Group Report

Abel Haris Harsono (3035834345)
Filbert David Telajaksana (3035945699)
Aryaman Bhardwaj (3035858755)
Syed Nafis Ishrak (3035946083)

ROLES

Group Member	Role for Pre-extension	Role for Extension
Abel Haris Harsono	FPGA implementation: Worked on implementing VHDL to FPGA by debugging and redesigning components	• Contributed to idea of data transfer with extended table: Keeping the BOS and EOS sequence for audio wave generation • Wrote the code for extension: modified mcdecoder with extra states and outputs
	Integrated symb_det, myuart, mcdecoder	
	Final designer of symb_det	
	Designer of • Myuart: All part of myuart • mcdecoder: partially did the next state logic, and all of storing inputs processes and output logic	
Aryaman Bhardwaj		
Filbert David Telajaksana		
Syed Nafis Ishrak		

DESCRIPTION

The system follows as what the course has described with the 3 main components (symb_det, mucodec, myuart) without the FIFO and dpop.

SYMB_DET

This module consists of 3 clocks named ctr1, ctr2, ctr3. ctr1 counts for 1/16 s and reset the other counters, ctr2 counts how many clock cycles within a single input cycle, and ctr3 counts how many zero crosses have happened respectively.

The module also has a delay line with 2 registers to compare 2 different input values for checking the zero crossing.

Symb_det will only start operation when it's no longer quiet. We do this by checking the magnitude of the input value. If greater than some predefined value, it will keep on listening and calculating.

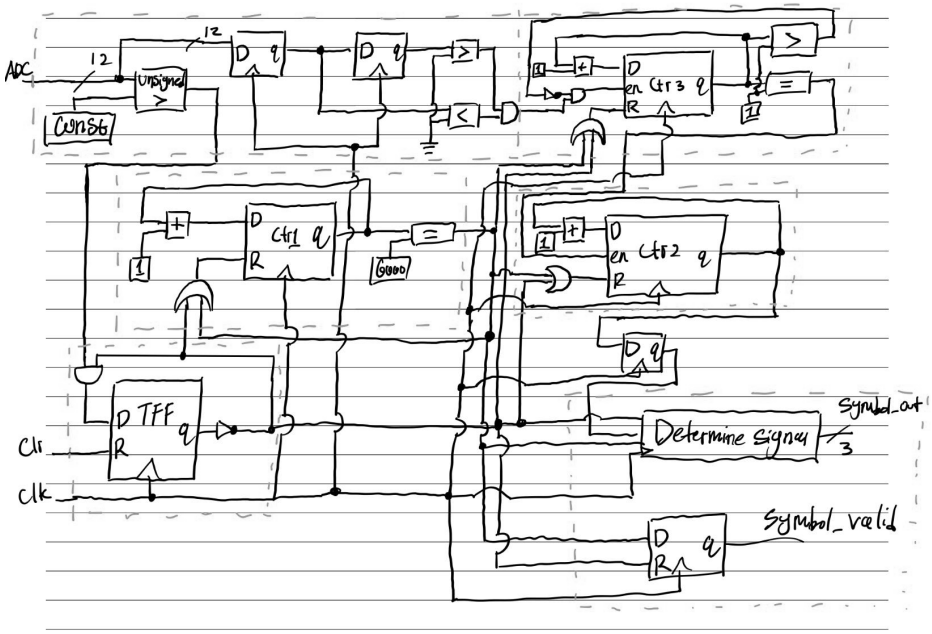


Diagram (The code is based on this)

MCDECODER

This module is simply an FSM, consisting of the following states:

