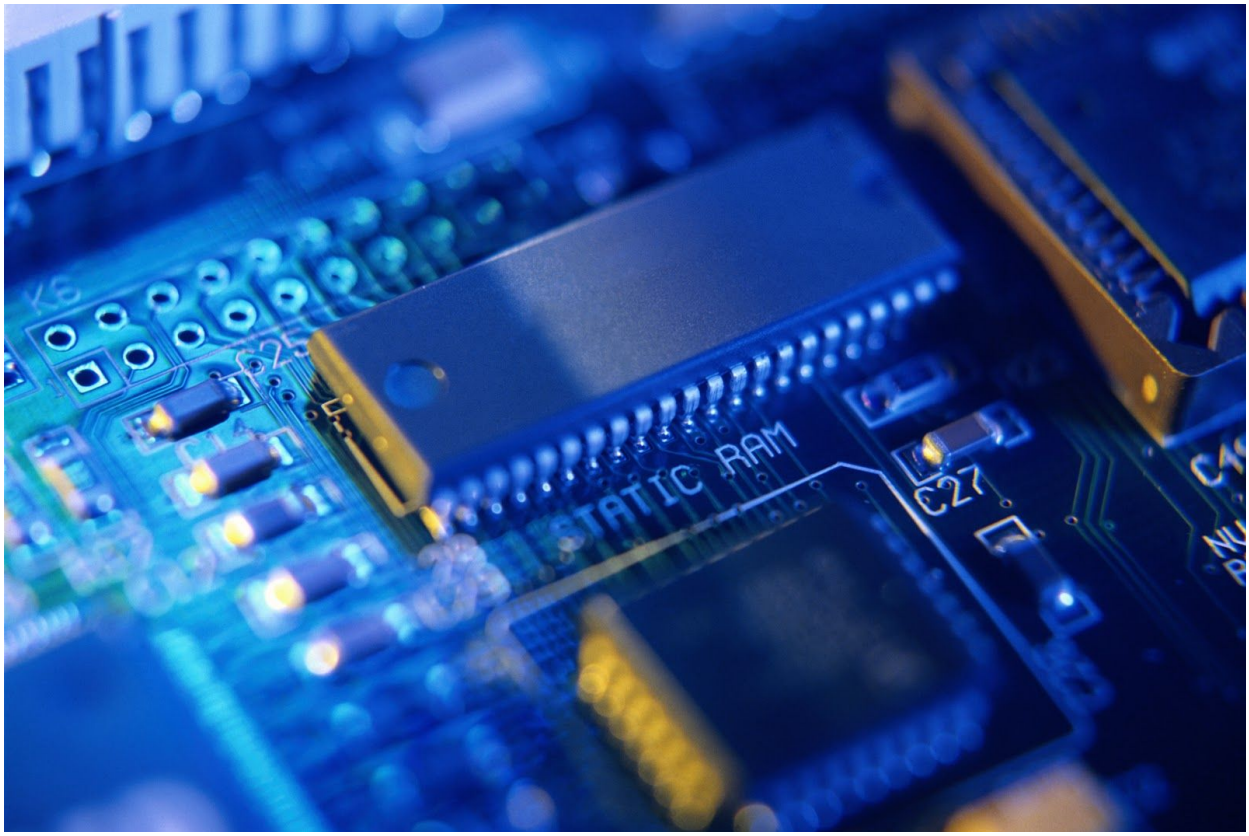


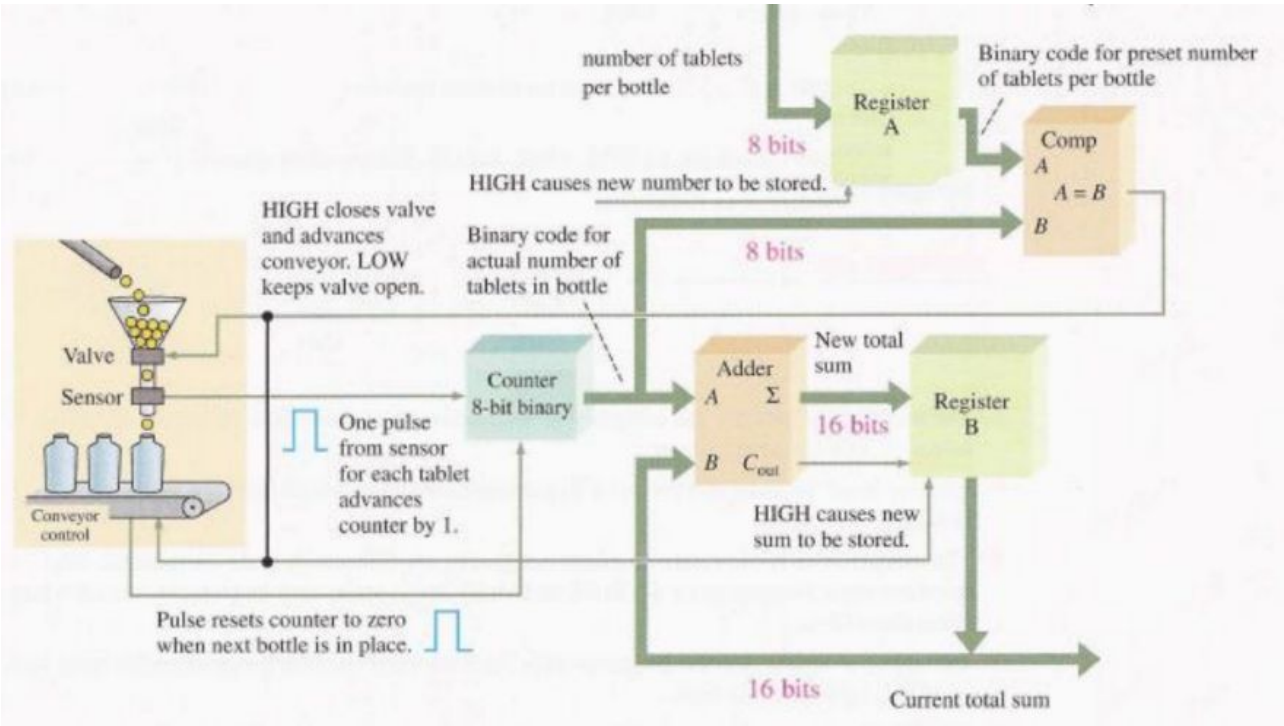
Digital Design Final Project

May, 2016

Team:
Gerardo Cruz Delgado
Juan Salvador
Natalia Almaguer

Overview

To continue developing our abilities of understanding digital design and the implementation of FSM, we were challenged to create a controller for a machine which fills bottles with pills as shown in the next diagram:



In the diagram we can see that our controller consists of the inputs of: Sensor of the pills falling, sensor of the bottle in place, the controls of the machine (Start, Stop and Reset) and the number of pills that will be per bottle. Also we have our outputs: The signal to move the belt, the signal to close the valve of the pills and the sum of all the pills that have been put in the bottles.

