Massachusetts Institute of Technology
Department of Electrical Engineering and Computer Science
6.111 - Introductory Digital Systems Laboratory

Problem Set 2 Solutions

Issued: February 27, 2005

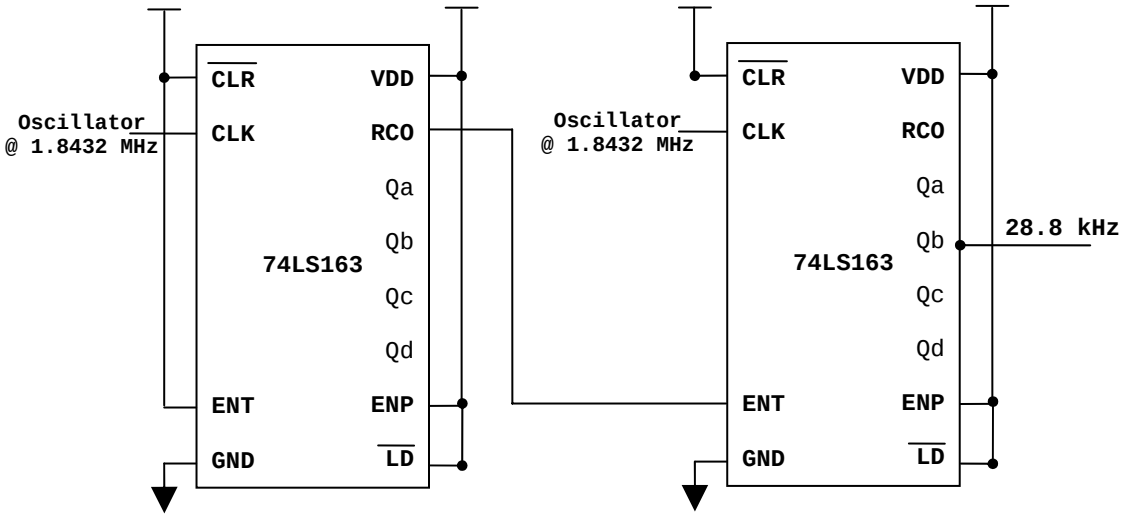
Problem 1: Counters

a) **74LS393** is an **asynchronous 4-bit counter**, implemented with **4 serial T-registers**. The outputs of registers are connected to the inputs of the subsequent registers. Thus, MSBs change only after LSBs change. and clk-to-MSB delay is four times of clk-to-q delay of each register. **74LS163**, on the hand, is a **synchronous 4-bit counter**, implemented by **4 parallel J-K registers**. All output bits change at the rising edge of clock and thus clk-to-MSB delay is similar to clk-to-q delay of individual register.

b) N^{th} output of ripple counters has twice the time period of $(N-1)^{th}$ output, and thus N^{th} output of ripple counters are 2^N slower than the input clock. Thus, to create a 28.8kHz clock,

$$\frac{1.8432MHz}{28.8kHz} = 64 = 2^6$$

we must divide the input clock by two for six times, requiring two 74LS163. Cascade the two 4-bit counters as done in Lab 1 and probe at Qb of the second counter as shown below:



<Figure 1: 74LS163 connections to provide a 28.8kHz clock>

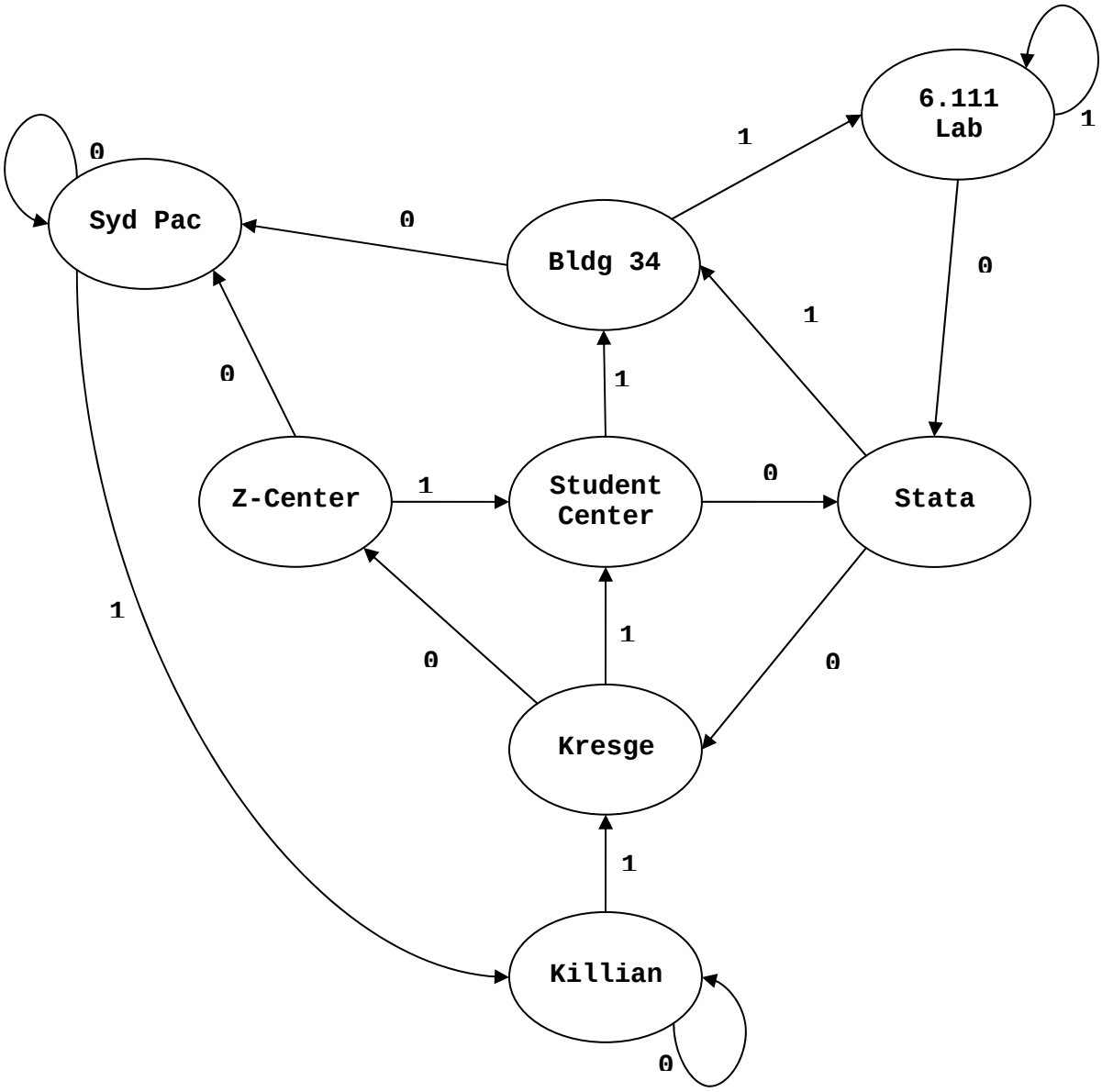
c) 1843200 is not an exact multiple of twos and thus we cannot use the method used in part b). Instead, using 74LS163s, we count up to 1843200 and pulse the output then to generate 1Hz clock. Since $2^{20} < 1843200 < 2^{21}$ and there are four bits per counter, we need **6 74LS163s** to produce 1 Hz clock.

d) Verilog Code for Counter Module:

```
`timescale 1ns/10ps

module counter(clk, reset, enable);

   input clk, reset;
   output enable;

   reg enable;
   reg [20:0] cnt;

   //1.8432MHz = 2^21

   always @ (posedge clk) begin
          if(reset == 1) begin
            cnt <= 21'd0;
            enable <= 1'b0;
          end

          else if(cnt == 21'd1843199) begin
          // else if(cnt == 21'd1) begin //for faster simulation
            enable <= 1'b1;
            cnt <= 21'd0;
          end

          else begin
            cnt <= cnt + 1;
            enable <= 1'b0;
          end //else
     end //always

endmodule
```

