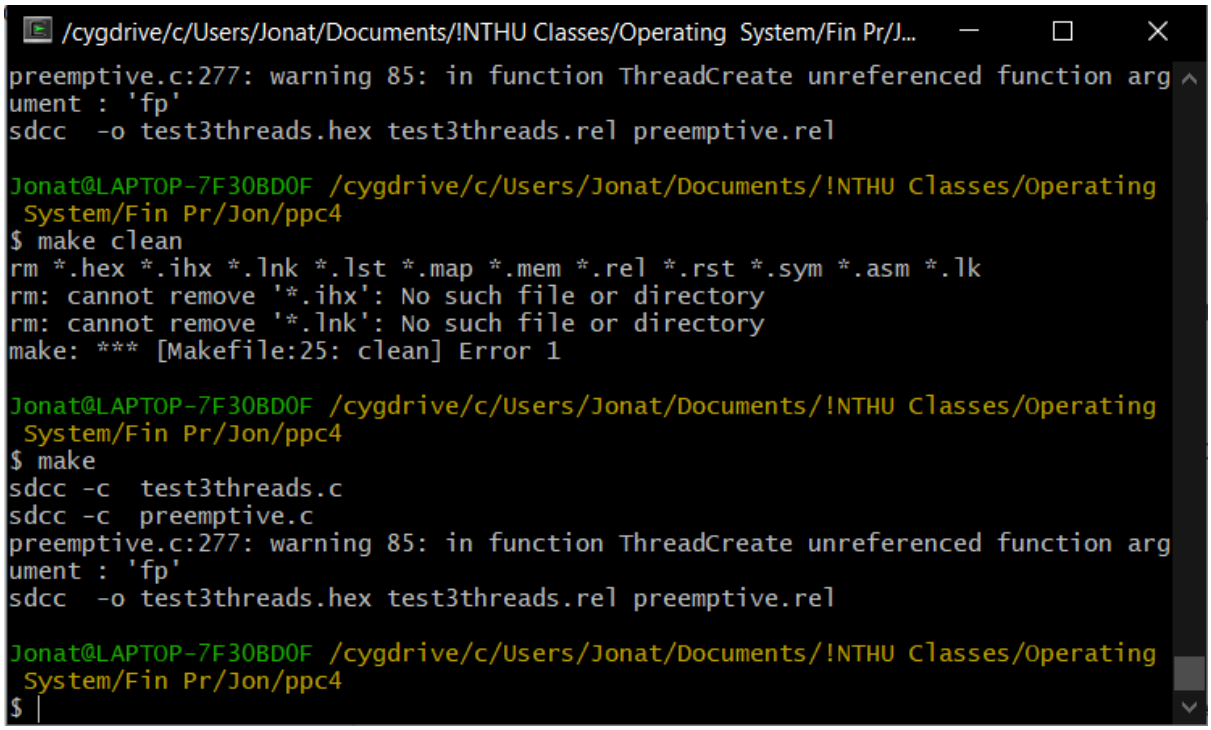
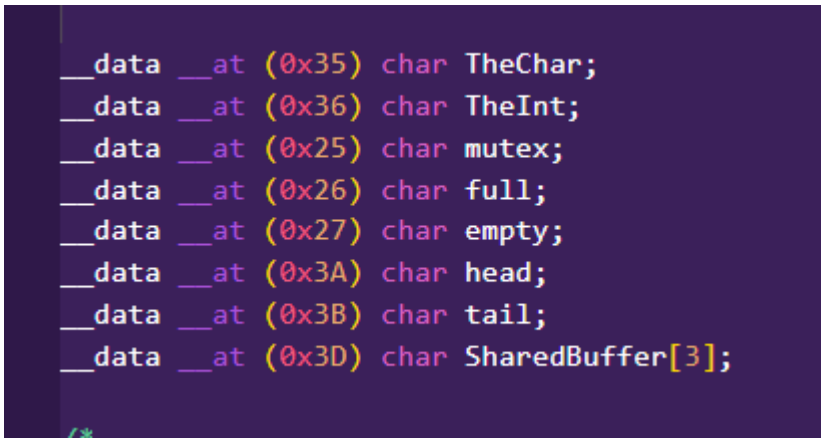
Report Checkpoint 4

Jonathan Sutedjo 鄭安良 111006207





For the test3threads.c file, I have an additional variable to store the temporary value of the numbers from producer2, the name is “TheInt”.

```c
void Producer1(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);
    }
}
```

This is my producer1, where it is the same with checkpoint 3, producing out A-Z alphabet to the shared buffer.

