Tomato Vending Machine Project -Rotten Tomato-Matic 9000

Owner	John Akujobi
Created time	@October 2, 2023 9:41 AM
Status	Not started

https://github.com/jakujobi/PR1_Vending_Machine

Team Members

- John Akujobi
- LNU Sukhman Singh
- Kayleigh Humphrey Valle

▼ Intro Video

https://prod-files-secure.s3.us-west-2.amazonaws.com/c7353ea1-17ee-41cd-96a3-a4c0c93c34c1/641f186f-4b72-473b-9671-4b487382200a/Introducing_the_Rotten_V2.mp4

▼ Overview

Introducing the Rotten Tomato-Matic 9000 - the world's first digital vending machine for rotten tomatoes! Built with SystemVerilog on a DE10-Lite board, this project is a fun demonstration of using hardware description languages to create interactive digital devices.

The user interface includes:

- Coin slots for farthings, ha'pennies, and pennies, using switches for inputs
- A 7-segment display that shows the total amount inserted
- An LED that lights up when a rotten tomato is dispensed
- LEDs indicating any change being returned in farthings and ha'pennies

Inside, a state machine tracks the running total as coins are inserted. Once enough has been inserted to purchase one of Willy's Wonky Rotten TomatoesTM, the tomato dispenser activates! Any excess coins are returned to the customer as change.

The SystemVerilog code defines modules for each component, from coin recognition to the state machine logic. The top-level module ties everything together into one cohesive rotten tomato-vending experience!

▼ Inputs

Signal	DE10-Lite I/O	Pin Assignment
Farthing (0.25d) - F	SW 0	PIN_C10
Half-Penny (0.5d) - H	SW 1	PIN_C11
Penny (1d) - P	SW 2	PIN_D12
CLK Input	KEY0	PIN_B8
RES Input	KEY1	PIN_A7

1. **Farthing (0.25d)** :
   - Lowest value input to the vending machine.
2. **Ha'penny (0.5d)** :
   - Another coin input to the vending machine.
3. **Penny (1d)** :
   - The highest value coin input to the vending machine.
4. **Raw Clock Input (CLK)** :
   - This is the clock signal for the state machine.
   - It will be passed through debouncer.sv
   - Connected to KEY0
   - Connects into CLK_deb
5. **Reset Input (RES)** :
   - This is an asynchronous reset signal that can reset the total back to zero.

▼ Outputs

Signal	DE10-Lite I/O	Pin Assignment
Current State Output	LEDR [S-1:0]	PIN_
Least Significant Binary	LEDR[0]	PIN_A8
	LEDR[1]	PIN_A9
	LEDR[2]	PIN_A10
Most Significant Binary	LEDR[3]	PIN_B10
OUTPUTS		
Tomato Output	LEDR[9]	PIN_B11
Ha'penny Output - Change	LEDR[8]	PIN_A11
Farthing Output - Change	LEDR[7]	PIN_D14
(State) Farthings	HEX5	
a	HEX50	PIN_J20
b	HEX51	PIN_K20
c	HEX52	PIN_L18
d	HEX53	PIN_N18
e	HEX54	PIN_M20

f	HEX55	PIN_N19
g	HEX56	PIN_N20
(Change) Farthings	HEX0	
a	HEX00	PIN_C14
b	HEX01	PIN_E15
c	HEX02	PIN_C15
d	HEX03	PIN_C16
e	HEX04	PIN_E16
f	HEX05	PIN_D17
g	HEX06	PIN_C17

1. **Rotten Tomato Dispense Signal** :
   - This signal is activated when the vending machine dispenses a rotten tomato.
2. **Change Dispense Signals** :
   - These signals are activated when the vending machine dispenses change in the form of farthings and ha'pennies.
3. **Current State Display** :
   - This is a 7-segment display showing the current amount of money in the machine (in farthings).
4. **Change Display** :
   - This is another 7-segment display showing the amount of change dispensed by the machine (in dollars).
5. **State Indicator LEDs** :
   - These LEDs indicate the current state of the machine.

▼ Assumptions

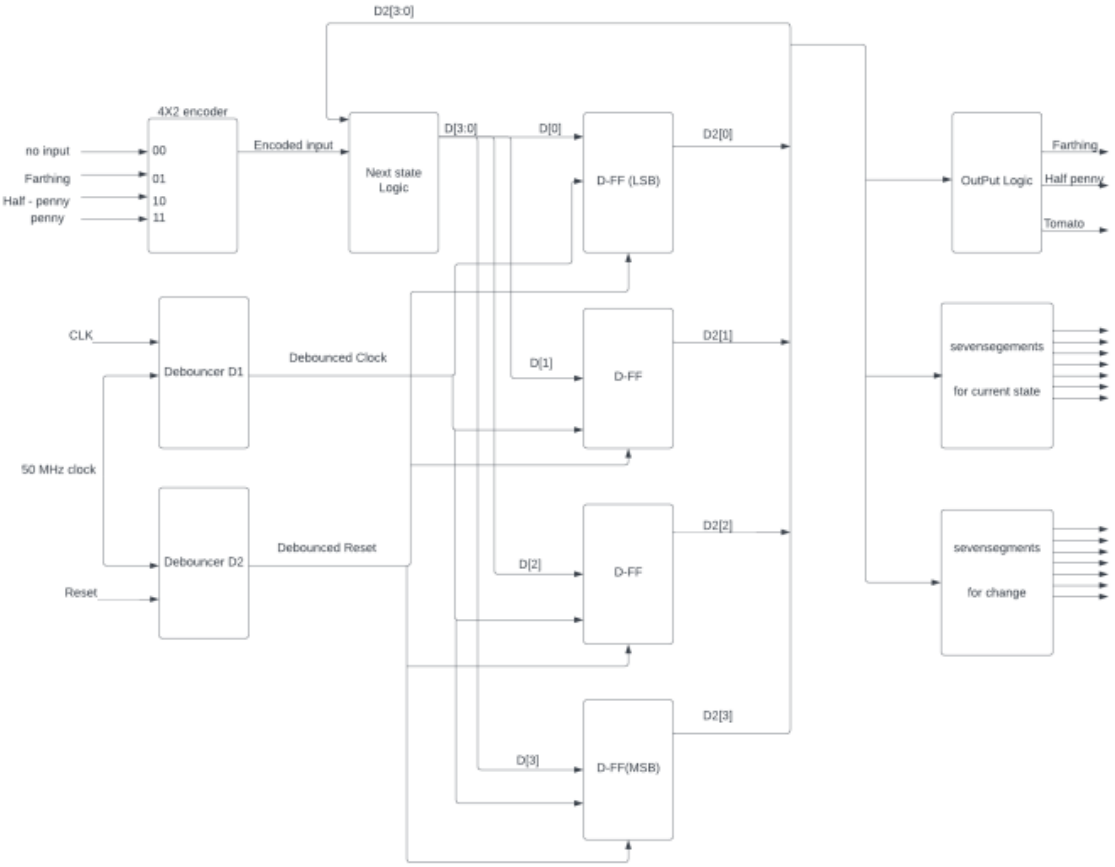
1. **Coin Input**: It's assumed that the coins are inserted one at a time and the machine can correctly identify each coin type (farthing, ha'penny, penny).
   That is, two coins are not inserted at the same time
2. **User Interaction**: It's assumed that users will wait for their change and the dispensed rotten tomato before inserting more coins for the next purchase.
3. **Reset Function**: It's assumed that the reset function, when activated, will successfully reset the machine to its initial state.
4. **Display Readability**: It's assumed that the 7-segment displays are clear and readable, allowing users to understand the current state of the machine and the amount of change. And that they work.
   For our DE-10 Lite boards, we don’t have to worry much about this
5. **Change Availability**: It's assumed that the machine always has enough farthings and ha'pennies to give change.
6. **Rotten Tomato Availability**: It's assumed that the machine is always stocked with rotten tomatoes to dispense. Though our DE-10 Lite boards don’t have tomatoes.
7. **Clock Signal**: It's assumed that the clock signal is stable and reliable for the operation of the state machine.
8. **Power Supply**: It's assumed that the machine has a continuous and stable power supply. That there are no blackouts
9. **Environment**: It's assumed that the machine is used in an environment that does not affect its operation (e.g., no extreme temperatures, humidity, etc.).

