Kennesaw State University

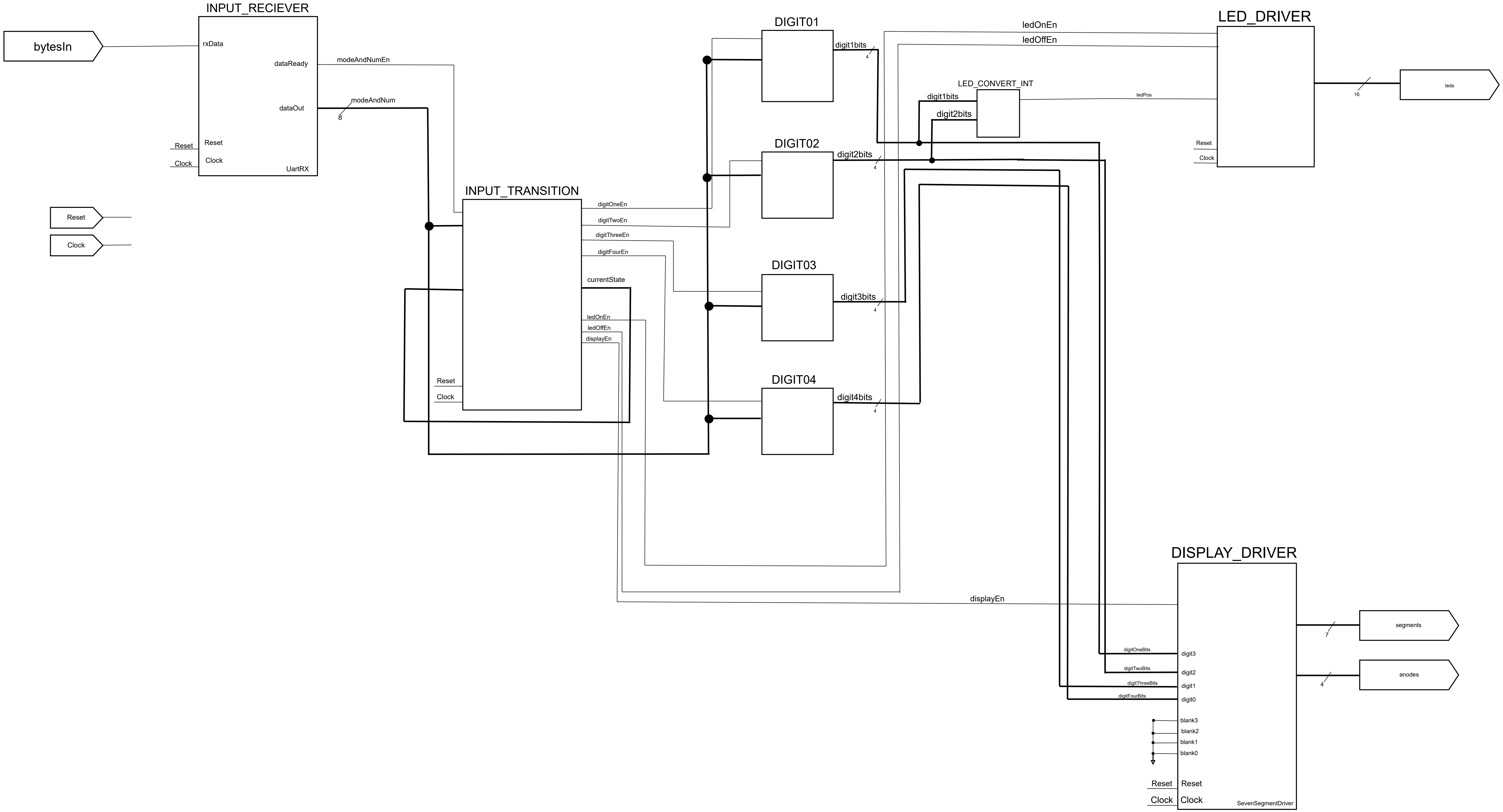
SerialControl Lab Report

Clarence Barron, Rodrigo Corral, Paul Hughes

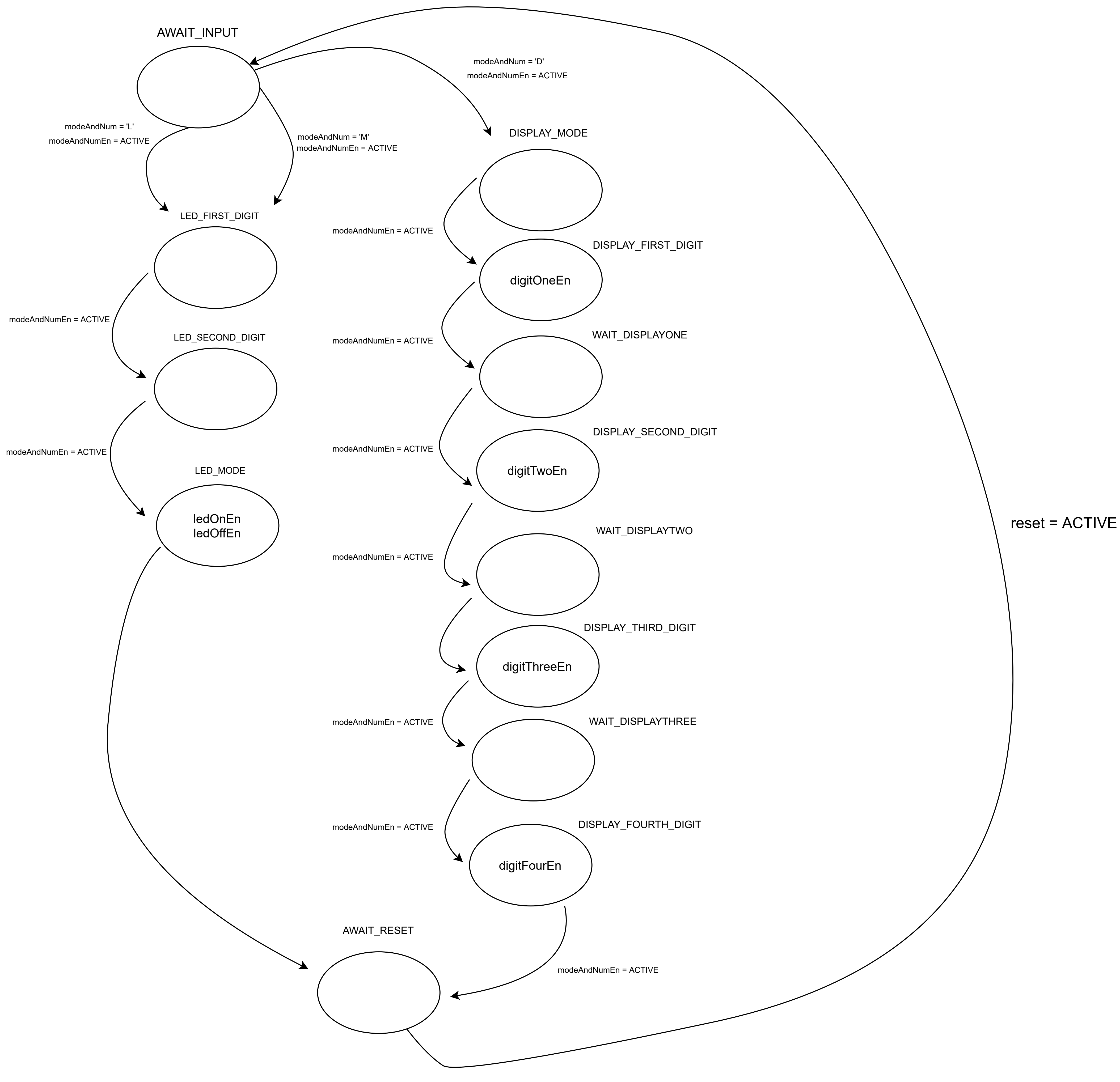
CPE 3020

Professor Scott Tippens

April 29, 2022



bytesIn
INPUT_RECIEVER
rxData
dataReady
dataOut
Reset
Clock
UartRX
modeAndNumEn
modeAndNum
8
Reset
Clock
INPUT_TRANSITION
digitOneEn
digitTwoEn
digitThreeEn
digitFourEn
currentState
ledOnEn
ledOffEn
displayEn
Reset
Clock
DIGIT01
digit1bits
DIGIT02
digit2bits
DIGIT03
digit3bits
DIGIT04
digit4bits
4
LED_CONVERT_INT
digit1bits
digit2bits
ledPos
ledOnEn
ledOffEn
LED_DRIVER
Reset
Clock
16
leds
DISPLAY_DRIVER
displayEn
digitOneBits
digitTwoBits
digitThreeBits
digitFourBits
digit3
digit2
digit1
digit0
blank3
blank2
blank1
blank0
Reset
Clock
SevenSegmentDriver
7
segments
4
anodes



AWAIT_INPUT
modeAndNum = 'D'
modeAndNumEn = ACTIVE
modeAndNum = 'L'
modeAndNumEn = ACTIVE
modeAndNum = 'M'
modeAndNumEn = ACTIVE
DISPLAY_MODE
LED_FIRST_DIGIT
modeAndNumEn = ACTIVE
DISPLAY_FIRST_DIGIT
digitOneEn
modeAndNumEn = ACTIVE
modeAndNumEn = ACTIVE
WAIT_DISPLAYONE
