]	Output	PIN_A8	7	B7_N0	PIN_A8	2.5 V		12mA...ult)	2 (default)
RX	Input	PIN_W8	3	B3_N0	PIN_W8	2.5 V		12mA...ult)	
pwm_out_left	Output	PIN_V10	3	B3_N0	PIN_V10	2.5 V		12mA...ult)	2 (default)
pwm_o...right	Output	PIN_W10	3	B3_N0	PIN_W10	2.5 V		12mA...ult)	2 (default)
reset_n	Input	PIN_B8	7	B7_N0	PIN_B8	2.5 V		12mA...ult)	
<<new node>>									

To be continued --- Part 2 of this lab is on D2L.