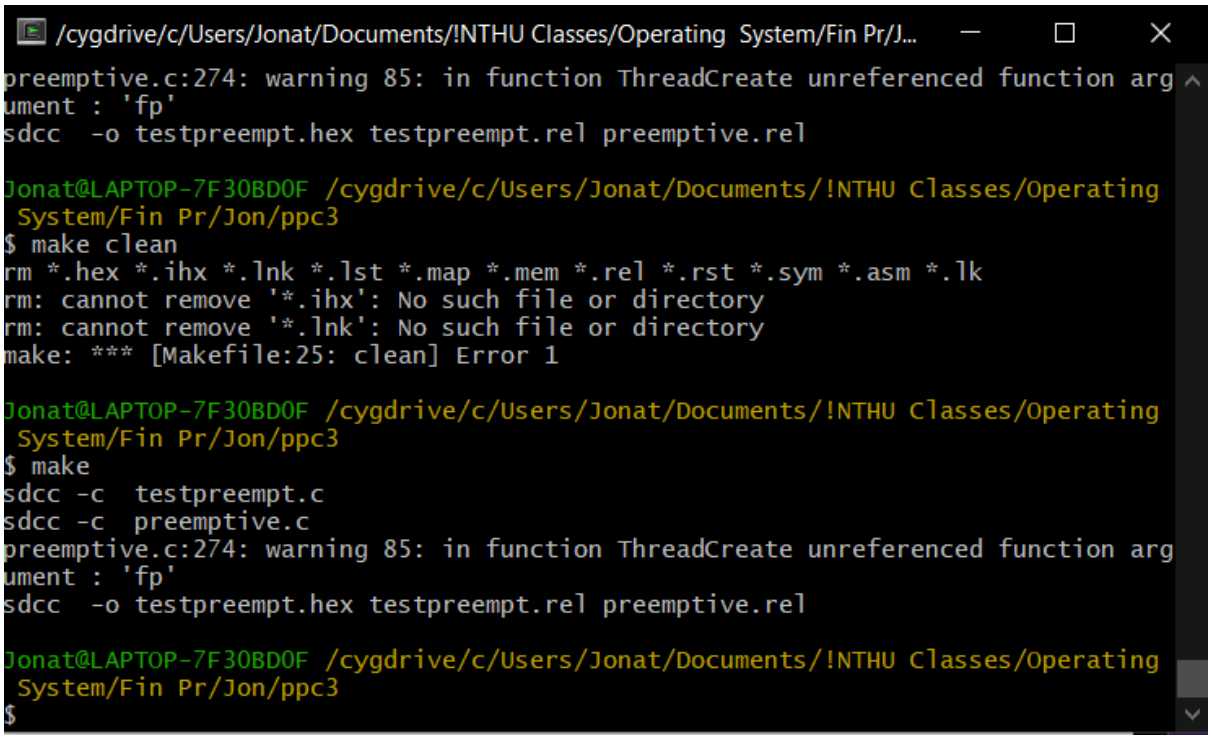
Report Checkpoint 3

Jonathan Sutedjo 鄭安良 111006207



```
 */

#ifndef __PREEMPTIVE_H__
#define __PREEMPTIVE_H__

#define MAXTHREADS 4 /* not including the scheduler */
/* the scheduler does not take up a thread of its own */


#define CNAME(s) _ ## s
#define CNAMELABEL(label) label ## $

#define SemaphoreWait(s){ \
    SemaphoreWaitBody(s, __COUNTER__) \
}


#define SemaphoreSignal(s){ \
    __asm \
        INC CNAME(s) \
    __endasm; \
}

#define SemaphoreWaitBody(S, label) \
    { __asm \
CNAMELABEL(label):  MOV ACC, CNAME(S) \
                    JZ CNAMELABEL(label) \
                    JB ACC.7, CNAMELABEL(label) \
                    dec  CNAME(S) \
    __endasm; }



typedef char ThreadID;
typedef void (*FunctionPtr)(void);

ThreadID ThreadCreate(FunctionPtr);
void ThreadYield(void);
void ThreadExit(void);

#endif // __COOPERATIVE_H__
```

