PPC4

ID: **111061894**
Name: **Snehit**

Setup

Software	Purpose
Cygwin (3.5.4)	For Unix-like Environment
SDCC (4.4.0)	compiler suite that targets the Intel MCS51 based microprocessors
Notepad++ (8.7.1)	Write and edit .c files
EdSim51DI (2.1.36)	Simulator for 8051

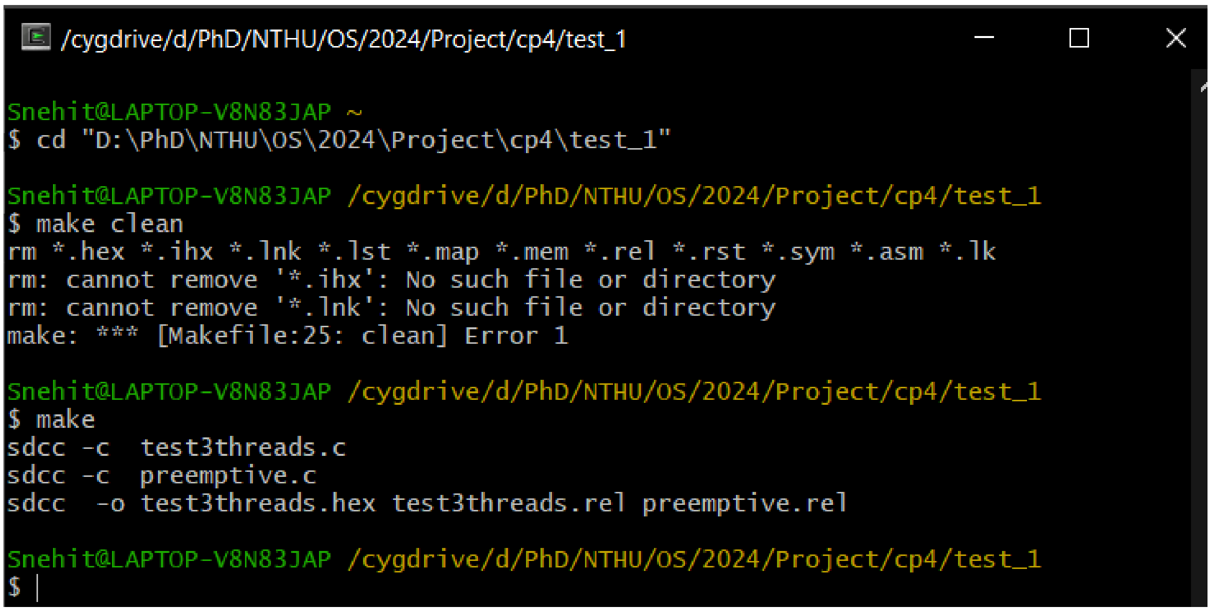
Table 1: Setup describing Software with respective version and purpose.

Creating and Compiling Makefile

Using same Makefile from CP3 with modifying filenames as "testpreempt" to "test3threads" so that it can be compatible with CP4. Running the following commands in Cygwin (3.5.4)

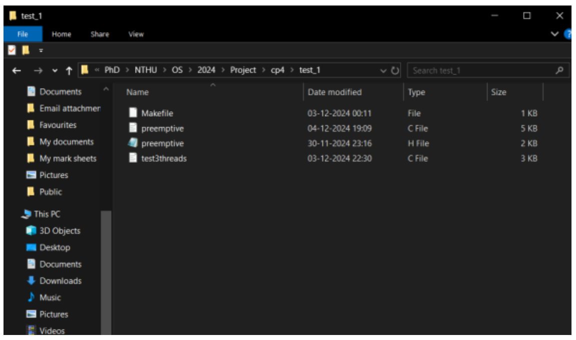
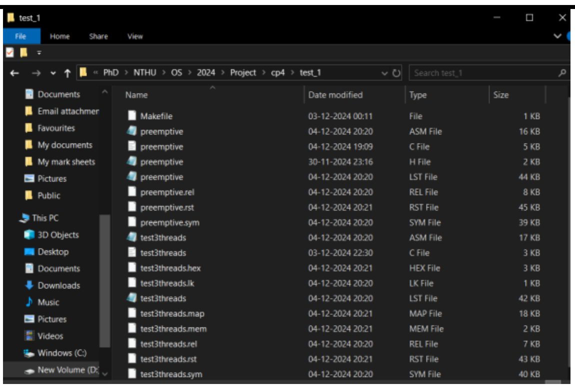
```
$ make clean
$ make
```