LED_SECOND_DIGIT
DISPLAY_SECOND_DIGIT
modeAndNumEn = ACTIVE
modeAndNumEn = ACTIVE
digitTwoEn
LED_MODE
ledOnEn
ledOffEn
reset = ACTIVE
WAIT_DISPLAYTWO
modeAndNumEn = ACTIVE
DISPLAY_THIRD_DIGIT
digitThreeEn
WAIT_DISPLAYTHREE
modeAndNumEn = ACTIVE
DISPLAY_FOURTH_DIGIT
modeAndNumEn = ACTIVE
digitFourEn
AWAIT_RESET
modeAndNumEn = ACTIVE

```vhdl
---------------------------------------------------------------------------------
-----
-- Group Members; Clarence Barron, Rodrigo Corral, Paul Hughes

-- This is the wrapper file for the Serial Control file of the system.
---------------------------------------------------------------------------------
-----


library ieee;
use ieee.std_logic_1164.all;

entity SerialControl_Basys3 is
    Port ( clk:    in std_logic;
           btnD:   in std_logic;
           RsRx:   in std_logic;
           led:    out std_logic_vector (15 downto 0);
           seg:    out std_logic_vector (6 downto 0);
           an:     out std_logic_vector (3 downto 0));
end SerialControl_Basys3;

architecture SerialControl_Basys3_ARCH of SerialControl_Basys3 is
           component SerialControl
            Port ( reset : in std_logic;
                   clock : in std_logic;
                   bytesIn : in std_logic;
                   segments : out std_logic_vector (6 downto 0);
                   anodes : out std_logic_vector (3 downto 0);
                   leds : out std_logic_vector (15 downto 0));
            end component SerialControl;
begin
SYSTEM_WORKINGS: SerialControl
            port map(
                   reset       => btnD,
                   clock       => clk,
                   bytesIn     => RsRx,
                   leds        => led,
                   segments    => seg,
                   anodes      => an);



end SerialControl_Basys3_ARCH;
```

```
-------------------------------------------------------------------------------
-----
-- Group Members; Clarence Barron, Rodrigo Corral, Paul Hughes

                          -- Serial Control --
-- This is the component is suppose to take into bytes from the pc serial
connection
-- and control the display and leds accordingly depending on the mode given.
