Digital Logic Project Report -Vending Machine

Group member: 陈泷清、罗景南、方唯可

Roles and responsibilities

1. Roles

Jingnan: Replenishment module, buzzer module, button debouncing, report
Longqing: User interface design, slides, report, graphic design
Weike: product inquiry module, purchase and payment module, seven-seg display design, report

2. Timeline

December 3: Project selection
December 13: Structural design, tasks division.
December 20: Module integration and debugging.
December 22: Slides, presentation preparations. Final debugging and testing on board.

Code

Source codes, constraint files, and test modules can be found via
https://github.com/ColinFang1009/SUSTech_CS207_finalproject/tree/master/source

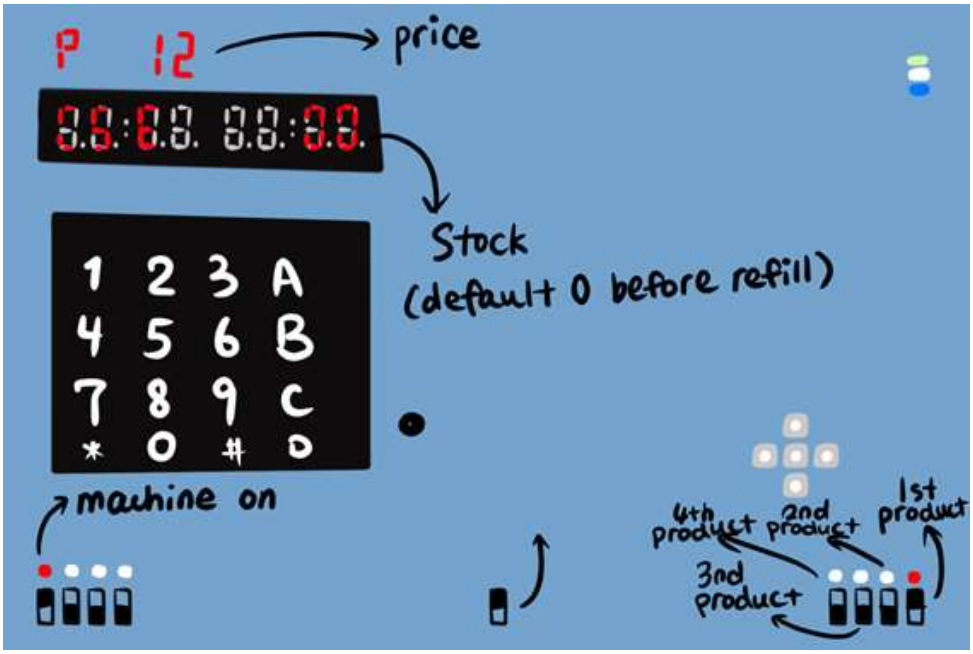
Design

Requirement and Functional Design

The project asks us to design a vending machine with four major modes: product inquiry mode, purchase and payment mode, product and inquiry mode, and sales inquiry mode.

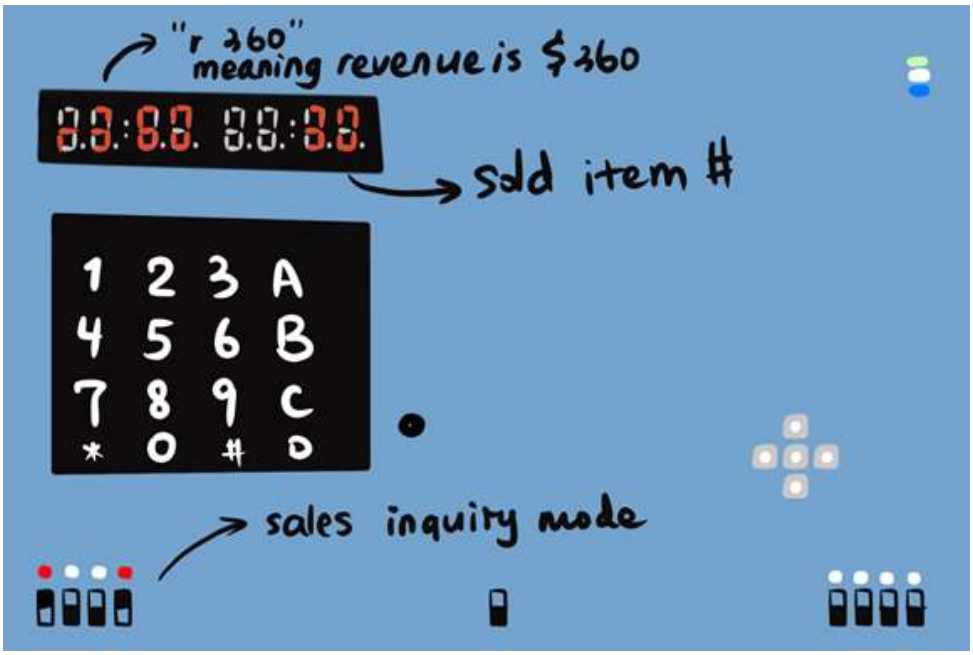
Below is the diagram for the product inquiry mode:

The three switches on the lower left corner (besides the master switch) represent purchase, replenishment, and sales inquiry mode. If none of them is switched on, the machine is in the product inquiry mode. Switches on the lower right corner represent our 4 products offered in the vending machine. When one switch is turned on (multiplexer with priority), the seven-seg tubes will scroll and show the name, price, and current quantity.

