CS303
Logic and Digital System Design
Project

Egg Timer

Mert Ekici 26772
ekicimert@sabanciuniv.edu

Yaşar Kerem Cimilli 26428
yasarkerem@sabanciuniv.edu

Introduction and Problem Definition

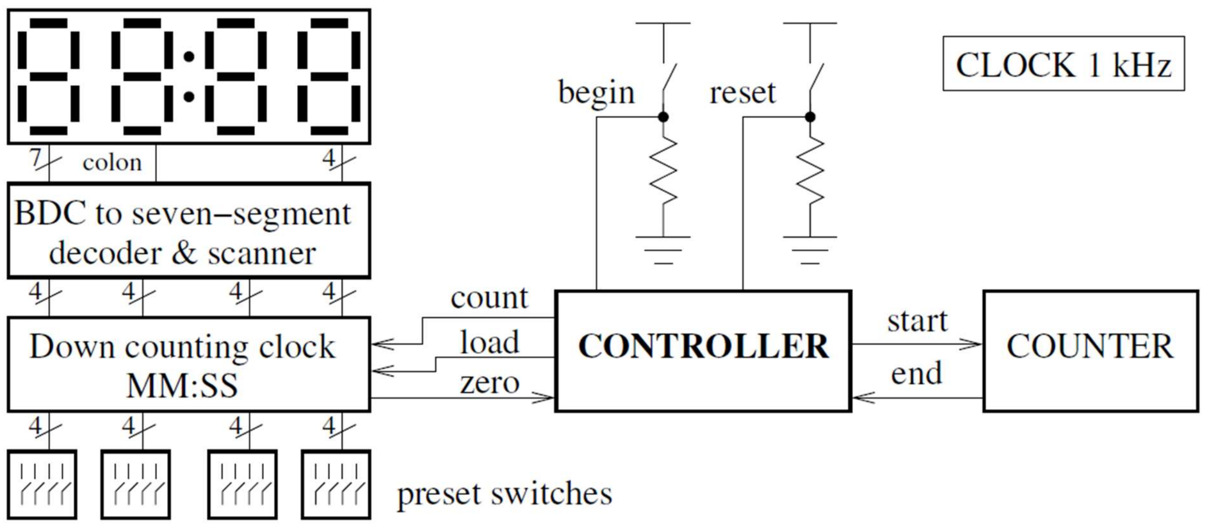

Figure 1: General Structure of Egg Timer

The aim of the project is to design an **egg timer**. In the provided circuit "*eggtimer.dig*", components **downcounter** and **controller** will be designed. As can be seen from Figure 1, the egg timer consists of a BCD to seven-segment decoder, 4 seven-segment displays, down counter, 16 switches for preset inputs, controller (algorithmic state machine) "begin" & "reset" buttons and counter to optimize frequency as 1Hz. When the button **reset** is pressed, the controller loads the preset value to the down-counter which is determined by the switches of the FPGA board (grouped as four 4-bit blocks). The countdown starts immediately when **begin** button is pressed and stops when it reaches to 0.

Downcounter

Downcounter is designed by extending the **cumulative works of previous lab sessions** which gradually helped us learn how to implement a **99.59 downcounter** with the steps below:

- Lab1: Introduction to FPGA board and NAND gates
- Lab2: Seven-segment display
- Lab3: Multiway switch circuit
- Lab4: BCD counter (9-0)
- Lab5: Two digit BCD downcounter (59 – 0)
- Lab6: Project Eggtimer

Thus, the work of Lab5 is simply modified and transformed into **4 digit BCD downcounter with "99:59" limitation**.

Downcounter has **4 smaller down-counters for each digit** and this downcounters have necessary **limits with load inputs** (9 for s1, 5 for s2, 9 for m1 and 9 for m10). However, other possible load values that might appear as a **user input by the 4-bit switches** are also included this time. Both load values (limits and inputs) are connected to a **2-1 multiplexer** and the selection input of the multilexer is **"load" input** which will be connected to the "**reset**" signal of the controller. In general, first s1 counts down and when it reaches zero it signals s10 and decrements it. Using the same methodology, small downcounters obtain a **countdown clock property together** as an ultimate downcounter. (To prevent oscillation errors zero output of downcounter is directly connected to m10-m1-s10-s1 values otherwise signal reaches one tick later.)