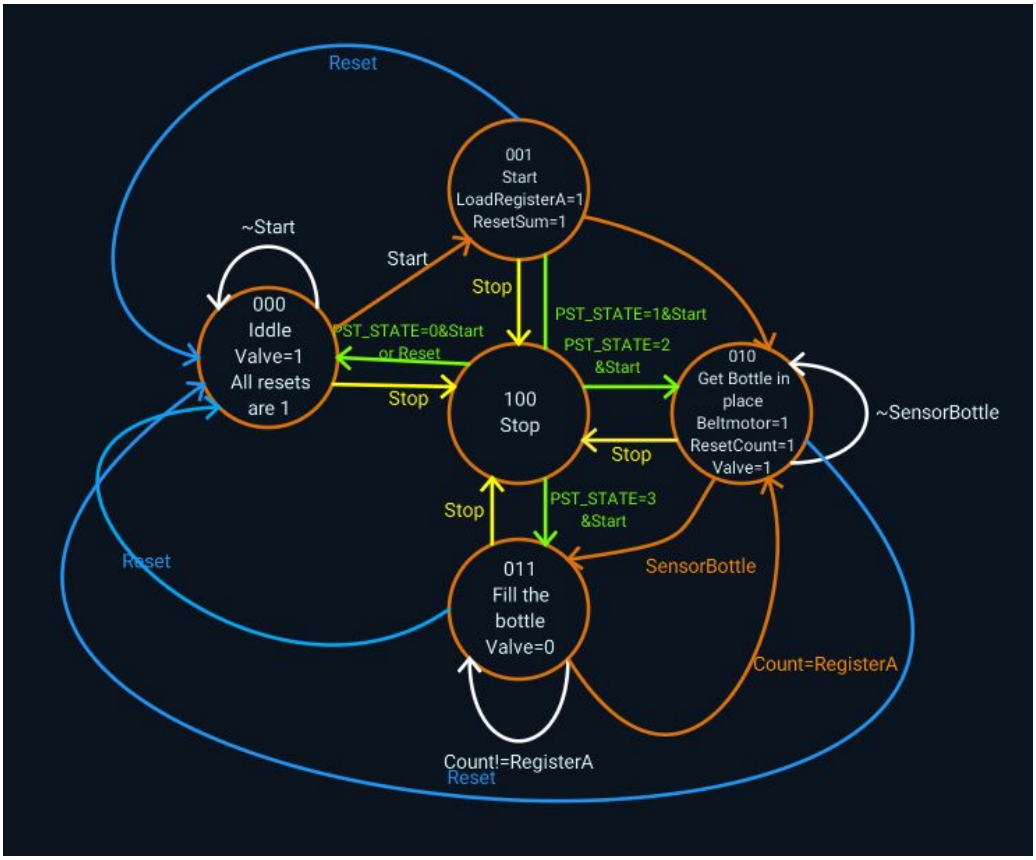
Design of the FSM

At first I decided to analyzed the states table that was included with the instructions, I was confused by some of the things that were proposed in there so I decided to change some of the states of the machine so it can be more realistic and also has some characteristics that are very appreciated in the industry of manufacture like an emergency stop that after you have resolved the problem you can continue working in where it stopped. Trying to

achieve this approach of including some of Lean ideologies in my design, I wrote another table.

State	Name of the State
000	Iddle
001	Start
010	Get bottle
011	Filling bottle
100	Stop

With the table maybe the difference is not that apparent (also that is not the full table that you need to do for designing). You can see the approach I took more clearly in this diagram:



In there after you stopped and press Start, it lets you go back to the previous State. The diagram was a great tool that helped a lot with this project.

Our code in Verilog

When I decided to include the past state in the design, I thought it was gonna be difficult but it was easy, only a little tricky. I create another register where the state that is happening gets saved in case the machine stops, but if it stops doesn't save the standby state. The other parts of the code were pretty straight forward because we have done things like this in class.

Top Module (FSM)

```
module FSM(
   input Rst,
   input Start,
         input Clk, //for fsm state changing
   input Stop,
         input SensorP,//Sensor of the pills
         input SensorB, //Sensor of bottle in place
         input [7:0] NumPill,
   output Valve,
   output [15:0] Sum,
         output BeltMotor
   );
wire Count;
wire ResetCount;
wire [7:0] SumCount;
wire EnAdder;
wire ResetAdder;
wire [7:0] NumPills;
wire ResetRegister;
wire LoadRegister;
assign ResetRegister=(STATE==0);
```

```
assign LoadRegister=(STATE==1);
assign Valve =(STATE!=3);
assign BeltMotor=(STATE==2);
assign ResetAdder=(STATE==0);
assign ResetCount=(STATE==2|STATE==0);
assign EnAdder= (STATE==2);
Register RegisterA
(.Clk(Clk),.Reset(ResetRegister),.A(NumPill),.Load(LoadRegister),.NumPills(NumPills));
Counter CounterA
(.SensorP(SensorP),.Reset(ResetCount),.Compare(NumPills),.Result(Count),.Count(SumCoun
t));
Adder AdderA (.En(EnAdder),.Reset(ResetAdder),.A(SumCount),.Sum(Sum));
reg [2:0] STATE, NXT_STATE,PST_STATE;
always @ * begin
                        case(STATE)
                        3'b000: if (Start)
                                NXT_STATE = 3'b001; //Iddle, valve is closed. Everything else is 0
                        3'b001: NXT_STATE = 3'b010; //Start, load A
                        3'b010: if (SensorB)  //State 2, Move bottle
                        NXT_STATE = 3'b011; //If there is a bottle in place, continue
                          else
                        NXT_STATE=3'b010; //wait if there is not bottle
                        3'b011: if (Count) //State3, getting the pills !open the valve!
                        NXT_STATE = 3'b010; //it has all the pills !close valve!
                          else
                        NXT_STATE= 3'b011; //continue adding
                3'b100: if (Start)
                        NXT_STATE = PST_STATE;      //Emergency stop
                        else
                        NXT_STATE= 3'b100;
```

```
                endcase
        end
        always @ (posedge Clk) begin
                if (Rst)
                STATE <= 0;
                else if (Stop)
                STATE<=4;
                else
                STATE<= NXT_STATE;
        end
always@ (negedge Clk) begin
if (STATE!=4)
PST_STATE<=NXT_STATE; //it takes the state and save ir for posterior use
end
endmodule
```

