**ĐẠI HỌC BÁCH KHOA THÀNH PHỐ HỒ CHÍ MINH**

**Khoa Điện - Điện tử Bộ môn Điện tử**



**KỸ THUẬT SỐ NÂNG CAO**

**BÁO CÁO LAB – PHẦN 1**

**GVHD: TS. Trần Hoàng Linh**

**SINH VIÊN: Phạm Xuân Thi - 1814120**

**Trần Văn Thành - 1814099**

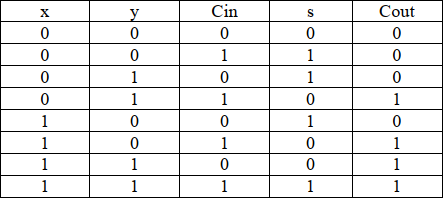
**Đặng Xuân Khu**

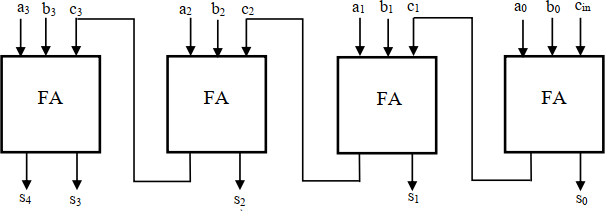
Tp. Hồ Chí Minh, 15 tháng 6 năm 2021

# LAB 1: THIẾT KẾ BỘ CỘNG 2 SỐ 4 BIT

## Mục tiêu

* + Thiết kế bộ cộng 2 số 4 bits bằng 2 cách:
    - Sử dụng bộ cộng toàn phần (full adder) để thực hiện (structural model)
    - Sử dụng mô tả hành vi để thực hiện (behavior model)

1. **Lý thuyết hoạt động của mạch**
   * Bộ cộng full adder bao gồm ba ngõ vào x,y và ci và có hai ngõ ra là kết quả tổng s và bit nhớ cout. Bảng sự thật của bộ cộng full adder được cho ở bảng sau:
   * Từ bảng sự thật có thể xác định được hàm của s và co, theo x,y và Ci như sau: s = x xor y xor ci ; co = (x&y) | (y&ci) | (y&ci).
   * Như vậy để thực hiện bộ cộng 2 số 4 bit dùng bộ cộng full adder ta thực hiện theo sơ đồ như sau:



1. **Code**

module adder4bit1(s,co,a,b); input [3:0] a,b;

output [3:0] s; output co; wire n1,n2,n3;

fulladder1 N1(s[0],n1,a[0],b[0],0);

fulladder1 N2(s[1],n2,a[1],b[1],n1);

fulladder1 N3(s[2],n3,a[2],b[2],n2);

fulladder1 N4(s[3],co,a[3],b[3],n3); endmodule

module fulladder1(s,co,x,y,ci); input x,y,ci;

output s,co; assign s=x^y^ci;

assign co=(x&y)|(x&ci)|(y&ci); endmodule

## Testbench

module testbench(); reg [3:0] a,b;

wire [3:0] s; wire co;

adder4bit1 U1(s,co,a,b); initial

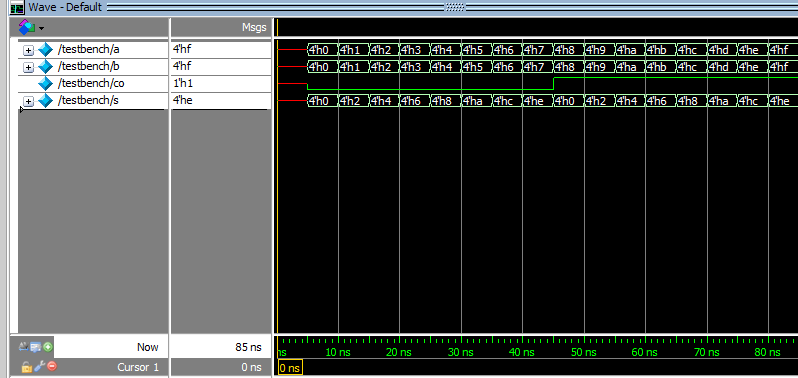
begin

#5 a = 4'b0000; b= 4'b0000; #5 a = 4'b0001; b= 4'b0001; #5 a = 4'b0010; b= 4'b0010; #5 a = 4'b0011; b= 4'b0011; #5 a = 4'b0100; b= 4'b0100; #5 a = 4'b0101; b= 4'b0101; #5 a = 4'b0110; b= 4'b0110; #5 a = 4'b0111; b= 4'b0111; #5 a = 4'b1000; b= 4'b1000; #5 a = 4'b1001; b= 4'b1001; #5 a = 4'b1010; b= 4'b1010; #5 a = 4'b1011; b= 4'b1011;

