107062381 楊孝偉

Checkpoint 4 Report

Testpreempt.c

In this file, I will two variables from the previous checkpoints: buffer and token. Buffer is used for the shared buffer between the producer and consumer. Token is used to generate characters from A to Z for the producer. There are also other additional variables : head and tail to keep track of the circular queue, mutex, full and empty for the semaphore variables.

In the main function, Token is initialized to A. Main will spawn consumer thread, while main itself calls producer. This is done to not waste main's thread, more economical. We also need to create three new semaphores : full, mutex and empty. Set full to 0, mutex to 1, and empty to 3 (since we have 3 deep buffer). I followed the materials taught during the lectures. Then we set head and tail to 0 to indicate that both of them point to the initial position of the 3 deep buffer.

In the producer function, before it enters the infinite loop, I initialized the token to 'A' first, to make sure that token is really initialized to 'A'. After it enters the loop, it will perform semaphorewaitbody for empty and mutex to check whether there is still an empty space in the buffer and whether it's the producer turn. Buffer will take the value in token, then update the value in token. When buffer is taking the value of token, it should be in critical section since SDCC suggests that you can surround the code fragment where they access the shared variable, in this case it's buffer, to ensure that the shared variables are accessed atomically. Updating the tail value should also be inside the critical section since if it's not, then interrupt might interrupt it before tails could update its index position and this might cause an unwanted error. Then signal the mutex to indicate that it's the consumer turn and signal full to indicate that buffer has been fully filled.

In the consumer function, before it enters the infinite loop, we initialize Tx for polling. After it enters the loop, it will semaphorewaitbody the full and mutex semaphores to check whether the data in buffer have been consumed and it's the consumer turn. SBUF(reading or writing register) will take the value in buffer, then we update the value of head, then we signal the mutex and empty semaphores for the producer, then we check if Tx is busy (serial port hasn't finished writing it yet), then we keep polling again. Finally, we reset It to 0 again.

Preemptive.h

I added a semaphore API to this file.

The SemaphoreCreate function the semaphore s to the value n.

The SemaphoreSignal only increment the value in the semaphore variable.

The SemaphoreWait basically reads the value of S into ACC then checks if ACC is <= 0 o not.

Test3threads.c

This file is mainly based on the test3threads.c. Here we have two producer functions, one function will produce an output from 'A' to 'Z' and the other will output '0' to '9'. At first, before I fixed this semaphore

thing, if I placed the producer1 above producer 2, then it will only output producer 1, and if I placed producer 2 above producer 1, then it wil output only producer 2.

Therefore, in main function, we need two more semaphores called producer1_ready which is initialized to 1 so that producer 1 will run first and producer2_ready which is initialized to 0 so that producer 2 will not run before producer 1 finish. Token2 is initialized to 0. Then we create two threads for producer 1 and producer 2.

In producer1, it's basically the same as the previous checkpoints, it's just now we add semaphorewaitbody producer_1 ready to check whether it's producer 1's turn. After we finish, then we signal producer_2 ready to indicate that now it's producer2's turn.

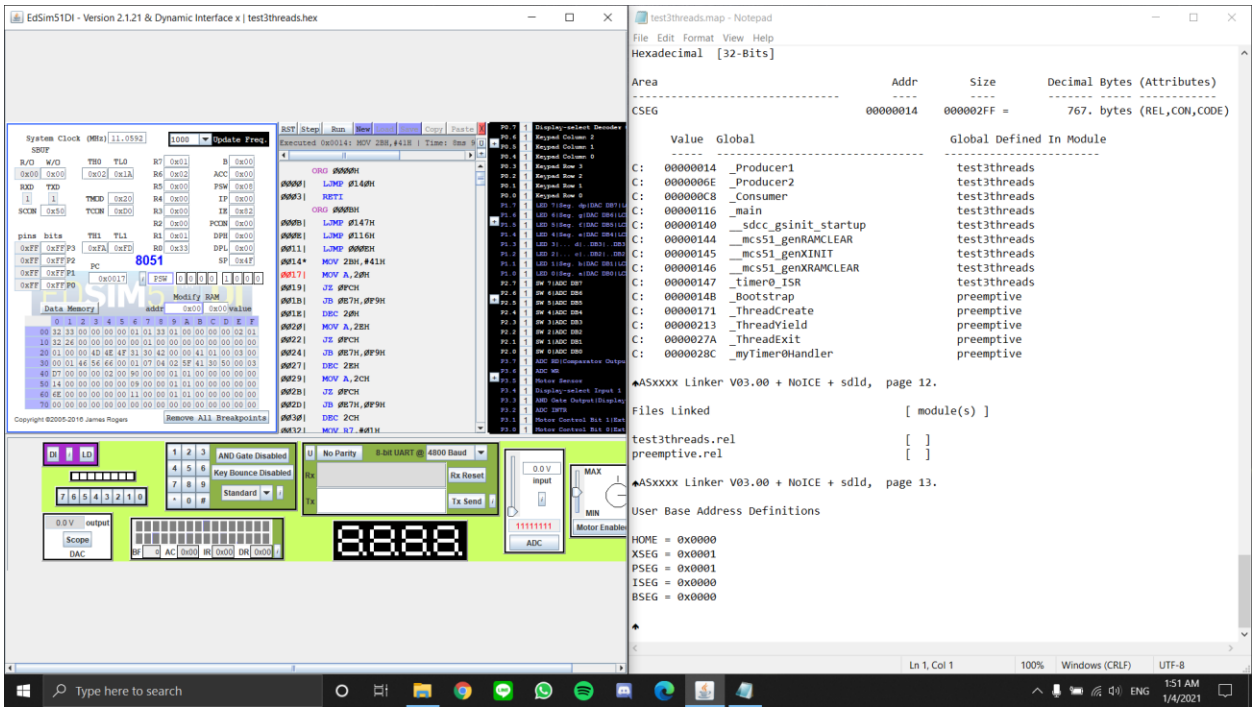
In producer2, it's also the same idea as producer 1. We add semaphorewaitbody for producer_2 to check whether now it's producer 2's turn or not. After we finish all, then we signal producer_1 ready to give the turn to producer 1.

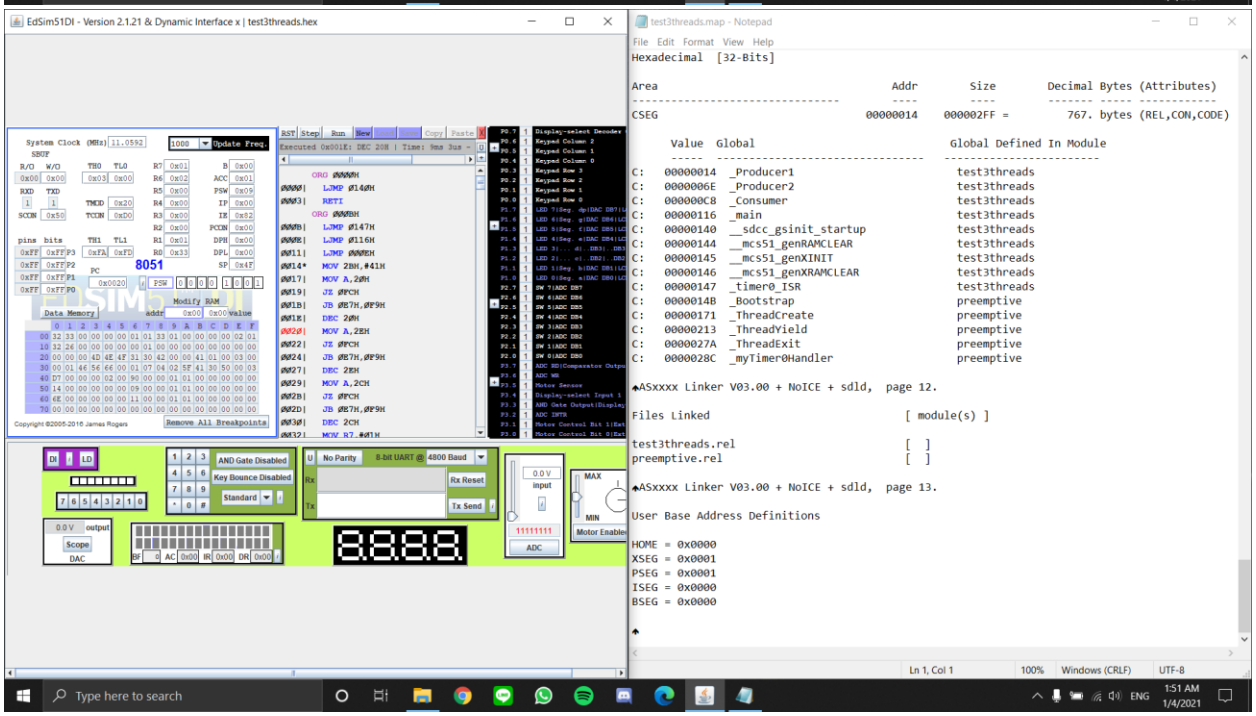
The rest are the same as checkpoint 3.

