THE UNIVERSITY OF DANANG

UNIVERSITY OF SCIENCE AND TECHNOLOGY

Faculty of Advanced Science and Technology

FINAL REPORT

SYSTEM ON CHIPS: DESIGN AND IMPLEMENTATION OF A UART SYSTEM

Instructor	**:** Nguyen Van Cuong
Group	**:**
Class	**:** 21ECE
Student	**:** Nguyen Thi Tam
	Tran Hoang Minh

Da Nang, March 26th 2025

Contents

Abstract

UART verification circuit: We use a loop-back circuit and a PC to verify the UART's operation. In the circuit, the USB-serial port of the DE2 board is connected to the serial port of a PC. When we send a character from the PC, the received data word is stored in the UART receiver's fourword FIFO buffer. When retrieved (via the r_data port), the data word is incremented by 1 and then sent back to the transmitter (via the w_data port). The debounced pushbutton switch produces a single one-clock-cycle tick when pressed and it is connected to the rd_uart and wr_uart signals. When the tick is generated, it removes one word from the receiver's FIFO and writes the incremented word to the transmitter's FIFO for transmission.

I. Introduction

A Universal Asynchronous Receiver and Transmitter (UART) is a critical component for serial communication, widely used with the RS-232 standard to enable data exchange between devices. Unlike synchronous communication, UART does not transmit clock information, requiring the transmitter and receiver to pre-agree on parameters such as baud rate, data bits, stop bits, and parity. This report details the design of a UART system implemented on the DE2 FPGA board, operating at a baud rate of 19,200 with 8 data bits, 1 stop bit, and no parity. The verification process is conducted using a loop-back circuit with a PC, leveraging Hercules to test the UART's operation, with results displayed on the DE2 board's LEDs and seven-segment display.

II. System Overview

The UART system consists of two main subsystems: the receiving subsystem and the transmitting subsystem, integrated into a complete UART core. The system employs a baud rate generator and interface circuits to support serial communication adhering to the RS-232 standard. The DE2 board is equipped with an RS-232 port and a voltage converter chip, enabling direct connection to a PC via a standard serial cable.

1. Transmission Format

Data transmission begins with a start bit (logic 0), followed by 8 data bits (LSB first), and concludes with a stop bit (logic 1). The serial line remains at logic 1 when idle. With the configuration set to 19,200 baud, 8 data bits, 1 stop bit, and no parity, these parameters must be predefined to ensure accurate communication.

III. UART Receiving Subsystem

1. *Oversampling Mechanism*

The receiving subsystem employs a 16x oversampling scheme to estimate the midpoints of incoming bits, compensating for the absence of clock information. The process is as follows:

Detect the falling edge of the start bit (logic 0) and initiate a sampling tick counter.

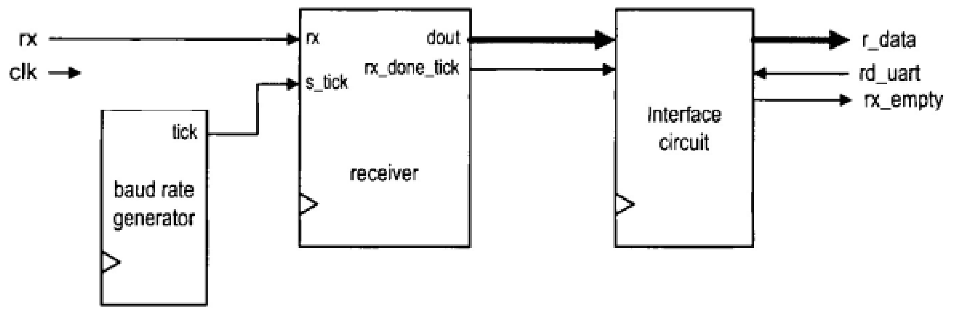
At tick 7, confirm the midpoint of the start bit, reset the counter, and proceed.

At tick 15, sample the first data bit, shift it into a register, and reset the counter.

Repeat step 3 for the remaining 7 data bits.

Sample the stop bit after 16 ticks.

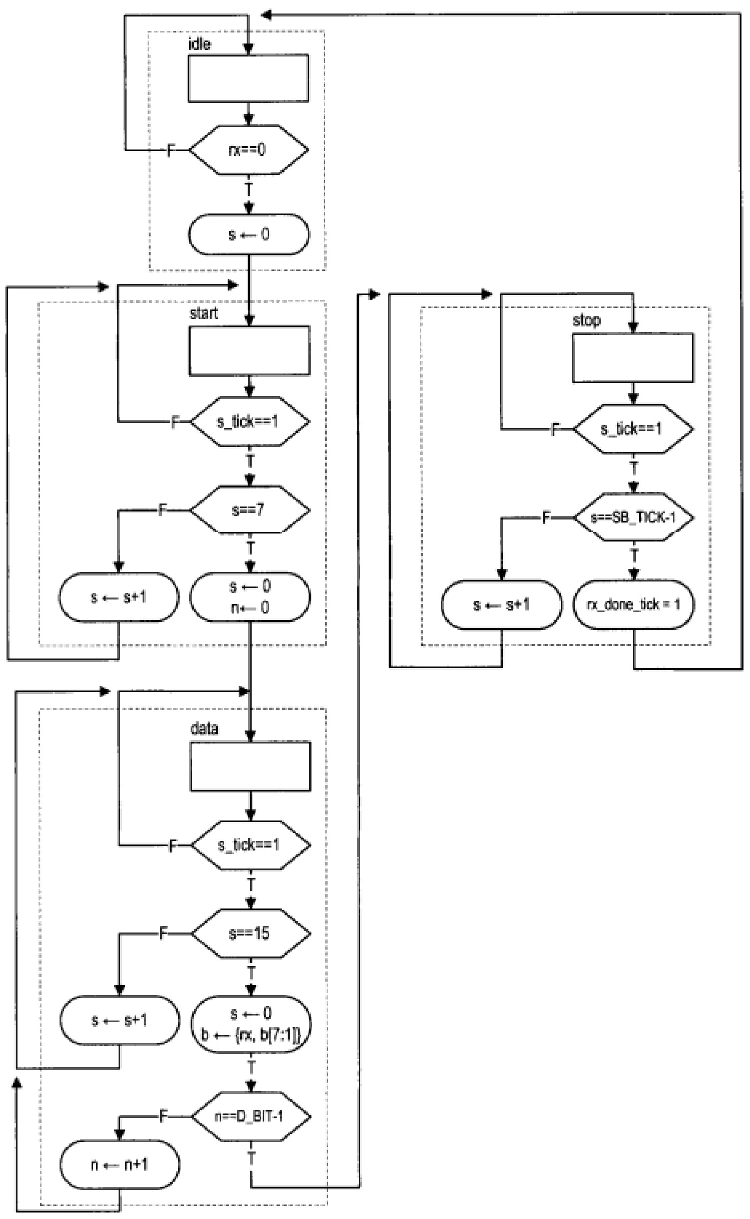
This method ensures accurate bit retrieval with a maximum timing error of 1/16 of a bit period, though it limits the baud rate relative to the 50 MHz system clock.



