Project – SerialControl

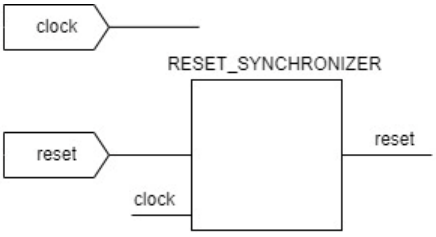
Sofiyan Monga

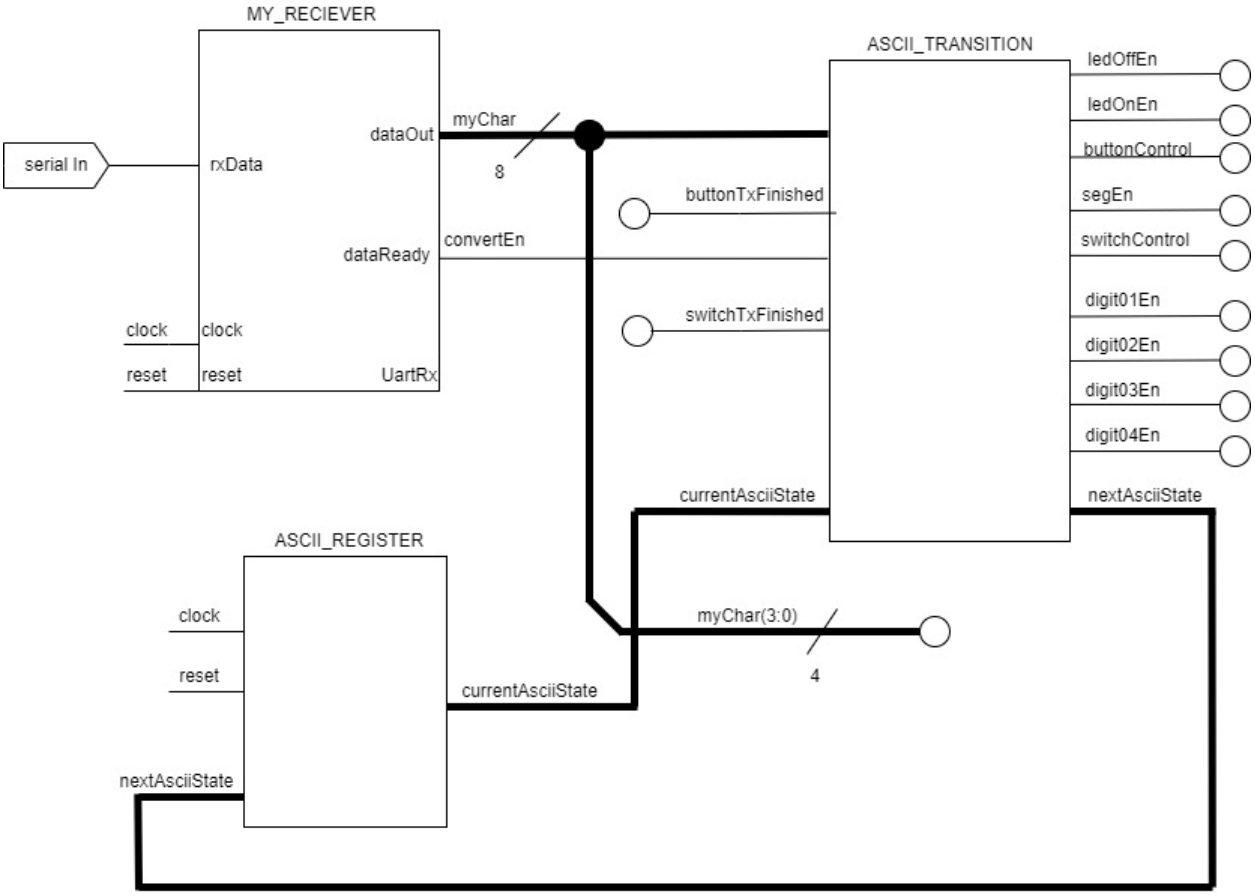
CpE 3020 – VHDL Design with FPGAs

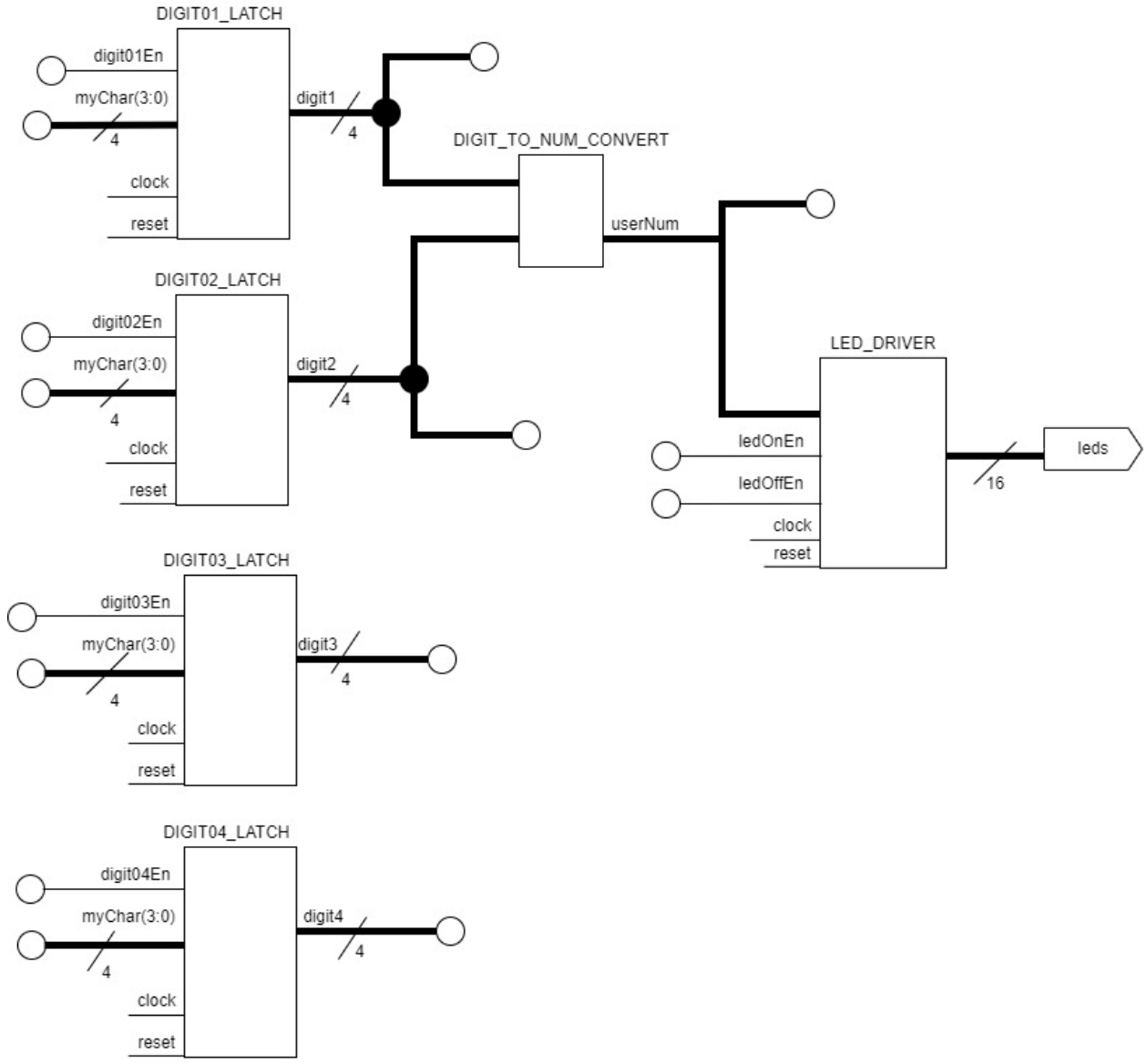
Spring 2022

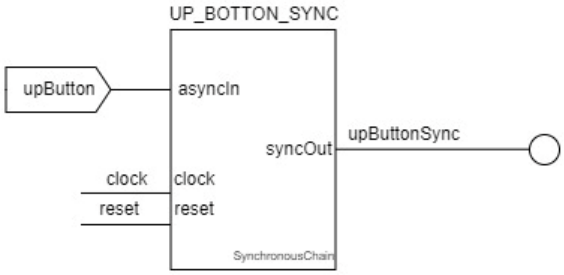
04/27/2022

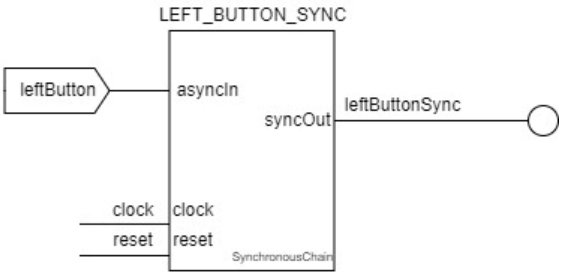
Block Diagram:

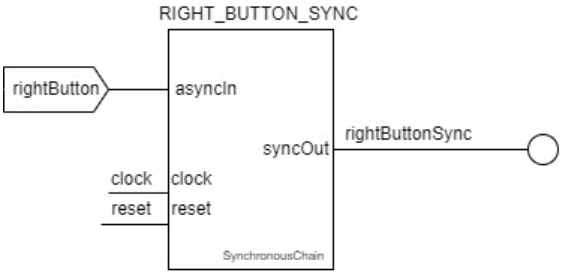


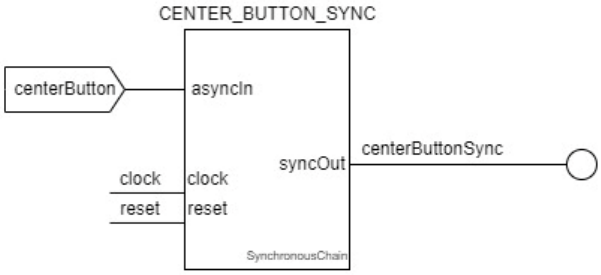


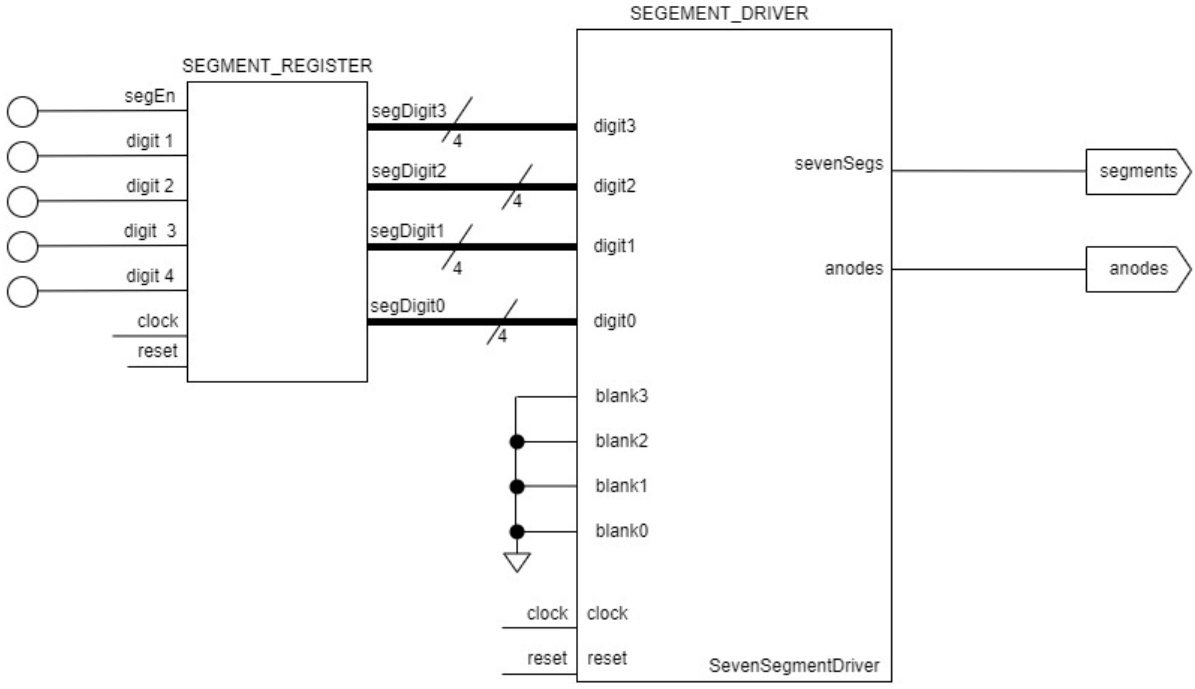

DIGIT01_LATCH
digit01En
myChar(3:0)
4
clock
reset
digit1
4
DIGIT_TO_NUM_CONVERT
userNum
DIGIT02_LATCH
digit02En
myChar(3:0)
4
clock
reset
digit2
4
LED_DRIVER
ledOnEn
ledOffEn
clock
reset
16
leds
DIGIT03_LATCH
digit03En
myChar(3:0)
4
clock
reset
digit3
4
DIGIT04_LATCH
digit04En
myChar(3:0)
4
clock
reset
digit4
4

