Christian Owen 杜銘勝 110006217

OS FINAL PROJECT REPORT (CHECKPOINT 4)

Introduction

In the beginning of project checkpoint, we are asked to implement the producer-consumer functions that produce alphabets in alphabetical order that loops for infinitely using the thread yielding that we have to code it ourselves with the help of the hints given by the project.

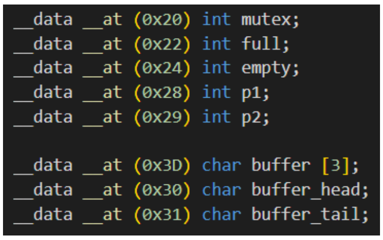
As the project progresses to the next checkpoint, we are asked to implement a scheduler which is called myTimer0Handler, and improve the testing file with the given instruction in the checkpoint assignment.

Third checkpoint starts to implement the semaphores to ensure the fairness of the code and prevent it from raising errors, especially the producer-consumer updates.

Going to the fourth checkpoint, the last change that is left to be made is adding another feature which is another producer that generates numbers ‘0’ to ‘9’ that loops for infinitely.

test3threads.c

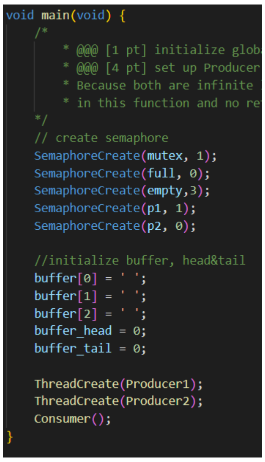
The code starts off with the global initialising of mutex, full, empty, p1 (turn for Producer1), p2 (turn for Producer2) which is mainly for the semaphores that is to be used between the producer-consumer function. As for the other global variables like buffer (3-deep char), buffer head, and buffer tail, it is mainly used to keep up with the bounded buffer circular queue structure.



```
__data __at (0x20) int mutex;
__data __at (0x22) int full;
__data __at (0x24) int empty;
__data __at (0x28) int p1;
__data __at (0x29) int p2;

__data __at (0x3D) char buffer [3];
__data __at (0x30) char buffer_head;
__data __at (0x31) char buffer_tail;
```

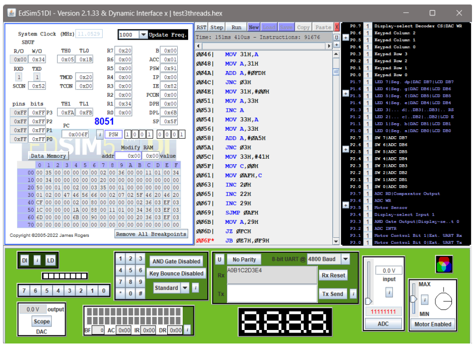
The main function first creates the semaphores that are introduced in the global variables by filling the initial space using the SemaphoreCreate function. Then it continues with initialising the buffer data with the empty character. The buffer head and tail initialise as 0 that is to be updated later on. Lastly, the thread created for Producer1 and Producer2 which is used to generate for Consumer.

The Producer1, Producer2, and Consumer function will be discussed deeper in the later part, but the main context used are the updates of buffer data using alphabets and numbers that are going to be fed to the consumer so that it can be output later on.



```
void main(void) {
    /*
        * @@@ [1 pt] initialize glob
        * @@@ [4 pt] set up Producer
        * Because both are infinite 
        * in this function and no re
    */
    // create semaphore
    SemaphoreCreate(mutex, 1);
    SemaphoreCreate(full, 0);
    SemaphoreCreate(empty,3);
    SemaphoreCreate(p1, 1);
    SemaphoreCreate(p2, 0);

    //initialize buffer, head&tail
    buffer[0] = ' ';
    buffer[1] = ' ';
    buffer[2] = ' ';
    buffer_head = 0;
    buffer_tail = 0;

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

UART Fair Version



EdSim51DI - Version 2.1.33 & Dynamic Interface x | test3threads.hex

```
System Clock (MHz) 11.0529        1000  Update Freq.
SBUF
R/O   W/O     TH0   TL0     R7 0x20     B    0x00
0x00  0x34    0x05  0x1B    R6 0x00     ACC  0x01
RXD   TXD                   R5 0x00     PSW  0x91
1     1       TMOD  0x20    R4 0x00     IP   0x00
SCON  0x52    TCON  0xD0    R3 0x00     IE   0x82
                            R2 0x00     PCON 0x00
pins  bits    TH1   TL1     R1 0x34     DPH  0x00
0xFF  0xFF P3 0xFA  0xFB    R0 0x00     DPL  0x6B
0xFF  0xFF P2                8051       SP   0x5F
0xFF  0xFF P1   PC 0x006F   PSW 1 0 0 1  0 0 0 1
0xFF  0xFF P0
                  Modify RAM
Data Memory       addr 0x00  0x00 value

    0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
00 00 35 00 00 00 00 00 02 00 36 00 00 11 01 00 34
10 00 34 00 00 00 00 00 20 00 00 00 00 00 00 00 00
20 50 00 01 00 02 00 03 35 00 01 00 00 00 00 00 00
30 01 02 00 47 46 56 66 00 02 07 02 5F 46 20 46 20
40 CF 00 00 00 02 00 80 00 00 00 00 02 36 03 EF 03
50 1C 00 00 00 1A 00 88 00 11 01 00 34 36 03 EF 03
60 6D 00 00 00 6B 00 90 00 00 00 00 20 36 03 EF 03
70 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