#5 a = 4'b1100; b= 4'b1100; #5 a = 4'b1101; b= 4'b1101; #5 a = 4'b1110; b= 4'b1110; #5 a = 4'b1111; b= 4'b1111;

#5 $finish; end endmodule

## Dạng sóng



* + Bảng sự thật cho một số trường hợp

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| x | y | ci | S | co |
| 0000 | 0000 | 0 | 0000 | 0 |
| 0001 | 0001 | 0 | 0010 | 0 |
| 0010 | 0010 | 0 | 0100 | 0 |
| 0011 | 0011 | 0 | 0110 | 0 |
| 0100 | 0100 | 0 | 1000 | 0 |
| 0101 | 0101 | 0 | 1010 | 0 |
| 0110 | 0110 | 0 | 1100 | 0 |
| 0111 | 0111 | 0 | 1110 | 0 |
| 1000 | 1000 | 0 | 0000 | 1 |
| 1001 | 1001 | 0 | 0010 | 1 |
| 1010 | 1010 | 0 | 0100 | 1 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1011 | 1011 | 0 | 0110 | 1 |
| 1100 | 1100 | 0 | 1000 | 1 |
| 1101 | 1101 | 0 | 1010 | 1 |
| 1110 | 1110 | 0 | 1100 | 1 |
| 1111 | 1111 | 0 | 1110 | 1 |

* + Nhận xét: Từ kết quả dạng sóng ngõ ra sau khi mô phỏng và kết quả trong bảng sự thật ta thấy hai kết quả giống nhau

1. **Kết luận**

Bộ cộng 2 số 4 bit dùng fulladder thiết kế cho kết quả đúng với bảng sự thật.

1. **Code Behavior**

module adder4bit(s,co,a,b); input [3:0] a,b;

output [3:0] s; output co; wire n1,n2,n3;

adder N1(s[0],n1,a[0],b[0],0);

adder N2(s[1],n2,a[1],b[1],n1);

adder N3(s[2],n3,a[2],b[2],n2);

adder N4(s[3],co,a[3],b[3],n3); endmodule

module adder(s,co,a,b,ci); input a,b,ci;

output s,co; reg s,co;

always @(a or b or ci)

if (a==1'b0 && b==1'b0 && ci==1'b0) begin

s=0;

co=0; end else

if (a==1'b0 && b==1'b0 && ci==1'b1) begin

s=1;

co=0; end else

if (a==1'b0 && b==1'b1 && ci==1'b0) begin

s=1;

co=0; end else

if (a==1'b0 && b==1'b1 && ci==1'b1) begin

s=0;

co=1; end else

if (a==1'b1 && b==1'b0 && ci==1'b0) begin

s=1;

co=0; end else

if (a==1'b1 && b==1'b0 && ci==1'b1) begin

s=0;

co=1; end else

if (a==1'b1 && b==1'b1 && ci==1'b0) begin

s=0;

co=1; end else

if (a==1'b1 && b==1'b1 && ci==1'b1) begin

s=1;

co=1; end

endmodule

## Testbench\_1

module testbench\_1(); reg [3:0] a,b;

wire [3:0] s; wire co;

adder4bit U1(s,co,a,b); initial

begin

#5 a = 4'b0000; b= 4'b0000; #5 a = 4'b0001; b= 4'b0001; #5 a = 4'b0010; b= 4'b0010; #5 a = 4'b0011; b= 4'b0011; #5 a = 4'b0100; b= 4'b0100; #5 a = 4'b0101; b= 4'b0101; #5 a = 4'b0110; b= 4'b0110; #5 a = 4'b0111; b= 4'b0111; #5 a = 4'b1000; b= 4'b1000; #5 a = 4'b1001; b= 4'b1001; #5 a = 4'b1010; b= 4'b1010; #5 a = 4'b1011; b= 4'b1011; #5 a = 4'b1100; b= 4'b1100; #5 a = 4'b1101; b= 4'b1101; #5 a = 4'b1110; b= 4'b1110; #5 a = 4'b1111; b= 4'b1111;