Problem 2: Finite State Machines (FSM)

a)



<Figure 2: State transition diagram for Stata>

b) 6.111 Lab

c) 6.111 Lab

d) Verilog Code for stata_FSM module:

```
`timescale 1ns/10ps

module stata_fsm(clk, fsmreset, fsminput, state);

// System Clk for state transition
input clk;

// Global Reset signal
input fsmreset;
input fsminput;

output[2:0] state;

// internal sate
reg [2:0] state;
reg [2:0] nextstate;

//State Declarations:
parameter KILLIAN = 0;
parameter KRESGE = 1;
parameter ZCENTER= 2;
parameter SYDPAC = 3;
parameter STUDENTCENTER = 4;
parameter BUILDING34 = 5;
parameter LAB = 6;
parameter STATACENTER = 7;

always @ (posedge clk) begin
    if (fsmreset) state <= KILLIAN;
    else state <= nextstate;
end

always @ (state or fsminput) begin
    nextstate = 3'b000;
    case (state)

    KILLIAN: if(fsminput) nextstate = KRESGE;
    else nextstate = KILLIAN;

    KRESGE: if(fsminput) nextstate = STUDENTCENTER;
    else nextstate = ZCENTER;

    ZCENTER: if(fsminput) nextstate = STUDENTCENTER;
    else nextstate = SYDPAC;

    SYDPAC: if(fsminput) nextstate = KILLIAN;
```

```
        else nextstate = SYDPAC;

        STUDENTCENTER: if(fsminput) nextstate = BUILDING34;
        else nextstate = STATACENTER;

        BUILDING34: if(fsminput) nextstate = LAB;
         else nextstate = SYDPAC;

        STATACENTER: if(fsminput) nextstate = BUILDING34;
         else nextstate = KRESGE;

        LAB: if(fsminput) nextstate = LAB;
        else nextstate = STATACENTER;

    endcase //
  end //
endmodule
```

Problem 3: Verilog Testbench

Verilog Code for Top Module:

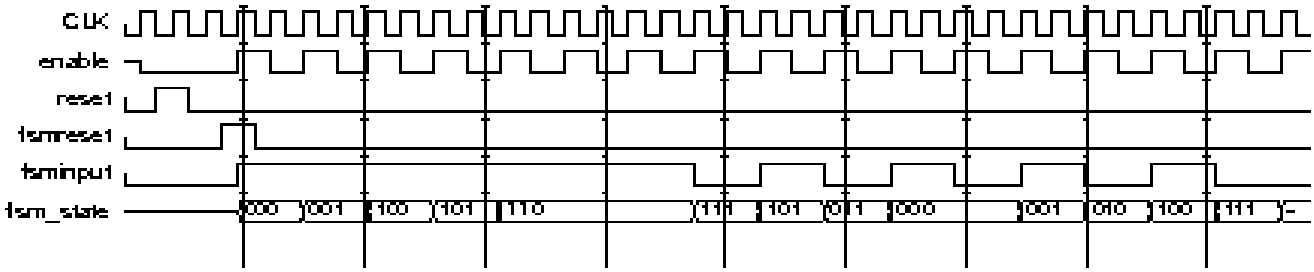
```
`timescale 1ns/10ps

module top(clk, reset, fsminput, fsmreset, state);

 input clk, reset, fsminput, fsmreset;
 output [2:0] state;
 wire enable;

 counter c1(clk, reset, enable);
 stata_fsm s1(enable, fsmreset, fsminput, state);

endmodule
```

Verilog Code for Test Bench Module:

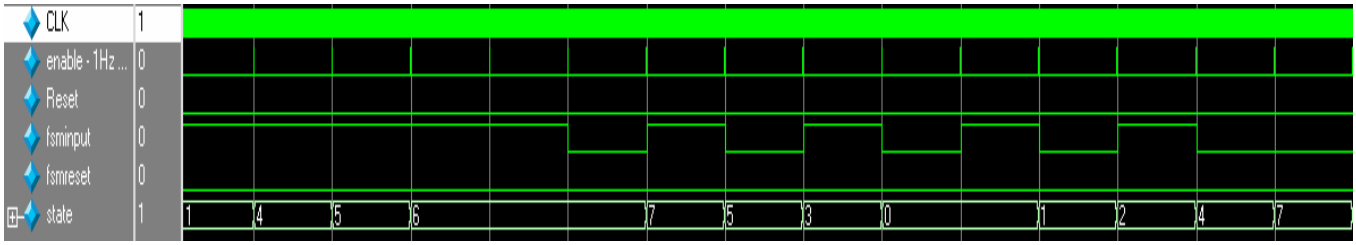
```
`timescale 1us/1ps

module testbench;
//Now use 1Hz enable clock:

 reg clk, reset, fsminput, fsmreset;
 wire [2:0] fsm_state;
 wire [2:0] fsm;
 integer i;
```

```
 // Instantiate the top module:
 top
t1(.clk(clk), .reset(reset), .fsminput(fsminput), .fsmreset(fsmreset), .state(fsm_state)
);

 // Specify clk to have 1.843 MHz:
 always #0.271 clk = ~clk;


initial begin

  //Initial signal:
  clk =0;
  reset =0;
  fsminput = 0;
  fsmreset =0;

  // Reset counter first:
  #0.541
  reset = 1;

  #0.541
  reset = 0;

  // Reset FSM:
  #0.271
  fsmreset = 1;

  #0.271
  fsminput = 1;

  #0.271
  fsmreset = 0;

  // We know fsminput should be high for seven times
  // Therefore, fsminput = 1 for clk_period * 7 = 1s*7 = 7s = 7000ms
  // Thus, 7ms - 271ns

  #7000000
  // Is Stata staying in 6.111Lab for three clock cycles?

  // Let's toggle between 0 and 1 for 8 clock cycles
  // until Stata comes back to Killian completing a tour
  // without visiting 6.111 Lab!
  fsminput = 0;

  for (i= 1; i<=4; i=i+1) begin
```

```
    #1000000
    fsminput = 1;

    #1000000
    fsminput = 0;
  end
end
endmodule
```



<Figure 3a: (Fast Simulation) Testbench verifies that Stata goes from Killian -> Kresge -> Student Center -> Building 34 -> 6.111 Lab and gets stuck for three clock cycle. Then, Stata gets out completing a round tour without visiting 6.111 >



<Figure 3b: (1Hz Simulation) Testbench verifies that Stata goes from Killian -> Kresge -> Student Center -> Building 34 -> 6.111 Lab and gets stuck for three clock cycle. Then, Stata gets out completing a round tour without visiting 6.111 >

Problem 4: Memory Tester

a)

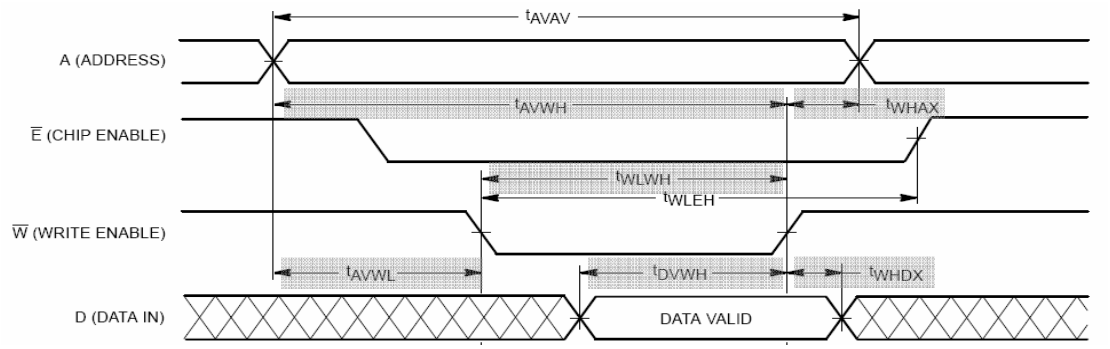
$\overline{E1}$	E2	$\overline{G}$	$\overline{W}$	Mode	Output	Cycle
H	X	X	X	N/S	High-Z	-
X	L	X	X	N/S	High-Z	-
L	H	H	H	Output Disabled	High-Z	-
L	H	L	H	Read	Dout	Read Cycle
L	H	L*	L	Write	High-Z	Write Cycle
L	H	X	L	Write	High-Z	Write Cycle