Breakpoints

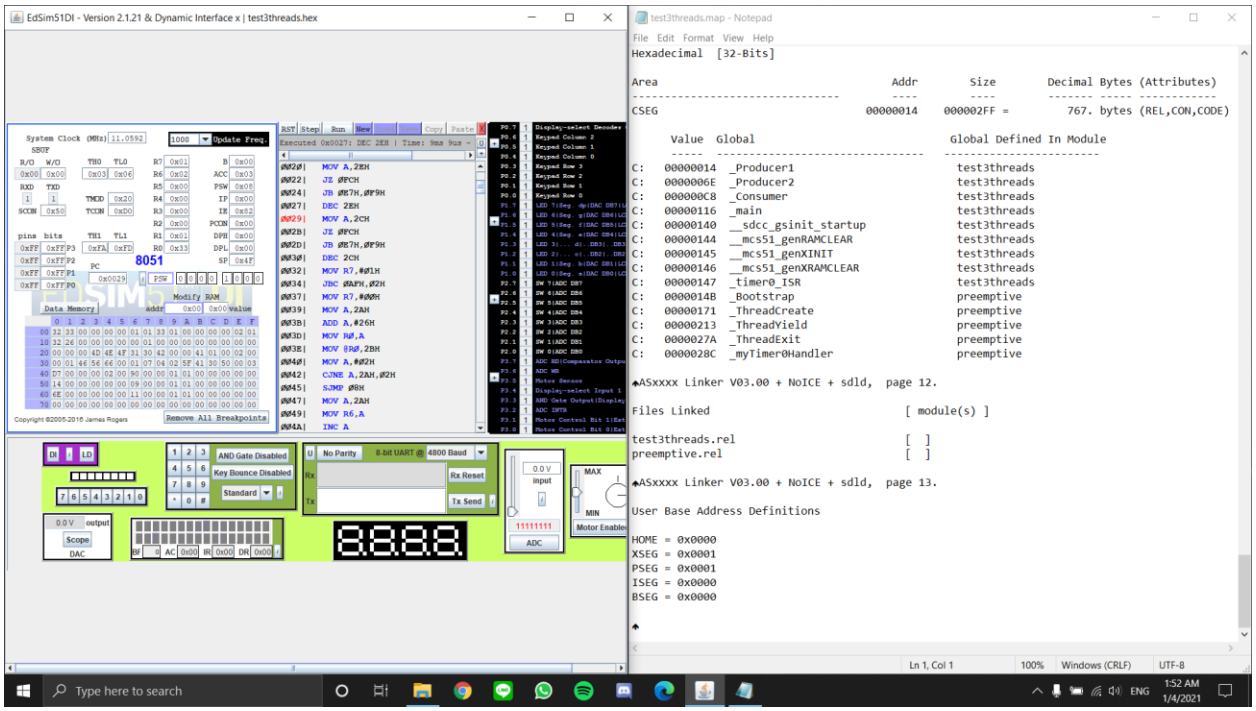
1. Producer1 is running and show semaphore changes

