ĐẠI HỌC BÁCH KHOA THÀNH PHỐ HỒ CHÍ MINH
Khoa Điện- Điện tử
Bộ Môn Điện tử

BÁO CÁO
BÀI TẬP LỚN

Môn học: Kỹ thuật số nâng cao

GVHD: TS. Trần Hoàng Linh
SVTH: Lê nguyễn Gia Thịnh
MSSV: 1613347

HỒ CHÍ MINH, THÁNG 11 NĂM 2020

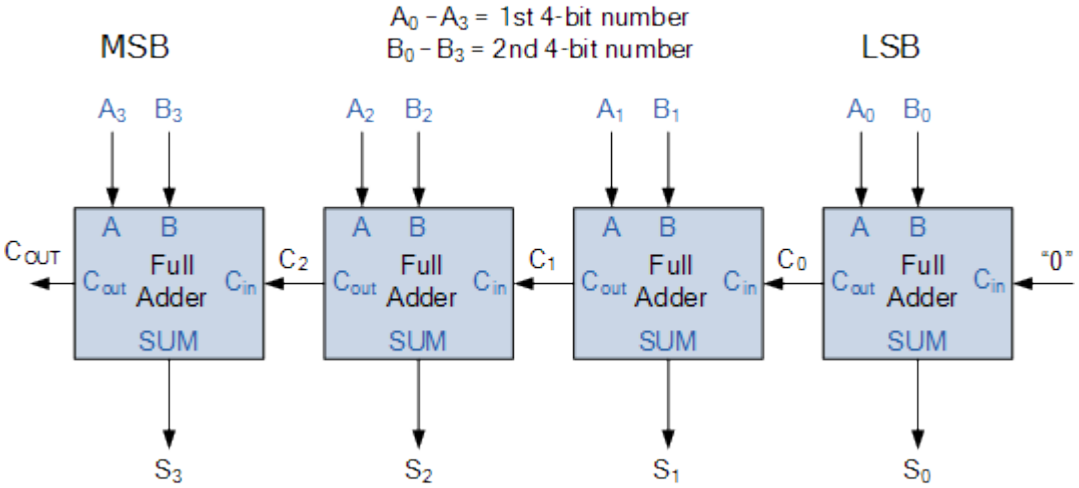
LAB 1: Thiết kế bộ cộng 4 bit

1. Mục tiêu:

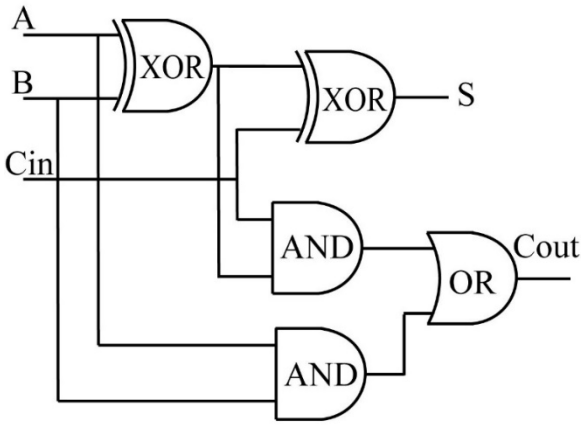
Sử dụng cho phép cộng trong bộ ALU của CPU, dùng để cộng các toán hạng với nhau hoặc cộng các địa chỉ với các số offset qua đó giúp thực hiện các lệnh nhảy của chương trình.

2. Cơ sở lý thuyết:

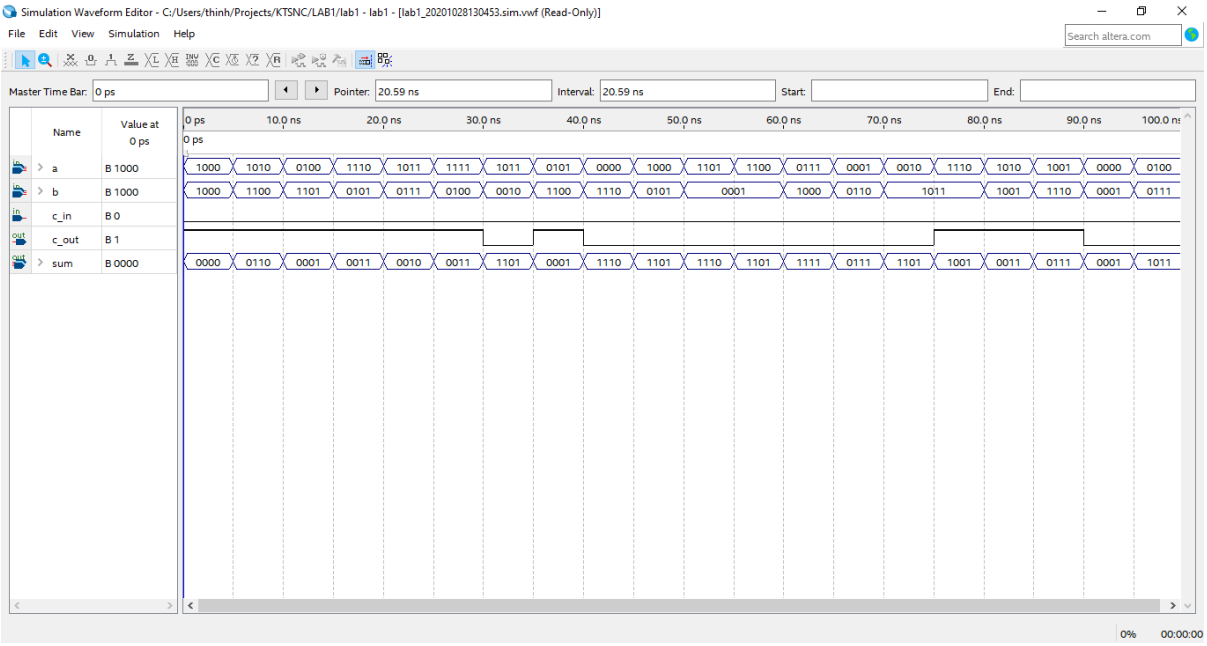
Mạch là sự kết hợp giữa 4 Khối full adder như trong hình. Khối full adder có nhiệm vụ thực hiện việc công cho một bit thứ nhất (có thêm carry in) và chuyển cờ carry out qua bộ full adder tiếp theo thực hiện với bit tiếp theo.



