

Report Checkpoint 4

Jonathan Sutedjo 鄭安良 111006207

```
/cygdrive/c/Users/Jonat/Documents/!NTHU Classes/Operating System/Fin Pr/J...
preemptive.c:277: warning 85: in function ThreadCreate unreferenced function arg
ument : 'fp'
sdcc -o test3threads.hex test3threads.rel preemptive.rel

Jonat@LAPTOP-7F30BD0F /cygdrive/c/Users/Jonat/Documents/!NTHU Classes/Operating
System/Fin Pr/Jon/ppc4
$ make clean
rm *.hex *.ihx *.lnk *.lst *.map *.mem *.rel *.rst *.sym *.asm *.lk
rm: cannot remove '*.ihx': No such file or directory
rm: cannot remove '*.lnk': No such file or directory
make: *** [Makefile:25: clean] Error 1

Jonat@LAPTOP-7F30BD0F /cygdrive/c/Users/Jonat/Documents/!NTHU Classes/Operating
System/Fin Pr/Jon/ppc4
$ make
sdcc -c test3threads.c
sdcc -c preemptive.c
preemptive.c:277: warning 85: in function ThreadCreate unreferenced function arg
ument : 'fp'
sdcc -o test3threads.hex test3threads.rel preemptive.rel

Jonat@LAPTOP-7F30BD0F /cygdrive/c/Users/Jonat/Documents/!NTHU Classes/Operating
System/Fin Pr/Jon/ppc4
$ |
```

```
__data __at (0x35) char TheChar;
__data __at (0x36) char TheInt;
__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];
```

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

```

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);

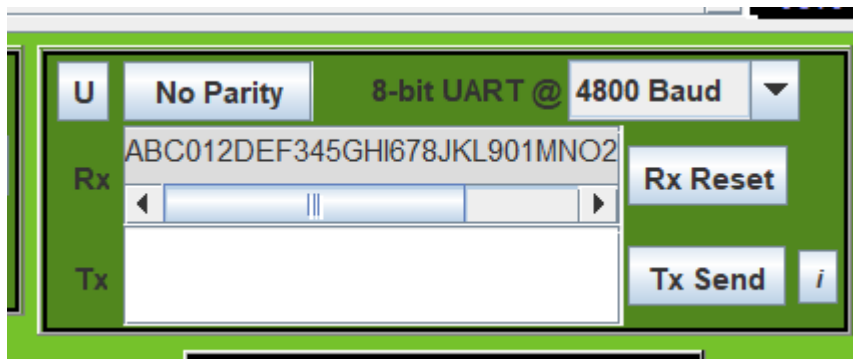
    head = 0;
    tail = 0;

    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 whether 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.

System Clock (MHz) 11.0592 1 Update Freq.

SBUF

R/O	W/O	TH0	TL0	R7	B
0x00	0x00	0x03	0x11	0x32	0x00

RxD	TxD	TMOD	TCN	R6	ACC
1	1	0x20	0xD0	0x03	0x3D

pins	bits	TH1	TL1	R5	PSW
0xFF	0xFF	0xFA	0xFD	0x31	0x09

SCON	PC	R4	IP	R3	IE
0x52	0x05D	0x00	0x00	0x00	0x02

R2	PCON	R1	DPH	R0	DPL
0x80	0x00	0x01	0x00	0x3D	0x00

SP	SP	SP	SP	SP	SP
0x4F	0	1	1	1	1

8051

Data Memory

addr	0x00	0x00	value
0	0	1	2

Remove All Breakpoints

RST Step Run New Load Save CPY Paste BP

Time: 9ms 32us - Instructions: 4200