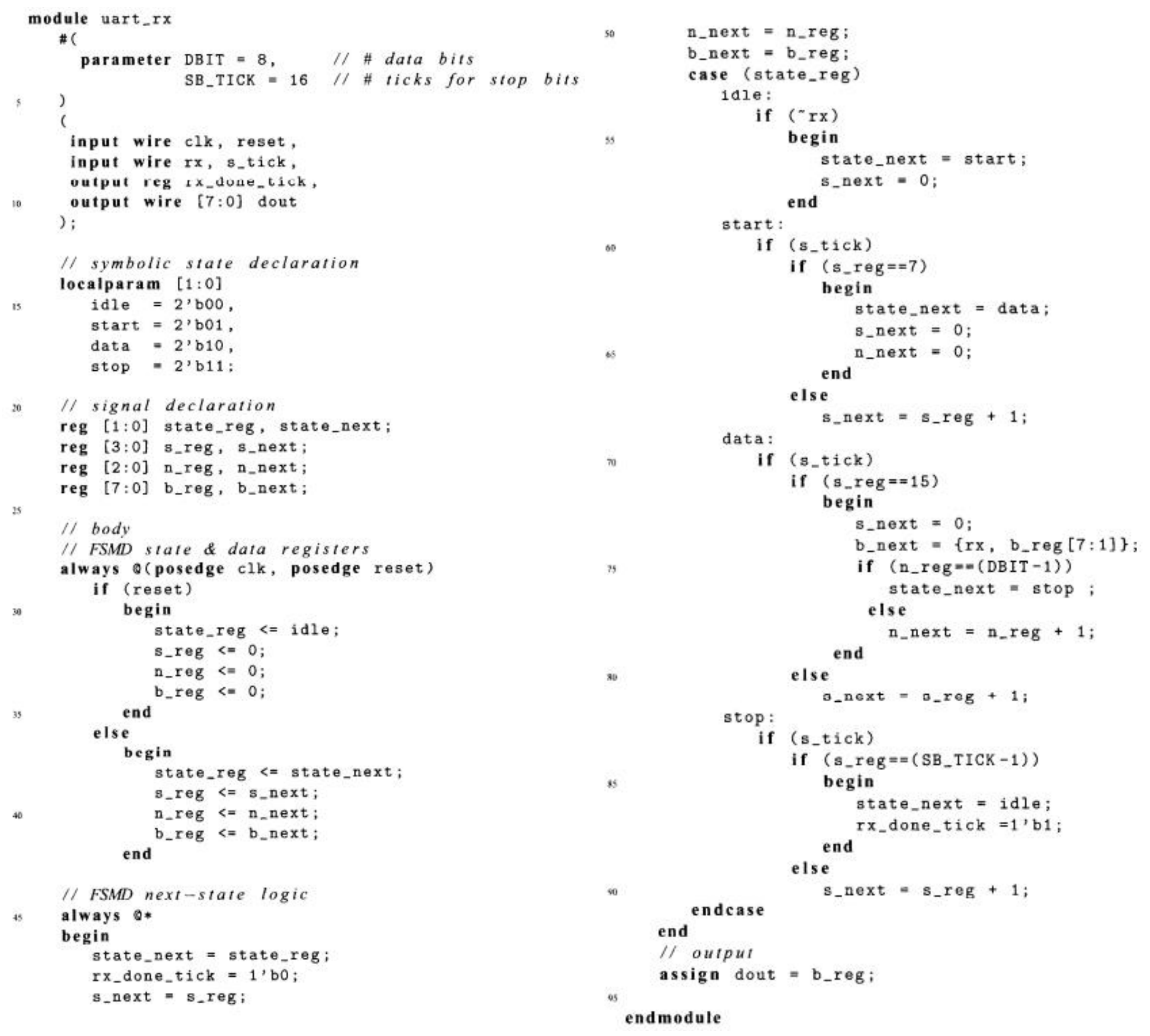
2. *Components*

The receiving subsystem includes:

UART Receiver: Implements oversampling to assemble the data word, with an ASMD chart defining the states: idle, start, data, and stop.



idle
rx==0
F
T
s ← 0
start
s_tick==1
F
T
s==7
F
T
s ← s+1
s ← 0
n← 0
data
s_tick==1
F
T
s==15
F
T
s ← s+1
s ← 0
b ← {rx, b[7:1]}
T
n==D_BIT-1
F
T
n ← n+1
stop
s_tick==1
F
T
s==SB_TICK-1
F
T
s ← s+1
rx_done_tick = 1

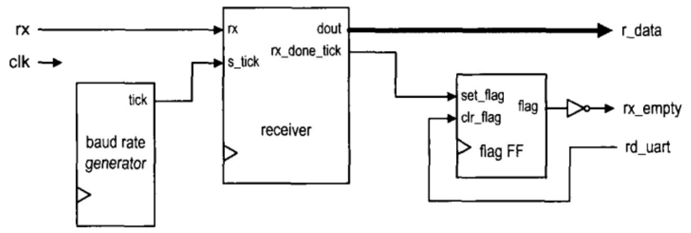


```verilog
module uart_rx
   #(
     parameter DBIT = 8,      // # data bits
               SB_TICK = 16   // # ticks for stop bits
   )
   (
    input wire clk, reset,
    input wire rx, s_tick,
    output reg rx_done_tick,
    output wire [7:0] dout
   );

   // symbolic state declaration
   localparam [1:0]
      idle  = 2'b00,
      start = 2'b01,
      data  = 2'b10,
      stop  = 2'b11;

   // signal declaration
   reg [1:0] state_reg, state_next;
   reg [3:0] s_reg, s_next;
   reg [2:0] n_reg, n_next;
   reg [7:0] b_reg, b_next;

   // body
   // FSMD state & data registers
   always @(posedge clk, posedge reset)
      if (reset)
         begin
            state_reg <= idle;
            s_reg <= 0;
            n_reg <= 0;
            b_reg <= 0;
         end
      else
         begin
            state_reg <= state_next;
            s_reg <= s_next;
            n_reg <= n_next;
            b_reg <= b_next;
         end

   // FSMD next-state logic
   always @*
   begin
      state_next = state_reg;
      rx_done_tick = 1'b0;
      s_next = s_reg;
      n_next = n_reg;
      b_next = b_reg;
      case (state_reg)
         idle:
            if (~rx)
               begin
                  state_next = start;
                  s_next = 0;
               end
         start:
            if (s_tick)
               if (s_reg==7)
                  begin
                     state_next = data;
                     s_next = 0;
                     n_next = 0;
                  end
               else
                  s_next = s_reg + 1;
         data:
            if (s_tick)
               if (s_reg==15)
                  begin
                     s_next = 0;
                     b_next = {rx, b_reg[7:1]};
                     if (n_reg==(DBIT-1))
                        state_next = stop ;
                      else
                        n_next = n_reg + 1;
                   end
               else
                  s_next = s_reg + 1;
         stop:
            if (s_tick)
               if (s_reg==(SB_TICK-1))
                  begin
                     state_next = idle;
                     rx_done_tick =1'b1;
                  end
               else
                  s_next = s_reg + 1;
      endcase
   end
   // output
   assign dout = b_reg;

endmodule
```

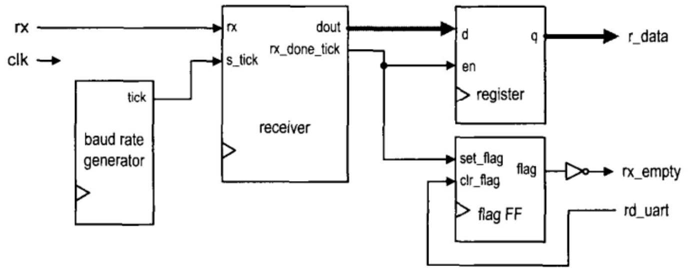
Baud Rate Generator: Generates sampling ticks at 307,200 Hz (19,200 × 16) using a mod-163 counter.

Interface Circuit: Manages data buffering and status signaling, offering three schemes: Flag FF, Flag FF with a one-word buffer, and FIFO buffer (4 words used in this design).

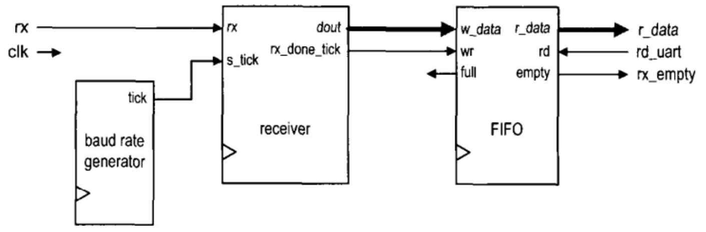
```verilog
module flag_buf
   #(parameter W = 8) // # buffer bits
   (
    input wire clk, reset,
    input wire clr_flag, set_flag,
    input wire [W-1:0] din,
    output wire flag,
    output wire [W-1:0] dout
   );

   // signal declaration
   reg [W-1:0] buf_reg, buf_next;
   reg flag_reg, flag_next;


   // body
   // FF & register
   always @(posedge clk, posedge reset)
      if (reset)
         begin
            buf_reg <= 0;
            flag_reg <= 1'b0;
         end
      else
         begin
            buf_reg <= buf_next;
            flag_reg <= flag_next;
         end

   // next-state logic
   always @*
   begin
      buf_next = buf_reg;
      flag_next = flag_reg;
      if (set_flag)
         begin
            buf_next = din;
            flag_next = 1'b1;
         end
      else if (clr_flag)
         flag_next = 1'b0;
   end
   // output logic
   assign dout = buf_reg;
   assign flag = flag_reg;

endmodule
```



(a) Flag FF



(b) Flag FF and one-word buffer



(c) FIFO buffer

IV. UART Transmitting Subsystem

1. Operation

The transmitting subsystem uses a shift register to serialize parallel data at the baud rate. It shares the baud rate generator’s ticks, shifting out bits every 16 ticks. The transmission progresses through idle, start, data, and stop states, with the tx_done_tick signal indicating completion.

