Oregon TECH

Oregon Institute of Technology

CST456 – Embedded Testing

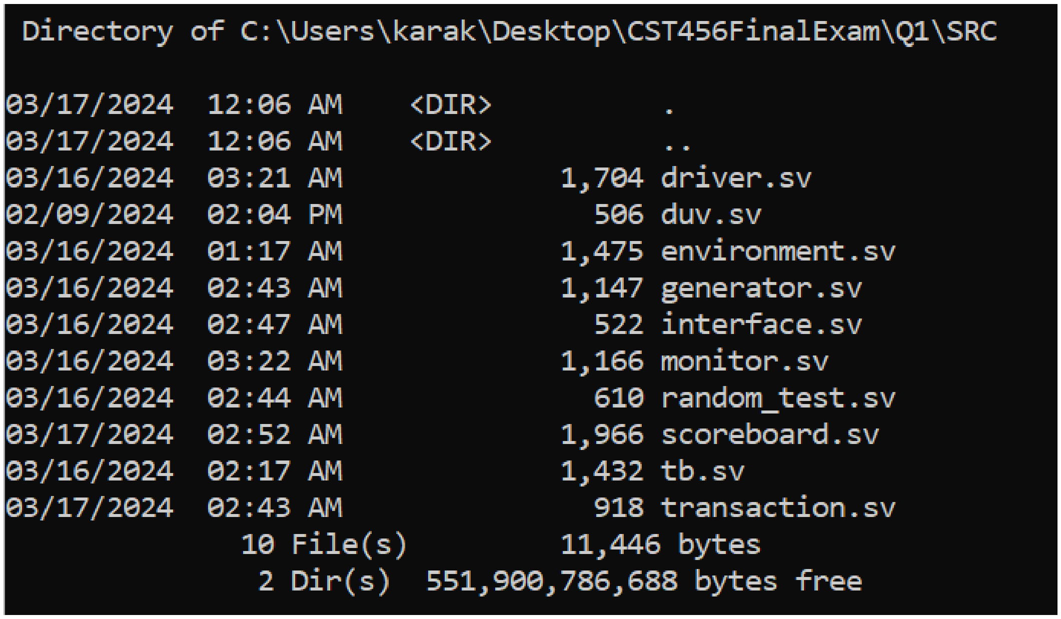
Final Exam

Due date: 03/19/2025 5PM – Any files uploaded after due date will not be accepted.

I will schedule a zoom office hour on Monday (03/17/2025) at 11:00 AM to answer your questions. No question send via email will be answered.

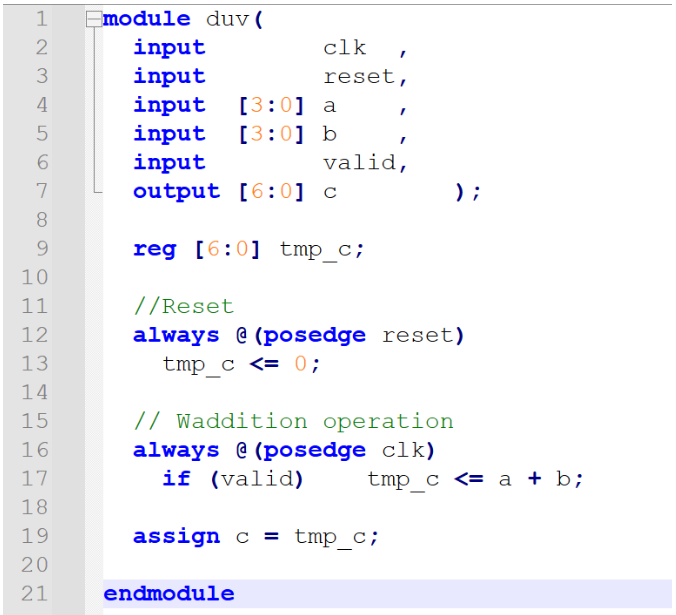
Q1 (40 Points):