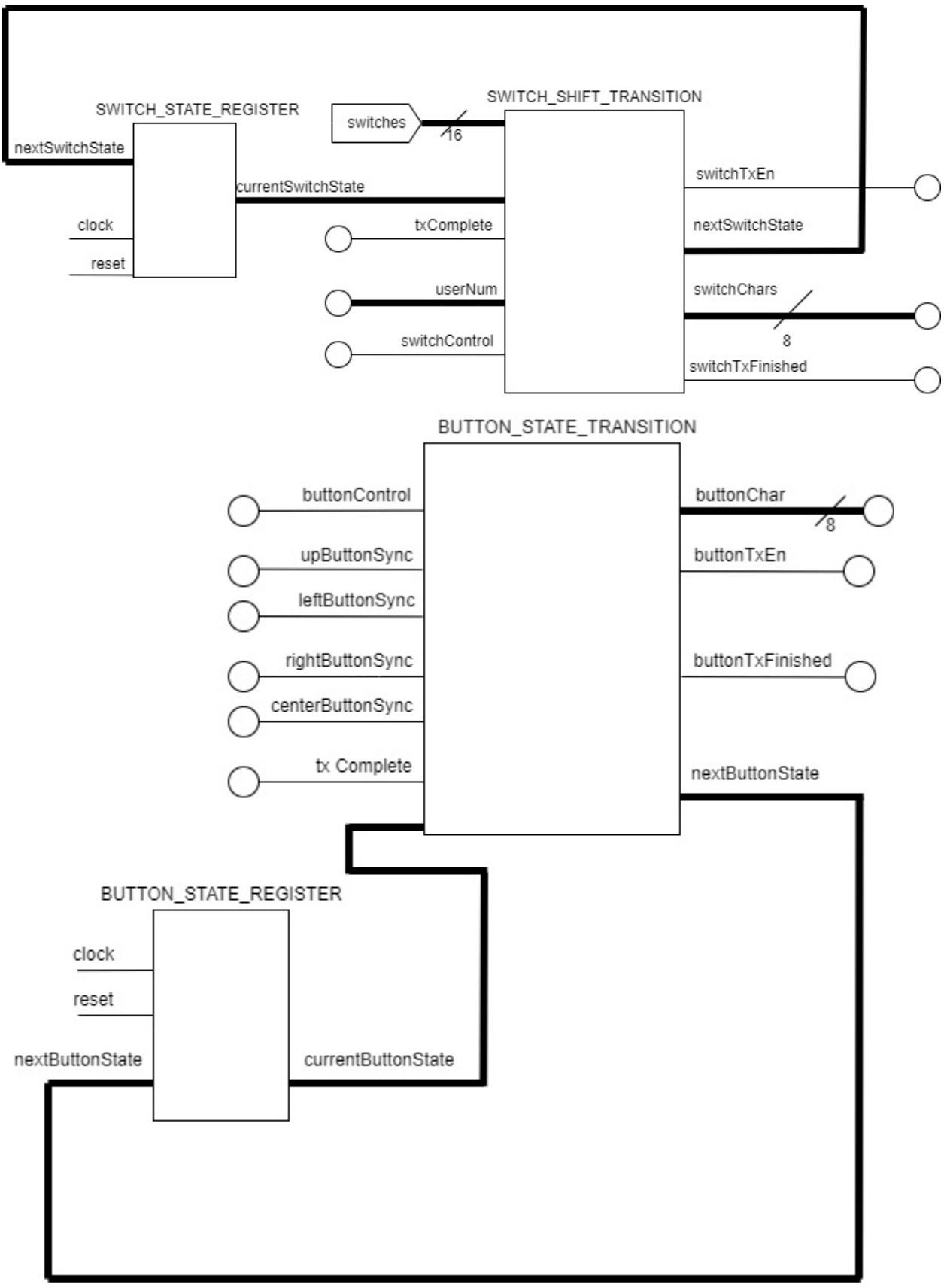

UP_BOTTON_SYNC
upButton
asyncIn
syncOut
upButtonSync
clock
clock
reset
reset
SynchronousChain

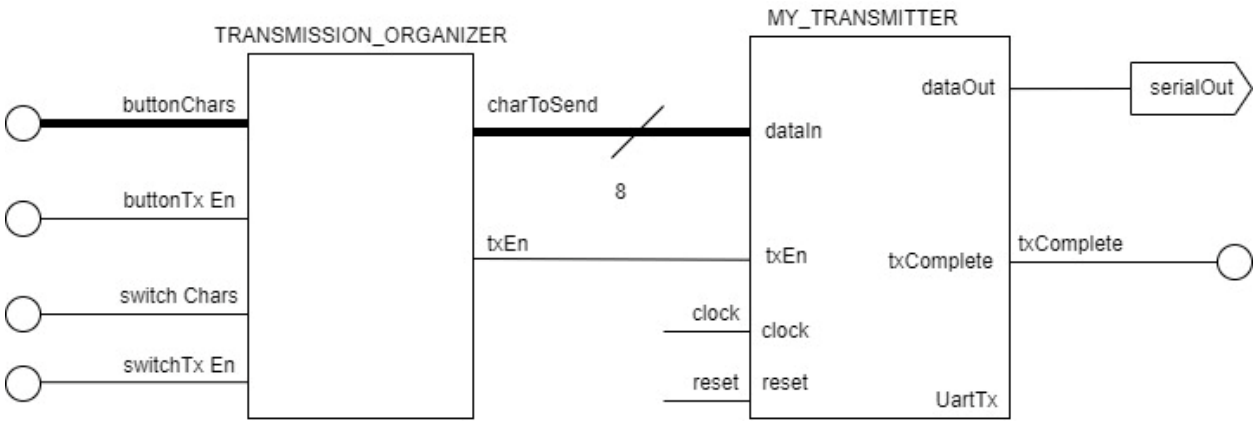

LEFT_BUTTON_SYNC
leftButton
asyncIn
syncOut
leftButtonSync
clock
clock
reset
reset
SynchronousChain


RIGHT_BUTTON_SYNC
rightButton
asyncIn
syncOut
rightButtonSync
clock
clock
reset
reset
SynchronousChain

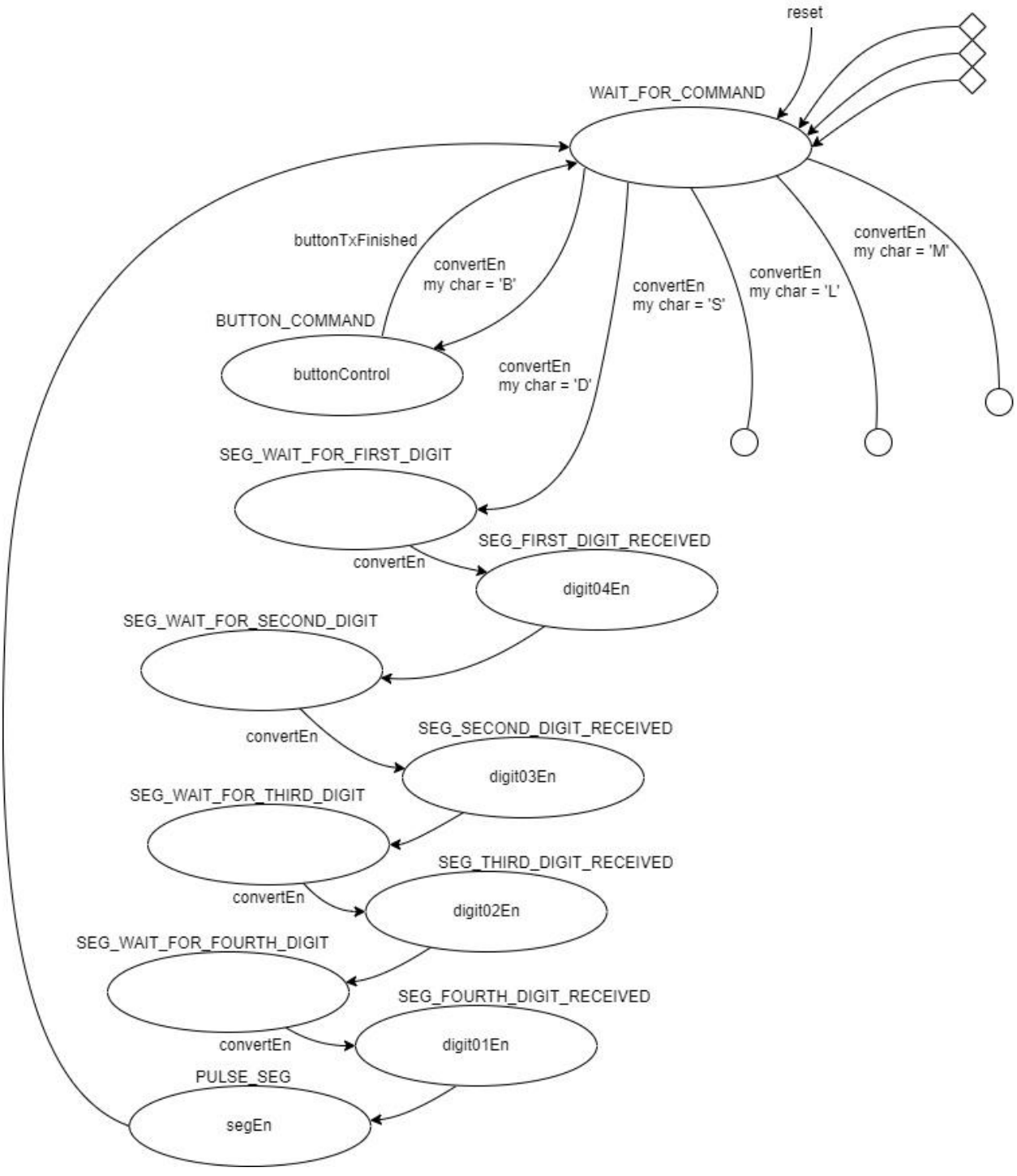

CENTER_BUTTON_SYNC
centerButton
asyncIn
syncOut
centerButtonSync
clock
clock
reset
reset
SynchronousChain

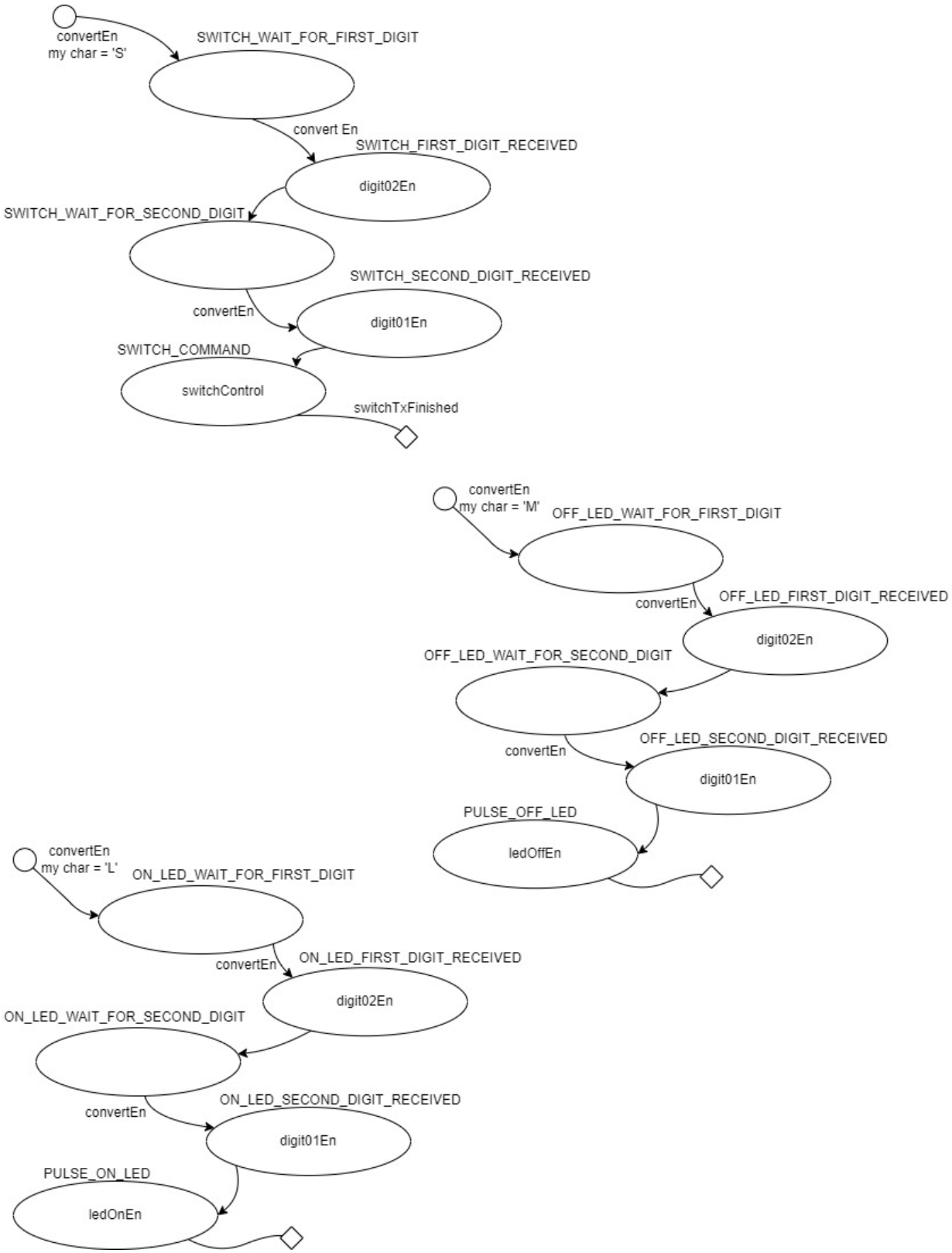

SEGEMENT_DRIVER
SEGMENT_REGISTER
segEn
digit 1
digit 2
digit 3
digit 4
clock
reset
segDigit3
segDigit2
segDigit1
segDigit0
4
4
4
4
digit3
digit2
digit1
digit0
blank3
blank2
blank1
blank0
clock
clock
reset
reset
SevenSegmentDriver
sevenSegs
anodes
segments
anodes


SWITCH_STATE_REGISTER
nextSwitchState
clock
reset
currentSwitchState
SWITCH_SHIFT_TRANSITION
switches
16
txComplete
userNum
switchControl
switchTxEn
nextSwitchState
switchChars
8
switchTxFinished
BUTTON_STATE_TRANSITION
buttonControl
upButtonSync
leftButtonSync
rightButtonSync
centerButtonSync
tx Complete
buttonChar
8
buttonTxEn
buttonTxFinished
nextButtonState
BUTTON_STATE_REGISTER
clock
reset
nextButtonState
currentButtonState