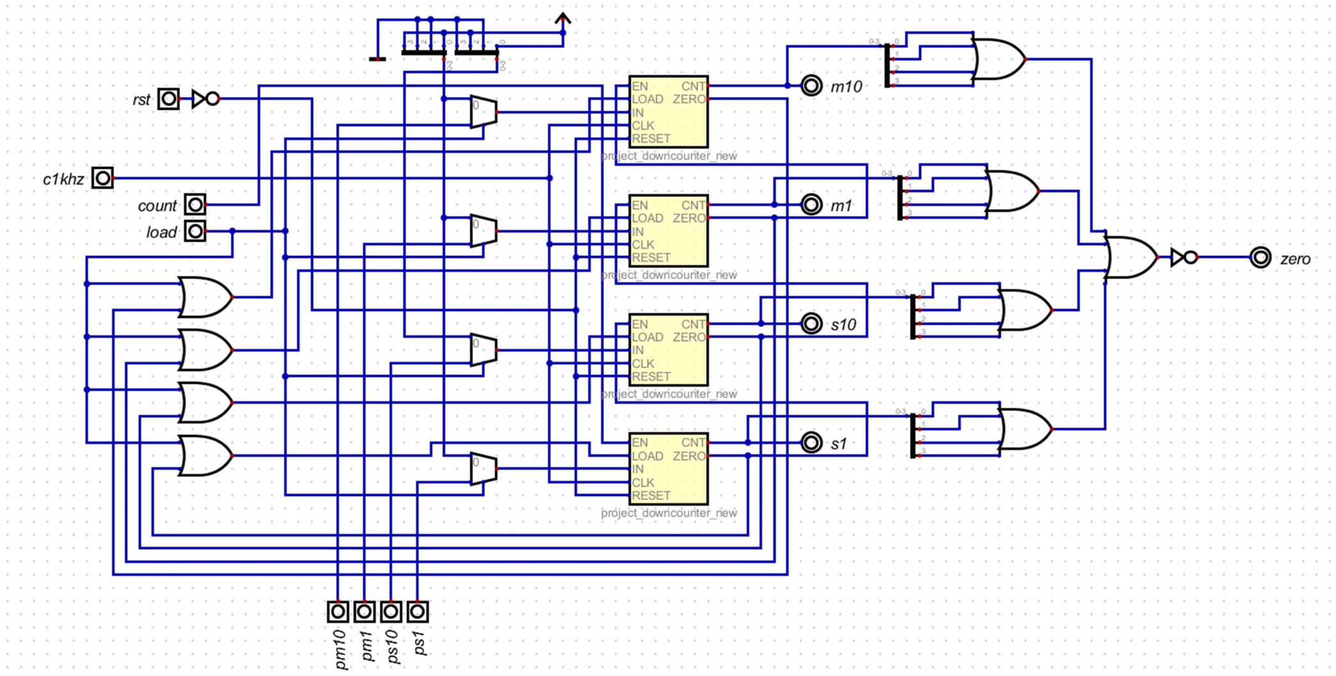

Figure 2: Downcounter

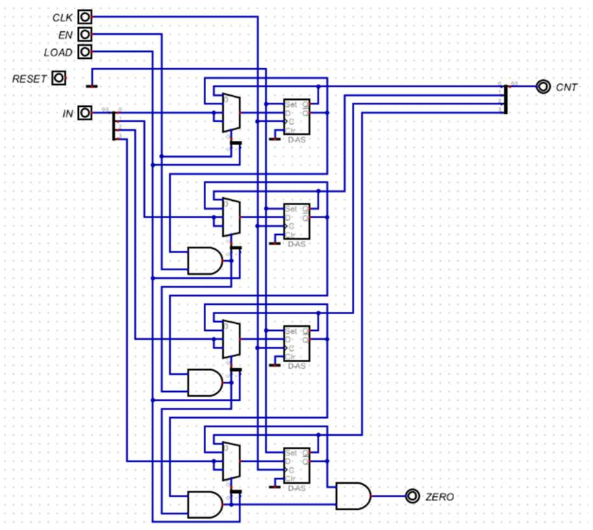

Figure 3: Sub-downcounters

Algorithmic State Machine

Controller is designed as an **algorithmic state machine** that changes its states based on the signals it receives and asserts output signals accordingly. In our case, when "**begin**" button is pressed controller should **start the clock** (if it is not zero) and if "**reset**" is pressed it should **set the value** determined by preset switches. Before starting the programming part of the controller, designing it as an algorithmic state machine (with the help of an ASM chart) is a solid strategy. **ASM chart** in Figure 5 shows the structure of the algorithmic state machine we designed for the controller. It has three states: **PAUSE, START** and **COUNT**. As an initial state, ASM starts with **PAUSE** state. After, it checks **reset signal** and sends **load** signal to **downcounter** if reset is 1. If it is 0, ASM moves to begin signal. Depending on the value of **begin** it enters to **START** state or **PAUSE** state. In the **START** state, controller checks **zero** signal and if it is asserted it goes back to **PAUSE** state, otherwise it sends **start** signal to **counter** (optimizer counter for 1Hz ticks) and moves to **COUNT** state. Finally, **COUNT** state checks **end** signal of **counter** and if it is 1 it goes back to **START** state to send another **start** signal if **zero** is not asserted. If end is 0 then the machine goes to **PAUSE** state again (timer has reached to 0).