▼ Required States

- There are 9 states for the machine
- 4 D flipflops are needed to store the 4 bits of states

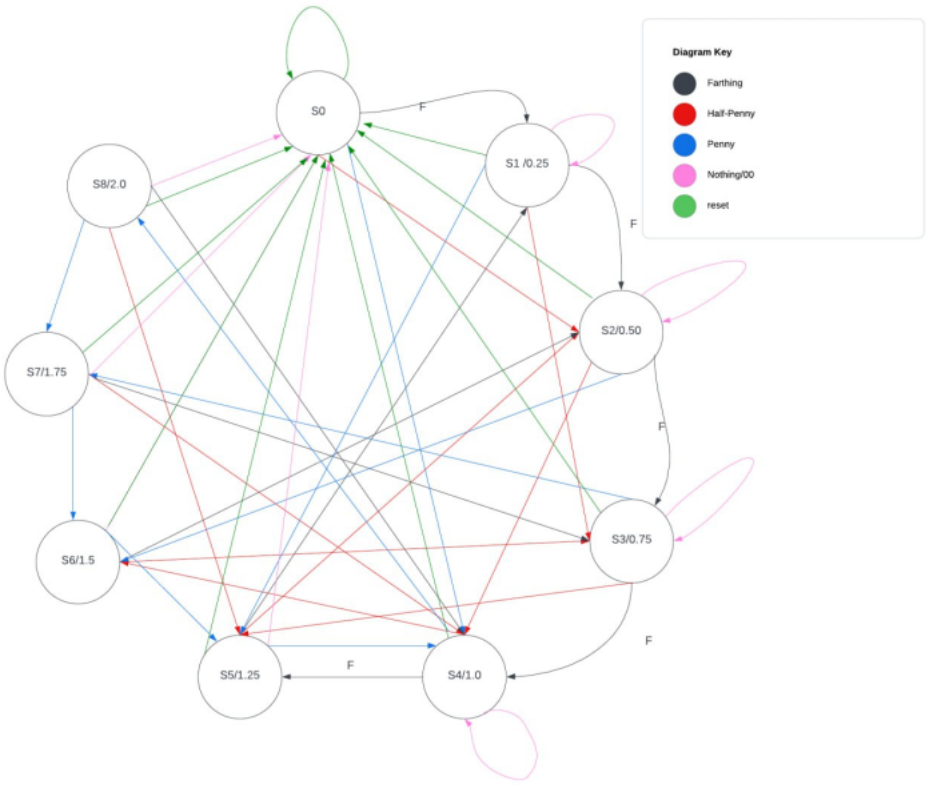
State	State	State Bit	Description
0	0.00	0000	Has no coins
1	0.25	0001	Received a farthing (0.25d)
2	0.5	0010	Total of 0.50 coins
3	0.75	0011	Total of 0.75
4	1.0	0100	Total of 1.00
5	1.25	0101	Total of 1.25
6	1.50	0110	Total of 1.50
7	1.75	0111	Total of 1.75
8	2.0	1000	Total of 2.00

▼ Top Level Design



block_digram.pdf

▼ State Transition Diagram



Blank_diagram.pdf

Blank_diagram.csv

▼ State Transition Table

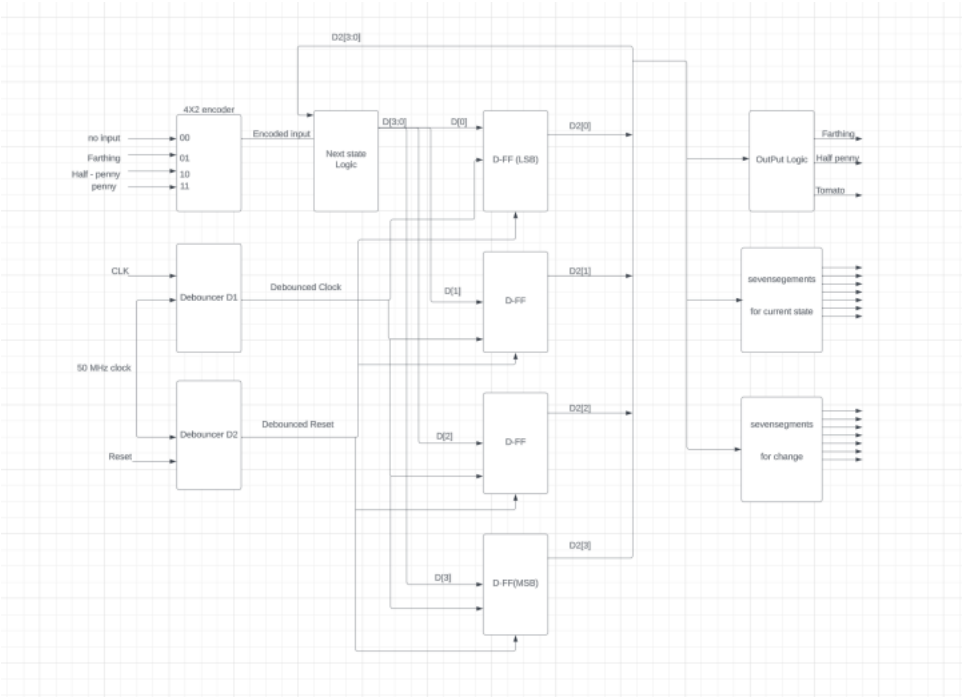
Current State	Encoded State	Inputs	Next State	Encoded Next State	Encoded inputs
S0	0 0 0 0	0 0 0	S0	0000	00
S0	0 0 0 0	0 0 1	S1	0001	01
S0	0 0 0 0	0 1 0	S2	0010	10
S0	0 0 0 0	0 1 1	x		
S0	0 0 0 0	1 0 0	S4	0100	11
S0	0 0 0 0	1 0 1	x		
S0	0 0 0 0	1 1 0	x		
S0	0 0 0 0	1 1 1	x		
S1	0 0 0 1	0 0 0	S1	0001	00
S1	0 0 0 1	0 0 1	S2	0010	01
S1	0 0 0 1	0 1 0	S3	0011	10
S1	0 0 0 1	0 1 1	x		