```
005D| MOV A, 3BH
005F| MOV R7, A
0060| INC A
0061| MOV 3BH, A
0063| MOV A, #03H
0065| CJNE A, 3BH, 03H
0068| MOV 3BH, #00H
006B| SETB 0AFH
006D| INC 25H
006F| INC 26H
0071| SJMP 0BEH
0073| CLR 0AFH
0075| MOV 36H, #2FH
0078| SETB 0AFH
007A| MOV 0E0H, 27H
007D| JZ 0FBH
007F| JB 0E7H, 0F8H
0082| DEC 27H
0084| MOV 0E0H, 25H
0087| JZ 0FBH
0089| JB 0E7H, 0F8H
```

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

DI LD

7 6 5 4 3 2 1 0

0.0V output

Scope DAC

AND Gate Disabled

Key Bounce Disabled

Standard

U No Parity 8-bit UART @ 4800 Baud

Rx Rx Reset

Tx Tx Send

0.0V input

11111111

ADC

MAX MIN

Motor Enabled

BF 0 AC 0x00 IR 0x00 DR 0x00

System Clock (MHz) 11.0592 1 Update Freq.

SBUF

R/O	W/O	TH0	TL0	R7	B
0x00	0x00	0x03	0x1F	0x00	0x00

RxD	TxD	TMOD	TCN	R6	ACC
1	1	0x20	0xD0	0x03	0x02

SCON	PC	R4	IP	R3	IE
0x52	0x034	0x00	0x00	0x00	0x82

R2	PCON	R1	DPH	R0	DPL
0x80	0x00	0x01	0x00	0x3D	0x00

SP	SP	SP	SP	SP	SP
0x4F	0	1	1	1	1

8051

Data Memory

addr	0x00	0x00	value
0	0	1	2

Remove All Breakpoints

RST Step Run New Load Save CPY Paste BP

Time: 9ms 47us - Instructions: 4211

```
0034| JZ 0FBH
0036| JB 0E7H, 0F8H
0039| DEC 27H
003B| MOV 0E0H, 25H
003E| JZ 0FBH
0040| JB 0E7H, 0F8H
0043| DEC 25H
0045| CLR 0AFH
0047| MOV A, #5AH
0049| CJNE A, 35H, 05H
004C| MOV 35H, #41H
004F| SJMP 05H
0051| MOV A, 35H
0053| INC A
0054| MOV 35H, A
0056| ADD A, #3DH
0058| MOV R0, A
005A| MOV R0, A
005B| MOV @R0, 35H
005D| MOV A, 3BH
005F| MOV R7, A
```

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

DI LD

7 6 5 4 3 2 1 0

0.0V output

Scope DAC

AND Gate Disabled

Key Bounce Disabled

Standard

U No Parity 8-bit UART @ 4800 Baud

Rx Rx Reset

Tx Tx Send

0.0V input

11111111

ADC

MAX MIN

Motor Enabled

BF 0 AC 0x00 IR 0x00 DR 0x00

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.

EdSim51DI - Version 2.1.38 & Dynamic Interface x | test3threads.hex

System Clock (MHz) 11.0592 100 Update Freq.

SBUF

R/O	W/O	TH0	TL0	R7	0x02	B	0x40
0x00	0x43	0xCF	0x00	R6	0x02	ACC	0x00
RVD	TxD			R5	0x31	PSW	0x10
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	0x3F	DPH	0x00
0xFF	0xFF	0xFA	0xFA	R0	0x3F	DPL	0x32
0xFF	0xFF					SP	0x3F
0xFF	0xFF						
0xFF	0xFF						

PC 0x00C9

8051

Data Memory

addr	0x00	0x00	value
0	0	1	2
1	00	00	26
2	00	00	00
3	00	00	00
4	00	00	27
5	01	31	3F
6	01	80	00
7	00	00	31
8	01	01	30
9	10	3F	3F
10	00	00	00
11	00	00	31
12	02	02	00
13	00	00	00
14	00	00	00
15	00	00	00
16	00	00	00
17	00	00	00
18	00	00	00
19	00	00	00
20	00	00	00
21	00	00	00
22	00	00	00
23	00	00	00
24	00	00	00
25	00	00	00
26	00	00	00
27	00	00	00
28	00	00	00
29	00	00	00
30	00	00	00
31	00	00	00
32	00	00	00
33	00	00	00
34	00	00	00
35	00	00	00
36	00	00	00
37	00	00	00
38	00	00	00
39	00	00	00
40	00	00	00
41	00	00	00
42	00	00	00
43	00	00	00
44	00	00	00
45	00	00	00
46	00	00	00
47	00	00	00
48	00	00	00
49	00	00	00
50	00	00	00
51	00	00	00
52	00	00	00
53	00	00	00
54	00	00	00
55	00	00	00
56	00	00	00
57	00	00	00
58	00	00	00
59	00	00	00
60	00	00	00
61	00	00	00
62	00	00	00
63	00	00	00
64	00	00	00
65	00	00	00
66	00	00	00
67	00	00	00
68	00	00	00
69	00	00	00
70	00	00	00
71	00	00	00
72	00	00	00
73	00	00	00
74	00	00	00
75	00	00	00
76	00	00	00
77	00	00	00
78	00	00	00
79	00	00	00

Modify RAM

addr 0x00 0x00 value

Remove All Breakpoints

RST Step Run New Load Save CPY Paste BP

Time: 24ms 986us - Instructions: 11600