-- L = Leds On, M = Leds Off, D = Display
-------------------------------------------------------------------------------
-----


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

entity SerialControl is
    Port ( reset : in std_logic;
           clock : in std_logic;
           bytesIn : in std_logic;
           segments : out std_logic_vector (6 downto 0);
           anodes : out std_logic_vector (3 downto 0);
           leds : out std_logic_vector (15 downto 0));
end SerialControl;

architecture SerialControl_ARCH of SerialControl is

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

component UartRx
        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)
            );
```

```vhdl
    end component UartRx;
    type      serialMode_t  is (AWAIT_INPUT, LED_MODE, AWAIT_RESET,
LED_FIRST_DIGIT, LED_SECOND_DIGIT, DISPLAY_MODE, DISPLAY_FIRST_DIGIT,
WAIT_DISPLAYONE, WAIT_DISPLAYTWO, WAIT_DISPLAYTHREE, DISPLAY_SECOND_DIGIT,
DISPLAY_THIRD_DIGIT, DISPLAY_FOURTH_DIGIT);
    type      ledMode_t     is (LED_WAIT, LED_OFF, LED_ON);
    constant ACTIVE:          std_logic := '1';
    signal    modeAndNum:     std_logic_vector(7 downto 0);
    signal    currentState:   serialMode_t;
    signal    nextState:      serialMode_t;
    signal    modeAndNumEn:   std_logic;
    signal    ledOnEn:        std_logic;
    signal    ledOffEn:       std_logic;
    signal    digitOneEn:     std_logic;
    signal    digitTwoEn:     std_logic;
    signal    digitThreeEn:   std_logic;
    signal    digitFourEn:    std_logic;
    signal    digitOneBits:   std_logic_vector(3 downto 0);
    signal    digitTwoBits:   std_logic_vector(3 downto 0);
    signal    digitThreeBits: std_logic_vector(3 downto 0);
    signal    digitFourBits:  std_logic_vector(3 downto 0);
    signal    ledMode:        ledMode_t;
    signal    ledPos:         integer range 0 to 16;

begin
-- The reciever taking in the bytes --
INPUT_RECEIVER: UartRx
       port map(
                 clock =>     clock,
                 reset =>     reset,
                 rxData =>    bytesIn,
                 dataReady => modeAndNumEn,
                 dataOut =>   modeAndNum);
-- Just a state proces to take in the current
-- state of the state machine
INPUT_STATE: process (reset,clock)
begin
      if (reset = ACTIVE) then
        currentState <= AWAIT_INPUT;
      elsif (rising_edge(clock)) then
            currentstate <= nextState;
      end if;
end process INPUT_STATE;

-- The state machine is meant to change depending on what is given
-- from the reciever and changes accordingly and
-- stores the digits given in from the reciever
INPUT_TRANSITION: process (modeAndNum, modeAndNumEn)
begin
    case currentState is
        when AWAIT_INPUT =>
             if(modeAndNumEn = ACTIVE and modeAndNum = x"4C") then
                ledMode <= LED_ON;
                nextState <= LED_FIRST_DIGIT;
             elsif(modeAndNumEn = ACTIVE and modeAndNum = x"4D") then
                ledMode <= LED_OFF;
                nextState <= LED_FIRST_DIGIT;
```

```
     elsif(modeAndNumEn = ACTIVE and modeAndNum = x"44") then
        nextState <= DISPLAY_MODE;
     else
        nextState <= AWAIT_INPUT;
     end if;


when LED_FIRST_DIGIT =>
    digitOneEn <= ACTIVE;
    digitTwoEn <= not ACTIVE;
    if (modeAndNumEn = ACTIVE) then
        nextState <= LED_SECOND_DIGIT;
    end if;

when LED_SECOND_DIGIT =>
    digitOneEn <= not ACTIVE;
    digitTwoEn <= ACTIVE;
    if(modeAndNumEn <= ACTIVE) then
        nextstate <= LED_MODE;
    end if;

 when LED_MODE =>
   if (ledMode = LED_ON) then
            ledOnEn <= ACTIVE;
            nextState <= AWAIT_RESET;
        elsif(ledMode = LED_OFF) then
            ledOffEn <= ACTIVE;
            nextState <= AWAIT_RESET;
  end if;

 when DISPLAY_MODE =>
    if (modeAndNumEn = ACTIVE) then
        nextState <= DISPLAY_FIRST_DIGIT;
    else
        nextState <= DISPLAY_MODE;
    end if;

 when DISPLAY_FIRST_DIGIT =>
      digitOneEn <= ACTIVE;
      nextState <= WAIT_DISPLAYONE;

 when WAIT_DISPLAYONE =>
       if (modeAndNumEn = ACTIVE) then
          nextState <= DISPLAY_SECOND_DIGIT;
       else
           nextState <= WAIT_DISPLAYONE;
           digitOneEn <= not ACTIVE;
       end if;

  when DISPLAY_SECOND_DIGIT =>
      digitTwoEn <= ACTIVE;
      nextState <= WAIT_DISPLAYTWO;

  when WAIT_DISPLAYTWO =>
       if (modeAndNumEn = ACTIVE) then
          nextState <= DISPLAY_THIRD_DIGIT;
