Due on 12/3, submit a single file as doc, docx, or pdf in Canvas.

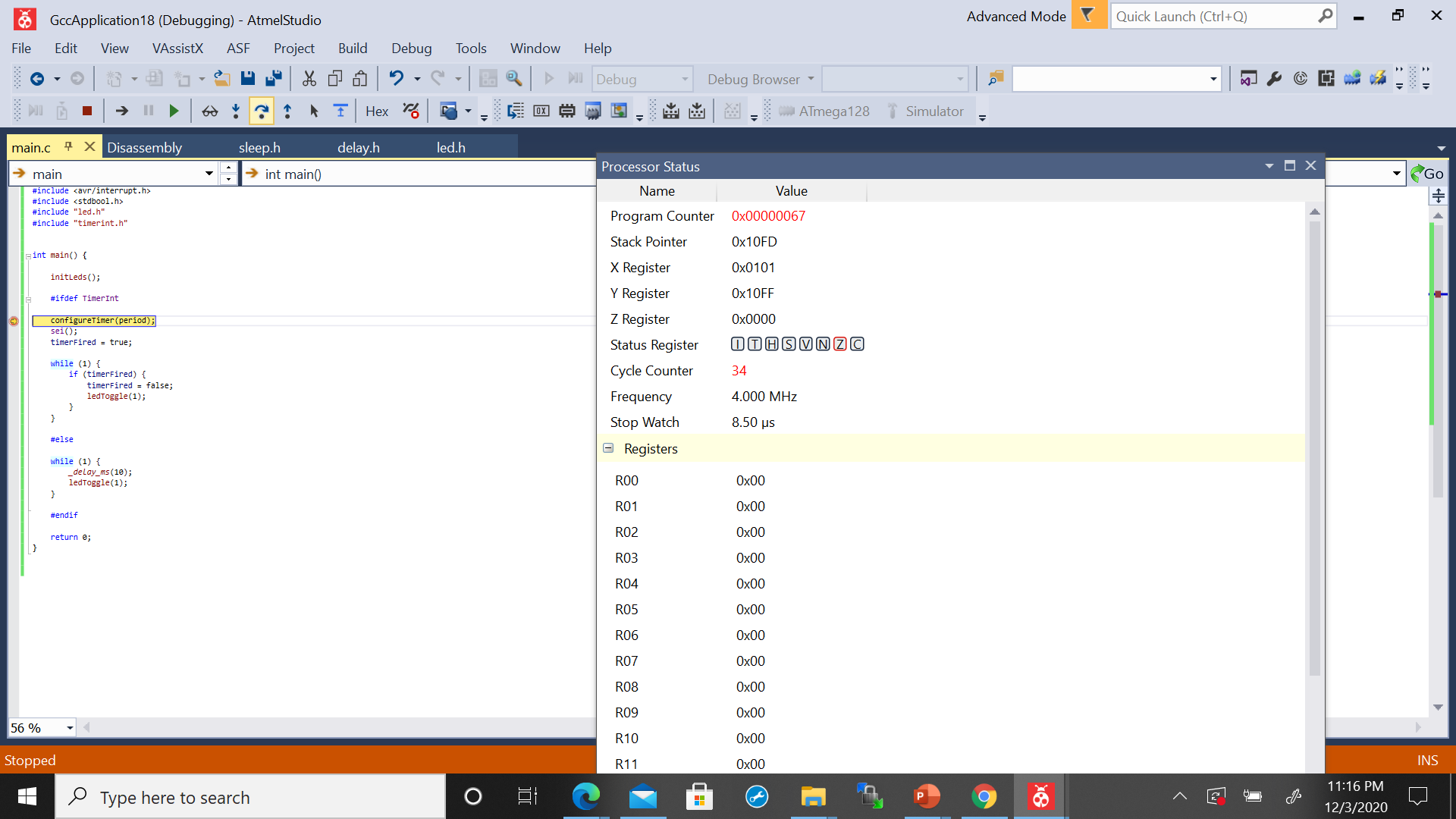