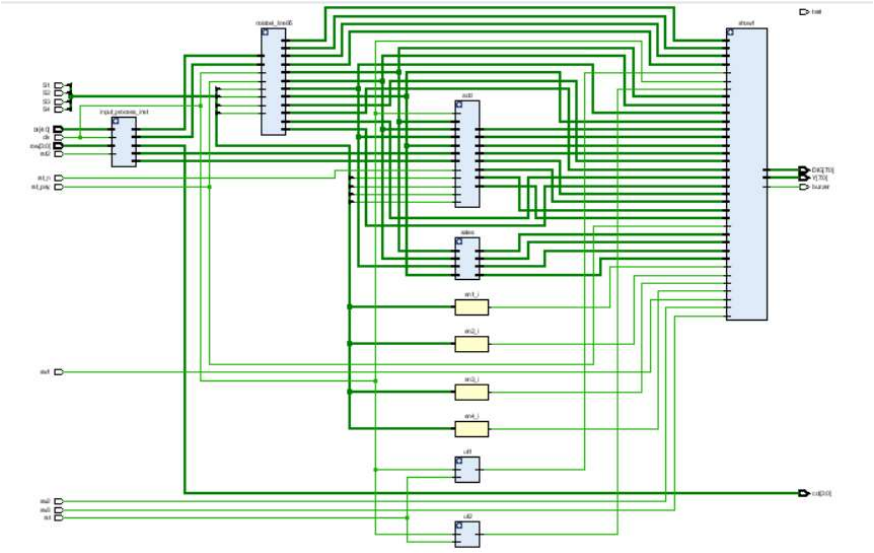


Sales inquiry mode

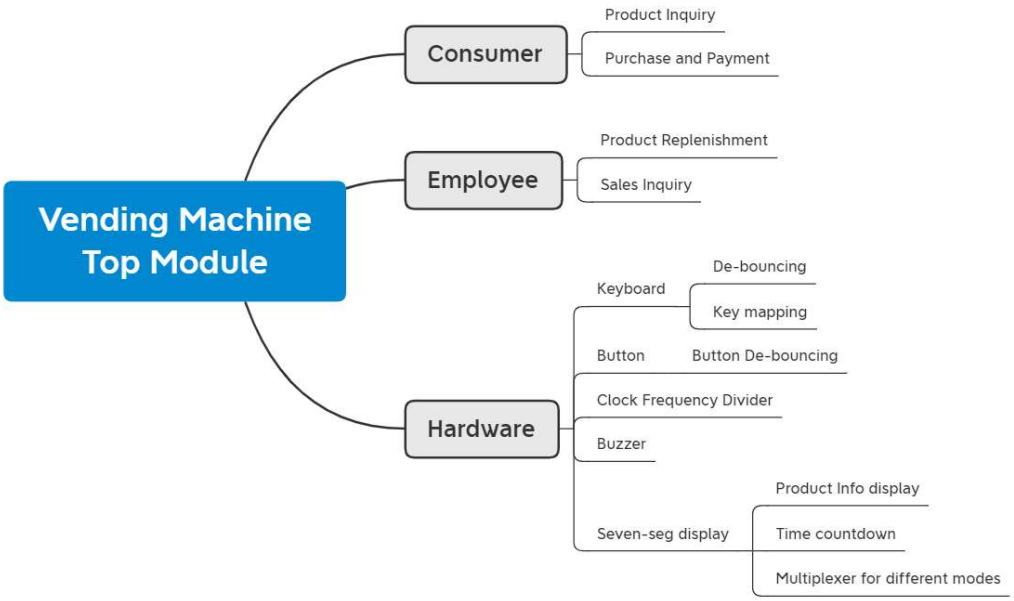
When the fourth button on the lower left corner is turned on, the vending machine is in the sales inquiry mode. Employees can view the quantity sold and revenue for each product.