Under source directory (SRC) of CST456FinalExam/Q1/src you will see the following files:



```
 Directory of C:\Users\karak\Desktop\CST456FinalExam\Q1\SRC

03/17/2024  12:06 AM    <DIR>          .
03/17/2024  12:06 AM    <DIR>          ..
03/16/2024  03:21 AM             1,704 driver.sv
02/09/2024  02:04 PM               506 duv.sv
03/16/2024  01:17 AM             1,475 environment.sv
03/16/2024  02:43 AM             1,147 generator.sv
03/16/2024  02:47 AM               522 interface.sv
03/16/2024  03:22 AM             1,166 monitor.sv
03/16/2024  02:44 AM               610 random_test.sv
03/17/2024  02:52 AM             1,966 scoreboard.sv
03/16/2024  02:17 AM             1,432 tb.sv
03/17/2024  02:43 AM               918 transaction.sv
              10 File(s)         11,446 bytes
               2 Dir(s)  551,900,786,688 bytes free
```

These are the files for a reference design. Reference design tests the following DUV file:



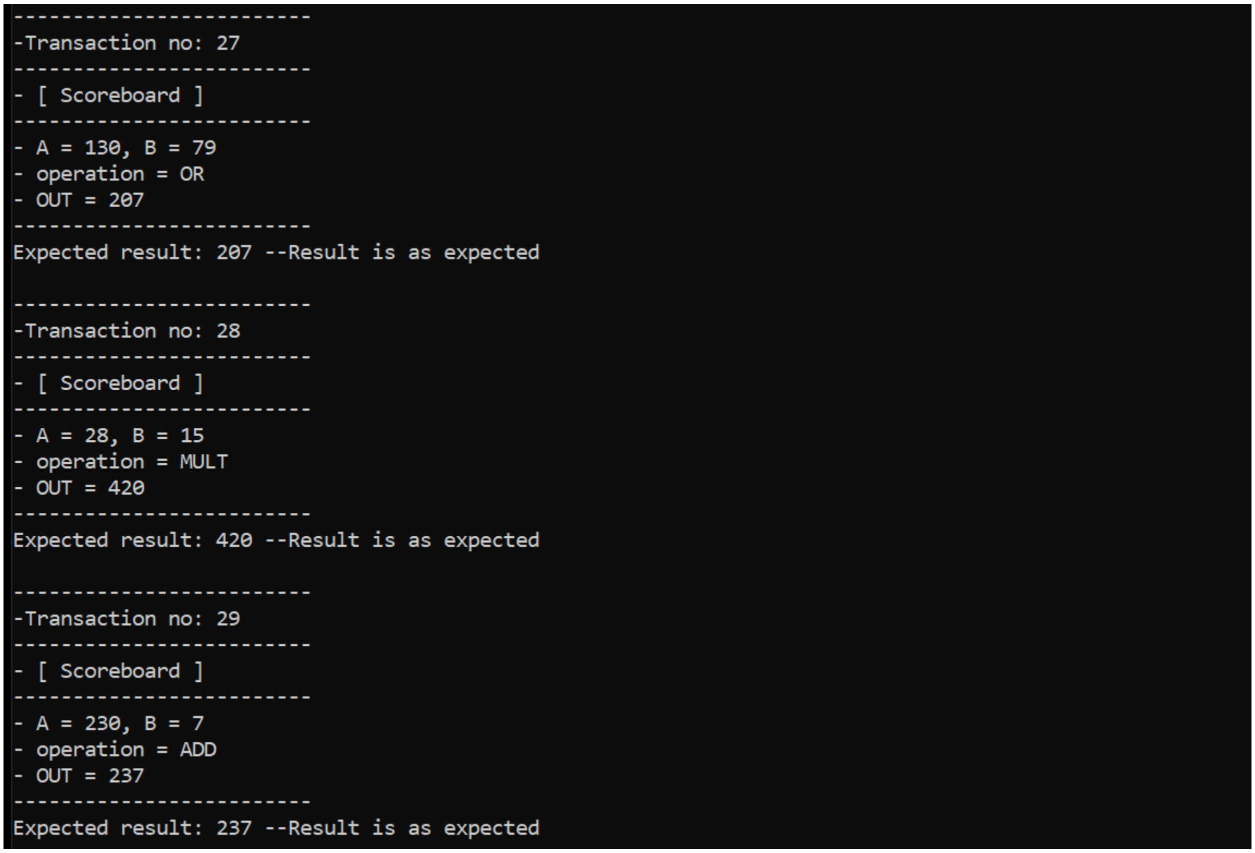
```
module duv(
  input         clk   ,
  input         reset,
  input   [3:0] a     ,
  input   [3:0] b     ,
  input         valid,
  output  [6:0] c          );

  reg [6:0] tmp_c;

  //Reset
  always @(posedge reset)
    tmp_c <= 0;

  // Waddition operation
  always @(posedge clk)
    if (valid)    tmp_c <= a + b;

  assign c = tmp_c;

endmodule
```