Mỗi khối full adder là tổ hợp của các phép logic như hình bên dưới gồm A, B là hai tín hiệu cần cộng, Cin là carry in, S là tổng của phép toán và Cout là carry out



3. Kết quả mô phỏng:



4. Nhận xét và kết luận:

Bộ cộng Full Adder 4 thực hiện đúng chức năng mong muốn và có thể ứng dụng cho các bộ ALU của CPU.

LAB 2: Xây dựng một ALU 4 bit

1. Mục tiêu:

Mục tiêu của bài thí nghiệm là xây dựng bộ xử lý logic về toán học (ALU- Arithmetic and Logic Unit).

ALU là một mạch tổ hợp để xử lý các tác vụ về logic và toán học dựa trên hai số hạng. Các tác vụ cho ALU thực hiện được điều khiển bằng ngõ nhập function-select.

Mục đích của bài thí nghiệm là thiết kế một ALU đơn giản như sau:

- Độ dài các toán hạng là 4 bit.

- Các ngõ nhập function-select gồm có: M, S0, S1.

- Các tác vụ ALU thực hiện được cho trong bảng 1.

2. Cơ sở lý thuyết:

Dùng nguyên tắc "chia để trị" để thiết kế ALU.

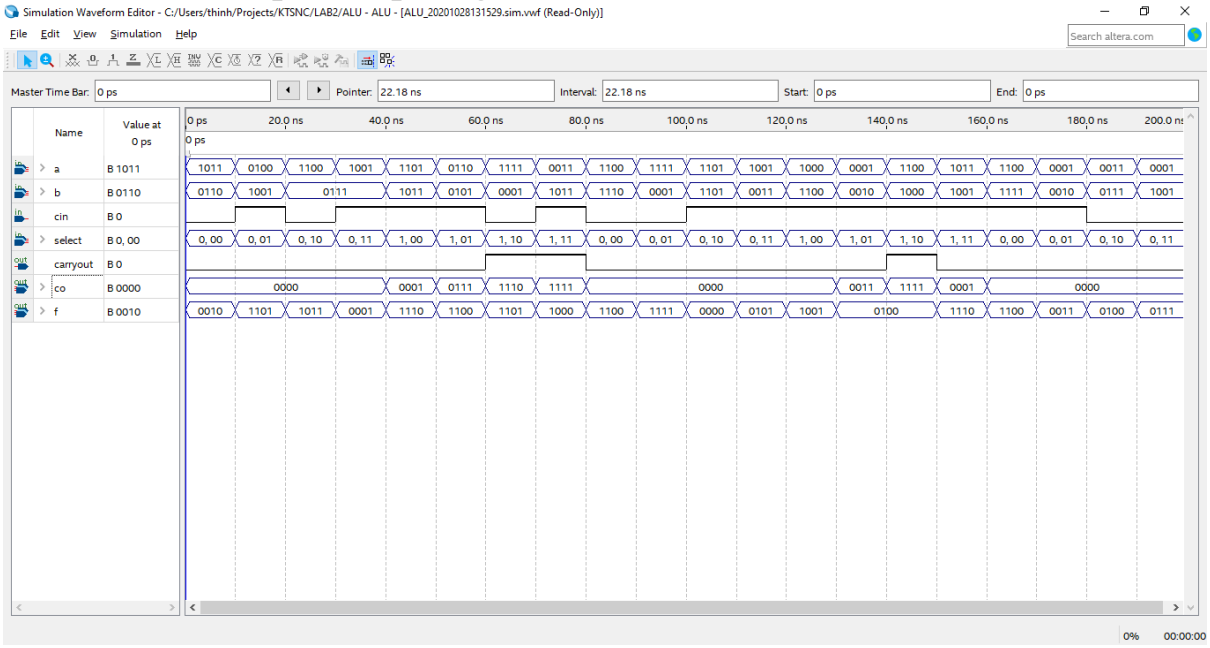
Có nghĩa là:

ALU 4-bit là 4 ALU 1-bit. Nếu xét về cấu trúc, ta có thể xem một ALU n-bit được cấu thành từ n ALU 1-bit. Như vậy để thiết kế một ALU 4-bit, ta chỉ cần thiết kế một ALU 1-bit. Sau đó, có thể ghép nối 4 ALU 1-bit này lại với nhau để tạo thành ALU 4-bit. Mỗi ALU 1-bit như vậy được gọi là một bit-slice.