Adder Module

```
module Adder(
   input [7:0] A,
   input En,
          input Reset,
   output reg [15:0] Sum
   );
reg [15:0] Z;
always @* begin
Z=A+Sum;
if (Reset)
Sum <= 0;
end
always @ (posedge En) begin
```

```
Sum<=Z ;
end
endmodule
```

Register Module

```
module Register(
   input Load,
   input [7:0] A,
         input Clk,
         input Reset,
   output reg [7:0] NumPills
   );
always @ (posedge Clk) begin
if (Load)
NumPills <= A;
else if (Reset)
NumPills <=0;
end
endmodule
```

Counter Module

```
module Counter(
   input SensorP,
   input Reset,
         input [7:0] Compare,
   output Result,
         reg [7:0] Count
   );
```

```
reg [7:0] Z;
assign Result = (Count ==Compare);
always @* begin
Z=Count+1;
if (Reset)
Count <= 0;
end
always @ (posedge SensorP) begin
/*if (Reset)
Count <= 0;
else*/
Count <= Z;
end
endmodule
```

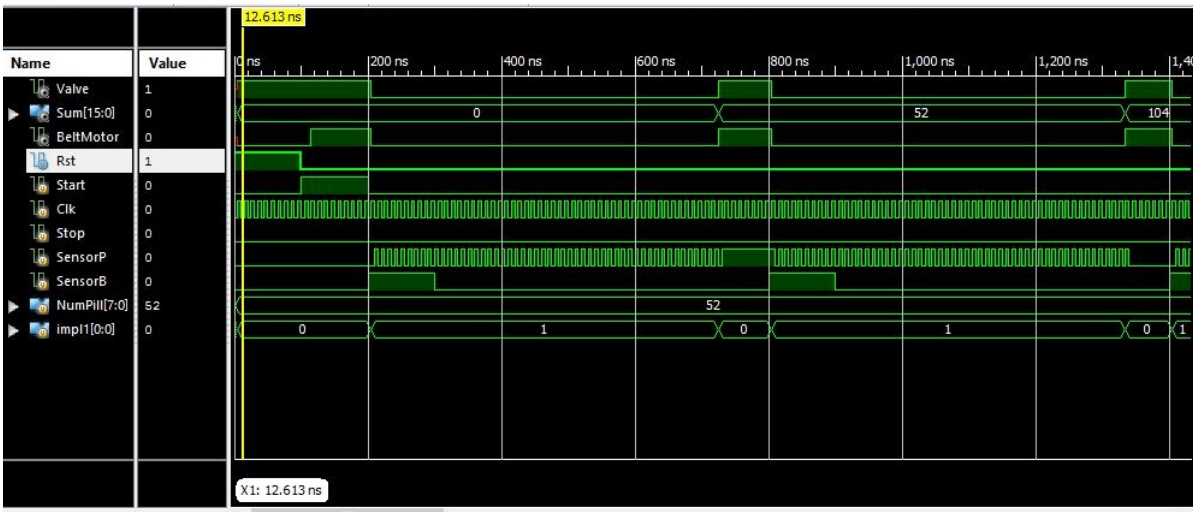
Testing and results

The test was the hardest part, I got in a lot of trouble when trying to make the inputs of the sensors automatic. The good part of what I achieved is that it simulates in a great way the work of the sensors, in the graphs is can be seen how the controller will work in a real scenario. It would have been easier in other ways, but I am really happy of the result.