Make the necessary modifications on required files so that reference design can test the following DUV file:

```
-----------------------------------------------------

module duv (
input logic clk,
input logic op_start,
input  logic [7:0] A,
input  logic [7:0] B,
input logic  [1:0] opcode,
output logic [15:0] OUT
);
logic [15:0] result_temp;
always_ff @(posedge clk)
begin
if( op_start == 1'b1)
begin
```

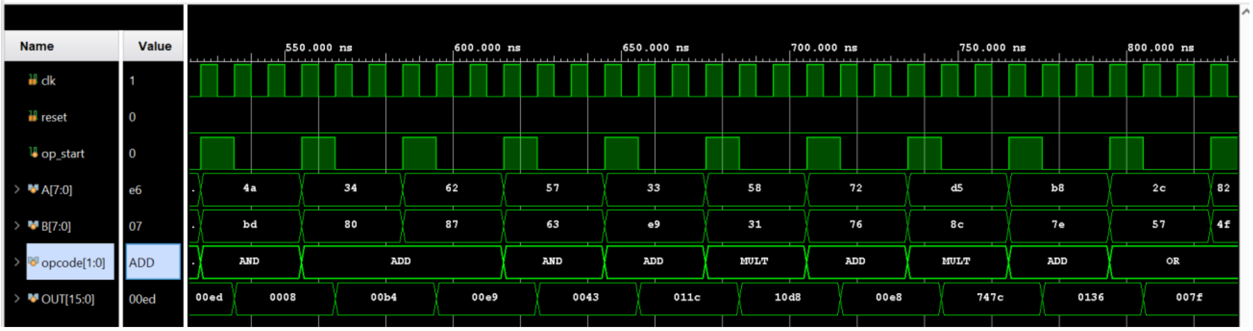
```
case (opcode)

      0:             result_temp = A + B;
      1:             result_temp = A * B;
      2:             result_temp = A | B;
      3:             result_temp = A & B;
    endcase

end

     OUT <= result_temp;

end
```

After successfully making the necessary changes on reference design it should print out info on screen similar to following screenshot:



```
-------------------------
-Transaction no: 27
-------------------------
- [ Scoreboard ]
-------------------------
- A = 130, B = 79
- operation = OR
- OUT = 207
-------------------------
Expected result: 207 --Result is as expected

-------------------------
-Transaction no: 28
-------------------------
- [ Scoreboard ]
-------------------------
- A = 28, B = 15
- operation = MULT
- OUT = 420
-------------------------
Expected result: 420 --Result is as expected

-------------------------
-Transaction no: 29
-------------------------
- [ Scoreboard ]
-------------------------
- A = 230, B = 7
- operation = ADD
- OUT = 237
-------------------------
Expected result: 237 --Result is as expected
```

The waveforms should look like as follows:


Name
Value
550.000 ns
600.000 ns
650.000 ns
700.000 ns
750.000 ns
800.000 ns
clk 1
reset 0
op_start 0
A[7:0] e6
4a 34 62 57 33 58 72 d5 b8 2c 82
B[7:0] 07
bd 80 87 63 e9 31 76 8c 7e 57 4f
opcode[1:0] ADD
AND ADD AND ADD MULT ADD MULT ADD OR
OUT[15:0] 00ed
00ed 0008 00b4 00e9 0043 011c 10d8 00e8 747c 0136 007f

Q2. PartI (30 Points):

The portable vital sign monitoring device measures a patient's heart rate and temperature and responds accordingly. When the patient's vital signs are normal the device is quiet. If the patient's vital signs deviate from normal the device will alert the patient. If the patient's vital signs become serious the device will alert a doctor via wireless connectivity.

Heart rate and temperature are monitored and their condition assessed according to their severity. The tables below show the assessment conditions for heart rate and temperature.