```

00AE | CJNE A, 3BH, 03H
00B1 | MOV 3BH, #00H
00B4 | SETB 0AFH
00B6 | INC 25H
00B8 | INC 26H
00BA | SJMP 0BEH
00BC | ORL 89H, #20H
00BF | MOV 8DH, #0FAH
00C2 | MOV 98H, #50H
00C5 | SETB 8EH
00C7 | SETB 99H
00C9 | MOV 0E0H, 26H
00CC | JZ 0FBH
00CE | JB 0E7H, 0F8H
00D1 | DEC 26H
00D3 | MOV 0E0H, 25H
00D6 | JZ 0FBH
00D8 | JB 0E7H, 0F8H
00DB | DEC 25H
00DD | CLR 0AFH
00DF | JNB 99H, 0FDH

```

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

DI | LD

7 6 5 4 3 2 1 0

0.0 V output

Scope

DAC

1 2 3 AND Gate Disabled

4 5 6 Key Bounce Disabled

7 8 9 Standard

0 #

U No Parity 8-bit UART @ 4800 Baud

Rx ABC

Rx Reset

Tx

Tx Send

BF 0 AC 0x00 IR 0x00 DR 0x00

0000

0.0 V input

11111111

ADC

MAX

MIN

Motor Enabled

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.

System Clock (MHz) 11.0592 1000 Update Freq.

SBUF

R/O	W/O	TH0	TL0	R7	B
0x00	0x43	0xAD	0x1C	0x02	0x00

RxD	TxD	TMOD	0x20	R5	PSW
1	1	0x52	0xD0	0x31	0x10

SCON	0x52	TCON	0xD0	R4	IP
				0x00	0x00

pins bits

TH1	TL1	R3	IE	R2	PCON
0xFF	0xFF	0xFA	0xFE	0x00	0x00

PC 0x007A

8051

Modify RAM

addr	0x00	0x00	value
0	0	1	2

Data Memory

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	00	26	00	00	00	27	01	31	3F	01	80	00	00	31	01

Copyright ©2005-2024 James Rogers

Remove All Breakpoints

RST Step Run New Load Save CPY Paste BP

Time: 32ms 725us - Instructions: 15200

```
007A| MOV 0E0H,27H
007D| JZ 0FBH
007F| JB 0E7H,0F8H
0082| DEC 27H
0084| MOV 0E0H,25H
0087| JZ 0FBH
0089| JB 0E7H,0F8H
008C| DEC 25H
008E| CLR 0AFH
0090| MOV A,#39H
0092| CJNE A,36H,05H
0095| MOV 36H,#30H
0098| SJMP 05H
009A| MOV A,36H
009C| INC A
009D| MOV 36H,A
009F| MOV A,3BH
00A1| ADD A,#3DH
00A3| MOV R0,A
00A4| MOV @R0,36H
00A6| MOV A,3BH
```

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

DI / LD

7 6 5 4 3 2 1 0

0.0V output

Scope

DAC

BF 0 AC 0x00 IR 0x00 DR 0x00

AND Gate Disabled

Key Bounce Disabled

Standard

U No Parity 8-bit UART @ 4800 Baud

Rx ABC

Rx Reset

Tx

Tx Send

0.0V input

11111111

ADC

MAX

MIN

Motor Enabled

System Clock (MHz) 11.0592 1000 Update Freq.

SBUF

R/O	W/O	TH0	TL0	R7	B
0x00	0x43	0xDF	0x1C	0x02	0x00

RxD	TxD	TMOD	0x20	R5	PSW
1	1	0x52	0xD0	0x31	0x10

SCON	0x52	TCON	0xD0	R4	IP
				0x00	0x00

pins bits

TH1	TL1	R3	IE	R2	PCON
0xFF	0xFF	0xFA	0xFC	0x00	0x00

PC 0x007A

8051

Modify RAM

addr	0x00	0x00	value
0	0	1	2

Data Memory

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	00	26	00	00	00	27	01	31	3F	01	80	00	00	31	01

Copyright ©2005-2024 James Rogers

Remove All Breakpoints

RST Step Run New Load Save CPY Paste BP

Time: 34ms 461us - Instructions: 16000