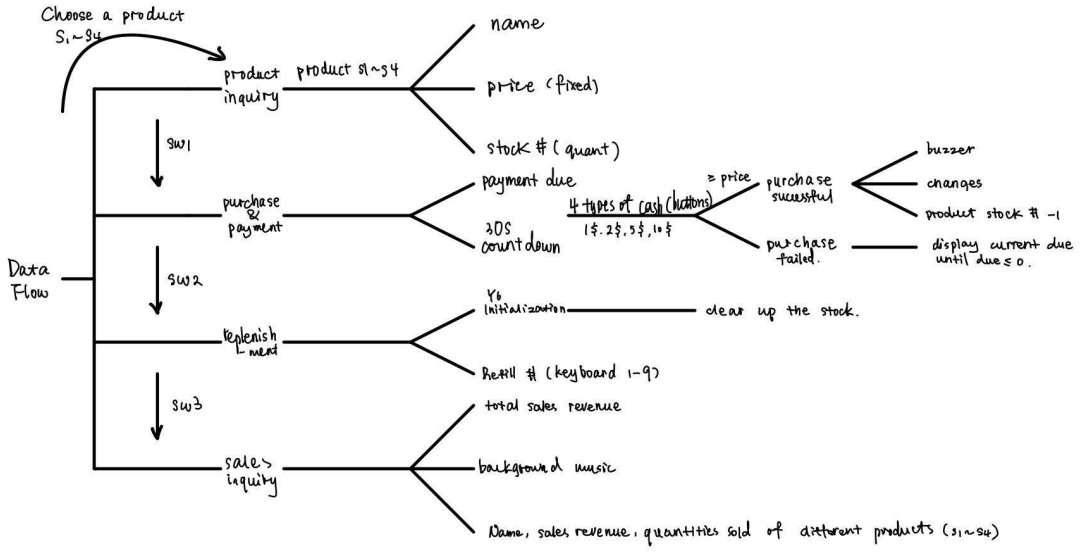
RTL Design Schematic

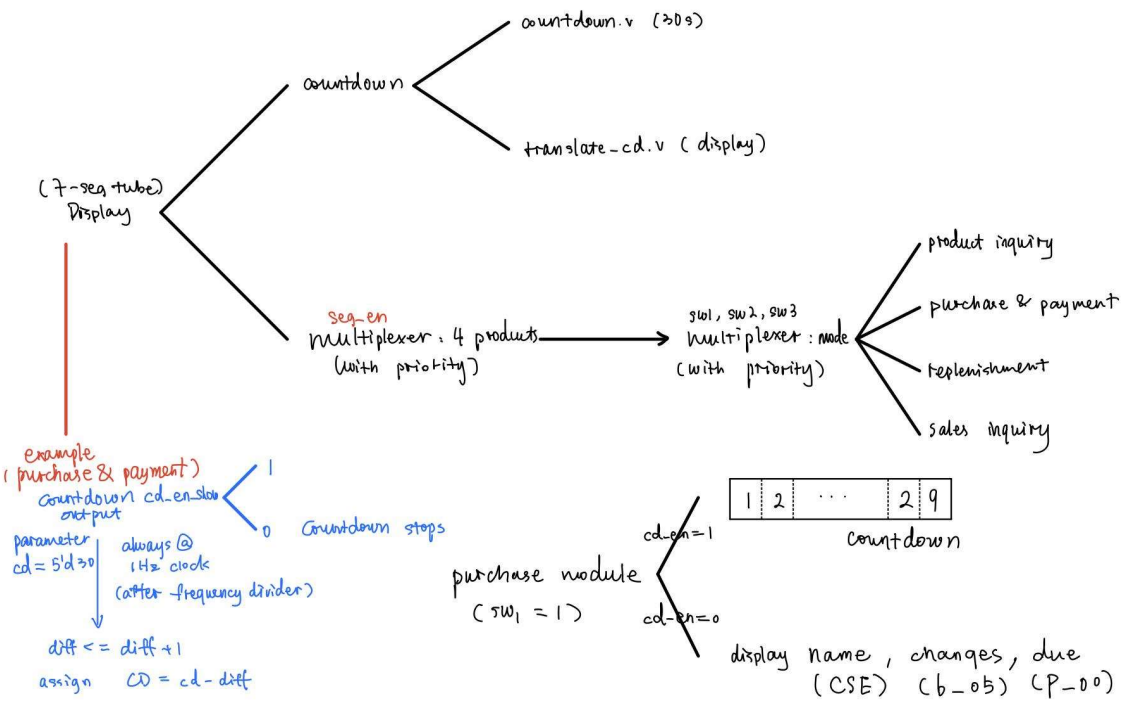


Mindmap: functional structure of the vending machine



System execution and data flow





Top modules and submodules

Top module (top.v)

It uses a multiplexer to instantiates display module (seven_seg.v), purchase module (pay.v), replenishment module (manage.v), sales and inquiry module (sales_data_processing.v), and frequency divider module (clock.v).

```
1.  module top(
2.      input rst,//reset for clock
3.      input rst2,//reset for key scanners
4.      input rst_pay,//reset for pay & display module
5.      input clk,
6.      input rst_n, //reset for manage module
7.      input sw1,sw2,sw3,  //modes(purchase, refill, sales inquiry)
8.      input S1,S2,S3,S4,  //4 products
9.      input [3:0] row,  //keys
10.     output [3:0] col,//keys
11.     Input [4:0] bt,  //5 buttons
12.     output [7:0] DIG,  //display
13.     output [7:0] Y,  //seven_seg
14.     output buzzer);
```

Sales inquiry(sales_data_processing.v)

This module is essentially calculating the sales revenue given the inputs of quantity sold. For

example, the product 1O1 has price $2, the module will calculate the revenue by multiplying price (as a fixed parameter) and quantity sold (count).

```
module sales_data_processing(
    input[3:0] count1, //quantity sold for product 1
    input[3:0] count2,
    input[3:0] count3,
    input[3:0] count4,
    output[3:0] sale1,//revenue for product 1
    output[3:0] sale2,
    output[3:0] sale3,
    output[3:0] sale4
    );
```

clock frequency divider (clock.v)

This module is used multiple times in the project. In the 7-seg display module, two clocks with different frequencies are used (one with 500 Hz to display different numbers on the tube, one with 1 Hz for "scrolling display")

```
module clock(rst,clk,clkout);
input rst, clk;
output reg clkout;
reg [31:0] cnt;
parameter period = 200000;
//clock1:200_000 for displaying different number/char on the tube
//clock2:100_000_000 for countdown and scrolling display
```

Input control hub(input_process.v)