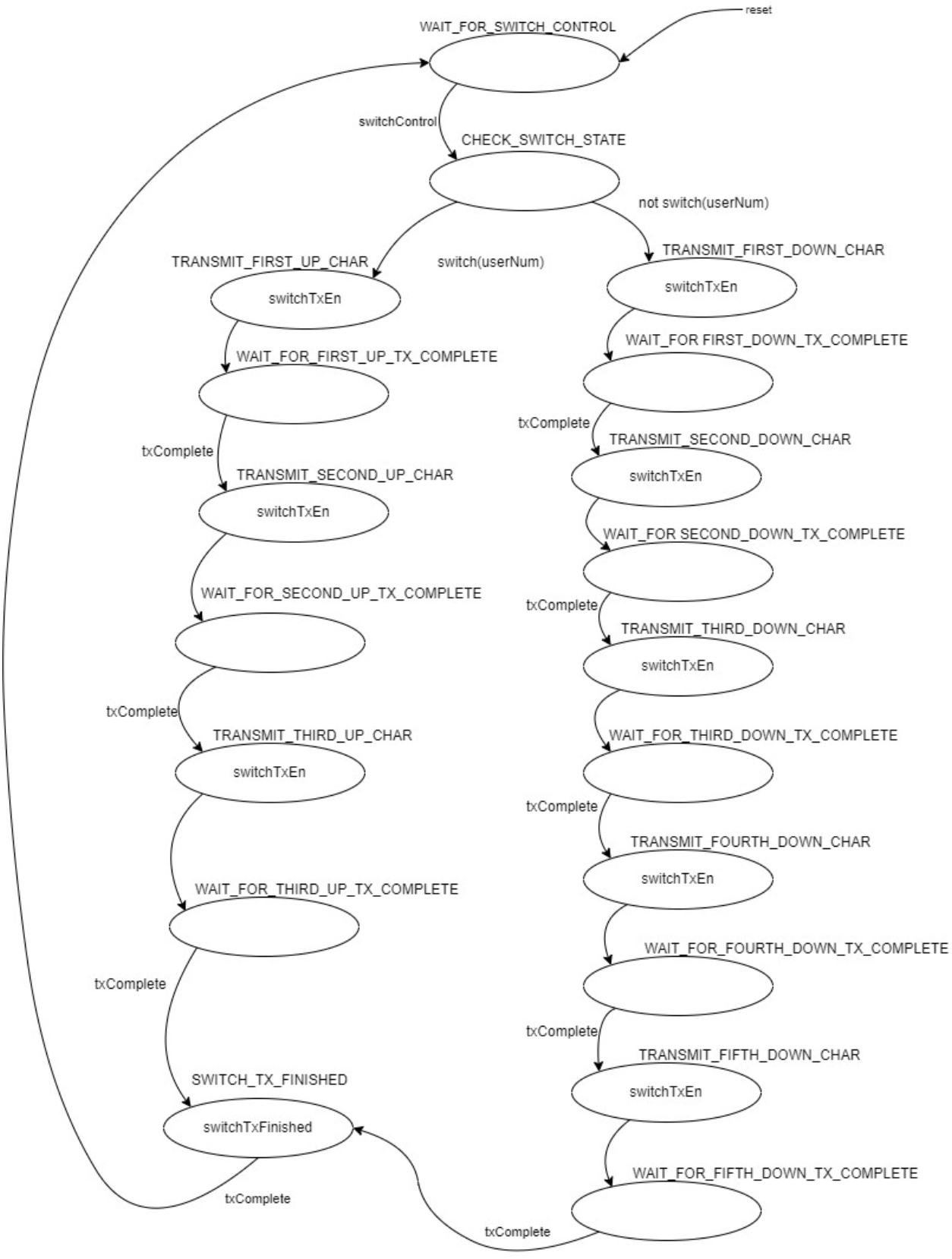

TRANSMISSION_ORGANIZER
MY_TRANSMITTER
buttonChars
buttonTx En
switch Chars
switchTx En
charToSend
8
txEn
clock
reset
dataIn
txEn
clock
reset
dataOut
txComplete
UartTx
serialOut
txComplete

Ascii State Diagram:

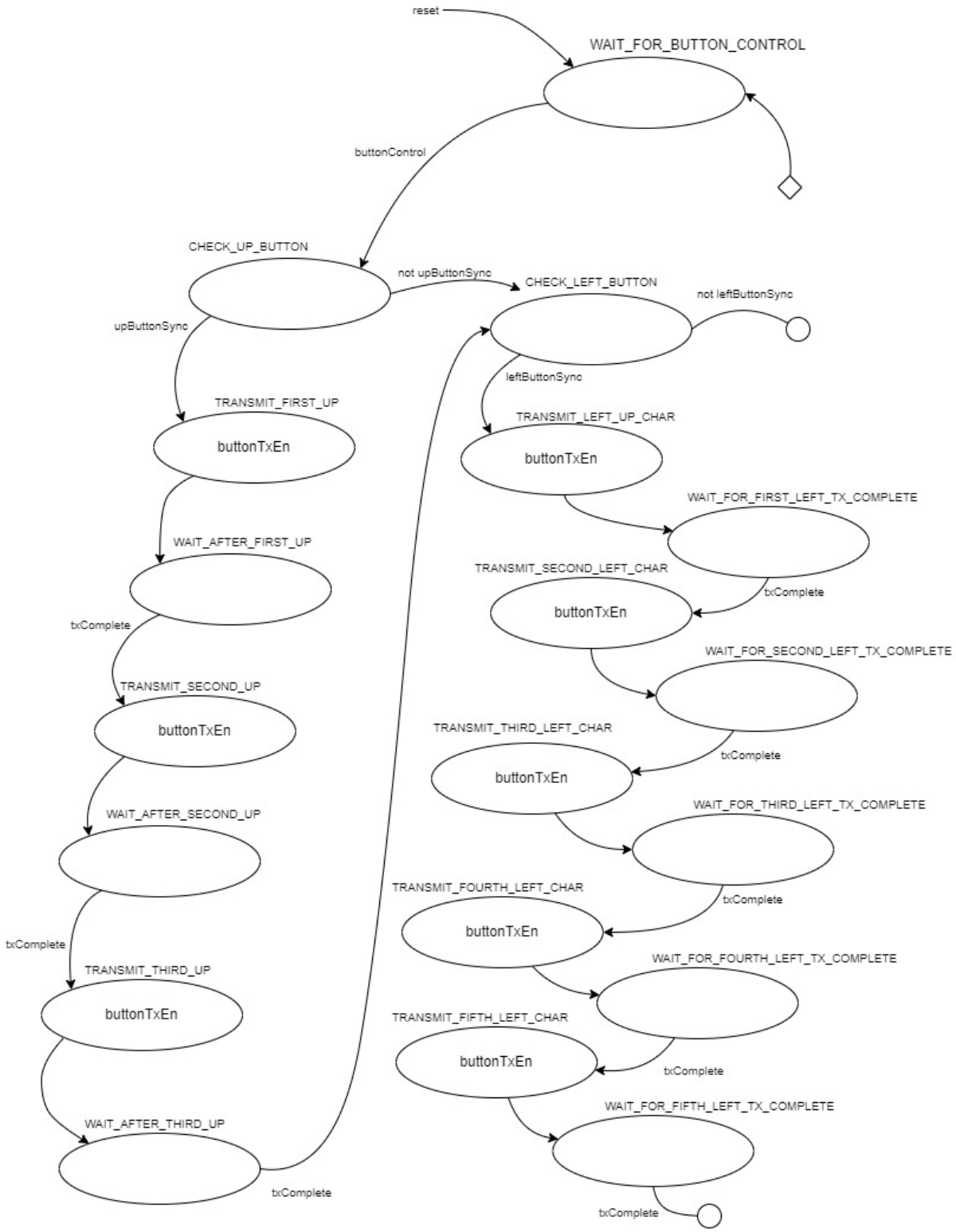


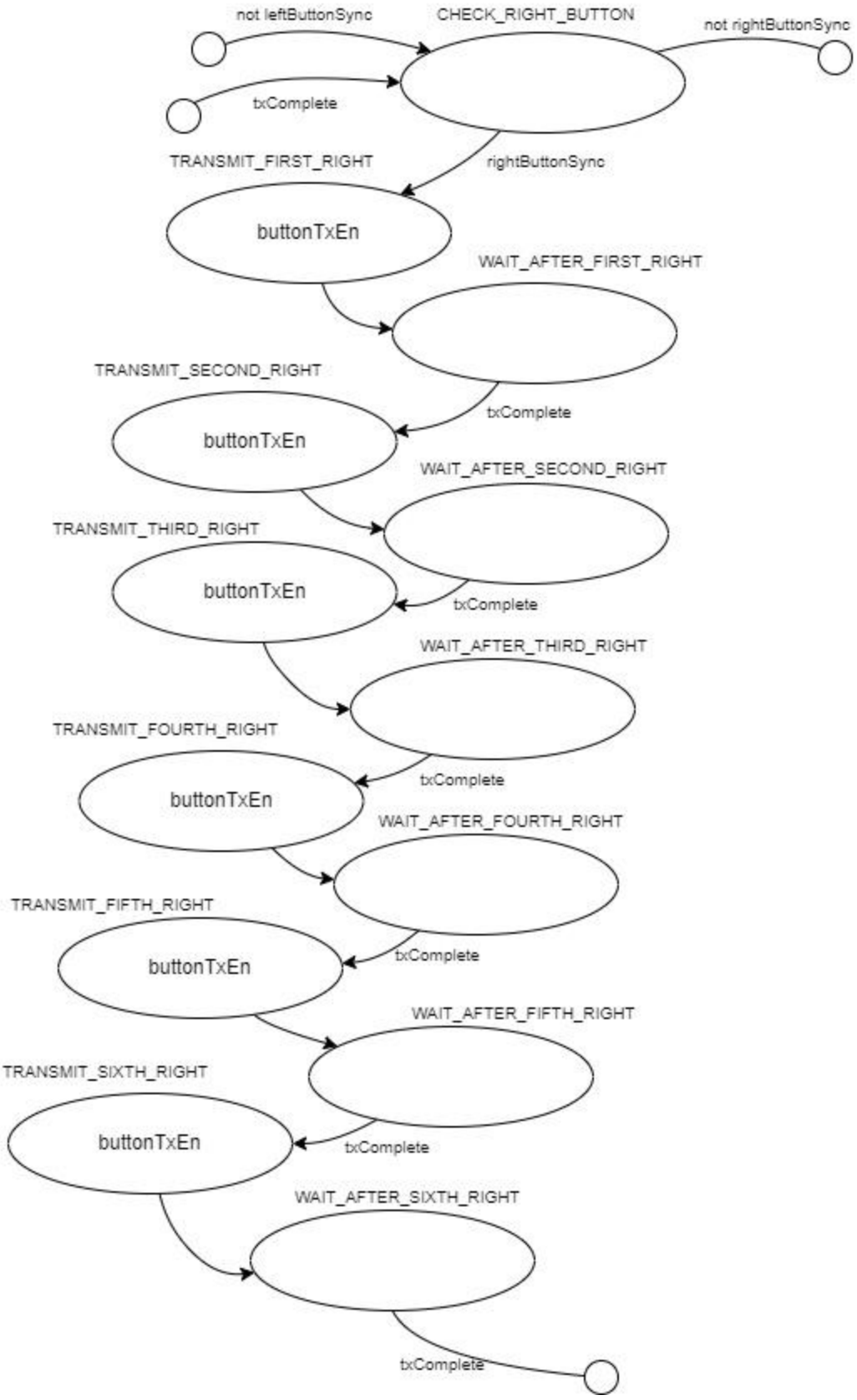

convertEn
my char = 'S'
SWITCH_WAIT_FOR_FIRST_DIGIT
convert En
SWITCH_FIRST_DIGIT_RECEIVED
digit02En
SWITCH_WAIT_FOR_SECOND_DIGIT
SWITCH_SECOND_DIGIT_RECEIVED
convertEn
digit01En
SWITCH_COMMAND
switchControl
switchTxFinished
convertEn
my char = 'M'
OFF_LED_WAIT_FOR_FIRST_DIGIT
convertEn
OFF_LED_FIRST_DIGIT_RECEIVED
digit02En
OFF_LED_WAIT_FOR_SECOND_DIGIT
convertEn
OFF_LED_SECOND_DIGIT_RECEIVED
digit01En
PULSE_OFF_LED
ledOffEn
convertEn
my char = 'L'
ON_LED_WAIT_FOR_FIRST_DIGIT
convertEn
ON_LED_FIRST_DIGIT_RECEIVED
digit02En
ON_LED_WAIT_FOR_SECOND_DIGIT
convertEn
ON_LED_SECOND_DIGIT_RECEIVED
digit01En
PULSE_ON_LED
ledOnEn