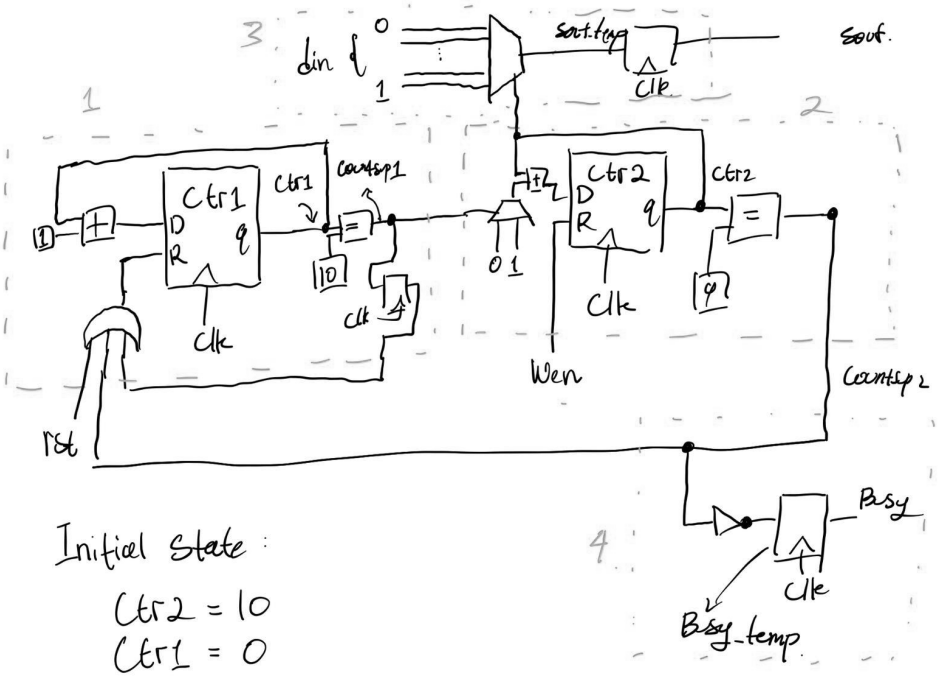
- St_RESET : Starting state and also the state which will be gone back to after clear is asserted. This state will also be inputting "000" which is the starting bit. If input is anything else, it will go to St_ERROR
- St_ERROR : Error state. This will be asserting a symbol invalid signal, and will go back to St_RESET after a clock cycle.

- LISTENING : State will also assert symbol invalid signal, but will be waiting for an input of 0-6. Afterwards, it will go to the specific L0-6 state, or if it receives ‘7’ it will go E1, otherwise it will go to St_ERROR
- B1-3: States will accept din ‘7’, ‘0’, ‘7’ consecutively in that order, and will progress through each Bx state until B3, where it will move to LISTENING after a clock cycle, otherwise it will go to St_ERROR.
- E1-3: States will accept din ‘0’, ‘7’, ‘0’ consecutively in that order, and will progress through each Ex state until E3, where it will move to St_RESET after a clock cycle, otherwise it will go to St_ERROR.
- L0-6: State represents the first number out of the 2 to be decoded into a character. Each Lx represents the number x. For each Lx state, it will receive the second number for decoding except itself, and go back to LISTENING state, otherwise go to St_ERROR.

The component consists of 3 processes as is typical with FSM paradigms: clocking process for async clear and ensure that code runs on a rising edge, next state process to move and route through the states, and an output logic to assert dout, dvalid and error.