M	S1	S0	Chức năng	Tác vụ
0	0	0	Ai.Bi	AND
0	0	1	Ai + Bi	OR
0	1	0	Ai (+) Bi	XOR
0	1	1	~(Ai (+) Bi)	XNOR
1	0	0	A + C0	Cộng A với Carry
1	0	1	A+B+C0	Cộng A, B và Carry
1	1	0	A+B'+C0	Cộng A với bù B và Carry
1	1	1	A'+B+C0	Cộng B với bù A và Carry

Nếu xét về chức năng, ta có thể phân ALU thành hai phần chuyên biệt, một về logic và một về toán học. Sau đó, có thể dùng một MUX 2:1 để kết hợp hai khối này. Cách này có ưu điểm là thiết kế từng khối nhỏ sẽ dễ hơn so với thiết kế một bit-slice, vốn cần thiết kế một ALU hoàn chỉnh. Hình sau thể hiện sơ đồ khối của một bit-slice ALU thực hiện theo ý tưởng này.

3. Kết quả mô phỏng:



4. Nhận xét và kết luận:

Sau khi kiểm tra cho thấy code chạy đúng và cho kết quả theo những yêu cầu cho trong bảng 1. Đã kiểm tra bằng tính tay.

Bộ ALU 4-bit được thiết kế đúng, cho kết quả chính xác.

LAB 3: Thiết kế máy bán nước ngọt tự động

1. Mục tiêu:

Mục tiêu của bài thí nghiệm là xây dựng một máy bán hàng tự động (Vending Machine) có đầy đủ các chức năng cơ bản.

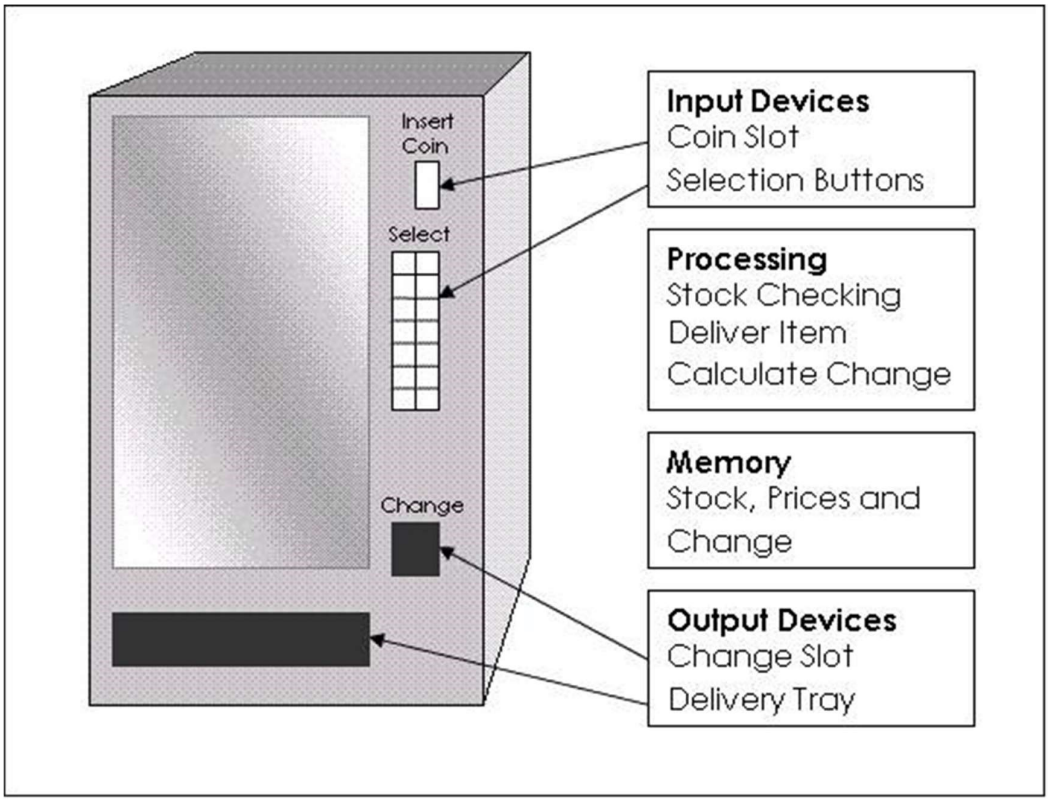
Giúp sinh viên nắm rõ được lý thuyết về máy trạng thái hữu hạn (Finite State Machine – FSM) và các úng dụng cơ bản của nó.

Nâng cao khả năng lập trình và sửa lỗi code, mô phỏng với môi trường ModelSim.