2. Components

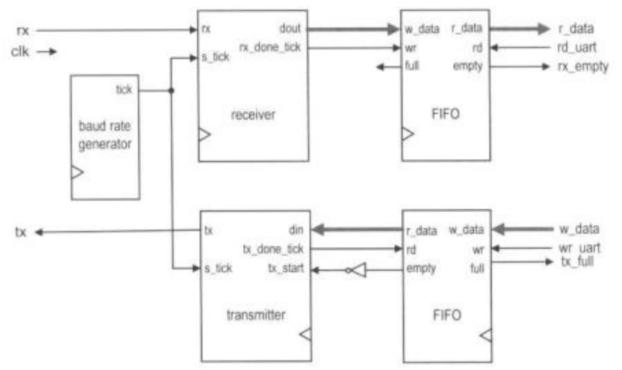
The transmitting subsystem comprises:

UART Transmitter: Shifts out bits with a 1-bit buffer (tx_reg) to filter glitches.

Baud Rate Generator: Reused from the receiver.

Interface Circuit: Similar to the receiver’s, with the main system writing data and the transmitter reading it.

```
module uart_tx
   #(
     parameter DBIT = 8,      // # data bits
               SB_TICK = 16   // # ticks for stop bits
   )
   (
    input wire clk, reset,
    input wire tx_start, s_tick,
    input wire [7:0] din,
    output reg tx_done_tick,
    output wire tx
   );

   // symbolic state declaration
   localparam [1:0]
      idle  = 2'b00,
      start = 2'b01,
      data  = 2'b10,
      stop  = 2'b11;

   // signal declaration
   reg [1:0] state_reg, state_next;
   reg [3:0] s_reg, s_next;
   reg [2:0] n_reg, n_next;
   reg [7:0] b_reg, b_next;
   reg tx_reg, tx_next;

   // body
   // FSMD state & data registers
   always @(posedge clk, posedge reset)
      if (reset)
         begin
            state_reg <= idle;
            s_reg <= 0;
            n_reg <= 0;
            b_reg <= 0;
            tx_reg <= 1'b1;
         end
      else
         begin
            state_reg <= state_next;
            s_reg <= s_next;
            n_reg <= n_next;
            b_reg <= b_next;
            tx_reg <= tx_next;
         end

   // FSMD next-state logic & functional units
   always @*
   begin
      state_next = state_reg;
      tx_done_tick = 1'b0;
      s_next = s_reg;
      n_next = n_reg;
      b_next = b_reg;
      tx_next = tx_reg ;
      case (state_reg)
         idle:
            begin
               tx_next = 1'b1;
               if (tx_start)
                  begin
                     state_next = start;
                     s_next = 0;
                     b_next = din;
                  end
            end
         start:
            begin
               tx_next = 1'b0;
               if (s_tick)
                  if (s_reg==15)
                     begin
                        state_next = data;
                        s_next = 0;
                        n_next = 0;
                     end
                  else
                     s_next = s_reg + 1;
            end
         data:
            begin
               tx_next = b_reg[0];
               if (s_tick)
                  if (s_reg==15)
                     begin
                        s_next = 0;
                        b_next = b_reg >> 1;
                        if (n_reg==(DBIT-1))
                           state_next = stop ;
                        else
                           n_next = n_reg + 1;
                     end
                  else
                     s_next = s_reg + 1;
            end
         stop:
            begin
               tx_next = 1'b1;
               if (s_tick)
                  if (s_reg==(SB_TICK-1))
                     begin
                        state_next = idle;
                        tx_done_tick = 1'b1;
                     end
                  else
                     s_next = s_reg + 1;
            end
      endcase
   end
   // output
   assign tx = tx_reg;

endmodule
```



V. Complete UART Core

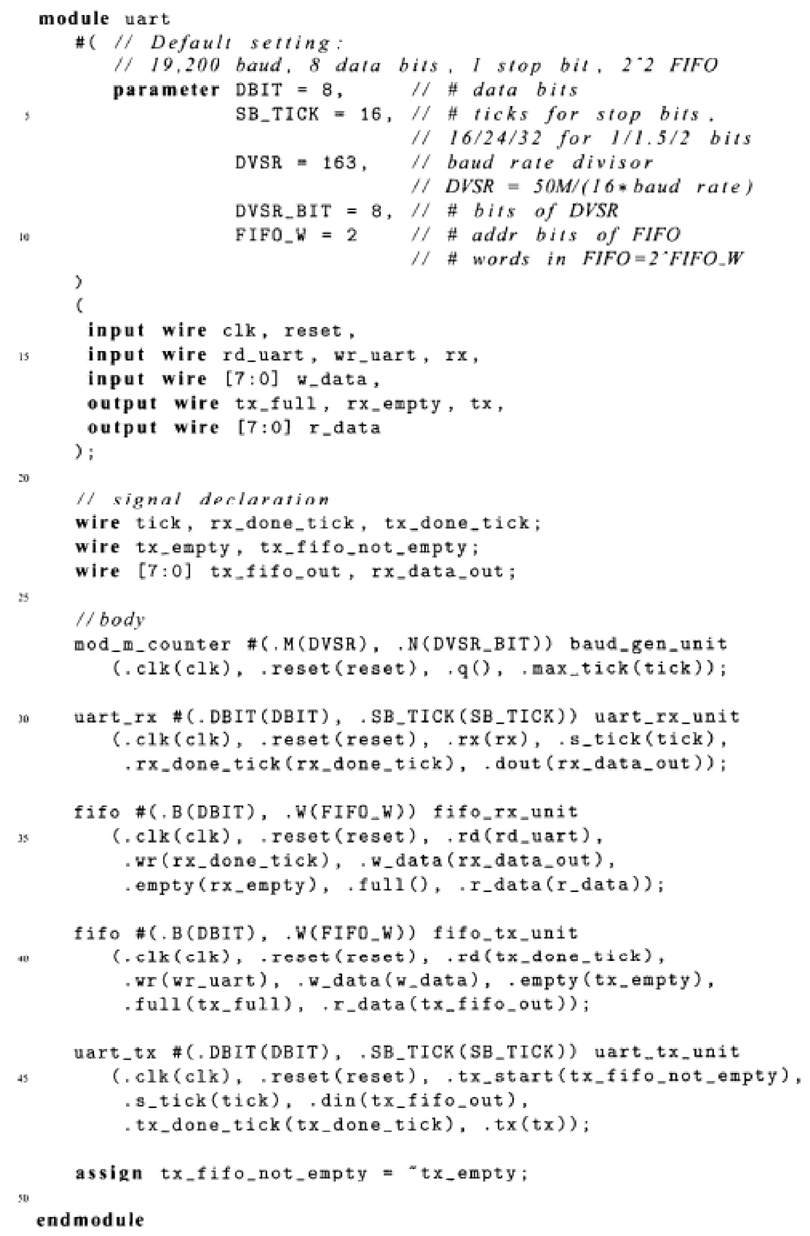
1. *Integration*

The complete UART core integrates the receiving and transmitting subsystems, using 4-word FIFO buffers for both directions. The Verilog module includes:

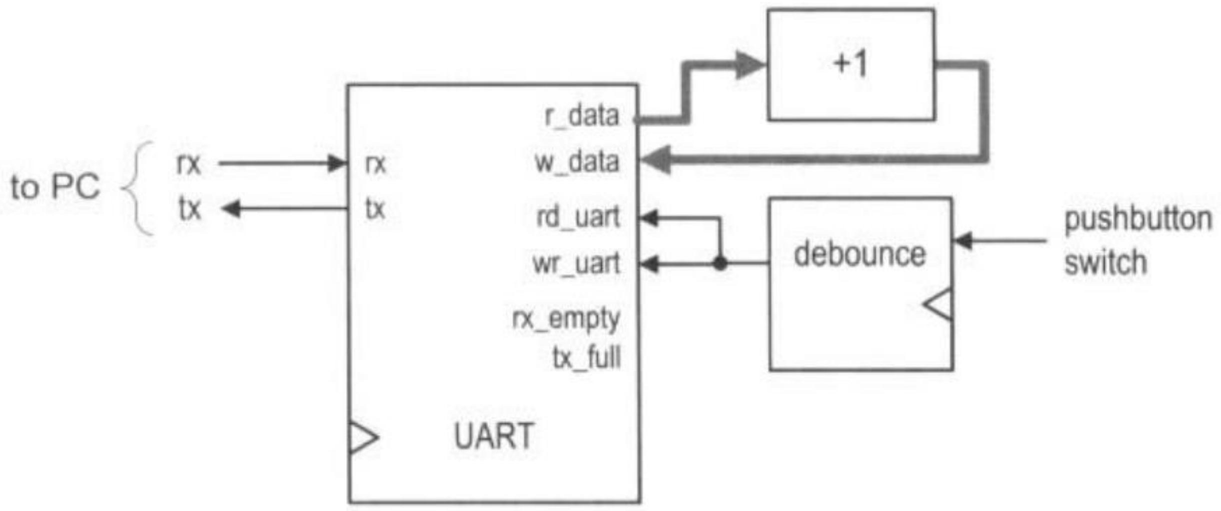
Baud rate generator (mod_m_counter).

Receiver (uart_rx) and transmitter (uart_tx).

Dual FIFO buffers (fifo_rx_unit, fifo_tx_unit) for data flow management.



```
module uart
   #( // Default setting:
      // 19,200 baud, 8 data bits, 1 stop bit, 2^2 FIFO
      parameter DBIT = 8,       // # data bits
                SB_TICK = 16, // # ticks for stop bits,
                              // 16/24/32 for 1/1.5/2 bits
                DVSR = 163,   // baud rate divisor
                              // DVSR = 50M/(16*baud rate)
                DVSR_BIT = 8, // # bits of DVSR
                FIFO_W = 2    // # addr bits of FIFO
                              // # words in FIFO=2^FIFO_W
   )
   (
    input wire clk, reset,
    input wire rd_uart, wr_uart, rx,
    input wire [7:0] w_data,
    output wire tx_full, rx_empty, tx,
    output wire [7:0] r_data
   );

   // signal declaration
   wire tick, rx_done_tick, tx_done_tick;
   wire tx_empty, tx_fifo_not_empty;
   wire [7:0] tx_fifo_out, rx_data_out;

   //body
   mod_m_counter #(.M(DVSR), .N(DVSR_BIT)) baud_gen_unit
      (.clk(clk), .reset(reset), .q(), .max_tick(tick));

   uart_rx #(.DBIT(DBIT), .SB_TICK(SB_TICK)) uart_rx_unit
      (.clk(clk), .reset(reset), .rx(rx), .s_tick(tick),
       .rx_done_tick(rx_done_tick), .dout(rx_data_out));

   fifo #(.B(DBIT), .W(FIFO_W)) fifo_rx_unit
      (.clk(clk), .reset(reset), .rd(rd_uart),
       .wr(rx_done_tick), .w_data(rx_data_out),
       .empty(rx_empty), .full(), .r_data(r_data));

   fifo #(.B(DBIT), .W(FIFO_W)) fifo_tx_unit
      (.clk(clk), .reset(reset), .rd(tx_done_tick),
       .wr(wr_uart), .w_data(w_data), .empty(tx_empty),
       .full(tx_full), .r_data(tx_fifo_out));

   uart_tx #(.DBIT(DBIT), .SB_TICK(SB_TICK)) uart_tx_unit
      (.clk(clk), .reset(reset), .tx_start(tx_fifo_not_empty),
       .s_tick(tick), .din(tx_fifo_out),
       .tx_done_tick(tx_done_tick), .tx(tx));

   assign tx_fifo_not_empty = ~tx_empty;

endmodule
```



2. *Verification*

The verification process utilizes a loop-back circuit with a PC to validate the UART's operation on the DE2 board. The setup and operation are as follows:

Configuration: The DE2 board's serial port is connected to the PC's serial port. The PC uses Hercules, configured for 19,200 baud, 8 data bits, 1 stop bit, and no parity.

Operation: When a character is sent from the PC (e.g., "HAL"), the received data word is stored in the UART receiver's 4-word FIFO buffer. Upon retrieval (via the r_data port), the data word is incremented by 1 (e.g., "HAL" → "IBM") and sent back to the transmitter (via the w_data port).

Control: A debounced pushbutton switch on the DE2 board generates a single one-clock-cycle tick when pressed. This tick is connected to the rd_uart and wr_uart signals, enabling the removal of one word from the receiver's FIFO and writing the incremented word to the transmitter's FIFO for transmission.

Display: The UART's r_data port is connected to the eight LEDs on the DE2 board to display the received data word. The tx_full and rx_empty signals are connected to the two horizontal bars of the rightmost digit of the seven-segment display, allowing observation of the transmitter and receiver FIFO states.

Illustrative Example: If the user types "HAL" on the PC, the three data words are stored in the receiver's FIFO buffer. Pressing the button on the DE2 board three times will transmit the three successive characters "IBM" back to the PC, which are then displayed on Hercules. Simultaneously, the LEDs on the DE2 board show the value of the received data word, and the seven-segment display indicates the FIFO status (full or empty).

```
module uart_test
   (
    input wire clk, reset,
    input wire rx,
    input wire [2:0] btn,
    output wire tx,
    output wire [3:0] an,
    output wire [7:0] sseg, led
   );

   // signal declaration
   wire tx_full, rx_empty, btn_tick;
   wire [7:0] rec_data, rec_data1;

   // body
   // instantiate uart
   uart uart_unit
      (.clk(clk), .reset(reset), .rd_uart(btn_tick),
       .wr_uart(btn_tick), .rx(rx), .w_data(rec_data1),
       .tx_full(tx_full), .rx_empty(rx_empty),
       .r_data(rec_data), .tx(tx));
   // instantiate debounce circuit
   debounce btn_db_unit
      (.clk(clk), .reset(reset), .sw(btn[0]),
       .db_level(), .db_tick(btn_tick));
   // incremented data loops back
   assign rec_data1 = rec_data + 1;
   // LED display
   assign led = rec_data;
   assign an = 4'b1110;
   assign sseg = {1'b1, ~tx_full, 2'b11, ~rx_empty, 3'b111};

 endmodule
```

VI. Discussion

This UART design strikes a balance between simplicity and functionality, effectively utilizing the DE2 board's resources. The oversampling approach limits high-speed data applications, but the 4-word FIFO buffer ensures reliability in low-to-moderate data rate scenarios. The loop-back verification demonstrates the UART's correct operation, with visual feedback via LEDs and the seven-segment display facilitating easy debugging and monitoring.

ASCII code: In Hercules, characters are sent in ASCII code, which is 7 bits and consists of 128 code words, including regular alphabets, digits, punctuation symbols, and nonprintable control characters. The characters and their code words (in hexadecimal for-mat) are shown in the following table:

Code	Char	Code	Char	Code	Char	Code	Char
00	(nul)	20	(sp)	40	@	60	‘
01	(soh)	21	!	41	A	61	a
02	(stx)	22	"	42	B	62	b
03	(etx)	23	#	43	C	63	c
04	(eot)	24	$	44	D	64	d
05	(enq)	25	%	45	E	65	e
06	(ack)	26	&	46	F	66	f
07	(bel)	27	’	47	G	67	g
08	(bs)	28	(	48	H	68	h
09	(ht)	29	)	49	I	69	i
0a	(nl)	2a	*	4a	J	6a	j
0b	(vt)	2b	+	4b	K	6b	k
0c	(np)	2c	,	4c	L	6c	l
0d	(cr)	2d	-	4d	M	6d	m
0e	(so)	2e	.	4e	N	6e	n
0f	(si)	2f	/	4f	O	6f	o
10	(dle)	30	0	50	P	70	p
11	(dc1)	31	1	51	Q	71	q
12	(dc2)	32	2	52	R	72	r
13	(dc3)	33	3	53	S	73	s
14	(dc4)	34	4	54	T	74	t
15	(nak)	35	5	55	U	75	u
16	(syn)	36	6	56	V	76	v
17	(etb)	37	7	57	W	77	w
18	(can)	38	8	58	X	78	x
19	(em)	39	9	59	Y	79	y
1a	(sub)	3a	:	5a	Z	7a	z
1b	(esc)	3b	;	5b	[	7b	{
1c	(fs)	3c	<	5c	\	7c	|
1d	(gs)	3d	=	5d	]	7d	}
1e	(rs)	3e	>	5e	^	7e	~
1f	(us)	3f	?	5f	_	7f	(del)

VII. Setting and Result

Download and install Quartus II (version 12.0 or later, as mentioned in the DE2 User Manual). Ensure the USB Blaster driver is installed to connect the DE2 board to computer (refer to "Getting Started with Altera's DE2 Board").