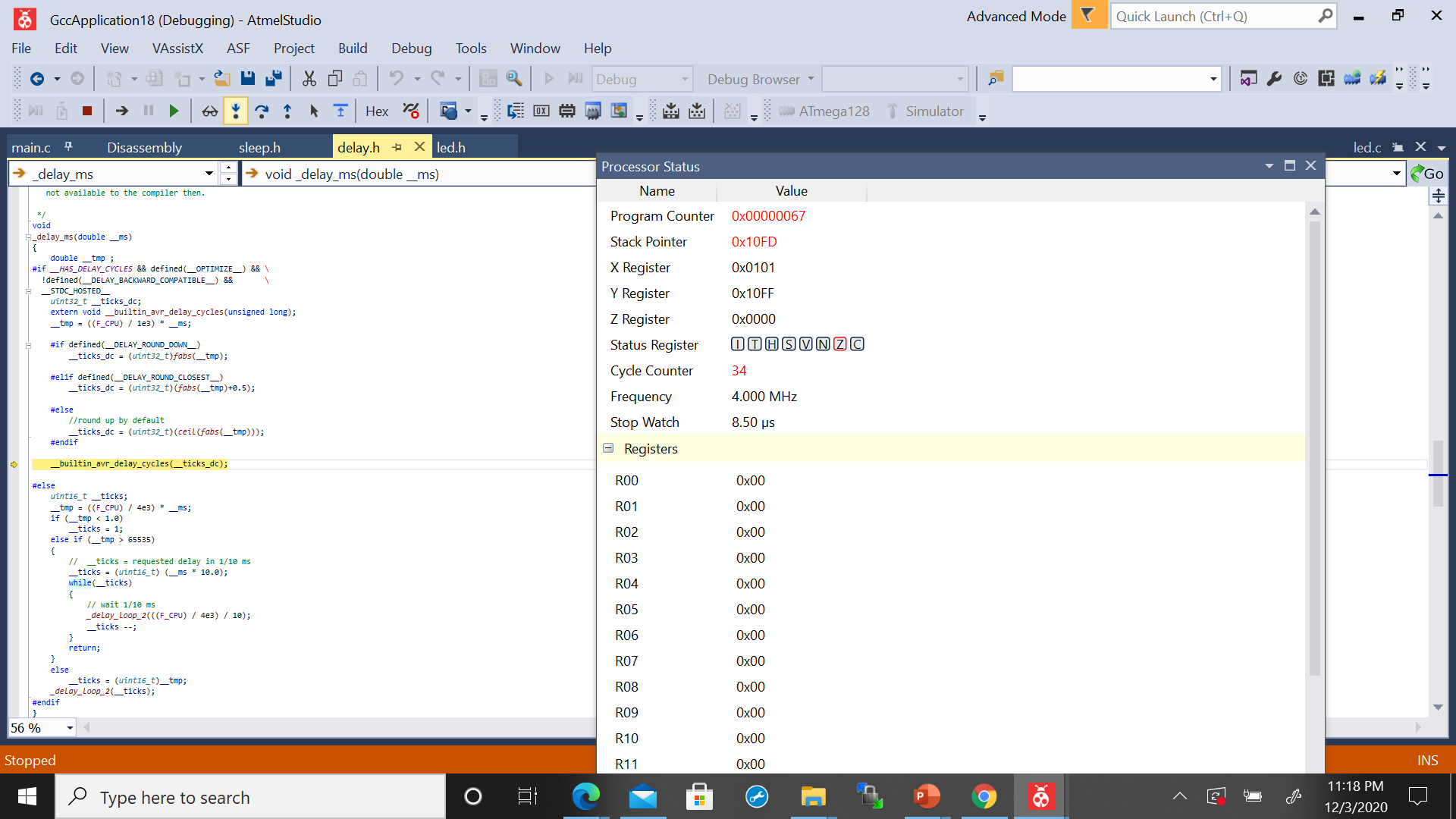
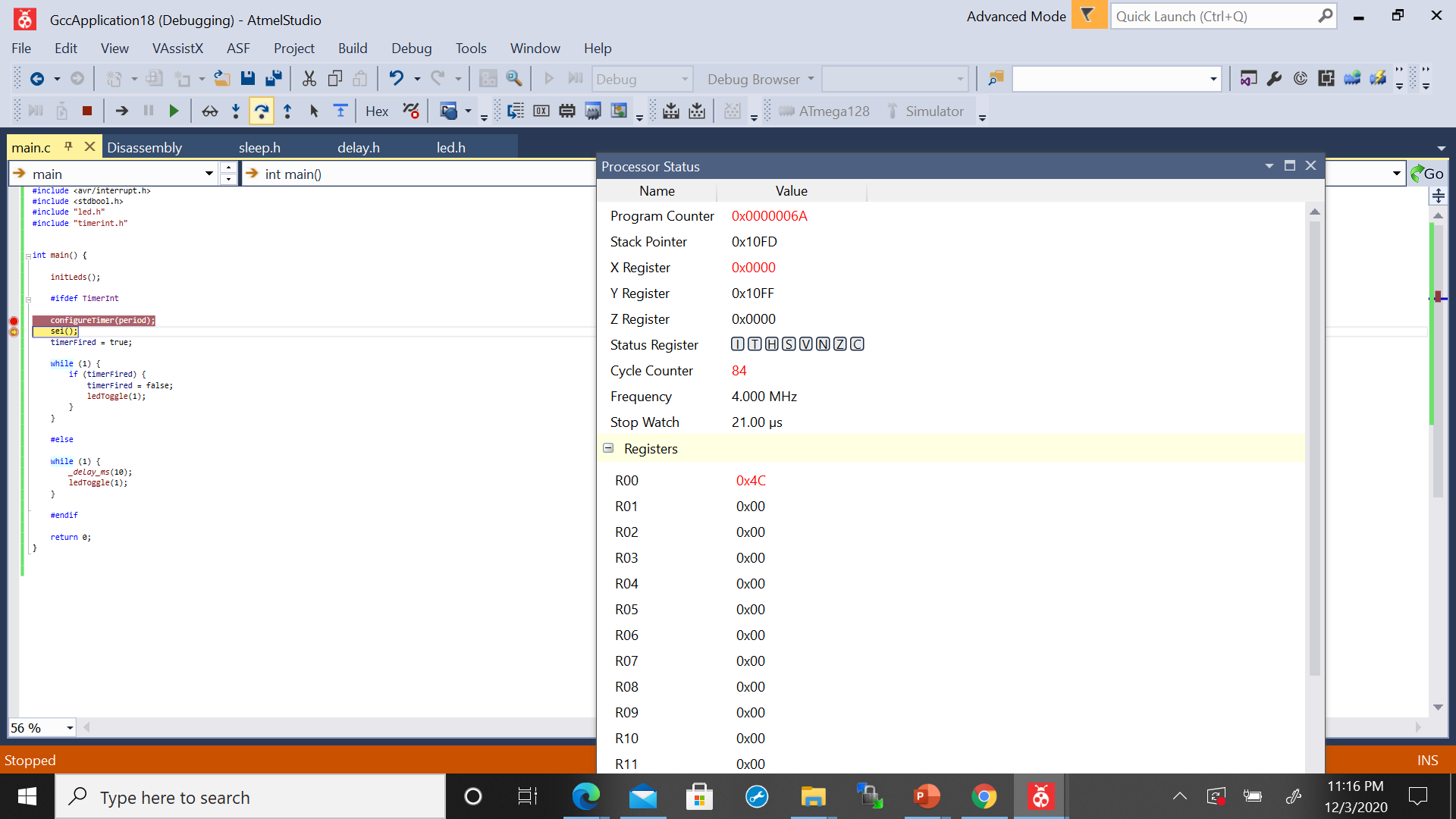
1. (20%) Run the code of hw6.q1 in the simulator of Atmel Studio.

