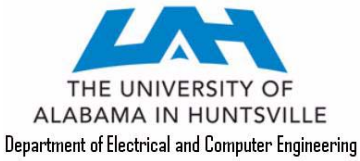


CPE 324 Advanced Logic Design Laboratory

Laboratory Assignment #5
Serial Communication Interface

(7.5% of Final Grade)

Purpose:

The purpose of this laboratory is to give students some practical experience implementing behavioral Verilog HDL models that can be used to create control elements that are representable as finite state machines. This laboratory also gives students the opportunity to apply design reuse techniques as they are required to integrate previously created IP core modules with newly created ones that they will develop. Students will also get their first laboratory experience creating a design that is entered in its entirety using Verilog HDL.

Design Problem

In this laboratory students are to create a set of modules that support the basic functionality of a standard Universal Asynchronous Receiver/Transmitter, UART, interface (but at a set baud rate, stop bit and parity configuration). These modules are to be fully integrated into a system that will allow for sending and receiving of information between the DE2-115 and a serial terminal device such as a computer. All data that is to be sent and received across this serial link is to be encoded in ASCII format. Data being received by the DE2-115 should to be displayed at the next character position on the DE2-115's 32-character LCD. The interface should send keypad character data from the DE2-115 to the receiving terminal each time a key is pressed on the keypad. This should reflect the ASCII symbol that is printed on the key that is being pressed.

Background: Asynchronous Serial Communication Protocol

The base asynchronous serial communication protocol has been used for many years and even though it has been replaced by more complex protocols it is still very useful for many applications. In this protocol the two communicating devices, sender and receiver, share a common logical nodal connection with one another. One device drives this node and the other receives from it. This logical connection between the two devices can be as simple as a wire that runs between the output port of one device to the input port of the other. A common ground wire will also have to be run between the two devices. More often than not to overcome noise and increase the distance between the devices the interface requires the use of drivers and receivers that perform voltage transformation/amplification or the use of differential pairs. Voltage transformation is the technique that is utilized by the RS232 compliant serial port in the laboratory. The major features of the asynchronous protocol is that no global clock is required to simultaneously drive the logic circuitry in both devices. Both devices have their own individual clocks which cannot be assumed to be in phase with one another and these clocks may even be operating at entirely different frequencies. The only constraint is that both devices operate at the same communication baud rate and share the other relevant communication parameters (stop bits, parity, etc.).

The base asynchronous data protocol is shown in Figure 1. In this protocol a hypothetical reference clock is shown that has a period that is equal to 1/baud rate. A bit time is equal to the value of that period. In the case where a single byte of data (8 data bits) is to be sent between the two devices with no parity and one stop bit then the protocol can be described as follows. Before sending any information and between data transfers the sending device drives its output to a Logic 1. When sending information it first drives the output from a Logic 1 to a Logic 0 for one bit time which creates a *Start Bit*. The receiving device will use this *Start Bit* to synchronize itself to receive the data (or payload) information that will follow. Knowing the *baud rate, data size, parity* options and the number of *Stop Bits* allows the receiver to correctly input the data that follows. After the *start bit* the data is sent one bit at a time least significant bit first with each bit being held for the duration of the bit time (period of the frequency of the reference clock). After the data has been sent, a parity bit could be sent (if there is parity) and one or more *Stop Bit(s)* are then sent after which a new data item can be transferred. The parity bit is sometimes used to detect if there is an error in the transmission. It this laboratory we will not utilize parity and thus will always assume that data transmission will be error free.