Heart Rate (bpm)	**Assessment**
< 40	Serious
40 to 59	Fair
60 to 90	Good
91 to 120	Fair
> 120	Serious

Temperature (°C)	**Assessment**
< 34	Serious
34 to 35	Fair
36 to 38	Good
39 to 40	Fair
> 40	Serious

Both the heart rate and temperature assessments are used to determine the alert state to recommend. The table below shows the criteria for the alert recommendation. The alert recommendation is to contact the doctor whenever the heart rate or temperature becomes serious. The alert recommendation is to alert the user whenever both the heart rate and temperature are fair. A no alert recommendation is given for the rest of the cases.

Heart Rate	**Temperature**	**Alert Recommendation**
Good	Good	No alert
Good	Fair	No alert
Good	Serious	Alert doctor
Fair	Good	No Alert
Fair	Fair	Alert user
Fair	Serious	Alert doctor
Serious	Good	Alert doctor
Serious	Fair	Alert doctor
Serious	Serious	Alert doctor

These are only recommendations though. A certain number of consecutive recommendations must occur before the output state of the device transitions from one state to the other, as shown in the table below.

Alert Recommendation	**Number of Consecutive Alert Recommendations Needed before Output State Transitions**
No Alert	3
Alert user	2
Alert doctor	1 (transition immediately)

The firmware consists of four functions: returnHeartRateConcern, returnTemperatureConcern, updateAlertStatus, and initAlertStatus. returnHeartRateConcern takes the heart rate as input and returns the corresponding assessment concern. returnTemperatureConcern takes the temperature as input and returns the corresponding assessment concern. initAlertStatus initializes the output of the device by setting it to no alert. updateAlertStatus takes both the temperature and heart rate as input and is the main body of the program. Calling the function represents a new measurement cycle which may or may not change the alert state of the device.

Instructions

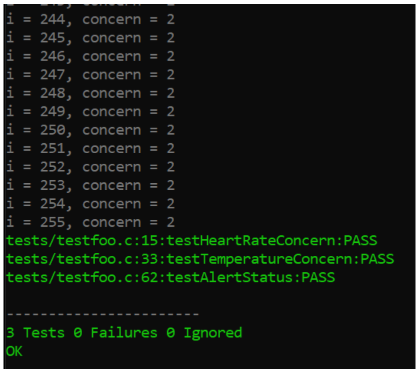
"CST456FinalExam/Q2/PartI/unity/test/tests/Testfoo.c" contains uncompleted tests for:

returnHeartRateConcern

returnTemperatureConcern

updateAlertStatus.

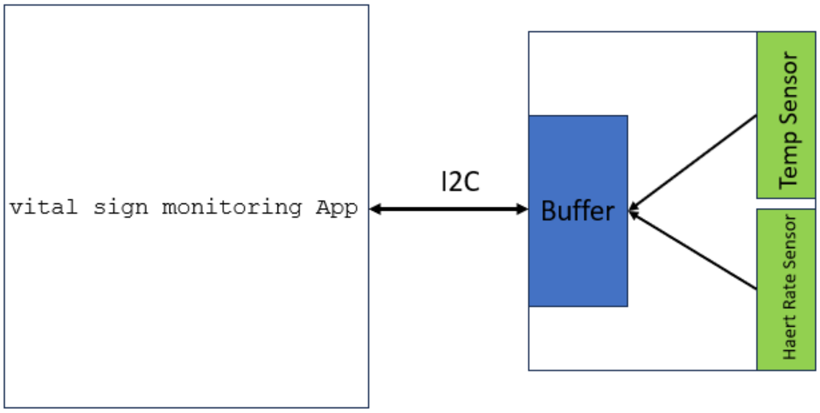
Complete Testfoo.c according to the given instructions inside the file. Succesfull run should print out following:



```
i = 244, concern = 2
i = 245, concern = 2
i = 246, concern = 2
i = 247, concern = 2
i = 248, concern = 2
i = 249, concern = 2
i = 250, concern = 2
i = 251, concern = 2
i = 252, concern = 2
i = 253, concern = 2
i = 254, concern = 2
i = 255, concern = 2
tests/testfoo.c:15:testHeartRateConcern:PASS
tests/testfoo.c:33:testTemperatureConcern:PASS
tests/testfoo.c:62:testAlertStatus:PASS

-----------------------
3 Tests 0 Failures 0 Ignored
OK
```

Q2 PartII (30 Points):

In PartII we will mock the I2C to read from a Buffer.