Use the stopwatch to profile the time of ledToggle, \_delay\_ms, and configureTimer.

a) Describe how you profile their times, such as where to set break points and how many profiling trials.

I set the breakpoints before the function and after the function ends. Then take the difference to find the time it takes.

b) Show the profiling results in report.



2. (20%)

a) Implement the bar function in hw6.q2.c. The bar function should do the same job as the foo function, but unroll the while loop of the foo function.

void foo(uint8\_t\* dst, uint8\_t\* src, uint8\_t len) {

while (len--) {

dst[len] = src[len];

}

}

// implement bar to unroll the loop in foo

void bar(uint8\_t\* dst, uint8\_t\* src, uint8\_t len) {

while(len--){

dst[len] = src[len];

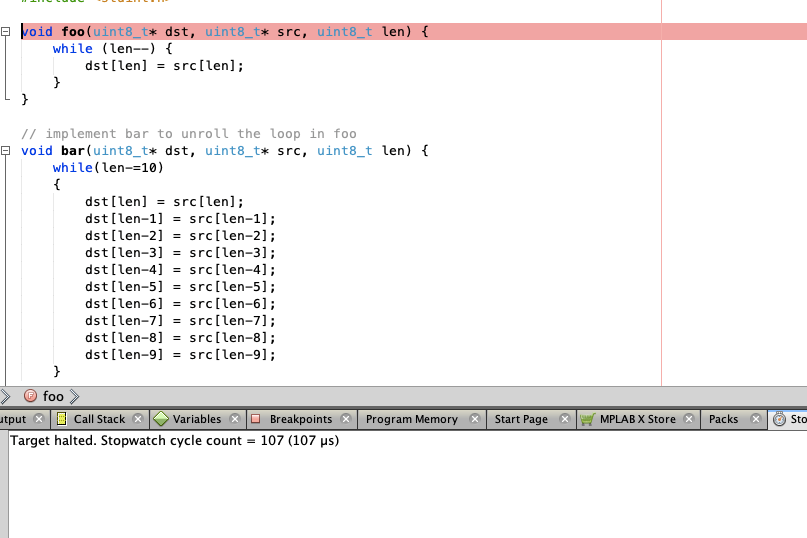
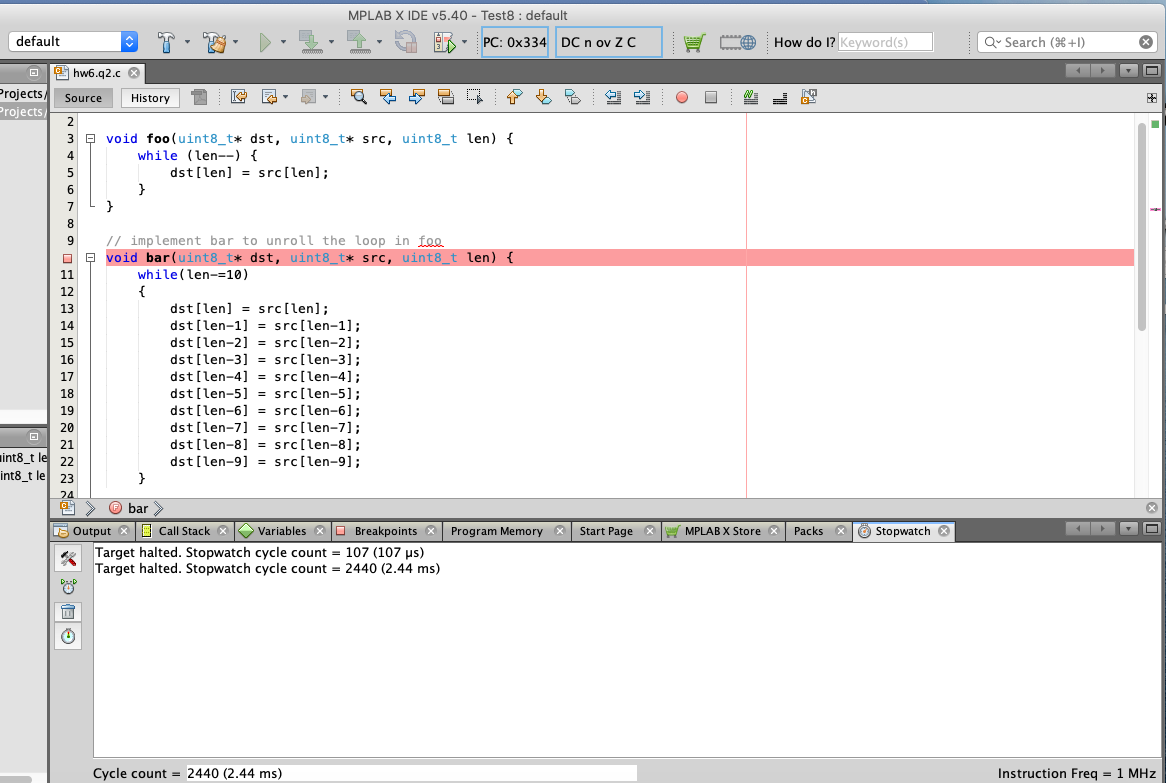
len--;