#5 $finish;

end endmodule

## Dạng sóng

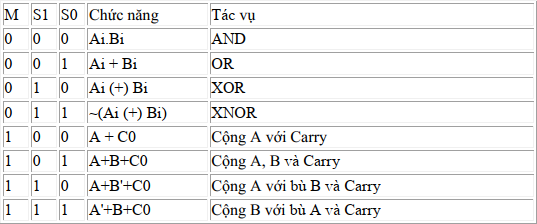


1. **Nhận xét**

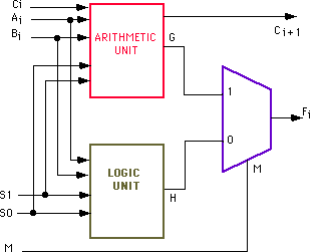
Kết quả của code dùng behavior giống với kết quả của code dùng Structual và giống với kết quả của bảng sự thật.

Vậy bài thiết kế bộ cộng 2 số 4 bit bằng 2 mô hình thành công.

***LAB 2: THIẾT KẾ BỘ ALU-4BIT***

1. **Mục đích**
   * Bộ xử lý logic về toán học (ALU - Arithmetic and Logic Unit) 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 các ngõ nhập function-select. Mục đích của bài thí nghiệm này 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 và S1.
     + Các tác vụ ALU thực hiện được cho trong bảng
2. **Giải pháp**
   * Chia bộ xử lý ALU thành 2 phần: phần A (Arithmetic Unit) và phần L (Logic Unit).
   * 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.
   * Cách này có tính sử dụng lại rất cao và được dùng khá nhiều trong kỹ thuật thiết kế phần cứng, chẳng hạn thiết kế bộ nhớ. Để thiết kế một bit- slice, có nhiều cách khác nhau. Một cách có thể là viết bảng sự thật để thiết kế. Bảng này có 6 ngõ nhập (M, S1, S0, C0, Ai, Bi) và hai ngõ xuất là Fi và Ci+1. Cách này nếu viết bằng tay thì khá công phu, nhưng hiệu quả. Trong VHDL (Verilog), có thể hiện thực thiết kế này bằng mô hình cấu trúc. 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.



1. **Code**

*module alu(co,f,ci,a,b,s1,s0,m); input ci,s1,s0,m;*

*input [3:0] a,b; output co; output [3:0] f;*

*wire [3:0] g,h;*

*arithmetic\_4bit u1(co,g,ci,a,b,s1,s0); logic\_4bit u2(h,a,b,s1,s0);*

*mux\_m m1(f,g,h,m); endmodule*

*module logic\_4bit(h,a,b,s1,s0); input [3:0] a,b;*

*input s1,s0; output [3:0] h;*

*logic u3(h[3],a[3],b[3],s1,s0);*

*logic u2(h[2],a[2],b[2],s1,s0);*

*logic u1(h[1],a[1],b[1],s1,s0);*

*logic u0(h[0],a[0],b[0],s1,s0); endmodule*

*module logic(h,a,b,s1,s0); input a,b,s1,s0;*

*output h;*

*assign h = ((~s1)&a&b)|((~s1)&s0&b)|(s0&a&b)|(s1&(~s0)&(~a)&b)|(s1&(~s0)&a&(~b))|( s1&s0&(~a)&(~b))|((~s1)&(s0)&a);*