```
0053| INC A
0054| MOV 35H,A
0056| MOV A,3BH
0058| ADD A,#3DH
005A| MOV R0,A
005B| MOV @R0,35H
005D| MOV A,3BH
005F| MOV R7,A
0060| INC A
0061| MOV 3BH,A
0063| MOV A,#03H
0065| CJNE A,3BH,03H
0068| MOV 3BH,#00H
006B| SETB 0AFH
006D| INC 25H
006F| INC 26H
0071| SJMP 0BEH
0073| CLR 0AFH
0075| MOV 36H,#2FH
0078| SETB 0AFH
007A| MOV 0E0H,27H
```

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

DI / LD

7 6 5 4 3 2 1 0

0.0V output

Scope

DAC

BF 0 AC 0x00 IR 0x00 DR 0x00

AND Gate Disabled

Key Bounce Disabled

Standard

U No Parity 8-bit UART @ 4800 Baud

Rx ABC

Rx Reset

Tx

Tx Send

0.0V input

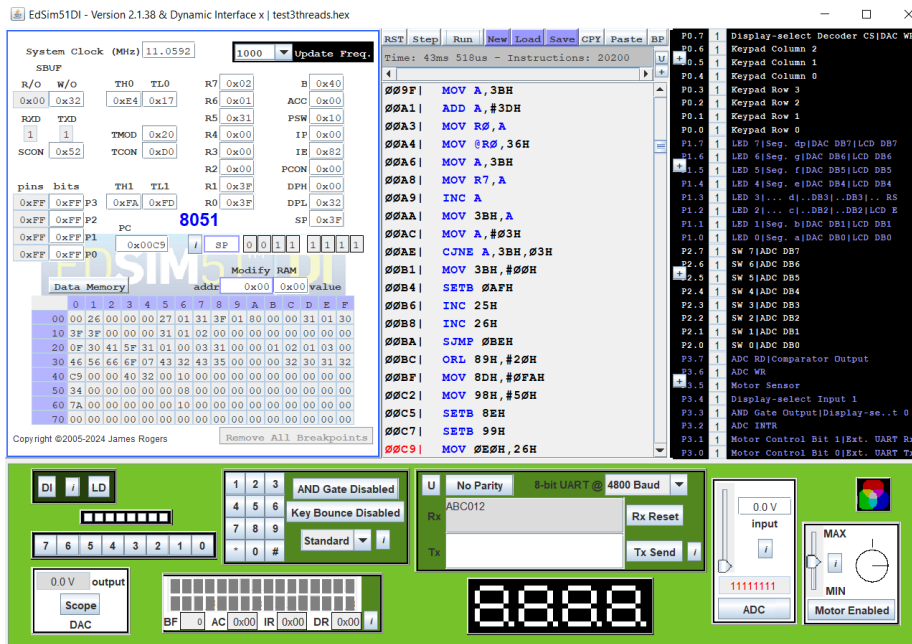
11111111

ADC

MAX

MIN

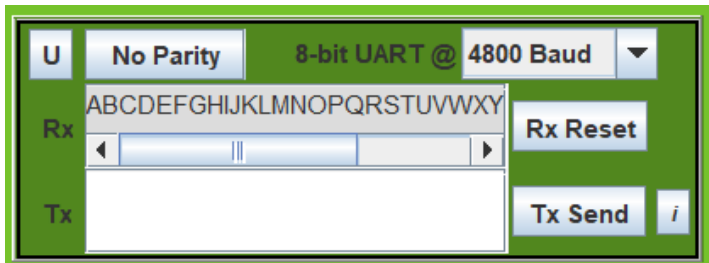
Motor Enabled



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.

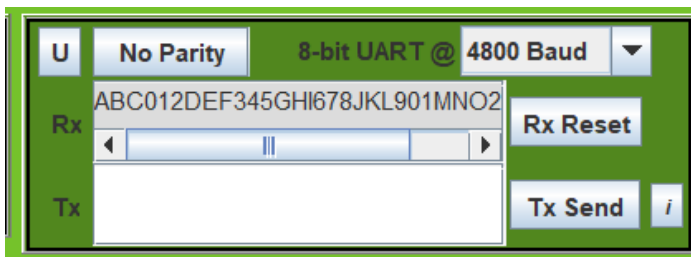
When it is an unfair UART, then only one producer gets to produce the characters, either alphabet or numbers.

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.