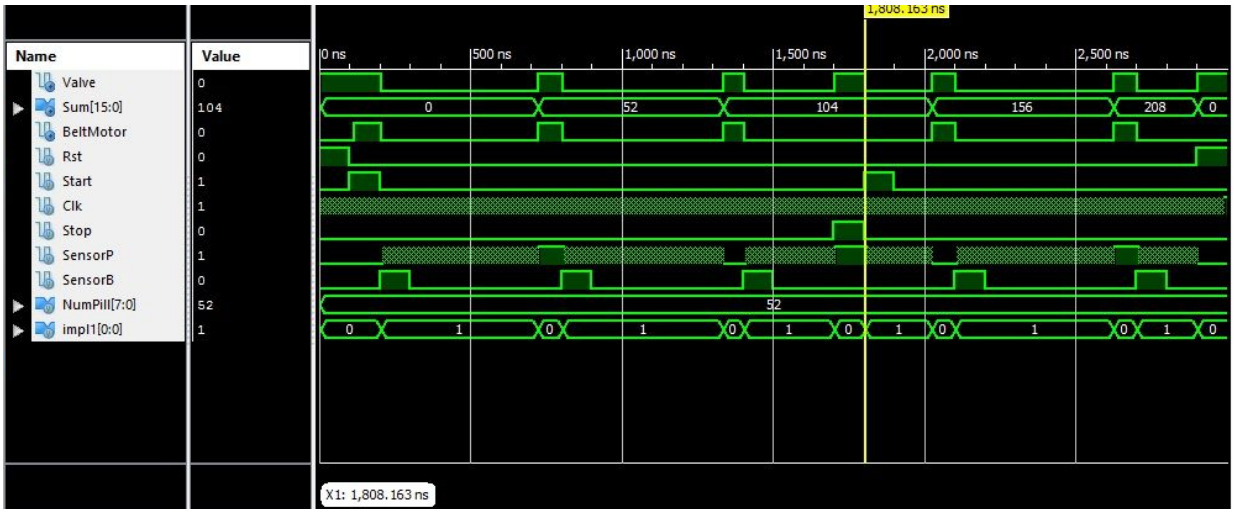
```
module Test3;
        // Inputs
        reg Rst;
        reg Start;
        reg Clk;
        reg Stop;
        reg SensorP;
        reg SensorB;
        reg [7:0] NumPill;
        // Outputs
```

```
wire Valve;
wire [15:0] Sum;
wire BeltMotor;
FSM uut (
        .Rst(Rst),
        .Start(Start),
        .Clk(Clk),
        .Stop(Stop),
        .SensorP(SensorP),
        .SensorB(SensorB),
        .NumPill(NumPill),
        .Valve(Valve),
        .Sum(Sum),
        .BeltMotor(BeltMotor)
);
initial begin
        // Initialize Inputs
        Clk=0;
        SensorB=0;
        SensorP=0;
        Rst = 1;
        NumPill=52;
        Start=0;
        Stop=0;
        #100;
        Rst=0;
        Start = 1;
        #100;
        Start=0;
```

```
                #1500;
                Stop = 1;
                #100;
                Stop=0;
                Start=1;
                #100;
                Start=0;
                #1000;
                Rst=1;
                #100;
        end
        initial begin
                forever #5 Clk = ~Clk;
        end
        always begin
         #100;
         SensorB=BeltMotor; //the simulation of the bottle sensor
        end
always @ (~Valve) begin  //the simulation of the fall of the pills
while(~Valve)
#5 SensorP=~SensorP;
end
endmodule
```



In the image above, it can be seen how the sensors are working and also that the sum, getting the bottles and the filling of them are working in a great way.



In the image above, it can be seen what happens when you stop the machine and press start after. It continues functioning as it was before. And also at the end, there is the reset which makes the machine go iddle.