dst[len] = src[len];

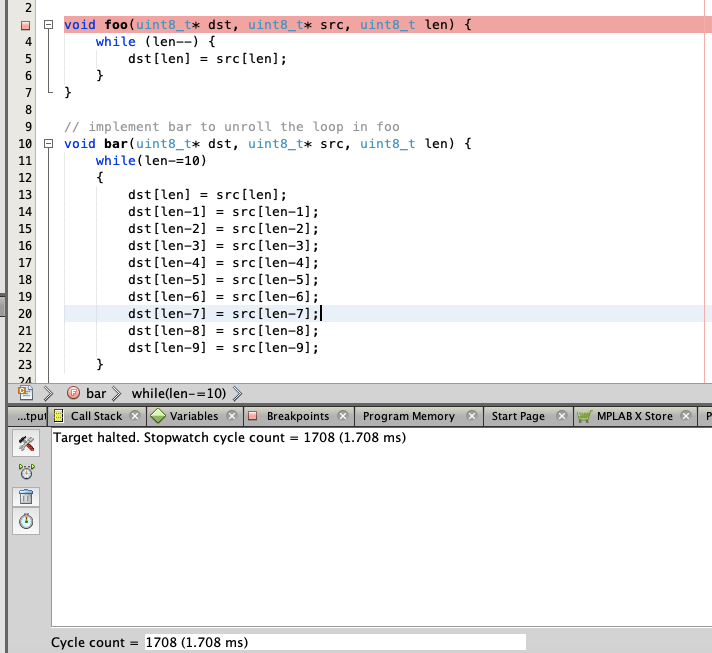
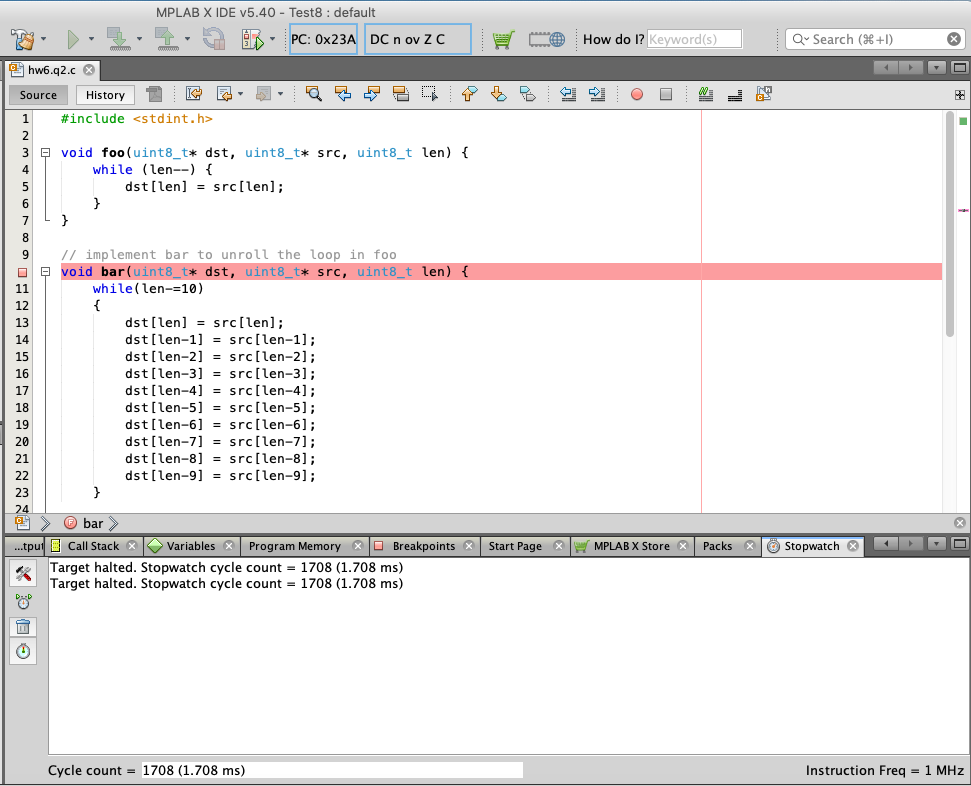
}

}

b) Compile hw6.q2 with no optimization option O0 and run the program in MPlab X. Use stopwatch to measure the times of running the foo function and the bar function. Report the times of the two functions.