In the preemptive.h file, I make changes to it for the semaphore by adding the SemaphoreWait which is to call the SemaphoreWaitBody and pass the parameter with the unique integer __COUNTER__. In the SemaphoreWaitBody, it first move the value of S, then check if the ACC is equal to zero or not, the PC will jump back to MOV ACC, CNAME(S) if it is equal to zero. This will make a loop. In the condition that it is not equal then it will check the 7^{th} bit of ACC if it is 1 or not, where this 7^{th} bis it to know if it is a negative number or not. If it is 1 then it will jump back to the top to be a loop, but if it is not then the PC will continue and the S will be decrease by 1. Here there is also the SemaphoreSignal which is to increase the S by 1.

```
__data __at (0x35) char TheChar;
__data __at (0x25) char mutex;
__data __at (0x26) char full;
__data __at (0x27) char empty;
__data __at (0x3A) char head;
__data __at (0x3B) char tail;
__data __at (0x3D) char SharedBuffer[3];
```

```
void SemaphoreCreate(char *s, char n){
    EA = 0;
    *s = n;
    EA = 1;
}
```

In the testpreempt.c file, I add mutex, full, empty, tail, and head, where the tail and head is for the shared buffer. Then The var to keep the A-Z is the “TheChar”.

The SemaphoreCreate is to put a value in char to the semaphore. Here I use a pointer so that the one that I update is the value from the source address of the semaphore.

```
void Producer(void)
{
    /*
     * [TODO]
     * initialize producer data structure, and then enter
     * an infinite loop (does not return)
     */
    EA = 0;
    TheChar = 'A'-1;
    EA = 1;

    while (1)
    {
        /* [TODO]
         * wait for the buffer to be available,
         * and then write the new data into the buffer */
        SemaphoreWait(empty);
        SemaphoreWait(mutex);
        EA = 0;

        if(TheChar == 'Z'){
                TheChar = 'A';
        }
        else{
                TheChar += 1;
        }
        SharedBuffer[tail] = TheChar;
        tail += 1;
        if(tail == 3){
            tail = 0;
        }

        EA = 1;
        SemaphoreSignal(mutex);
        SemaphoreSignal(full);
    }
}
```

For the producer part, now I use the bounded buffer such that I need to wait for the semaphore to be ready before doing anything to the shared buffer. The producer will need to wait for the “empty” to be not 0 and the “mutex” to be 1 before it can produce more character. After the character is produced of course it is assigned to the SharedBuffer[tail]. After it is done, it will return the value of mutex and signal full. The full signal is so that the consumer know there is new value inside the shared buffer. The mutex is so that other thread can use it.

```c
void Consumer(void)
{
    /*
     * [TODO]
     * initialize Tx for polling
     */
    TMOD |= 0x20;
    TH1 = (char)-6;
    SCON = 0x50;
    TR1 = 1;
    TI = 1;

    while (1)
    {
        /*
         * [TODO]
         * wait for new data from producer
         * write data to serial port Tx,
         * poll for Tx to finish writing (TI),
         * then clear the flag
         */

        SemaphoreWait(full);
        SemaphoreWait(mutex);
        EA = 0;

        while (!TI){
        }
        SBUF = SharedBuffer[head];
        TI = 0;
        head += 1;
        if(head == 3){
            head = 0;
        }

        EA = 1;
        SemaphoreSignal(mutex);
        SemaphoreSignal(empty);
    }
}
```

In the consumer part, I use the bounded buffer such that the consumer will have to wait for the producer to signal full to get the new value in the shared buffer. After the signal is received, it will wait for the mutex. Consumer will signal mutex to let other thread run after it is done. It will also signal empty to let the producer know the sharedbuffer have spot available for new characters now.

```c
void main(void)
{
    /*
     * [TODO]
     * initialize globals
     */

    SemaphoreCreate(&mutex, 1);
    SemaphoreCreate(&full, 0);
    SemaphoreCreate(&empty, 3);

    head = 0;
    tail = 0;

    SharedBuffer[0] = ' ';
    SharedBuffer[1] = ' ';
    SharedBuffer[2] = ' ';

    /*
     * [TODO]
     * set up Producer and Consumer.
     * Because both are infinite loops, there is no loop
     * in this function and no return.
     */
    ThreadCreate(Producer);
    Consumer();
}
```