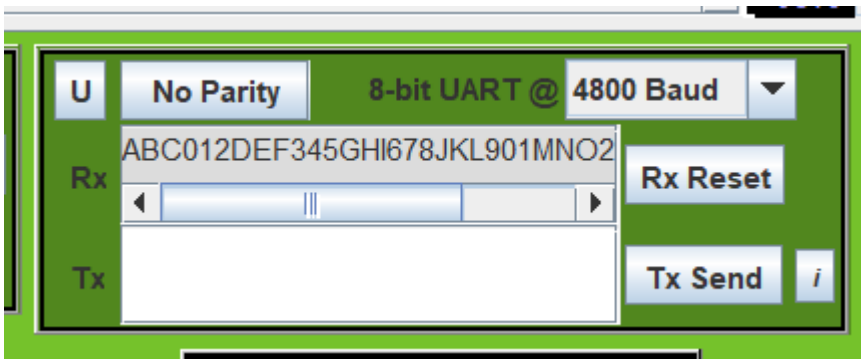
```
void Producer2(void){
    EA = 0;
    TheInt = '0'-1;
    EA = 1;

    while (1)
    {
        SemaphoreWait(empty);
        SemaphoreWait(mutex);
        EA = 0;

        if(TheInt == '9'){
            TheInt = '0';
        }
        else{
            TheInt += 1;
        }
        SharedBuffer[tail] = TheInt;
        tail += 1;
        if(tail == 3){
            tail = 0;
        }

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

This is my producer2, where it is similar to producer1 but instead of alphabet it produces number from 0 to 9 and stored it to the shared buffer.

For my consumer, it stays the same as from checkpoint 3.

```
 */
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(Producer1);
    ThreadCreate(Producer2);
    Consumer();
}
```

In my main, I write the ThreadCreate in that sequence, and that would result in the printing of alphabets for the first 3, then the next 3 are numbers from 0 to 2, and then it change to alphabets again and so on. This change happen for the fairness. If I change the sequence such that Producer2 is first in the line of code, then it will print out the number from 0 to 2 first then comes the 3 alphabets a to c, and so on.



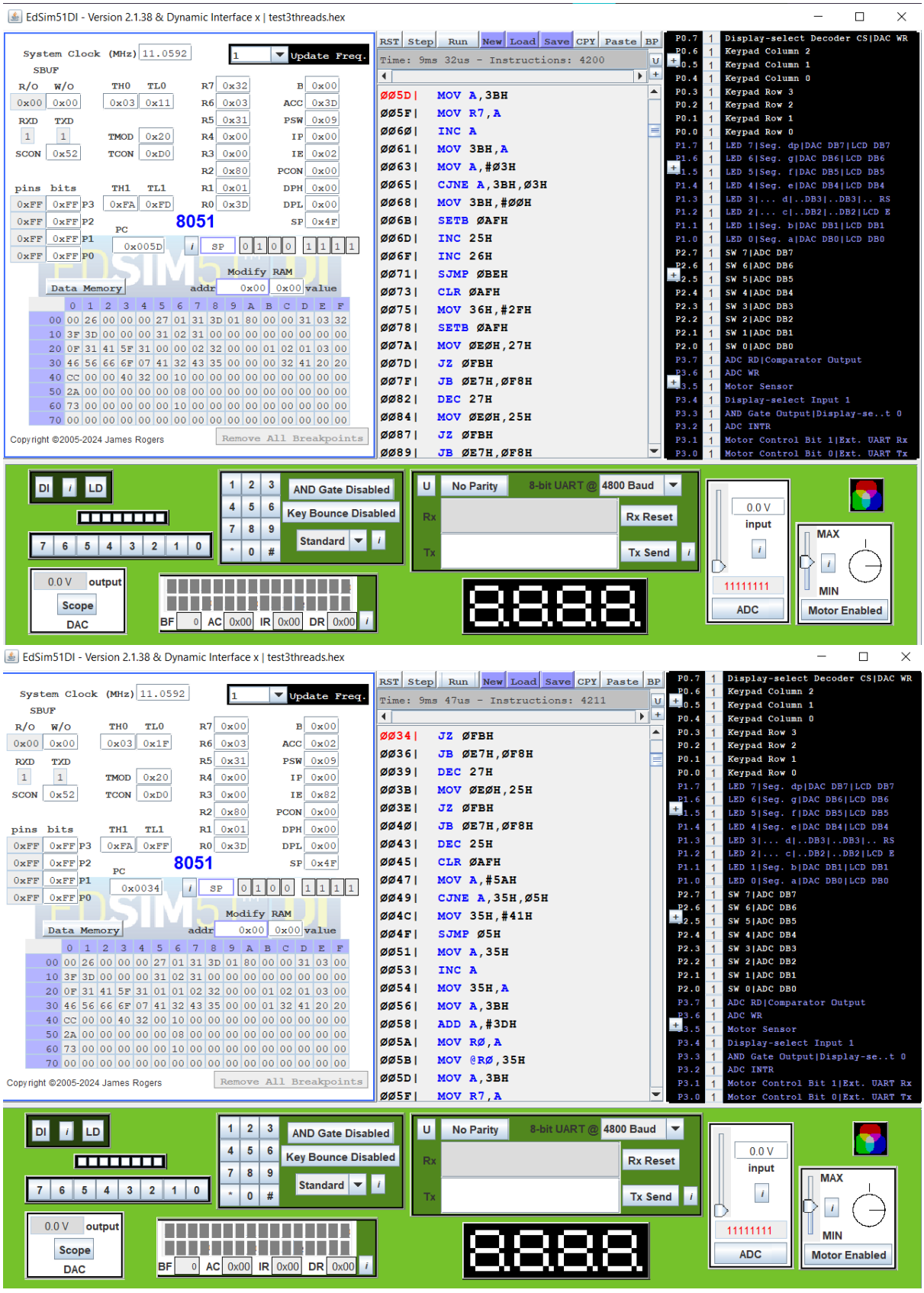
From the result of the Rx here, producer1 will fill in the first 3 of the so called shared buffer with their products and then the consumer will print it out or display it. After that, the other producer fill in the shared buffer and then the consumer will then print out the result of producer2. With this, it can be seen that it is being printed alternatively between producer 1 and 2 by switching every 3 characters.