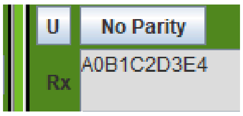
```
Time: 151ms 410us - Instructions: 91676
Ø046|  MOV 31H,A
Ø048|  MOV A,31H
Ø04A|  ADD A,#ØFDH
Ø04C|  JNC Ø3H
Ø04E|  MOV 31H,#ØØH
Ø051|  MOV A,33H
Ø053|  INC A
Ø054|  MOV 33H,A
Ø056|  MOV A,33H
Ø058|  ADD A,#ØA5H
Ø05A|  JNC Ø3H
Ø05C|  MOV 33H,#41H
Ø05F|  MOV C,ØØH
Ø061|  MOV ØAFH,C
Ø063|  INC 2ØH
Ø065|  INC 22H
Ø067|  INC 29H
Ø069|  SJMP ØAFH
Ø06B|  MOV A,29H
Ø06D|  JZ ØFCH
Ø06F* JB ØE7H,ØF9H
```

```
P0.7 1 Display-select Decoder CS|DAC WR
P0.6 1 Keypad Column 2
P0.5 1 Keypad Column 1
P0.4 1 Keypad Column 0
P0.3 1 Keypad Row 3
P0.2 1 Keypad Row 2
P0.1 1 Keypad Row 1
P0.0 1 Keypad Row 0
P1.7 1 LED 7|Seg. dp|DAC DB7|LCD DB7
P1.6 1 LED 6|Seg. g|DAC DB6|LCD DB6
P1.5 1 LED 5|Seg. f|DAC DB5|LCD DB5
P1.4 1 LED 4|Seg. e|DAC DB4|LCD DB4
P1.3 1 LED 3|... d|..DB3|..DB3|.. RS
P1.2 1 LED 2|... c|..DB2|..DB2|LCD E
P1.1 1 LED 1|Seg. b|DAC DB1|LCD DB1
P1.0 1 LED 0|Seg. a|DAC DB0|LCD DB0
P2.7 1 SW 7|ADC DB7
P2.6 1 SW 6|ADC DB6
P2.5 1 SW 5|ADC DB5
P2.4 1 SW 4|ADC DB4
P2.3 1 SW 3|ADC DB3
P2.2 1 SW 2|ADC DB2
P2.1 1 SW 1|ADC DB1
P2.0 1 SW 0|ADC DB0
P3.7 1 ADC RD|Comparator Output
P3.6 1 ADC WR
P3.5 1 Motor Sensor
P3.4 1 Display-select Input 1
P3.3 1 AND Gate Output|Display-se..t 0
P3.2 1 ADC INTR
P3.1 1 Motor Control Bit 1|Ext. UART Rx
P3.0 1 Motor Control Bit 0|Ext. UART Tx
```

No Parity 8-bit UART @ 4800 Baud

Rx: A0B1C2D3E4

The Producer and Consumer uses the semaphores to maintain its fairness (mutex & empty) whereas the Producer1 and Producer2 maintain it by relying on the turns (Producer1 & Producer2) which is to be switch upon finishing its respective task so that alphabets and digits alternately generated for the consumer to send it to the SBUF. Through this, we can see the result on the right: ‘A0B1C2…’ which shows that the fairness that is designed in the code is working.

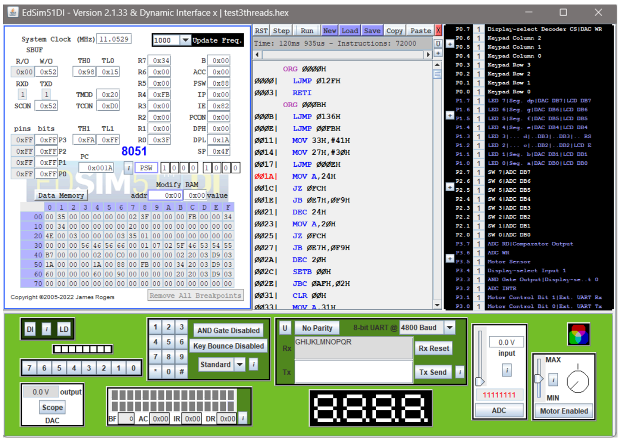


U No Parity

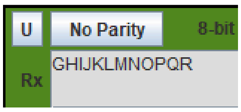
Rx: A0B1C2D3E4

Christian Owen 杜銘勝 110006217

UART Unfair Version

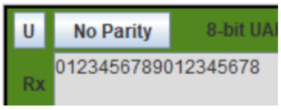

EdSim51DI - Version 2.1.33 & Dynamic Interface x | test3threads.hex
8051
No Parity
8-bit UART @ 4800 Baud
GHIJKLMNOPQR
Rx Reset
Tx Send

After removing the fairness using turns (Producer1 & Producer2) semaphores, the result would appear in 2 different formats where first one will only print out alphabets and second would only print out digits. This is the problem that can be prevented using additional semaphores to keep the generated data in track.