       else
```

```vhdl
                    nextState <= WAIT_DISPLAYTWO;
                    digitTwoEn <= not ACTIVE;
                end if;

           when DISPLAY_THIRD_DIGIT =>
                digitThreeEn <= ACTIVE;
                nextState <= WAIT_DISPLAYTHREE;

          when WAIT_DISPLAYTHREE =>
                if (modeAndNumEn = ACTIVE) then
                   nextState <= DISPLAY_FOURTH_DIGIT;
                   digitThreeEn <= not ACTIVE;
                else
                    nextState <= WAIT_DISPLAYTHREE;
                    digitThreeEn <= not ACTIVE;
                end if;

            when DISPLAY_FOURTH_DIGIT =>
               digitFourEn <=  ACTIVE;
               digitTwoEn <= not ACTIVE;
               if (modeAndNumEn = ACTIVE) then
                 digitFourEn <= not ACTIVE;
                 nextState <= AWAIT_RESET;
            end if;

            when AWAIT_RESET =>
                 digitOneEn      <=   not ACTIVE;
                 digitTwoEn      <=   not ACTIVE;
                 digitThreeEn    <=   not ACTIVE;
                 digitFourEn     <=   not ACTIVE;
    end case;
end process INPUT_TRANSITION;

-- Holds the first digit given by the last 3 bits of
-- the byte given from the reciever
-- The bits from the byte given [3 downto 0]
-- is consitant for all the digit processes
DIGIT01: process(reset,clock)
 begin
    if (reset = ACTIVE) then
        digitOneBits <= (others => '0');
    elsif (rising_edge(clock))then
          if (digitOneEn = ACTIVE) then
            digitOneBits <= modeAndNum(3 downto 0);
          end if;
  end if;
 end process DIGIT01;

-- Holds the second digit --
DIGIT02: process(reset,clock)
 begin
    if (reset = ACTIVE) then
        digitTwoBits <= (others => '0');
    elsif (rising_edge(clock))then
          if (digitTwoEn = ACTIVE) then
            digitTwoBits <= modeAndNum(3 downto 0);
          end if;
```

```vhdl
  end if;
 end process DIGIT02;

 -- Holds the third digit --
DIGIT03: process(reset,clock)
 begin
    if (reset = ACTIVE) then
        digitThreeBits <= (others => '0');
    elsif (rising_edge(clock))then
          if (digitThreeEn = ACTIVE) then
            digitThreeBits <= modeAndNum(3 downto 0);
          end if;
  end if;
 end process DIGIT03;
 -- Holds the fourth digit --
DIGIT04: process(reset,clock)
 begin
    if (reset = ACTIVE) then
        digitFourBits <= (others => '0');
    elsif (rising_edge(clock))then
          if (digitFourEn = ACTIVE) then
            digitFourBits <= modeAndNum(3 downto 0);
          end if;
  end if;
end process DIGIT04;

-- Converts the Digit 1 and Digit 2 values
-- to integers
LED_CONVERT_INT: process(digitOneBits, digitTwoBits)
variable tensDigit: integer range 0 to 9;
variable onesDigit: integer range 0 to 9;
begin
    tensDigit := (to_integer(unsigned(digitOneBits)) * 10);
    onesDigit := to_integer(unsigned(digitTwoBits));
    ledPos <= (tensDigit + onesDigit);
end process LED_CONVERT_INT;

-- Drives the LEDs depending on if they turn off or turn on --
LED_DRIVER: process (reset,clock)
begin
    if (reset = ACTIVE) then
        leds <= "0000000000000000";
    elsif (rising_edge(clock)) then
        if (ledOnEn = ACTIVE) then
            leds(ledPos) <= ACTIVE;
        elsif (ledOffEn = ACTIVE) then
            leds(ledPos) <= not ACTIVE;
        end if;
    end if;
end process LED_DRIVER;

DISPLAY_DRIVER:  SevenSegmentDriver port map(
                           reset        => reset,
                           clock        => clock,
                           digit3       => digitOneBits,
                           digit2       => digitTwoBits,
                           digit1       => digitThreeBits,
```

```
                                digit0         => digitFourBits,
                                blank3         => not ACTIVE,
                                blank2         => not ACTIVE,
                                blank1         => not ACTIVE,
                                blank0         => not ACTIVE,
                                sevenSegs      => segments,
                                anodes         => anodes);
end SerialControl_ARCH;
```

```
--------------------------------------------------------------------------------
-----
-- Group Members; Clarence Barron, Rodrigo Corral, Paul Hughes

-- This is the test bench to test to see if the Serial Control component
-- works with the design using the LEDs and the Displays.