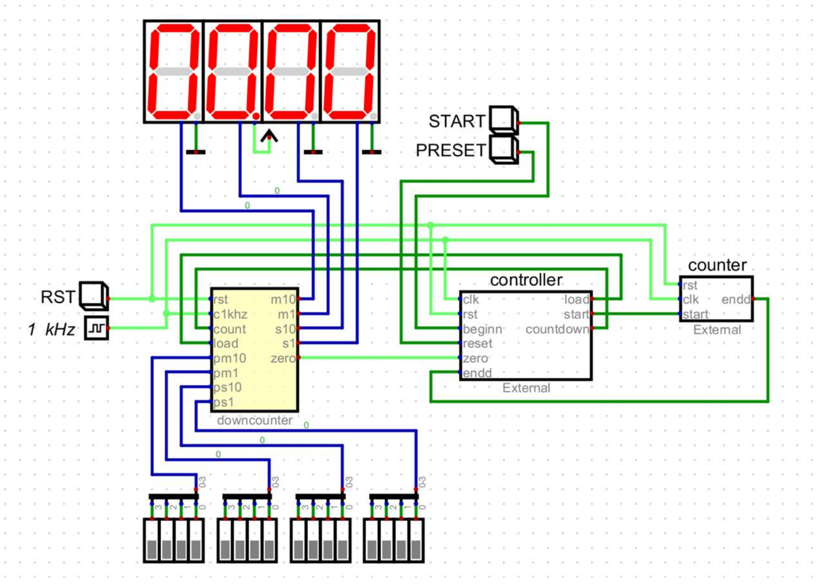

Figure 4: Egg Timer Circuit

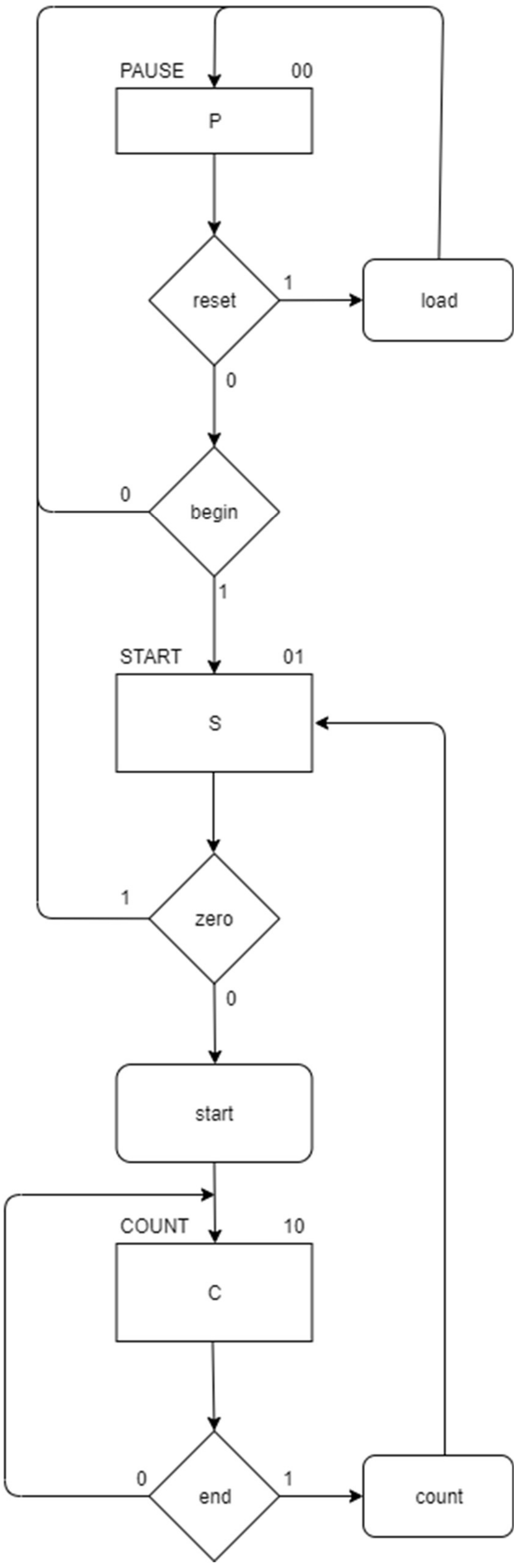

Figure 5: ASM Chart for Controller

Controller

Based on the **algorithmic state machine design** described in the previous section, **controller component** is programmed in **Verilog** which makes it much **easier** than **directly designing the circuit**. The three states of the ASM design are defined as 2-bit register variable **state** ("0" as "PAUSE" state, "1" as "COUNT" state and "2" as "START" state). There are three inputs and three outputs that concern us which are **beginn, reset** and **endd** for inputs & **load, start** and **countdown** for outputs. **Countdown** is assigned to a value depending on **endd**, **start** and **state**. When **endd** is asserted and **start** is not if controller is not in the **PAUSE** state then countdown signal is sent to **downcounter** ("**~start**" is included to avoid **double decrement** problem). **Start** is 1 when state is 2 (**START**) and not 0 (**PAUSE**). **Load** on the other hand, is directly equal to **reset** input. After the assignments (using **assign** function), program starts to **always** part where it executes **always** when **positive edge of clk** input and **negative edge of rst** input is observed. First condition block checks **rst** and if it is 0 **state** is assigned as 0. Otherwise, **algorithmic state machine starts** and behaves as explained in the ASM section. First **reset** value is checked and if it is 1, **state becomes 0** again (**machine pauses**). Load assignment cannot be done in the **always block** since it is defined as a **wire** hence initial assignment of **"load = reset**" would be sufficient. If reset is not 1, we check the state of the algorithmic state machine and move accordingly. In this case, if the state is 0 (**we are still in PAUSE state**), **beginn** signal is checked. If **beginn is