2. Mục đích:

Thiết kế máy bán nước ngot tự động (vending machine) tuân thủ các nguyên tắc sau:

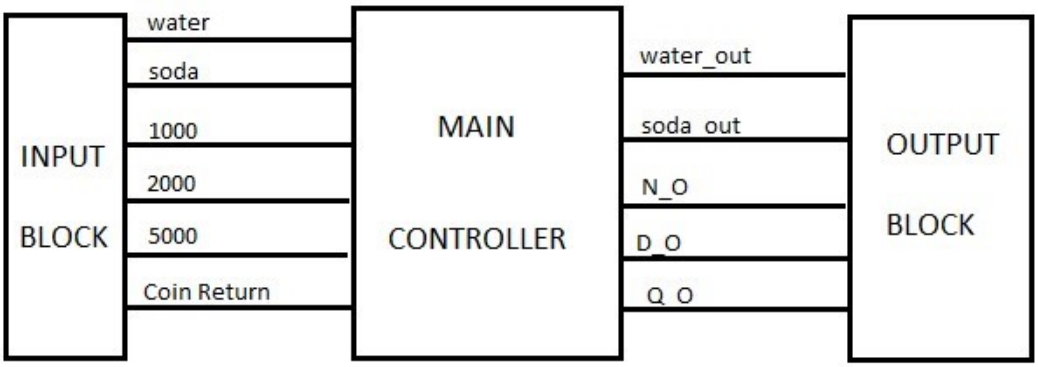
- Nước ngọt (Soda) giá 9000, nước suối (Water) giá 7000
- Máy nhận xu: 1000, 2000 và 5000 (N, D, Q)
- Số tiền trả lại sao cho số xu ít nhất (Give change in the smallest # coins possible) Nếu số tiền bỏ vào lớn hơn 9000 máy sẽ tự trả lại tiền vừa bỏ vào sau.
- Máy có nút Coin Return (CR) dùng để trả lại hết tiền vừa bỏ vào.
- Nếu không có nút nào được ấn thì máy trạng thái giữ nguyên trạng thái cũ.
- Các ngõ ra:
  - Coin Return out (trả hết tiền khi CR được bấm)
  - Water out (WO) (mua nước suối) o Soda out (SO) (mua nước ngọt) o Change (CO) (trả tiền thừa)



3. Cơ sở lý thuyết:

Máy bán hàng tự động được thiết kế gồm 3 khối chức năng chính: Khối tín hiệu vào, Khối xử lý trung tâm và khối tín hiệu ra.

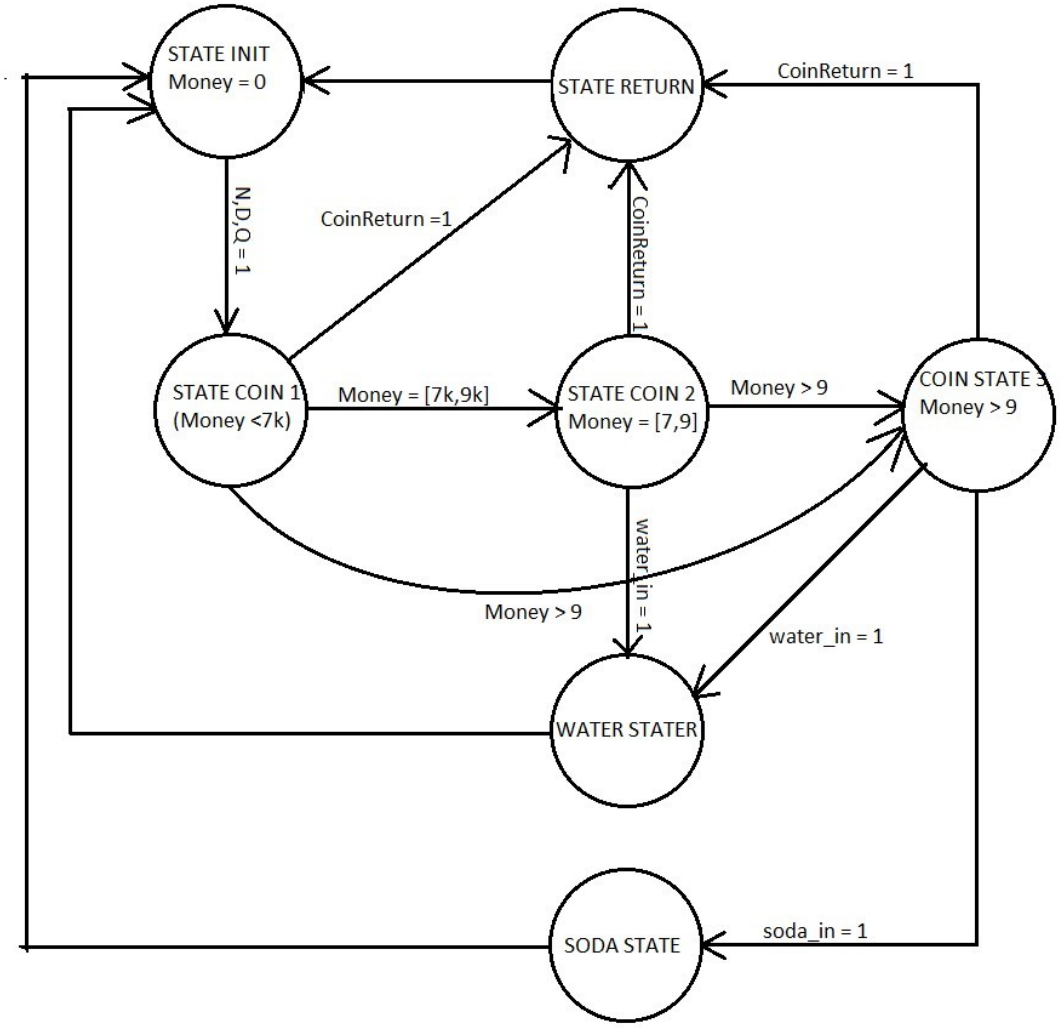
- Khối tín hiệu vào bao gồm 6 ngõ vào: water, soda, 1000 (N), 2000 (D), 5000 (Q), Coin Return (CR).
- Khối điều khiển trung tâm là khối chức năng chính của hệ thông được thiết kế theo FSM dựa theo các yêu cầu đã đề ra.
- Khối tín hiệu ra bao gồm 5 tín hiệu: water_out, soda_out, N_O, D_O, Q_O.