Here, I need to first initialized the semaphores by using the SemaphoreCreate for full, empty, and mutex. The value of full will be 0 because it means there are no new characters there. The value for mutex is so that only 1 thread run at a time. The empty will be 3 because there are 3 slots available for new characters and that is what we want (3-deep).

```
Area                                    Addr      Size         Decimal Bytes (Attributes)
--------------------------------  ----  ----  -------  -----  ------------
CSEG                                    00000014  000004AF =      1199. bytes (REL,CON,CODE)

      Value  Global                                 Global Defined In Module
      -----  -------------------------------------  -------------------------
C:    00000014  _SemaphoreCreate                      testpreempt
C:    0000002A  _Producer                             testpreempt
C:    00000073  _Consumer                             testpreempt
C:    000000B8  _main                                 testpreempt
C:    000000F4  __sdcc_gsinit_startup                 testpreempt
C:    000000F8  __mcs51_genRAMCLEAR                   testpreempt
C:    000000F9  __mcs51_genXINIT                      testpreempt
C:    000000FA  __mcs51_genXRAMCLEAR                  testpreempt
C:    000000FB  _timer0_ISR                           testpreempt
C:    000000FF  _Bootstrap                            preemptive
C:    00000153  _ThreadCreate                         preemptive
C:    00000207  _ThreadYield                          preemptive
C:    000002E7  _ThreadExit                           preemptive
C:    000003CE  _myTimer0Handler                      preemptive
C:    000004A8  __gptrput                             _gptrput
ASxxxx Linker V03.00/V05.40 + sdld,  page 13.
```

This is some of the map file content.

