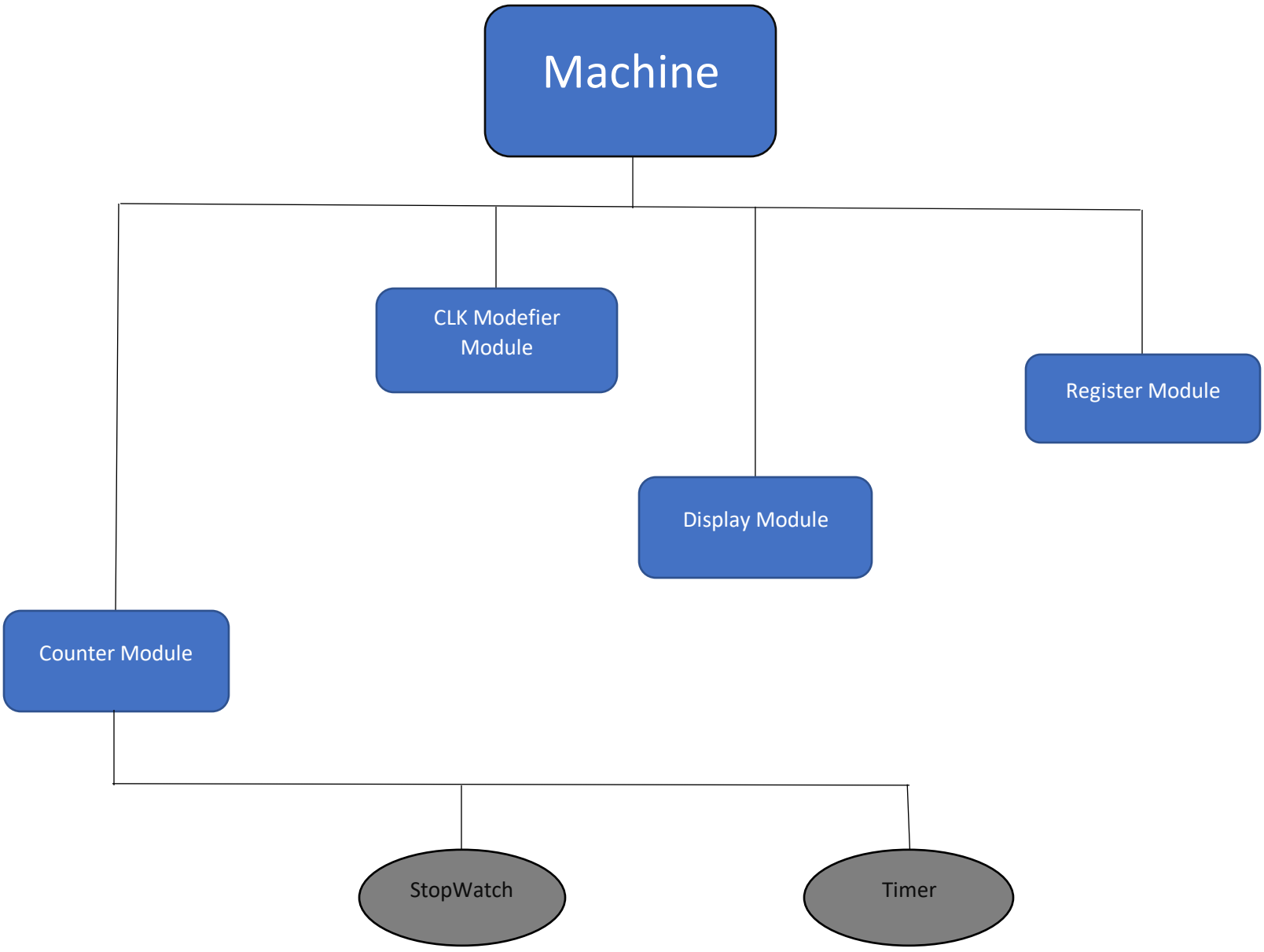
Digital Design And Computer Architecture Project Report One:

TEAM 24		
Ahmed Mohamed Ibrahem	**201902227**	**s-ahmed.ibrahem@zewailcity.edu.eg**
Amr Ahmed Elmasry	**201901202**	**s-amr.elmasry@zewailcity.edu.eg**

Project Introduction:

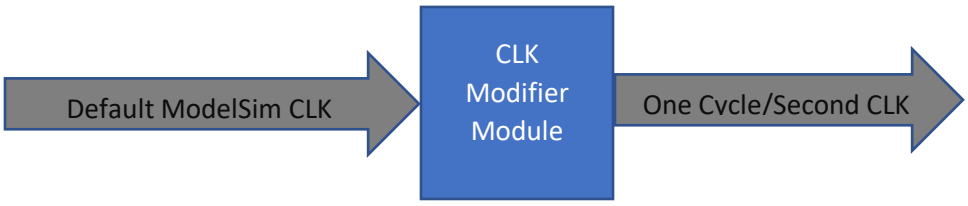
In this project we are required to form a stopwatch and timer machine using ModelSim software to write the code but at first we need to know the building units (Modules) of this machine and how it's work with the given input to produce the required output and this diagram shows this building units and some specifications about them.



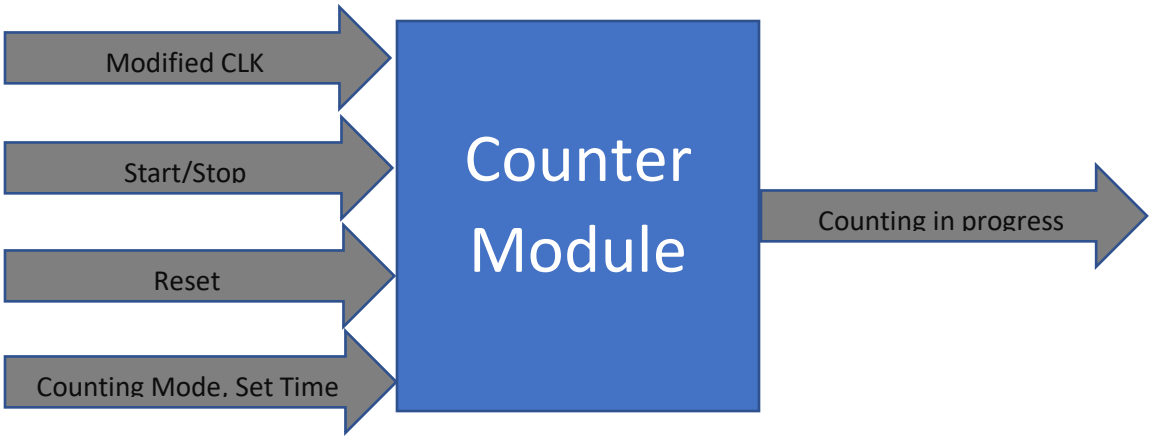
Project Methodology:

The machine should be used as a stopwatch to count up by maximum range one hour (59:59) and also could be used as a timer to count down from the same maximum range.

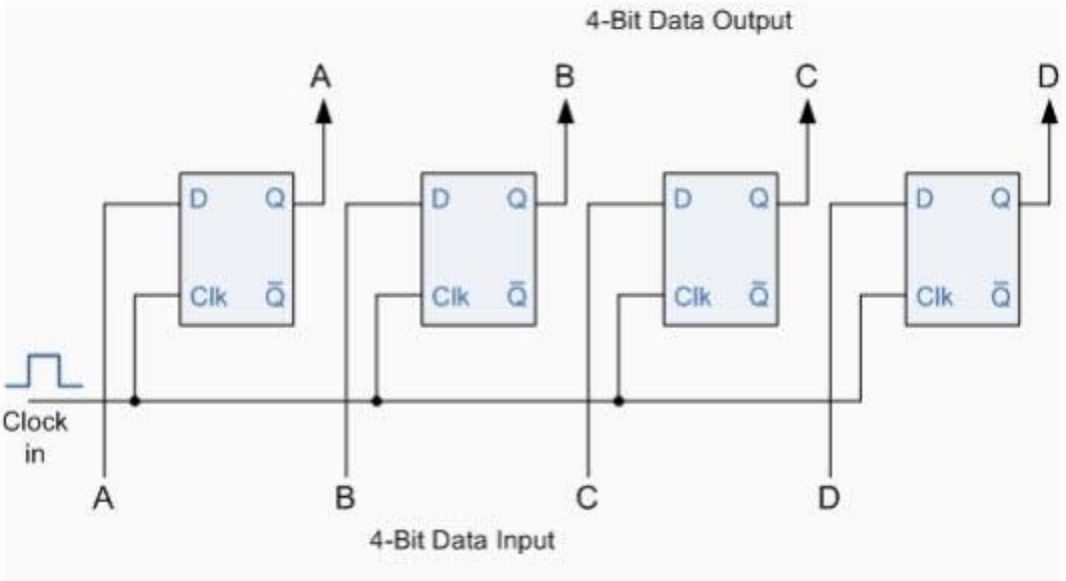
-So at first we need to modify the ModelSim software clock so that its frequency should be one cycle per second to rise every second.