Máy trạng thái hữu hạn

Máy trạng thái được thiết kế sẽ bao gồm các trạng thái sau:

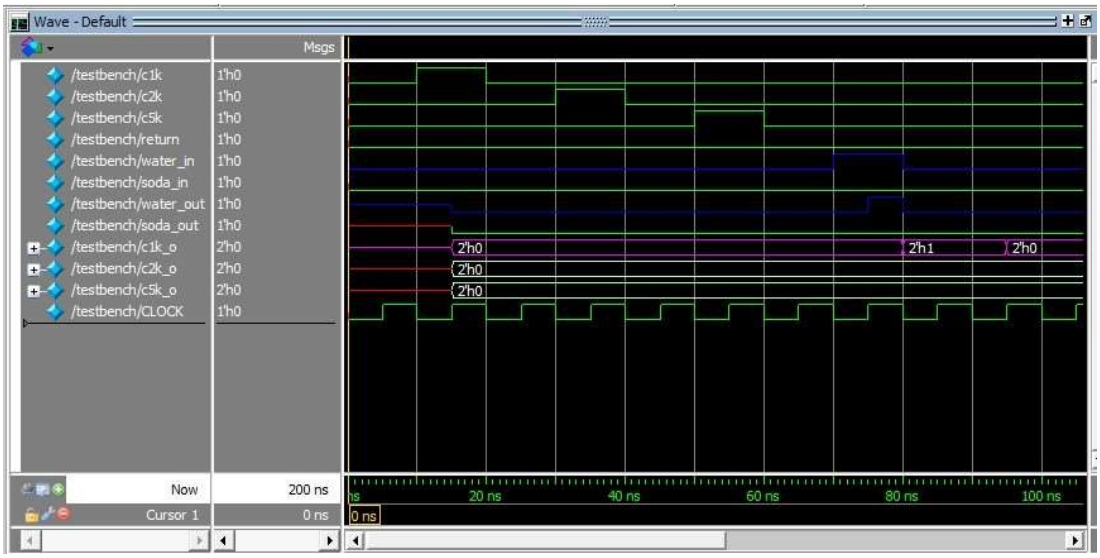
- State Init: Tất cả output bằng 0, máy trong trạng thái chờ.
- State Coin 1: Số tiền máy nhận được nhỏ hơn 7000.
- State Coin 2: Số tiền máy nhận được trong khoảng 7000 đến 9000.
- State Coin 3: Số tiền máy nhận được lớn hơn 9000. Nếu đưa thêm tiền vào máy sẽ tự động trả lại số tiền thừa.
- State Water: Máy trả về tín hiệu Water (đưa nước cho người mua) và trả về số tiền thừa.
- State Soda: Máy trả về tín hiệu Soda (đưa soda cho người mua) và trả về số tiền thừa.
- State Return: Máy trả lại toàn bộ số tiền đã nhận nếu người mua bấm Coin Return.



4. Kết quả mô phỏng:

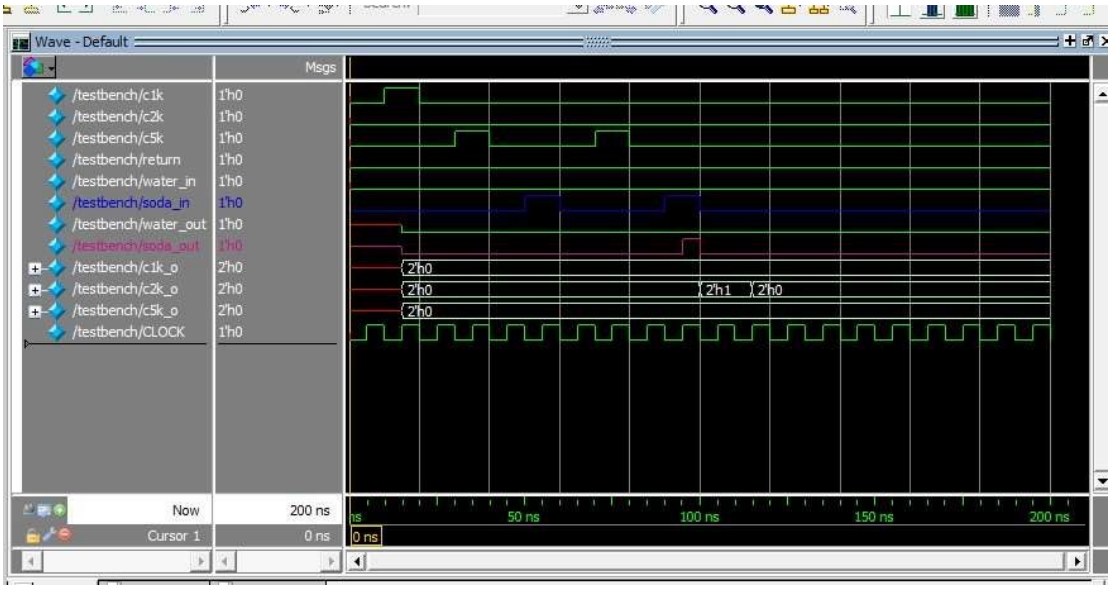
Do mô hình máy bán hàng tự động đòi hỏi hoạt động trong nhiều trường hợp khác nhau. Để thuận tiện cho việc mô phỏng chúng ta sẽ tiến hành giả đinh các trường hợp cụ thể.