Buffer has 8 address and it can save 4 pairs of data from sensors:

Address	Data
0x00	Haert Rate - 1
0x01	Temperature - 1
0x02	Haert Rate – 2
0x03	Temperature – 2
0x04	Haert Rate - 3
0x05	Temperature - 3
0x06	Haert Rate – 4
0x07	Temperature - 4

After successive 4 readings from the sensors the buffer has the following content:

Address	Data
0x00	75
0x01	37
0x02	39
0x03	33
0x04	91
0x05	34
0x06	150
0x07	41

And we want to test **only** the "*updateAlertStatus*" function using the data available in the buffer. Data reading from the buffer will be performed by mocking i2c. For i2c reading following function must be used:

```
uint16_t i2c_readBuffer(uint8_t BufferAddress)
```

It returns the data present in the given buffer address.

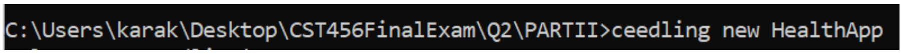
Two successive reading must be done to get a data pair (heart rate and temperature):

```
uint16_t i2c_readBuffer(uint8_t heartrateBufferAddress)
```

```
uint16_t i2c_readBuffer(uint8_t temperatureBufferAddress)
```

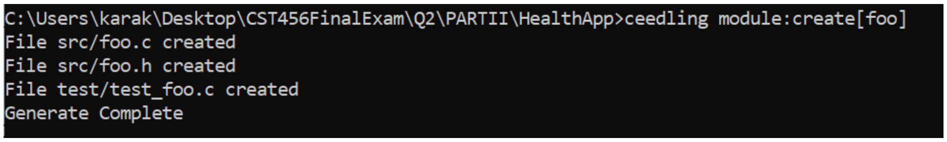
For this part do all your work under "CST456FinalExam/Q2/PartII".

First create a project named "HealthApp" using:



```
C:\Users\karak\Desktop\CST456FinalExam\Q2\PARTII>ceedling new HealthApp
```

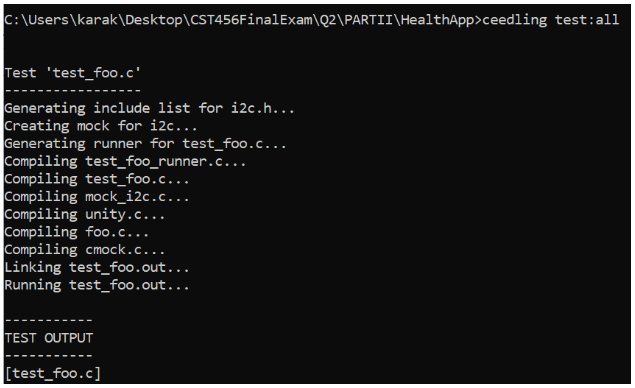
Go under "HealthApp" directory and create a new project named "foo" using:



```
C:\Users\karak\Desktop\CST456FinalExam\Q2\PARTII\HealthApp>ceedling module:create[foo]
File src/foo.c created
File src/foo.h created
File test/test_foo.c created
Generate Complete
```

Replace "foo.c" and "foo.h" with the given ones. Add/modify files as needed for successful testing of the "*updateAlertStatus*" function for the data present in the buffer.

After modifying "foo.c", "foo.h" and "test_foo.c" files and adding new file needed for i2c, start testing by using following command:



```
C:\Users\karak\Desktop\CST456FinalExam\Q2\PARTII\HealthApp>ceedling test:all


Test 'test_foo.c'
-----------------
Generating include list for i2c.h...
Creating mock for i2c...
Generating runner for test_foo.c...
Compiling test_foo_runner.c...
Compiling test_foo.c...
Compiling mock_i2c.c...
Compiling unity.c...
Compiling foo.c...
Compiling cmock.c...
Linking test_foo.out...
Running test_foo.out...

-----------
TEST OUTPUT
-----------
[test_foo.c]
```

Hint: Lecture8 and Lecture9 slides will be a great help

Submission

Zip the entire contents of the “CST456FinalExam” directory and submit it to Canvas.