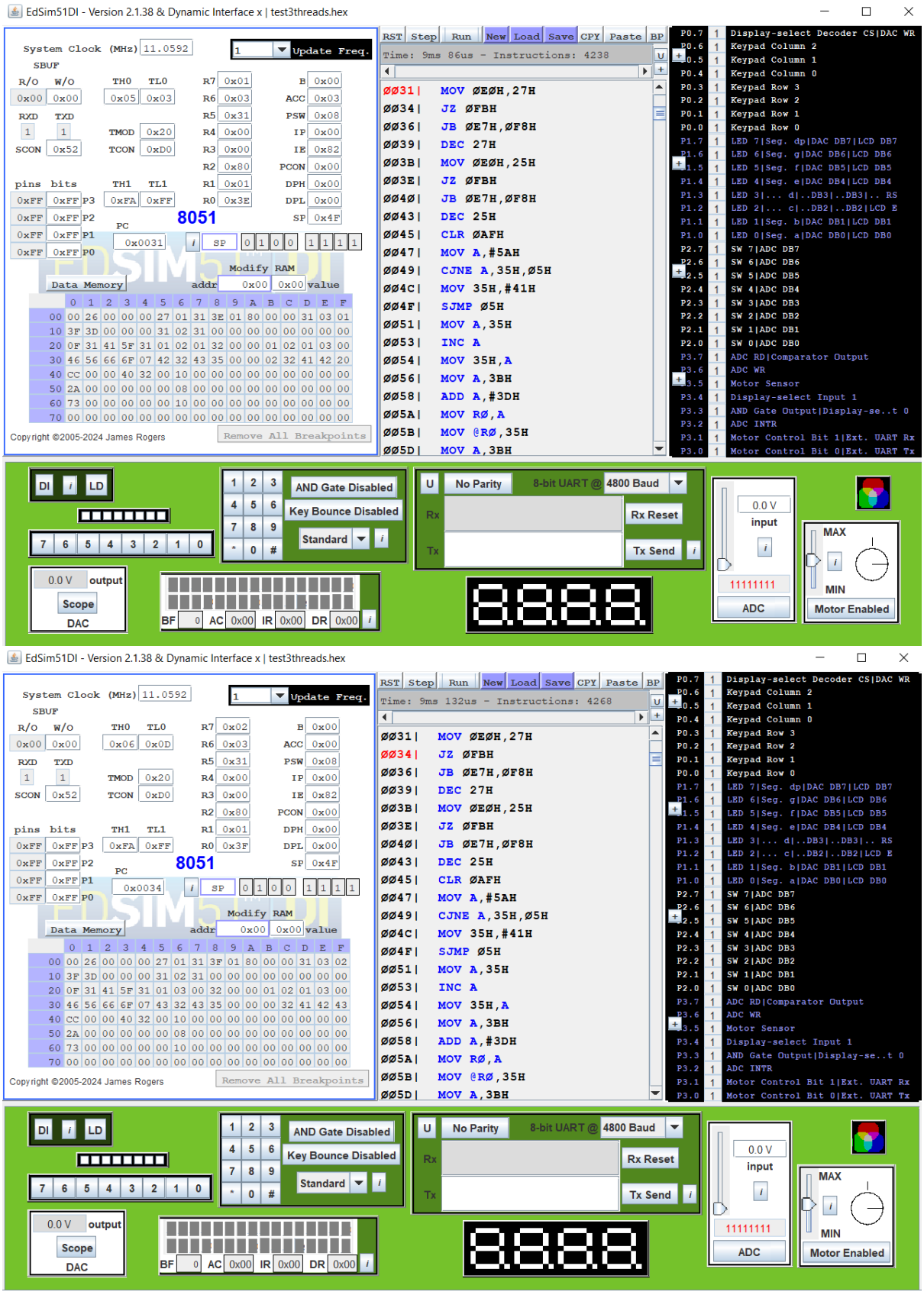
```
void myTimer0Handler(void){
    EA = 0;
    SAVESTATE;
    do{

        if (curThreadID != '0') {
            // From producer1 or producer2 to consumer
            curThreadID = '0';
        }
        else {
            // From consumer to producers
            // Need to decide to avoid starvation
            curThreadID = ProdThreadID;
            ProdThreadID += 1;
            if (ProdThreadID == '3') {
                ProdThreadID = '1';
            }
        }

        switch (curThreadID) {
            case '0':
                if ((ValidBitMap & 0b0001) == 0b0001) break;
                continue;
            case '1':
                if ((ValidBitMap & 0b0010) == 0b0010) break;
                continue;
            case '2':
                if ((ValidBitMap & 0b0100) == 0b0100) break;
                continue;
            case '3':
                if ((ValidBitMap & 0b1000) == 0b1000) break;
                continue;
        }
        break;
      } while (1);
      RESTORESTATE;