MYUART

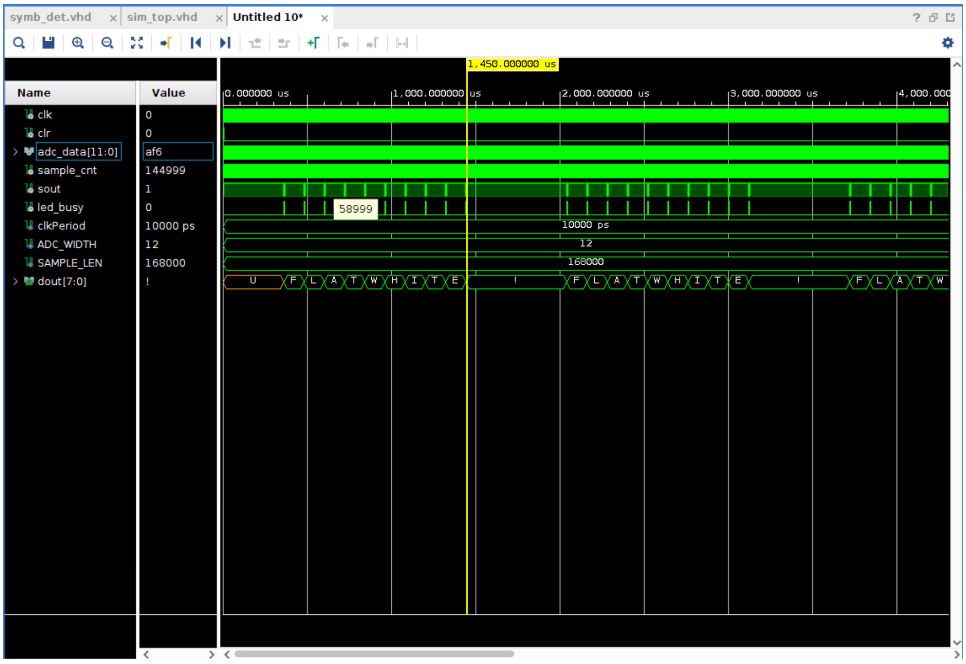
The main component of the UART comes down to 2 clocks, ctr1 and ctr2. ctr1 is used to make sure we can hold the output for 10 clock cycles and ctr2 is used to cycle through din with the start and end bit.

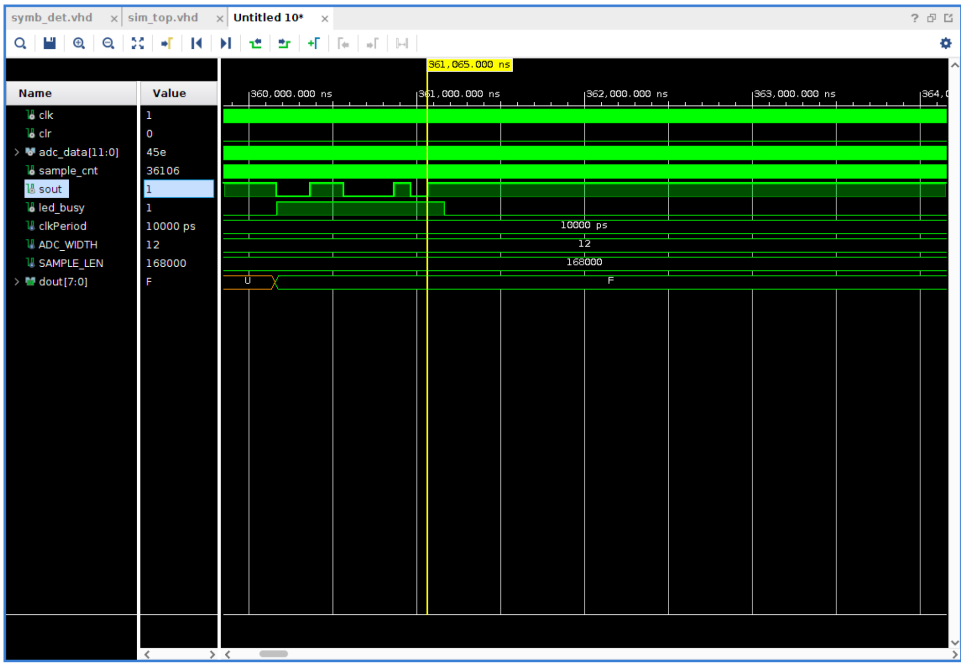


LEVEL OF COMPLETENESS

MILESTONE 1

We have successfully simulated our design on Vivado. Refer to RESULTS section for proof





MILESTONE 2

We have successfully simulated our design on Vivado. Refer to milestone2_video for proof.