pressed**, then depending on **zero** value (if not zero) **start** is sent to **counter** and **countdown** starts or goes back to pause state (if begin is pressed when counter shows **00:00**). If begin is not pressed in PAUSE state, then nothing happens obviously. As the other possibility, if the machine is not in state 0 (not paused) the value of **endd** signal is checked. If **endd** is asserted, machine goes to state 2 (**START**) again to send **start** to **counter** when zero is 0 and goes to state 0 (**PAUSE**) if zero is reached. However; if **endd** is 0 (**endd has not asserted yet**), then machine stays in **COUNT** state (state 1) if not **zero**.

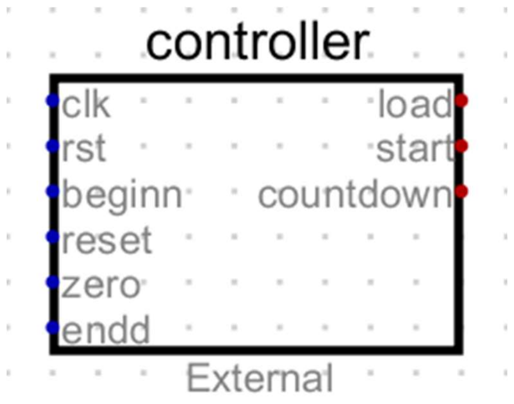

Figure 6: Controller – Inputs and Outputs

Verilog Code

```
module controller(
    input clk,
    input rst,
    input beginn,
    input reset,
    input zero,
    input endd,
    output load,
    output start,
    output countdown
    );

    reg [1:0] state;

    //"0" as "PAUSE" state, "1" as "COUNT" state and "2" as "START" state

    assign countdown = (endd && ~start && state != 0);
    assign start = (state == 2 && state != 0);
    assign load = reset;

    always @(posedge clk, negedge rst)
    begin
      if (~rst) state <= 0;
      else
        begin
         if(reset) state <= 0;
         else
           begin
             if(state == 0)
             begin
                if(beginn)
                begin
                  if(~zero) state <= 2;
                  else state <= 0;
                end
             end
             else
             begin
                if(endd)
                begin
                  if(~zero) state <= 2;
                  else state <= 0;
                end
                else
                begin
                  if(~zero) state <= 1;
                  else state <= 0;
                end
              end
           end
        end
    end //always
endmodule
```