      EA = 1;
      __asm
        RETI
      __endasm;
}
```

```
__data __at (0x30) char Pointer[MAXTHREADS];
__data __at (0x3C) ThreadID newThreadID;
__data __at (0x28) ThreadID ProdThreadID;
__data __at (0x34) int ValidBitMap;

__data __at (0x21) ThreadID curThreadID;
__data __at (0x22) char tempSP;
__data __at (0x23) char newSP;
__data __at (0x24) char ProductionFlag;
```

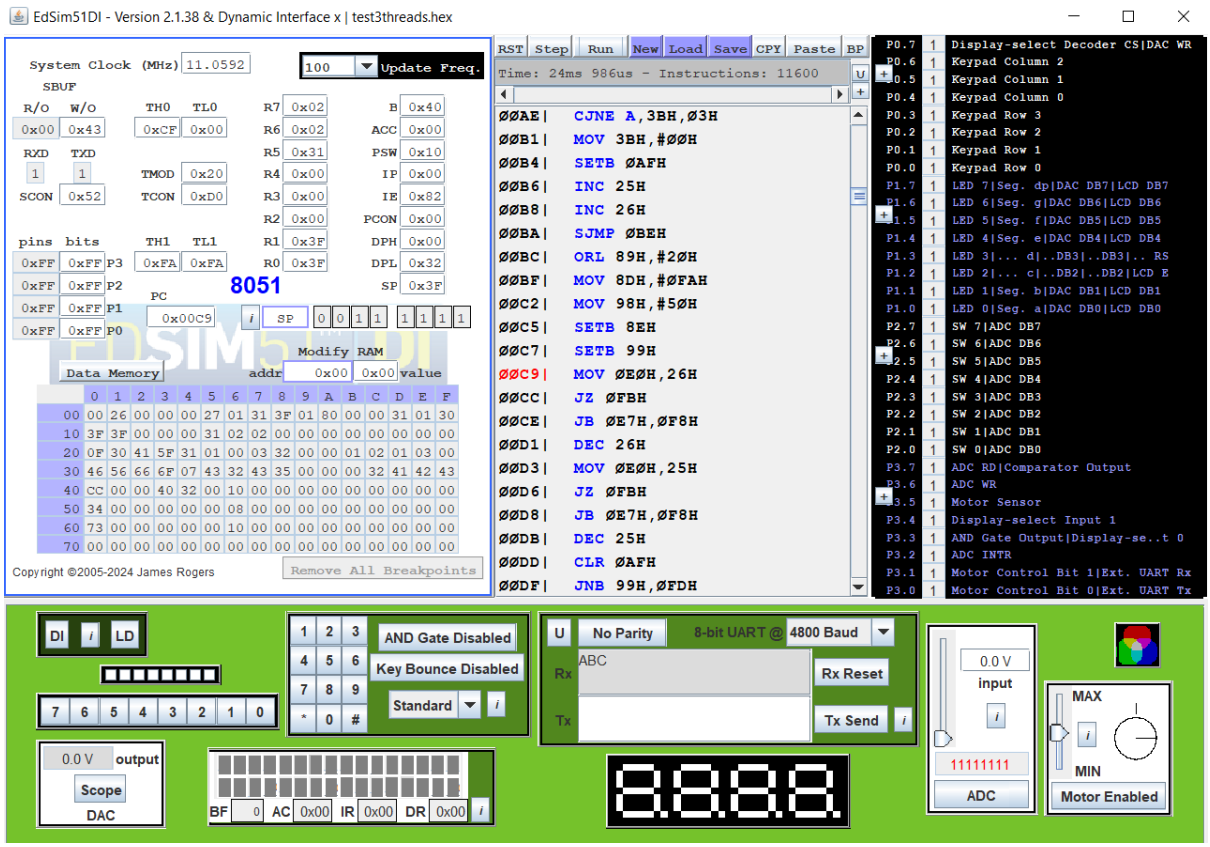
This is my code for the MyTimer0Handler, where I implemented another thing so there can be fairness there. In the preemptive.c file, I added another variable which is the "ProdCountID". In the myTimer0Handler, it will first check wether the curThreadID or the current thread ID is 0 or not. In this case, if it is 0, it means it is for consumer, and the

curThreadId will be assigned to the ProdThreadID that will then be loop between 1 to 3 every time, so that the next producer will not be the last one. If the current thread ID is not 0, then the curThreadID will be assign 0 so that the consumer will free the shared buffer.