Image Context 1

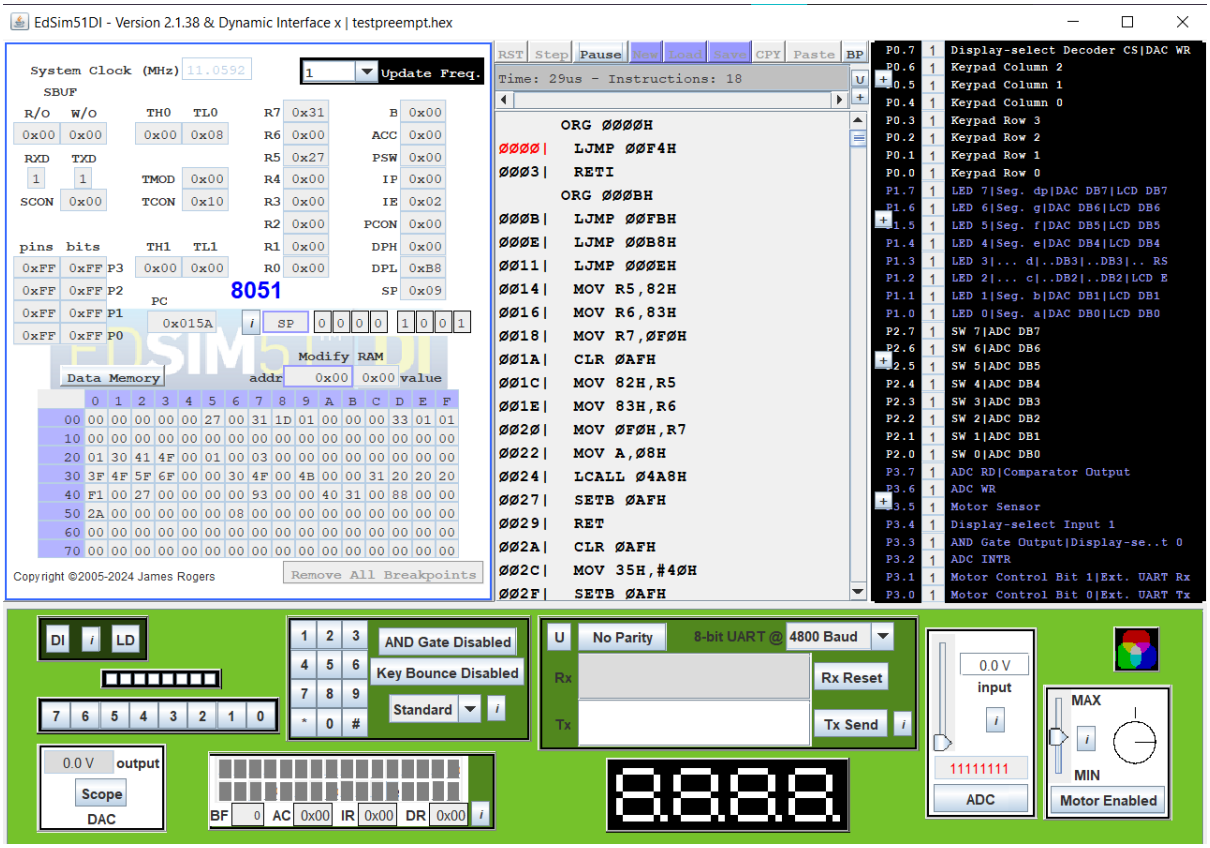


Image Context 2

