Peisen Hu 455 L2 Report

Part 1:

1. The whole state function is independent from the FSM framework in my implementation, with a general structure similar to this:

```
unsigned int stateX(void) //State for 2V<ADC=<3V
{
        //Enter Code
        …
        //Loop Containing Execute Code and Exit Code
        while(1){
                // Execute Code
                …
                //Exit Code
                if(…//Exit Condition){
                        …//Actual Exit Code for Each Exit Condition
                }else if
                        …
        }
}
```

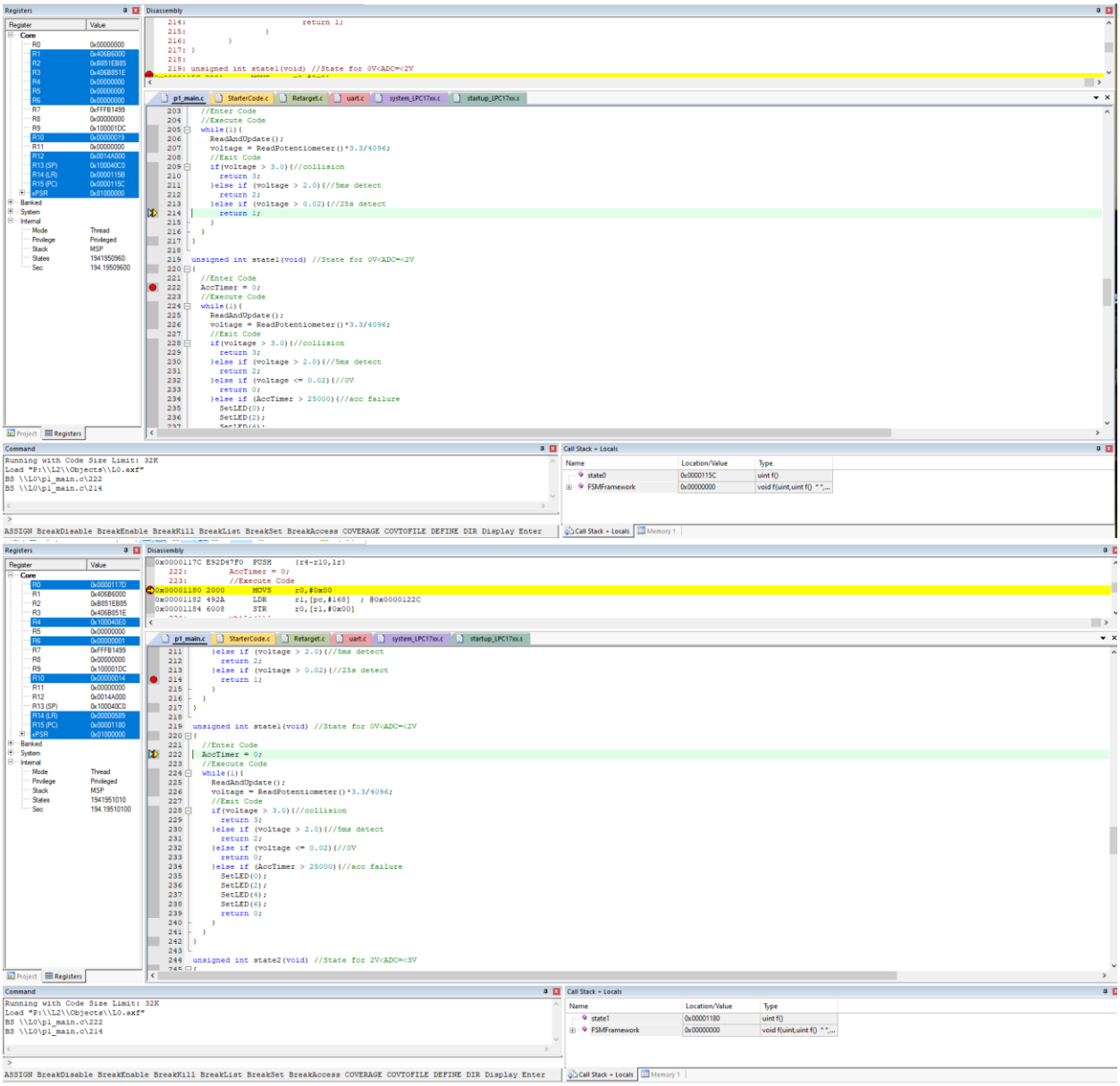
So the worst situation with slowest response is that the exit condition changes (the user requests a state change) right after its respective exit condition check statement has passed. So we can see that "the WCET from the point the user requests a state change to the beginning of the first instruction of the exit code of the current state" will take place under this situation, which is:

Time(i.e. WCET) to run all if checks in the Exit Code + Time to run the Execute Code

2. As described above and in manual (report) for lab 1, the time from the end of the last instruction of the exit code of a state to the beginning of the first instruction of the entry code of the next requested state is actually (mostly) constant. This because the only path here is fixed:

End of Exit Code & Return from current state func -> Go to the next iteration in the infinite loop in FSMFramework () and call the func for the next state -> Go to the beginning of Enter Code

So we can just set two breakpoints, one at the end of the last instruction of the exit code of a random state, the other at the beginning of the first instruction of the entry code of the next requested state, and then run the debugger to check the WCET:



*We may use similar method to measure things like the runtime of a function below. But the snips won't be shown to keep this report simple and readable.
The difference in the states is:
1941951010-1941950960=50 (states/cycles)
The clock freq. is 100MHz. So the time elapsed, i.e. the WCET, is:
50*10^-8=**$5*10^{-7}$(s)**
***Notice that in the physical world, this value can be affected by stuff like skew and propagation delay.**

3. This is similar to 2. There is only one virtually fixed path, and it only consists of running the Initializing & Start running the while loop. Similarly, we set two breakpoints, one at the end of

the last instruction of the entry code of a state, the other at the beginning of the first instruction of the active code of that state. Then we get:
The difference in the states is:
281391-281388=3 (states/cycles)
The clock freq. is 100MHz. So the time elapsed, i.e. the WCET, is:
3*10^-8=**$3*10^{-8}$(s)**

//

Part 3:

The code of state3, which is in charge of launching the airbag and printing the log upon crash, is copied below for your convenience of referring:

```
unsigned int state3(void) //State for 3V<ADC Imminent collision, brace for
impact!
{
    //Enter Code
    SetLED(0);
    SetLED(1);
    SetLED(2);
    SetLED(3);
    SetLED(4);
    SetLED(5);
    SetLED(6);
    SetLED(7);
    PrintLog();
    //printf("Crashed!\r\n");
    while(1){
        //use an infinite loop to represent that the car is crashed and ECG is no
longer responding
    }
    //Execute Code
    //Exit Code
}
```

1. Now we want to know the WCET from the point the accelerometer goes above 3V to the LEDs turning on indicating the airbag has activated. This will take place when the condition change (i.e. the accelerometer goes above 3V) happens immediately after its respective if check statement has passed, and the Execute Code, as well as the if/else if block, are in there WCET conditions. The WCET here is given by:

WCET to run all if checks in the Exit Code + WCET to run the Execute Code + Time to run exit code (this is fixed for this case and simply *return 3;*) + Time to go to State3() and run SetLED(0) – SetLED(7)

The latter 2 items are fixed. And for the first item, it will take place in State1 and State2 where there're 4 if checks (there's only 3 in State0). The second item is also quite simple since the execute code in State0,1,2 are identical:

```
      ReadAndUpdate();
      voltage = ReadPotentiometer()*3.3/4096;
```