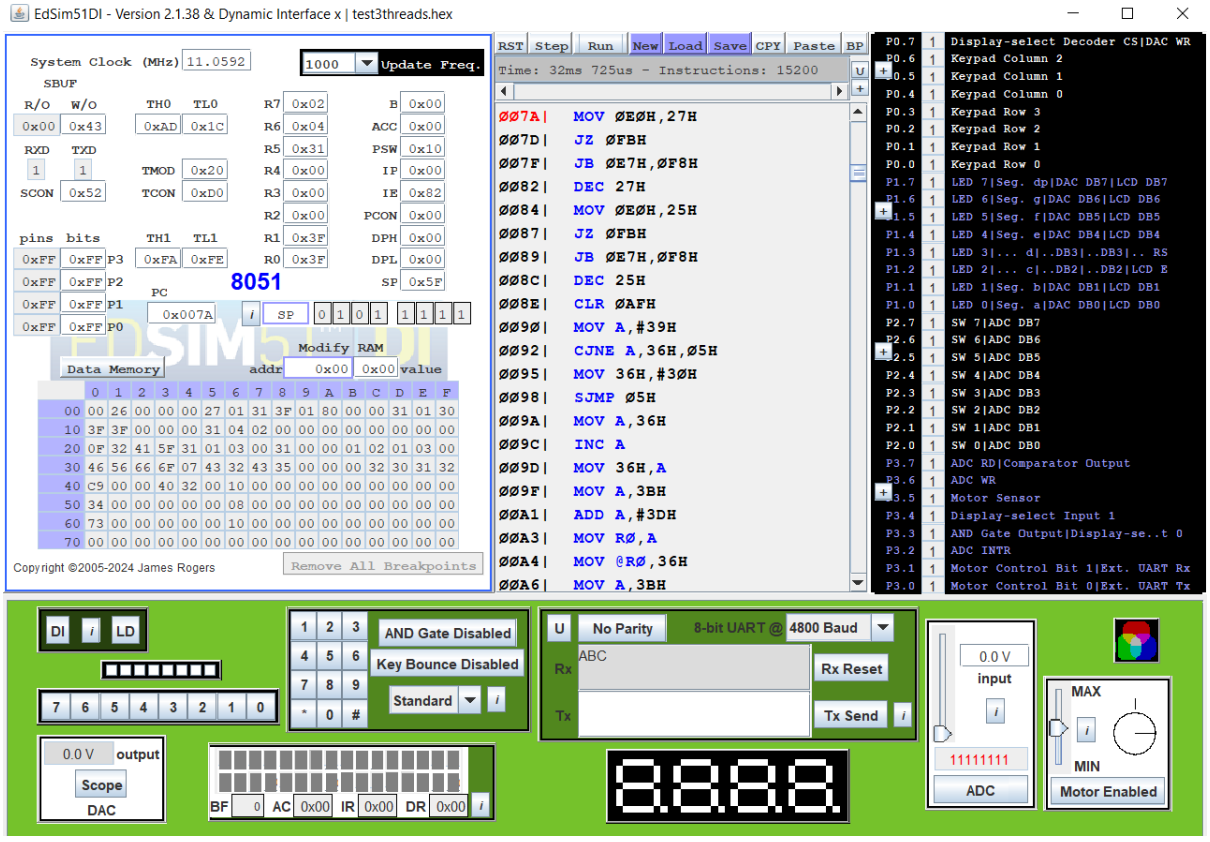

EdSim51DI - Version 2.1.38 & Dynamic Interface x | test3threads.hex
System Clock (MHz) 11.0592
Update Freq.