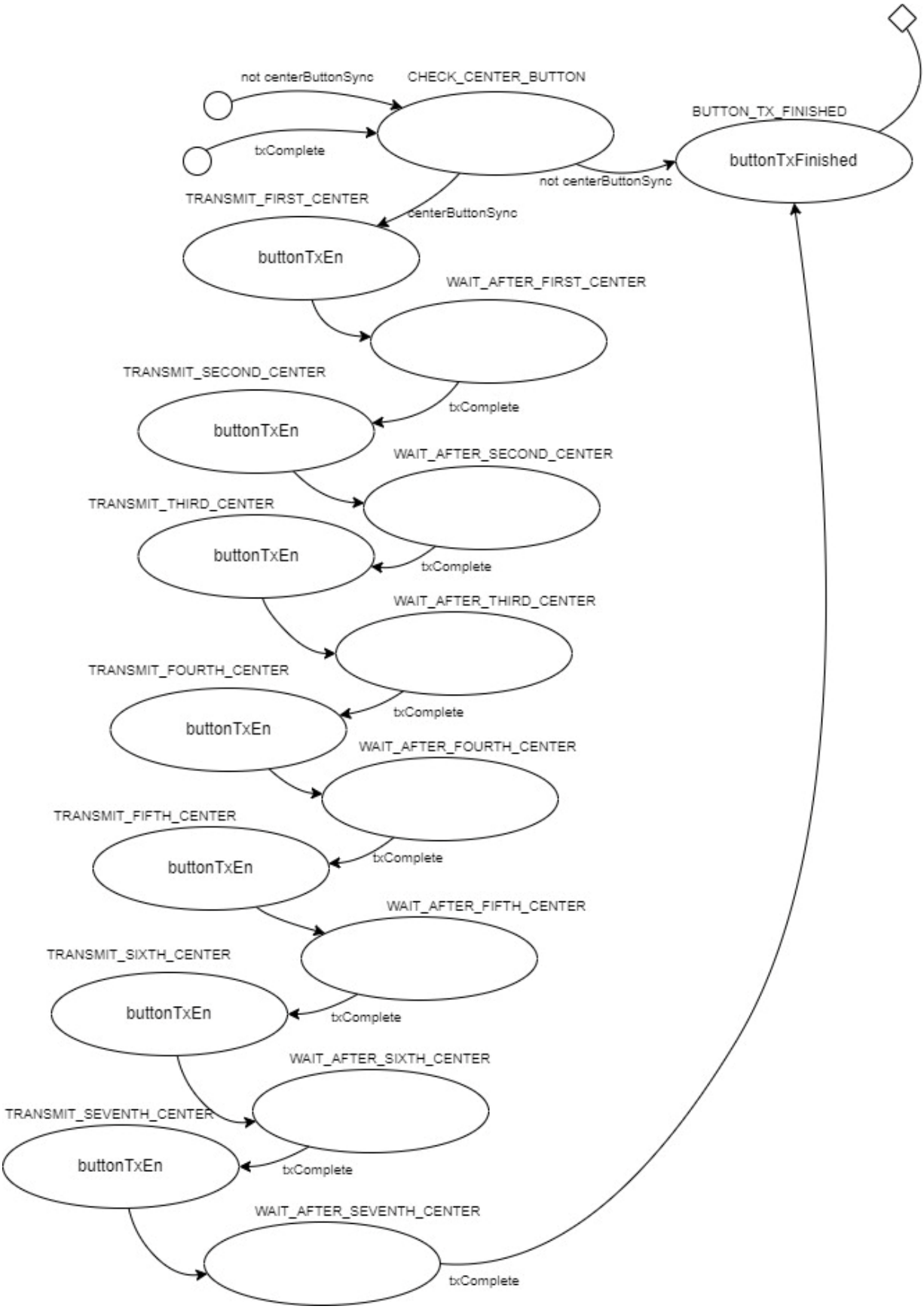
Switch State Diagram:



Button State Machine:




not leftButtonSync
CHECK_RIGHT_BUTTON
not rightButtonSync
txComplete
TRANSMIT_FIRST_RIGHT
rightButtonSync
buttonTxEn
WAIT_AFTER_FIRST_RIGHT
TRANSMIT_SECOND_RIGHT
txComplete
buttonTxEn
WAIT_AFTER_SECOND_RIGHT
TRANSMIT_THIRD_RIGHT
buttonTxEn
txComplete
WAIT_AFTER_THIRD_RIGHT
TRANSMIT_FOURTH_RIGHT
txComplete
buttonTxEn
WAIT_AFTER_FOURTH_RIGHT
TRANSMIT_FIFTH_RIGHT
buttonTxEn
txComplete
WAIT_AFTER_FIFTH_RIGHT
TRANSMIT_SIXTH_RIGHT
buttonTxEn
txComplete
WAIT_AFTER_SIXTH_RIGHT
txComplete


not centerButtonSync
CHECK_CENTER_BUTTON
txComplete
BUTTON_TX_FINISHED
buttonTxFinished
not centerButtonSync
TRANSMIT_FIRST_CENTER
centerButtonSync
buttonTxEn
WAIT_AFTER_FIRST_CENTER
TRANSMIT_SECOND_CENTER
txComplete
buttonTxEn
WAIT_AFTER_SECOND_CENTER
TRANSMIT_THIRD_CENTER
buttonTxEn
txComplete
WAIT_AFTER_THIRD_CENTER
TRANSMIT_FOURTH_CENTER
txComplete
buttonTxEn
WAIT_AFTER_FOURTH_CENTER
TRANSMIT_FIFTH_CENTER
buttonTxEn
txComplete
WAIT_AFTER_FIFTH_CENTER
TRANSMIT_SIXTH_CENTER
buttonTxEn
txComplete
WAIT_AFTER_SIXTH_CENTER
TRANSMIT_SEVENTH_CENTER
buttonTxEn
txComplete
WAIT_AFTER_SEVENTH_CENTER
txComplete