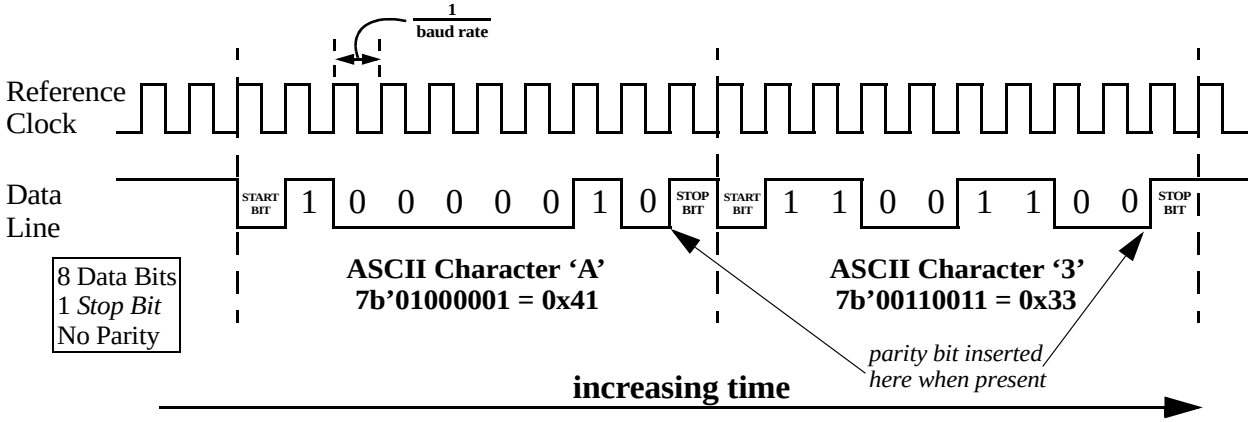


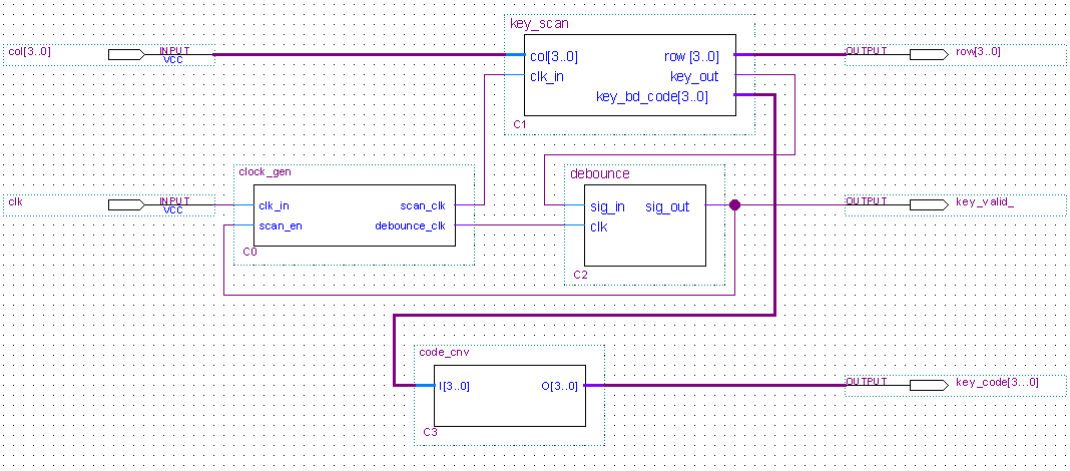
Figure 1: Asynchronous Serial Data Transfer Format

To be understood, data must be encoded using some standard format. A common one-byte standard is the ASCII character format (see .http://www.asciitable.com). Figure 1 illustrates when the ASCII encoded characters 'A' and '3' are sent in quick succession using the asynchronous serial protocol for the case where there is the minimum amount of time allowed between characters and there is only one *Stop Bit*. For more information on the Asynchronous Serial Communication protocol please refer to http://en.wikipedia.org/wiki/Asynchronous_serial_communication.

Design Reuse -- Keypad Scanner

Often components of previous designs can be utilized to form parts of new designs. If designed in a general manner these components can become a part of valuable Intellectual Property, IP, libraries that can greatly speedup the design cycle. In this laboratory it is desirable to utilize a keypad as a data entry device. Since that was the focus of the previous laboratory it is relatively easy to encapsulate the major portions of this previous design into a building block module that can be used in the current one. Since the Keypad Scanner is just one component of the design it aids in managing the complexity to treat this entire design as a single component in the new design that is being created in this laboratory. It is also apparent that there is good utility in not including the output module *hex_seven_shift* element within this module since it performs a more generic and distinctly different function than is expressed by process of keypad scanning.

Figure 2 illustrates the resulting *keypad_scanner* configuration, which is modeled structurally in Verilog HDL by incorporating module C0 through C3 of the Laboratory 4 configuration but creating a more generic *key_valid_* output which goes to zero whenever a key is pressed and the output is valid.



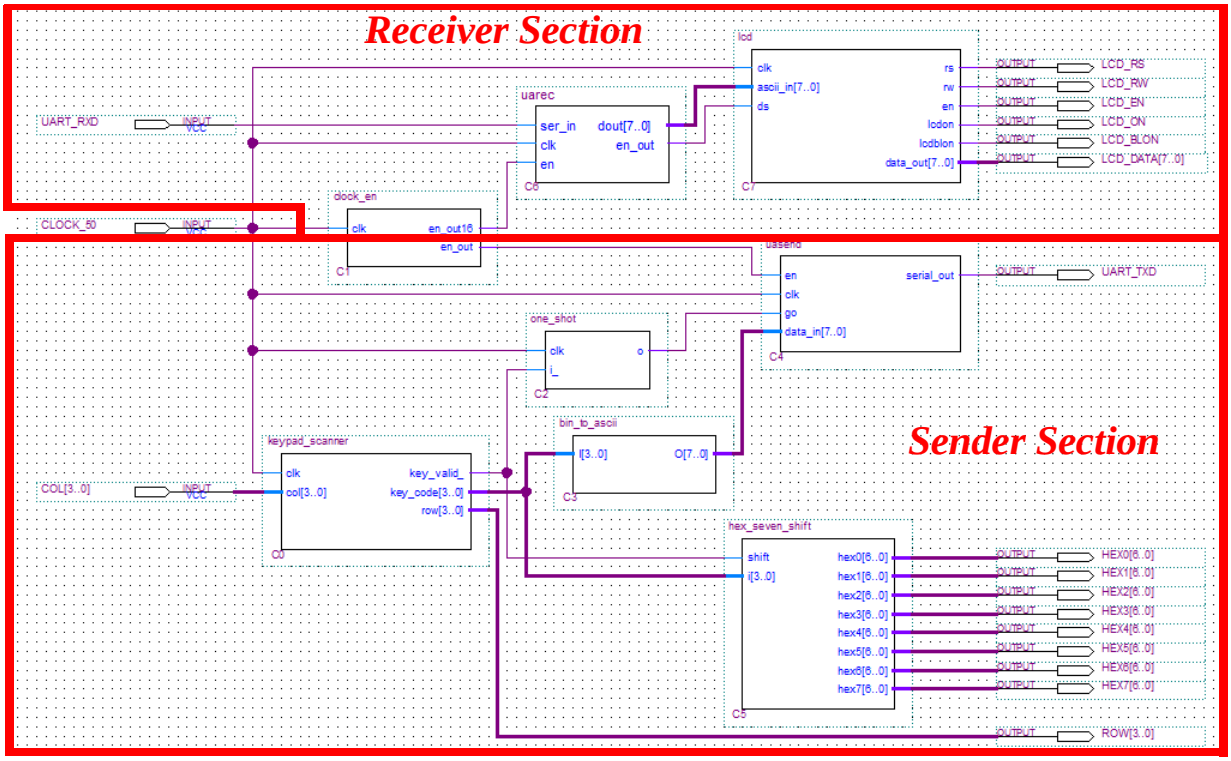

```
module keypad_scanner(clk,col,key_valid_,key_code,row);
   input  clk;
   input  [3:0] col;
   output [3:0] row;
   output key_valid_;
   output [3:0] key_code;

   // internal wires and bus
   wire n0,n1,n2;
   wire [3:0] n3;

   clock_gen C0(.clk_in(clk),.scan_en(key_valid_),.scan_clk(n0),.debounce_clk(n2));

   key_scan C1(.clk_in(n0),.col(col),.row(row),.key_out(n1),.key_bd_code(n3));

   debounce C2(.sig_in(n1),.clk(n2),.sig_out(key_valid_));

   code_cnv C3(.O(key_code),.I(n3));
endmodule
```