8051
Data Memory
Modify RAM
Copyright ©2005-2024 James Rogers
Remove All Breakpoints
RST Step Run New Load Save CPY Paste BP
Time: 9ms 32us - Instructions: 4200
ØØ5D| MOV A,3BH
ØØ5F| MOV R7,A
ØØ6Ø| INC A
ØØ61| MOV 3BH,A
ØØ63| MOV A,#Ø3H
ØØ65| CJNE A,3BH,Ø3H
ØØ68| MOV 3BH,#ØØH
ØØ6B| SETB ØAFH
ØØ6D| INC 25H
ØØ6F| INC 26H
ØØ71| SJMP ØBEH
ØØ73| CLR ØAFH
ØØ75| MOV 36H,#2FH
ØØ78| SETB ØAFH
ØØ7A| MOV ØEØH,27H
ØØ7D| JZ ØFBH
ØØ7F| JB ØE7H,ØF8H
ØØ82| DEC 27H
ØØ84| MOV ØEØH,25H
ØØ87| JZ ØFBH
ØØ89| JB ØE7H,ØF8H
P0.7 Display-select Decoder CS|DAC WR
P0.6 Keypad Column 2
P0.5 Keypad Column 1
P0.4 Keypad Column 0
P0.3 Keypad Row 3
P0.2 Keypad Row 2
P0.1 Keypad Row 1
P0.0 Keypad Row 0
P1.7 LED 7|Seg. dp|DAC DB7|LCD DB7
P1.6 LED 6|Seg. g|DAC DB6|LCD DB6
P1.5 LED 5|Seg. f|DAC DB5|LCD DB5
P1.4 LED 4|Seg. e|DAC DB4|LCD DB4
P1.3 LED 3|... d|..DB3|..DB3|.. RS
P1.2 LED 2|... c|..DB2|..DB2|LCD E
P1.1 LED 1|Seg. b|DAC DB1|LCD DB1
P1.0 LED 0|Seg. a|DAC DB0|LCD DB0
P2.7 SW 7|ADC DB7
P2.6 SW 6|ADC DB6
P2.5 SW 5|ADC DB5
P2.4 SW 4|ADC DB4
P2.3 SW 3|ADC DB3
P2.2 SW 2|ADC DB2
P2.1 SW 1|ADC DB1
P2.0 SW 0|ADC DB0
P3.7 ADC RD|Comparator Output
P3.6 ADC WR
P3.5 Motor Sensor
P3.4 Display-select Input 1
P3.3 AND Gate Output|Display-se..t 0
P3.2 ADC INTR
P3.1 Motor Control Bit 1|Ext. UART Rx
P3.0 Motor Control Bit 0|Ext. UART Tx
AND Gate Disabled