U No Parity 8-bit

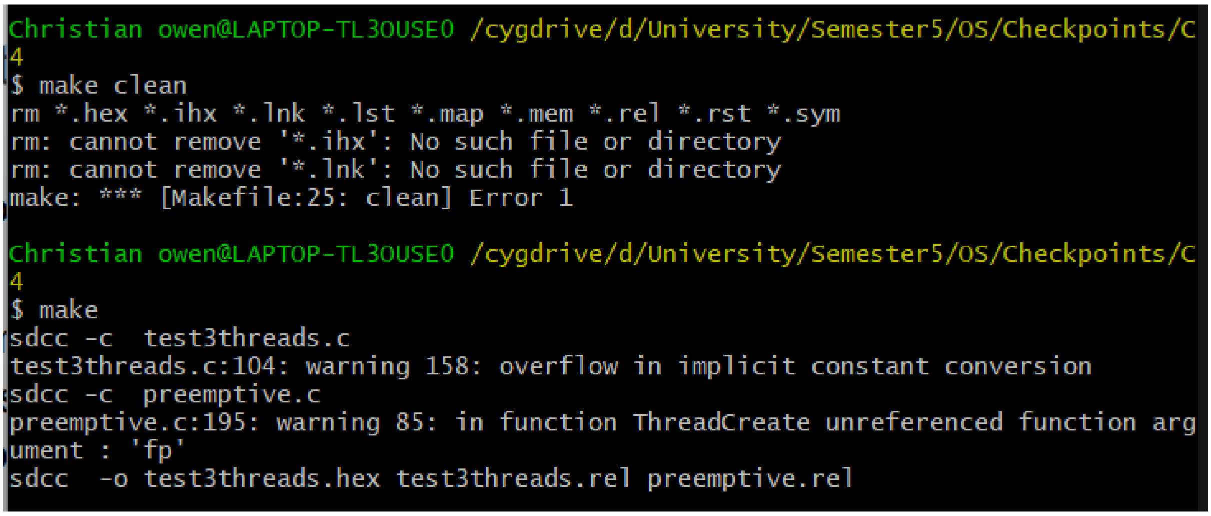
Rx GHIJKLMNOPQR



U No Parity 8-bit UAI

Rx 0123456789012345678

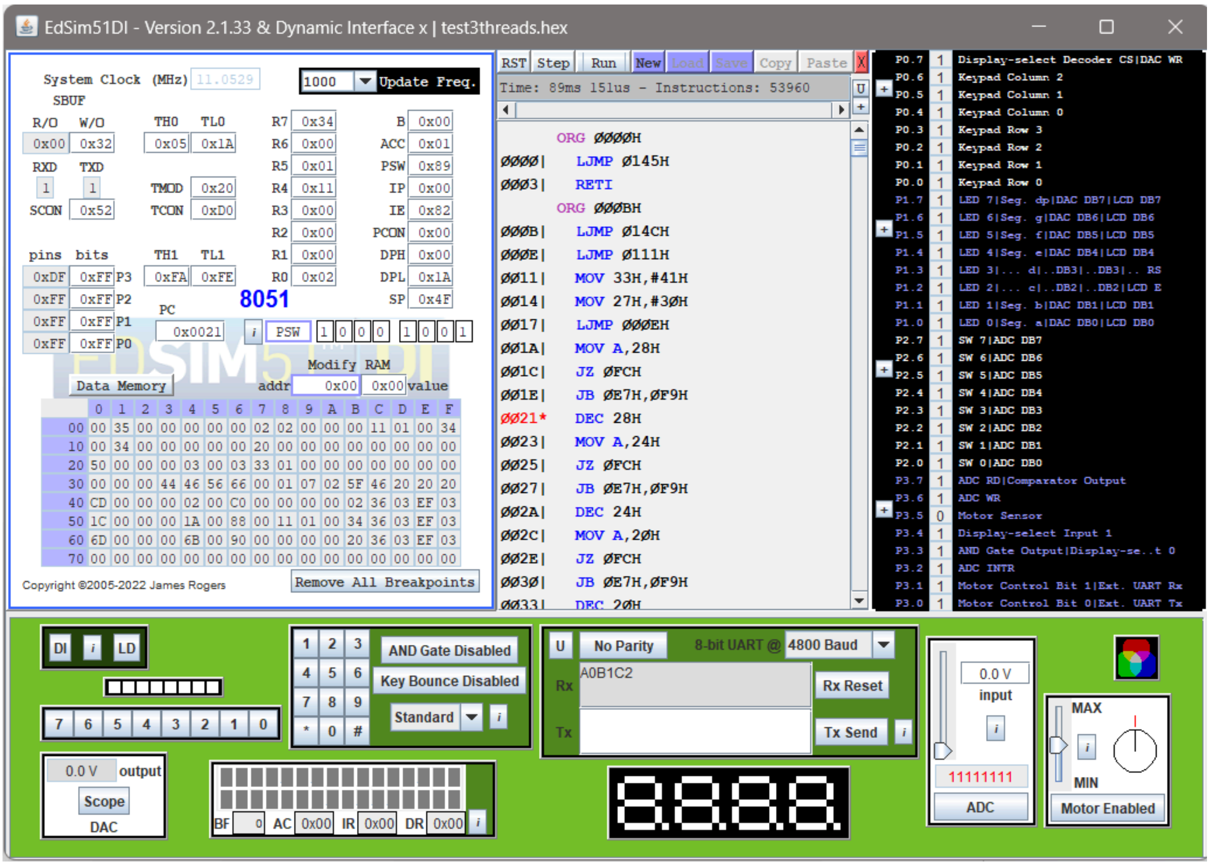
Typescript



```
Christian owen@LAPTOP-TL3OUSEO /cygdrive/d/University/Semester5/OS/Checkpoints/C
4
$ make clean
rm *.hex *.ihx *.lnk *.lst *.map *.mem *.rel *.rst *.sym
rm: cannot remove '*.ihx': No such file or directory
rm: cannot remove '*.lnk': No such file or directory
make: *** [Makefile:25: clean] Error 1

Christian owen@LAPTOP-TL3OUSEO /cygdrive/d/University/Semester5/OS/Checkpoints/C
4
$ make
sdcc -c  test3threads.c
test3threads.c:104: warning 158: overflow in implicit constant conversion
sdcc -c  preemptive.c
preemptive.c:195: warning 85: in function ThreadCreate unreferenced function arg
ument : 'fp'
sdcc  -o test3threads.hex test3threads.rel preemptive.rel
```

Christian Owen 杜銘勝 110006217

Running Producer1:


EdSim51DI - Version 2.1.33 & Dynamic Interface x | test3threads.hex

```
System Clock (MHz) 11.0529        1000 Update Freq.
SBUF
R/O   W/O      TH0   TL0      R7  0x34      B     0x00
0x00  0x32     0x05  0x1A     R6  0x00      ACC   0x01
RXD   TXD                     R5  0x01      PSW   0x89
1     1        TMOD  0x20     R4  0x11      IP    0x00
SCON  0x52     TCON  0xD0     R3  0x00      IE    0x82
                              R2  0x00      PCON  0x00
pins  bits     TH1   TL1      R1  0x00      DPH   0x00
0xDF  0xFF P3  0xFA  0xFE     R0  0x02      DPL   0x1A
0xFF  0xFF P2                 8051          SP    0x4F
0xFF  0xFF P1  PC
0xFF  0xFF P0  0x0021    PSW 1 0 0 0  1 0 0 1
                         Modify RAM
Data Memory        addr 0x00  0x00 value
```