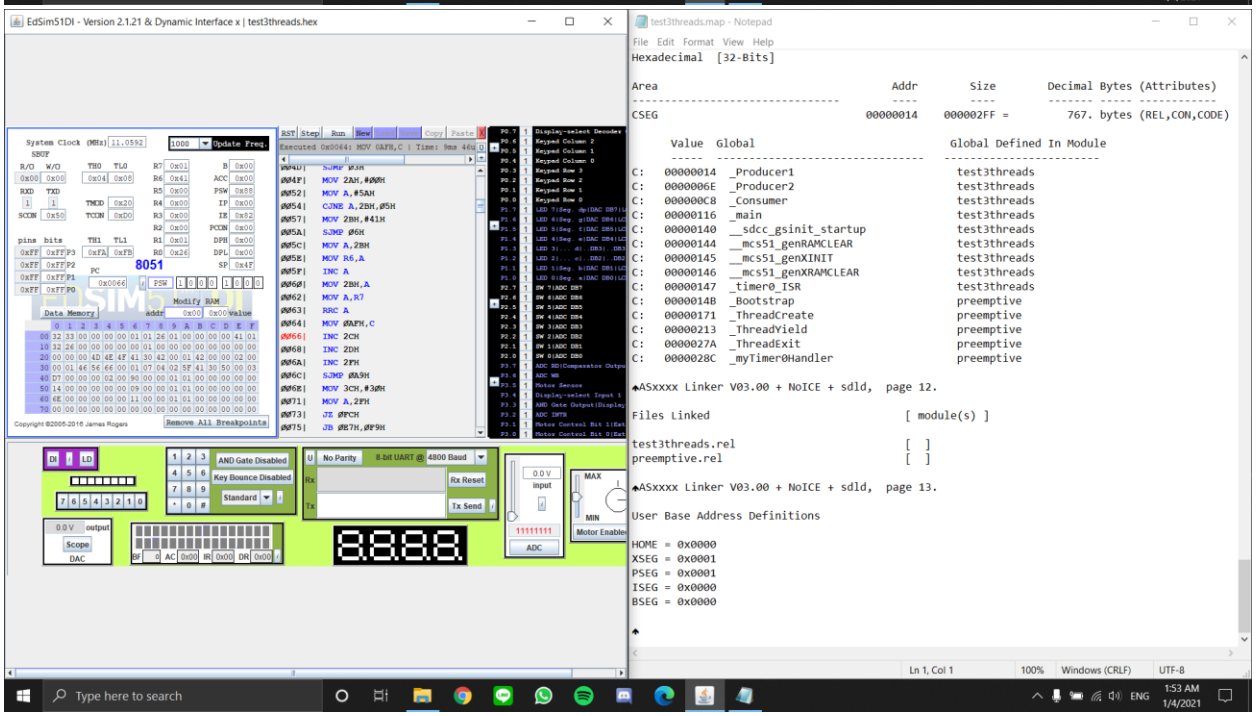
107062381 楊孝偉



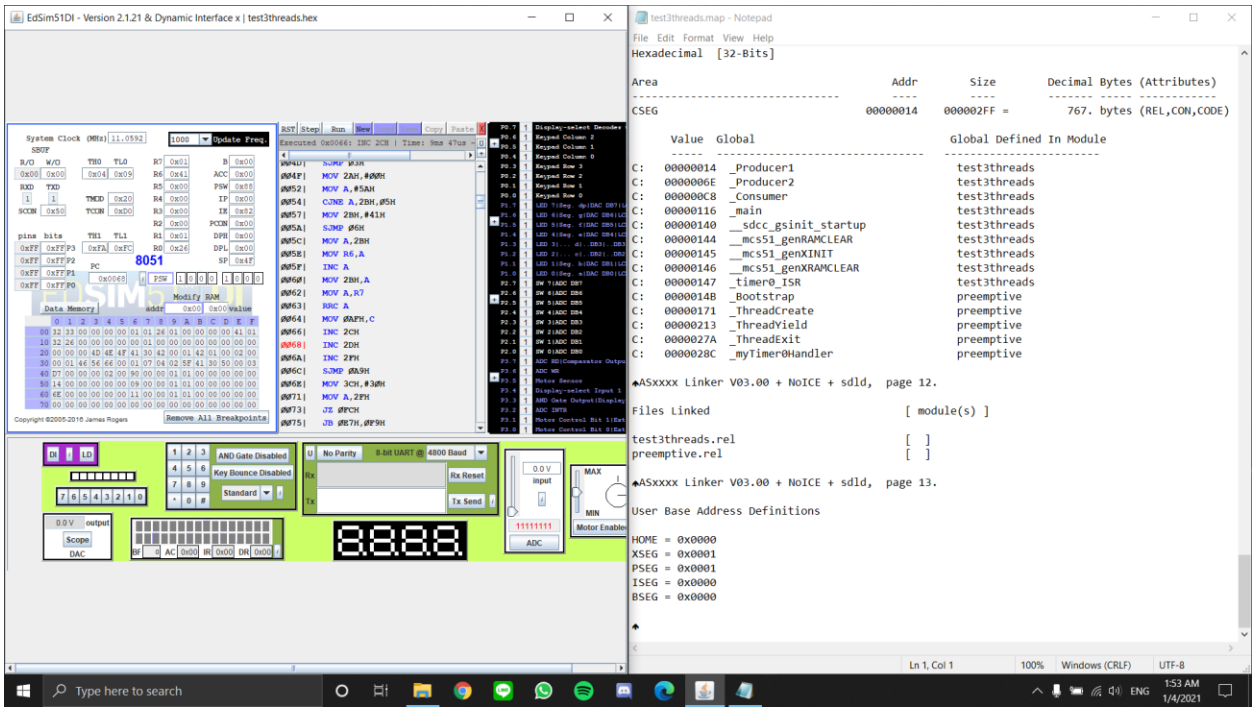


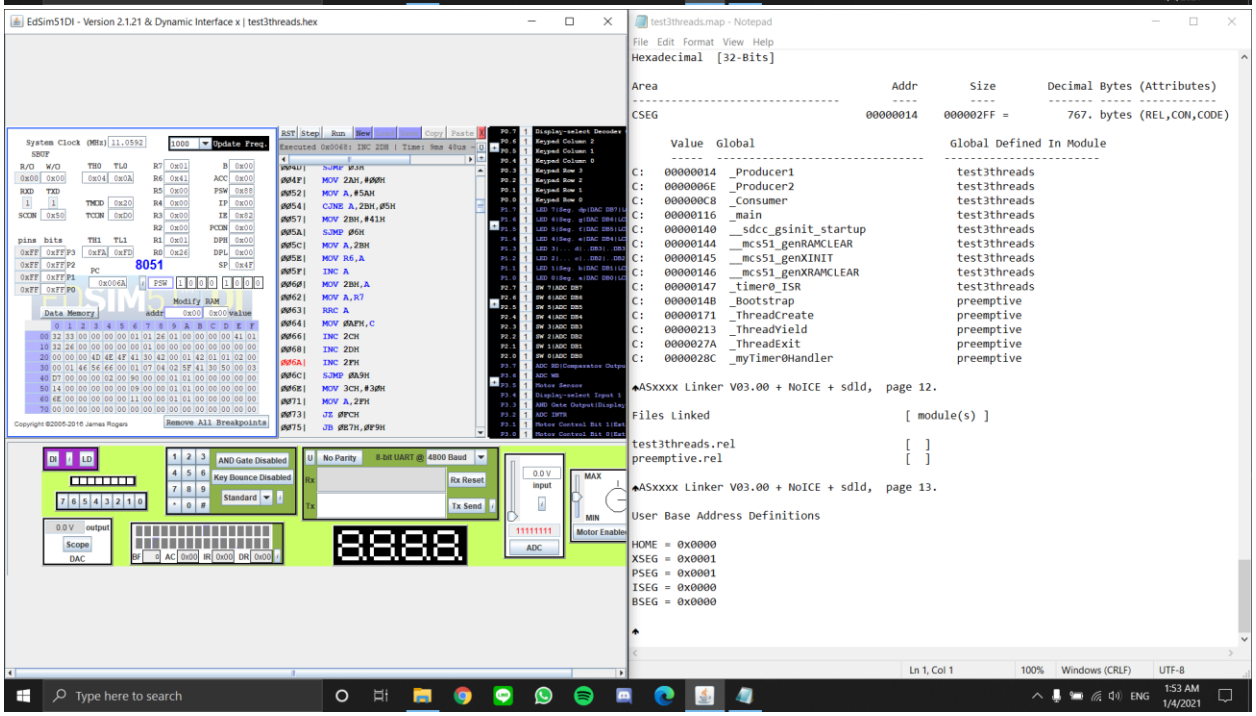
107062381 楊孝偉