Key Bounce Disabled
Standard
No Parity
8-bit UART @ 4800 Baud
Rx Reset
Tx Send
0.0 V input
11111111
ADC
MAX
MIN
Motor Enabled
0.0 V output
Scope
DAC
EdSim51DI - Version 2.1.38 & Dynamic Interface x | test3threads.hex
System Clock (MHz) 11.0592
Update Freq.
8051
Data Memory
Modify RAM
Copyright ©2005-2024 James Rogers
Remove All Breakpoints
RST Step Run New Load Save CPY Paste BP
Time: 9ms 47us - Instructions: 4211
ØØ34| JZ ØFBH
ØØ36| JB ØE7H,ØF8H
ØØ39| DEC 27H
ØØ3B| MOV ØEØH,25H
ØØ3E| JZ ØFBH
ØØ4Ø| JB ØE7H,ØF8H
ØØ43| DEC 25H
ØØ45| CLR ØAFH
ØØ47| MOV A,#5AH
ØØ49| CJNE A,35H,Ø5H
ØØ4C| MOV 35H,#41H
ØØ4F| SJMP Ø5H
ØØ51| MOV A,35H
ØØ53| INC A
ØØ54| MOV 35H,A
ØØ56| MOV A,3BH
ØØ58| ADD A,#3DH
ØØ5A| MOV RØ,A
ØØ5B| MOV @RØ,35H
ØØ5D| MOV A,3BH
ØØ5F| MOV R7,A
AND Gate Disabled
Key Bounce Disabled
Standard
No Parity
8-bit UART @ 4800 Baud
Rx Reset
Tx Send
0.0 V input
11111111
ADC
MAX
MIN
Motor Enabled
0.0 V output
Scope
DAC

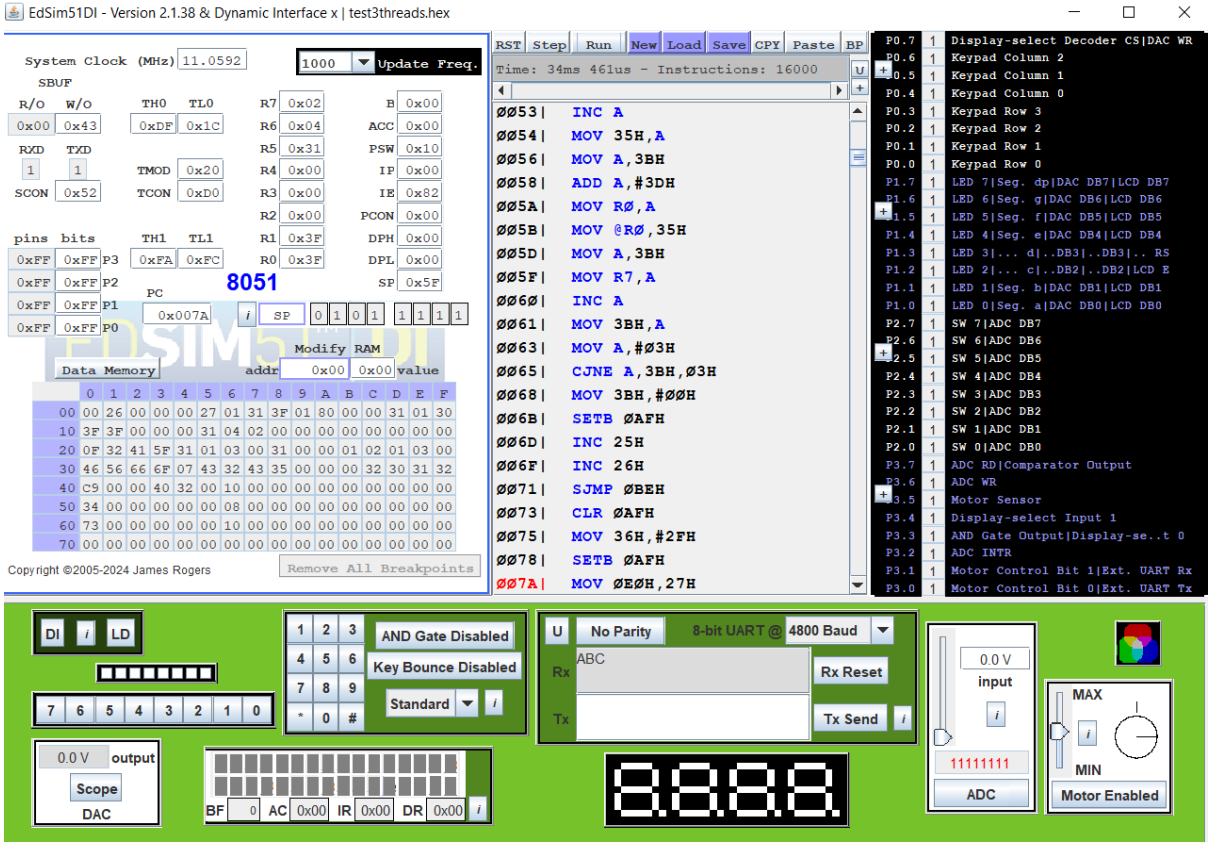


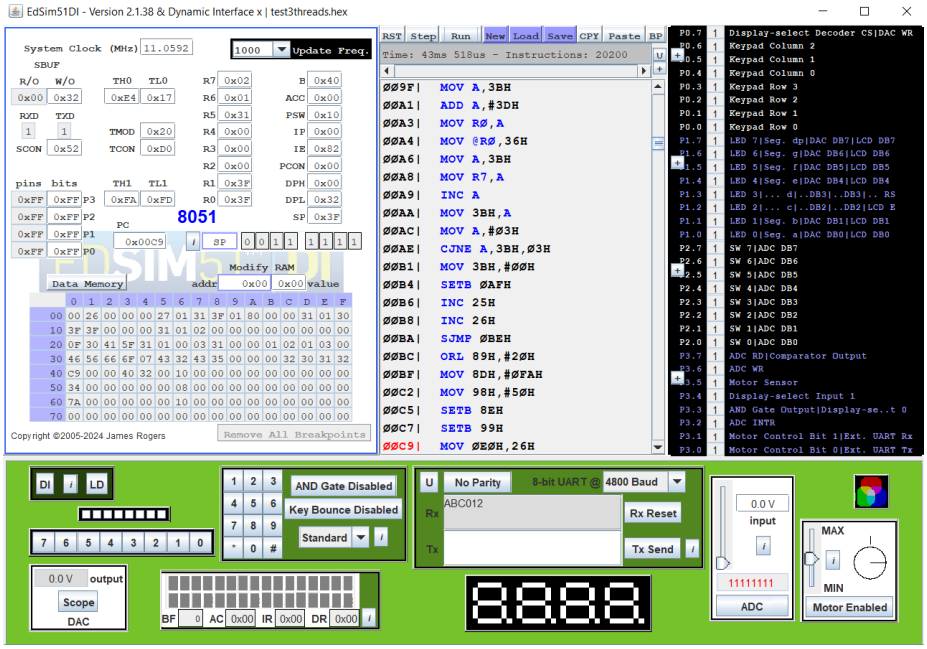
Here, for we to see when the Producer1 is running, we can see the mutex(0x25), full(0x26), and empty(0x27). First, the empty will decrease by 1 and then the mutex and full increased by 1 to become 1-1-2. Then the empty will be decreased by 1 and loop till the empty is 0 and the full becomes 3 in this case.