Trường hợp 1: Mua nước lọc: đưa vào lần lượt 1k -> 2k -> 5k -> bấm water_in



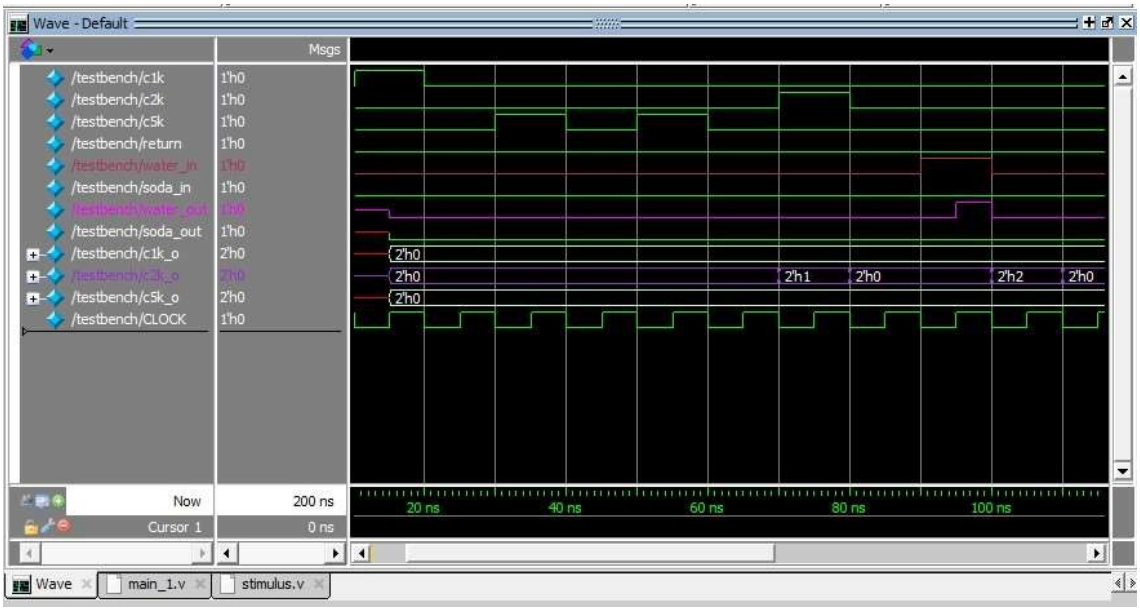
Nhận xét: sau khi water_in được bấm và số tiền bỏ vào bằng 8k >7k. Thỏa điều kiện để lấy được nước. khi đó water_out = 1 và số tiền trả lại là 1k.

Trường hợp 2: đưa vào lần lượt : 1k -> 5k-> chọn soda -> 5k-> chọn soda lần nữa



Nhận xét: khi bấm chọn soda lần đầu tiên, máy nhận thấy số tiền tối thiểu bỏ vào là chưa đủ (6k) nên sẽ không xuất soda_out = 1 và tiếp tục ở trạng thái chờ. Sau khi tiếp tục đưa thêm 5k (money = 11k) và bấm chọn soda. Tại lúc này đã thỏa điều kiện -> soda_out = 1 và trả lại số tiền thừa 11k – 9k = 2k.

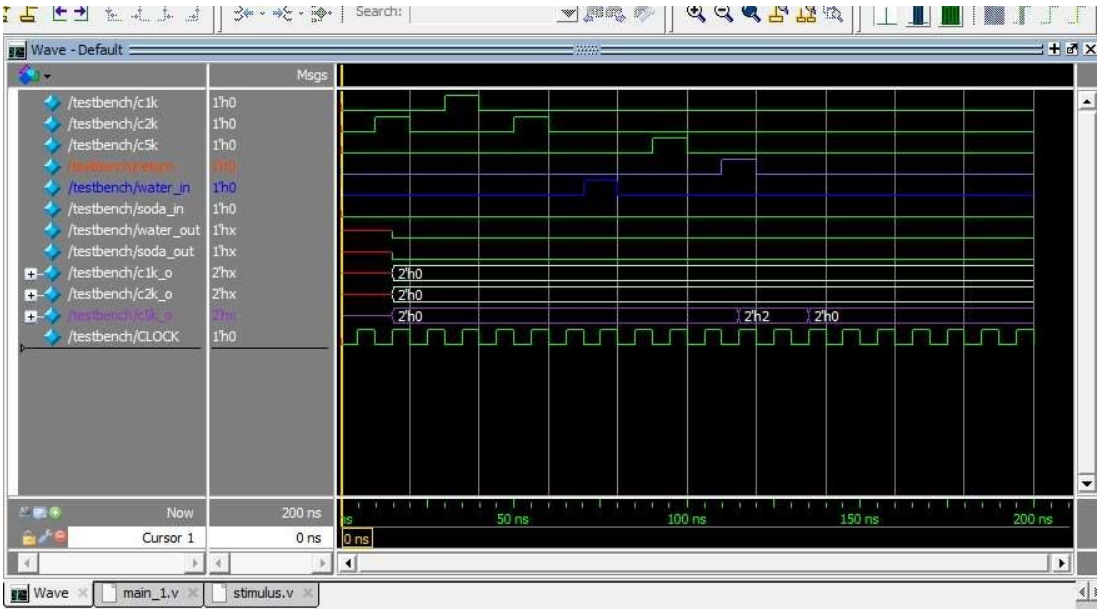
Trường hợp 3: Đưa vào 1k -> 5k -> 5k->2k-> chọn mua water