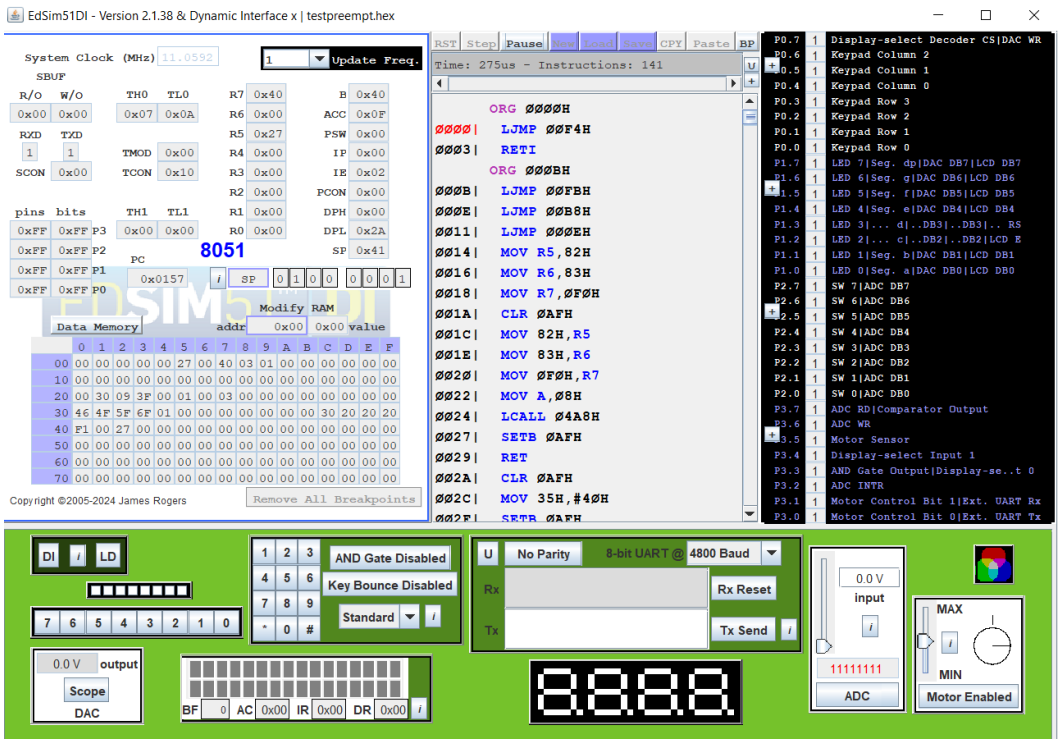


Image Context 3