 Create a New Project:

- Open Quartus II and create a new project:
  - File → New Project Wizard.
  - Name the project (e.g., UART_DE2_Test).
  - Select a project directory.
  - Choose the FPGA device: **Cyclone II EP2C35F896C6** (or **EP2C35F896C6N** per the DE2_70 or DE2_105 User Manual).
- Add the provided Verilog files in text book:
  - fifo.v (Listing 4.20)
  - mod_m_counter.v (Listing 4.3)
  - debounce.v (Listing 6.2)
  - uart.v (Listing 8.4)
  - uart_test.v (Listing 8.5)
- Set uart_test.v as the top-level module.

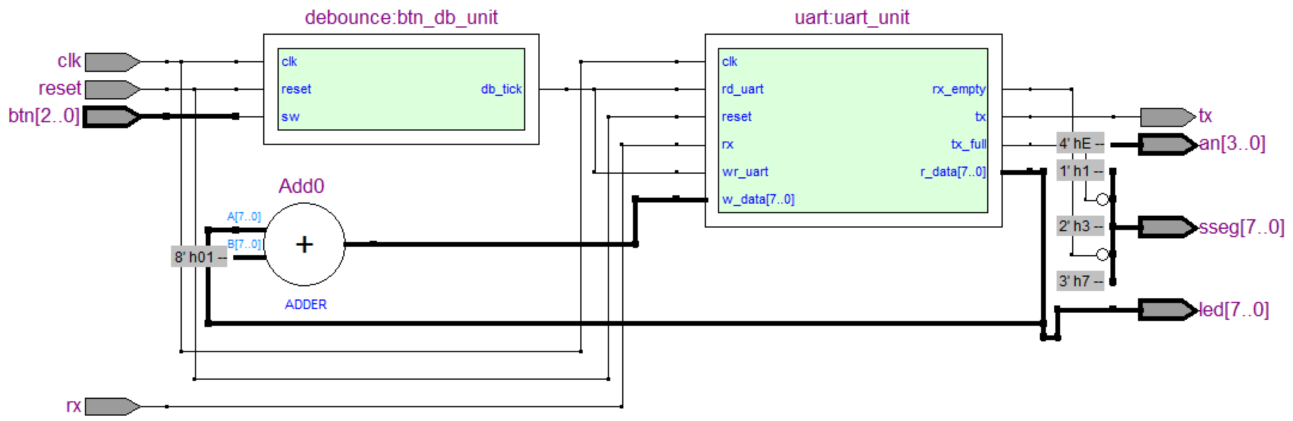
 Assign Pins (Pin Assignments):

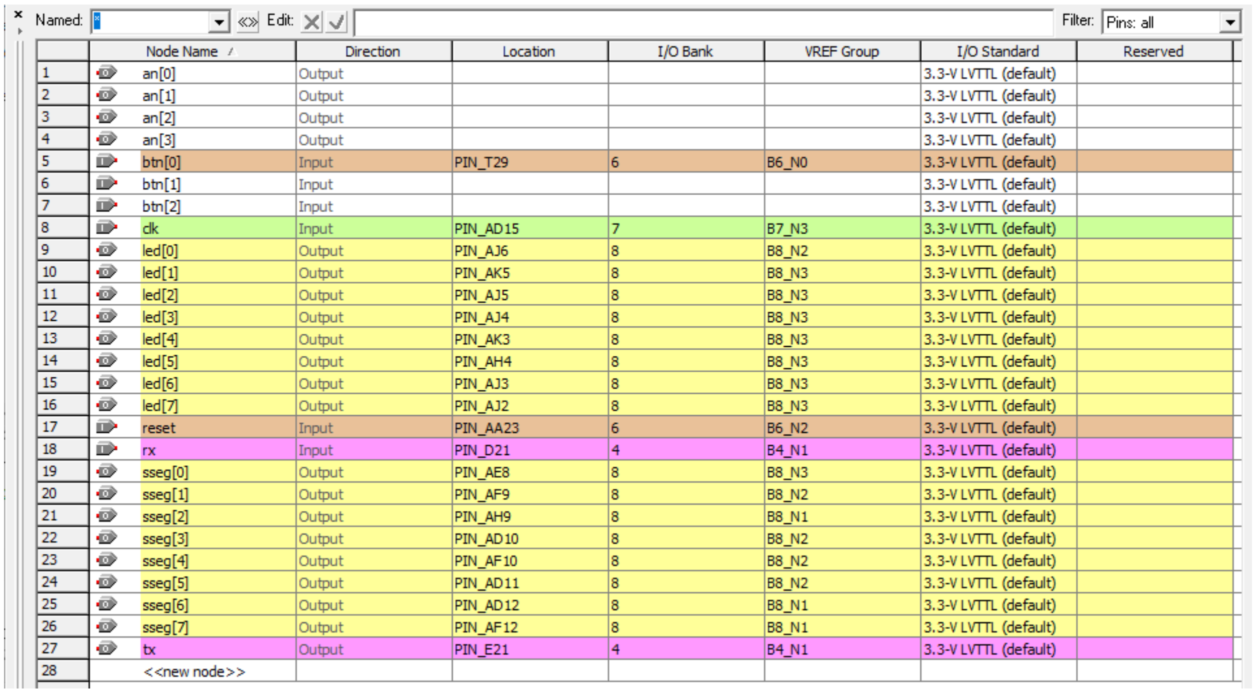
- Use the pin assignments provided earlier:
  - clk → PIN_AD15 (50 MHz clock)
  - reset → PIN_AA23 (SW[0])
  - btn[0] → PIN_T29 (KEY[0])
  - btn[1] →
  - btn[2] →
  - rx → PIN_D21 (UART_RXD)
  - tx → PIN_E21 (UART_TXD)
  - led[7:0] → (LEDG[7:0])
  - sseg[6:0] → (LEDR[6:0])
- Open **Assignments → Pin Planner** in Quartus II and enter these pin assignments. Alternatively, add them to the project's .qsf file (see prior response for pin details).

 Compile the Project:

- Click **Processing → Start Compilation** to compile the project.
- Ensure there are no errors or critical warnings. If errors occur, check the Verilog syntax or pin assignments.

Connect the DE2 board to computer via the USB Blaster port (using the provided USB cable). Plug in the 7.5V DC power supply to the DE2 board. Set the RUN/PROG switch on the DE2 board to RUN.