SerialControl Component Code:

```
----------------------------------------------------------------------------
-- SerialControl
-- Sofiyan Monga
--
--      SerialControl takes in a serial input and does various tasks with it.
--      These tasks include turning any LED on or off, checking the position
of any
--      switch, checking which buttons are currently pressed, and outputting
a
--      number of the user's choice on the seven-segment display.
--
--      These commands are invoked by certain characters being detected by
the
--      reciever component, which are L, M, S, B, and D respectively.
--
--      A transmitter component is also used to return the position of any
switch
--      and for which buttons are pressed.
--
----------------------------------------------------------------------------


library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity SerialControl is
    Port ( clock           : in std_logic;
           rawReset        : in std_logic;
           serialIn        : in std_logic;
           upButton        : in std_logic;
           leftButton      : in std_logic;
           rightButton     : in std_logic;
           centerButton    : in std_logic;
           switches        : in std_logic_vector (15 downto 0);
           leds            : out std_logic_vector (15 downto 0);
           segments        : out std_logic_vector (6 downto 0);
           anodes          : out std_logic_vector (3 downto 0);
           serialOut       : out std_logic);
end SerialControl;

architecture SerialControl_ARCH of SerialControl is
    --constants-----------------------------------------------------------
    constant ACTIVE    : std_logic := '1';
    constant CLOCK_FREQ: integer := 100_000_000;
    constant BAUD_RATE : integer := 115_200;

    constant ZERO_BINARY : std_logic_vector(3 downto 0) := "0000";
    constant ONE_BINARY  : std_logic_vector(3 downto 0) := "0001";
    constant TWO_BINARY  : std_logic_vector(3 downto 0) := "0010";
    constant THREE_BINARY: std_logic_vector(3 downto 0) := "0011";
    constant FOUR_BINARY : std_logic_vector(3 downto 0) := "0100";
    constant FIVE_BINARY : std_logic_vector(3 downto 0) := "0101";
    constant SIX_BINARY  : std_logic_vector(3 downto 0) := "0110";
    constant SEVEN_BINARY: std_logic_vector(3 downto 0) := "0111";
```

```
    constant EIGHT_BINARY: std_logic_vector(3 downto 0) := "1000";
    constant NINE_BINARY : std_logic_vector(3 downto 0) := "1001";

    constant L_ASCII              : std_logic_vector(7 downto 0) :=
"01001100";
    constant M_ASCII              : std_logic_vector(7 downto 0) :=
"01001101";
    constant S_ASCII              : std_logic_vector(7 downto 0) :=
"01010011";
    constant B_ASCII              : std_logic_vector(7 downto 0) :=
"01000010";
    constant D_ASCII              : std_logic_vector(7 downto 0) :=
"01000100";
    constant COMMA_ASCII          : std_logic_vector(7 downto 0) :=
"00101100";

    --arrays for the states below (std_logic_vectors are ASCII---------------
-------------------
    -- Up_a := ('u', 'p', '\n')
    type Up_a is array (0 to 2) of std_logic_vector(7 downto 0);
    signal Up: Up_a := ("01110101", "01110000", COMMA_ASCII);
    -- Down_a := ('d', 'o', 'w', 'n', '\n')
    type Down_a is array (0 to 4) of std_logic_vector(7 downto 0);
    signal Down: Down_a := ("01100100", "01101111", "01110111", "01101110",
COMMA_ASCII);
    -- Right_a := ('r', 'i', 'g', 'h', 't', '\n')
    type Right_a is array (0 to 5) of std_logic_vector(7 downto 0);
    signal Right: Right_a := ("01110010", "01101001", "01100111", "01101000",
"01110100", COMMA_ASCII);
    -- Left_a := ('l', 'e', 'f', 't', '\n')
    type Left_a is array (0 to 4) of std_logic_vector(7 downto 0);
    signal Left: Left_a := ("01101100", "01100101", "01100110", "01110100",
COMMA_ASCII);
    -- Center)a := ('c', 'e', 'n', 't', 'e', 'r', '\n')
    type Center_a is array (0 to 6) of std_logic_vector(7 downto 0);
    signal Center: Center_a := ("01100011", "01100101", "01101110",
"01110100", "01100101", "01110010", COMMA_ASCII);

    --ascii state declaration-------------------------------------------------
--
    type Ascii_t is (WAIT_FOR_COMMAND, BUTTON_COMMAND,
SEG_WAIT_FOR_FIRST_DIGIT, SEG_FIRST_DIGIT_RECEIVED,
                     SEG_WAIT_FOR_SECOND_DIGIT, SEG_SECOND_DIGIT_RECEIVED,
SEG_WAIT_FOR_THIRD_DIGIT,
                     SEG_THIRD_DIGIT_RECEIVED, SEG_WAIT_FOR_FOURTH_DIGIT,
SEG_FOURTH_DIGIT_RECEIVED,
                     SWITCH_WAIT_FOR_FIRST_DIGIT,
SWITCH_FIRST_DIGIT_RECEIVED, SWITCH_WAIT_FOR_SECOND_DIGIT,
                     SWITCH_SECOND_DIGIT_RECEIVED,
OFF_LED_WAIT_FOR_FIRST_DIGIT, OFF_LED_FIRST_DIGIT_RECEIVED,
                     OFF_LED_WAIT_FOR_SECOND_DIGIT,
OFF_LED_SECOND_DIGIT_RECEIVED, ON_LED_WAIT_FOR_FIRST_DIGIT,
                     ON_LED_FIRST_DIGIT_RECEIVED,
ON_LED_WAIT_FOR_SECOND_DIGIT, ON_LED_SECOND_DIGIT_RECEIVED, PULSE_SEG,
                     PULSE_ON_LED, PULSE_OFF_LED, SWITCH_COMMAND);
    signal currentAsciiState: Ascii_t;
    signal nextAsciiState   : Ascii_t;
```

```
    --switch state declaration----------------------------------------
    type Switch_t is (WAIT_FOR_SWITCH_CONTROL, CHECK_SWITCH_STATE,
TRANSMIT_FIRST_UP_CHAR, WAIT_FOR_FIRST_UP_TX_COMPLETE,
                      TRANSMIT_SECOND_UP_CHAR,
WAIT_FOR_SECOND_UP_TX_COMPLETE, TRANSMIT_THIRD_UP_CHAR,
                      WAIT_FOR_THIRD_UP_TX_COMPLETE,
TRANSMIT_FIRST_DOWN_CHAR, WAIT_FOR_FIRST_DOWN_TX_COMPLETE,
                      TRANSMIT_SECOND_DOWN_CHAR,
WAIT_FOR_SECOND_DOWN_TX_COMPLETE, TRANSMIT_THIRD_DOWN_CHAR,
                      WAIT_FOR_THIRD_DOWN_TX_COMPLETE,
TRANSMIT_FOURTH_DOWN_CHAR, WAIT_FOR_FOURTH_DOWN_TX_COMPLETE,
                      TRANSMIT_FIFTH_DOWN_CHAR,
WAIT_FOR_FIFTH_DOWN_TX_COMPLETE, SWITCH_TX_FINISHED);
    signal currentSwitchState: Switch_t;
    signal nextSwitchState   : Switch_t;

    --button state declaration----------------------------------------
    type Button_t is (WAIT_FOR_BUTTON_CONTROL, CHECK_UP_BUTTON,
TRANSMIT_FIRST_UP, WAIT_AFTER_FIRST_UP, TRANSMIT_SECOND_UP,
                      WAIT_AFTER_SECOND_UP, TRANSMIT_THIRD_UP,
WAIT_AFTER_THIRD_UP, CHECK_LEFT_BUTTON, TRANSMIT_FIRST_LEFT,
                      WAIT_AFTER_FIRST_LEFT, TRANSMIT_SECOND_LEFT,
WAIT_AFTER_SECOND_LEFT, TRANSMIT_THIRD_LEFT,
                      WAIT_AFTER_THIRD_LEFT, TRANSMIT_FOURTH_LEFT,
WAIT_AFTER_FOURTH_LEFT, TRANSMIT_FIFTH_LEFT,
                      WAIT_AFTER_FIFTH_LEFT, CHECK_RIGHT_BUTTON,
TRANSMIT_FIRST_RIGHT, WAIT_AFTER_FIRST_RIGHT,
                      TRANSMIT_SECOND_RIGHT, WAIT_AFTER_SECOND_RIGHT,
TRANSMIT_THIRD_RIGHT, WAIT_AFTER_THIRD_RIGHT,
                      TRANSMIT_FOURTH_RIGHT, WAIT_AFTER_FOURTH_RIGHT,
TRANSMIT_FIFTH_RIGHT, WAIT_AFTER_FIFTH_RIGHT,
                      TRANSMIT_SIXTH_RIGHT, WAIT_AFTER_SIXTH_RIGHT,
CHECK_CENTER_BUTTON, TRANSMIT_FIRST_CENTER,
                      WAIT_AFTER_FIRST_CENTER, TRANSMIT_SECOND_CENTER,
WAIT_AFTER_SECOND_CENTER, TRANSMIT_THIRD_CENTER,
                      WAIT_AFTER_THIRD_CENTER, TRANSMIT_FOURTH_CENTER,
WAIT_AFTER_FOURTH_CENTER, TRANSMIT_FIFTH_CENTER,
                      WAIT_AFTER_FIFTH_CENTER, TRANSMIT_SIXTH_CENTER,
WAIT_AFTER_SIXTH_CENTER, TRANSMIT_SEVENTH_CENTER,
                      WAIT_AFTER_SEVENTH_CENTER, BUTTON_TX_FINISHED);
    signal currentButtonstate: Button_t;
    signal nextButtonState   : Button_t;

    --SevenSegmentDriver component-----------------------------------
    component SevenSegmentDriver
        port(
            reset: in std_logic;
            clock: in std_logic;

            digit3: in std_logic_vector(3 downto 0);    --leftmost digit
            digit2: in std_logic_vector(3 downto 0);    --2nd from left digit
            digit1: in std_logic_vector(3 downto 0);    --3rd from left digit
            digit0: in std_logic_vector(3 downto 0);    --rightmost digit

            blank3: in std_logic;    --leftmost digit
            blank2: in std_logic;    --2nd from left digit
```

```
            blank1: in std_logic;    --3rd from left digit
            blank0: in std_logic;    --rightmost digit

            sevenSegs: out std_logic_vector(6 downto 0);    --MSB=g, LSB=a
            anodes:    out std_logic_vector(3 downto 0)    --MSB=leftmost
digit
    );
    end component;

    --SynchronizerChain component----------------------------------------
    component SynchronizerChain is
        generic (CHAIN_SIZE: positive);
        port (
            reset:    in  std_logic;
            clock:    in  std_logic;
            asyncIn:  in  std_logic;
            syncOut:  out std_logic
    );
    end component;

    --UartRx component---------------------------------------------------
    component UartRx is
        generic (
            BAUD_RATE:  positive;
            CLOCK_FREQ: positive
            );
        port (
            clock: in std_logic;
            reset: in std_logic;
            rxData: in std_logic;
            dataReady: out std_logic;
            dataOut: out std_logic_vector(7 downto 0)
            );
    end component;

    --UartTx component---------------------------------------------------
    component UartTx is
        generic(
            BAUD_RATE:  positive;
            CLOCK_FREQ: positive
            );
        port(
            clock:       in   std_logic;
            reset:       in   std_logic;
            txEn:        in   std_logic;
            dataIn:      in   std_logic_vector(7 downto 0);
            txComplete:  out  std_logic;
            dataOut:     out  std_logic
            );
    end component;

    --other internal signals---------------------------------------------
    signal convertEn       : std_logic;
    signal buttonTxFinished: std_logic;
    signal switchTxFinished: std_logic;
    signal ledOffEn        : std_logic;
    signal ledOnEn         : std_logic;
```

```
    signal buttonControl   : std_logic;
    signal segEn           : std_logic;
    signal switchControl   : std_logic;
    signal digit01En       : std_logic;
    signal digit02En       : std_logic;
    signal digit03En       : std_logic;
    signal digit04En       : std_logic;
    signal txEn            : std_logic;
    signal centerButtonSync: std_logic;
    signal rightButtonSync : std_logic;
    signal leftButtonSync  : std_logic;
    signal upButtonSync    : std_logic;
    signal buttonTxEn      : std_logic;
    signal txComplete      : std_logic;
    signal switchTxEn      : std_logic;
    signal reset           : std_logic;
    signal userNum         : integer range 0 to 16;
    signal digit1          : std_logic_vector (3 downto 0);
    signal digit2          : std_logic_vector (3 downto 0);
    signal digit3          : std_logic_vector (3 downto 0);
    signal digit4          : std_logic_vector (3 downto 0);
    signal segDigit0       : std_logic_vector (3 downto 0);
    signal segDigit1       : std_logic_vector (3 downto 0);
    signal segDigit2       : std_logic_vector (3 downto 0);
    signal segDigit3       : std_logic_vector (3 downto 0);
    signal switchChar      : std_logic_vector (7 downto 0);
    signal buttonChar      : std_logic_vector (7 downto 0);
    signal charToSend      : std_logic_vector (7 downto 0);
    signal myChar          : std_logic_vector (7 downto 0);

begin
    --port map for SevenSegmentDriver-------------------------------------
    SEGMENT_DRIVER: SevenSegmentDriver
        port map (
            clock     => clock,
            reset     => reset,
            digit3    => segDigit3,
            digit2    => segDigit2,
            digit1    => segDigit1,
            digit0    => segDigit0,
            blank3    => not ACTIVE,
            blank2    => not ACTIVE,
            blank1    => not ACTIVE,
            blank0    => not ACTIVE,
            sevenSegs => segments,
            anodes    => anodes
        );

    --port map for SynchronizerChain-------------------------------------
    UP_BUTTON_SYNC: SynchronizerChain
        generic map (
            CHAIN_SIZE => 2
        )
        port map (
            reset   => reset,
            clock   => clock,
            asyncIn => upButton,
```

```
        syncOut => upButtonSync
    );

--port map for SynchronizerChain-----------------------------------
LEFT_BUTTON_SYNC: SynchronizerChain
    generic map (
        CHAIN_SIZE => 2
    )
    port map (
        reset   => reset,
        clock   => clock,
        asyncIn => leftButton,
        syncOut => leftButtonSync
    );

--port map for SynchronizerChain-----------------------------------
RIGHT_BUTTON_SYNC: SynchronizerChain
    generic map (
        CHAIN_SIZE => 2
    )
    port map (
        reset   => reset,
        clock   => clock,
        asyncIn => rightButton,
        syncOut => rightButtonSync
    );

--port map for SynchronizerChain-----------------------------------
CENTER_BUTTON_SYNC: SynchronizerChain
    generic map (
        CHAIN_SIZE => 2
    )
    port map (
        reset   => reset,
        clock   => clock,
        asyncIn => centerButton,
        syncOut => centerButtonSync
    );

--port map for UartRx-----------------------------------------------
MY_RECEIVER: UartRx
    generic map (
        BAUD_RATE => BAUD_RATE,
        CLOCK_FREQ => CLOCK_FREQ
    )
    port map (
        reset     => reset,
        clock     => clock,
        rxData    => serialIn,
        dataOut   => myChar,
        dataReady => convertEn
    );

--port map for UartTx-----------------------------------------------
MY_TRANSMITTER: UartTx
    generic map (
        BAUD_RATE => BAUD_RATE,
```

```
            CLOCK_FREQ => CLOCK_FREQ
        )
        port map (
            reset      => reset,
            clock      => clock,
            txEn       => txEn,
            dataIn     => charToSend,
            dataOut    => serialOut,
            txComplete => txComplete
        );

    --synchronize reset signal to end on a clock edge-----------------------
---
    RESET_SYNCHRONIZER: process (rawReset, clock)
        variable chain: std_logic_vector(1 downto 0);
    begin
        if (rawReset = ACTIVE) then
            chain := (others => ACTIVE);
        elsif (rising_edge(clock)) then
            chain := chain(0) & not ACTIVE;
        end if;
        reset <= chain(1);
    end process RESET_SYNCHRONIZER;

    --state register for ASCII_TRANSITION-----------------------------------
---
    ASCII_REGISTER: process (reset, clock)
    begin
        if (reset = ACTIVE) then
            currentAsciiState <= WAIT_FOR_COMMAND;
        elsif (rising_edge(clock)) then
            currentAsciiState <= nextAsciiState;