S1	0 0 0 1	1 0 0	S5	0101	11
S1	0 0 0 1	1 0 1	x		
S1	0 0 0 1	1 1 0	x		
S1	0 0 0 1	1 1 1	x		
S2	0 0 1 0	0 0 0	S2	0010	00
S2	0 0 1 0	0 0 1	S3	0011	01
S2	0 0 1 0	0 1 0	S4	0100	10
S2	0 0 1 0	0 1 1	x		
S2	0 0 1 0	1 0 0	S6	0110	11
S2	0 0 1 0	1 0 1	x		
S2	0 0 1 0	1 1 0	x		
S2	0 0 1 0	1 1 1	x		
S3	0 0 1 1	0 0 0	S3	0011	00
S3	0 0 1 1	0 0 1	S4	0100	01
S3	0 0 1 1	0 1 0	S5	0101	10
S3	0 0 1 1	0 1 1	x	x	
S3	0 0 1 1	1 0 0	S7	0111	11
S3	0 0 1 1	1 0 1	x	x	
S3	0 0 1 1	1 1 0	x	x	
S3	0 0 1 1	1 1 1	x	x	
S4	0 1 0 0	0 0 0	S4	0100	00
S4	0 1 0 0	0 0 1	S5	0101	01
S4	0 1 0 0	0 1 0	S6	0110	10
S4	0 1 0 0	0 1 1	x	x	
S4	0 1 0 0	1 0 0	S8	1000	11
S4	0 1 0 0	1 0 1	x	x	
S4	0 1 0 0	1 1 0	x	x	
S4	0 1 0 0	1 1 1	x	x	
S5	0 1 0 1	0 0 0	S0	0000	00
S5	0 1 0 1	0 0 1	S1	0001	01
S5	0 1 0 1	0 1 0	S2	0010	10
S5	0 1 0 1	0 1 1	x	x	
S5	0 1 0 1	1 0 0	S4	0100	11
S5	0 1 0 1	1 0 1	x	x	
S5	0 1 0 1	1 1 0	x	x	
S5	0 1 0 1	1 1 1	x	x	
S6	0 1 1 0	0 0 0	S0	0000	00
S6	0 1 1 0	0 0 1	S2	0010	01
S6	0 1 1 0	0 1 0	S3	0011	10
S6	0 1 1 0	0 1 1	x	x	
S6	0 1 1 0	1 0 0	S5	0101	11
S6	0 1 1 0	1 0 1	x	x	
S6	0 1 1 0	1 1 0	x	x	
S6	0 1 1 0	1 1 1	x	x	

S7	0 1 1 1	0 0 0	S0	0000	00
S7	0 1 1 1	0 0 1	S3	0011	01
S7	0 1 1 1	0 1 0	S4	0100	10
S7	0 1 1 1	0 1 1	x	x	
S7	0 1 1 1	1 0 0	S6	0110	11
S7	0 1 1 1	1 0 1	x	x	
S7	0 1 1 1	1 1 0	x	x	
S7	0 1 1 1	1 1 1	x	x	
S8	1 0 0 0	0 0 0	S0	0000	00
S8	1 0 0 0	0 0 1	S4	0100	01
S8	1 0 0 0	0 1 0	S5	0101	10
S8	1 0 0 0	0 1 1	x	x	
S8	1 0 0 0	1 0 0	S7	0111	11
S8	1 0 0 0	1 0 1	x	x	
S8	1 0 0 0	1 1 0	x	x	
S8	1 0 0 0	1 1 1	x	x	

▼ System Verilog Modules

▼ Vending Machine



Filename

- vendingmachine.sv

Description

- This module implements the top level module of a vending machine that accepts coins and dispenses tomatoes and change.

Inputs

- coin[2:0]: 3-bit input representing the coins inserted into the machine. coin[0] represents pennies, coin[1] represents ha'pennies, and coin[2] represents farthings.
- clk: clock signal from the button
- clk50m: 50 MHz internal clock
- res: reset button

Outputs

- tom: output signal to the tomato LED
- [6:0]: 7-bit output signal to the HEX display showing the state of the machine
- c[6:0]: 7-bit output signal to the HEX display showing the change
- outled[3:0]: 4-bit output signal to the LEDs showing the state of the machine
- HapCH: output signal to the LED showing the change in ha'pennies
- FarthCH: output signal to the LED showing the change in farthings