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	00	35	00	00	00	00	00	02	02	00	00	00	11	01	00	34
10	00	34	00	00	00	00	00	20	00	00	00	00	00	00	00	00
20	50	00	00	00	03	00	03	33	01	00	00	00	00	00	00	00
30	00	00	00	44	46	56	66	00	01	07	02	5F	46	20	20	20
40	CD	00	00	00	02	00	C0	00	00	00	00	02	36	03	EF	03
50	1C	00	00	00	1A	00	88	00	11	01	00	34	36	03	EF	03
60	6D	00	00	00	6B	00	90	00	00	00	00	20	36	03	EF	03
70	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

Copyright ©2005-2022 James Rogers

Time: 89ms 151us - Instructions: 53960

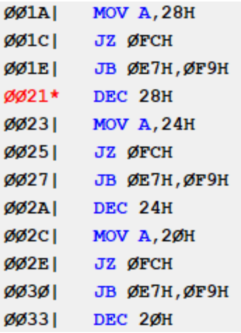
```
        ORG ØØØØH
ØØØØ|   LJMP Ø145H
ØØØ3|   RETI
        ORG ØØØBH
ØØØB|   LJMP Ø14CH
ØØØE|   LJMP Ø111H
ØØ11|   MOV 33H,#41H
ØØ14|   MOV 27H,#3ØH
ØØ17|   LJMP ØØØEH
ØØ1A|   MOV A,28H
ØØ1C|   JZ ØFCH
ØØ1E|   JB ØE7H,ØF9H
ØØ21*   DEC 28H
ØØ23|   MOV A,24H
ØØ25|   JZ ØFCH
ØØ27|   JB ØE7H,ØF9H
ØØ2A|   DEC 24H
ØØ2C|   MOV A,2ØH
ØØ2E|   JZ ØFCH
ØØ3Ø|   JB ØE7H,ØF9H
ØØ33|   DEC 2ØH
```

```
P0.7 1 Display-select Decoder CS|DAC WR
P0.6 1 Keypad Column 2
P0.5 1 Keypad Column 1
P0.4 1 Keypad Column 0
P0.3 1 Keypad Row 3
P0.2 1 Keypad Row 2
P0.1 1 Keypad Row 1
P0.0 1 Keypad Row 0
P1.7 1 LED 7|Seg. dp|DAC DB7|LCD DB7
P1.6 1 LED 6|Seg. g|DAC DB6|LCD DB6
P1.5 1 LED 5|Seg. f|DAC DB5|LCD DB5
P1.4 1 LED 4|Seg. e|DAC DB4|LCD DB4
P1.3 1 LED 3|... d|..DB3|..DB3|.. RS
P1.2 1 LED 2|... c|..DB2|..DB2|LCD E
P1.1 1 LED 1|Seg. b|DAC DB1|LCD DB1
P1.0 1 LED 0|Seg. a|DAC DB0|LCD DB0
P2.7 1 SW 7|ADC DB7
P2.6 1 SW 6|ADC DB6
P2.5 1 SW 5|ADC DB5
P2.4 1 SW 4|ADC DB4
P2.3 1 SW 3|ADC DB3
P2.2 1 SW 2|ADC DB2
P2.1 1 SW 1|ADC DB1
P2.0 1 SW 0|ADC DB0
P3.7 1 ADC RD|Comparator Output
P3.6 1 ADC WR
P3.5 0 Motor Sensor
P3.4 1 Display-select Input 1
P3.3 1 AND Gate Output|Display-se..t 0
P3.2 1 ADC INTR
P3.1 1 Motor Control Bit 1|Ext. UART Rx
P3.0 1 Motor Control Bit 0|Ext. UART Tx
```

No Parity 8-bit UART @ 4800 Baud

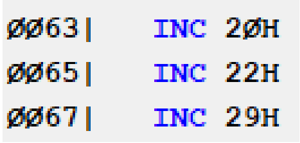
Rx A0B1C2

Below is a semaphore wait implementation for producer1 that decreases the turn (producer1), empty, and mutex.



```
ØØ1A|   MOV A,28H
ØØ1C|   JZ ØFCH
ØØ1E|   JB ØE7H,ØF9H
ØØ21*   DEC 28H
ØØ23|   MOV A,24H
ØØ25|   JZ ØFCH
ØØ27|   JB ØE7H,ØF9H
ØØ2A|   DEC 24H
ØØ2C|   MOV A,2ØH
ØØ2E|   JZ ØFCH
ØØ3Ø|   JB ØE7H,ØF9H
ØØ33|   DEC 2ØH
```

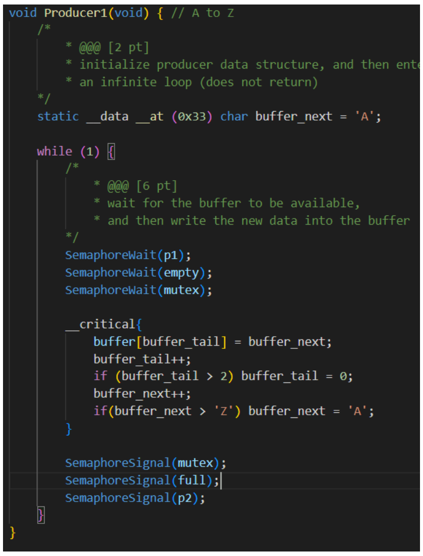
Below is a semaphore signal implementation for producer 1 that increases the turn (producer2), mutex, and full.



```
ØØ63|   INC 2ØH
ØØ65|   INC 22H
ØØ67|   INC 29H
```