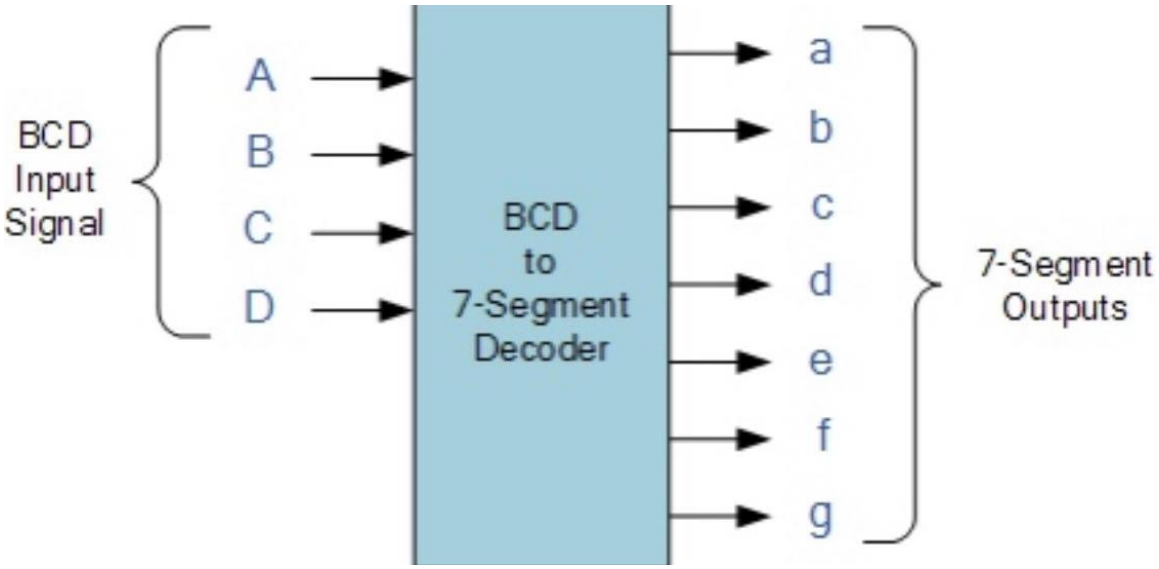
-Secondly, this modified CLK will be used as an input at the most important module in the machine, The Counter Module, The Counter Module should take some other inputs like The counting mode (up/down) with the set time if down, Start/Stop counting and reset and it will be connected with the Display Module so that the counting displayed on the display screen and the counting process for the two seven-segment (as we use four seven-segment for displaying) responsible for seconds will work as follow, the most right seven-segment starts counting from one to nine then at this moment two process will be done, the first process is that the value of this seven-segment will be reset to be zero again and start doing the same job and the second process is that the seven-segment on the left will be increased by one and this processes will continue until we reach 5 in the left seven-segment so the first seven-segment of the minute will be increase by one and a similar process will be held in the two seven-segment responsible for minutes but with some dependence on that responsible for seconds until we reach 59:59 the counter reset all values to zero again. 4-bit will be used to the seven-segment count up to 9 and 3-bit for those that will count up to 5. This process describes the stopwatch mode and the Timer mode process will be totally opposite.



-then the time of The Register Module has come. The register module consists of four registers each register ia a 4-bit register used to save the bits of the counting or the bits of the time that user had set as if we would need them again and a 4-D flip-flop will be implemented with the 4-bit register. The Register Module works as a bits transporter from the Counter module to The Display Module.



-Finally, we need to display so we use four seven-segment decoders as we mentioned before, two are responsible for displaying seconds counting and the other two are responsible for displaying the minutes counting . Each decoder takes as an input 4-bit comes from the counter after passing through the register to save them and the decoder will display the counting to keep the user updated.