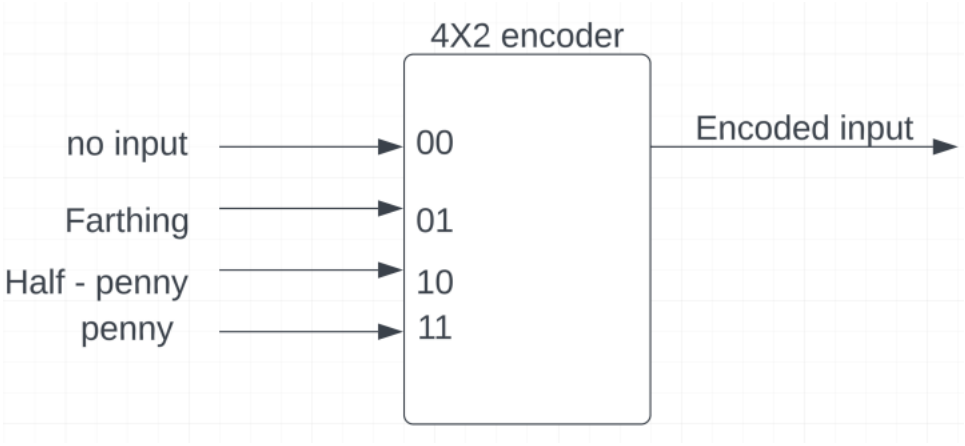
Code

```
// Authors: Kayleigh Humphrey, LNU Sukhman Singh, John Akujobi
// Date:  Oct. 10, 2023
// Name:  Vending Machine
// File name: vendingmachine.sv

/**
 * This module implements the top level module of a vending machine that accepts coins and dispenses tomatoes.
 *
 * Inputs:
 * - coin[2:0]: 3-bit input representing the coins inserted into the machine. coin[0] represents pennies, coin[1] represents ha'penn
 * - clk: clock signal from the button
 * - clk50m: 50 MHz internal clock
 * - res: reset button
 *
 * Outputs:
 * - tom: output signal to the tomato LED
 * - s[6:0]: 7-bit output signal to the HEX display showing the state of the machine
 * - c[6:0]: 7-bit output signal to the HEX display showing the change
 * - outled[3:0]: 4-bit output signal to the LEDs showing the state of the machine
 * - HapCH: output signal to the LED showing the change in ha'pennies
 * - FarthCH: output signal to the LED showing the change in farthings
 */

module vendingmachine(
  input  logic coin[2:0], // coin inputs
             // coin[0] = Pennies, coin[1] = Ha'pennies & coin[2] = Farthings
  input  logic clk, // clock signal from button
  input  logic clk50m,  // 50 MHz internal clock
  input  logic res,   // reset button

  output logic tom ,  // to tomato LED
  output logic s[6:0] , // to HEX showing the state of the machine
  output logic c[6:0],  // to HEX showing the change
  output logic outled[3:0], // to LEDs showing the state of the machine
  output logic HapCH, FarthCH // LEDs showing change in Farthings and Ha'pennies
);

logic enc_I[1:0]; // encoded 2 bit output
logic d[3:0]; // 4 bit output from the state logic to the flip flops
logic d2[3:0];  // 4 bit output from the flip flops to the tomato and outpuy logic

//!Debounce signals_____________________________________________
logic clkde , resde ; // debounced signals
// debounce clock signal
debouncer clk_db(
  .A_noisy(clk),
  .CLK50M(clk50m),
```

```
  .A(clkde)
);

// debounce reset signal
debouncer res_db(
  .A_noisy(res),
  .CLK50M(clk50m),
  .A(resde)
);


//!Encode the coin inputs_____________________________________________
// encode inputs
encoder (.y(coin), .out(enc_I));

// call nextstate logic
statelogic(
  .s (d2),
  .y (enc_I),
  .d (d)
);

//!Instantiate the flip flops ____________________________________
D_FF_neg zero (
  .D (d[0]),
  .CLKb (clkde),
  .RSTb (resde),
  .Q (d2[0])
);

D_FF_neg one (
  .D (d[1]),
  .CLKb (clkde),
  .RSTb (resde),
  .Q (d2[1])
);

D_FF_neg two (
  .D (d[2]),
  .CLKb (clkde),
  .RSTb (resde),
  .Q (d2[2])
);

D_FF_neg three(
  .D (d[3]),
  .CLKb (clkde),
  .RSTb (resde),
  .Q (d2[3])
);
//LED showing the state of the machine
assign outled = d2;

//!Tomato LED Module_____________________________________________
tomd416(
  .a0 (d2[0]) ,
  .a1 (d2[1]) ,
  .a2 (d2[2]) ,
  .a3 (d2[3]) ,
  .t (tom));

//!Output Logic_____________________________________________________
outputlogic(
  .A (d2),
  .curstate (s),
  .curchange (c),
  .far (FarthCH),
  .halfp (HapCH)
);

endmodule
```

.

▼ Encoder 4:2



Filename

- encoder.sv

Description

- The encoder module takes a 3-bit input and encodes it into a 2-bit output.