        end if;
    end process ASCII_REGISTER;

    --state transition block for the Ascii State Machine--------------------
-------------
    ASCII_TRANSITION: process (myChar, buttonTxFinished, switchTxFinished,
convertEn, currentAsciiState)
    begin
        --default values for outputs---------------------------------------
-
        ledOffEn      <= not ACTIVE;
        ledOnEn       <= not ACTIVE;
        buttonControl <= not ACTIVE;
        segEn         <= not ACTIVE;
        switchControl <= not ACTIVE;
        digit01En     <= not ACTIVE;
        digit02En     <= not ACTIVE;
        digit03En     <= not ACTIVE;
        digit04En     <= not ACTIVE;

        case currentAsciiState is

            --WAIT_FOR_COMMAND state---------------------------------------
-
            when WAIT_FOR_COMMAND =>
```

```
        ledOffEn      <= not ACTIVE;
        ledOnEn       <= not ACTIVE;
        buttonControl <= not ACTIVE;
        segEn         <= not ACTIVE;
        switchControl <= not ACTIVE;
        digit01En     <= not ACTIVE;
        digit02En     <= not ACTIVE;
        digit03En     <= not ACTIVE;
        digit04En     <= not ACTIVE;

        if (convertEn = ACTIVE) then
            if (myChar = B_ASCII) then
                nextAsciiState <= BUTTON_COMMAND;
            elsif (myChar = D_ASCII) then
                nextAsciiState <= SEG_WAIT_FOR_FIRST_DIGIT;
            elsif (myChar = S_ASCII) then
                nextAsciiState <= SWITCH_WAIT_FOR_FIRST_DIGIT;
            elsif (myChar = M_ASCII) then
                nextAsciiState <= OFF_LED_WAIT_FOR_FIRST_DIGIT;
            elsif (myChar = L_ASCII) then
                nextAsciiState <= ON_LED_WAIT_FOR_FIRST_DIGIT;
            end if;
        else
            nextAsciiState <= WAIT_FOR_COMMAND;
        end if;

    --BUTTON_CONTROL state---------------------------------------
    when BUTTON_COMMAND =>
        buttonControl <= ACTIVE;

        if (buttonTxFinished = ACTIVE) then
            nextAsciiState <= WAIT_FOR_COMMAND;
        else
            nextAsciiState <= BUTTON_COMMAND;
        end if;


    --Segment section of the Ascii State Machine-----------------
    --SEG_WAIT_FOR_FIRST_DIGIT-----------------------------------
    when SEG_WAIT_FOR_FIRST_DIGIT =>
        if (convertEn = ACTIVE) then
            nextAsciiState <= SEG_FIRST_DIGIT_RECEIVED;
        else
            nextAsciiState <= SEG_WAIT_FOR_FIRST_DIGIT;
        end if;

    --SEG_FIRST_DIGIT_RECEIVED-----------------------------------
    when SEG_FIRST_DIGIT_RECEIVED =>
        digit04En <= ACTIVE;
        nextAsciiState <= SEG_WAIT_FOR_SECOND_DIGIT;

    --SEG_WAIT_FOR_SECOND_DIGIT----------------------------------
    when SEG_WAIT_FOR_SECOND_DIGIT =>
        if (convertEn = ACTIVE) then
            nextAsciiState <= SEG_SECOND_DIGIT_RECEIVED;
        else
            nextAsciiState <= SEG_WAIT_FOR_SECOND_DIGIT;
```

```
    end if;

--SEG_SECOND_DIGIT_RECEIVED---------------------------------
when SEG_SECOND_DIGIT_RECEIVED =>
    digit03En <= ACTIVE;
    nextAsciiState <= SEG_WAIT_FOR_THIRD_DIGIT;

--SEG_WAIT_FOR_THIRD_DIGIT-----------------------------------
when SEG_WAIT_FOR_THIRD_DIGIT =>
    if (convertEn = ACTIVE) then
        nextAsciiState <= SEG_THIRD_DIGIT_RECEIVED;
    else
        nextAsciiState <= SEG_WAIT_FOR_THIRD_DIGIT;
    end if;

--SEG_THIRD_DIGIT_RECEIVED-----------------------------------
when SEG_THIRD_DIGIT_RECEIVED =>
    digit02En <= ACTIVE;
    nextAsciiState <= SEG_WAIT_FOR_FOURTH_DIGIT;

--SEG_WAIT_FOR_FOURTH_DIGIT-----------------------------------
when SEG_WAIT_FOR_FOURTH_DIGIT =>
    if (convertEn = ACTIVE) then
        nextAsciiState <= SEG_FOURTH_DIGIT_RECEIVED;
    else
        nextAsciiState <= SEG_WAIT_FOR_FOURTH_DIGIT;
    end if;

--SEG_FOURTH_DIGIT_RECEIVED-----------------------------------
when SEG_FOURTH_DIGIT_RECEIVED =>
    digit01En <= ACTIVE;
    nextAsciiState <= PULSE_SEG;

--PULSE_SEG---------------------------------------------------
when PULSE_SEG =>
    segen <= ACTIVE;
    nextAsciiState <= WAIT_FOR_COMMAND;

--Switch section of the Ascii State Machine-----------------
--SWITCH_WAIT_FOR_FIRST_DIGIT-------------------------------
when SWITCH_WAIT_FOR_FIRST_DIGIT =>
    if (convertEn = ACTIVE) then
        nextAsciiState <= SWITCH_FIRST_DIGIT_RECEIVED;
    else
        nextAsciiState <= SWITCH_WAIT_FOR_FIRST_DIGIT;
    end if;

--SWITCH_FIRST_DIGIT_RECEIVED---------------------------------
when SWITCH_FIRST_DIGIT_RECEIVED =>
    digit02En <= ACTIVE;
    nextAsciiState <= SWITCH_WAIT_FOR_SECOND_DIGIT;

--SWITCH_WAIT_FOR_SECOND_DIGIT-------------------------------
when SWITCH_WAIT_FOR_SECOND_DIGIT =>
    if (convertEn = ACTIVE) then
        nextAsciiState <= SWITCH_SECOND_DIGIT_RECEIVED;
    else
```

```
            nextAsciiState <= SWITCH_WAIT_FOR_SECOND_DIGIT;
        end if;

    --SWITCH_SECOND_DIGIT_RECEIVED-------------------------------
    when SWITCH_SECOND_DIGIT_RECEIVED =>
        digit01En <= ACTIVE;
        nextAsciiState <= SWITCH_COMMAND;

    --SWITCH_COMMAND----------------------------------------
    when SWITCH_COMMAND =>
        switchControl <= ACTIVE;

        if (switchTxFinished = ACTIVE) then
            nextAsciiState <= WAIT_FOR_COMMAND;
        else
            nextAsciiState <= SWITCH_COMMAND;
        end if;


    --LED OFF section of the Ascii State Machine----------------
    --OFF_LED_WAIT_FOR_FIRST_DIGIT----------------------------
    when OFF_LED_WAIT_FOR_FIRST_DIGIT =>
        if (convertEn = ACTIVE) then
            nextAsciiState <= OFF_LED_FIRST_DIGIT_RECEIVED;
        else
            nextAsciiState <= OFF_LED_WAIT_FOR_FIRST_DIGIT;
        end if;

    --OFF_LED_FIRST_DIGIT_RECEIVED--------------------------------
    when OFF_LED_FIRST_DIGIT_RECEIVED =>
        digit02En <= ACTIVE;
        nextAsciiState <= OFF_LED_WAIT_FOR_SECOND_DIGIT;

    --OFF_LED_WAIT_FOR_SECOND_DIGIT----------------------------
    when OFF_LED_WAIT_FOR_SECOND_DIGIT =>
        if (convertEn = ACTIVE) then
            nextAsciiState <= OFF_LED_SECOND_DIGIT_RECEIVED;
        else
            nextAsciiState <= OFF_LED_WAIT_FOR_SECOND_DIGIT;
        end if;

    --OFF_LED_SECOND_DIGIT_RECEIVED-------------------------
    when OFF_LED_SECOND_DIGIT_RECEIVED =>
        digit01En <= ACTIVE;
        nextAsciiState <= PULSE_OFF_LED;

    --PULSE_OFF_LED-----------------------------------------
    when PULSE_OFF_LED =>
        ledOffEn <= ACTIVE;
        nextAsciiState <= WAIT_FOR_COMMAND;


    --LED ON section of the Ascii State Machine--------------
    --ON_LED_WAIT_FOR_FIRST_DIGIT--------------------------
    when ON_LED_WAIT_FOR_FIRST_DIGIT =>
        if (convertEn = ACTIVE) then
            nextAsciiState <= ON_LED_FIRST_DIGIT_RECEIVED;
```

```
                else
                    nextAsciiState <= ON_LED_WAIT_FOR_FIRST_DIGIT;
                end if;

            --ON_LED_FIRST_DIGIT_RECEIVED-------------------------------
            when ON_LED_FIRST_DIGIT_RECEIVED =>
                digit02En <= ACTIVE;
                nextAsciiState <= ON_LED_WAIT_FOR_SECOND_DIGIT;

            --ON_LED_WAIT_FOR_SECOND_DIGIT-----------------------------
            when ON_LED_WAIT_FOR_SECOND_DIGIT =>
                if (convertEn = ACTIVE) then
                    nextAsciiState <= ON_LED_SECOND_DIGIT_RECEIVED;
                else
                    nextAsciiState <= ON_LED_WAIT_FOR_SECOND_DIGIT;
                end if;

            --ON_LED_SECOND_DIGIT_RECEIVED------------------------
            when ON_LED_SECOND_DIGIT_RECEIVED =>
                digit01En <= ACTIVE;
                nextAsciiState <= PULSE_ON_LED;

            --PULSE_ON_LED-----------------------------------
            when PULSE_ON_LED =>
                ledOnEn <= ACTIVE;
                nextAsciiState <= WAIT_FOR_COMMAND;

        end case;
    end process ASCII_TRANSITION;

    --state register for SWITCH_SHIFT_TRANSITION--------------------
    SWITCH_STATE_REGISTER: process (reset, clock)
    begin
        if (reset = ACTIVE) then
            currentSwitchState <= WAIT_FOR_SWITCH_CONTROL;
        elsif (rising_edge(clock)) then
            currentSwitchState <= nextSwitchState;
        end if;
    end process SWITCH_STATE_REGISTER;

    --state transition block for the Switch State Machine-------------------
--------------
    SWITCH_SHIFT_TRANSITION: process (switches, txComplete, userNum,
switchControl, currentSwitchState)
    begin
        --default values for outputs-------------------------------
        switchTxEn <= not ACTIVE;
        switchChar <= (others => not ACTIVE);
        switchTxFinished <= not ACTIVE;

        case currentSwitchState is

            --WAIT_FOR_SWITCH_CONTROL---------------------------------
            when WAIT_FOR_SWITCH_CONTROL =>
                switchTxEn <= not ACTIVE;
                switchChar <= (others => not ACTIVE);
                switchTxFinished <= not ACTIVE;
```

```
    if (switchControl = ACTIVE) then
        nextSwitchState <= CHECK_SWITCH_STATE;
    else
        nextSwitchState <= WAIT_FOR_SWITCH_CONTROL;
    end if;

--CHECK_SWITCH_STATE----------------------------------------
when CHECK_SWITCH_STATE =>
    if(switches(userNum) = ACTIVE) then
        nextSwitchState <= TRANSMIT_FIRST_UP_CHAR;
    else
        nextSwitchState <= TRANSMIT_FIRST_DOWN_CHAR;
    end if;

--TRANSMIT_FIRST_UP_CHAR-----------------------------------
when TRANSMIT_FIRST_UP_CHAR =>
    switchChar <= Up(0);
    switchTxEn <= ACTIVE;
    nextSwitchState <= WAIT_FOR_FIRST_UP_TX_COMPLETE;

--WAIT_FOR_FIRST_UP_TX_COMPLETE----------------------------
when WAIT_FOR_FIRST_UP_TX_COMPLETE =>
    if (txComplete = ACTIVE) then
        nextSwitchState <= TRANSMIT_SECOND_UP_CHAR;
    else
        nextSwitchState <= WAIT_FOR_FIRST_UP_TX_COMPLETE;
    end if;

--TRANSMIT_SECOND_UP_CHAR----------------------------------
when TRANSMIT_SECOND_UP_CHAR =>
    switchChar <= Up(1);
    switchTxEn <= ACTIVE;
    nextSwitchState <= WAIT_FOR_SECOND_UP_TX_COMPLETE;

--WAIT_FOR_SECOND_UP_TX_COMPLETE-----------------------
when WAIT_FOR_SECOND_UP_TX_COMPLETE =>
    if (txComplete = ACTIVE) then
        nextSwitchState <= TRANSMIT_THIRD_UP_CHAR;
    else
        nextSwitchState <= WAIT_FOR_SECOND_UP_TX_COMPLETE;
    end if;

--TRANSMIT_THIRD_UP_CHAR-----------------------------
when TRANSMIT_THIRD_UP_CHAR =>
    switchChar <= Up(2);
    switchTxEn <= ACTIVE;
    nextSwitchState <= WAIT_FOR_THIRD_UP_TX_COMPLETE;

--WAIT_FOR_THIRD_UP_TX_COMPLETE-------------------------
when WAIT_FOR_THIRD_UP_TX_COMPLETE =>
    if (txComplete = ACTIVE) then
        nextSwitchState <= SWITCH_TX_FINISHED;
    else
        nextSwitchState <= WAIT_FOR_THIRD_UP_TX_COMPLETE;
    end if;
```

```
--TRANSMIT_FIRST_DOWN_CHAR----------------------------
when TRANSMIT_FIRST_DOWN_CHAR =>
    switchChar <= Down(0);
    switchTxEn <= ACTIVE;
    nextSwitchState <= WAIT_FOR_FIRST_DOWN_TX_COMPLETE;

--WAIT_FOR_FIRST_DOWN_TX_COMPLETE-----------------------
when WAIT_FOR_FIRST_DOWN_TX_COMPLETE =>
    if (txComplete = ACTIVE) then
        nextSwitchState <= TRANSMIT_SECOND_DOWN_CHAR;
    else
        nextSwitchState <= WAIT_FOR_FIRST_DOWN_TX_COMPLETE;
    end if;

--TRANSMIT_SECOND_DOWN_CHAR----------------------------
when TRANSMIT_SECOND_DOWN_CHAR =>
    switchChar <= Down(1);
    switchTxEn <= ACTIVE;
    nextSwitchState <= WAIT_FOR_SECOND_DOWN_TX_COMPLETE;

--WAIT_FOR_SECOND_DOWN_TX_COMPLETE-----------------------
when WAIT_FOR_SECOND_DOWN_TX_COMPLETE =>
    if (txComplete = ACTIVE) then
        nextSwitchState <= TRANSMIT_THIRD_DOWN_CHAR;
    else
        nextSwitchState <= WAIT_FOR_SECOND_DOWN_TX_COMPLETE;
    end if;

--TRANSMIT_THIRD_DOWN_CHAR-----------------------------------
when TRANSMIT_THIRD_DOWN_CHAR =>
    switchChar <= Down(2);
    switchTxEn <= ACTIVE;
    nextSwitchState <= WAIT_FOR_THIRD_DOWN_TX_COMPLETE;

--WAIT_FOR_THIRD_DOWN_TX_COMPLETE
when WAIT_FOR_THIRD_DOWN_TX_COMPLETE =>
    if (txComplete = ACTIVE) then
        nextSwitchState <= TRANSMIT_FOURTH_DOWN_CHAR;
    else
        nextSwitchState <= WAIT_FOR_THIRD_DOWN_TX_COMPLETE;
    end if;

--TRANSMIT_FOURTH_DOWN_CHAR----------------------------
when TRANSMIT_FOURTH_DOWN_CHAR =>
    switchChar <= Down(3);
    switchTxEn <= ACTIVE;
    nextSwitchState <= WAIT_FOR_FOURTH_DOWN_TX_COMPLETE;

--WAIT_FOR_FOURTH_DOWN_TX_COMPLETE---------------------
when WAIT_FOR_FOURTH_DOWN_TX_COMPLETE =>
    if (txComplete = ACTIVE) then
        nextSwitchState <= TRANSMIT_FIFTH_DOWN_CHAR;
    else
        nextSwitchState <= WAIT_FOR_FOURTH_DOWN_TX_COMPLETE;
    end if;

--TRANSMIT_FIFTH_DOWN_CHAR-----------------------------
```

```
            when TRANSMIT_FIFTH_DOWN_CHAR =>
                switchChar <= Down(4);
                switchTxEn <= ACTIVE;
                nextSwitchState <= WAIT_FOR_FIFTH_DOWN_TX_COMPLETE;

            --WAIT_FOR_FIFTH_DOWN_TX_COMPLETE-----------------------
            when WAIT_FOR_FIFTH_DOWN_TX_COMPLETE =>
                if (txComplete = ACTIVE) then
                    nextSwitchState <= SWITCH_TX_FINISHED;
                else
                    nextSwitchState <= WAIT_FOR_FIFTH_DOWN_TX_COMPLETE;
                end if;

            --SWITCH_TX_FINISHED-------------------------------------
            when SWITCH_TX_FINISHED =>
                switchTxFinished <= ACTIVE;
                nextSwitchState <= WAIT_FOR_SWITCH_CONTROL;
        end case;
    end process SWITCH_SHIFT_TRANSITION;

    --state register for BUTTON_STATE_TRANSITION-----------------------------
    BUTTON_STATE_REGISTER: process (reset, clock)
    begin
        if (reset = ACTIVE) then
            currentButtonState <= WAIT_FOR_BUTTON_CONTROL;
        elsif (rising_edge(clock)) then
            currentButtonState <= nextButtonState;
        end if;
    end process BUTTON_STATE_REGISTER;

    --state transition for the Button State Machine-------------------------
--
    BUTTON_STATE_TRANSITION: process (buttonControl, upButtonSync,
leftButtonSync, rightButtonSync, centerButtonSync, txComplete,
currentButtonState)
    begin
        --default value for the outputs---------------------------------
        buttonTxEn <= not ACTIVE;
        buttonChar <= (others => not ACTIVE);
        buttonTxFinished <= not ACTIVE;

        case currentButtonState is

            --WAIT_FOR_BUTTON_CONTROL-------------------------------------
            when WAIT_FOR_BUTTON_CONTROL =>
                buttonTxEn <= not ACTIVE;
                buttonChar <= (others => not ACTIVE);
                buttonTxFinished <= not ACTIVE;