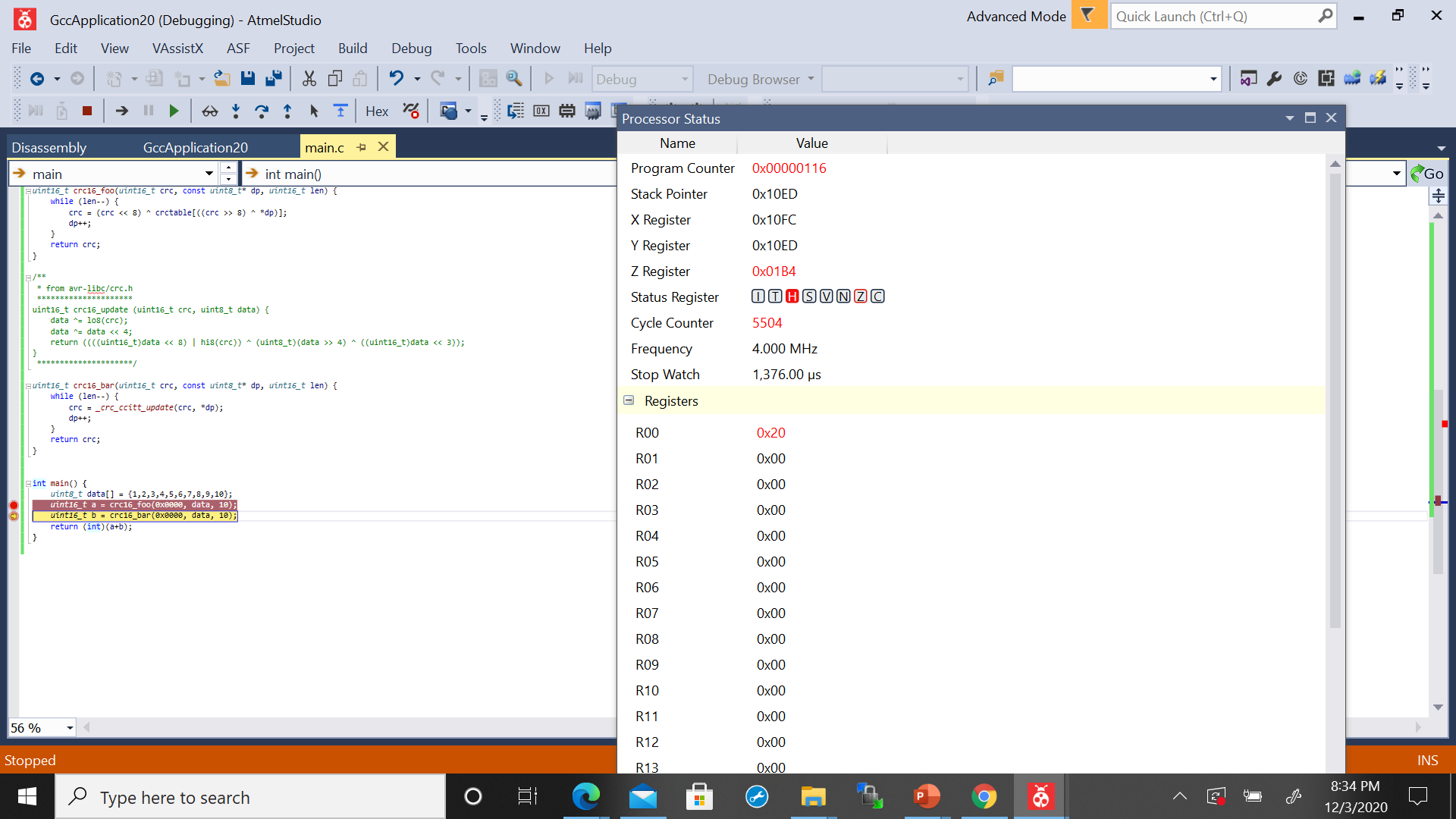
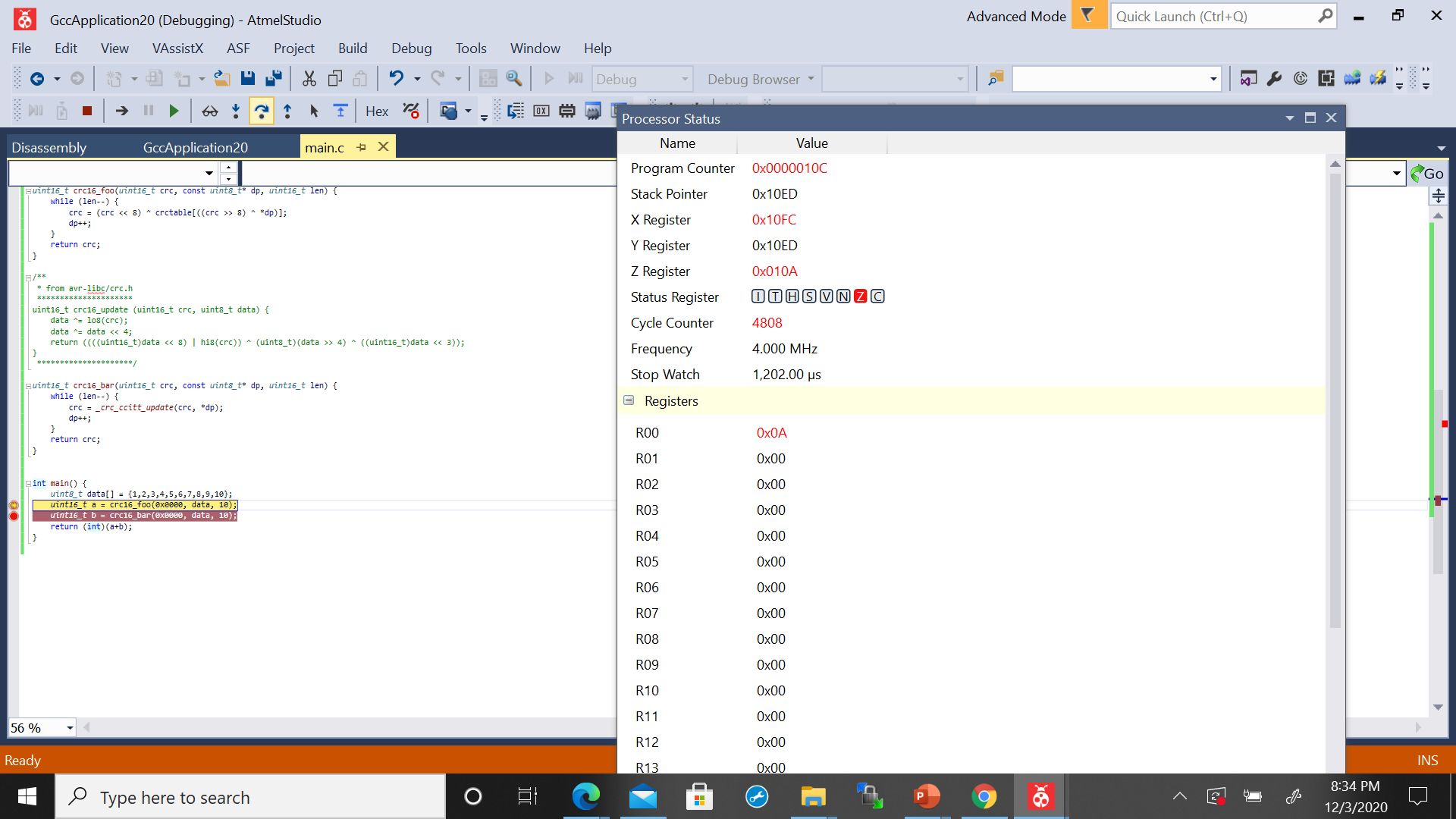