Inputs

- y - 3-bit input corresponding to Pennies, Ha'pennies & Farthings

Outputs

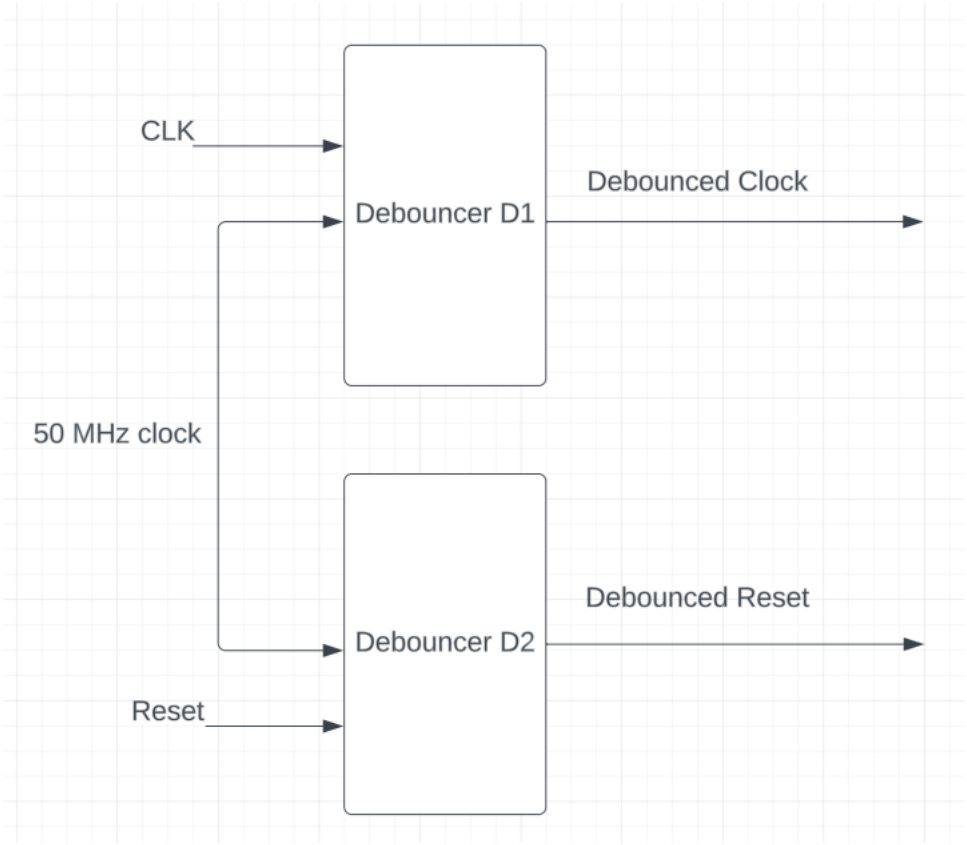
- out - 2-bit encoded output

Code

```
// Authors: Kayleigh Humphrey, LNU Sukhman Singh, John Akujobi
// Date:  October, Fall, 2023
// Name:    encoder
// Filename: encoder.sv

// The encoder module takes a 3-bit input and encodes it into a 2-bit output.
//    y - 3-bit input corresponding to Pennies, Ha'pennies & Farthings
//    out - 2-bit encoded output

module encoder(
  input logic y[2:0], //- 3-bit input corresponding to Pennies, Ha'pennies & Farthings
  output logic out[1:0]
);

assign out[0] = y[2] | y[0];
assign out[1] = y[2] | y[1];

endmodule
```

.

▼ Debouncer



Name

- debouncer

Filename

- debouncer.sv

Description

- We use this to clean up the signals from the clock and reset buttons

Inputs

- `A_noisy` - signal to be debounced
- `CLK50M` - 50 MHz clock from the DE10 board (PIN_P11)

Outputs

- `A` - debounced signal to be used in your circuit

Module Function Call

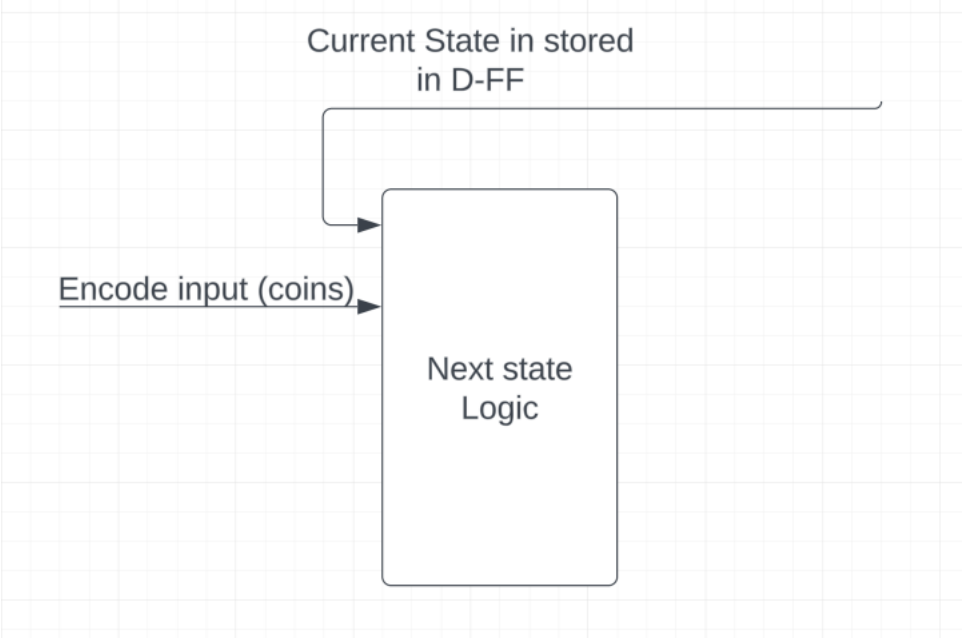
```
debouncer mod1 (
  .A_noisy(#), //noisy signal input
  .CLK50M(#), //internal clock of
  .A(#) //clean output signal
);
```

Module Code

```
/*
Author: Dr. Hansen
Date: Feb. 9, 2022
Description: takes a 1-bit noisy input and outputs a 1-bit clean signal

Inputs:
  A_noisy - signal to be debounced
  CLK50M  - 50 MHz clock from the DE10 board (PIN_P11)

Outputs:
  A       - debounced signal to be used in your circuit
*/


module debouncer(
  input logic A_noisy,
  input logic CLK50M,
  output logic A
);

  logic [15:0] COUNT;
  parameter [15:0] COMPARE = 50_000; //1 millisecond

  logic t_d, t_r, Anext;

  /*
  1 ms timer
  */
  always_ff@(posedge CLK50M)
  begin
    if(t_r)
      COUNT <= 16'd0;
    else
      COUNT <= COUNT + 16'd1;
  end
  assign t_d = (COUNT >= COMPARE);

  //next-state logic
  assign Anext = (A_noisy & t_d) | (A & ~t_d);

  //state
  always_ff@(posedge CLK50M)
    A <= Anext;

  //output logic
  assign t_r = t_d | (~A & ~A_noisy) | (A & A_noisy);

endmodule
```

.

▼ State Logic*



Filename

- statelogic.sv

Description

- The module `statelogic` implements the state logic of a vending machine.
- It takes in the current state `s` and the input `y` and outputs the next state `d`.
- It uses the Next State Logic equations obtained through K-maps from the State Transition Table.

Inputs

- s The current state of the vending machine represented as a 4-bit vector.
- y The input to the vending machine represented as a 2-bit vector.

Outputs

- d - The next state of the vending machine represented as a 4-bit vector.