Figure 2: Structural Implementation of the *keypad_scanner* Module

This will be used as one of the IP Core Modules already created for use in the design.

Design Overview

The top level of the communication interface is shown in Figure 3. It is composed of the eight submodules which in effect add serial communication capability to the keypad scanning design of Laboratory 4. The design itself can be composed into two main sections, a receiver and a sender section which share a common *clock_en* module. Each of these components will now be discussed briefly.




```
module lab5(input CLOCK_50,UART_RXD, output UART_TXD, input [3:0] COL, output [3:0] ROW,
   output LCD_RS,LCD_RW,LCD_EN,LCD_ON,LCD_BLON, output [7:0] LCD_DATA,
   output [6:0] HEX0,HEX1,HEX2,HEX3,HEX4,HEX5,HEX6,HEX7);

   wire n0,n1,n2,n3,n4;
   wire [3:0] b0;
   wire [7:0] b1,b2;

   keypad_scanner   C0(.clk(CLOCK_50),.col(COL),.key_valid_(n0),.key_code(b0),.row(ROW));

   clock_en         C1(.clk(CLOCK_50),.en_out16(n3),.en_out(n1));

   one_shot         C2(.clk(CLOCK_50),.i_(n0),.o(n2));

   bin_to_ascii     C3(.I(b0),.O(b1));

   uasend           C4(.en(n1),.clk(CLOCK_50),.go(n2),.data_in(b1),.serial_out(UART_TXD));

   hex_seven_shift  C5(.shift(n0),.i(b0),.hex0(HEX0),.hex1(HEX1),.hex2(HEX2),
                       .hex3(HEX3),.hex4(HEX4),.hex5(HEX5),.hex6(HEX6),.hex7(HEX7));

   uarec            C6(.ser_in(UART_RXD),.clk(CLOCK_50),.en(n3),.en_out(n4),.dout(b2));

   lcd              C7(.clk(CLOCK_50),.ds(n4),.ascii_in(b2),.rs(LCD_RS),.rw(LCD_RW),
                       .en(LCD_EN),.lcdon(LCD_ON),.lcdblon(LCD_BLON),.data_out(LCD_DATA));

endmodule
```

Figure 3: Top-level *lab5* Module

The *keypad_scanner* module was the focus of the previous laboratory and has been described in the previous section. It is reunited so to speak with the *hex_seven_shift* module to allow for the debounced display of the last eight keys that are pressed using the DE2-115's seven-segment LEDs thereby providing the same functionality as the in Laboratory 4. The functionality of the remaining submodules will now be discussed

Clock Enable, *clock_en*, Module

The clock enable, *clock_en*, module is an IP core module that creates two pulse streams in the manner illustrated in Figure 4. The pulse stream that feeds the sender section of the design has a frequency that is equal to the baud rate that the serial port is to operate. The pulse stream that feeds the receiver section has a frequency that is equal to 16 times this baud rate. The width of each pulse is equal to the period of the driving clock which in this case is 1/50 Mhz. This forms the enabling signal to the sequential logic elements thereby allowing transitions to occur at a rate that is equal to either the specified 9600 baud rate or 16 times this rate. The timing for these signals were derived directly from the 50 Mhz clock using behavioral Verilog cyclic statements that count the number of 50 Mhz cycles needed to implement the specified bit time for 9600 baud operation (i.e. 1/9600 seconds).