Example for the 4-bit input seven-segment decoder:

Tasks Distribution	
Ahmed Mohamed Ibrahem	**Amr Ahmed Elmasry**
CLK Modifier Module in both code and report	Report Introduction
Display Module in both code and report	Counter Module in both code and report
Multiplexer in code	Register Module in both code and report

HDL code:

1)Counter Code

```
1  //Up and Down counter
2  //to be modified in phase2 we integrating
3  module counter #(parameter N)(control, clk, reset, out);
4  input logic control;
5  input logic reset;
6  input logic clk;
7  output logic [N:0] out;
8
9  always_ff @(posedge clk, posedge reset)
10 begin
11 if (reset)
12         out <= 0;
13 else
14 begin
15         if (control)
16                 out <= out - 1;
17         else
18                 out <= out + 1;
19 end
20 end
21
22
23 endmodule
24
```

2)ClK Modifier code

```
1   //used to modify the clock to fit in the system
2   module ClockModifier (clkin, clkout);
3   input logic clkin;
4   output logic clkout;
5   int count;
6
7   always_ff @(posedge clkin)
8   begin
9   //here for example if the clock is 1000Hz
10  count <= count + 1;
11  if (count == 500)
12  begin
13  if (clkout == 0)
14  begin
15          clkout <= 1;
16          count <= 0;
17  end
18  else
19  begin
20          clkout <= 0;
21          count <= 0;
22  end
23  end
24  end
25  endmodule
```

3)Register code

```
1   //to store the value of initial timer
2   module register(input logic set,
3   input logic reset,
4   input logic [3:0] in,
5   output logic [3:0] out);
6
7   always_ff @(posedge set)
8   if (reset) out <= 4'b0;
9   else out <= in;
10
11  endmodule
12
```

4)Multiplexer code

```
1   module Mux2 (A, B, sel, Y);
2   input logic A, B, sel;
3   output logic Y;
4
5   always_comb
6   begin
7
8   if (sel == 1)
9           Y = A;
10  else
11          Y = B;
12  end
13  endmodule
14
15
```

5)Decoders code

```
1  //seven segment decoder for Minutes Tens display
2  //It is implemented by its own as there is an additional case when there is an error
3  module MinutesTensDecoder (in, segments, Error);
4  input logic [2:0] in;
5  input logic Error;
6  output logic [6:0] segments;
7
8  always_comb
9  if (Error)
10         segments = 7'b100_1110;
11 else
12 case(in)
13
14 0: segments = 7'b111_1110;
15 1: segments = 7'b011_0000;
16 2: segments = 7'b110_1101;
17 3: segments = 7'b111_1001;
18 4: segments = 7'b011_0011;
19 5: segments = 7'b101_1011;
20 default: segments = 7'b000_0000;
21
22 endcase
23 endmodule
24
```

```
1  //seven segment decoder for Minutes units display
2  //It is implemented by its own as there is an additional case when there is an error
3  module MinutesUnitDecoder (in, segments, Error);
4  input logic [3:0] in;
5  input logic Error;
6  output logic [6:0] segments;
7
8  always_comb
9  if (Error)
10        segments = 7'b111_1000;
11 else
12 case(in)
13
14 0: segments = 7'b111_1110;
15 1: segments = 7'b011_0000;
16 2: segments = 7'b110_1101;
17 3: segments = 7'b111_1001;
18 4: segments = 7'b011_0011;
19 5: segments = 7'b101_1011;
20 6: segments = 7'b101_1111;
21 7: segments = 7'b111_0000;
22 8: segments = 7'b111_1111;
23 9: segments = 7'b111_0011;
24 default: segments = 7'b000_0000;
25
26 endcase
27 endmodule
```

```
1  //Seven segment Decoder used for seconds display
2  module sevenSegmentDecoder(in, segments);
3  input logic [3:0] in;
4  output logic [6:0] segments;
5
6  always_comb
7
8  case(in)
9
10 0: segments = 7'b111_1110;
11 1: segments = 7'b011_0000;
12 2: segments = 7'b110_1101;
13 3: segments = 7'b111_1001;
14 4: segments = 7'b011_0011;
15 5: segments = 7'b101_1011;
16 6: segments = 7'b101_1111;
17 7: segments = 7'b111_0000;
18 8: segments = 7'b111_1111;
19 9: segments = 7'b111_0011;
20 default: segments = 7'b000_0000;
21
22 endcase
23 endmodule
24
```