Equations

```
d[0] = (
  ( y[0]  .   y'[1]   .   s'[0]   .   s'[1]   .   s'[2]   .   s'[3] ) +
  (  y'[0]  .   y'[1]   .    s[0]   .   s'[2]   .   s'[3] ) +
  (   y[1]  .    s[0]   .   s'[2]   .   s'[3] ) +
  ( y[0]  .   y'[1]   .   s'[0]   .   s[1]  .   s'[2]   .   s'[3] ) +
  ( y[0]  .   y'[1]   .   s'[1]   .   s[2]  .   s'[3] ) +
  ( y[0]  .   y'[1]   .    s[0]   .   s[1]  .   s[2]  .   s'[3] ) +
  ( y[1]  .   s'[0]   .    s[1]   .   s[2]  .   s'[3] ) +
  (   y[1]  .   s'[0]   .   s'[1]   .   s'[2]   .   s[3]  )
);

d[1] = (
  ( y[0]  .   y'[1]   .   s[0]  .   s'[1]   .   s'[2]   .   s'[3] ) +
  ( y'[0]   .   y[1]  .   s'[1]   .   s'[2]   .   s'[3] ) +
  ( y'[0]   .   y'[1]   .   s[1]  .   s'[2]   .   s'[3] ) +
  ( y'[1]   .   s'[0]   .   s[1]  .   s'[2]   .   s'[3] ) +
  ( y[0]  .   s'[0]   .   s[1]  .   s'[2]   .   s'[3] ) +
  ( y[0]  .   y[1]  .   s[1]  .   s'[2]   .   s'[3] ) +
  ( y[0]  .   y'[1]   .   s[1]  .   s[2]  .   s'[3] ) +
```

```
  ( y[0]  .   s[0]  .   s[1]  .   s[2]  .   s'[3] ) +
  ( y'[0]   .   y[1]  .   s'[1]   .   s[2]  .   s'[3] ) +
  ( y'[0]   .   y[1]  .   s'[0]   .   s[1]  .   s[2]  .   s'[3] ) +
  ( y[0]  .   y[1]  .   s'[0]   .   s'[1]   .   s'[2]   .   s[3]  )
);

d[2] = (
  (   y[0]  .   y[1]  .   s'[2]   .   s'[3]   .   s'[1] ) +
  (   y[0]  .   s[0]  .   s[1]  .   s'[2]   .   s'[3] ) +
  (   y[1]  .   s[1]  .   s'[2]   .   s'[3] ) +
  (   y'[1]   .   s'[0]   .   s'[1]   .   s[2]  .   s'[3] ) +
  (   y'[0]   .   s'[0]   .   s'[1]   .   s[2]  .   s'[3] ) +
  (   y[0]  .   y[1]  .   s[0]  .   s[2]  .   s'[3] ) +
  (  y[1]   .   s[0]  .   s[1]  .   s[2]  .   s'[3] ) +
  (   y[0]  .   y[1]  .   s[1]  .   s[2]  .   s'[3] ) +
  (   y[0]  .   s'[0]   .   s'[1]   .   s'[2]   .   s[3]  ) +
  (   y[1]  .   s'[0]   .   s'[1]   .   s'[2]   .   s[3]  )
);

assign d[3] = (
  ( y[0]  .   y[1]  .   s'[0]   .   s'[1]   .   s[2]  .   s'[3] )
);
```

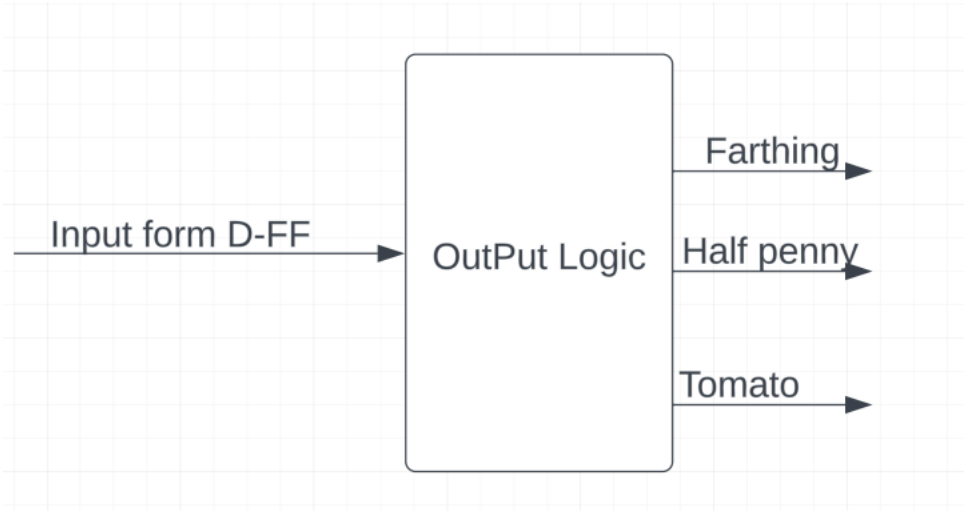
Code

```
// Authors: Kayleigh Humphrey, LNU Sukhman Singh, John Akujobi
// Date:  October, Fall, 2023
// Name:    statelogic
// Filename: statelogic.sv

/** Description
 *  The module `statelogic` implements the state logic of a vending machine.
 *  It takes in the current state `s` and the input `y` and outputs the next state `d`.
 *  It uses the Next State Logic equations obtained through K-maps from the State Transition Table.

 *
 *  s The current state of the vending machine represented as a 4-bit vector.
 *  y The input to the vending machine represented as a 2-bit vector.
 *  d The next state of the vending machine represented as a 4-bit vector.
 */
module statelogic(
  input logic s[3:0], //current state of the vending machine
  input logic y[1:0],   //encoded input to the vending machine
  output logic d[3:0]   //next state of the vending machine
);

assign d[0] = (
  ( y[0]  &   ~y[1]   &   ~s[0]   &   ~s[1]   &   ~s[2]   &   ~s[3] ) |
  (  ~y[0]  &   ~y[1]   &    s[0]   &   ~s[2]   &   ~s[3] ) |
  (   y[1]  &    s[0]   &   ~s[2]   &   ~s[3] ) |
  ( y[0]  &   ~y[1]   &   ~s[0]   &   s[1]  &   ~s[2]   &   ~s[3] ) |
  ( y[0]  &   ~y[1]   &   ~s[1]   &   s[2]  &   ~s[3] ) |
  ( y[0]  &   ~y[1]   &    s[0]   &   s[1]  &   s[2]  &   ~s[3] ) |
  ( y[1]  &   ~s[0]   &    s[1]   &   s[2]  &   ~s[3] ) |
  (   y[1]  &   ~s[0]   &   ~s[1]   &   ~s[2]   &   s[3]  )
);

assign d[1] = (
  ( y[0]  &   ~y[1]   &   s[0]  &   ~s[1]   &   ~s[2]   &   ~s[3] ) |
  ( ~y[0]   &   y[1]  &   ~s[1]   &   ~s[2]   &   ~s[3] ) |
  ( ~y[0]   &   ~y[1]   &   s[1]  &   ~s[2]   &   ~s[3] ) |
  ( ~y[1]   &   ~s[0]   &   s[1]  &   ~s[2]   &   ~s[3] ) |
  ( y[0]  &   ~s[0]   &   s[1]  &   ~s[2]   &   ~s[3] ) |
  ( y[0]  &   y[1]  &   s[1]  &   ~s[2]   &   ~s[3] ) |
  ( y[0]  &   ~y[1]   &   s[1]  &   s[2]  &   ~s[3] ) |
  ( y[0]  &   s[0]  &   s[1]  &   s[2]  &   ~s[3] ) |
  ( ~y[0]   &   y[1]  &   ~s[1]   &   s[2]  &   ~s[3] ) |
  ( ~y[0]   &   y[1]  &   ~s[0]   &   s[1]  &   s[2]  &   ~s[3] ) |
  ( y[0]  &   y[1]  &   ~s[0]   &   ~s[1]   &   ~s[2]   &   s[3]  )
);

assign d[2] = (
  (   y[0]  &   y[1]  &   ~s[2]   &   ~s[3]   &   ~s[1] ) |
  (   y[0]  &   s[0]  &   s[1]  &   ~s[2]   &   ~s[3] ) |
  (   y[1]  &   s[1]  &   ~s[2]   &   ~s[3] ) |
```

```
  (   ~y[1]   &   ~s[0]   &   ~s[1]   &   s[2]  &   ~s[3] ) |
  (   ~y[0]   &   ~s[0]   &   ~s[1]   &   s[2]  &   ~s[3] ) |
  (   y[0]  &   y[1]  &   s[0]  &   s[2]  &   ~s[3] ) |
  (  y[1]   &   s[0]  &   s[1]  &   s[2]  &   ~s[3] ) |
  (   y[0]  &   y[1]  &   s[1]  &   s[2]  &   ~s[3] ) |
  (   y[0]  &   ~s[0]   &   ~s[1]   &   ~s[2]   &   s[3]  ) |
  (   y[1]  &   ~s[0]   &   ~s[1]   &   ~s[2]   &   s[3]  )
);

assign d[3] = (
  ( y[0]  &   y[1]  &   ~s[0]   &   ~s[1]   &   s[2]  &   ~s[3] )
);

endmodule
```