MILESTONE 3

Ideas

Extend the code table so that it can also include numbers without adding extra frequencies to be generated

Changes implemented

To accommodate for the new encoding, we have decided

	0	**1**	**2**	**3**	**4**	**5**	**6**
0	NA	1	2	3	4	5	6
1	7	NA	B	D	H	L	R
2	8	A	NA	G	K	Q	V
3	9	C	F	NA	P	U	Z
4	0	E	J	O	NA	Y	.
5	[	I	N	T	X	NA	?
6	]	M	S	W	!	SPACE	NA

To match our finite state machine, we simply added the state L0, thus making it easily integrated into the system. In the previous design, during LISTENING state, if the input is 0, next state would be an error. We have changed this so that it transfers into L0 instead. We took into account the transition to ending bytes by not implementing the added encoding with sentinel value 7, since that will disrupt the transition into E1 state.

The new state will be transitioned into when 0 is received and can receive any number between 1 and 6 inclusive for output. If correct input is received, then it will simply output the value according to the table above.

Proof of this implementation's success can be found in milestone_3. The output text should be "THIS IS FLATWHITE NUMER 13!"

APPENDIX

SYMB_DET

```
-- symb_det
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE ieee.std_logic_unsigned.ALL;
USE IEEE.NUMERIC_STD.ALL;
ENTITY symb_det IS
    PORT (
        clk : IN STD_LOGIC;
        clr : IN STD_LOGIC;
        adc_data : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
        symbol_valid : OUT STD_LOGIC;
        symbol_out : OUT STD_LOGIC_VECTOR(2 DOWNTO 0)
    );
END symb_det;
ARCHITECTURE Behavioral OF symb_det IS
    SIGNAL countsup1, crossed, start_count, silent, start_scanning, toggle,
countsup3 : STD_LOGIC;
    SIGNAL ctr1, ctr2, ctr2_temp, ctr3 : unsigned(12 DOWNTO 0);
    SIGNAL val, next_val : STD_LOGIC_VECTOR(11 DOWNTO 0);
    SIGNAL symb_out_temp : STD_LOGIC_VECTOR(2 DOWNTO 0);
    SIGNAL symb_valid_temp : STD_LOGIC;
    SIGNAL adc_data_temp : STD_LOGIC_VECTOR(11 DOWNTO 0);
BEGIN
    adc_data_temp <= adc_data - "100000000000";
    --Processes for checking whether ADC data is big enough to start the circuit
(A.K.A. The setup)

    --Checks if adc_data pass a certain threshold
    PROCESS (adc_data)BEGIN
        IF unsigned(adc_data_temp) > 4 THEN
            start_scanning <= '1';
        ELSE
            start_scanning <= '0';
        END IF;
    END PROCESS;
    toggle <= start_scanning AND silent;

    --TFF for when adc_data does become big
    PROCESS (toggle, clk, clr)BEGIN
        IF clr = '1' THEN
            silent <= '1';
        ELSIF rising_edge(clk) THEN
            IF toggle = '1' THEN
                silent <= NOT silent;
            END IF;
        END IF;
```

```
END PROCESS;

--counter1 processes
--The counter
PROCESS (clk, silent, countsup1)BEGIN
    IF silent = '1' OR countsup1 = '1' THEN
        ctr1 <= (OTHERS => '0');