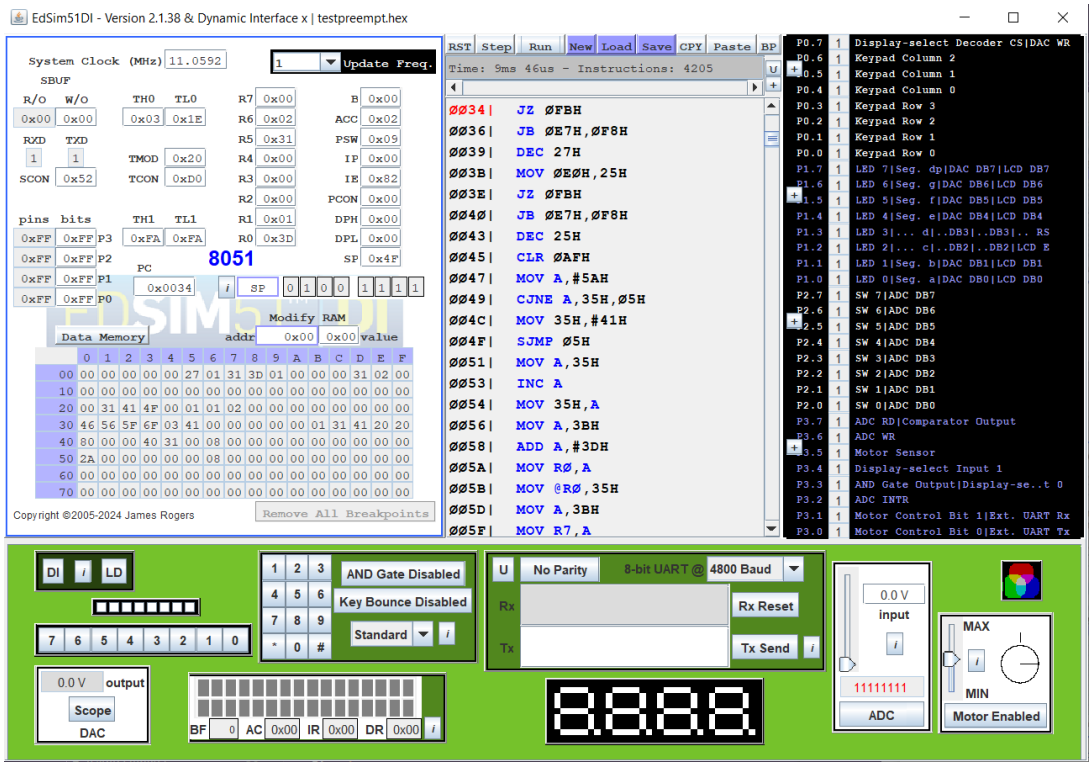


Image Context 4