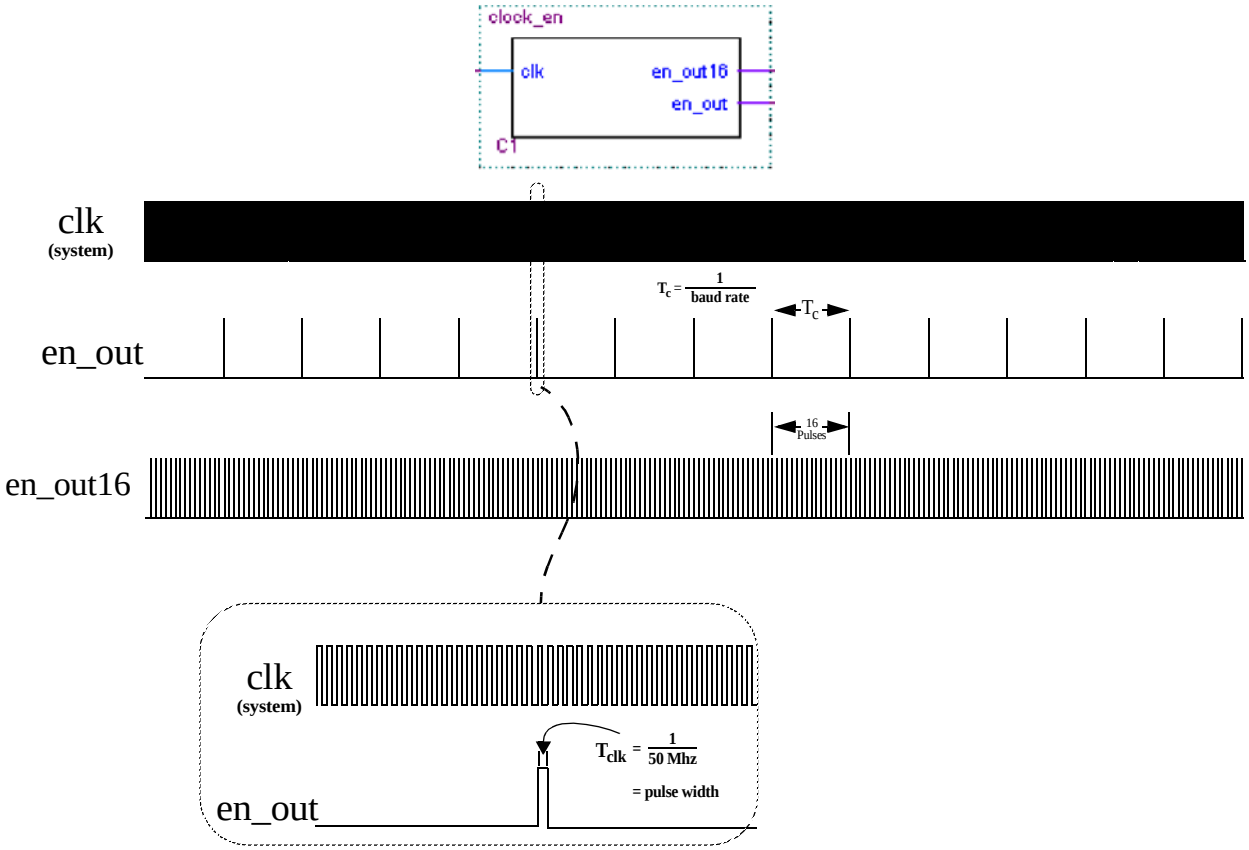


Figure 4: Timing for the *clock_en* Module

Active-low One-Shot, *one_shot*, Module

Whenever the keypad is pressed and held its output signal, *key_valid_* will remain low. Output devices that are edge triggered, like the *hex_seven_shift* module will only produce one output during this time because there is only one edge. Level sensitive output devices that are clock by a high speed clock are likely to be triggered multiple times, however. To interface with these devices a simple one cycle pulse is desired. To accomplish this one could incorporate a one shot logic element. Such and element will produce a single pulse whenever its input goes from its inactive state to its active state no matter how long the input is held in its active state. To produce another pulse in such a device requires that the input goes inactive at least for one clock cycle.

One shot elements can be active high or active low and can produce positive or negative pulses. A positive pulse is normally low but transitions to high and then back to low. A negative pulse is just the opposite. In this design an active low element is desired since the act of pressing a key produces a

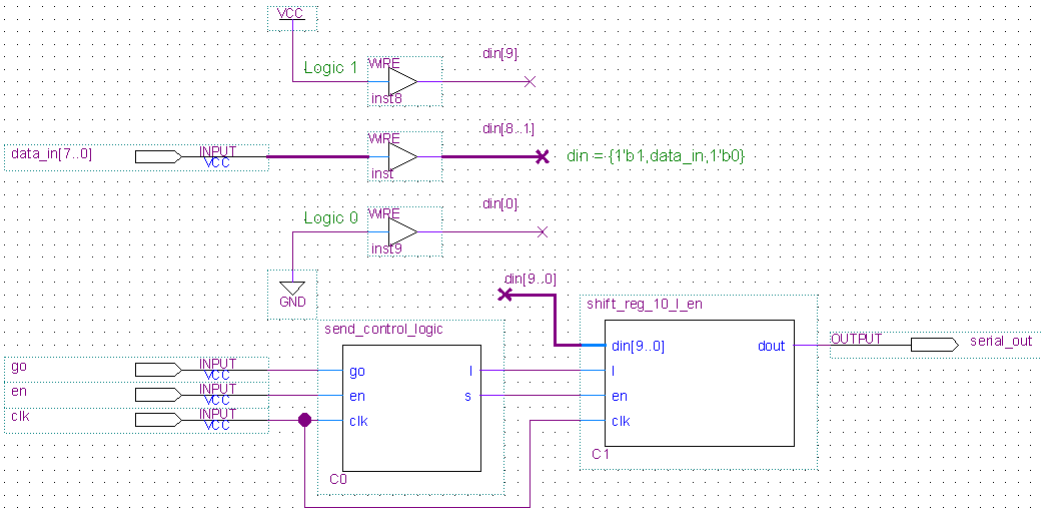
Logic 0. A positive pulse will be needed to active the subsequent *uasend* module. Figure 5 illustrates the general operation of the active low one-shot element that produces a positive pulse with a pulse width equal to the period of the 50 Mhz global clock. This module also has been implemented in behavioral Verilog HDL and is included as a preexisting IP core element.

Binary to ASCII converter, *bin_to_ascii*, Module.

The output of the keypad must be converted into a format that can be recognized by the terminal application that monitors the serial line. It is currently in a 4-bit format where each of the keys were encoded in the unique manner dictated by the laboratory assignment. This code needs to be converted into an 8-bit ASCII encoding that will allow each character that is present on the key to be displayed on the terminal screen. This module is to be completed by the student using Verilog HDL.