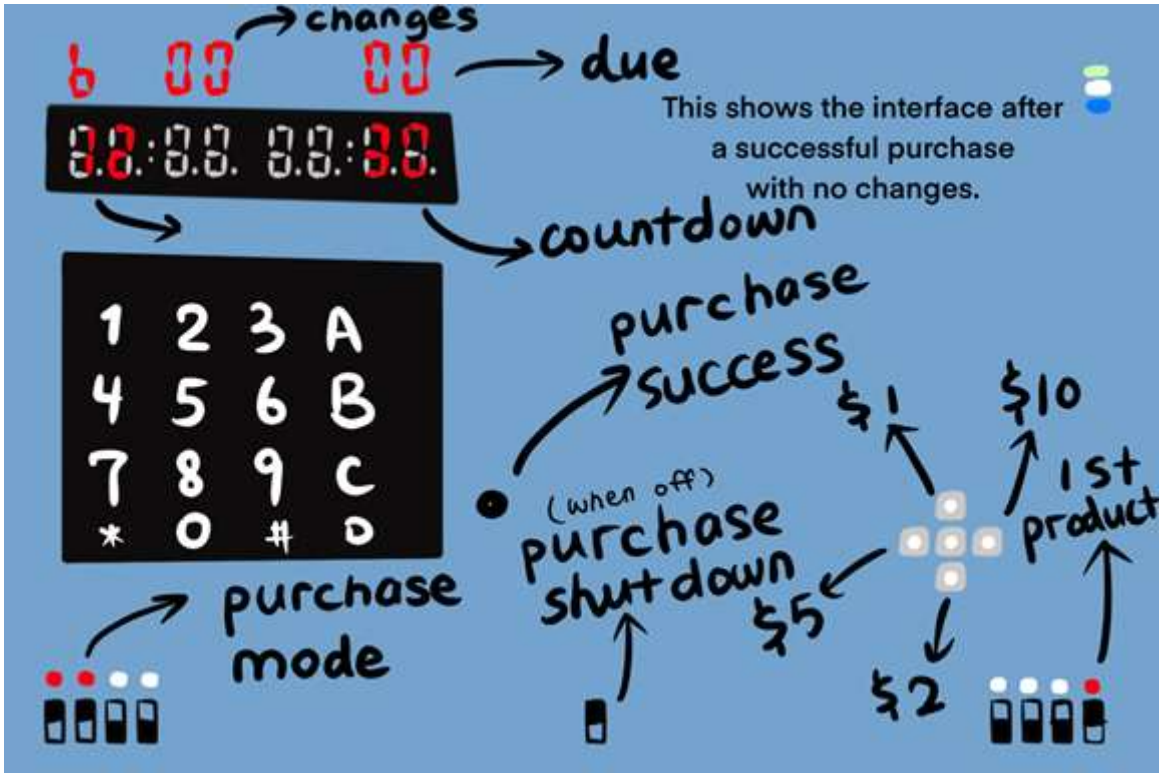
This module manages the keyboard inputs and button inputs. It instantiates the button de-bouncing module and button edge module to generate valid output signal and store them in bt_edge and key_edge, respectively.

```
module input_process(
    input clk, rst, //clock, reset
    input[3:0] row, //row input for keyboard
    input[4:0] bt,//five buttons
    output[3:0] col, //column output for keyboard
    output[4:0] bt_press,//button pressing after debouncing
    output[4:0] bt_edge,//edge for button pressing
    output[15:0] key_press, //key pressing after debouncing
    output[15:0] key_edge //edge for key pressing
);
```

Purchase and payment (pay.v)

This pay.v module manages the data flow during purchase and payment. The buttons represent $1, $2, $5, $10. Once a button is pressed, the vending machine will notice the

corresponding amount is paid, and indicate the amount to be paid (remain variable). Once the countdown ends, the machine will calculate the money back to consumers (back variable), and add 1 to the quanntity sold.




```
module pay(
    input sw1, sw2, sw3, sw4,//product 1/2/3/4
    input clk,//clock
    input rst_n,//reset
    input[4:0] bt_press,
    input[4:0] bt_edge,
    output reg[3:0] remain1, //amount to be paid
    output reg[3:0] remain2,
    output reg[3:0] remain3,
    output reg[3:0] remain4,
    output reg[3:0] back1, //change back
    output reg[3:0] back2,
    output reg[3:0] back3,
    output reg[3:0] back4,
    output reg [3:0] count1,//quantity sold
    output reg [3:0] count2,
    output reg [3:0] count3,
    output reg [3:0] count4);
```

Payment time countdown (countdown.v)

The module generates a countdown system (from 30 to 0) and buzzer output when the countdown ends. (Credit to www.cnblogs.com/Clouds42/p/11897777.html for inspiration)

```
module countdown(
```

```
2.  input CK, //1Hz clock
3.  input clk_system,//system 500MHz clock
4.  input rst, //reset
5.  output [4:0] CD, //CountDown
6.  output cd_en_show, //bool for countdown status
7.  output reg buzz //buzzer
8.  );
```

translate_cd.v

The module translate the countdown value (30-0) to values that can be shown on the 7-seg tubes.

```
1.  module translate_cd(
2.  input [4:0] CD, //countdown
3.  output reg [7:0] CD_show1, //seg1 for first digit
4.  output reg [7:0] CD_show2); //seg2 for second digit
```

7-seg display (seven_seg.v)

This module manages the all display on the 7-seg tubes. It instantiates the product_show.v module for each product.

It uses a priority multiplexer to decide the mode (sw1, sw2, sw3) and display the information accordingly. The 4 seg_en variables are the enable input for the four products, allowing the machine to know which product to show.

```
1.  module seven_seg(
2.      input rst,
3.      input clk_system,clk,clk2, //100MHz,500Hz, 1Hz
4.      input sw1,sw2,sw3, //Mode
5.      input [3:0] quant1, //quantity of product1
6.      input [3:0] quant2,
7.      input [3:0] quant3,
8.      input [3:0] quant4,
9.      input [3:0] max_add1, //maximum # of replenishment
10.     input [3:0] max_add2,
11.     input [3:0] max_add3,
12.     input [3:0] max_add4,
13.     input [3:0] pay_remain, //amount to be paid
14.     input [3:0] pay_remain2,
15.     input [3:0] pay_remain3,
16.     input [3:0] pay_remain4,
17.     input [3:0] back1,  //change back
18.     input [3:0] back2,
19.     input [3:0] back3,
20.     input [3:0] back4,
21.     input [3:0] sale1, //sales revenue
22.     input [3:0] sale2,
23.     input [3:0] sale3,
24.     input [3:0] sale4,
25.     input [3:0] count1, //quantity sold
26.     input [3:0] count2,
```

```
    input [3:0] count3,
    input [3:0] count4,
    input seg_en, seg_en2,seg_en3, seg_en4,//enable input
    output [7:0] DIG,//bit selection
    output [7:0] Y,//seg selection
    output buzz  );//buzzer
```