Also the code for ReadAndUpdate() is:

```
void ReadAndUpdate() {
        joy = ReadJoystick();
        if (joy == 16 && joy_flag == 0 && JoyFreezeTimer > 200){
            joy_flag = 1;
            JoyFreezeTimer = 0;
            //pedal engaged, log and start to care for ABS
            //printf("pedal engaged, log and start to care for ABS\r\n");
            LogUpdate(0, 0);
            if (btn_flag == 0){
                brake_flag = 1;
                LogUpdate(2, 0);
            }
        } else if (joy == 0 && joy_flag == 1){
            joy_flag = 0;
            //pedal disengaged, log
            //printf("pedal disengaged, log\r\n");
            LogUpdate(0, 1);
            if (brake_flag == 1){
                brake_flag = 0;
                LogUpdate(2, 1);
            }
        }
        //if (joy_flag == 1){
        btn = ReadPushbutton();
        if (btn != 0 && btn_flag == 0 && BtnFreezeTimer > 200){
            btn_flag = 1;
            BtnFreezeTimer = 0;
            //ABS engaged, log
            //printf("/ABS engaged, log\r\n");
            LogUpdate(1, 0);
            if (brake_flag == 1){
                brake_flag = 0;
                LogUpdate(2, 1);
            }
```

```
        } else if (btn == 0 && btn_flag == 1){
            btn_flag = 0;
            LogUpdate(1, 1);
            if (joy_flag == 1){
                brake_flag = 1;
                LogUpdate(2, 0);
            }
        }
        //}
}
```

So it can be seen that its WCET condition is that for each if block, at least one condition is met so that the statements inside will be run. (We don't consider the WCET for LogUpdate() because that's virtually fixed.) This can take place in the situation below:

Originally both ABS Button and Pedal are in the released state. Also the Freeze Timers have exceeded 200. Now both of them are pushed!
(In this case, all of the following are True:

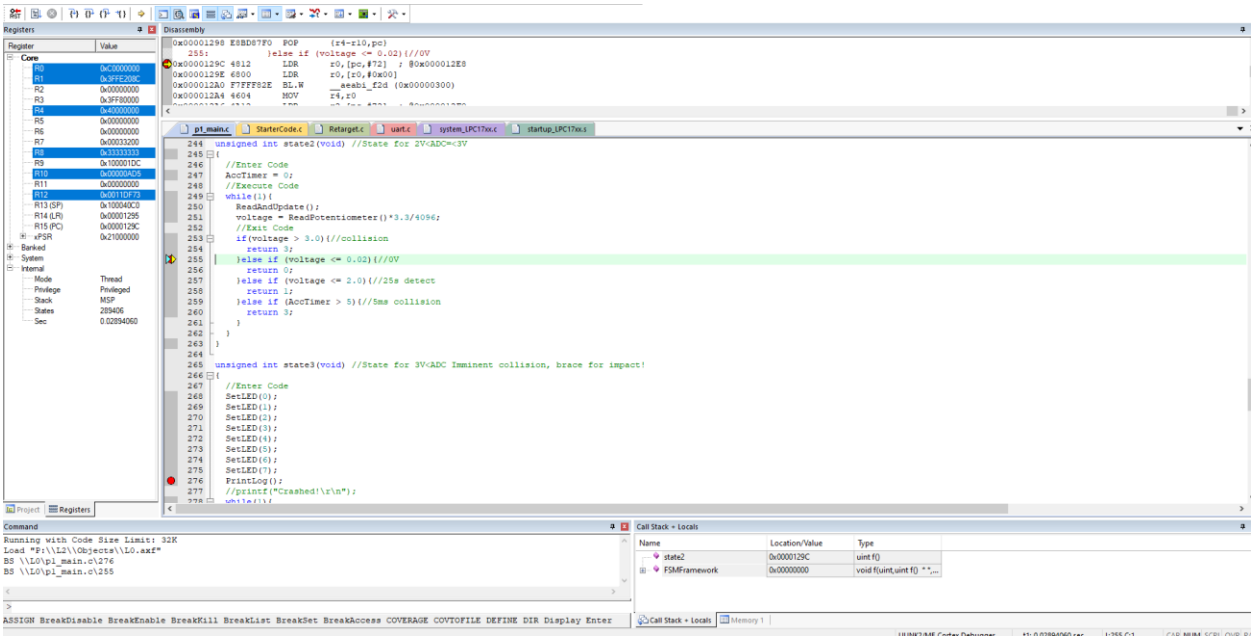
joy == 16 && joy_flag == 0 && JoyFreezeTimer > 200