Universal Asynchronous Sender Module, *uasend*, Module

Figure 5 shows the structural configuration of the *uasend* module whose purpose is to actually create the asynchronous data stream for each ASCII character byte that is to be sent from the DE2-115. It is composed of 10-bit shift register with parallel load, *l*, and an enable, *en*. Both inputs are synchronous and active high. The load has priority and when active loads the 10-bit value into the shift register. The output of the shift register drives the serial output. The least significant bit of the shift register's parallel input is always driven to a Logic 0 and the most bit of this port is always driven to a Logic 1 with the middle 8 bits connecting directly to the *data_in* port. In this way both the *Start* and *Stop* bits can be concatenated to the data portion of each 10 bit packet.




```
module uasend(input go,clk,en, input [7:0] data_in, output serial_out);

   wire n0,n1;

   send_control_logic C0(.go(go),.en(en),.clk(clk),.l(n0),.s(n1));

   shift_reg_10_l_en  C1(.l(n0),.en(n1),.clk(clk),.din({1'b1,data_in,1'b0}),.dout(serial_out));

endmodule
```

Figure 5: The *uasend* Module

The *send_control_logic* module is responsible for controlling the shift register so that it can perform the asynchronous transfers. It is directly connected to the *go* and *en* inputs. The *en* inputs are

driven with enabling pulses at the specified (9600 baud) bit rate and the *go* input is activated by a one cycle pulse that indicates that data on its *din* inputs are valid and that the asynchronous sending can occur. The outputs *s* and *l* are connected directly to the shift registers enable and load inputs, respectively. This allows the element to control these operations by implementing a finite state machine.

Figure 6 represents an Extended State Graph that performs this function. **Students are required to implement this diagram behaviorally in Verilog HDL within the *send_control_logic* module as part of this laboratory.** It should be noted that in an Extended State Graph only the inputs that affect a state transition are part of the Boolean equations that initiate state transfer (the other inputs are don't cares) and an output only appears on the graph if it is a Logic 1.