product_show.v

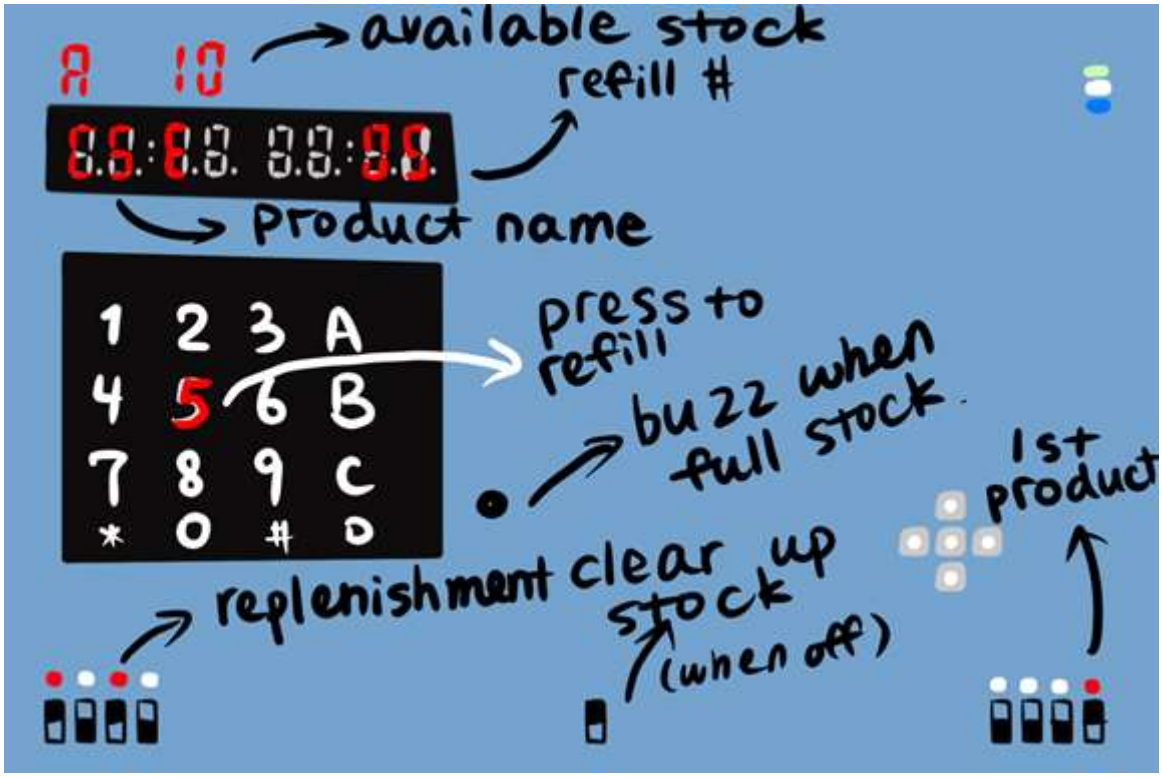
This module manages the bit selection (DIG_r) and essential information for each product. It maps the numerical value of quantity, maximal # of refill, changes back, quantity sold, etc. to displayable outputs on the 7-seg tubes (for example, quant_show_out1, quant_show_out2). Moreover, it decides the bits on the 7-seg tube to show the information.

```
module product_show(
    input [3:0] quant, //quantity
    input [3:0] max_add,//maximal # of replenishment
    input [3:0] pay_remain,// amount to be paid
   input [3:0] back, //change back
   input [3:0] sale, //sales revenue
   input [3:0] count,//quantity sold
   input seg_en,//enable
   input cd_en,//countdown_status
    input clk, clk2, //500Hz, 1Hz
    input rst,//reset
    input sw1,sw2,sw3,//mode
    output [3:0] scan_cnt_show,//display index
    output [1:0] scan_cd_show,//countdown index
    output reg [7:0] DIG_r, //bit selection
    output [7:0] quant_show_out1, //display digit1 for quantity
    output [7:0] quant_show_out2, //display digit2 for quantity
    output [7:0] max_add_out1,
    output [7:0] max_add_out2,
    output [7:0] pay_remain_out1,
    output [7:0] pay_remain_out2,
    output [7:0] back_out1,
    output [7:0] back_out2,
    output [7:0] sale_out1,
    output [7:0] sale_out2,
    output [7:0] count_out1,
    output [7:0] count_out2
    );
```

Replenishment module (manage.v)

The module allows the manager to replenish the products. We set four switches which represent four products, and a reset button to clear the chosen product. We also activate the keyboard for the manager to input the quatity of replenishments. During the whole process the basic information of the product will be shown on the screen(name, current quantites and

the maximum quantities to add) .




```
module manage(
    input sw1,sw2,sw3,sw4, //4 products
    input clk,
    input rst_n,
    input[15:0] key_press, //key pressing out
    input[15:0] key_edge, //key pressing edge
    input [3:0]count1, //quantity sold for product1
    input [3:0]count2,
    input [3:0]count3,
    input [3:0]count4,
    output  [3:0]max1,//maximal # of replenishment for product1
    output  [3:0]max2,
    output  [3:0]max3,
    output  [3:0]max4,
    output [3:0] quant1,//current quantity of product1
    output [3:0] quant2,
    output [3:0] quant3,
    output [3:0] quant4);
```

music

This module instantiates the buzzer player module and make it to play different frequencies (pitch) and generate music pieces of our choice.