Explanation:

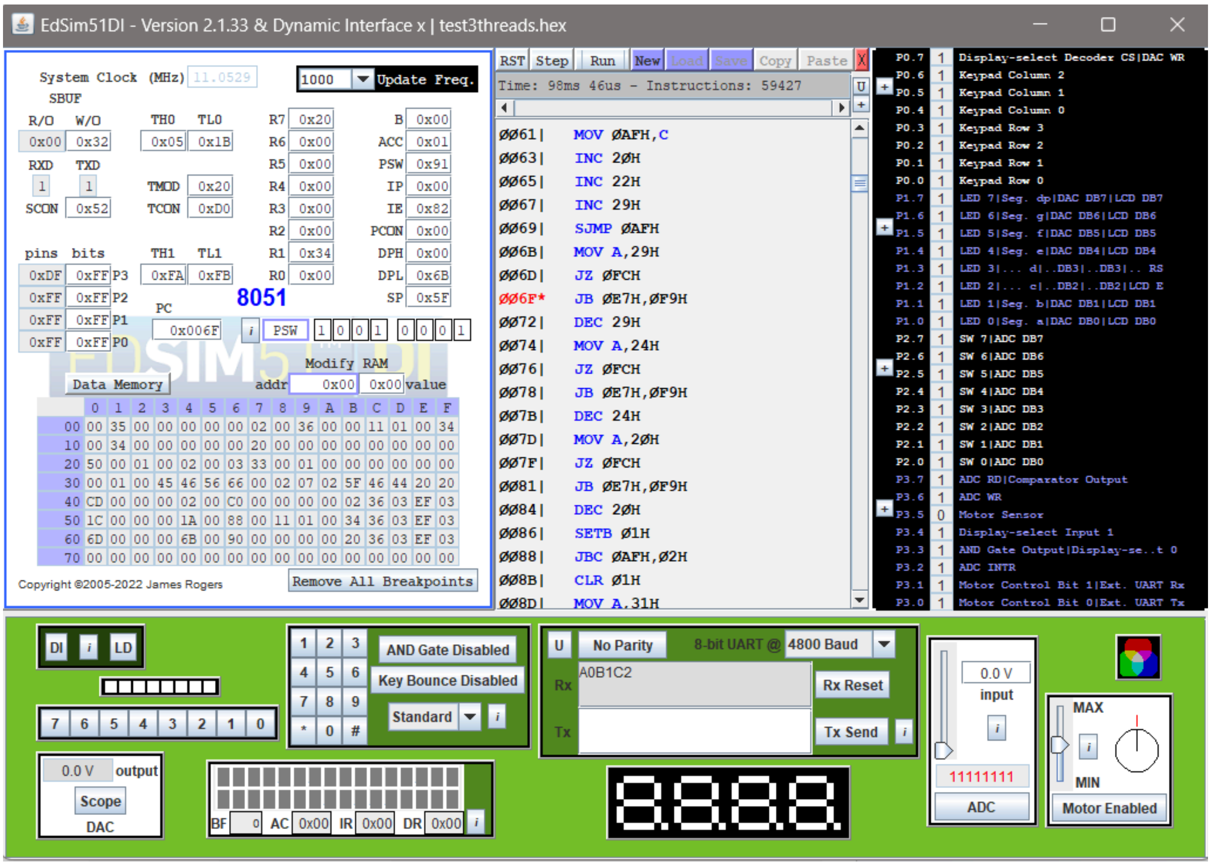
This part of my test3threads file is the producer who updates the buffer data using loops alphabets from ‘A’ to ‘Z’ back to ‘A’ and so on while making sure that the buffer tail is properly updated so that Producer2 and Consumer can be updated by the changes. The semaphore is also used here to keep in track with the switching system between Producer1, Producer2 and Consumer so that it will not raise an error.



```
void Producer1(void) { // A to Z
    /*
        * @@@ [2 pt]
        * initialize producer data structure, and then ente
        * an infinite loop (does not return)
    */
    static __data __at (0x33) char buffer_next = 'A';

    while (1) {
        /*
            * @@@ [6 pt]
            * wait for the buffer to be available,
            * and then write the new data into the buffer
        */
        SemaphoreWait(p1);
        SemaphoreWait(empty);
        SemaphoreWait(mutex);

        __critical{
            buffer[buffer_tail] = buffer_next;
            buffer_tail++;
            if (buffer_tail > 2) buffer_tail = 0;
            buffer_next++;
            if(buffer_next > 'Z') buffer_next = 'A';
        }

        SemaphoreSignal(mutex);
        SemaphoreSignal(full);
        SemaphoreSignal(p2);
    }
}
```

Christian Owen 杜銘勝 110006217

Running Producer2:


EdSim51DI - Version 2.1.33 & Dynamic Interface x | test3threads.hex

```
System Clock (MHz) 11.0529        1000  Update Freq.
SBUF
R/O   W/O       TH0   TL0       R7 0x20      B    0x00
0x00  0x32      0x05  0x1B      R6 0x00      ACC  0x01
RXD   TXD                       R5 0x00      PSW  0x91
1     1         TMOD  0x20      R4 0x00      IP   0x00
SCON  0x52      TCON  0xD0      R3 0x00      IE   0x82
                                R2 0x00      PCON 0x00
pins  bits      TH1   TL1       R1 0x34      DPH  0x00
0xDF  0xFF P3   0xFA  0xFB      R0 0x00      DPL  0x6B
0xFF  0xFF P2              8051              SP   0x5F
0xFF  0xFF P1   PC
0xFF  0xFF P0   0x006F     PSW 1 0 0 1  0 0 0 1
                                Modify RAM
Data Memory               addr 0x00   0x00 value
```