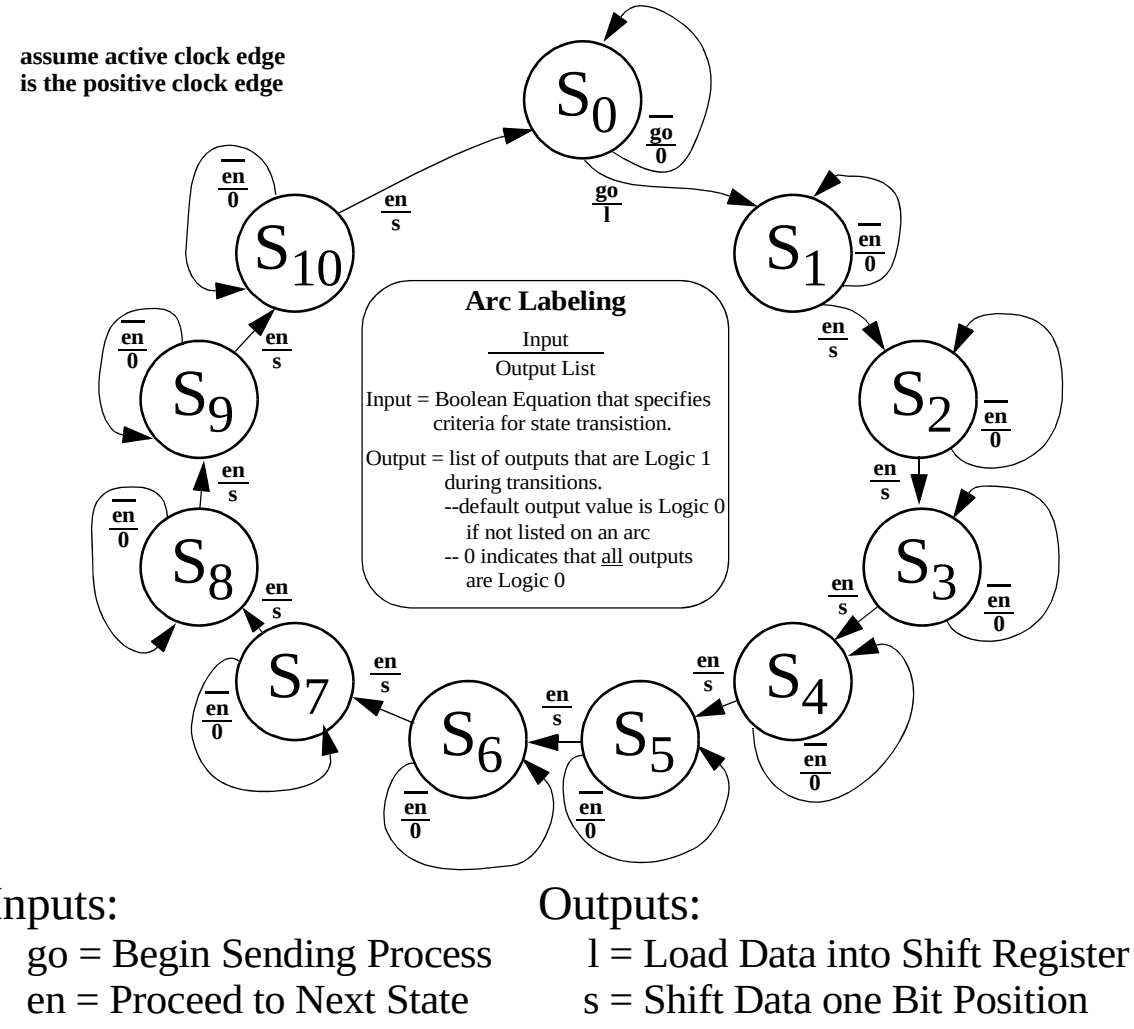


Figure 6: Extended State Graph for the *send_control_logic* Module

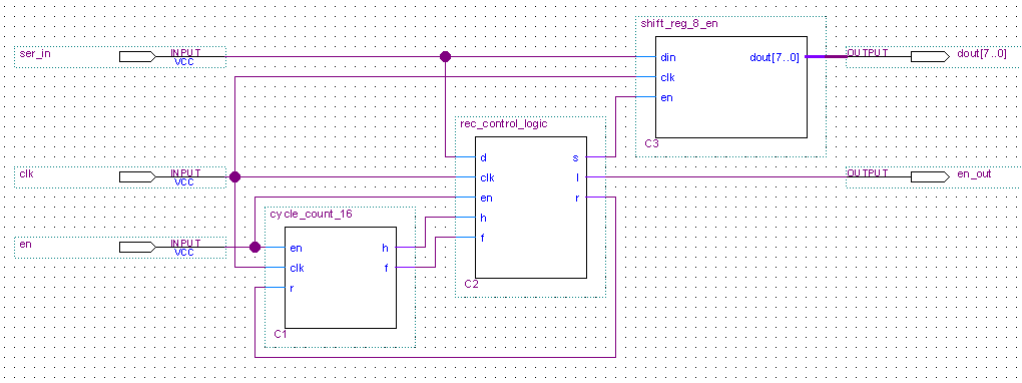
The LCD, *lcd*, Module

The receiver section depends upon a hardware module that drives the LCD on the DE2-115. This module is written in behavioral Verilog HDL and is one of the IP cores that is already part of the IP core library. The module inputs the 8-bit ASCII encode value on it data on its *ascii_in* port whenever it receives a positive one cycle pulse on its data strobe, *ds*, input. It then implements the

bus access signals that conform to the LCD data input standard. The character will appear on the next character position of the 32 character LCD with new characters overwriting the old when wrap-around occurs.

Universal Asynchronous Receiver Module, *uarec*, Module

Figure 7 shows the structural configuration of the *uarec* module whose purpose is to receive the asynchronous data stream from the terminal and make it available to the LCD module as an 8-bit ASCII encoded character and to then send out a positive pulse to this module to direct it to begin the LCD display process.



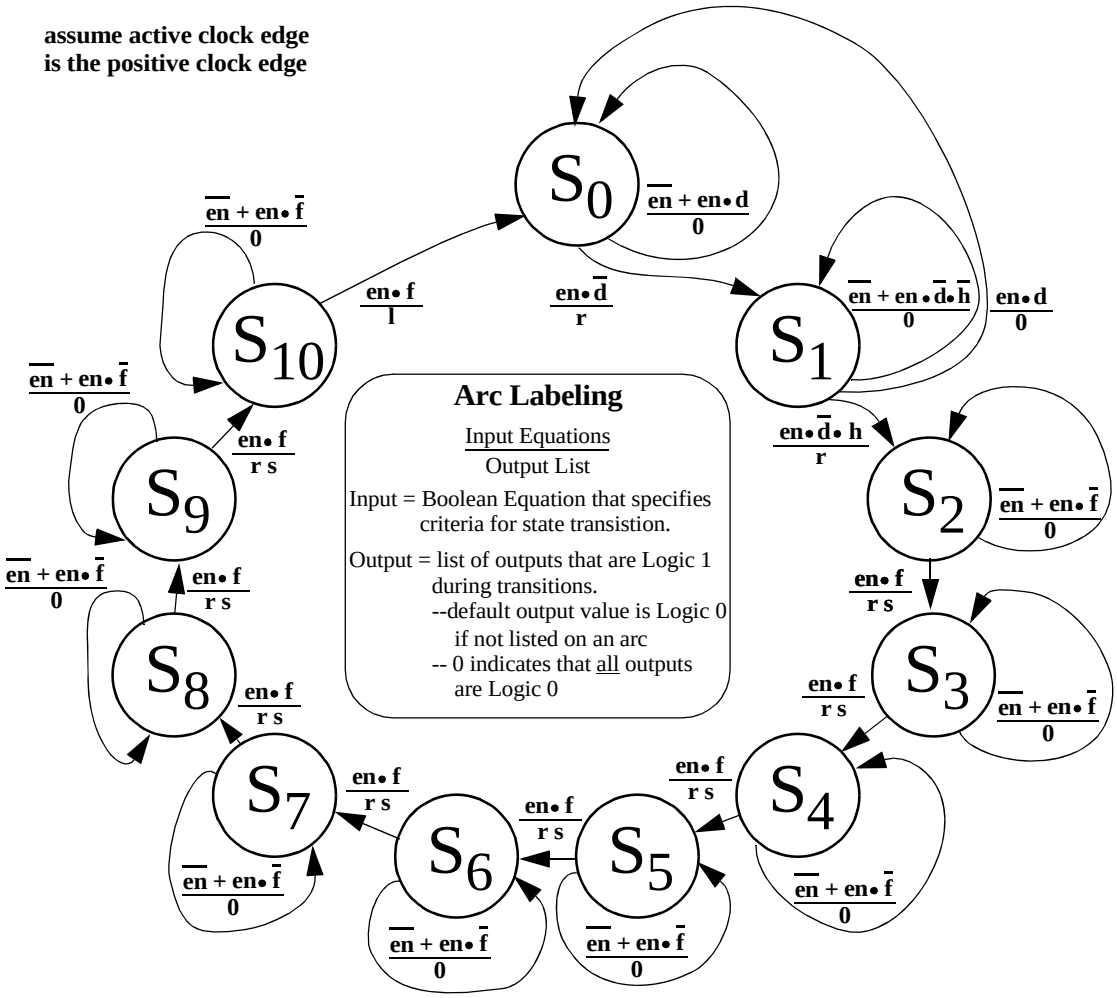

```
module uarec(input en,clk,ser_in,output en_out, output [7:0] dout);

   wire n0,n1,n2,n3;

   cycle_count_16     C1(.en(en),.clk(clk),.r(n0),.h(n1),.f(n2));

   rec_control_logic  C2(.d(ser_in),.clk(clk),.en(en),.h(n1),.f(n2),.s(n3),.l(en_out),.r(n0));

   shift_reg_8_en     C3(.din(ser_in),.clk(clk),.en(n3),.dout(dout));

endmodule
```