.

▼ Output Logic



Name

- outputlogic.sv

Description

- This is the module for the output logic
- This module processes the output of the vending machine

Inputs

- A[3:0]: 4-bit input from the state

Outputs

- curstate [6:0]: 7-bit output to the HEX showing the state of the machine
- curchange [6:0]: 7-bit output to the HEX showing the change of the machine
- far: 1-bit output for Farthing LED
- halfp: 1-bit output for Ha'penny LED

Code

```
// Authors: John Akujobi, LNU Sukhman Singh
// Date:  October, Fall, 2023
```

```
// Name:    outputlogic
// Filename: outputlogic.sv

// Description: This is the module for the output logic
// This module processes the output of the vending machine
// Inputs
// - A[3:0]: 4-bit input from the state
// Outputs
// - curstate[6:0]: 7-bit output to the HEX showing the state of the machine
// - curchange[6:0]: 7-bit output to the HEX showing the change of the machine
// - far: 1-bit output for Farthing LED
// - halfp: 1-bit output for Ha'penny LED

module outputlogic (
    input logic A[3:0],
    output logic curstate[6:0], // output for state
    output logic curchange[6:0],    // output for change
    output logic far ,  // output for Farthing LED
    output logic halfp  // output for Ha'penny LED
);

//process the states to the current state and show HEX
seven_seg (
    .A(A),
    .S(curstate)    // output for state
);

// module to output the change in Ha'penny and Farthing
changesev (
    .A(A) ,
    .S(curchange),
    .h(halfp),
    .f(far)
) ;

endmodule
```

▼ Decoder 4:16 *

Filename

- d416.sv

Description

- This module is a 4-to-16 decoder that takes a 4 inputs and generates a 9-bit output.

Inputs

- a0, a1, a2, a3 - 4 inputs

Outputs

- y[8:0] - 9-bit output

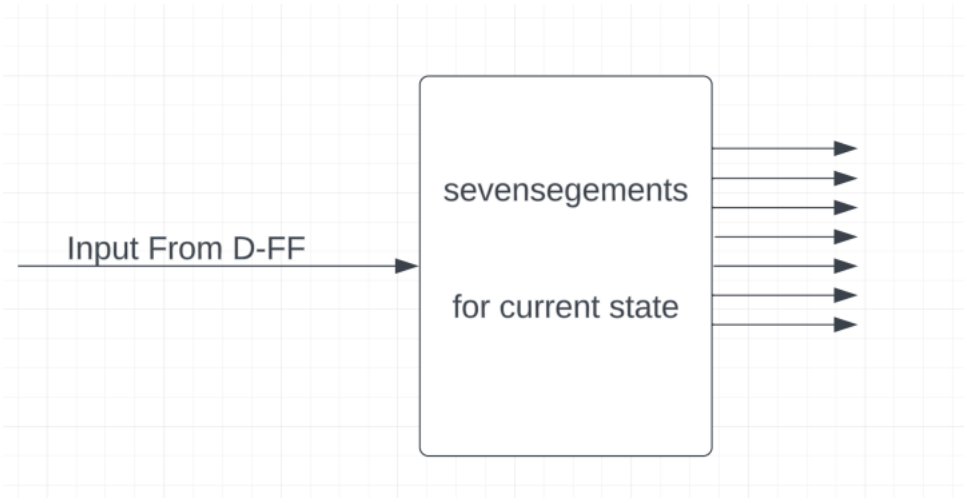
Code

```
// Authors: John Akujobi & LNU Sukhman Singh
// Date: October, Fall 2023
// Description:
//    This module is a 4-to-16 decoder that takes a 4 inputs and generates a 9-bit output.
// Input - a0, a1, a2, a3 - 4 inputs
// Output - y[8:0] - 9-bit output

module d416 (
  input  logic a0, a1, a2, a3, // 4 inputs
  output logic y[8:0] // 16-bit output
);
```

```
assign y[0]  = ~a3 & ~a2 & ~a1 & ~a0;
assign y[1]  = ~a3 & ~a2 & ~a1 &  a0;
assign y[2]  = ~a3 & ~a2 &  a1 & ~a0;
assign y[3]  = ~a3 & ~a2 &  a1 &  a0;
assign y[4]  = ~a3 &  a2 & ~a1 & ~a0;
assign y[5]  = ~a3 &  a2 & ~a1 &  a0;
assign y[6]  = ~a3 &  a2 &  a1 & ~a0;
assign y[7]  = ~a3 &  a2 &  a1 &  a0;
assign y[8]  =  a3 & ~a2 & ~a1 & ~a0;

endmodule
```

.