After producer1 has finished printed out by consumer, the mutex full and empty will go back to 1-0-3 and here comes the producer2.


EdSim51DI - Version 2.1.38 & Dynamic Interface x | test3threads.hex
System Clock (MHz) 11.0592
1000 Update Freq.
8051
RST Step Run New Load Save CPY Paste BP
Time: 32ms 725us - Instructions: 15200
ØØ7A| MOV ØEØH,27H
ØØ7D| JZ ØFBH
ØØ7F| JB ØE7H,ØF8H
ØØ82| DEC 27H
ØØ84| MOV ØEØH,25H
ØØ87| JZ ØFBH
ØØ89| JB ØE7H,ØF8H
ØØ8C| DEC 25H
ØØ8E| CLR ØAFH
ØØ9Ø| MOV A,#39H
ØØ92| CJNE A,36H,Ø5H
ØØ95| MOV 36H,#3ØH
ØØ98| SJMP Ø5H
ØØ9A| MOV A,36H
ØØ9C| INC A
ØØ9D| MOV 36H,A
ØØ9F| MOV A,3BH
ØØA1| ADD A,#3DH
ØØA3| MOV RØ,A
ØØA4| MOV @RØ,36H
ØØA6| MOV A,3BH
Copyright ©2005-2024 James Rogers
Remove All Breakpoints
AND Gate Disabled
Key Bounce Disabled
Standard
No Parity
8-bit UART @ 4800 Baud
Rx Reset
Tx Send
0.0 V input
11111111
ADC
MAX
MIN
Motor Enabled
0.0 V output
Scope
DAC

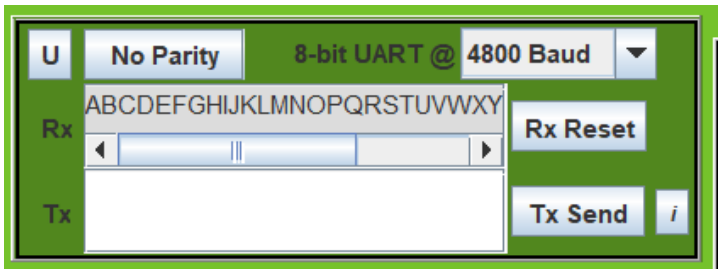

EdSim51DI - Version 2.1.38 & Dynamic Interface x | test3threads.hex
System Clock (MHz) 11.0592
1000 Update Freq.
8051
RST Step Run New Load Save CPY Paste BP
Time: 34ms 461us - Instructions: 16000
ØØ53| INC A
ØØ54| MOV 35H,A
ØØ56| MOV A,3BH
ØØ58| ADD A,#3DH
ØØ5A| MOV RØ,A
ØØ5B| MOV @RØ,35H
ØØ5D| MOV A,3BH
ØØ5F| MOV R7,A
ØØ6Ø| INC A
ØØ61| MOV 3BH,A
ØØ63| MOV A,#Ø3H
ØØ65| CJNE A,3BH,Ø3H
ØØ68| MOV 3BH,#ØØH
ØØ6B| SETB ØAFH
ØØ6D| INC 25H
ØØ6F| INC 26H
ØØ71| SJMP ØBEH
ØØ73| CLR ØAFH
ØØ75| MOV 36H,#2FH
ØØ78| SETB ØAFH
ØØ7A| MOV ØEØH,27H
Copyright ©2005-2024 James Rogers
Remove All Breakpoints
AND Gate Disabled
Key Bounce Disabled
Standard
No Parity
8-bit UART @ 4800 Baud
Rx Reset
Tx Send
0.0 V input
11111111
ADC
MAX
MIN
Motor Enabled
0.0 V output
Scope
DAC



For producer2, now we can see similar behaviour with when producer1 is running, we can see the mutex, full, and empty, where it first from 1-0-3 goes to 1-3-0, that is the full producer2 running.

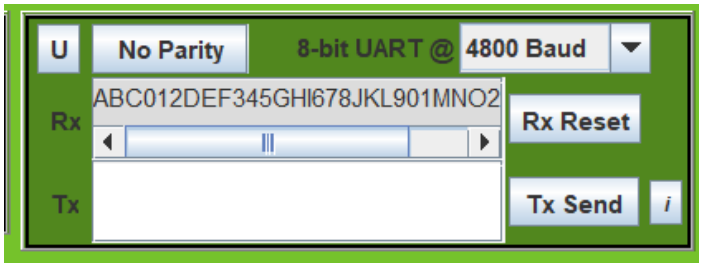
When it is an unfair UART, then only one producer gets to produce the characters, either alphabet or numbers.

Here is an example of the unfair UART:



Above shown the condition where only the producer1 gets to produce and so only alphabets gets printed out.

Here is an example of the fair UART:



Here we can see that each producer produce 3 characters to the share buffer and then gets printed out by the consumer alternately.