-- In order to work the test bench, comment out either the LED
-- Process or the Display Process then change the middle 8 bits.
--------------------------------------------------------------------------------
-----


library ieee;
use ieee.std_logic_1164.all;

entity SerialControl_tb is
end SerialControl_tb;

architecture SerialControl_tb_ARCH of SerialControl_tb is
    component SerialControl
    Port ( reset : in std_logic;
           clock : in std_logic;
           bytesIn : in std_logic;
           segments : out std_logic_vector (6 downto 0);
           anodes : out std_logic_vector (3 downto 0);
           leds : out std_logic_vector (15 downto 0));
    end component SerialControl;
signal reset:    std_logic;
signal clock:    std_logic;
signal bytesIn:  std_logic;
signal segments: std_logic_vector (6 downto 0);
signal anodes:   std_logic_vector (3 downto 0);
signal leds:     std_logic_vector (15 downto 0);
constant ACTIVE:         std_logic := '1';
constant BAUD_RATE:   integer :=  115_200;
constant BIT_TIME:    time    := (1_000_000_000/BAUD_RATE) * 1 ns;
constant CLOCK_FREQ:  integer := 100_000_000;
begin
UUT: SerialControl port map(
            reset => reset,
            clock => clock,
            bytesIn => bytesIn,
            segments => segments,
            anodes => anodes,
            leds => leds);

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

 RESET_CLOCK: process
    begin
        reset <= ACTIVE;
```

```
        wait for  200ns;
        reset <= not ACTIVE;
        wait;
    end process;

-- Test Bench for the LEDS --
--  LED_TEST_DRIVER: process
--    begin
--        ----set serial line to idle---------------------------------------
-------
--        bytesIn <= '1';
--        wait for 316 ns;


--        ----transmit L------------------------------------------------------
----
--        bytesIn <= '0';
--        wait for BIT_TIME;
--        bytesIn <= '0';
--        wait for BIT_TIME;
--        bytesIn <= '0';
--        wait for BIT_TIME;
--        bytesIn <= '1';
--        wait for BIT_TIME;
--        bytesIn <= '1';
--        wait for BIT_TIME;
--        bytesIn <= '0';
--        wait for BIT_TIME;
--        bytesIn <= '0';
--        wait for BIT_TIME;
--        bytesIn <= '1';
--        wait for BIT_TIME;
--        bytesIn <= '0';
--        wait for BIT_TIME;
--        bytesIn <= '1';
--        wait for BIT_TIME;


--        ----transmit 1 for Tens place -------------------------------------
--------------------
--        bytesIn <= '0';
--        wait for BIT_TIME;
--        bytesIn <= '1';
--        wait for BIT_TIME;
--        bytesIn <= '0';
--        wait for BIT_TIME;
--        bytesIn <= '0';
--        wait for BIT_TIME;
--        bytesIn <= '0';
--        wait for BIT_TIME;
--        bytesIn <= '1';
--        wait for BIT_TIME;
--        bytesIn <= '1';
--        wait for BIT_TIME;
--        bytesIn <= '0';
--        wait for BIT_TIME;
--        bytesIn <= '0';
```

```
--        wait for BIT_TIME;
--        bytesIn <= '1';
--        wait for BIT_TIME;


--        ----transmit 5 for Ones place -------------------------------------
--------------------
--        bytesIn <= '0';
--        wait for BIT_TIME;
--        bytesIn <= '1';
--        wait for BIT_TIME;
--        bytesIn <= '0';