    ELSIF rising_edge(clk) THEN
        ctr1 <= ctr1 + 1;
    END IF;
END PROCESS;

--countsup1
PROCESS (ctr1, clk)BEGIN
    IF rising_edge(clk) THEN
        IF ctr1 = 6000 THEN
            countsup1 <= '1';
        ELSE
            countsup1 <= '0';
        END IF;
    END IF;
END PROCESS;

--crossing processes
--Delay line
PROCESS (clk, adc_data_temp)BEGIN
    IF rising_edge(clk) THEN
        val <= next_val;
        next_val <= adc_data_temp;
    END IF;
END PROCESS;

--cross checking
crossed <= '1' WHEN(val(11) = '0' AND next_val(11) = '1') ELSE
    '0';

--ctr3 process
PROCESS (clk, crossed, countsup3, countsup1, silent)BEGIN
    IF countsup1 = '1' OR silent = '1' THEN
        ctr3 <= (OTHERS => '0');
    ELSIF rising_edge(clk) THEN
        IF countsup3 = '0' AND crossed = '1' THEN
            ctr3 <= ctr3 + 1;
        ELSE
            ctr3 <= ctr3 + 0;
        END IF;
    END IF;
END PROCESS;

countsup3 <= '1' WHEN(ctr3 > 1) ELSE
    '0';
start_count <= '1' WHEN(ctr3 = 1) ELSE
```

```
    '0';

--ctr2 processes
--ctr2 process
PROCESS (clk, countsup1, silent, start_count)BEGIN
    IF (countsup1 = '1' OR silent = '1') THEN
        ctr2 <= (OTHERS => '0');
    ELSIF rising_edge(clk) THEN
        IF start_count = '1' THEN
            ctr2 <= ctr2 + 1;
        ELSE
            ctr2 <= ctr2 + 0;
        END IF;
    END IF;
END PROCESS;

--ctr2_temp process (for output)
PROCESS (clk)BEGIN
    IF rising_edge(clk) THEN
        ctr2_temp <= ctr2;
    END IF;
END PROCESS;

--output processes
--symbol_out process
PROCESS (ctr2_temp, countsup1)BEGIN
    IF countsup1 = '1' THEN
        IF ctr2_temp >= 165 AND ctr2 <= 203 THEN
            symb_out_temp <= "111";
        ELSIF ctr2_temp < 165 AND ctr2_temp >= 135 THEN
            symb_out_temp <= "110";
        ELSIF ctr2_temp < 135 AND ctr2_temp >= 111 THEN
            symb_out_temp <= "101";
        ELSIF ctr2_temp < 111 AND ctr2_temp >= 90 THEN
            symb_out_temp <= "100";
        ELSIF ctr2_temp < 90 AND ctr2_temp >= 76 THEN
            symb_out_temp <= "011";
        ELSIF ctr2_temp < 76 AND ctr2_temp >= 62 THEN
            symb_out_temp <= "010";
        ELSIF ctr2_temp < 62 AND ctr2_temp >= 51 THEN
            symb_out_temp <= "001";
        ELSIF ctr2_temp < 51 AND ctr2_temp >= 41 THEN
            symb_out_temp <= "000";
        END IF;
    END IF;
END PROCESS;

PROCESS (symb_out_temp, clk, silent)BEGIN
    IF silent = '1' THEN
        symbol_out <= "000";
    ELSIF rising_edge(clk) THEN
        IF countsup1 = '1' THEN
```

```
                symbol_out <= symb_out_temp;
            END IF;
        END IF;
    END PROCESS;

    --symbol_valid processes
    PROCESS (countsup1, clk, silent)BEGIN
        IF silent = '1' THEN
            symbol_valid <= '0';
        ELSIF rising_edge(clk) THEN
            symbol_valid <= countsup1;
        END IF;
    END PROCESS;
END Behavioral;
```