*endmodule*

*module arithmetic\_4bit(co,g,ci,a,b,s1,s0); input ci,s1,s0;*

*input [3:0] a,b;*

*output [3:0] g; output co;*

*wire [3:1] c1,c2,c3,c4;*

*wire [3:0] g1,g2,g3,g4,c;*

*reg [3:0] g; reg co;*

*arithmetic u00(c1[1],g1[0],ci,a[0],0);*

*arithmetic u01(c1[2],g1[1],c1[1],a[1],0);*

*arithmetic u02(c1[3],g1[2],c1[2],a[2],0);*

*arithmetic u03(c[0],g1[3],c1[3],a[3],0);*

*arithmetic u10(c2[1],g2[0],ci,a[0],b[0]);*

*arithmetic u11(c2[2],g2[1],c2[1],a[1],b[1]);*

*arithmetic u12(c2[3],g2[2],c2[2],a[2],b[2]);*

*arithmetic u13(c[1],g2[3],c2[3],a[3],b[3]);*

*arithmetic u20(c3[1],g3[0],ci,a[0],(~b[0]));*

*arithmetic u21(c3[2],g3[1],c3[1],a[1],(~b[1]));*

*arithmetic u22(c3[3],g3[2],c3[2],a[2],(~b[2]));*

*arithmetic u23(c[2],g3[3],c3[3],a[3],(~b[3]));*

*arithmetic u30(c4[1],g4[0],ci,(~a[0]),b[0]);*

*arithmetic u31(c4[2],g4[1],c4[1],(~a[1]),b[1]);*

*arithmetic u32(c4[3],g4[2],c4[2],(~a[2]),b[2]);*

*arithmetic u33(c[3],g4[3],c4[3],(~a[3]),b[3]);*

*always@ (ci or a or b or s1 or s0) begin*

*if (s1==1'b0 && s0==1'b0) begin*

*g=g1; co=c[0]; end else*

*if (s1==1'b0 && s0==1'b1) begin*

*g=g2; co=c[1]; end else*

*if (s1==1'b1 && s0==1'b0) begin*

*g=g3; co=c[2]; end else*

*if (s1==1'b1 && s0==1'b1) begin*

*g=g4; co=c[3]; end*

*end endmodule*

*module arithmetic(co,g,ci,a,b); input ci,a,b;*

*output co,g;*

*assign g = a^b^ci;*

*assign co=(a&b)|(a&ci)|(b&ci);*

*endmodule*

*module mux\_m(out,m1,m0,select); input select;*

*input [3:0] m1,m0;*

*output [3:0] out;*

*assign out = select ? m1 : m0; endmodule*

## Testbench()

module testbench(); reg ci,s1,s0,m; reg [3:0] a,b; wire co;

wire [3:0] f;

alu u(co,f,ci,a,b,s1,s0,m); initial

begin

#5 m=0; s1=0; s0=0; a=4'b1010; b=4'b0101; ci=0; #5 s0=1;

#5 s1=1; s0=0;

#5 s0=1;

#5 m=1; s1=0; s0=0; #5 s0=1;

#5 s1=1; s0=0;

#5 s0=1;

#5 m=0; s1=0; s0=0; a=4'b1100; b=4'b1100; ci=1; #5 s0=1;

#5 s1=1; s0=0;

#5 s0=1;

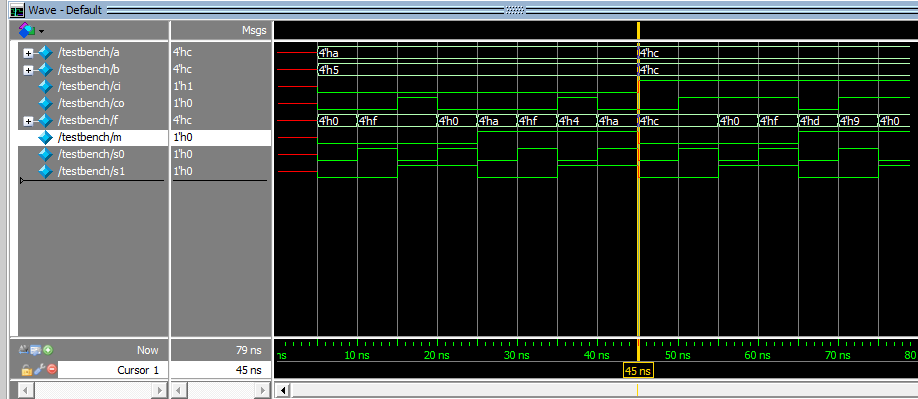
#5 m=1; s1=0; s0=0; #5 s0=1;

#5 s1=1; s0=0;

end

endmodule

## Dạng sóng



* + Bảng sự thật cho một số trường hợp:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| M | S1 | S0 | A | B | Ci | F | Co |
| 0 | 0 | 0 | 1010 | 0101 | 0 | 0000 | 0 |
| 0 | 0 | 1 | 1010 | 0101 | 0 | 1111 | 0 |
| 0 | 1 | 0 | 1010 | 0101 | 0 | 1111 | 1 |
| 0 | 1 | 1 | 1010 | 0101 | 0 | 0000 | 0 |
| 1 | 0 | 0 | 1010 | 0101 | 0 | 1010 | 0 |
| 1 | 0 | 1 | 1010 | 0101 | 0 | 1111 | 0 |
| 1 | 1 | 0 | 1010 | 0101 | 0 | 0100 | 1 |
| 1 | 1 | 1 | 1010 | 0101 | 0 | 1010 | 0 |
| 0 | 0 | 0 | 1100 | 0011 | 1 | 0000 | 0 |
| 0 | 0 | 1 | 1100 | 0011 | 1 | 1111 | 1 |
| 0 | 1 | 0 | 1100 | 0011 | 1 | 1111 | 1 |
| 0 | 1 | 1 | 1100 | 0011 | 1 | 0000 | 0 |
| 1 | 0 | 0 | 1100 | 0011 | 1 | 1101 | 0 |
| 1 | 0 | 1 | 1100 | 0011 | 1 | 0000 | 1 |
| 1 | 1 | 0 | 1100 | 0011 | 1 | 1001 | 1 |

* + Nhận xét: Kết quả mô phỏng đúng với bảng sự thật.