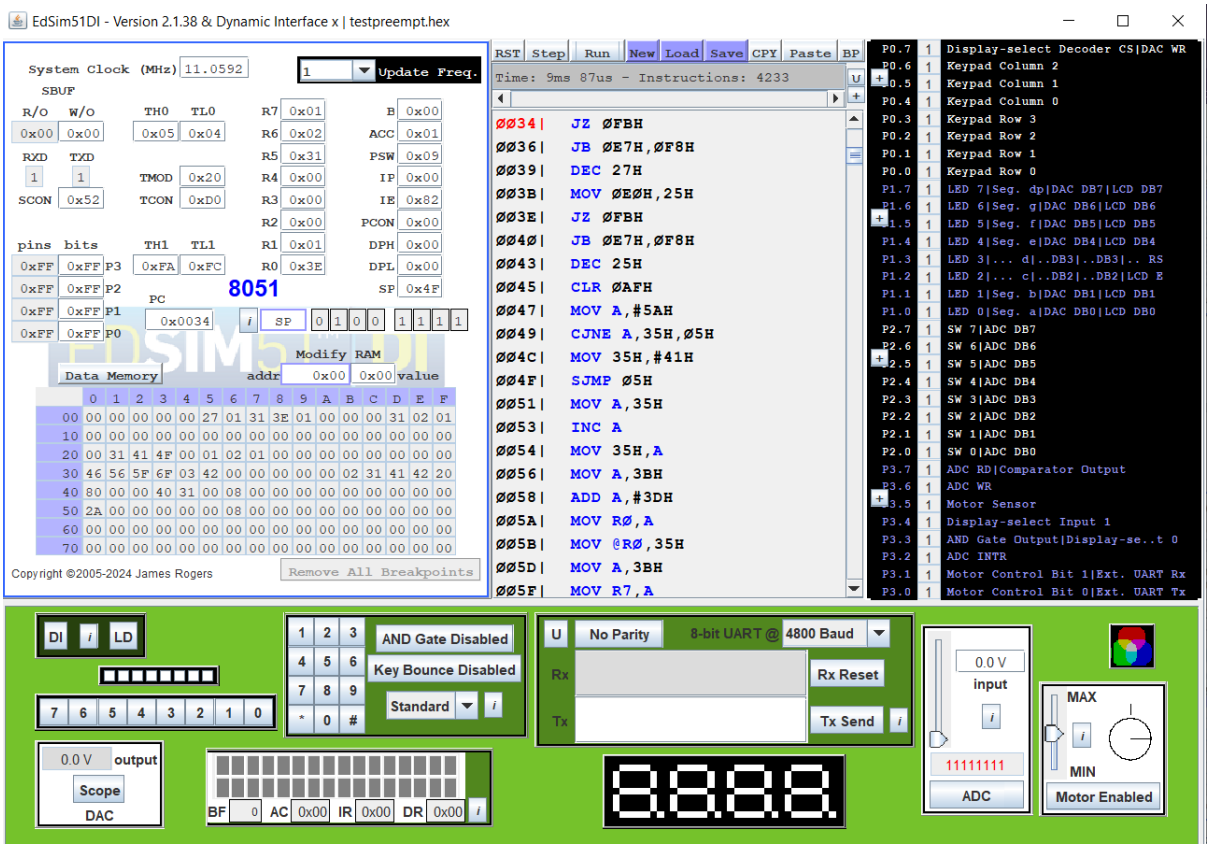
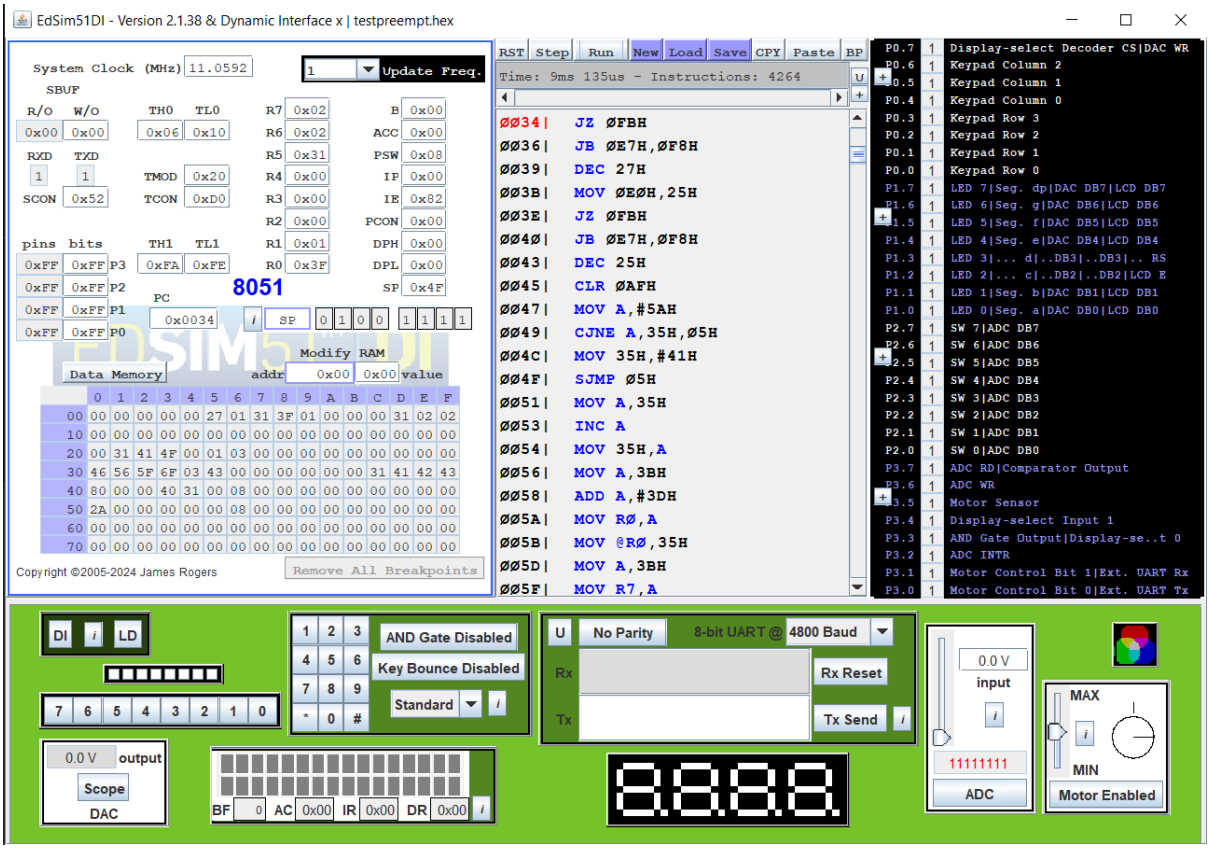