MCDECODER

```
--mcdecoder
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;


ENTITY mcdecoder IS
    PORT (
        din : IN STD_LOGIC_VECTOR(2 DOWNTO 0);
        valid : IN STD_LOGIC;
        clr : IN STD_LOGIC;
        clk : IN STD_LOGIC;
        dout : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
        dvalid : OUT STD_LOGIC;
        error : OUT STD_LOGIC);
END mcdecoder;


ARCHITECTURE Behavioral OF mcdecoder IS
    TYPE state_type IS (St_RESET, St_ERROR, LISTENING,
                  B1, B2, B3,
                  E1, E2, E3,
                  L1, L2, L3, L4, L5, L6);
    SIGNAL state, next_state : state_type := St_RESET;
    SIGNAL temp_dout : STD_LOGIC_VECTOR (7 DOWNTO 0);
    SIGNAL temp_dvalid, temp_error : STD_LOGIC;
BEGIN
    sync_process : PROCESS (clk, clr)
    BEGIN
        IF clr = '1' THEN
            -- Your code here
            state <= St_RESET;
            dvalid <= '0';
        ELSIF rising_edge(clk) THEN
            -- Put your code here
            state <= next_state;
            dout <= temp_dout;
            dvalid <= temp_dvalid;
        END IF;
    END PROCESS;


    --next state logic
    state_logic : PROCESS (state, din, valid)
    BEGIN
        -- Complete the following:
        next_state <= state;
        IF valid = '1' THEN
            CASE (state) IS
                WHEN St_RESET =>
                    IF din = "000" THEN
```

```
            next_state <= B1;
        ELSE
            next_state <= St_ERROR;
        END IF;
    WHEN B1 =>
        IF din = "111" THEN
            next_state <= B2;
        ELSE
            next_state <= St_ERROR;
        END IF;
    WHEN B2 =>
        IF din = "000" THEN
            next_state <= B3;
        ELSE
            next_state <= St_ERROR;
        END IF;
    WHEN B3 =>
        IF din = "111" THEN
            next_state <= LISTENING;
        ELSE
            next_state <= St_ERROR;
        END IF;
    WHEN LISTENING =>
        CASE din IS
            WHEN "111" =>
                next_state <= E1;
            WHEN "001" =>
                next_state <= L1;
            WHEN "010" =>
                next_state <= L2;
            WHEN "011" =>
                next_state <= L3;
            WHEN "100" =>
                next_state <= L4;
            WHEN "101" =>
                next_state <= L5;
            WHEN "110" =>
                next_state <= L6;
            WHEN OTHERS =>
                next_state <= St_ERROR;
        END CASE;
    WHEN L1 =>
        CASE din IS
            WHEN "010" =>
                next_state <= LISTENING;
            WHEN "011" =>
                next_state <= LISTENING;
            WHEN "100" =>
                next_state <= LISTENING;
            WHEN "101" =>
                next_state <= LISTENING;
            WHEN "110" =>
```

```
                next_state <= LISTENING;
            WHEN OTHERS =>
                next_state <= St_ERROR;
        END CASE;
    WHEN L2 =>
        CASE din IS
            WHEN "001" =>
                next_state <= LISTENING;
            WHEN "011" =>
                next_state <= LISTENING;
            WHEN "100" =>
                next_state <= LISTENING;
            WHEN "101" =>
                next_state <= LISTENING;
            WHEN "110" =>
                next_state <= LISTENING;
            WHEN OTHERS =>
                next_state <= St_ERROR;
        END CASE;
    WHEN L3 =>
        CASE din IS
            WHEN "001" =>
                next_state <= LISTENING;
            WHEN "010" =>
                next_state <= LISTENING;
            WHEN "100" =>
                next_state <= LISTENING;
            WHEN "101" =>
                next_state <= LISTENING;
            WHEN "110" =>
                next_state <= LISTENING;
            WHEN OTHERS =>
                next_state <= St_ERROR;
        END CASE;
    WHEN L4 =>
        CASE din IS
            WHEN "001" =>
                next_state <= LISTENING;
            WHEN "010" =>
                next_state <= LISTENING;
            WHEN "011" =>
                next_state <= LISTENING;
            WHEN "101" =>
                next_state <= LISTENING;
            WHEN "110" =>
                next_state <= LISTENING;
            WHEN OTHERS =>
                next_state <= St_ERROR;
        END CASE;
    WHEN L5 =>
        CASE din IS
            WHEN "001" =>
```

```
                next_state <= LISTENING;
            WHEN "010" =>
                next_state <= LISTENING;
            WHEN "011" =>
                next_state <= LISTENING;
            WHEN "100" =>
                next_state <= LISTENING;
            WHEN "110" =>
                next_state <= LISTENING;
            WHEN OTHERS =>
                next_state <= St_ERROR;
        END CASE;
    WHEN L6 =>
        CASE din IS
            WHEN "001" =>
                next_state <= LISTENING;
            WHEN "010" =>
                next_state <= LISTENING;
            WHEN "011" =>
                next_state <= LISTENING;
            WHEN "100" =>
                next_state <= LISTENING;
            WHEN "101" =>
                next_state <= LISTENING;
            WHEN OTHERS =>
                next_state <= St_ERROR;
        END CASE;
    WHEN E1 =>
        CASE din IS