* Mode is **write** and thus assume **~G** goes low coincident with or after **~W** goes low.

b) 1 MHz clock has the time period of 1000ns and thus all the timing constraints specified in the data sheet (<20ns) should meet well within one clock period. However, if we used a clock faster than 50MHz (i.e. if the time period is smaller than 20ns), we must use conservative methods, such as multi-cycle writes and reads shown in the Lecture 6.

1) Writing Data

As discussed in lecture 6, data is latched when `Wbar` or `E1bar` goes high. Thus, we can choose between `Wbar` controlled write (write cycle 1) or `E1bar` controlled write (write cycle 2). For this solution, we do `Wbar` controlled write such that we don't have to disable the chip while writing. The timing constraints from datasheet are illustrated below:

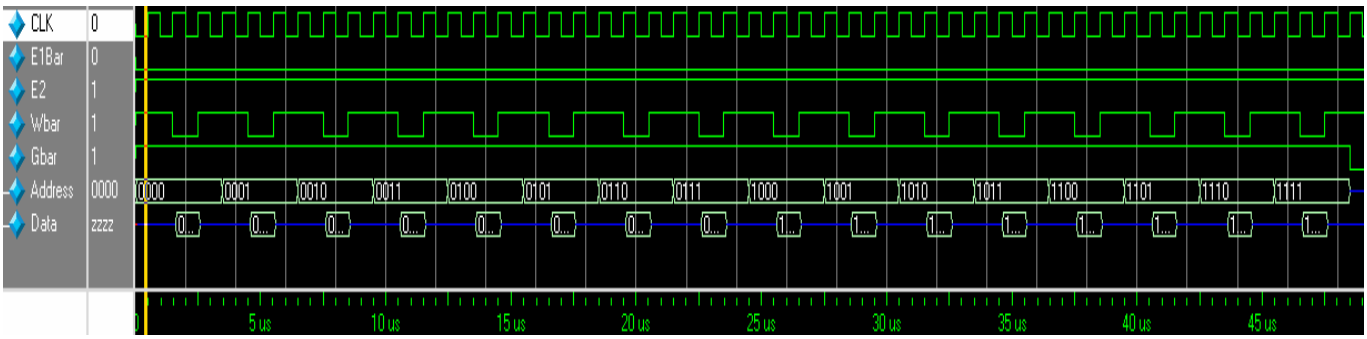


<Figure 4: Timing constraints for `Wbar` controlled writes>

Having `E1bar` always low, the timing constraints to be met are the ones that start or end along the rising and falling edge of `Wbar` signal. i.e. t_{AVWH} (20ns), t_{WHAX}(0ns), t_{WLWH}(20ns), t_{AVWL} (0ns), t_{DVWH} (12ns), and t_{WHDX} (0ns). Thus, the write pulse width must be greater than t_{WLWH} (20ns), your data must be valid for t_{DVWH} (12ns), and your address must be t_{AVWH} (20ns) before the rising edge of `Wbar`.

Address setup time (t_{AVWL}) is 0ns but this time refers to the time between the valid `address` to the falling edge of `Wbar`. Thus, you `address` MUST be valid before the

falling edge of `Wbar`. Data hold time ($t_{\_WDHX}$) is also 0ns. But, to perform safe writes, we use the multi-cycle write as shown in the lecture; `address` are held valid for three clock cycle, where `Wbar` pulse takes place on the second clock cycle to ensure safe writes. `Data-In` must be valid at the rising edge of `Wbar` as well as shown below:



<Figure 5: Control signals to write data safely with clock period of 1000ns>

What you need to remember from this problem:

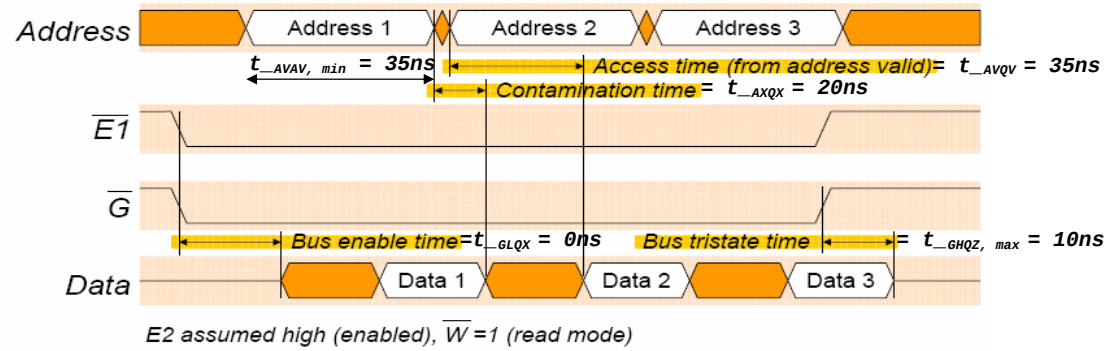
1. `Address` must be valid before the falling edge of `Wbar`.
2. `Address` must be valid during the entire write pulse.
3. Only at the rising edge of `Wbar`, `data-in` gets written to `address`.

For Grading: For `Wbar` controlled writes, your waveforms must show:

1. `E1bar` = 0, `E2` = 1, `Gbar` = 1.
2. `Wbar` pulses down for every write.
3. `Address` changes from 0000 -> 1111 incrementally.
4. Meet the first two constraints specified in 'What you need to remember from this problem'.

2) Reading Data:

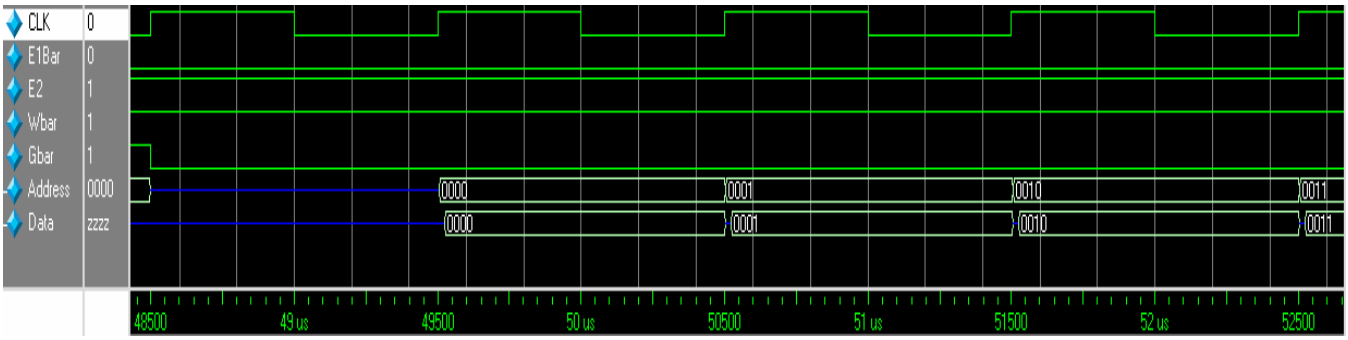
Similar to the write cycle, there are two ways to read: `Gbar` and `Address` controlled. For this solution, we do `address` controlled write such that we can perform multiple reads without disabling chip. The timing constraints for `address` controlled write are illustrated below:



<Figure 6: Timing constraints for `address` controlled read from Lecture 6 with added timing constraints from the datasheet>

The important constraint is t_{AVQV}, which tells you when you can start reading your data for the `address` you are accessing. For example, if you read `data-out` at 22ns after your `address` changes, your `data-out` is most likely invalid.