as shown in Fig. 1. *make clean* will clear the files generated from previous execution (if any) and then *make* command will create new require file as per the code written in *.c* files. Table 2 shows the result of respective make command. Note, screenshots (Fig. 1 and Table 2) below are for test3threads using Approach 2 (discussed onward).



```
/cygdrive/d/PhD/NTHU/OS/2024/Project/cp4/test_1

Snehit@LAPTOP-V8N83JAP ~
$ cd "D:\PhD\NTHU\OS\2024\Project\cp4\test_1"

Snehit@LAPTOP-V8N83JAP /cygdrive/d/PhD/NTHU/OS/2024/Project/cp4/test_1
$ 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

Snehit@LAPTOP-V8N83JAP /cygdrive/d/PhD/NTHU/OS/2024/Project/cp4/test_1
$ make
sdcc -c  test3threads.c
sdcc -c  preemptive.c
sdcc  -o test3threads.hex test3threads.rel preemptive.rel

Snehit@LAPTOP-V8N83JAP /cygdrive/d/PhD/NTHU/OS/2024/Project/cp4/test_1
$ |
```

Fig. 1: Screenshot of Cygwin after running *make clean* and *make* command.

After *$ make clean*	**After *$ make***
	

After *$ make clean* — test_1 (PhD > NTHU > OS > 2024 > Project > cp4 > test_1)

Name	Date modified	Type	Size
Makefile	03-12-2024 00:11	File	1 KB
preemptive	04-12-2024 19:09	C File	5 KB
preemptive	30-11-2024 23:16	H File	2 KB
test3threads	03-12-2024 22:30	C File	3 KB

After *$ make* — test_1 (PhD > NTHU > OS > 2024 > Project > cp4 > test_1)

Name	Date modified	Type	Size
Makefile	03-12-2024 00:11	File	1 KB
preemptive	04-12-2024 20:20	ASM File	16 KB
preemptive	04-12-2024 19:09	C File	5 KB
preemptive	30-11-2024 23:16	H File	2 KB
preemptive	04-12-2024 20:20	LST File	44 KB
preemptive.rel	04-12-2024 20:20	REL File	8 KB
preemptive.rst	04-12-2024 20:21	RST File	45 KB
preemptive.sym	04-12-2024 20:20	SYM File	39 KB
test3threads	04-12-2024 20:20	ASM File	17 KB
test3threads	03-12-2024 22:30	C File	3 KB
test3threads.hex	04-12-2024 20:21	HEX File	3 KB
test3threads.lk	04-12-2024 20:20	LK File	1 KB
test3threads	04-12-2024 20:20	LST File	42 KB
test3threads.map	04-12-2024 20:21	MAP File	18 KB
test3threads.mem	04-12-2024 20:21	MEM File	2 KB
test3threads.rel	04-12-2024 20:20	REL File	7 KB
test3threads.rst	04-12-2024 20:21	RST File	43 KB
test3threads.sym	04-12-2024 20:20	SYM File	40 KB

Table 2: results of Makefile compilation

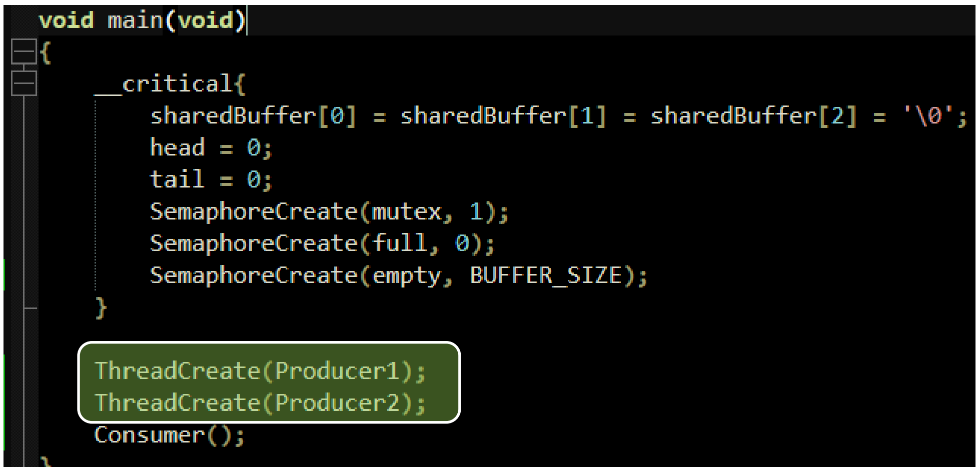
Fairness:

For checkpoint 4 there are two producers, Producer1 is for producing characters “A” – “Z” and repeating the same and Producer2 producing characters “0” – “9” also repeating. Trying different orders of spawning threads with keeping the remaining part of code same as of checkpoint 3.

Spawning order 1:

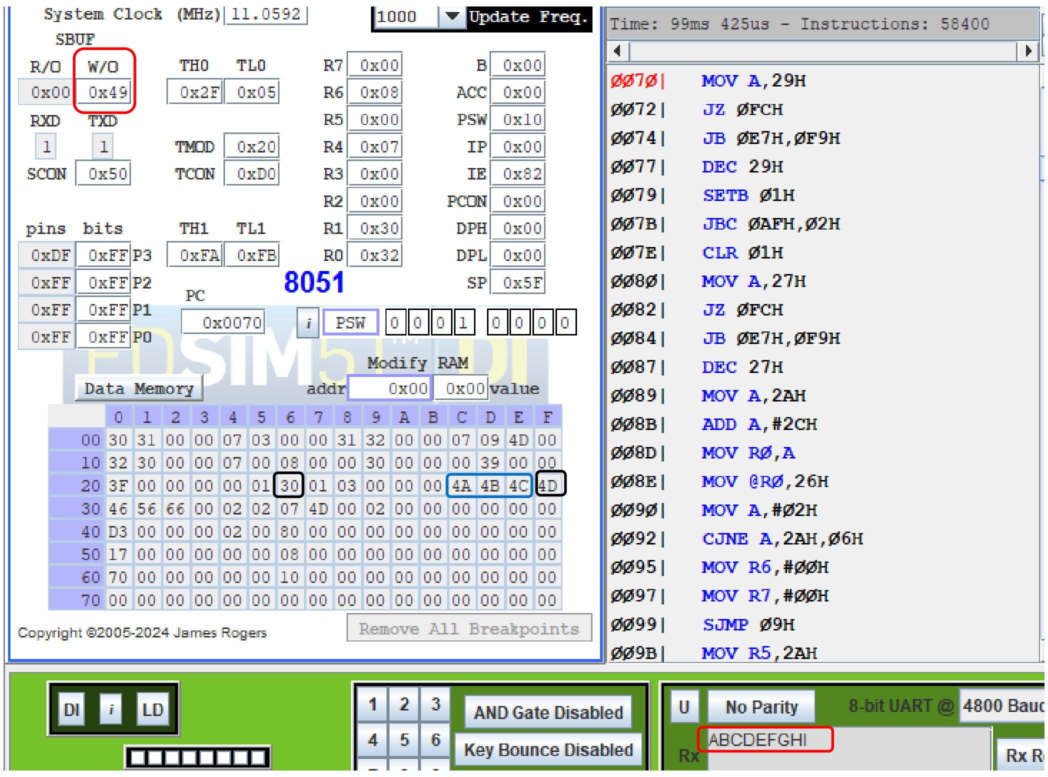
Using the order of spawning as Producer1 and then Producer2, as shown in Fig.2 the output of which can be observed in the Fig. 3. Here the output as from the UART shows execution of only Producer1. Also, the variable sharedBuffer at memory location 0x2C to 0x2E shows the next characters to be transferred are 4AH, 4BH, 4CH (i.e. “J”, “K”, “L”).