For this machine, we set the music to be 彩虹 by Jay Chou and 青春修炼手册 by TFBoys, which are recorded in the video submitted. Moreover, the LED will light and twinkles while the music is playing.

```
module music_player(
```

```
    input clk,//clock
    input rst,//reset P20
    output[23:0] led,//LED twinkle while playing music
    output buzzer
);
```

button_edge.v

This module instantiates the butter_jitter module to handle the jitter while pressing, after which, it generates a buton_edge signal which lasts for one period of the clock. (Credit to https://github.com/Gogomoe for inspiration and codes)

```
module button_edge(
    input clk, //clock
    input but_in, //button input
    output but_press, //button press after de-bouncing
    output but_edge //edge of button press
);
```

button_jitter.v

This module is used to avoid the jitter while pressing the button (de-bouncing). It regards the pressing valid only when the pressing lasts for a certain period of time. (Credit to https://github.com/Gogomoe for inspiration and codes)

```
module button_debouncing(
    input clk,//clock
    input but_in,//button input
    output but_out //button output after de-bouncing
);
```

clock_buzz.v

It is essentially the same with the clock module above, but sets the period to be 1000, thus generating a clock of frequency 500,000 Hz for the buzzer.

```
module clock_buzz(rst,clk,clkout);
input rst, clk; //reset, clock
output reg clkout;//clock output for buzzer
```

buzzer_play.v

This module uses the clock in the clock_buzz module and allow the buzzer to play the given notes (frequencies). (Credit to https://github.com/Gogomoe for inspiration and codes)

```
module buzzer_play(
    input clk, //clock
    input[11:0] hz_next, //frequency (Hz) played by buzzer
    output reg buzzer //buzzer
);
```

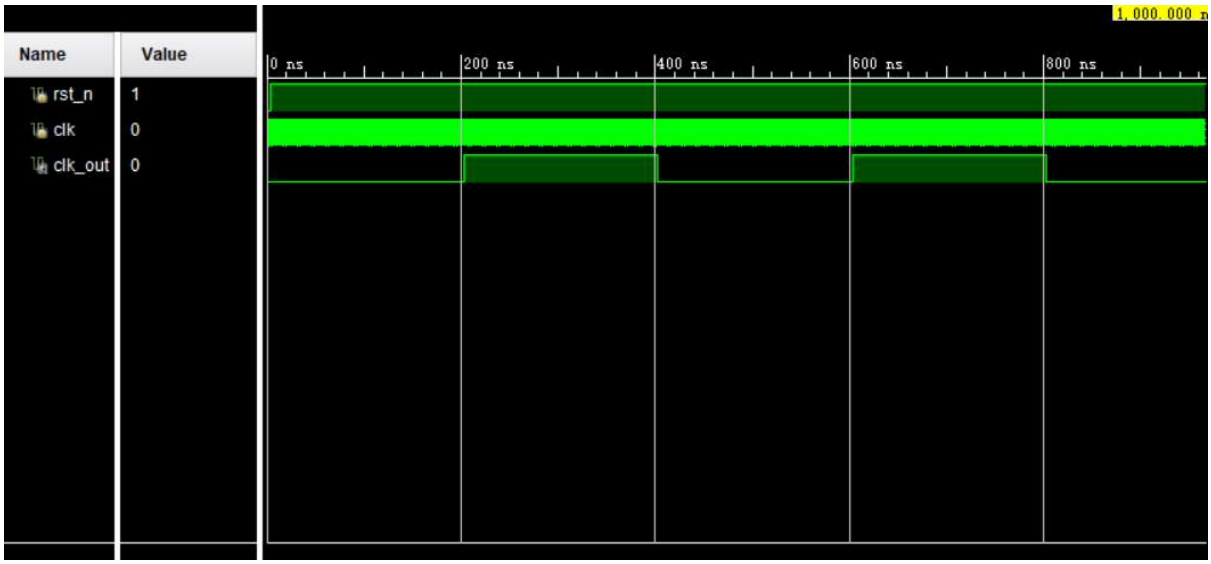
Constraint files can be viewed on the github link. It's not pasted here since it's lengthy.

Testing

Testbench / Simulation

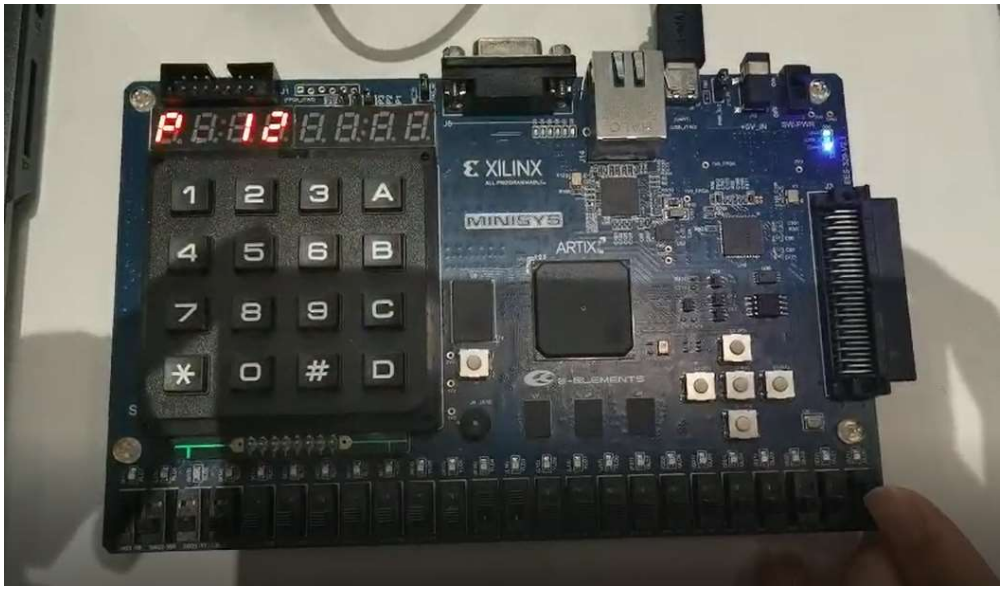
Scrolling display (two clock with different frequencies, 500Hz and 1Hz)