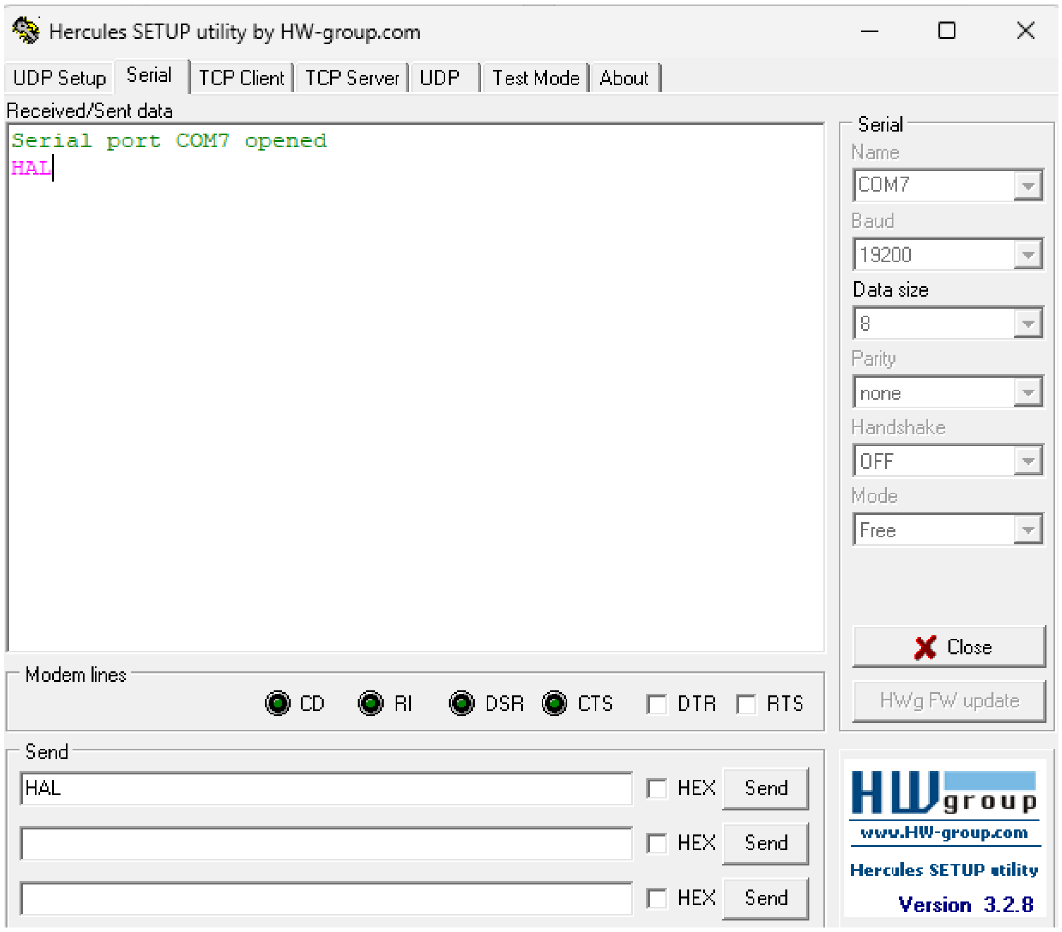




	Node Name	Direction	Location	I/O Bank	VREF Group	I/O Standard	Reserved
1	an[0]	Output				3.3-V LVTTL (default)	
2	an[1]	Output				3.3-V LVTTL (default)	
3	an[2]	Output				3.3-V LVTTL (default)	
4	an[3]	Output				3.3-V LVTTL (default)	
5	btn[0]	Input	PIN_T29	6	B6_N0	3.3-V LVTTL (default)	
6	btn[1]	Input				3.3-V LVTTL (default)	
7	btn[2]	Input				3.3-V LVTTL (default)	
8	clk	Input	PIN_AD15	7	B7_N3	3.3-V LVTTL (default)	
9	led[0]	Output	PIN_AJ6	8	B8_N2	3.3-V LVTTL (default)	
10	led[1]	Output	PIN_AK5	8	B8_N3	3.3-V LVTTL (default)	
11	led[2]	Output	PIN_AJ5	8	B8_N3	3.3-V LVTTL (default)	
12	led[3]	Output	PIN_AJ4	8	B8_N3	3.3-V LVTTL (default)	
13	led[4]	Output	PIN_AK3	8	B8_N3	3.3-V LVTTL (default)	
14	led[5]	Output	PIN_AH4	8	B8_N3	3.3-V LVTTL (default)	
15	led[6]	Output	PIN_AJ3	8	B8_N3	3.3-V LVTTL (default)	
16	led[7]	Output	PIN_AJ2	8	B8_N3	3.3-V LVTTL (default)	
17	reset	Input	PIN_AA23	6	B6_N2	3.3-V LVTTL (default)	
18	rx	Input	PIN_D21	4	B4_N1	3.3-V LVTTL (default)	
19	sseg[0]	Output	PIN_AE8	8	B8_N3	3.3-V LVTTL (default)	
20	sseg[1]	Output	PIN_AF9	8	B8_N2	3.3-V LVTTL (default)	
21	sseg[2]	Output	PIN_AH9	8	B8_N1	3.3-V LVTTL (default)	
22	sseg[3]	Output	PIN_AD10	8	B8_N2	3.3-V LVTTL (default)	
23	sseg[4]	Output	PIN_AF10	8	B8_N2	3.3-V LVTTL (default)	
24	sseg[5]	Output	PIN_AD11	8	B8_N2	3.3-V LVTTL (default)	
25	sseg[6]	Output	PIN_AD12	8	B8_N1	3.3-V LVTTL (default)	
26	sseg[7]	Output	PIN_AF12	8	B8_N1	3.3-V LVTTL (default)	
27	tx	Output	PIN_E21	4	B4_N1	3.3-V LVTTL (default)	
28	<<new node>>						

Connect the RS-232 port on the DE2 board to computer using an USB-to-RS232 adapter. Install a terminal program Hercules (e.g., PuTTY, Tera Term, or HyperTerminal) on computer for RS-232 communication. Configure the terminal with the following settings

- Baud Rate: 115,200 (or 19,200 if using the UART's default setting).
- Parity Check Bit: None
- Data Bits: 8
- Stop Bits: 1
- Flow Control: ON (or OFF if not required).

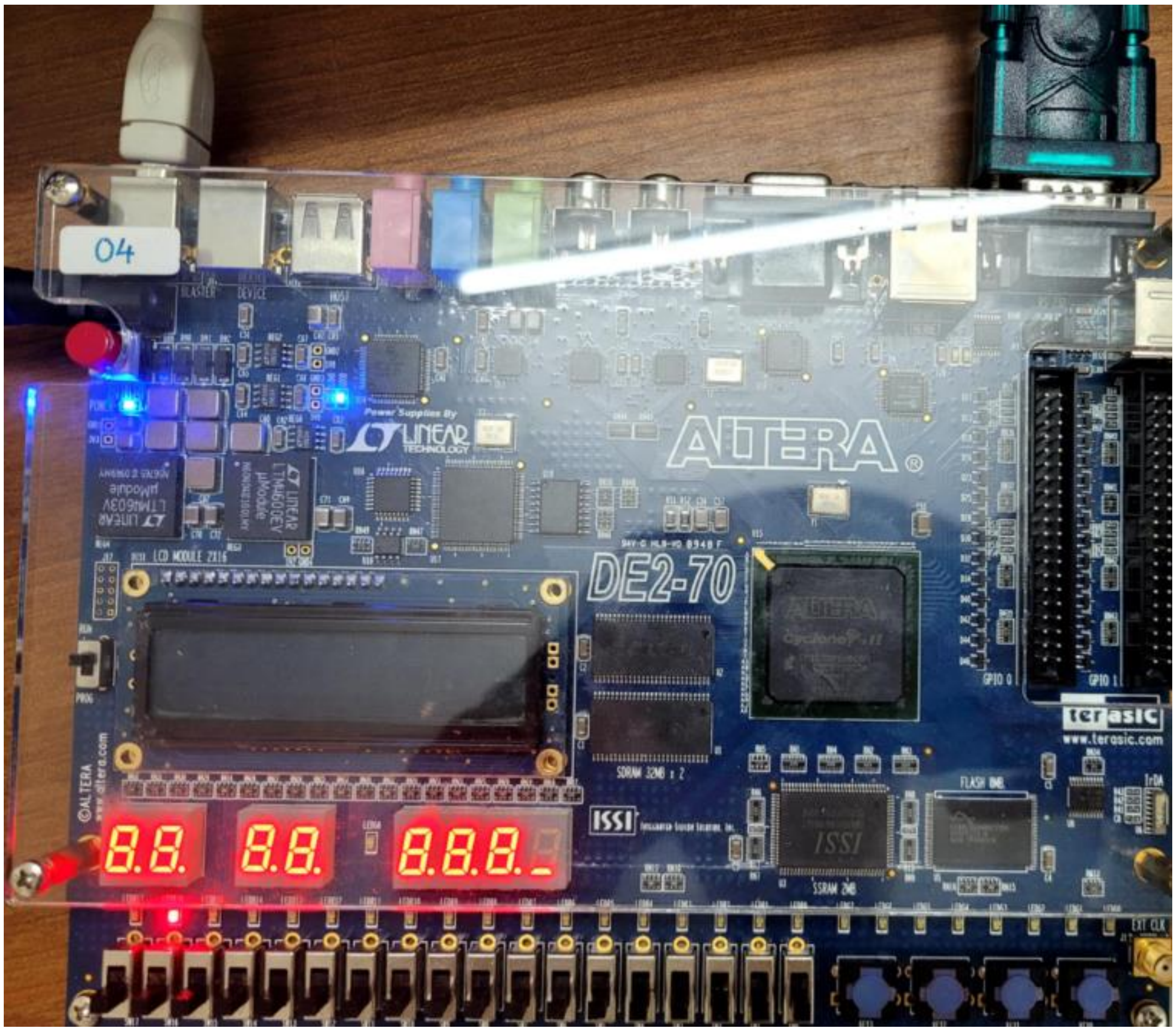


```
Serial port COM7 opened
HAL
```

Serial
Name: COM7
Baud: 19200
Data size: 8
Parity: none
Handshake: OFF
Mode: Free