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	00	35	00	00	00	00	00	02	00	36	00	00	11	01	00	34
10	00	34	00	00	00	00	00	20	00	00	00	00	00	00	00	00
20	50	00	01	00	02	00	03	33	00	01	00	00	00	00	00	00
30	00	01	00	45	46	56	66	00	02	07	02	5F	46	44	20	20
40	CD	00	00	00	02	00	C0	00	00	00	00	02	36	03	EF	03
50	1C	00	00	00	1A	00	88	00	11	01	00	34	36	03	EF	03
60	6D	00	00	00	6B	00	90	00	00	00	00	20	36	03	EF	03
70	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

Copyright ©2005-2022 James Rogers

Time: 98ms 46us - Instructions: 59427

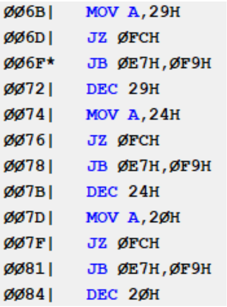
```
ØØ61|   MOV ØAFH,C
ØØ63|   INC 2ØH
ØØ65|   INC 22H
ØØ67|   INC 29H
ØØ69|   SJMP ØAFH
ØØ6B|   MOV A,29H
ØØ6D|   JZ ØFCH
ØØ6F*   JB ØE7H,ØF9H
ØØ72|   DEC 29H
ØØ74|   MOV A,24H
ØØ76|   JZ ØFCH
ØØ78|   JB ØE7H,ØF9H
ØØ7B|   DEC 24H
ØØ7D|   MOV A,2ØH
ØØ7F|   JZ ØFCH
ØØ81|   JB ØE7H,ØF9H
ØØ84|   DEC 2ØH
ØØ86|   SETB Ø1H
ØØ88|   JBC ØAFH,Ø2H
ØØ8B|   CLR Ø1H
ØØ8D|   MOV A,31H
```

```
P0.7  1  Display-select Decoder CS|DAC WR
P0.6  1  Keypad Column 2
P0.5  1  Keypad Column 1
P0.4  1  Keypad Column 0
P0.3  1  Keypad Row 3
P0.2  1  Keypad Row 2
P0.1  1  Keypad Row 1
P0.0  1  Keypad Row 0
P1.7  1  LED 7|Seg. dp|DAC DB7|LCD DB7
P1.6  1  LED 6|Seg. g|DAC DB6|LCD DB6
P1.5  1  LED 5|Seg. f|DAC DB5|LCD DB5
P1.4  1  LED 4|Seg. e|DAC DB4|LCD DB4
P1.3  1  LED 3|... d|..DB3|..DB3|.. RS
P1.2  1  LED 2|... c|..DB2|..DB2|LCD E
P1.1  1  LED 1|Seg. b|DAC DB1|LCD DB1
P1.0  1  LED 0|Seg. a|DAC DB0|LCD DB0
P2.7  1  SW 7|ADC DB7
P2.6  1  SW 6|ADC DB6
P2.5  1  SW 5|ADC DB5
P2.4  1  SW 4|ADC DB4
P2.3  1  SW 3|ADC DB3
P2.2  1  SW 2|ADC DB2
P2.1  1  SW 1|ADC DB1
P2.0  1  SW 0|ADC DB0
P3.7  1  ADC RD|Comparator Output
P3.6  1  ADC WR
P3.5  0  Motor Sensor
P3.4  1  Display-select Input 1
P3.3  1  AND Gate Output|Display-se..t 0
P3.2  1  ADC INTR
P3.1  1  Motor Control Bit 1|Ext. UART Rx
P3.0  1  Motor Control Bit 0|Ext. UART Tx
```

No Parity · 8-bit UART @ 4800 Baud

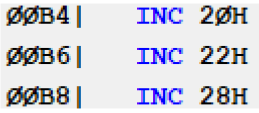
Rx: A0B1C2

Below is a semaphore wait implementation for producer2 that decreases the turn (producer2), empty, and mutex.



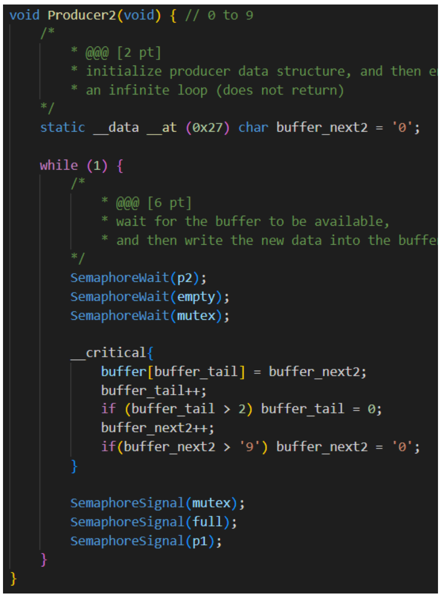
```
ØØ6B|   MOV A,29H
ØØ6D|   JZ ØFCH
ØØ6F*   JB ØE7H,ØF9H
ØØ72|   DEC 29H
ØØ74|   MOV A,24H
ØØ76|   JZ ØFCH
ØØ78|   JB ØE7H,ØF9H
ØØ7B|   DEC 24H
ØØ7D|   MOV A,2ØH
ØØ7F|   JZ ØFCH
ØØ81|   JB ØE7H,ØF9H
ØØ84|   DEC 2ØH
```

Below is a semaphore signal implementation for producer2 that increases the turn (producer1), mutex, and full.



```
ØØB4|   INC 2ØH
ØØB6|   INC 22H
ØØB8|   INC 28H
```