For convenience, the period is set to 200. As we can see, the clock ticks every 200ns. As we can see from the simulation, the program outputs correctly. In the real code, the period is set to 200,000 and 100,000,000 repectively for scrolling display.

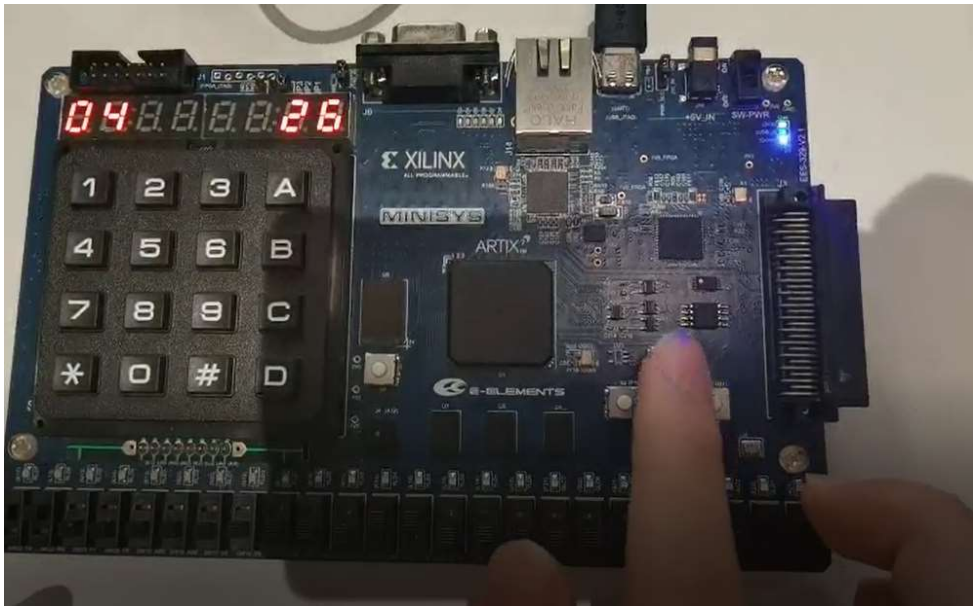


Testing on Minisys Develop Board

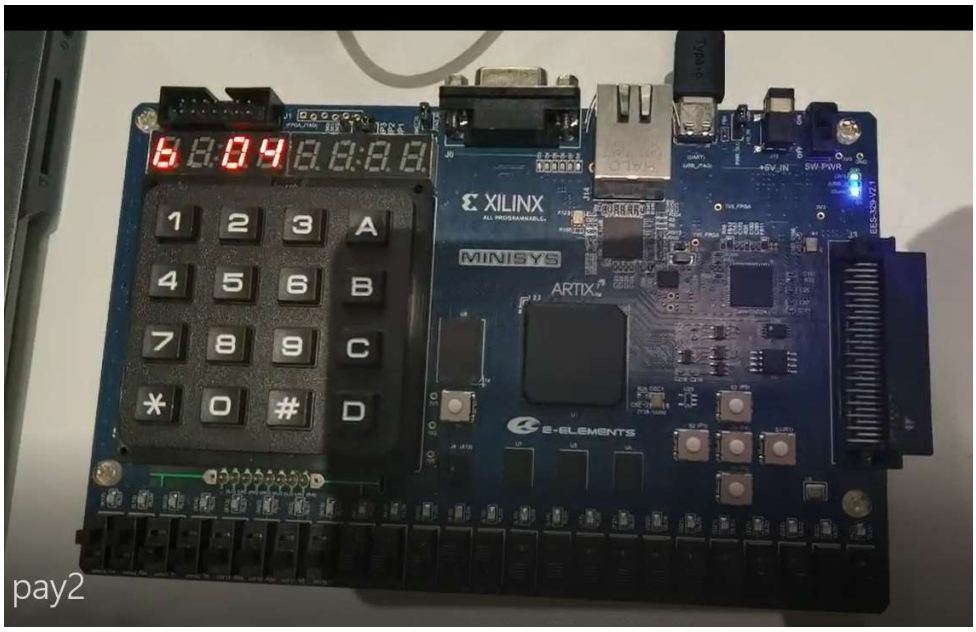
1. Product inquiry mode: when a product is selected, the 7-seg tube will scroll and show the name, price and current quantity. Below is the screenshot for the product CSE, with price $12 and current quantity 0.