Send: HAL

To send the word "HAL" through the UART system on the Altera DE2 board and verify its operation based on the setup described earlier, follow these steps. I'll explain how the system processes each character, what should observe on the DE2 board, and what appears on the terminal. This assumes UART system is configured as per the previous guide (e.g., baud rate 19,200, DVSR = 163, 50 MHz clock, and the uart_test module incrementing received data by 1 before sending it back).



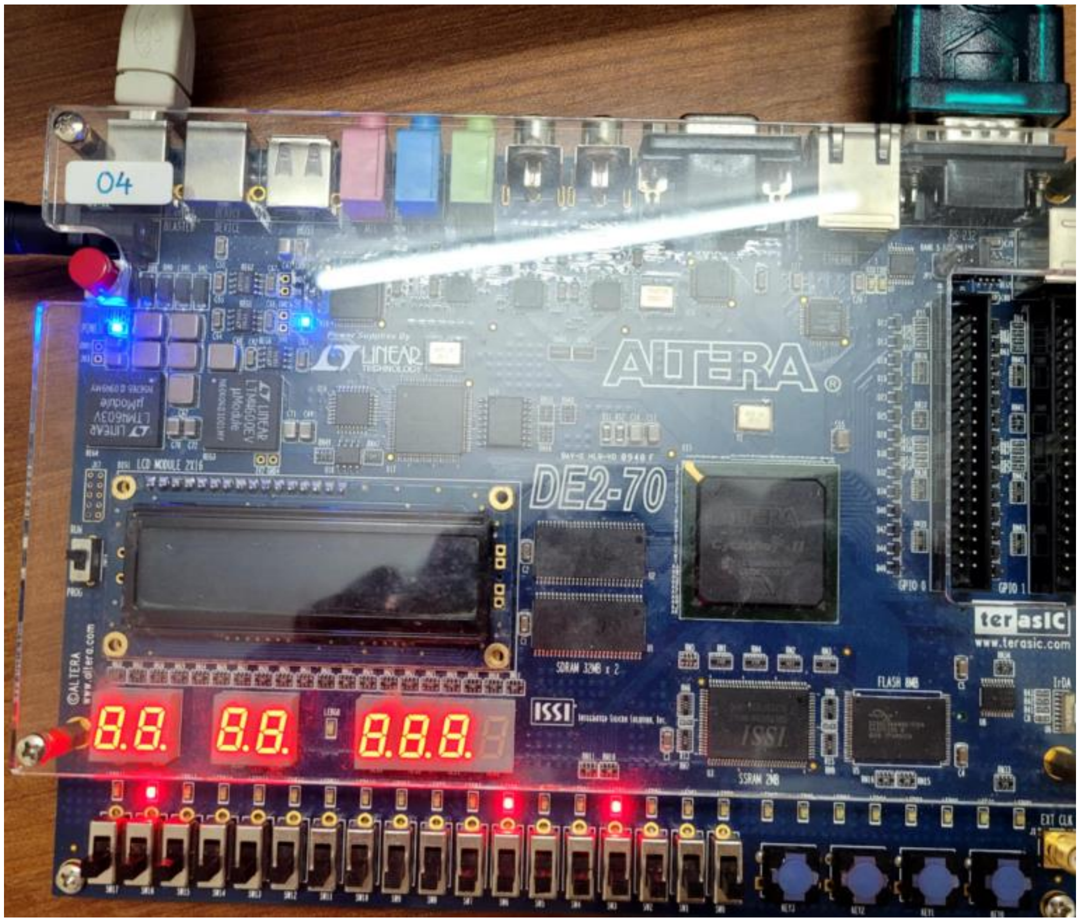
Sending "HAL" and Observing the Results

The word "HAL" consists of three characters: 'H', 'A', and 'L'. We send each character individually from the terminal, observe the received data on the LEDs, press a button to process it, and check the transmitted data on the terminal. The system increments each received character by 1 before sending it back (e.g., 'H' → 'I').

1. Send 'H'

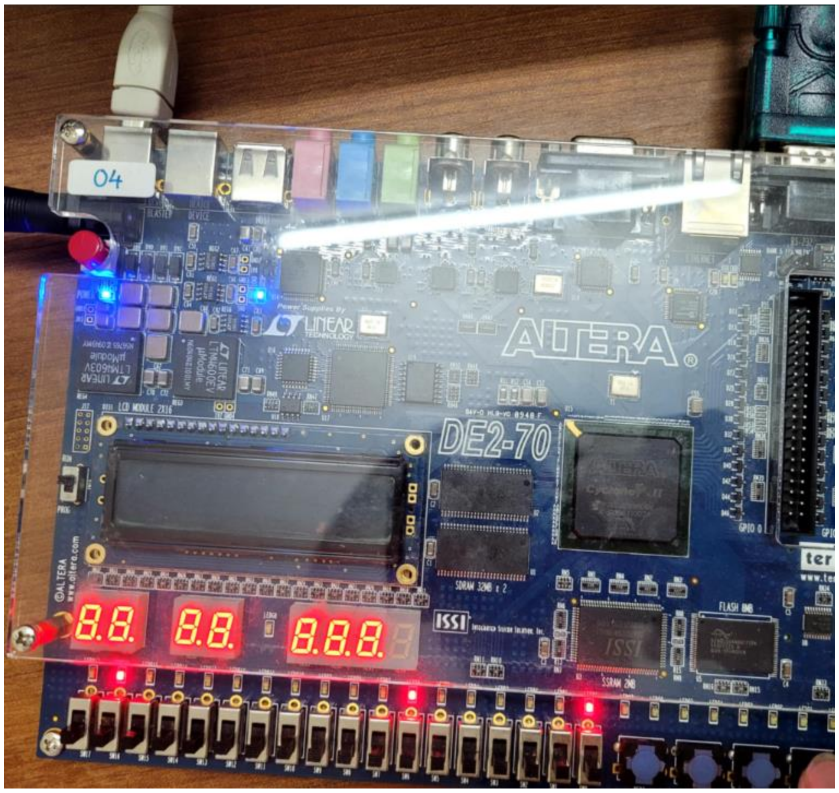
- **ASCII Code**: 'H' = 0x48 (binary: 0100_1000).
- **Action**: Type 'H' in the terminal and press Enter (or send it directly depending on terminal settings).
- **What Happens on DE2 Board**:

- The UART receives 'H' via rx (PIN_F14).
  - The received data (rec_data) is stored in the receive FIFO.
  - rx_empty changes from 1 to 0 → segment d (HEX0[3]) turns off.
  - LEDG[7:0] displays 0x48:
    - 0100_1000 → LEDG[7] = 0, LEDG[6] = 1, LEDG[5] = 0, LEDG[4] = 0, LEDG[3] = 1, LEDG[2] = 0, LEDG[1] = 0, LEDG[0] = 0.
    - LEDs 6 and 3 light up; others remain off.
- **Process the Data**:
  - Press KEY[1] (PIN_R21) to generate btn_tick.
  - The system reads 0x48 from the receive FIFO, increments it by 1 (0x48 + 1 = 0x49), and writes 0x49 (ASCII 'I') to the transmit FIFO.
  - The UART transmits 'I' via tx (PIN_G12).
- **Terminal Output**: 'I' appears on the terminal.
- **Post-Action Check**:
  - If no more data is in the receive FIFO, rx_empty = 1 → segment d lights up again.
  - If the transmit FIFO isn't full, tx_full = 0 → segment g remains off.