To meet these constraints with our clock of 1000ns, we can simply increment `address` every clock period (1000ns >> t_{AVAV}) and read `data` at rising edge of clock as shown below:



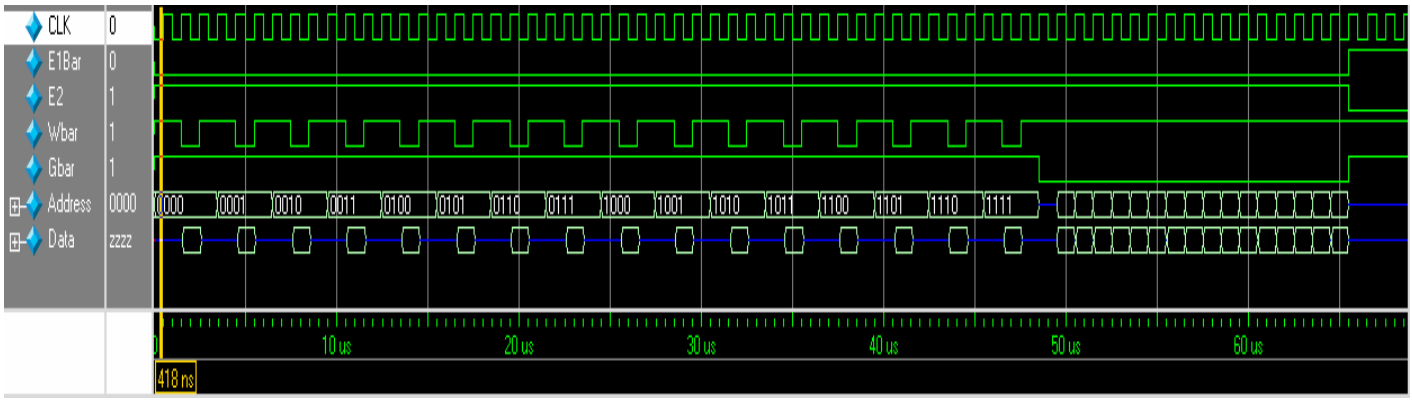
<Figure 7: Control signals to read data properly with clock period of 1000ns>

For Grading: For `address` controlled reads, your waveforms must show:

1. `E1bar` = 0, `E2` = 1, `Gbar` = 0, and `Wbar` = 1
2. `Address` changes from 0000 -> 1111 incrementally.
3. `Address` must be hold for at least 35ns.

3) Overall Control Signals:

The overall control signals to write 16 times and read 16 times are shown below with its Verilog codes.



<Figure 8: Control signals to write and read from 16 addresses>

```
module timing(clk, E1bar, E2,
Wbar, Gbar, A, Data);

input clk;
output E1bar, E2, Wbar, Gbar;
output [3:0] A, Data;
integer i;
reg E1bar, E2, Wbar, Gbar;

reg [3:0]  A, Data;

initial  begin
  clk = 0;
  A = 4'b0000;
  i = 0;
  Wbar = 1;
  E1bar = 0;
  E2 = 1;
  Gbar = 1;
 end


always @ (posedge clk)  begin
   if (i < 48)
    begin
     if (i % 3 == 0)
         A = i/3;
     else if (i % 3 == 1)
        Wbar = 0;
     else
        Wbar = 1;
    i = i + 1;
    end //if
```

```
    else if (i == 48)  begin
        Gbar <= 0;
        A <= 4'bZ;
        Data <= 4'bZ;
        i = i + 1;
    end

    else if (i < 65) begin
          Gbar = 0;
          #4 Data <= 4'bZ;
          A <= i - 32 - 1;
          i <= i + 1;
          #20 Data <= A;
    end // else if

    else begin
          A <= 4'bZ;
          Data <= 4'bZ;
          Wbar = 1;
          Gbar = 1;
          E1bar = 1;
          E2 = 0;
         end // else
    end // if

    always @ (Wbar) begin
       if (i <49)
        begin
         if (Wbar) #100 Data = 4'bZ;
         else #100 Data = A;
        end //if (i < 48)
    end

endmodule
```