▼ Seven Segment Display



Filename

- seven_seg.sv

Description

- Description: This is the module for the seven segment display
- This helps us decide when a number on the seven segment will be given
- Hence, it controls the LEDs for seven segment display

Inputs

- A[3:0] - 4 bit input

Outputs

- S[6:0] - 7 bit output

Code

```
// Authors: John Akujobi, LNU Sukhman Singh
// Date:  October, Fall, 2023
// Name:    seven segment
// Filename: seven_seg.sv
// Description: This is the module for the seven segment display
// This helps us decide when a number on the seven segment will be given
// Hence, it controls the LEDs for seven segment display

module seven_seg (
```

```
  input logic  A[3:0],  // 4 bit input
  output logic S[6:0] // 7 bit output
);

logic Y[8:0];

// The decoder decodes the 4 bit input into a 9 bit output Y[8:0  ]
d416 (
  .a0(A[0]),
  .a1(A[1]),
  .a2(A[2]),
  .a3(A[3]),
  .y(Y)
);

assign S[0] = ~(Y[0] | Y[2] | Y[3] | Y[5] | Y[6] | Y[7] | Y[8] );
assign S[1] = ~(Y[0] | Y[1] | Y[2] | Y[3] | Y[4] | Y[7] | Y[8] );
assign S[2] = ~(Y[0] | Y[1] | Y[3] | Y[4] | Y[5] | Y[6] | Y[7] | Y[8] );
assign S[3] = ~(Y[0] | Y[2] | Y[3] | Y[5] | Y[6] | Y[8] );
assign S[4] = ~(Y[0] | Y[2] | Y[6] | Y[8] );
assign S[5] = ~(Y[0] | Y[4] | Y[5] | Y[6] | Y[8] );
assign S[6] = ~(Y[2] | Y[3] | Y[4] | Y[5] | Y[6] | Y[8]);

endmodule
```

.

▼ Tomato Decoder*

Filename

- tomd416.sv

Description

- This is the module for the Tomato LED
- This helps us decide when a Tomato will be given
- Hence, it controls the LED for Tomato

Inputs

- A [3:0]: 4 bit input from the state

Outputs

- t: 1-bit output for Tomato LED

Code

```
// Author: LNU Sukhman Singh & John Akujobi
// Date: October, Fall 2023
// Description: This is the module for the Tomato LED
// This helps us decide when a Tomato will be given
// Hence, it controls the LED for Tomato

// Input
// - A[3:0]: 4 bit input from the state
// Output
// - t: 1-bit output for Tomato LED

module tomd416 (
  input  logic a0, a1, a2, a3,  // 4 inputs for the states
  output t  //Tomato Output LED
);

logic y[8:0];

assign y[0]  = ~a3 & ~a2 & ~a1 & ~a0;   //First State
assign y[1]  = ~a3 & ~a2 & ~a1 &  a0; //Second State
assign y[2]  = ~a3 & ~a2 &  a1 & ~a0; //Third State
```

```
assign y[3]  = ~a3 & ~a2 &  a1 &  a0; //Fourth State
assign y[4]  = ~a3 &  a2 & ~a1 & ~a0; //Fifth State
assign y[5]  = ~a3 &  a2 & ~a1 &  a0; //Sixth State
assign y[6]  = ~a3 &  a2 &  a1 & ~a0; //Seventh State
assign y[7]  = ~a3 &  a2 &  a1 &  a0; //Eighth State
assign y[8]  =  a3 & ~a2 & ~a1 & ~a0; //Ninth State

assign t = y[5] + y[6] + y[7] + y[8]; //Tomato LED lights for these states

endmodule
```

▼ Change Seven Segment Display*

Filename

- changesev.sv

Description

- This is the module for the change in Ha'penny and Farthing
- It takes in a 4 bit input from the state and outputs a 7 bit output to the HEX display.
- It also outputs two control signals for the LEDs

Inputs

- A [3:0]: 4-bit input from the state

Outputs

- S [6:0]: 7-bit output to the HEX display showing the change in Farthings
- h: 1-bit output for Ha'penny LED
- f: 1-bit output for Farthing LED

Equations

S[0] = (Y[0] + Y[2] + Y[3] + Y[5] + Y[6] + Y[7] + Y[8])
S[1] = (Y[0] + Y[1] + Y[2] + Y[3] + Y[4] + Y[7] + Y[8])
S[2] = (Y[0] + Y[1] + Y[3] + Y[4] + Y[5] + Y[6] + Y[7] + Y[8])
S[3] = (Y[0] + Y[2] + Y[3] + Y[5] + Y[6] + Y[8])
S[4] = (Y[0] + Y[2] + Y[6] + Y[8])
S[5] = (Y[0] + Y[4] + Y[5] + Y[6] + Y[8])
S[6] = (Y[2] + Y[3] + Y[4] + Y[5] + Y[6] + Y[8])

Code

```
// Authors: John Akujobi, LNU Sukhman Singh
// Date:  October, Fall, 2023
// Name:    changesev
// Filename: changesev.sv

// Description: This is the module for the change in Ha'penny and Farthing
//  It takes in a 4 bit input from the state and outputs a 7 bit output to the HEX display.
//  It also outputs two control signals for the LEDs

// Inputs
// - A[3:0]: 4-bit input from the state

// Outputs
// - S[6:0]: 7-bit output to the HEX display showing the change in Farthings
// - h: 1-bit output for Ha'penny LED
// - f: 1-bit output for Farthing LED
```

```
module changesev (
  input logic  A[3:0],  // 4 bit input from the state
  output logic S[6:0] , // 7 nit output to the Hex display showing the change in Farthings
  output logic h , f //controls the led for Ha'penny and Farthing change
);

logic Y[8:0]; // 9 bit output decoded from the 4 bit input

d416 (
  .a0(A[0]),
  .a1(A[1]),
  .a2(A[2]),
  .a3(A[3]),
  .y(Y)
);

assign S[0] = ~( Y[7] | Y[8] );
assign S[1] = ~( Y[6] | Y[7] | Y[8] );
assign S[2] = ~( Y[6] | Y[8] );
assign S[3] = ~( Y[7] | Y[8] );
assign S[4] = ~( Y[7] );
assign S[5] = 1 ; // Keeps this set off (It is 1 because the HEX is low enable)
assign S[6] = ~( Y[7] | Y[8]);

assign h = (Y[7] | Y[8]); // controlls the LED for Ha'penny change
assign f = (Y[6] | Y[8]) ;  // controlls the LED for Farthing change

endmodule
```

▼ Mod Temp

Filename

-

Description

- .

Inputs

- .

Outputs

- .

Code

```

```

.

▼ Thank You