1. **Kết luận**
   * Bài thiết kế mô hình ALU thành công.

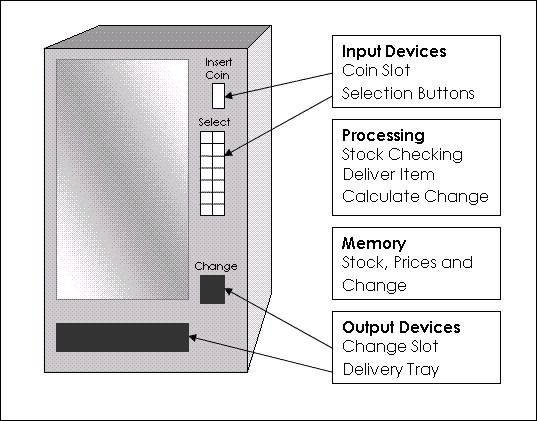
***LAB 3:***

***THIẾT KẾ MÁY BÁN NƯỚC NGỌT TỰ ĐỘNG***

1. **Mục tiêu**
   * 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

o Máy nhận xu: 1000, 2000 và 5000 (in1, in2, in5)

* + - 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)
      * Soda out (SO) (mua nước ngọt)
      * Change (CO) (trả tiền thừa)



## Lựa chọn ngắn gọn giải pháp

* + Sử dụng mô hình cấu trúc mô tả hành vi (behavior), sử dụng nhiều lệnh if\_else và case để giải quyết bài toán.

1. **Source Code HDL**
   * Module vending\_machine:

module vending\_machine(in1,in2,in5,clk,wo,so,cr,co,out1,out2,out5); input [3:0] in1,in2,in5;

input clk,wo,so,cr,co; output [3:0] out1,out2,out5;

reg [3:0] total,out1,out2,out5;

wire [3:0] in1,in2,in5;

reg [3:0] tam,tam1,t;

always @(posedge clk or negedge clk) begin

if (in1\*1+in2\*2+in5\*5 <=9) tam<=in1\*1+in2\*2+in5\*5; if (in1\*1+in2\*2+in5\*5 >9)

begin tam=tam1;

t=in1\*1+in2\*2+in5\*5-tam; case (t)

5: out5=1;

2: out2=1;

1: out1=1; endcase

end

if (wo==1) total<=tam-7; if (so==1) total<=tam-9;

if (wo==0 && so==0) total<=tam; if (cr==1)

begin

case (total) 9:

begin out5=1; out2=2; out1=0; end

8:

begin out5=1; out2=1; out1=1; end

7:

begin out5=1; out2=1; out1=0; end

6:

begin out5=1; out1=1; out2=0; end

5:

begin out5=1; out1=0; out2=0; end

4:

begin out5=0; out2=2; out1=0; end

3:

begin out5=0; out2=1; out1=1; end

2:

begin out5=0; out2=1; out1=0; end

1:

begin out5=0; out2=0; out1=1; end default: begin out5=0; out2=0; out1=0; end endcase end

if (co==1) begin

case (total) 2:

begin out5=0; out2=1; out1=0; end

1:

begin out5=0; out2=0; out1=1; end default: begin out5=0;

out2=0; out1=0; end endcase end

tam1<=tam;

end

endmodule

* + Module test\_vm()

module test\_vm();

reg [3:0] in1,in2,in5; reg clk,wo,so,co,cr;

wire [3:0] out1,out2,out5;

vending\_machine u1(in1,in2,in5,clk,wo,so,cr,co,out1,out2,out5); initial begin

clk=0; in1=0; in2=0; in5=0; wo=0; so=0; cr=0; co=0; #4 in1=1;

#4 in2=1;

#4 in5=1;

#4 in1=2;

#4 in5=2;

#4 so=1;