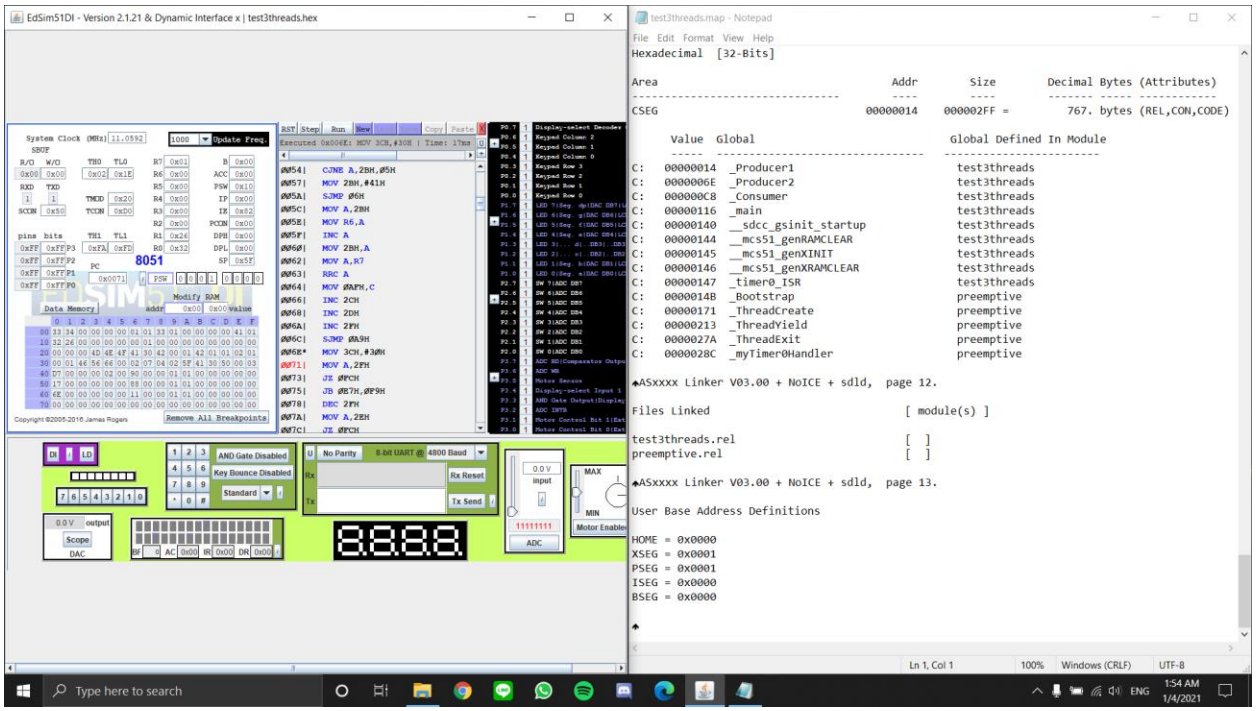
107062381 楊孝偉

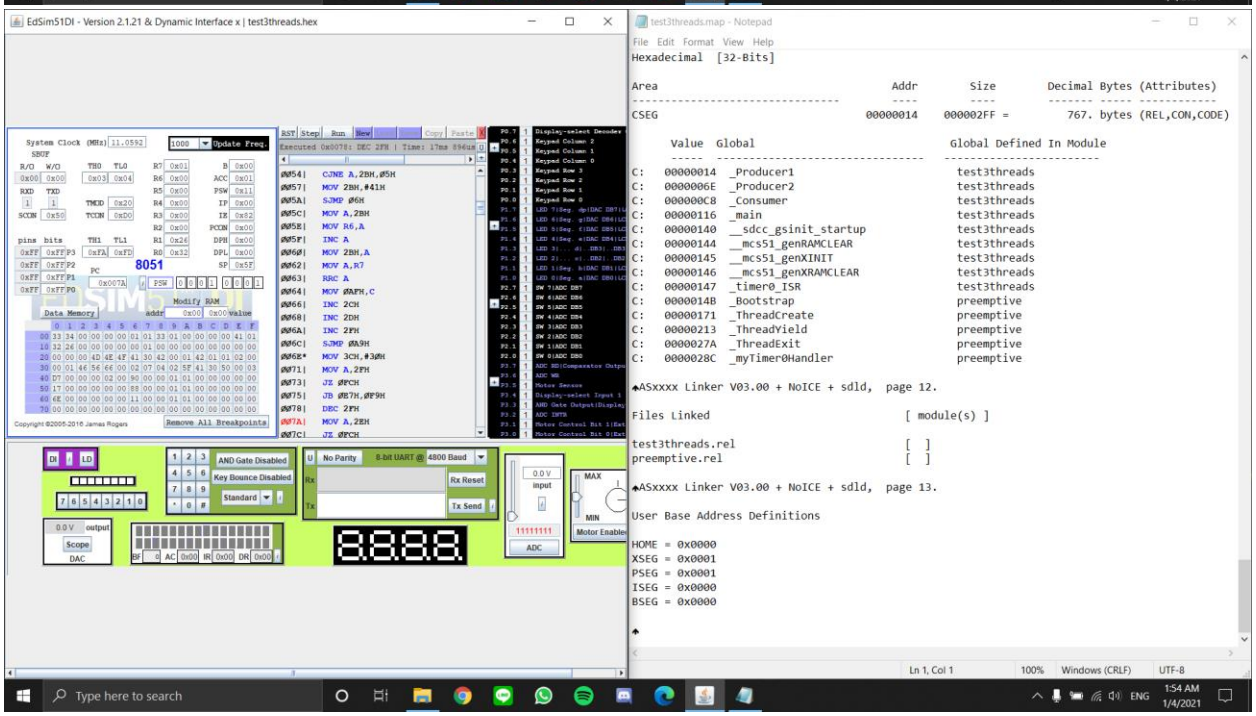




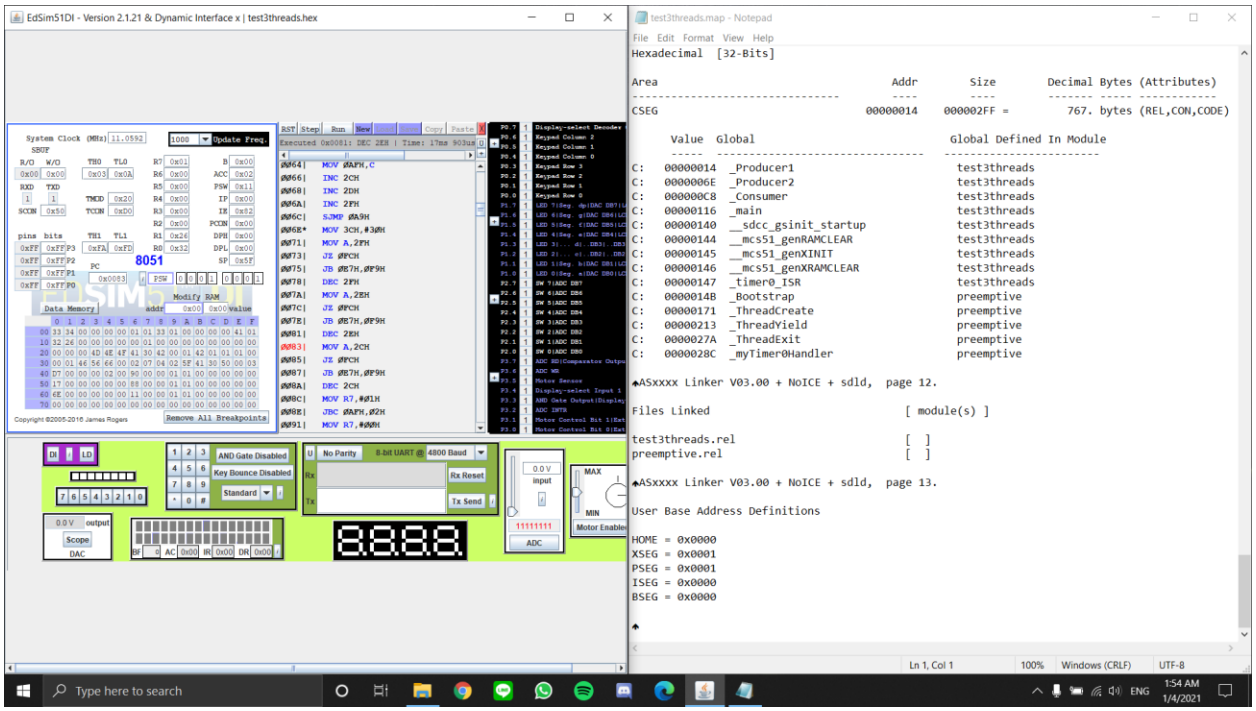
2. Producer 2 is running and show semaphore changes