Figure 7: The *uarec* Module

The *uarec* module is composed of two submodules that are to be controlled by the *rec_control_logic* module that is to be modelled as a finite state machine using behavioral Verilog HDL by the student. The models for the other components are part of the IP core library and have already been created. *The cycle_cout_16* is a counter module that has an active high enable, *en*, input and reset, *r*, inputs. These inputs are synchronous to the clock with input *r* having priority. The input *r* resets the internal count value of the counter when it is active and the *en* input causes the counter to recognize the clock pulse when the *en* signal is active (Logic 1). The two outputs, labeled *h*, and *f*, are active high when the count reaches its half way value of 8 and when it reaches its full value of 16, respectively. The *shift_reg_8_en* is a standard 8-bit shift register with a serial input, *din*, and an enabling input, *en*. Its serial input is directly connected to the serial input of the design with the enabling input, *en*, being connected to the controlling *s* output of the *rec_control_logic* module.

In addition to the system clock the *rec_control_logic* module has an enable, *en*, input which comes from the enabling pulse generator the generates these pulses at 16 times the baud rate. It also has an *h* and *f* input which come directly form the *cycle_count_16* counter. The *s* output controls the shift operation of the *shift_reg_8_en*. Its *r* output controls the reset operation of the counter, and its *l* output is passed out of the module being connected to the *ds* input of the *lcd* module. This interconnection scheme allows the *rec_control_logic* module to control these functional units by implementing in this module a finite state machine.

Figure 8 is an Extended State Graph for a finite state machine implementation that is designed to use these external components to perform the receive operation. It is more complicated that the STG for the *send_control_logic* but is easily implemented behaviorally in Verilog HDL. Such an implementation is a required element for this laboratory.



Inputs:

d = Serial Data Input
h = Half Count Reached
f = Full Count Reached
en = Enable Input

Outputs:

r = Reset Counter
s = Shift Data
l = Latch Result

Figure 8: Extended State Graph for the *rec_control_logic* Module

Laboratory Assignment:

This laboratory assignment is composed of two phases, with each phase culminating with the current state of the design being validated by prototyping it on the Altera DE2-115 platform. Students must successfully complete and demonstrate the valid functionality of a given phase to their design to the lab instructor before proceeding to the next phase. Demonstration will involve configuring the Altera DE2-115 board and verifying that the specified design objectives associated with that phase were completed and integrated into the overall design. To facilitate design entry a template of the design is provided on the course Angel website. This template should be downloaded into the *lab5* folder in the lab5 folder on each student's usb drive.

Serial Communication Environment Setup:

To successfully demonstrate make sure that the equipment is set in the following manner.

- A serial cable is connected to COM port 1 into the serial port of the DE2-115
- A terminal session is opened using the Hyperterm^{tm} application.

  Configuration Parameters should be set as follows:

  Baud Rate: 9600
  Data Size: 8 bits
  No. Stop Bits: 1
  Parity: None
  Flow Control: None

Demonstration Criteria

- Upon successful completion of both phases typing in the Hyperterm^{tm} window will cause the characters typed to be displayed on the DE2-115 LCD (but not in the terminal window unless half duplex option is turned on) and pressing a key on the keypad should display the symbol associated with the key that is pressed on the screen.

Phase I: Completion of the Sender Section of the Design

In this phase, students are to first verify that the keypad scanner is working in the same manner it did in Laboratory 4. They are then to modify both the *bin_to_ascii* module and the *send_control_logic* submodule of the *uasend* module by implementing the extended finite state machine specified in Figure 6 using behavioral modeling techniques. Upon successful completion students are to demonstrate that the appropriate key is displayed on the Hyperterm^{tm} on the computer each time a key is pressed on the keypad (and only a single value is displayed for each key that is pressed). Students must obtain the laboratory instructor's approval before going on to Phase II of this laboratory.

Phase II: Creation of the Receiver Section of the Design

In this phase, students are to implement the *rec_control_logic* submodule of the *uarec* module by implementing the extended finite state machine specified in Figure 8 using behavioral modeling techniques. Upon successful completion students are to demonstrate that each time a character is pressed on the Hyperterm^{tm} terminal application on the computer that it is displayed (one time) on the DE2-115's LCD display. Students must demonstrate this to their lab instructor before proceeding to answer the post lab questions.

Post-Lab Questions:

Answer the following questions and include your answers in the appropriate section of the final laboratory report.