c) Compile hw6.q2 with optimization option O1 and run the program in MPlab X. Use stopwatch to measure the times of running the foo function and the bar function. Report the times of the two functions.



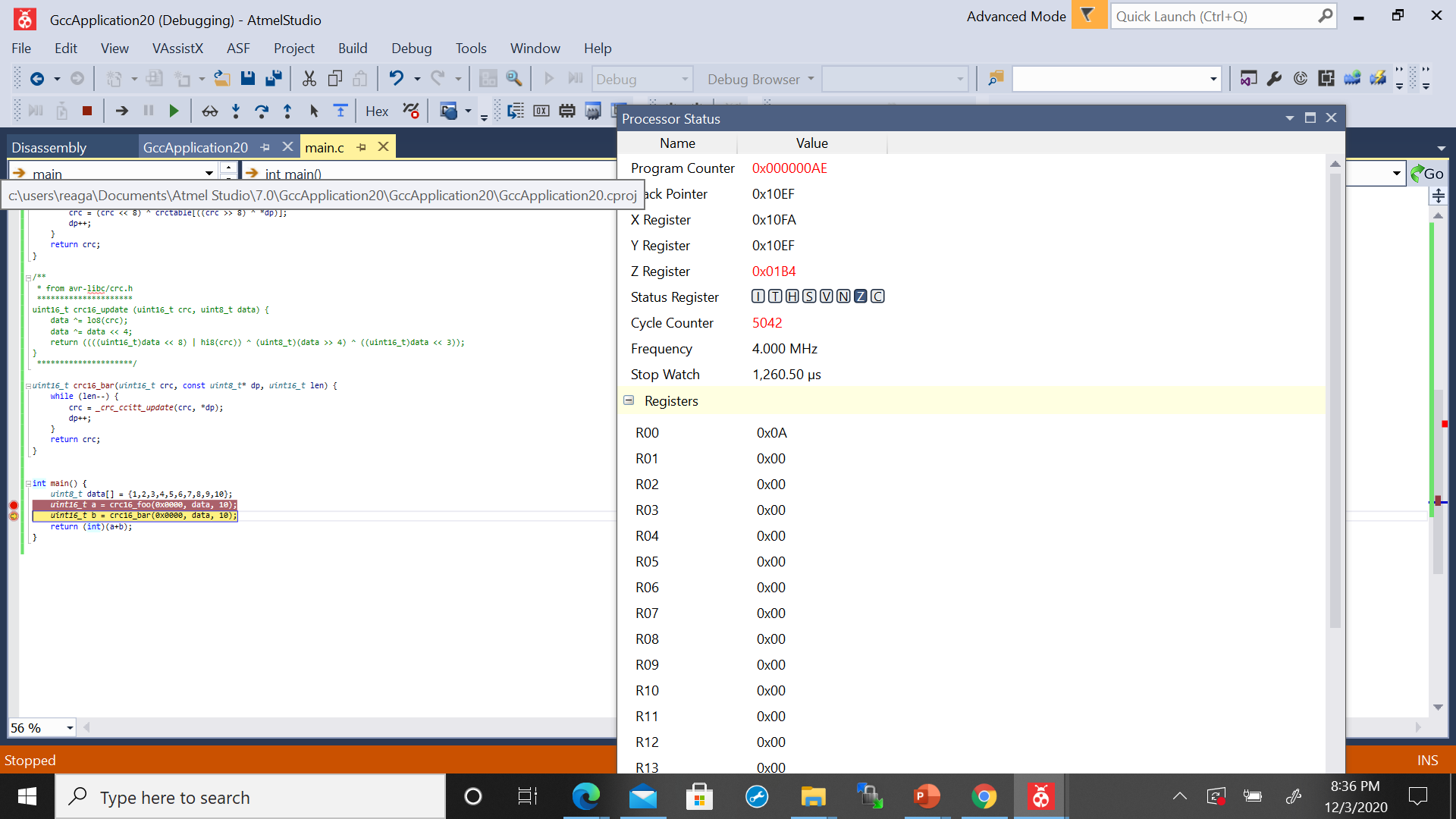
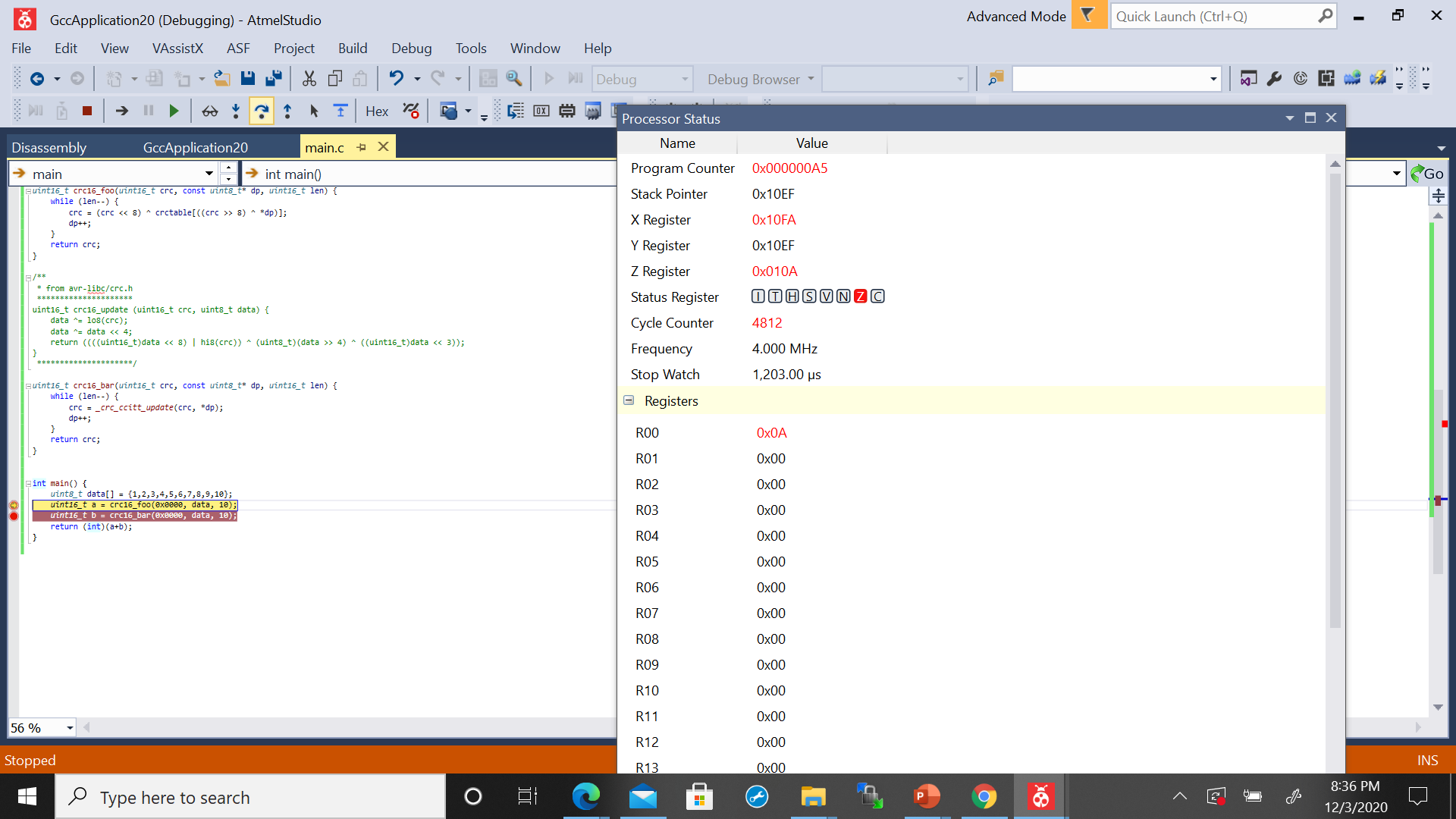
3. (20%)