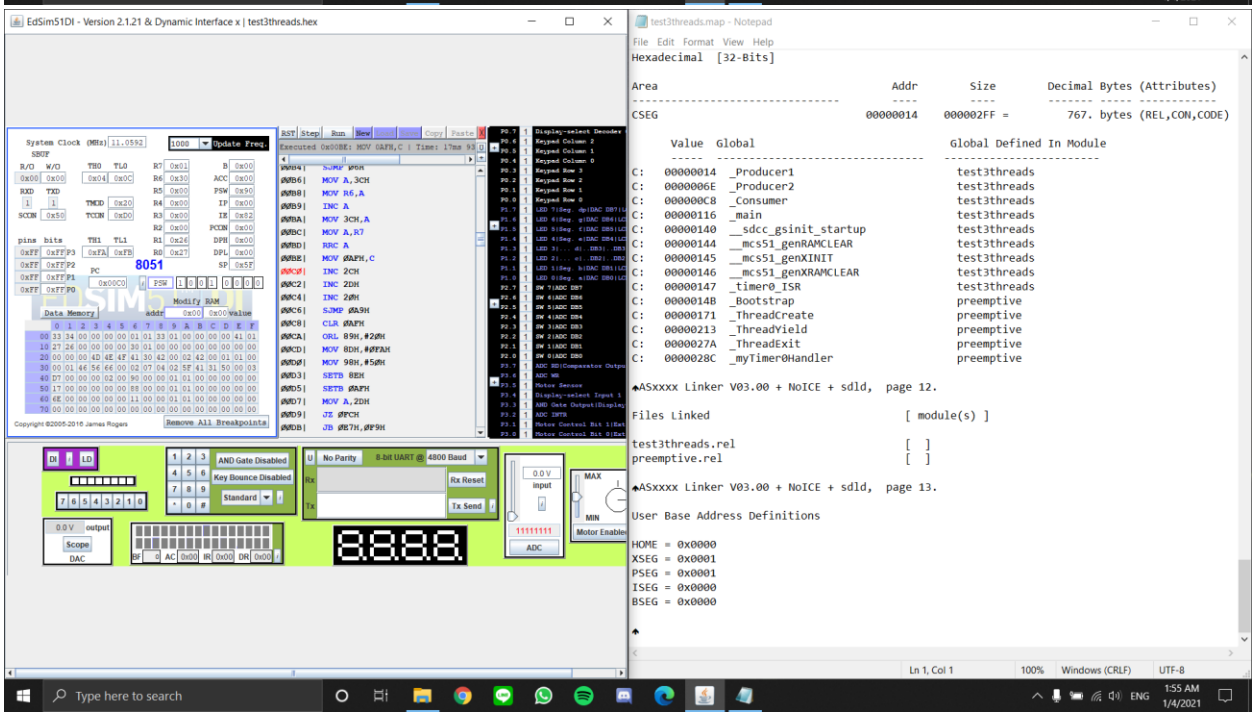
107062381 楊孝偉



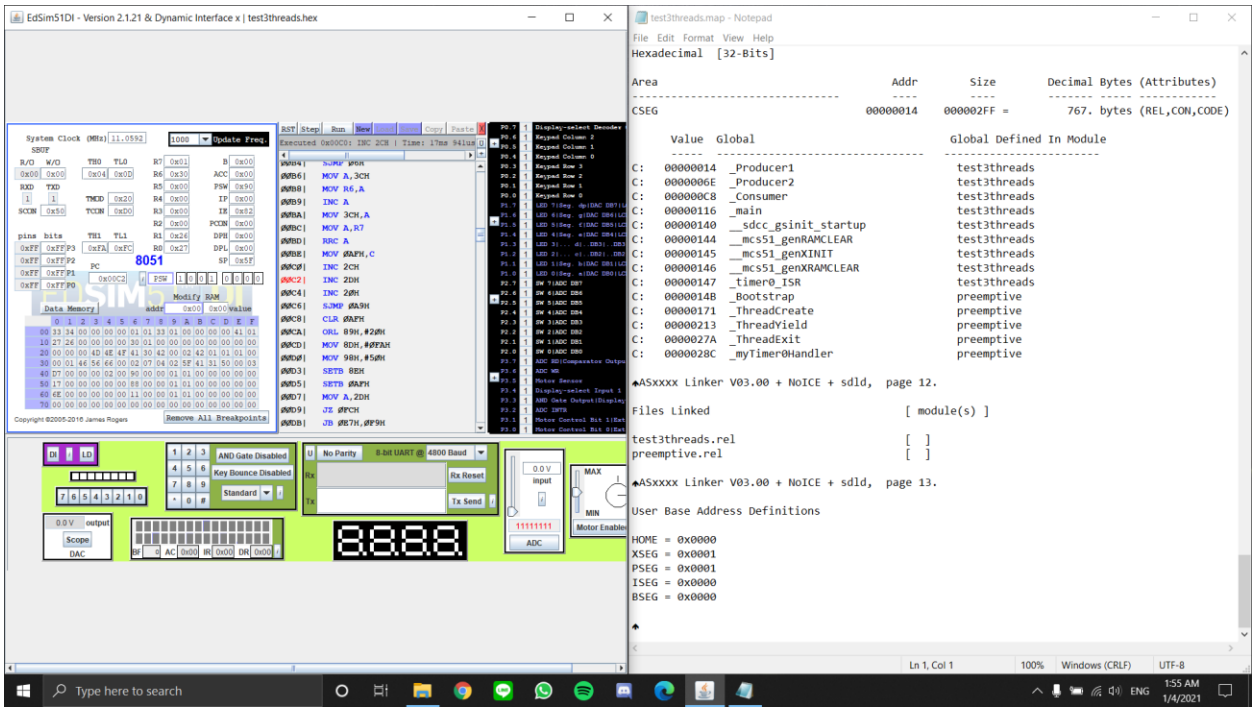


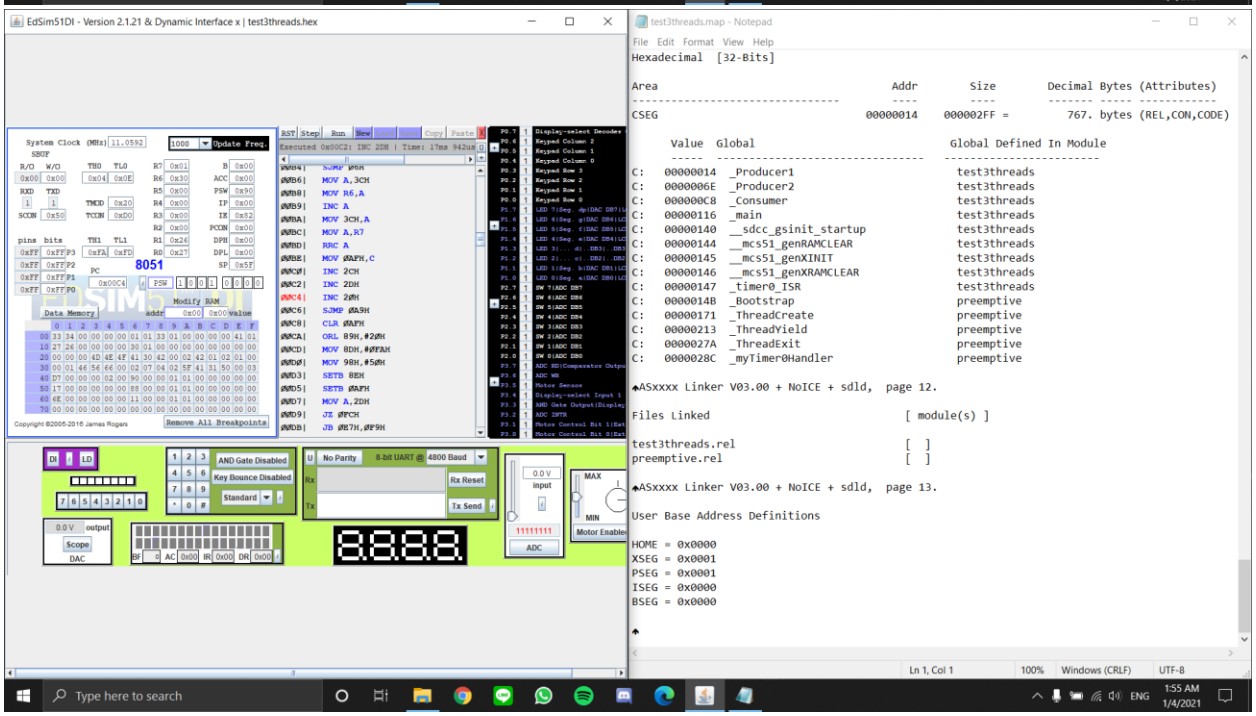
107062381 楊孝偉