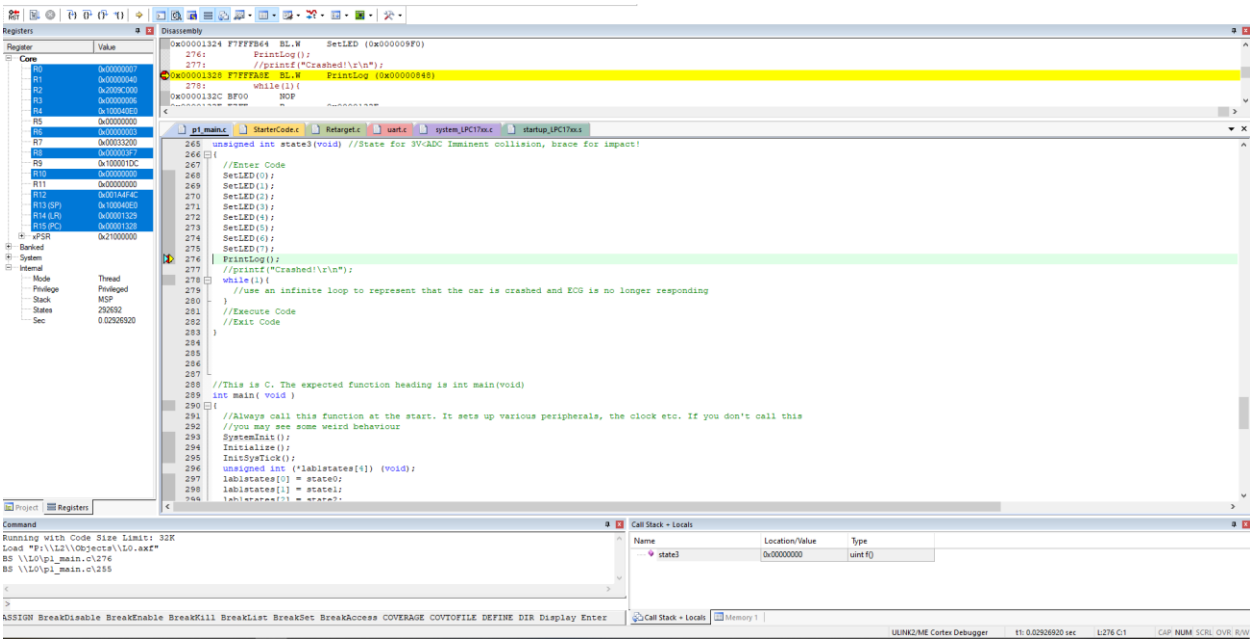
btn_flag == 0

btn != 0 && btn_flag == 0 && BtnFreezeTimer > 200

brake_flag == 1)

Now we just go to experiment the execution time for such a situation in the debugger. Similar to part 1, we set breakpoints and then we get:





The difference in the states is:
292692-289406=3286 (states/cycles)
Also we consider the only interrupt, SysTick_Handler(). It will happen every ms, and takes 64 (gained from experiment similar to above) cycles each time.
The clock freq. is 100MHz. So the time elapsed, i.e. the WCET, is:
(3286+64*1)*10^-8=**$3.350*10^{-5}$(s)**

2. The WCET from the point the accelerometer has been above 2V for 5ms to the LEDs turning on indicating the airbag has activated: This is actually identical to 1. in my implementation as you can see from above! The respective condition-checking if statement switch from *voltage > 3.0* to *AccTimer > 5* (both in State2 where we did the experiment). But this doesn't affect the WCET so it's still **$3.350*10^{-5}$(s).**

3. & 4. The WCET from the point the accelerometer goes above 3V to the end of the final bit of the log being written to the UART & The WCET from the point the accelerometer has been above 2V for 5ms to the end of the final bit of the log being written to the UART: Similar to 1. & 2., these are still the same. And they are basically **$3.350*10^{-5}$(s) + WCET to run PrintLog().** *PrintLog() is the function in charge of printing out the log, so its execution time heavily depends on the log's length and therefore unpredictable. But I tried a log with a practical length of 20+ items and the execution time for PrintLog() was:

(52618573-52412082)*10^-8 = 206491*10^-8 = $2.06491*10^{-3}$ (s) = 2.06491(ms)

so the total WCET for 3. & 4. Should be reasonable (i.e. within 10 ms).