Nhận xét: Ngay tại thời điểm trươc khi đưa 2k vào máy. Hiện máy đang có money = 11k và đang đợi xem người mua chọn input water, soda hoặc return. Tuy nhiên sau đó người mua lại tiếp tục đưa thêm 2k vào máy -> số tiền vượt quá mức dư của máy -> máy trả lại tờ tiền vừa đưa vào trước đó là 2k. Sau đó người mua chọn mua nước. Máy lại hoạt động bình thường cho water_out = 1 và số tiền trả lại là 4k = 2x2k.

Trường hợp 4: người mua không muốn mua nước và muốn nhận lại tiền:

Đưa vào máy 2k -> 1k -> 2k ->mua nước-> 5k -> nhấn CoinReturn;



Khi người đưa vào 5k và muốn mua nước nhưng không đủ số dư nên tiếp tục đưa thêm 5k. Nhưng sau đó lại thay đổi ý đinh và lấy lại tiền. Hiện tại

trong máy có money = 10k. khi CoinReturn = 1 thì máy trả về 10k cho người mua với mệnh giá 10=2x5k.

5. Kết luận:

Chương trình được thiết kế có thể hoàn thành tốt các yêu cầu đặt ra.

Đoạn code tuy còn dài nhưng nhìn chung vẫn đáp ứng được tất cả các trường hợp có thể xảy ra trong thực tế.

Phụ lục

1. Source code LAB 1: module *full_adder*

```
module full_adder(sum, c_out, a, b, c_in);
output [3:0] sum;
output c_out;
input [3:0] a,b;
input c_in;
wire p0,g0, p1,g1, p2,g2, p3,g3;
wire c4, c3, c2, c1;
assign   p0 = a[0] ^ b[0],
         p1 = a[1] ^ b[1],
         p2 = a[2] ^ b[2],
         p3 = a[3] ^ b[3];
assign  g0 = a[0] & b[0],
         g1 = a[1] & b[1],
         g2 = a[2] & b[2],
         g3 = a[3] & b[3];
assign  c1 = g0 | (p0 & c_in),
         c2 = g1 | (p1 & g0) | (p1 & p0 & c_in),
         c3 = g2 | (p2 & g1) | (p2 & p1 & g0) | (p2 & p1 & p0
& c_in),
         c4 = g3 | (p3 & g2) | (p3 & p2 & g1) | (p3 & p2 & p1
& g0) | (p3 & p2 & p1 & p0 & c_in);
assign  sum[0] = p0 ^ c_in,
         sum[1] = p1 ^ c1,
         sum[2] = p2 ^ c2,
         sum[3] = p3 ^ c3;
assign c_out = c4;
endmodule
```

2. Source code LAB 2: module *alu*

```
module alu(cin,m,s,a,b,co,f,carryout);
input [3:0] a,b;
input [1:0] s;
input m,cin;
```

```
output [3:0] f,co;
output carryout;
alu_1bit alu_1bit_0(a[0],b[0],s,m,cin, f[0],co[0]);
alu_1bit alu_1bit_1(a[1],b[1],s,m,co[0],f[1],co[1]);
alu_1bit alu_1bit_2(a[2],b[2],s,m,co[1],f[2],co[2]);
alu_1bit alu_1bit_3(a[3],b[3],s,m,co[2],f[3],co[3]);
assign carryout = co[3];
endmodule
module alu_1bit(x,y,sel,m,c0,f1,c2);
input x,y,m,c0;
input [1:0] sel;
output c2,f1;
reg [1:0] temp;
reg g,h,c2,f1;
always @(x or y or sel or m or c0)
begin
temp = 0;
case (sel)
0: begin temp = x+c0; h = x & y; end
1: begin temp = x+y+c0; h = x | y; end
2: begin temp = x+(!y)+c0; h = x ^ y; end
3: begin temp = (!x)+y+c0; h = x ~^ y; end
endcase
c2=temp[1];
g=temp[0];
case (m)
0: begin f1= h; g=0; c2=0;end
1: begin f1= g; h=0; end
endcase
end
endmodule
```