#4 co=1;

end

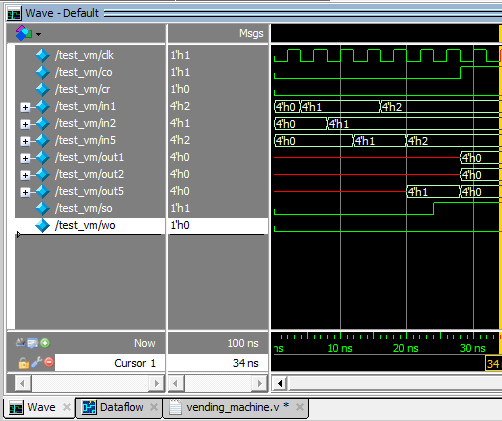
always #2 clk=~clk; endmodule

* + Giải thích thuật toán:

Ý nghĩa của các biến sử dụng trong thuật toán trên tam: lưu tổng tạm thời của số xu bỏ vào tam1: lưu tổng tạm thời sau mỗi xung clock (dùng để so sánh với 9000, yêu cầu là khi số tiền lớn hơn 9000 thì trả lại xu mới nhất) t: lưu giá trị chênh lệch khi số tiền lớn hơn 9000

total: lưu giá trị tiền còn lại khi mua một món nước

1. **Kết quả giả lập**



clk=0; in1=0; in2=0; in5=0; wo=0; so=0; cr=0; co=0; #4 in1=1;

#4 in2=1;

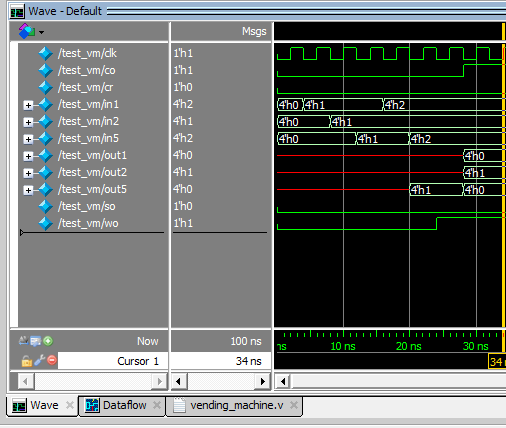
#4 in5=1;

#4 in1=2;

#4 in5=2;

#4 so=1;

#4 co=1;



clk=0; in1=0; in2=0; in5=0; wo=0; so=0; cr=0; co=0; #4 in1=1;

#4 in2=1;

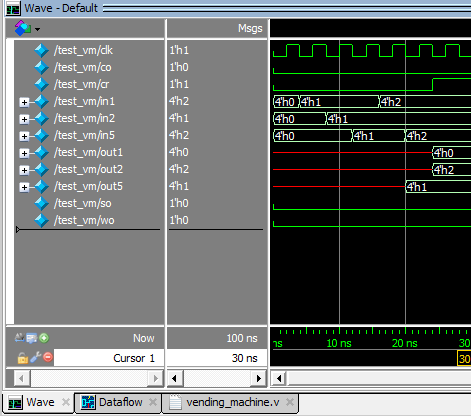
#4 in5=1;

#4 in1=2;

#4 in5=2;

#4 wo=1;

#4 co=1;



clk=0; in1=0; in2=0; in5=0; wo=0; so=0; cr=0; co=0; #4 in1=1;

#4 in2=1;

#4 in5=1;

#4 in1=2;

#4 in5=2;

#4 cr=1;

* + Ở trường hợp 1: Lần lượt bỏ vào xu 1000, 2000, 5000, 1000 và 5000. Thì xu 5000 sẽ bị trả lại vì tổng lớn hơn 9000. Sau đó nhấn nút mua nước ngọt (so) thì số tiền vừa hết. Sau đó nhấn nút (co) thì các ngõ ra bằng 0.
  + Ở trường hợp 2: Lần lượt bỏ vào các xu tương tự, xu 5000 cũng bị trả lại. Sau đó nhấn nút mua nước suối (wo) thì số tiền thừa là 2000. Sau đó nhấn nút (co) thì tiền thừa được trả lại với số lượng xu nhỏ nhất tức là 1 xu 2000.
  + Ở trường hợp 3: Lần lượt bỏ vào các xu tương tự, xu 5000 cũng bị trả lại. Sau đó không mua nước gì mà nhấn nút (cr) thì tiền thừa được trả lại với số lượng xu nhỏ nhất tức là 1 xu 5000 và 2 xu 2000.

1. **Kết luận**
   * Các kết quả chứng minh bài làm đã đúng. Vậy mô hình thiết kế thành công.