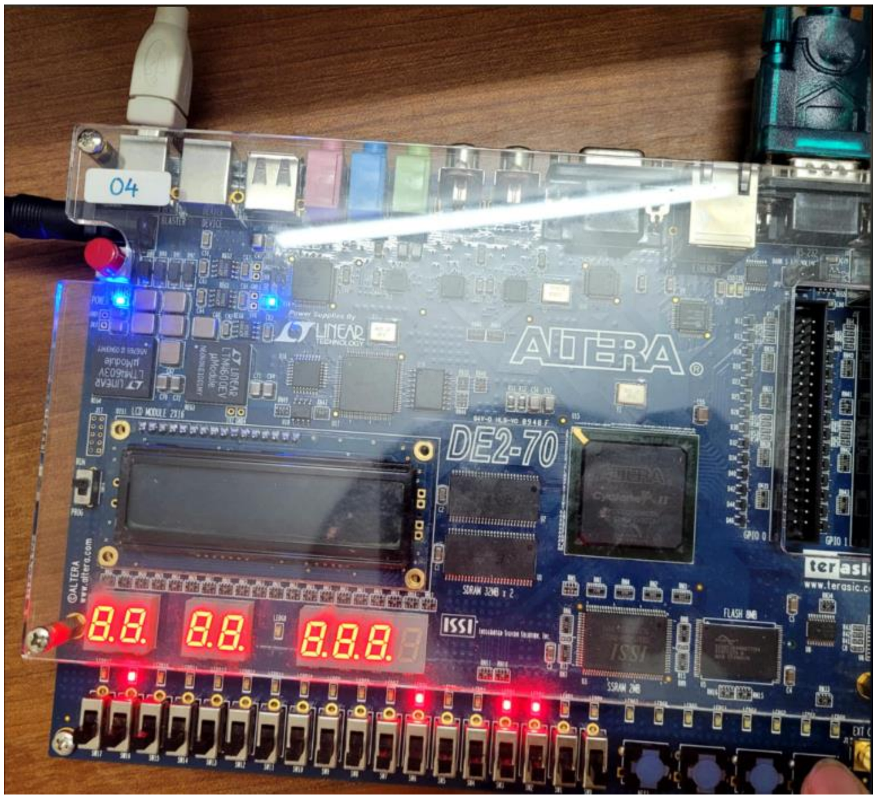
2. Send 'A'

- **ASCII Code**: 'A' = 0x41 (binary: 0100_0001).
- **Action**: Type 'A' in the terminal and press Enter.
- **What Happens on DE2 Board**:
  - The UART receives 'A' via rx.
  - rec_data = 0x41 is stored in the receive FIFO.
  - rx_empty = 0 → segment d turns off.
  - LEDG[7:0] displays 0x41:
    - 0100_0001 → LEDG[7] = 0, LEDG[6] = 1, LEDG[5] = 0, LEDG[4] = 0, LEDG[3] = 0, LEDG[2] = 0, LEDG[1] = 0, LEDG[0] = 1.
    - LEDs 6 and 0 light up; others remain off.
- **Process the Data**:
  - Press KEY[1] to generate btn_tick.
  - The system reads 0x41, increments it (0x41 + 1 = 0x42), and writes 0x42 (ASCII 'B') to the transmit FIFO.
  - The UART transmits 'B'.
- **Terminal Output**: 'B' appears on the terminal (after 'I', so we see "IB").
- **Post-Action Check**:
  - rx_empty = 1 → segment d lights up (if no more data).
  - tx_full = 0 → segment g remains off.

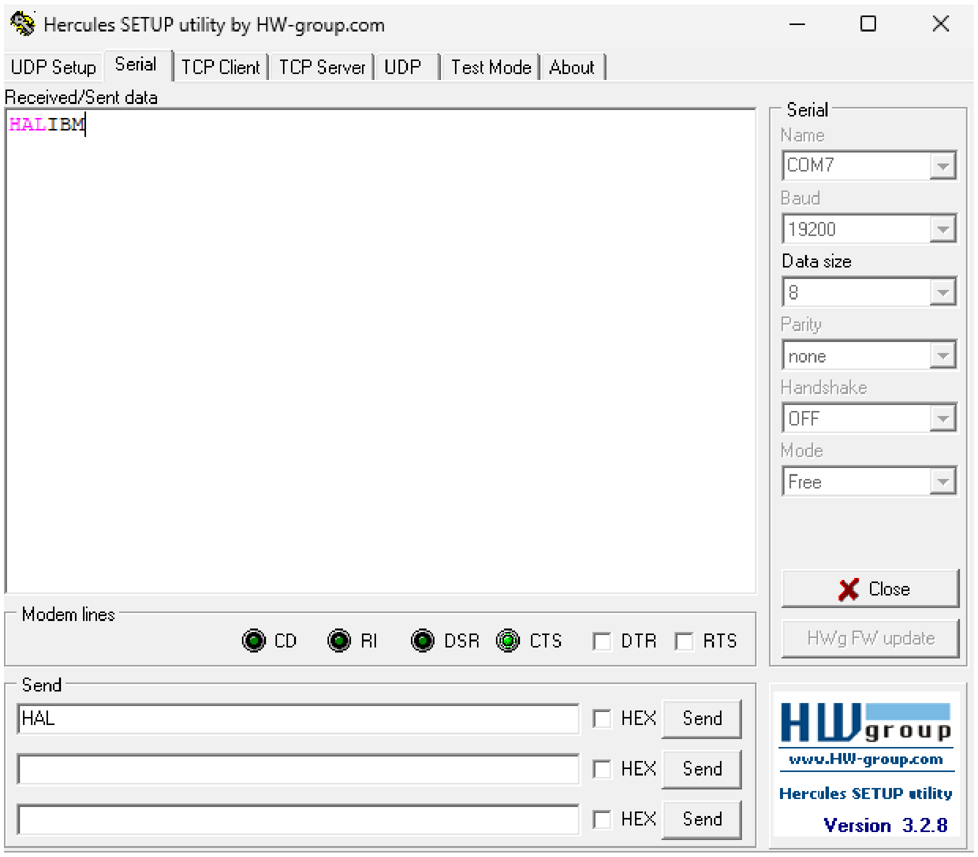


3. Send 'L'

- **ASCII Code**: 'L' = 0x4C (binary: 0100_1100).
- **Action**: Type 'L' in the terminal and press Enter.
- **What Happens on DE2 Board**:
  - The UART receives 'L' via rx.
  - rec_data = 0x4C is stored in the receive FIFO.
  - rx_empty = 0 → segment d turns off.
  - LEDG[7:0] displays 0x4C:
    - 0100_1100 → LEDG[7] = 0, LEDG[6] = 1, LEDG[5] = 0, LEDG[4] = 0, LEDG[3] = 1, LEDG[2] = 1, LEDG[1] = 0, LEDG[0] = 0.
    - LEDs 6, 3, and 2 light up; others remain off.
- **Process the Data**:
  - Press KEY[1] to generate btn_tick.
  - The system reads 0x4C, increments it (0x4C + 1 = 0x4D), and writes 0x4D (ASCII 'M') to the transmit FIFO.
  - The UART transmits 'M'.
- **Terminal Output**: 'M' appears on the terminal (after "IB", so we see "IBM").
- **Post-Action Check**:
  - rx_empty = 1 → segment d lights up (if no more data).
  - tx_full = 0 → segment g remains off.



After pressing KEY[1] for each character, we receive "IBM" (I → B → M).


Hercules SETUP utility by HW-group.com

Received/Sent data

```
HALIBM
```

Serial
Name: COM7
Baud: 19200
Data size: 8
Parity: none
Handshake: OFF
Mode: Free

Send
HAL

VIII. Conclusion

This report has detailed the design and verification of a UART system on the DE2 FPGA board, configured for 19,200 baud, 8 data bits, 1 stop bit, and no parity. The receiving subsystem employs oversampling, the transmitting subsystem uses a shift register, and the integrated UART core with FIFO buffering provides a robust serial communication solution. The loop-back verification, combined with LED and seven-segment display outputs, confirms the design's practicality, making it a valuable example for FPGA-based prototyping.

References

Chu, P. P. (2008). *FPGA Prototyping by Verilog Examples*. John Wiley & Sons, Inc.