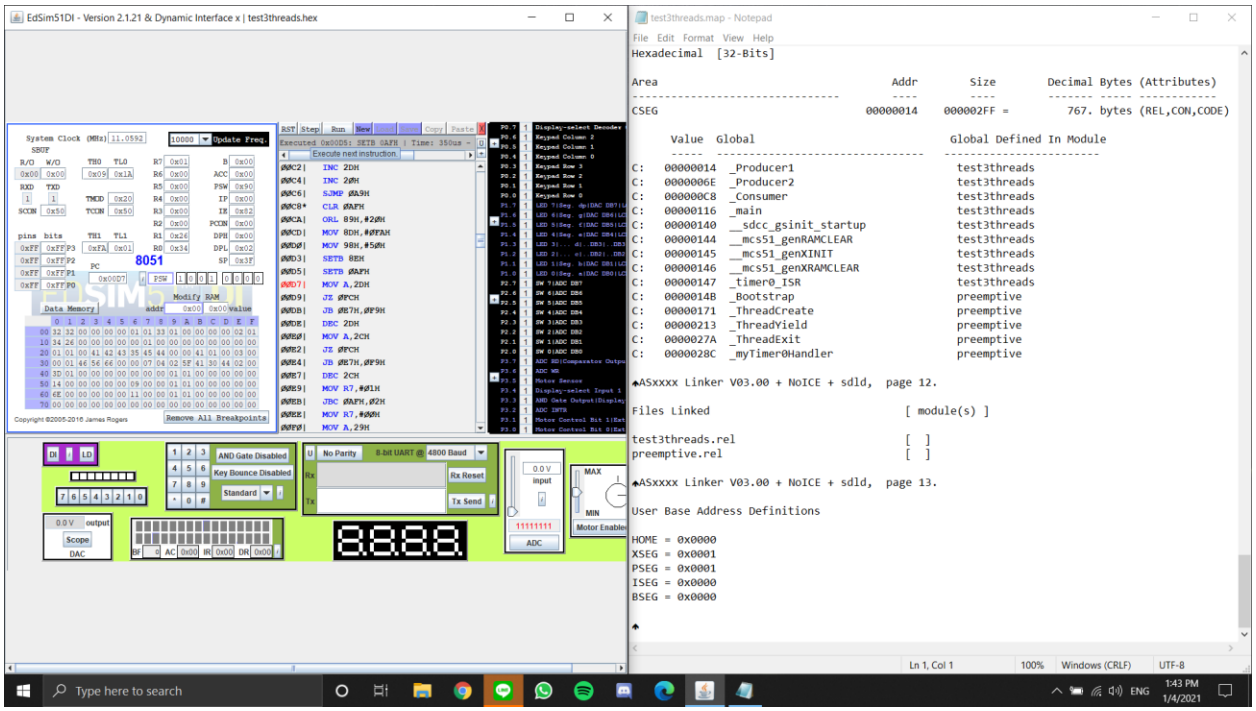
107062381 楊孝偉

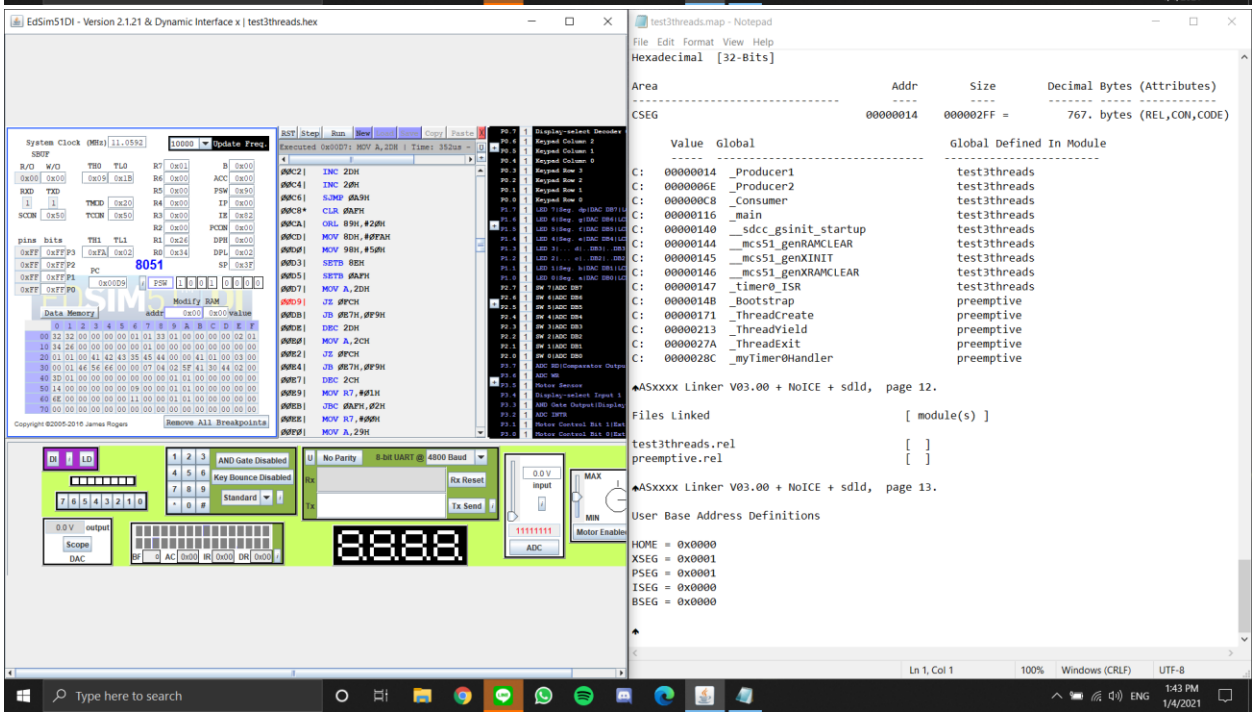




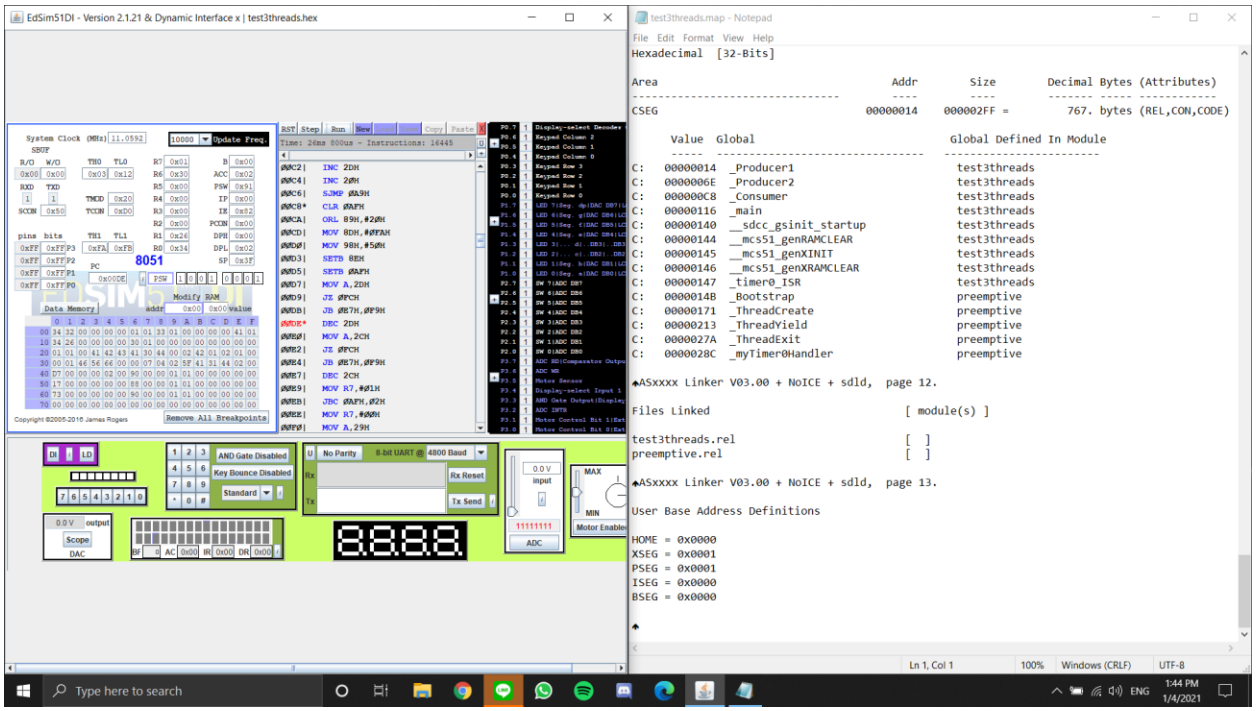
3. Consumer is running and show semaphore changes