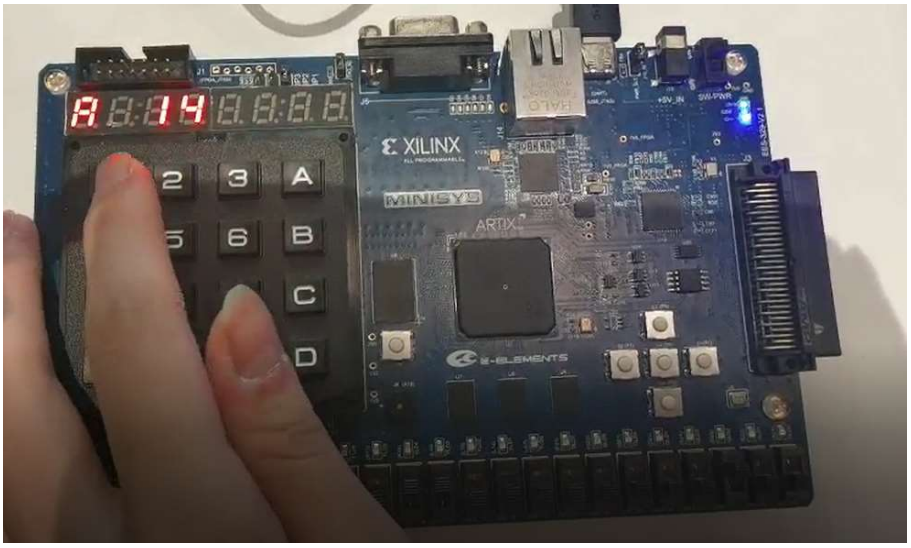
2. **Purchase and Payment**: the two bits on the right show the coundown (from 30 to 0) and the two bits on the left show the amount of of money to be paid. The white buttons, when pressed, will prompt the payment due to reduce accordingly.

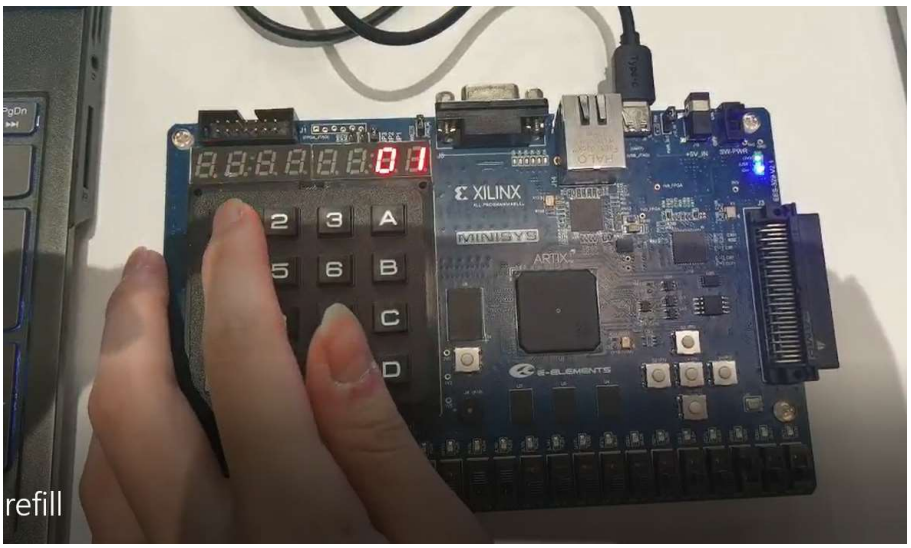


Once the payment is successful, the 7-seg tube will scroll and show the name and change back to users if the payment exceeds the price. Below shows the machine is returning $4 to the customer.



3. Replenishment: the total quantity that can be put on the vending machine for each product is 15. In this mode, when the employee press the keys (1-9), the available stock and current quantity will change accordingly. In the demo below, the employee put 1 product on, so the available stock becomes 14 and the quantity becomes 1.





More board testing can be seen in the video. All functionalities, besides the music player, are checked after the presentation.

Conclusion

Problems and Fixes

Problem1: There were no reactions from the keyboard when we wrote down the code for keyboard fot the first time.

Solution: the variable 'column' for scanning should be an output variable, and we misunderstanding its function at first.

Problem2: We need to change the quatity of products while both purchasing and managing, but it's not allowed to change one variable in two submodules.

Solution: Define a medial variable count, which represents how many products has been bought, and we can use this new variable to implement the change of qualities in one module.

Problem3: In the purchase and payment mode, we need to first display the countdown & payment due, and then change back. We found it hard initially for integrating the countdown module into our payment module.

Solution: We add a cd_en signal as a boolean to show the status of the countdown. If the countdown ends, the cd_en will become 0, and then the 7-seg tubes can display name, change back, and payment due in the scrolling manner.

Highlights and Improvement

1. Mode selection: we set different switches for consumer mode (product inquiry and purchasing) and employee mode (replenishment and sales inquiry), so consumers are unlikely to enter the employee mode unintenally.
2. User friendly design: the machine has clear functional structure and is easy to use by consumers. The keyboard and buttons have different functions: the former for replenishment and the latter for payment. Therefore, it avoids unintended activation or touches by users.
3. Music mode: the machine will play music when it's on the sales inquiry mode.
4. High stability: although the machine doesn't have fancy features (such as behaving like a human being), it has high stability for all basic functionalities. For examples, it correctly monitors the quantity across purchasing mode, replenishment mode, and resets. Moreover, since all keys and buttons are "de-bounced," customers and employees are unlikely to touch by mistake, largely enhancing the stability of the vending machine.

Places to improve:

1. As Professor Georgios mentions during our presentation, the machine can be more intelligent and send reminders to employees and customers when the quantity of the product is 0.
2. The purchase and payment module can be improved. Currently, it doesn't stop counting even if the payment due is 0, which is a bit waste of time for users.
3. Reset buttons: since there are a lot of reset buttons required, such as the master switch, reset for replenishment, reset for countdown, reset for payment, etc., we can have a more clever design and give more detailed instruction to user so that they won't be confused.