--        wait for BIT_TIME;
--        bytesIn <= '1';
--        wait for BIT_TIME;
--        bytesIn <= '0';
--        wait for BIT_TIME;
--        bytesIn <= '1';
--        wait for BIT_TIME;
--        bytesIn <= '1';
--        wait for BIT_TIME;
--        bytesIn <= '0';
--        wait for BIT_TIME;
--        bytesIn <= '0';
--        wait for BIT_TIME;
--        bytesIn <= '1';
--        wait for BIT_TIME;

--        wait;
--    end process LED_TEST_DRIVER;

DISPLAY_TEST: process
 begin
        ----set serial line to idle------------------------------------------
-----
        bytesIn <= '1';
        wait for 316 ns;


        ----transmit 'D'-----------------------------------------------------
----
        bytesIn <= '0';
        wait for BIT_TIME;
        bytesIn <= '0';
        wait for BIT_TIME;
        bytesIn <= '0';
        wait for BIT_TIME;
        bytesIn <= '1';
        wait for BIT_TIME;
        bytesIn <= '0';
        wait for BIT_TIME;
        bytesIn <= '0';
        wait for BIT_TIME;
        bytesIn <= '0';
        wait for BIT_TIME;
        bytesIn <= '1';
        wait for BIT_TIME;
```

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


        ----transmit 1 for Tens place ---------------------------------------
------------------
        bytesIn <= '0';
        wait for BIT_TIME;
        bytesIn <= '1';
        wait for BIT_TIME;
        bytesIn <= '0';
        wait for BIT_TIME;
        bytesIn <= '0';
        wait for BIT_TIME;
        bytesIn <= '0';
        wait for BIT_TIME;
        bytesIn <= '1';
        wait for BIT_TIME;
        bytesIn <= '1';
        wait for BIT_TIME;
        bytesIn <= '0';
        wait for BIT_TIME;
        bytesIn <= '0';
        wait for BIT_TIME;
        bytesIn <= '1';
        wait for BIT_TIME;


        ----transmit 5 for Ones place ---------------------------------------
------------------
        bytesIn <= '0';
        wait for BIT_TIME;
        bytesIn <= '1';
        wait for BIT_TIME;
        bytesIn <= '0';
        wait for BIT_TIME;
        bytesIn <= '1';
        wait for BIT_TIME;
        bytesIn <= '0';
        wait for BIT_TIME;
        bytesIn <= '1';
        wait for BIT_TIME;
        bytesIn <= '1';
        wait for BIT_TIME;
        bytesIn <= '0';
        wait for BIT_TIME;
        bytesIn <= '0';
        wait for BIT_TIME;
        bytesIn <= '1';
        wait for BIT_TIME;

          ----transmit 8 for Ones place -------------------------------------
--------------------
        bytesIn <= '0';
        wait for BIT_TIME;
```

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

          ----transmit 2 for Ones place -------------------------------------
--------------------
        bytesIn <= '0';
        wait for BIT_TIME;
        bytesIn <= '0';
        wait for BIT_TIME;
        bytesIn <= '1';
        wait for BIT_TIME;
        bytesIn <= '0';
        wait for BIT_TIME;
        bytesIn <= '0';
        wait for BIT_TIME;
        bytesIn <= '1';
        wait for BIT_TIME;
        bytesIn <= '1';
        wait for BIT_TIME;
        bytesIn <= '0';
        wait for BIT_TIME;
        bytesIn <= '0';
        wait for BIT_TIME;
        bytesIn <= '1';
        wait for BIT_TIME;
end process DISPLAY_TEST;

end SerialControl_tb_ARCH;
```