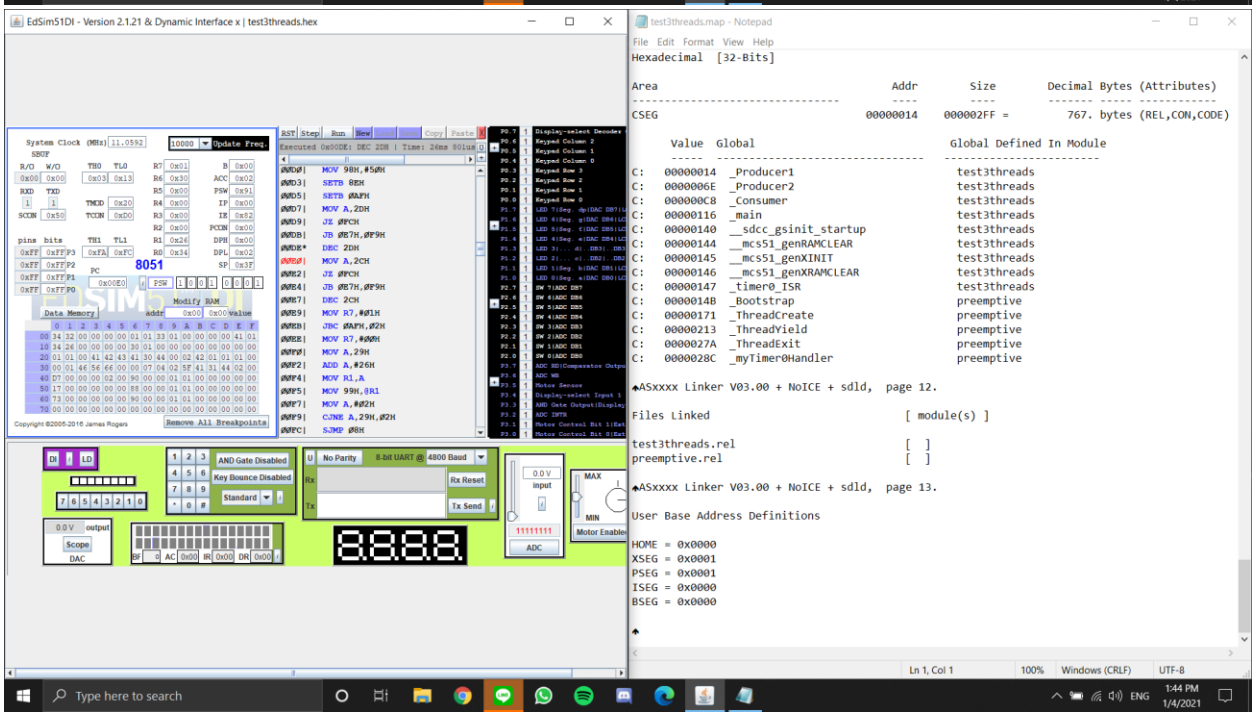
107062381 楊孝偉



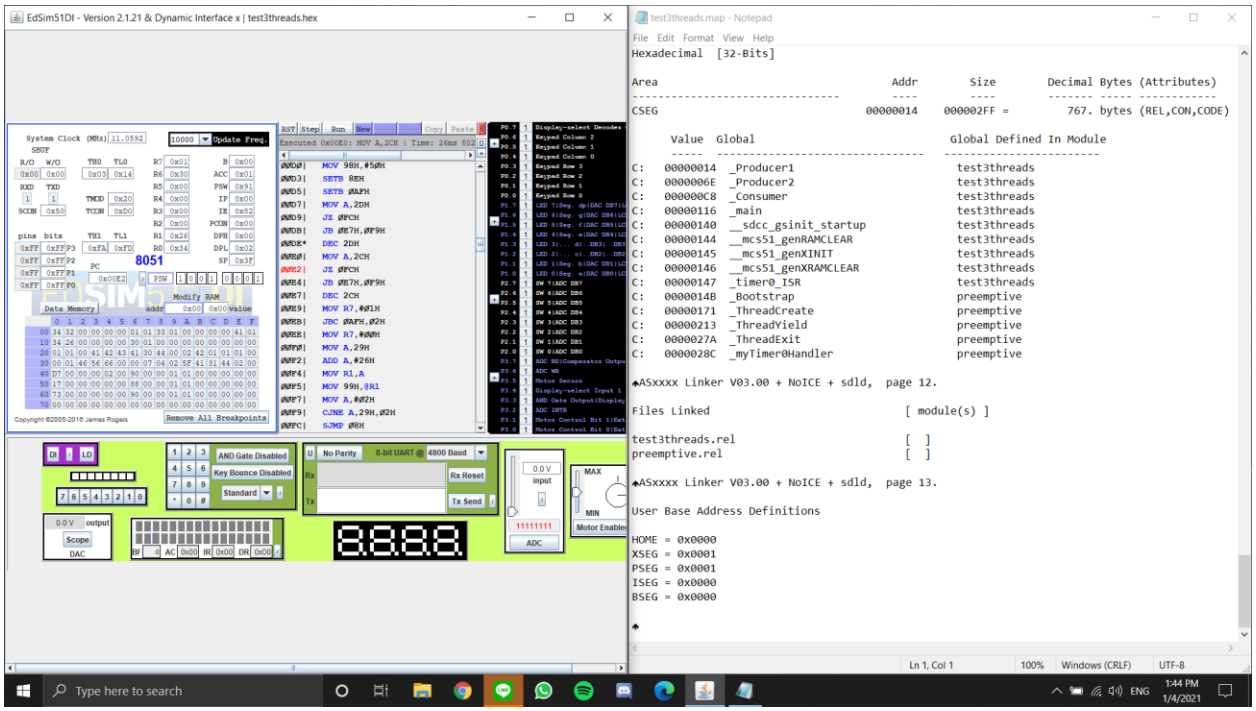


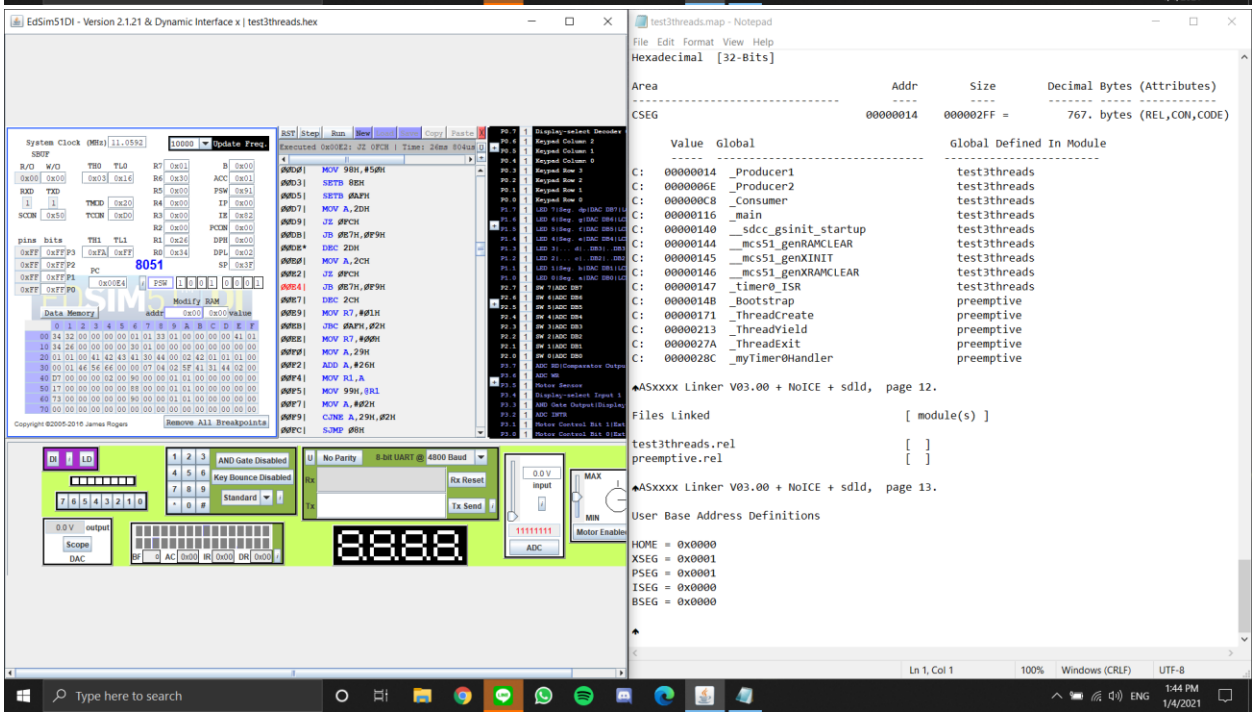
107062381 楊孝偉



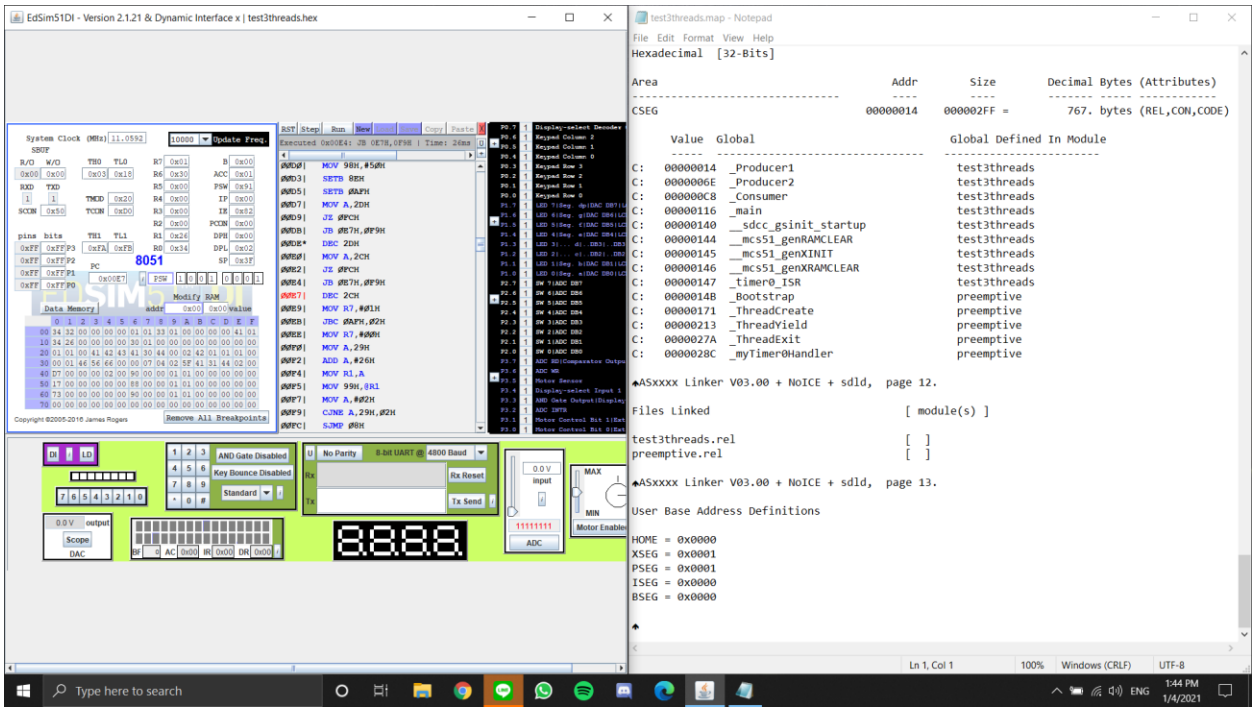


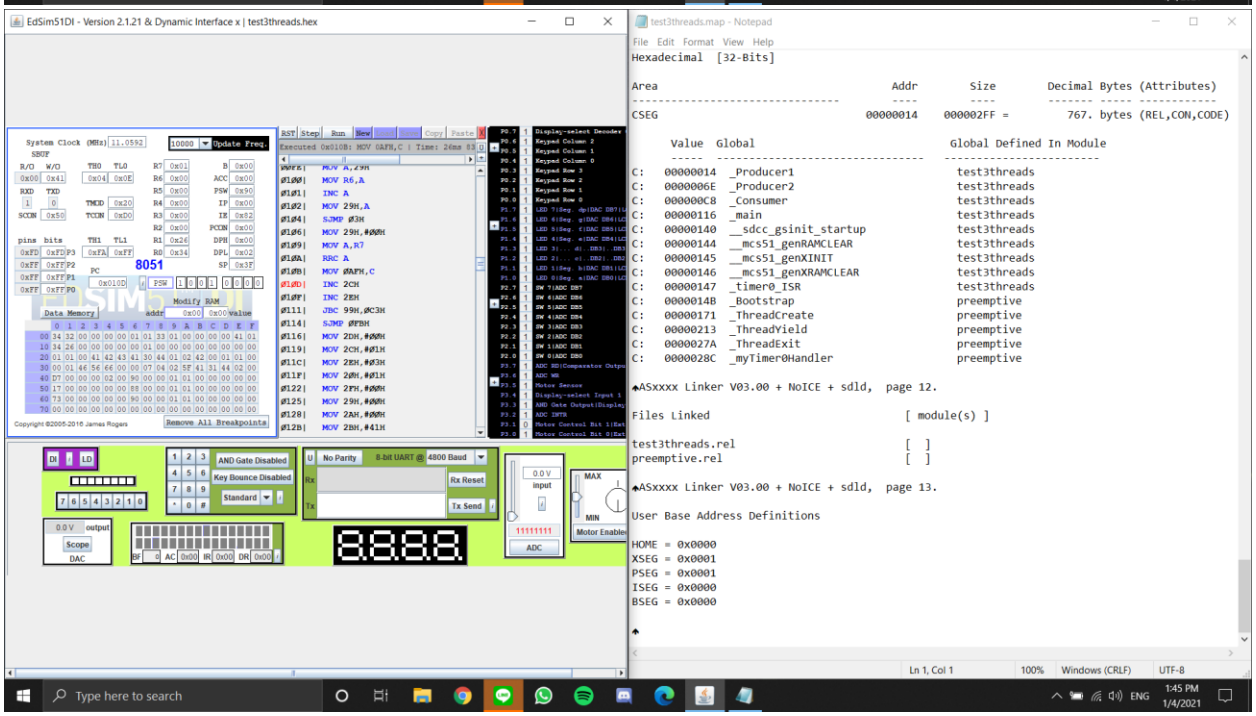
107062381 楊孝偉



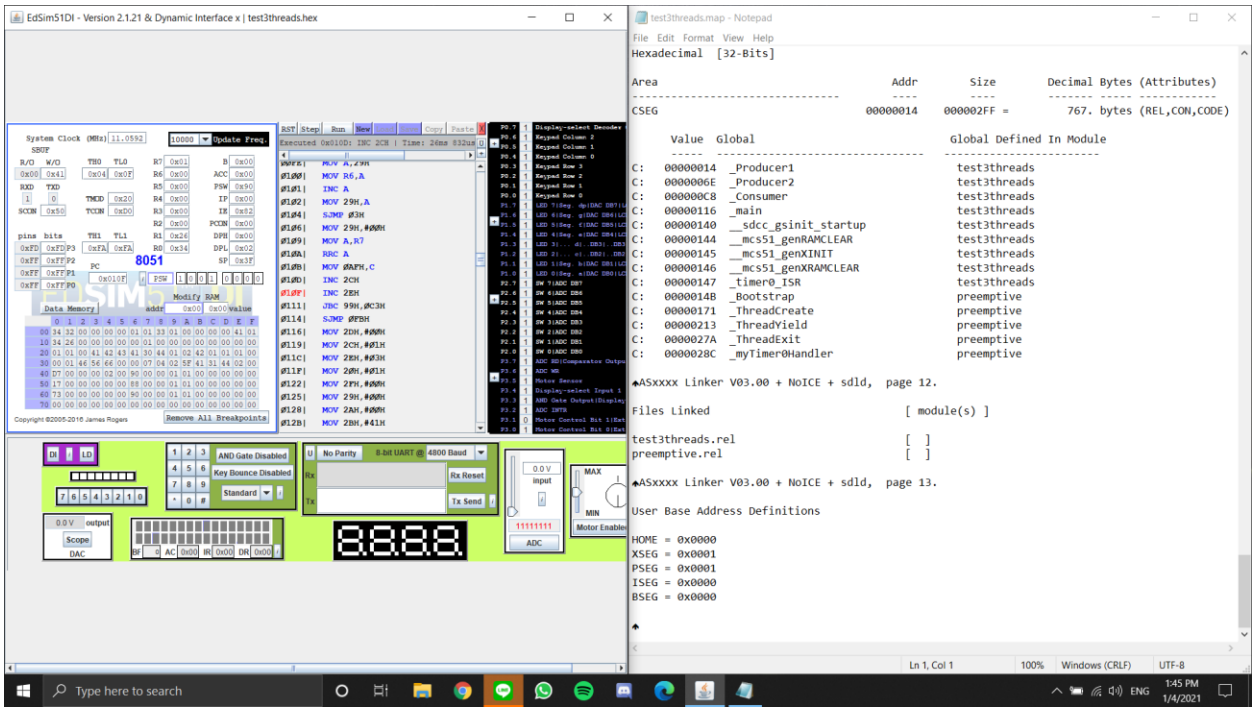


107062381 楊孝偉

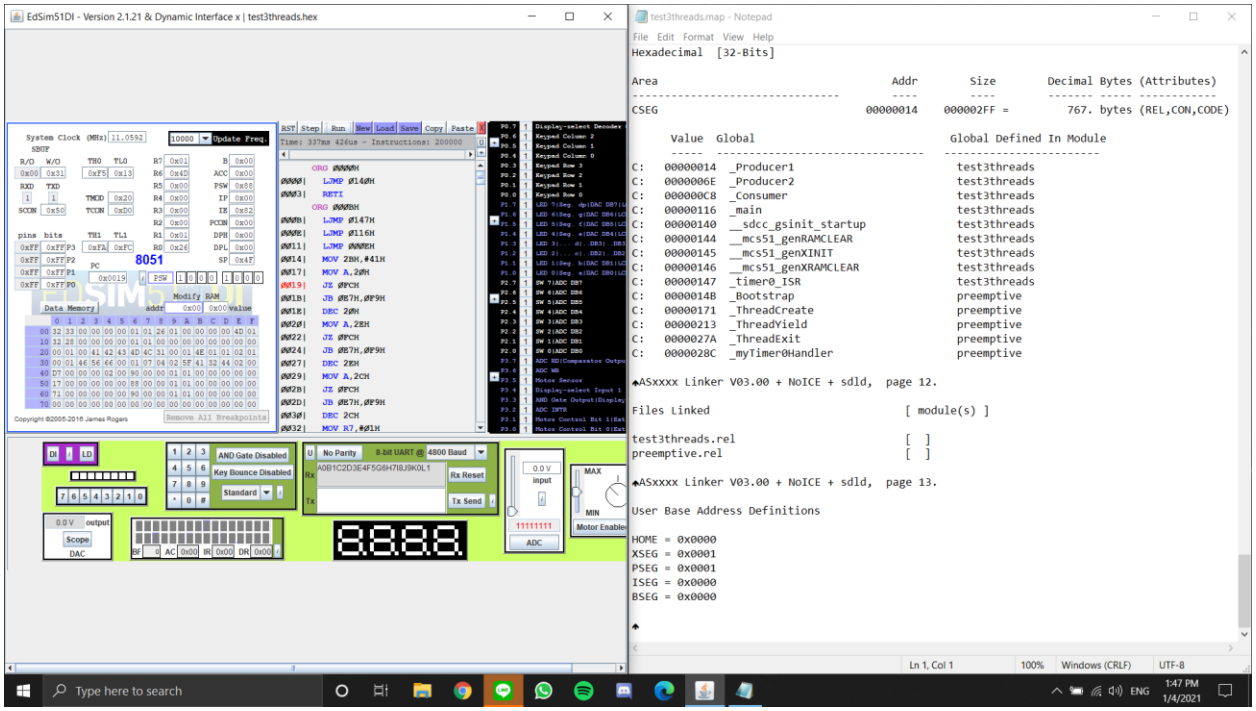




107062381 楊孝偉



4. Show and explain UART output to show the fair version.



As you can see, the producer1 and producer2 gets equal chance to output data as the data shown is A0B1C2… This can be done due to using two semaphores of producer1_ready and producer2_ready. After producer1 has produced, it will signal producer2_ready so that the next turn, it's the producer2's turn, not producer1 again, and we will have to wait for producer2 to finish and signal producer1_ready In order for producer 1 to be able to output again.