1. Examine the Verilog HDL model of the *clock_en* module that is shown below. .

```
module clock_en(en_out16,en_out,clk);
   input clk;
   output reg en_out16,en_out;

   reg [8:0] count1;
   reg [3:0] count2;

   always @(posedge clk)
      begin
      en_out = 0;
      en_out16 = 0;
      if (count1==325)
         begin
            en_out16 = 1;
            count1 = 0;
            if (count2==15)
               begin
                  en_out = 1;
                  count2 = 0;
               end
            else
               begin
                 count2 = count2 + 1;
               end
         end
      else
         begin
            count1 = count1 + 1;
         end
      end
endmodule
```

   - Why do you think the *count1* and *count2* register values have the dimensions they have been assigned in the *reg* declaration?
   - If the *clk* input is driven exactly at 50 Mhz calculate the actual frequency of the *en_out16* signal. How much does it differ from the targeted 16x9600 frequency?

2. In your own words, describe the operations that are being performed by the algorithms that are represented by the two Extended State Graphs described in Figures 6 and 8.

3. Is it possible to implement all of the *uasend* or *uarec* modules behaviorally in a single module without directly incorporating the external shift register and counter elements? If not explain why this is not possible, if so give a high-level discussion of how this might be done and discuss the advantages and disadvantages of such an approach.

Protoboard Wiring/DE2-115 Pin Location Information

Table 1: Keyboard Row/Column Connections

Keypad Pin Number	Design Signal Name (design inputs)	DE2-115 Pin Location	Expansion Cable Wire	Keypad Pin Number	Design Signal Name (design outputs)	DE2-115 Pin Location	Expansion Cable Wire
1	COL[0]	PIN_AC19	blue/green0	5	ROW[0]	PIN_AF24	blue/green4
2	COL[1]	PIN_AF16	blue/green1	6	ROW[1]	PIN_AE21	blue/green5
3	COL[2]	PIN_AD19	blue/green2	7	ROW[2]	PIN_AF25	blue/green6
4	COL[3]	PIN_AF15	blue/green3	8	ROW[3]	PIN_AC22	blue/green7

Table 2: DE2-115 Pin Locations for 7-Segment Hexadecimal LED Displays HEX0 -- HEX7

Design Signal Name	Direction	DE2-115 Pin Location	Design Signal Name	Direction	DE2-115 Pin Location
HEX0[6]	Output	PIN_H22	HEX4[6]	Output	PIN_AE18
HEX0[5]	Output	PIN_J22	HEX4[5]	Output	PIN_AF19
HEX0[4]	Output	PIN_L25	HEX4[4]	Output	PIN_AE19
HEX0[3]	Output	PIN_L26	HEX4[3]	Output	PIN_AH21
HEX0[2]	Output	PIN_E17	HEX4[2]	Output	PIN_AG21
HEX0[1]	Output	PIN_F22	HEX4[1]	Output	PIN_AA19
HEX0[0]	Output	PIN_G18	HEX4[0]	Output	PIN_AB19
HEX1[6]	Output	PIN_U24	HEX5[6]	Output	PIN_AH18
HEX1[5]	Output	PIN_U23	HEX5[5]	Output	PIN_AF18
HEX1[4]	Output	PIN_W25	HEX5[4]	Output	PIN_AG19
HEX1[3]	Output	PIN_W22	HEX5[3]	Output	PIN_AH19
HEX1[2]	Output	PIN_W21	HEX5[2]	Output	PIN_AB18
HEX1[1]	Output	PIN_Y22	HEX5[1]	Output	PIN_AC18
HEX1[0]	Output	PIN_M24	HEX5[0]	Output	PIN_AD18
HEX2[6]	Output	PIN_W28	HEX6[6]	Output	PIN_AC17
HEX2[5]	Output	PIN_W27	HEX6[5]	Output	PIN_AA15
HEX2[4]	Output	PIN_Y26	HEX6[4]	Output	PIN_AB15
HEX2[3]	Output	PIN_W26	HEX6[3]	Output	PIN_AB17
HEX2[2]	Output	PIN_Y25	HEX6[2]	Output	PIN_AA16
HEX2[1]	Output	PIN_AA26	HEX6[1]	Output	PIN_AB16
HEX2[0]	Output	PIN_AA25	HEX6[0]	Output	PIN_AA17
HEX3[6]	Output	PIN_Y19	HEX7[6]	Output	PIN_AA14
HEX3[5]	Output	PIN_AF23	HEX7[5]	Output	PIN_AG18
HEX3[4]	Output	PIN_AD24	HEX7[4]	Output	PIN_AF17
HEX3[3]	Output	PIN_AA21	HEX7[3]	Output	PIN_AH17
HEX3[2]	Output	PIN_AB20	HEX7[2]	Output	PIN_AG17
HEX3[1]	Output	PIN_U21	HEX7[1]	Output	PIN_AE17
HEX3[0]	Output	PIN_V21	HEX7[0]	Output	PIN_AD17

Table 3: DE2-115 Clock, Serial Port and LCD Pin Assignments

Design Signal Name	Direction	DE2-115 Pin Location	Design Signal Name	Direction	DE2-115 Pin Location
CLOCK_50	Input	PIN_Y2	LCD_DATA[7]	Output	PIN_M5
UART_RXD	Input	PIN_G12	LCD_DATA[6]	Output	PIN_M3
UART_TXD	Output	PIN_G9	LCD_DATA[5]	Output	PIN_K2
LCD_BLON	Output	PIN_L6	LCD_DATA[4]	Output	PIN_K1
LCD_EN	Output	PIN_L4	LCD_DATA[3]	Output	PIN_K7
LCD_ON	Output	PIN_L5	LCD_DATA[2]	Output	PIN_L2
LCD_RS	Output	PIN_M2	LCD_DATA[1]	Output	PIN_L1
LCD_RW	Output	PIN_M1	LCD_DATA[0]	Output	PIN_L3