Image Context 5



For context, the mutex is 0x25, full us 0x26, and empty is 0x27.

From Image context 1-5, we can see that the value of full will be increase by 1 every time a new character is produced by the producer. While the full is increased by 1, the empty will decrease by 1.

Image Context 6

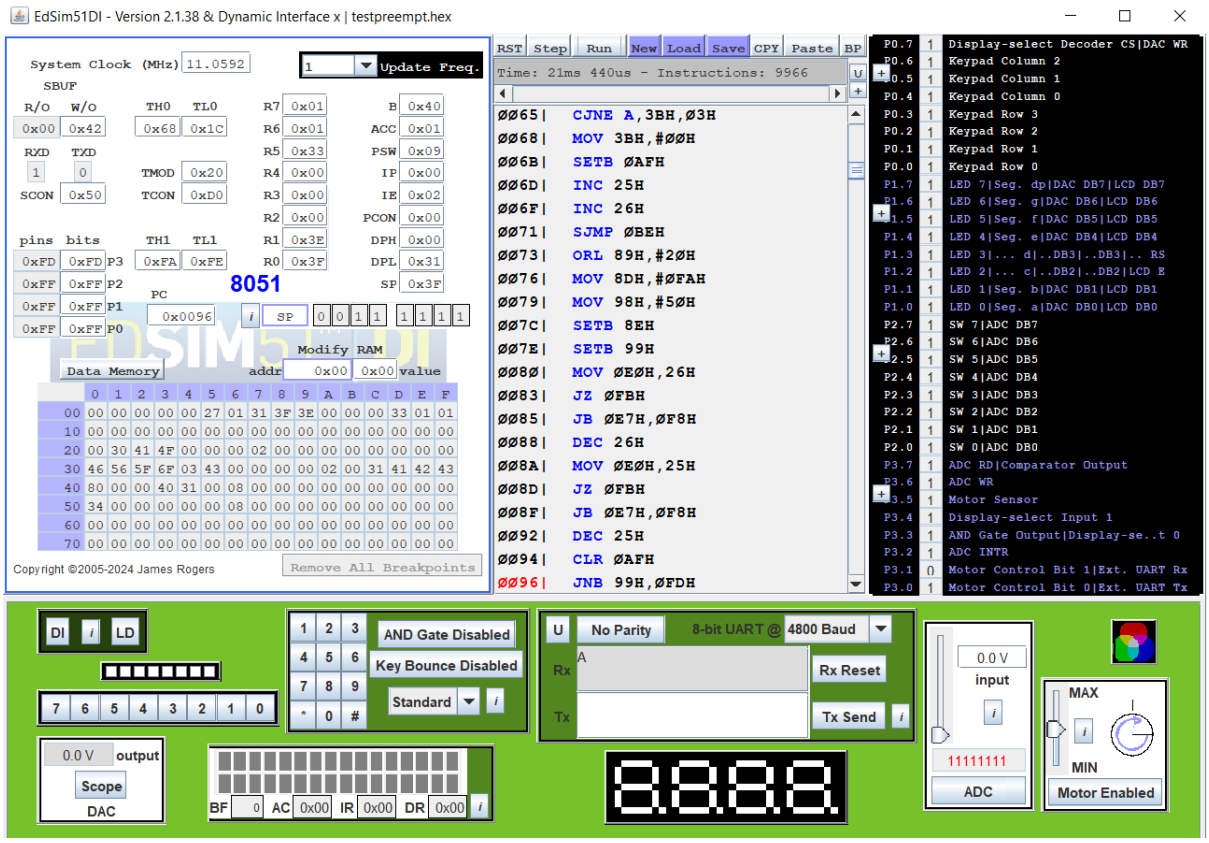
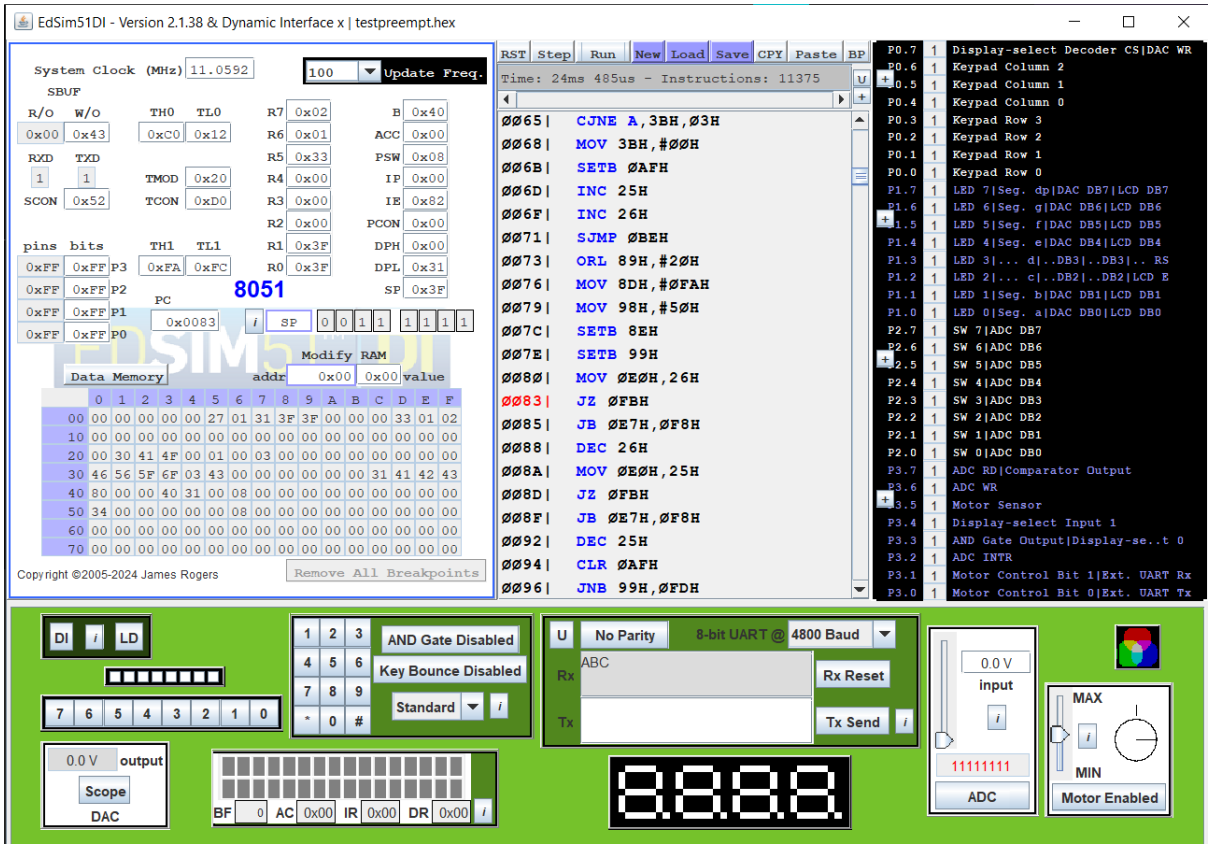


Image context 7



From image context 5-7 we can see that the full will decrease by 1 every time when it print out 1 character. The empty will increase by 1 also when the full is decreasing.