3. Source code LAB 3: module *FSM*

```
module FSM(clk, reset, N, D, Q, coin_return, water_in, soda_in,
water_out, soda_out, NO, DO,
```

```
QO);
//Inputs
input wire clk, reset;
input N, D, Q, coin_return, water_in, soda_in;
//Outputs
output reg water_out, soda_out;
output reg NO, DO, QO;
integer sum = 0;
reg [2:0] currentState = 0 , nextState = 0;
parameter STATE_INIT = 3'd0,
STATE_COIN1 = 3'd1,
STATE_COIN2 = 3'd2,
STATE_COIN3 = 3'd3,
STATE_WATER = 3'd4,
STATE_SODA = 3'd5,
STATE_RETURN = 3'd6;
//----------------------------------------
always @(posedge clk )
begin
if(reset) currentState <= STATE_INIT ;
else currentState <= nextState ;
end
//----------------------------------------------------------
always @(N or D or Q) begin
nextState = currentState ;
case (currentState)
STATE_INIT: begin
sum =0 ;
water_out = 0 ; soda_out = 0; NO = 0; DO = 0; QO = 0;
if (N) begin sum = 1; nextState =
STATE_COIN1; end
else if (D) begin sum = 2; nextState = STATE_COIN1; end
else if (Q) begin sum = 5; nextState = STATE_COIN1; end
end
STATE_COIN1:begin
```

```
water_out = 0 ; soda_out = 0; NO = 0; DO = 0; QO = 0;
if (N) sum = sum + 1;
else if(D) sum = sum + 2;
else if (Q) sum = sum + 5;
if (coin_return) nextState = STATE_RETURN;
if ((sum >= 7)&( sum < 9)) nextState = STATE_COIN2;
else if (sum >= 9) nextState = STATE_COIN3;
end
STATE_COIN2: begin
water_out = 0 ; soda_out = 0; NO = 0; DO = 0; QO = 0;
if (water_in) nextState = STATE_WATER;
else if (N) sum = sum + 1;
else if (Q) sum = sum + 5;
else if (D) sum = sum + 2;
if (sum >= 9) nextState = STATE_COIN3 ;
if (coin_return) nextState = STATE_RETURN ;
end
STATE_COIN3: begin
water_out = 0 ; soda_out = 0;
if (N) NO = 1;
else if (D) DO = 1;
else if (Q) QO = 1;
if (water_in) nextState = STATE_WATER;
else if (soda_in) nextState = STATE_SODA;
else if (coin_return) nextState = STATE_RETURN;
end
STATE_WATER: begin
water_out = 1; soda_out = 0;
case(sum - 7)
5: begin QO = 2'd1; end
4: begin DO = 2'd2; end
3: begin DO = 2'd1; NO = 2'd1 ; end
2: begin DO = 2'd1; end
1: begin NO = 2'd1; end
default : begin QO = 2'd0; DO = 2'd0; NO = 2'd0 ; end
```

```
endcase
nextState = STATE_INIT;
end
STATE_SODA: begin
soda_out = 1; water_out = 0;
case ( sum - 9)
4: begin DO = 2'd2; end
3: begin DO = 2'd1; NO = 2'd1; end
2: begin DO = 2'd1; end
1: begin NO = 2'd1; end
default : begin QO = 2'd0; DO = 2'd0; NO = 2'd0 ; end
endcase
nextState = STATE_INIT ;
end
STATE_RETURN: begin
case (sum)
9 : begin QO = 2'd1; DO = 2'd2; end
8 : begin QO = 2'd1; DO = 2'd1; NO = 2'd1; end
7 : begin QO = 2'd1; DO = 2'd1; end
6 : begin QO = 2'd1; NO = 2'd1; end
5 : begin QO = 2'd1; end
4 : begin DO = 2'd2; end
3 : begin DO = 2'd1; NO=2'd1; end
2 : begin DO = 2'd1 ; end
1 : begin NO = 2'd1 ; end
default: begin DO = 0; NO = 0 ; end
endcase
nextState = STATE_INIT ;
end
endcase
end
always @(currentState ) begin
case(currentState)
STATE_INIT : begin
sum =0 ;
```

```
water_out = 0 ; soda_out = 0; NO = 0; DO = 0; QO = 0; end
STATE_COIN1 : begin
water_out = 0 ; soda_out = 0; NO = 0; DO = 0; QO = 0; end
STATE_COIN2 : begin
water_out = 0 ; soda_out = 0; NO = 0; DO = 0; QO = 0; end
STATE_COIN3 : begin
water_out = 0 ; soda_out = 0; NO = 0; DO = 0; QO = 0; end
STATE_WATER : begin
water_out = 1 ;
case(sum - 7)
5: begin QO = 2'd1; end
4: begin DO = 2'd2; end
3: begin DO = 2'd1; NO = 2'd1 ; end
2: begin DO = 2'd1; end
1: begin NO = 2'd1; end
default : begin QO = 2'd0; DO = 2'd0; NO = 2'd0 ; end
endcase
end
STATE_SODA: begin
soda_out = 1;
case ( sum - 9)
4: begin DO = 2'd2; end
3: begin DO = 2'd1; NO = 2'd1; end
2: begin DO = 2'd1; end
1: begin NO = 2'd1; end
default : begin QO = 2'd0; DO = 2'd0; NO = 2'd0 ; end
endcase
end
STATE_RETURN: begin
if (sum >= 10) begin QO = 2'd2; sum = sum - 10; end
case (sum)
4 : begin DO = 2'd2; end
3 : begin DO = 2'd1; NO=2'd1; end
2 : begin DO = 2'd1 ; end
1 : begin NO = 2'd1 ; end
```

```
default: begin DO = 0; NO = 0 ; end
endcase
end
endcase
end
endmodule
```