            WHEN "000" =>
                next_state <= E2;
            WHEN OTHERS =>
                next_state <= St_ERROR;
        END CASE;
    WHEN E2 =>
        CASE din IS
            WHEN "111" =>
                next_state <= E3;
            WHEN OTHERS =>
                next_state <= St_ERROR;
        END CASE;
    WHEN E3 =>
        CASE din IS
            WHEN "000" =>
                next_state <= St_RESET;
            WHEN OTHERS =>
                next_state <= St_ERROR;
        END CASE;
    WHEN St_ERROR =>
        next_state <= St_RESET;
    WHEN OTHERS =>
        next_state <= St_ERROR;
```

```
        END CASE;
    END IF;
END PROCESS;


--Output logic
output_logic : PROCESS (state, din, valid)
BEGIN
    IF valid = '1' THEN
        CASE state IS
            WHEN L1 =>
                CASE din IS
                    WHEN "010" =>
                        temp_dout <= "01000010";
                        temp_dvalid <= '1';
                    WHEN "011" =>
                        temp_dout <= "01000100";
                        temp_dvalid <= '1';
                    WHEN "100" =>
                        temp_dout <= "01001000";
                        temp_dvalid <= '1';
                    WHEN "101" =>
                        temp_dout <= "01001100";
                        temp_dvalid <= '1';
                    WHEN "110" =>
                        temp_dout <= "01010010";
                        temp_dvalid <= '1';
                    WHEN OTHERS =>
                END CASE;
            WHEN L2 =>
                CASE din IS
                    WHEN "001" =>
                        temp_dout <= "01000001";
                        temp_dvalid <= '1';
                    WHEN "011" =>
                        temp_dout <= "01000111";
                        temp_dvalid <= '1';
                    WHEN "100" =>
                        temp_dout <= "01001011";
                        temp_dvalid <= '1';
                    WHEN "101" =>
                        temp_dout <= "01010001";
                        temp_dvalid <= '1';
                    WHEN "110" =>
                        temp_dout <= "01010110";
                        temp_dvalid <= '1';
                    WHEN OTHERS =>
                END CASE;
            WHEN L3 =>
                CASE din IS
                    WHEN "001" =>
                        temp_dout <= "01000011";
```

```
                temp_dvalid <= '1';
            WHEN "010" =>
                temp_dout <= "01000110";
                temp_dvalid <= '1';
            WHEN "100" =>
                temp_dout <= "01010000";
                temp_dvalid <= '1';
            WHEN "101" =>
                temp_dout <= "01010101";
                temp_dvalid <= '1';
            WHEN "110" =>
                temp_dout <= "01011010";
                temp_dvalid <= '1';
            WHEN OTHERS =>
        END CASE;
    WHEN L4 =>
        CASE din IS
            WHEN "001" =>
                temp_dout <= "01000101";
                temp_dvalid <= '1';
            WHEN "010" =>
                temp_dout <= "01001010";
                temp_dvalid <= '1';
            WHEN "011" =>
                temp_dout <= "01001111";
                temp_dvalid <= '1';
            WHEN "101" =>
                temp_dout <= "01011001";
                temp_dvalid <= '1';
            WHEN "110" =>
                temp_dout <= "00101110";
                temp_dvalid <= '1';
            WHEN OTHERS =>
        END CASE;
    WHEN L5 =>
        CASE din IS
            WHEN "001" =>
                temp_dout <= "01001001";
                temp_dvalid <= '1';
            WHEN "010" =>
                temp_dout <= "01001110";
                temp_dvalid <= '1';
            WHEN "011" =>
                temp_dout <= "01010100";
                temp_dvalid <= '1';
            WHEN "100" =>
                temp_dout <= "01011000";
                temp_dvalid <= '1';
            WHEN "110" =>
                temp_dout <= "00111111";
                temp_dvalid <= '1';
            WHEN OTHERS =>
```

```
                END CASE;
            WHEN L6 =>
                CASE din IS
                    WHEN "001" =>
                        temp_dout <= "01001101";
                        temp_dvalid <= '1';
                    WHEN "010" =>
                        temp_dout <= "01010011";
                        temp_dvalid <= '1';
                    WHEN "011" =>
                        temp_dout <= "01010111";
                        temp_dvalid <= '1';
                    WHEN "100" =>
                        temp_dout <= "00100001";
                        temp_dvalid <= '1';
                    WHEN "101" =>
                        temp_dout <= "00100000";
                        temp_dvalid <= '1';
                    WHEN OTHERS =>
                END CASE;
            WHEN St_ERROR =>
                error <= '1';
            WHEN LISTENING =>
                temp_dvalid <= '0';
            WHEN OTHERS =>
                --                temp_dout<="00000000";
                error <= '0';
                temp_dvalid <= '0';
        END CASE;
    ELSE
        temp_dvalid <= '0';
        error <= '0';
    END IF;
END PROCESS;
END Behavioral;
```