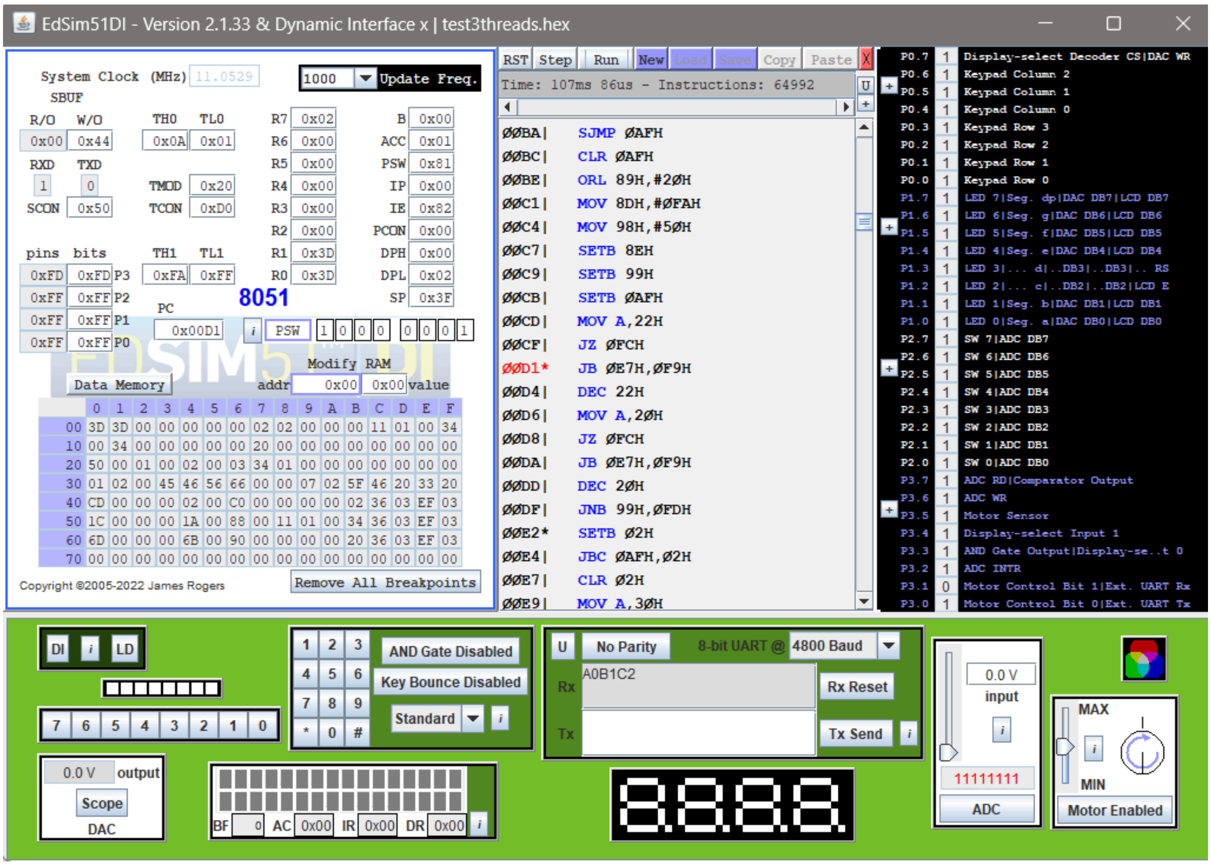
Explanation:
The other part of the Producer function in the file is the Producer2 that has the same functionality as Producer1 but in this case, the buffer data is updated with loop numbers from ‘0’ to ‘9’ back to ‘0’ and so on. The semaphore that is implemented here is also the same as Producer1, except the semaphore for turns will be flipped to let Producer1 update the buffer data in the next turn.



```
void Producer2(void) { // 0 to 9
    /*
        * @@@ [2 pt]
        * initialize producer data structure, and then e
        * an infinite loop (does not return)
    */
    static __data __at (0x27) char buffer_next2 = '0';

    while (1) {
        /*
            * @@@ [6 pt]
            * wait for the buffer to be available,
            * and then write the new data into the buffe
        */
        SemaphoreWait(p2);
        SemaphoreWait(empty);
        SemaphoreWait(mutex);

        __critical{
            buffer[buffer_tail] = buffer_next2;
            buffer_tail++;
            if (buffer_tail > 2) buffer_tail = 0;
            buffer_next2++;
            if(buffer_next2 > '9') buffer_next2 = '0';
        }

        SemaphoreSignal(mutex);
        SemaphoreSignal(full);
        SemaphoreSignal(p1);
    }
}
```

Christian Owen 杜銘勝 110006217

Running Consumer:



EdSim51DI - Version 2.1.33 & Dynamic Interface x | test3threads.hex

System Clock (MHz) 11.0529

1000 Update Freq.

SBUF R/O 0x00 W/O 0x44

RXD 1 TXD 0

SCON 0x50

TH0 0x0A TL0 0x01

TMOD 0x20 TCON 0xD0

TH1 0xFA TL1 0xFF

pins	bits	
0xFD	0xFD	P3
0xFF	0xFF	P2
0xFF	0xFF	P1
0xFF	0xFF	P0

R7 0x02, R6 0x00, R5 0x00, R4 0x00, R3 0x00, R2 0x00, R1 0x3D, R0 0x3D

B 0x00, ACC 0x01, PSW 0x81, IP 0x00, IE 0x82, PCON 0x00, DPH 0x00, DPL 0x02, SP 0x3F

8051

PC 0x00D1

PSW 1 0 0 0 0 0 0 1

Modify RAM addr 0x00 0x00 value

Data Memory

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	3D	3D	00	00	00	00	00	02	02	00	00	00	11	01	00	34
10	00	34	00	00	00	00	00	20	00	00	00	00	00	00	00	00
20	50	00	01	00	02	00	03	34	01	00	00	00	00	00	00	00
30	01	02	00	45	46	56	66	00	00	07	02	5F	46	20	33	20
40	CD	00	00	00	02	00	C0	00	00	00	00	02	36	03	EF	03
50	1C	00	00	00	1A	00	88	00	11	01	00	34	36	03	EF	03
60	6D	00	00	00	6B	00	90	00	00	00	00	20	36	03	EF	03
70	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