a) Run the program in hw6.q3 with no optimization option O0 in Atmel Studio. Use stopwatch to measure the times of the crc16\_foo function and the crc16\_bar function. Report the times of the two functions.



1,376.00µs - 1,202.00µs

= 174µs

b) Run the program in hw6.q3 with optimization option O1 in Atmel Studio. Use stopwatch to measure the times of the crc16\_foo function and the crc16\_bar function. Report the times of the two functions.

1,260.50 - 1,203.00µs

= 57.50µs

c) You may observe that using table is faster than no table with O0, but using table is similar to no table with O1. Analyze the reason for this observation.

This is because uint16\_t is a 16 bit unsigned integer. This allows code to be portable among machines. It is faster than code without uint16\_t with O0 or O1 because optimization makes the compiler attempt to improve the performance at the compilation time but it can not be decided how much memory will be taken by a variable of a particular data type at compile time.

4. (25%)

An application has three events: E1, E2 and E3. The application runs in an embedded system with a 40MHz clock. E1 occurs every 100ms and needs 20ms for execution. E2 occurs every 30ms and needs 3ms for execution. E3 occurs every 40ms and needs 3ms for execution.

a.1) Over 3 seconds, how many times do E1, E2 and E3 occur respectively, and how much execution time do E1, E2 and E3 need respectively?

E1 occurs 30 times at 600ms

E2 occurs 100 times at 300ms

E3 occurs 75 times at 225ms

Total execution time = 1.125sec

a.2) Can E1, E2 and E3 be scheduled in this application?

Yes E1,E2,E3 can be scheduled in this application

Power is proportional to frequency, i.e. P=cF. Assume the operating power at 40MHz is 40mW and the idle power at any frequency is 0.1mW.

b.1) To reduce power, the application wants to lower the clock frequency. The three events will occur at the same rate, but their execution times will be enlarged. What is the minimum frequency that the three events can be scheduled?

Sum = (20ms/100ms) + (3ms/30ms) + (3ms/40ms)

Sum =0.375

= 40MHz \* 0.375

= 15MHz

b.2) Over 3 seconds, how much energy is consumed at 40MHz.

Operating time: 1.125sec =600ms+300ms+225ms

Operating power: 40mW

Idle power: 0.1mW

Total idle time = Total time - Total execution time

Total idle time = 3sec - 1.125sec

Total idle time = 1.875s

Energy = ((40mW \* (1W/1000mW)\*1.125sec)+((0.1mW\*(1W/1000mW)\*1.875sec)

Energy = 0.0451875Joules

b.3) Over 3 seconds, how much energy is consumed at the minimum frequency?

Energy = ((15mW \*(1W/1000mW)\*1.125sec)+((0.1mW \* 1W/1000mW)\*1.875sec))

Energy = 0.017063Joules

5. (15%)

The program below uses sleep to save power when no events to handle. Read http://www.nongnu.org/avr-libc/user-manual/group\_\_avr\_\_sleep.html

a) Find and list all sleep modes defined for Atmega128 in avr/sleep.h

#define SLEEP\_MODE\_IDLE (0)

#define SLEEP\_MODE\_ADC \_BV(SM0)

#define SLEEP\_MODE\_PWR\_DOWN \_BV(SM1)

#define SLEEP\_MODE\_PWR\_SAVE (\_BV(SM0) | \_BV(SM1))

#define SLEEP\_MODE\_STANDBY (\_BV(SM1) | \_BV(SM2))

#define SLEEP\_MODE\_EXT\_STANDBY (\_BV(SM0) | \_BV(SM1) | \_BV(SM2))

|  |  |
| --- | --- |
|  |  |
|  | |
|  |  |
|  | |
|  |  |
|  | |
|  |  |
|  | |
|  |  |

b) Implement the setSleep() function for Atmega128 so that the CPU goes idle when sleeping. Copy and paste code in your report.

void setSleep(){

set\_sleep\_mode(SLEEP\_MODE\_IDLE);

}

c) Implement the sleep() function for Atmega128. Copy and paste code in your report.

void sleep(){

sleep\_mode();

}

int main() {

setSleep();

initQueue(&queue);

while (1) {

// queue loop

while (!isEmpty(&queue)) {

// process the events in queue

uint8\_t event = pop(&queue);

handleEvent(event);

}

sleep();

}

}