MYUART

```
-- myuart
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.NUMERIC_STD.ALL;

ENTITY myuart IS
    PORT (
        din : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
        busy : OUT STD_LOGIC;
        wen : IN STD_LOGIC;
        sout : OUT STD_LOGIC;
        clr : IN STD_LOGIC;
        clk : IN STD_LOGIC);
END myuart;


ARCHITECTURE rtl OF myuart IS
    SIGNAL countsup1, countsup2, sout_temp, busy_temp : STD_LOGIC;
    SIGNAL ctr1, ctr2 : unsigned(7 DOWNTO 0);
    SIGNAL din_temp : STD_LOGIC_VECTOR(7 DOWNTO 0);
BEGIN
    PROCESS (clk, wen, countsup2)BEGIN
        IF countsup2 = '1' THEN
            din_temp <= "00000000";
        ELSIF rising_edge(clk) THEN
            IF wen = '1' THEN
                din_temp <= din;
            END IF;
        END IF;
    END PROCESS;
    --counter 1
    PROCESS (clk, countsup2, countsup1, clr)BEGIN
        IF countsup2 = '1' OR countsup1 = '1' OR clr = '1' THEN
            ctr1 <= (OTHERS => '0');
        ELSIF rising_edge(clk) THEN
            ctr1 <= ctr1 + 1;
        END IF;
    END PROCESS;


    PROCESS (clk) BEGIN
        IF rising_edge(clk) THEN
            IF ctr1 = 8 THEN
                countsup1 <= '1';
            ELSE
                countsup1 <= '0';
            END IF;
        END IF;
    END PROCESS;
```

```
    --counter 2
    PROCESS (clk, wen, countsup1, clr)BEGIN
        IF clr = '1' THEN
            ctr2 <= to_unsigned(10, 8);
        ELSIF wen = '1' THEN
            ctr2 <= (OTHERS => '0');
        ELSIF rising_edge(clk) THEN
            CASE countsup1 IS
                WHEN '0' =>
                    ctr2 <= ctr2 + 0;
                WHEN OTHERS =>
                    ctr2 <= ctr2 + 1;
            END CASE;
        END IF;
    END PROCESS;
    countsup2 <= '1' WHEN (ctr2 = 10) ELSE '0';


    --sout processes
    PROCESS (ctr2)BEGIN
        IF ctr2 = 0 THEN
            sout_temp <= '0';
        ELSIF ctr2 = 9 OR ctr2 = 10 THEN
            sout_temp <= '1';
        ELSIF ctr2 < 9 THEN
            sout_temp <= din_temp(to_integer(ctr2) - 1);
        END IF;
    END PROCESS;


    PROCESS (clk, clr) BEGIN
        IF clr = '1' THEN
            sout <= '1';
        ELSIF rising_edge(clk) THEN
            sout <= sout_temp;
        END IF;
    END PROCESS;


    --busy processes
    PROCESS (clk, clr)BEGIN
        IF clr = '1' THEN
            busy <= '0';
        ELSIF rising_edge(clk) THEN
            busy <= NOT countsup2;
        END IF;
    END PROCESS;


END rtl;
```