Copyright ©2005-2022 James Rogers

Remove All Breakpoints

Time: 107ms 86us - Instructions: 64992

```
ØØBA|    SJMP ØAFH
ØØBC|    CLR ØAFH
ØØBE|    ORL 89H,#2ØH
ØØC1|    MOV 8DH,#ØFAH
ØØC4|    MOV 98H,#5ØH
ØØC7|    SETB 8EH
ØØC9|    SETB 99H
ØØCB|    SETB ØAFH
ØØCD|    MOV A,22H
ØØCF|    JZ ØFCH
ØØD1*    JB ØE7H,ØF9H
ØØD4|    DEC 22H
ØØD6|    MOV A,2ØH
ØØD8|    JZ ØFCH
ØØDA|    JB ØE7H,ØF9H
ØØDD|    DEC 2ØH
ØØDF|    JNB 99H,ØFDH
ØØE2*    SETB Ø2H
ØØE4|    JBC ØAFH,Ø2H
ØØE7|    CLR Ø2H
ØØE9|    MOV A,3ØH
```

```
P0.7  1  Display-select Decoder CS|DAC WR
P0.6  1  Keypad Column 2
P0.5  1  Keypad Column 1
P0.4  1  Keypad Column 0
P0.3  1  Keypad Row 3
P0.2  1  Keypad Row 2
P0.1  1  Keypad Row 1
P0.0  1  Keypad Row 0
P1.7  1  LED 7|Seg. dp|DAC DB7|LCD DB7
P1.6  1  LED 6|Seg. g|DAC DB6|LCD DB6
P1.5  1  LED 5|Seg. f|DAC DB5|LCD DB5
P1.4  1  LED 4|Seg. e|DAC DB4|LCD DB4
P1.3  1  LED 3|... d|..DB3|..DB3|.. RS
P1.2  1  LED 2|... c|..DB2|..DB2|LCD E
P1.1  1  LED 1|Seg. b|DAC DB1|LCD DB1
P1.0  1  LED 0|Seg. a|DAC DB0|LCD DB0
P2.7  1  SW 7|ADC DB7
P2.6  1  SW 6|ADC DB6
P2.5  1  SW 5|ADC DB5
P2.4  1  SW 4|ADC DB4
P2.3  1  SW 3|ADC DB3
P2.2  1  SW 2|ADC DB2
P2.1  1  SW 1|ADC DB1
P2.0  1  SW 0|ADC DB0
P3.7  1  ADC RD|Comparator Output
P3.6  1  ADC WR
P3.5  1  Motor Sensor
P3.4  1  Display-select Input 1
P3.3  1  AND Gate Output|Display-se..t 0
P3.2  1  ADC INTR
P3.1  0  Motor Control Bit 1|Ext. UART Rx
P3.0  1  Motor Control Bit 0|Ext. UART Tx
```

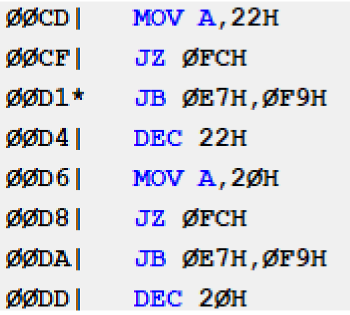
Rx: A0B1C2

input 0.0 V, 11111111

output 0.0 V

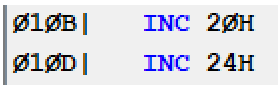
BF 0 AC 0x00 IR 0x00 DR 0x00

Below is a semaphore wait implementation for Consumer that decreases the empty and mutex semaphore.



```
ØØCD|    MOV A,22H
ØØCF|    JZ ØFCH
ØØD1*    JB ØE7H,ØF9H
ØØD4|    DEC 22H
ØØD6|    MOV A,2ØH
ØØD8|    JZ ØFCH
ØØDA|    JB ØE7H,ØF9H
ØØDD|    DEC 2ØH
```

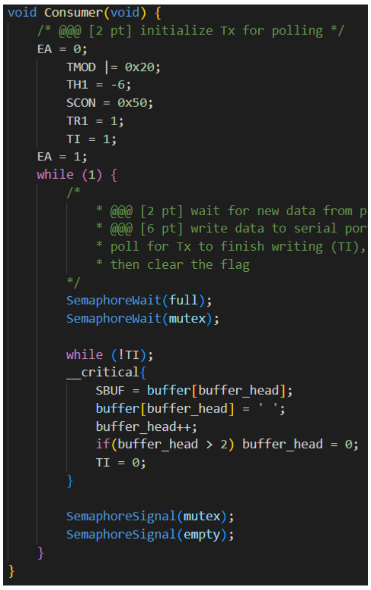
Below is a semaphore signal implementation for producer2 that increases the mutex and full semaphore.



```
Ø1ØB|    INC 2ØH
Ø1ØD|    INC 24H
```

Explanation:

Just like the Producer1 and Producer2, the Consumer relies on the semaphore mutex and full to wait for its turn to take the buffer data into SBUF. It means that it needs to wait till the full semaphore increases till 3 to load the head of buffer data into SBUF. After that, the buffer head will be updated.



```
void Consumer(void) {
    /* @@@ [2 pt] initialize Tx for polling */
    EA = 0;
        TMOD |= 0x20;
        TH1 = -6;
        SCON = 0x50;
        TR1 = 1;
        TI = 1;
    EA = 1;
    while (1) {
        /*
            * @@@ [2 pt] wait for new data from p
            * @@@ [6 pt] write data to serial por
            * poll for Tx to finish writing (TI),
            * then clear the flag
        */
        SemaphoreWait(full);
        SemaphoreWait(mutex);

        while (!TI);
        __critical{
            SBUF = buffer[buffer_head];
            buffer[buffer_head] = ' ';
            buffer_head++;
            if(buffer_head > 2) buffer_head = 0;
            TI = 0;
        }

        SemaphoreSignal(mutex);
        SemaphoreSignal(empty);
    }
}
```