                if (buttonControl = ACTIVE) then
                    nextButtonState <= CHECK_UP_BUTTON;
                else
                    nextButtonState <= WAIT_FOR_BUTTON_CONTROL;
                end if;

            --CHECK_UP_BUTTON-----------------------------------------
            when CHECK_UP_BUTTON =>
```

```
    if (upButtonSync = ACTIVE) then
        nextButtonState <= TRANSMIT_FIRST_UP;
    else
        nextButtonState <= CHECK_LEFT_BUTTON;
    end if;

--TRANSMIT_FIRST_UP-----------------------------------------
when TRANSMIT_FIRST_UP =>
    buttonChar <= Up(0);
    buttonTxEn <= ACTIVE;
    nextButtonState <= WAIT_AFTER_FIRST_UP;

--WAIT_AFTER_FIRST_UP---------------------------------------
when WAIT_AFTER_FIRST_UP =>
    if (txComplete = ACTIVE) then
        nextButtonState <= TRANSMIT_SECOND_UP;
    else
        nextButtonState <= WAIT_AFTER_FIRST_UP;
    end if;

--TRANSMIT_SECOND_UP----------------------------------------
when TRANSMIT_SECOND_UP =>
    buttonChar <= Up(1);
    buttonTxEn <= ACTIVE;
    nextButtonState <= WAIT_AFTER_SECOND_UP;

--WAIT_AFTER_SECOND_UP--------------------------------------
when WAIT_AFTER_SECOND_UP =>
    if (txComplete = ACTIVE) then
        nextButtonState <= TRANSMIT_THIRD_UP;
    else
        nextButtonState <= WAIT_AFTER_SECOND_UP;
    end if;

--TRANSMIT_THIRD_UP-----------------------------------------
when TRANSMIT_THIRD_UP =>
    buttonChar <= Up(2);
    buttonTxEn <= ACTIVE;
    nextButtonState <= WAIT_AFTER_THIRD_UP;

--WAIT_AFTER_THIRD_UP---------------------------------------
when WAIT_AFTER_THIRD_UP =>
    if (txComplete = ACTIVE) then
        nextButtonState <= CHECK_LEFT_BUTTON;
    else
        nextButtonState <= WAIT_AFTER_THIRD_UP;
    end if;

--CHECK_LEFT_BUTTON-----------------------------------
when CHECK_LEFT_BUTTON =>
    if (leftButtonSync = ACTIVE) then
        nextButtonState <= TRANSMIT_FIRST_LEFT;
    else
        nextButtonState <= CHECK_RIGHT_BUTTON;
    end if;

--TRANSMIT_FIRST_LEFT---------------------------------
```

```
when TRANSMIT_FIRST_LEFT =>
    buttonChar <= Left(0);
    buttonTxEn <= ACTIVE;
    nextButtonState <= WAIT_AFTER_FIRST_LEFT;

--WAIT_AFTER_FIRST_LEFT----------------------------
when WAIT_AFTER_FIRST_LEFT =>
    if (txComplete = ACTIVE) then
        nextButtonState <= TRANSMIT_SECOND_LEFT;
    else
        nextButtonState <= WAIT_AFTER_FIRST_LEFT;
    end if;

--TRANSMIT_SECOND_LEFT-------------------------------
when TRANSMIT_SECOND_LEFT =>
    buttonChar <= Left(1);
    buttonTxEn <= ACTIVE;
    nextButtonState <= WAIT_AFTER_SECOND_LEFT;

--WAIT_AFTER_SECOND_LEFT----------------------------
when WAIT_AFTER_SECOND_LEFT =>
    if (txComplete = ACTIVE) then
        nextButtonState <= TRANSMIT_THIRD_LEFT;
    else
        nextButtonState <= WAIT_AFTER_SECOND_LEFT;
    end if;

--TRANSMIT_THIRD_LEFT--------------------------------
when TRANSMIT_THIRD_LEFT =>
    buttonChar <= Left(2);
    buttonTxEn <= ACTIVE;
    nextButtonState <= WAIT_AFTER_THIRD_LEFT;

--WAIT_AFTER_THIRD_LEFT-----------------------------------
when WAIT_AFTER_THIRD_LEFT =>
    if (txComplete = ACTIVE) then
        nextButtonState <= TRANSMIT_FOURTH_LEFT;
    else
        nextButtonState <= WAIT_AFTER_THIRD_LEFT;
    end if;

--TRANSMIT_FOURTH_LEFT-------------------------------
when TRANSMIT_FOURTH_LEFT =>
    buttonChar <= Left(3);
    buttonTxEn <= ACTIVE;
    nextButtonState <= WAIT_AFTER_FOURTH_LEFT;

--WAIT_AFTER_FOURTH_LEFT----------------------------
when WAIT_AFTER_FOURTH_LEFT =>
    if (txComplete = ACTIVE) then
        nextButtonState <= TRANSMIT_FIFTH_LEFT;
    else
        nextButtonState <= WAIT_AFTER_FOURTH_LEFT;
    end if;

--TRANSMIT_FIFTH_LEFT-------------------------------
when TRANSMIT_FIFTH_LEFT =>
```

```
        buttonChar <= Left(4);
        buttonTxEn <= ACTIVE;
        nextButtonState <= WAIT_AFTER_FIFTH_LEFT;

    --WAIT_AFTER_FIFTH_LEFT--------------------------
    when WAIT_AFTER_FIFTH_LEFT =>
        if (txComplete = ACTIVE) then
            nextButtonState <= CHECK_RIGHT_BUTTON;
        else
            nextButtonState <= WAIT_AFTER_FIFTH_LEFT;
        end if;

    --CHECK_RIGHT_BUTTON-----------------------------
    when CHECK_RIGHT_BUTTON =>
        if (rightButtonSync = ACTIVE) then
            nextButtonState <= TRANSMIT_FIRST_RIGHT;
        else
            nextButtonState <= CHECK_CENTER_BUTTON;
        end if;

    --TRANSMIT_FIRST_RIGHT---------------------------
    when TRANSMIT_FIRST_RIGHT =>
        buttonChar <= Right(0);
        buttonTxEn <= ACTIVE;
        nextButtonState <= WAIT_AFTER_FIRST_RIGHT;

    --WAIT_AFTER_FIRST_RIGHT-------------------------
    when WAIT_AFTER_FIRST_RIGHT =>
        if (txComplete = ACTIVE) then
            nextButtonState <= TRANSMIT_SECOND_RIGHT;
        else
            nextButtonState <= WAIT_AFTER_FIRST_RIGHT;
        end if;

    --TRANSMIT_SECOND_RIGHT--------------------------
    when TRANSMIT_SECOND_RIGHT =>
        buttonChar <= Right(1);
        buttonTxEn <= ACTIVE;
        nextButtonState <= WAIT_AFTER_SECOND_RIGHT;

    --WAIT_AFTER_SECOND_RIGHT------------------------
    when WAIT_AFTER_SECOND_RIGHT =>
        if (txComplete = ACTIVE) then
            nextButtonState <= TRANSMIT_THIRD_RIGHT;
        else
            nextButtonState <= WAIT_AFTER_SECOND_RIGHT;
        end if;

    --TRANSMIT_THIRD_RIGHT---------------------------
    when TRANSMIT_THIRD_RIGHT =>
        buttonChar <= Right(2);
        buttonTxEn <= ACTIVE;
        nextButtonState <= WAIT_AFTER_THIRD_RIGHT;

    --WAIT_AFTER_THIRD_RIGHT-------------------------
    when WAIT_AFTER_THIRD_RIGHT =>
        if (txComplete = ACTIVE) then
```

```
            nextButtonState <= TRANSMIT_FOURTH_RIGHT;
        else
            nextButtonState <= WAIT_AFTER_THIRD_RIGHT;
        end if;

    --TRANSMIT_FOURTH_RIGHT------------------------
    when TRANSMIT_FOURTH_RIGHT =>
        buttonChar <= Right(3);
        buttonTxEn <= ACTIVE;
        nextButtonState <= WAIT_AFTER_FOURTH_RIGHT;

    --WAIT_AFTER_FOURTH_RIGHT-------------------------
    when WAIT_AFTER_FOURTH_RIGHT =>
        if (txComplete = ACTIVE) then
            nextButtonState <= TRANSMIT_FIFTH_RIGHT;
        else
            nextButtonState <= WAIT_AFTER_FOURTH_RIGHT;
        end if;

    --TRANSMIT_FIFTH_RIGHT------------------------
    when TRANSMIT_FIFTH_RIGHT =>
        buttonChar <= Right(4);
        buttonTxEn <= ACTIVE;
        nextButtonState <= WAIT_AFTER_FIFTH_RIGHT;

    --WAIT_AFTER_FIFTH_RIGHT-------------------------
    when WAIT_AFTER_FIFTH_RIGHT =>
        if (txComplete = ACTIVE) then
            nextButtonState <= TRANSMIT_SIXTH_RIGHT;
        else
            nextButtonState <= WAIT_AFTER_FIFTH_RIGHT;
        end if;

    --TRANSMIT_SIXTH_RIGHT------------------------
    when TRANSMIT_SIXTH_RIGHT =>
        buttonChar <= Right(5);
        buttonTxEn <= ACTIVE;
        nextButtonState <= WAIT_AFTER_SIXTH_RIGHT;

    --WAIT_AFTER_SIXTH_RIGHT-------------------------
    when WAIT_AFTER_SIXTH_RIGHT =>
        if (txComplete = ACTIVE) then
            nextButtonState <= CHECK_CENTER_BUTTON;
        else
            nextButtonState <= WAIT_AFTER_SIXTH_RIGHT;
        end if;

    --CHECK_CENTER_BUTTON------------------------
    when CHECK_CENTER_BUTTON =>
        if (centerButtonSync = ACTIVE) then
            nextButtonState <= TRANSMIT_FIRST_CENTER;
        else
            nextButtonState <= BUTTON_TX_FINISHED;
        end if;

    --TRANSMIT_FIRST_CENTER------------------------
    when TRANSMIT_FIRST_CENTER =>
```

```
        buttonChar <= Center(0);
        buttonTxEn <= ACTIVE;
        nextButtonState <= WAIT_AFTER_FIRST_CENTER;

    --WAIT_AFTER_FIRST_CENTER-------------------------
    when WAIT_AFTER_FIRST_CENTER =>
        if (txComplete = ACTIVE) then
            nextButtonState <= TRANSMIT_SECOND_CENTER;
        else
            nextButtonState <= WAIT_AFTER_FIRST_CENTER;
        end if;

    --TRANSMIT_SECOND_CENTER-------------------------
    when TRANSMIT_SECOND_CENTER =>
        buttonChar <= Center(1);
        buttonTxEn <= ACTIVE;
        nextButtonState <= WAIT_AFTER_SECOND_CENTER;

    --WAIT_AFTER_SECOND_CENTER-------------------------
    when WAIT_AFTER_SECOND_CENTER =>
        if (txComplete = ACTIVE) then
            nextButtonState <= TRANSMIT_THIRD_CENTER;
        else
            nextButtonState <= WAIT_AFTER_SECOND_CENTER;
        end if;

    --TRANSMIT_THIRD_CENTER-------------------------
    when TRANSMIT_THIRD_CENTER =>
        buttonChar <= Center(2);
        buttonTxEn <= ACTIVE;
        nextButtonState <= WAIT_AFTER_THIRD_CENTER;

    --WAIT_AFTER_THIRD_CENTER-------------------------
    when WAIT_AFTER_THIRD_CENTER =>
        if (txComplete = ACTIVE) then
            nextButtonState <= TRANSMIT_FOURTH_CENTER;
        else
            nextButtonState <= WAIT_AFTER_THIRD_CENTER;
        end if;

    --TRANSMIT_FOURTH_CENTER-------------------------
    when TRANSMIT_FOURTH_CENTER =>
        buttonChar <= Center(3);
        buttonTxEn <= ACTIVE;
        nextButtonState <= WAIT_AFTER_FOURTH_CENTER;

    --WAIT_AFTER_FOURTH_CENTER-------------------------
    when WAIT_AFTER_FOURTH_CENTER =>
        if (txComplete = ACTIVE) then
            nextButtonState <= TRANSMIT_FIFTH_CENTER;
        else
            nextButtonState <= WAIT_AFTER_FOURTH_CENTER;
        end if;

    --TRANSMIT_FIFTH_CENTER-------------------------
    when TRANSMIT_FIFTH_CENTER =>
        buttonChar <= Center(4);
```

```
                buttonTxEn <= ACTIVE;
                nextButtonState <= WAIT_AFTER_FIFTH_CENTER;

            --WAIT_AFTER_FIFTH_CENTER-------------------------
            when WAIT_AFTER_FIFTH_CENTER =>
                if (txComplete = ACTIVE) then
                    nextButtonState <= TRANSMIT_SIXTH_CENTER;
                else
                    nextButtonState <= WAIT_AFTER_FIFTH_CENTER;
                end if;

            --TRANSMIT_SIXTH_CENTER-------------------------
            when TRANSMIT_SIXTH_CENTER =>
                buttonChar <= Center(5);
                buttonTxEn <= ACTIVE;
                nextButtonState <= WAIT_AFTER_SIXTH_CENTER;

            --WAIT_AFTER_SIXTH_CENTER-------------------------
            when WAIT_AFTER_SIXTH_CENTER =>
                if (txComplete = ACTIVE) then
                    nextButtonState <= TRANSMIT_SEVENTH_CENTER;
                else
                    nextButtonState <= WAIT_AFTER_SIXTH_CENTER;
                end if;

            --TRANSMIT_SEVENTH_CENTER-------------------------
            when TRANSMIT_SEVENTH_CENTER =>
                buttonChar <= Center(6);
                buttonTxEn <= ACTIVE;
                nextButtonState <= WAIT_AFTER_SEVENTH_CENTER;

            --WAIT_AFTER_SEVENTH_CENTER-------------------------
            when WAIT_AFTER_SEVENTH_CENTER =>
                if (txComplete = ACTIVE) then
                    nextButtonState <= BUTTON_TX_FINISHED;
                else
                    nextButtonState <= WAIT_AFTER_SEVENTH_CENTER;
                end if;

            --BUTTON_TX_FINISHED---------------------------------
            when BUTTON_TX_FINISHED =>
                buttonTxFinished <= ACTIVE;
                nextButtonState <= WAIT_FOR_BUTTON_CONTROL;
        end case;
    end process BUTTON_STATE_TRANSITION;

    --Latches for the Digits(enable signals comes from Ascii State Machine)--
----------\
    --Digit1----------------------------
    DIGIT01_LATCH:process (reset, clock)
    begin
        if (reset = ACTIVE) then
            digit1 <= (others => not ACTIVE);
        elsif (rising_edge(clock)) then
            if (digit01En = ACTIVE) then
                digit1 <= myChar(3 downto 0);
            end if;
```

```
    end if;
end process DIGIT01_LATCH;

--Digit2------------------------------
DIGIT02_LATCH:process (reset, clock)
begin
    if (reset = ACTIVE) then
        digit2 <= (others => not ACTIVE);
    elsif (rising_edge(clock)) then
        if (digit02En = ACTIVE) then
            digit2 <= myChar(3 downto 0);
        end if;
    end if;
end process DIGIT02_LATCH;

--Digit3------------------------------
DIGIT03_LATCH:process (reset, clock)
begin
    if (reset = ACTIVE) then
        digit3 <= (others => not ACTIVE);
    elsif (rising_edge(clock)) then
        if (digit03En = ACTIVE) then
            digit3 <= myChar(3 downto 0);
        end if;
    end if;
end process DIGIT03_LATCH;

--Digit4------------------------------
DIGIT04_LATCH:process (reset, clock)
begin
    if (reset = ACTIVE) then
        digit4 <= (others => not ACTIVE);
    elsif (rising_edge(clock)) then
        if (digit04En = ACTIVE) then
            digit4 <= myChar(3 downto 0);
        end if;
    end if;
end process DIGIT04_LATCH;

DIGIT_TO_NUM_CONVERT: process (digit1, digit2)
    variable onesDigit: integer range 0 to 9;
    variable tensDigit: integer range 0 to 9;
begin
    onesDigit := to_integer(unsigned(digit1));
    tensDigit := to_integer(unsigned(digit2));

    tensDigit := tensDigit * 10;
    userNum <= tensDigit + onesDigit;
end process DIGIT_TO_NUM_CONVERT;

--sets segDigits to the digits from the Ascii State Machine--------------
SEG_REGISTER:process (reset, clock)
begin
    if (reset = ACTIVE) then
        segDigit3 <= (others => not ACTIVE);
        segDigit2 <= (others => not ACTIVE);
        segDigit1 <= (others => not ACTIVE);
```

```
            segDigit0 <= (others => not ACTIVE);
        elsif (rising_edge(clock)) then
            if (segEn = ACTIVE) then
                segDigit3 <= digit4;
                segDigit2 <= digit3;
                segDigit1 <= digit2;
                segDigit0 <= digit1;
            end if;
        end if;
    end process SEG_REGISTER;

    --Driver to turn LEDs on or off depending on userNum--------------------
    LED_DRIVER: process (reset,clock)
        variable tempLeds: std_logic_vector(15 downto 0);
    begin
        if (reset = ACTIVE) then
            leds <= (others => not ACTIVE);
            tempLeds := (others => not ACTIVE);
        elsif (rising_edge(clock)) then
            if (ledOnEn = ACTIVE) then
                tempLeds(userNum) := ACTIVE;
            elsif (ledOffEn = ACTIVE) then
                tempLeds(userNum) := not ACTIVE;
            end if;
            leds <= tempLeds;
        end if;
    end process LED_DRIVER;

    --Transmission Organizer to only send one char and enable line to
Transmitter----
    charToSend <= buttonChar or switchChar;
    txEn <= buttonTxEn or switchTxEn;

end SerialControl_ARCH;
```

Test Bench Code:

```
----------------------------------------------------------------------------
-- SerialControl_tb
-- Sofiyan Monga
--
--      Testbench for SerialControl component. Apply certain inputs and
control
--      serialIn to determine correct performance of the receiver,
transmitter, and
--      the rest of the component.
--
----------------------------------------------------------------------------


library ieee;
use ieee.std_logic_1164.all;

entity SerialControl_tb is
--  Port ( );
end SerialControl_tb;

architecture SerialControl_tb_ARCH of SerialControl_tb is

    --constants-------------------------------------------
    constant ACTIVE    : std_logic := '1';
    constant CLOCK_FREQ: integer   := 100_000_000;
    constant BAUD_RATE : integer   := 115_200;
    constant BIT_TIME  : time      := (1_000_000_000/BAUD_RATE) * 1 ns;

    --unit under test-------------------------------------
    component SerialControl
        Port (
           clock           : in std_logic;
           rawReset        : in std_logic;
           serialIn        : in std_logic;
           upButton        : in std_logic;
           leftButton      : in std_logic;
           rightButton     : in std_logic;
           centerButton    : in std_logic;
           switches        : in std_logic_vector (15 downto 0);
           leds            : out std_logic_vector (15 downto 0);
           segments        : out std_logic_vector (6 downto 0);
           anodes          : out std_logic_vector (3 downto 0);
           serialOut       : out std_logic);
    end component;

    --UUT signals-----------------------------------------
    signal clock       : std_logic;
    signal downButton  : std_logic;
    signal rightButton : std_logic;
    signal leftButton  : std_logic;
    signal centerButton: std_logic;
    signal upButton    : std_logic;
    signal serialIn    : std_logic;
    signal serialOut   : std_logic;
    signal switches    : std_logic_vector(15 downto 0);
```

```
    signal leds         : std_logic_vector(15 downto 0);
    signal segments     : std_logic_vector(6 downto 0);
    signal anodes       : std_logic_vector(3 downto 0);

begin
    --unit-under-test-----------------------------------------------
    UUT: SerialControl port map (
        clock        => clock,
        rawReset     => downButton,
        leftButton   => leftButton,
        rightButton  => rightButton,
        centerButton => centerButton,
        upButton     => upButton,
        serialIn     => serialIn,
        serialOut    => serialOut,
        switches     => switches,
        leds         => leds,
        segments     => segments,
        anodes       => anodes);

    --clock-definition-----------------------------------------------
    SYS_CLOCK: process
    begin
        clock <= not ACTIVE;
        wait for 5 ns;
        clock <= ACTIVE;
        wait for 5 ns;
    end process SYS_CLOCK;

    --reset-driver---------------------------------------------------
    SYS_RESET: process
    begin
        downButton <= ACTIVE;
        wait for 15 ns;
        downButton <= not ACTIVE;
        wait;
    end process SYS_RESET;

    --input-driver---------------------------------------------------
    INPUT_DRIVER: process
    begin
        --initialize inputs to zero---------------------------------
        switches <= (others => not ACTIVE);
        upButton <= not ACTIVE;
        centerButton <= not ACTIVE;
        leftButton <= not ACTIVE;
        rightButton <= not ACTIVE;

       --set serial line to idle-----------------------------------
       serialIn <= '1';
       wait for 316 ns;

       --transmit 'L'------------01001100------------------------
        serialIn <= '0';
        wait for BIT_TIME;

        serialIn <= '0';
```

```
wait for BIT_TIME;
serialIn <= '0';
wait for BIT_TIME;
serialIn <= '1';
wait for BIT_TIME;
serialIn <= '1';
wait for BIT_TIME;
serialIn <= '0';
wait for BIT_TIME;
serialIn <= '0';
wait for BIT_TIME;
serialIn <= '1';
wait for BIT_TIME;
serialIn <= '0';
wait for BIT_TIME;

serialIn <= '1';
wait for BIT_TIME;

--transmit first digit for 'L'-------00110001-------------
serialIn <= '0';
wait for BIT_TIME;

serialIn <= '1';
wait for BIT_TIME;
serialIn <= '0';
wait for BIT_TIME;
serialIn <= '0';
wait for BIT_TIME;
serialIn <= '0';
wait for BIT_TIME;
serialIn <= '1';
wait for BIT_TIME;
serialIn <= '1';
wait for BIT_TIME;
serialIn <= '0';
wait for BIT_TIME;
serialIn <= '0';
wait for BIT_TIME;

serialIn <= '1';
wait for BIT_TIME;

--transmit second digit for 'L'-------00110010-------------
serialIn <= '0';
wait for BIT_TIME;

serialIn <= '0';
wait for BIT_TIME;
serialIn <= '1';
wait for BIT_TIME;
serialIn <= '0';
wait for BIT_TIME;
serialIn <= '0';
wait for BIT_TIME;
serialIn <= '1';
wait for BIT_TIME;
```

```
serialIn <= '1';
wait for BIT_TIME;
serialIn <= '0';
wait for BIT_TIME;
serialIn <= '0';
wait for BIT_TIME;

serialIn <= '1';
wait for BIT_TIME;



--transmit 'M'------------01001101-------------------------
serialIn <= '0';
wait for BIT_TIME;

serialIn <= '1';
wait for BIT_TIME;
serialIn <= '0';
wait for BIT_TIME;
serialIn <= '1';
wait for BIT_TIME;
serialIn <= '1';
wait for BIT_TIME;
serialIn <= '0';
wait for BIT_TIME;
serialIn <= '0';
wait for BIT_TIME;
serialIn <= '1';
wait for BIT_TIME;
serialIn <= '0';
wait for BIT_TIME;

serialIn <= '1';
wait for BIT_TIME;

--transmit first digit for 'M'-------00110001-------------
serialIn <= '0';
wait for BIT_TIME;

serialIn <= '1';
wait for BIT_TIME;
serialIn <= '0';
wait for BIT_TIME;
serialIn <= '0';
wait for BIT_TIME;
serialIn <= '0';
wait for BIT_TIME;
serialIn <= '1';
wait for BIT_TIME;
serialIn <= '1';
wait for BIT_TIME;
serialIn <= '0';
wait for BIT_TIME;
serialIn <= '0';
wait for BIT_TIME;
```

```
serialIn <= '1';
wait for BIT_TIME;

--transmit second digit for 'M'-------00110010-------------
serialIn <= '0';
wait for BIT_TIME;

serialIn <= '0';
wait for BIT_TIME;
serialIn <= '1';
wait for BIT_TIME;
serialIn <= '0';
wait for BIT_TIME;
serialIn <= '0';
wait for BIT_TIME;
serialIn <= '1';
wait for BIT_TIME;
serialIn <= '1';
wait for BIT_TIME;
serialIn <= '0';
wait for BIT_TIME;
serialIn <= '0';
wait for BIT_TIME;

serialIn <= '1';
wait for BIT_TIME;



--transmit 'D'-------------01000100-----------------------
serialIn <= '0';
wait for BIT_TIME;

serialIn <= '0';
wait for BIT_TIME;
serialIn <= '0';
wait for BIT_TIME;
serialIn <= '1';
wait for BIT_TIME;
serialIn <= '0';
wait for BIT_TIME;
serialIn <= '0';
wait for BIT_TIME;
serialIn <= '0';
wait for BIT_TIME;
serialIn <= '1';
wait for BIT_TIME;
serialIn <= '0';
wait for BIT_TIME;

serialIn <= '1';
wait for BIT_TIME;

--transmit first digit for 'D'-------00110001-------------
serialIn <= '0';
```

```
wait for BIT_TIME;

serialIn <= '1';
wait for BIT_TIME;
serialIn <= '0';
wait for BIT_TIME;
serialIn <= '0';
wait for BIT_TIME;
serialIn <= '0';
wait for BIT_TIME;
serialIn <= '1';
wait for BIT_TIME;
serialIn <= '1';
wait for BIT_TIME;
serialIn <= '0';
wait for BIT_TIME;
serialIn <= '0';
wait for BIT_TIME;

serialIn <= '1';
wait for BIT_TIME;

--transmit second digit for 'D'-------00110010-------------
serialIn <= '0';
wait for BIT_TIME;

serialIn <= '0';
wait for BIT_TIME;
serialIn <= '1';
wait for BIT_TIME;
serialIn <= '0';
wait for BIT_TIME;
serialIn <= '0';
wait for BIT_TIME;
serialIn <= '1';
wait for BIT_TIME;
serialIn <= '1';
wait for BIT_TIME;
serialIn <= '0';
wait for BIT_TIME;
serialIn <= '0';
wait for BIT_TIME;

serialIn <= '1';
wait for BIT_TIME;

--transmit third digit for 'D'-------00110011-------------
serialIn <= '0';
wait for BIT_TIME;

serialIn <= '1';
wait for BIT_TIME;
serialIn <= '1';
wait for BIT_TIME;
serialIn <= '0';
wait for BIT_TIME;
serialIn <= '0';
```

```
wait for BIT_TIME;
serialIn <= '1';
wait for BIT_TIME;
serialIn <= '1';
wait for BIT_TIME;
serialIn <= '0';
wait for BIT_TIME;
serialIn <= '0';
wait for BIT_TIME;

serialIn <= '1';
wait for BIT_TIME;

--transmit fourth digit for 'D'-------00110100-------------
serialIn <= '0';
wait for BIT_TIME;

serialIn <= '0';
wait for BIT_TIME;
serialIn <= '0';
wait for BIT_TIME;
serialIn <= '1';
wait for BIT_TIME;
serialIn <= '0';
wait for BIT_TIME;
serialIn <= '1';
wait for BIT_TIME;
serialIn <= '1';
wait for BIT_TIME;
serialIn <= '0';
wait for BIT_TIME;
serialIn <= '0';
wait for BIT_TIME;

serialIn <= '1';
wait for BIT_TIME;



--transmit 'B' after pushing buttons-------------01000010------------
upButton <= ACTIVE;
leftButton <= ACTIVE;

serialIn <= '0';
wait for BIT_TIME;

serialIn <= '0';
wait for BIT_TIME;
serialIn <= '1';
wait for BIT_TIME;
serialIn <= '0';
wait for BIT_TIME;
serialIn <= '0';
wait for BIT_TIME;
serialIn <= '0';
wait for BIT_TIME;
serialIn <= '0';
```

```
        wait for BIT_TIME;
        serialIn <= '1';
        wait for BIT_TIME;
        serialIn <= '0';
        wait for BIT_TIME;

        serialIn <= '1';
        wait for BIT_TIME;


        --transmit 'S' after activating one of the switches-------01010011---
--        switches <= "0000000000000001";


--        serialIn <= '0';
--        wait for BIT_TIME;

--        serialIn <= '1';
--        wait for BIT_TIME;
--        serialIn <= '1';
--        wait for BIT_TIME;
--        serialIn <= '0';
--        wait for BIT_TIME;
--        serialIn <= '0';
--        wait for BIT_TIME;
--        serialIn <= '1';
--        wait for BIT_TIME;
--        serialIn <= '0';
--        wait for BIT_TIME;
--        serialIn <= '1';
--        wait for BIT_TIME;
--        serialIn <= '0';
--        wait for BIT_TIME;

--        serialIn <= '1';
--        wait for BIT_TIME;

--        --transmit first digit for 'S'-------00110000-------------
--        serialIn <= '0';
--        wait for BIT_TIME;

--        serialIn <= '0';
--        wait for BIT_TIME;
--        serialIn <= '0';
--        wait for BIT_TIME;
--        serialIn <= '0';
--        wait for BIT_TIME;
--        serialIn <= '0';
--        wait for BIT_TIME;
--        serialIn <= '1';
--        wait for BIT_TIME;
--        serialIn <= '1';
--        wait for BIT_TIME;
--        serialIn <= '0';
--        wait for BIT_TIME;
--        serialIn <= '0';
```

```
--        wait for BIT_TIME;

--        serialIn <= '1';
--        wait for BIT_TIME;

--        --transmit second digit for 'S'-------00110000-------------
--        serialIn <= '0';
--        wait for BIT_TIME;

--        serialIn <= '0';
--        wait for BIT_TIME;
--        serialIn <= '0';
--        wait for BIT_TIME;
--        serialIn <= '0';
--        wait for BIT_TIME;
--        serialIn <= '0';
--        wait for BIT_TIME;
--        serialIn <= '1';
--        wait for BIT_TIME;
--        serialIn <= '1';
--        wait for BIT_TIME;
--        serialIn <= '0';
--        wait for BIT_TIME;
--        serialIn <= '0';
--        wait for BIT_TIME;

--        serialIn <= '1';
--        wait for BIT_TIME;


        wait;
    end process INPUT_DRIVER;

end SerialControl_tb_ARCH;
```