Memory location 0x26 used in Producer2 with variable name currentDig is used to count the digits from “0” to “9”, is at value 30H (i.e. “0”) which means Producer2 is initiated but this character never got transferred to SBUF to display at UART. And hence never updated later to new values.



```c
void main(void)
{
    __critical{
        sharedBuffer[0] = sharedBuffer[1] = sharedBuffer[2] = '\0';
        head = 0;
        tail = 0;
        SemaphoreCreate(mutex, 1);
        SemaphoreCreate(full, 0);
        SemaphoreCreate(empty, BUFFER_SIZE);
    }

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

Fig 2: Screenshot of main Order of spawning Producer1 then Producer2



```
System Clock (MHz) 11.0592        1000   Update Freq.
SBUF
R/O   W/O      TH0   TL0     R7 0x00      B 0x00
0x00  0x49     0x2F  0x05    R6 0x08    ACC 0x00
RXD   TXD                    R5 0x00    PSW 0x10
 1     1       TMOD  0x20    R4 0x07     IP 0x00
SCON  0x50     TCON  0xD0    R3 0x00     IE 0x82
                             R2 0x00   PCON 0x00
pins  bits     TH1   TL1     R1 0x30    DPH 0x00
0xDF  0xFF P3  0xFA  0xFB    R0 0x32    DPL 0x00
0xFF  0xFF P2             8051           SP 0x5F
0xFF  0xFF P1  PC
0xFF  0xFF P0  0x0070     i  PSW  0 0 0 1  0 0 0 0
                          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	30	31	00	00	07	03	00	00	31	32	00	00	07	09	4D	00
10	32	30	00	00	07	00	08	00	00	30	00	00	00	39	00	00
20	3F	00	00	00	00	01	30	01	03	00	00	00	4A	4B	4C	4D
30	46	56	66	00	02	02	07	4D	00	02	00	00	00	00	00	00
40	D3	00	00	00	02	00	80	00	00	00	00	00	00	00	00	00
50	17	00	00	00	00	00	08	00	00	00	00	00	00	00	00	00
60	70	00	00	00	00	00	10	00	00	00	00	00	00	00	00	00
70	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

Copyright ©2005-2024 James Rogers

```
Time: 99ms 425us - Instructions: 58400

ØØ7Ø| MOV A,29H
ØØ72| JZ ØFCH
ØØ74| JB ØE7H,ØF9H
ØØ77| DEC 29H
ØØ79| SETB Ø1H
ØØ7B| JBC ØAFH,Ø2H
ØØ7E| CLR Ø1H
ØØ8Ø| MOV A,27H
ØØ82| JZ ØFCH
ØØ84| JB ØE7H,ØF9H
ØØ87| DEC 27H
ØØ89| MOV A,2AH
ØØ8B| ADD A,#2CH
ØØ8D| MOV RØ,A
ØØ8E| MOV @RØ,26H
ØØ9Ø| MOV A,#Ø2H
ØØ92| CJNE A,2AH,Ø6H
ØØ95| MOV R6,#ØØH
ØØ97| MOV R7,#ØØH
ØØ99| SJMP Ø9H
ØØ9B| MOV R5,2AH
```

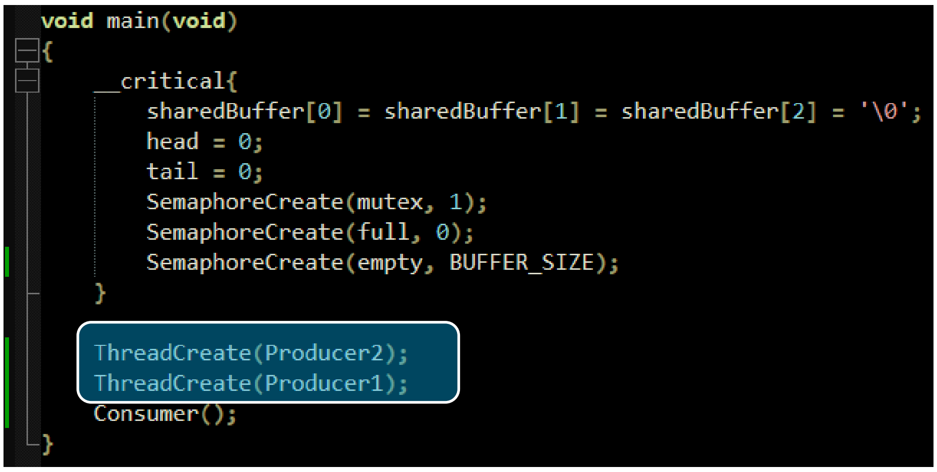
```
ABCDEFGHI
```

Fig 4: EdSim51 showing result for Order of spawning Producer1 then Producer2

Spawning order 2:

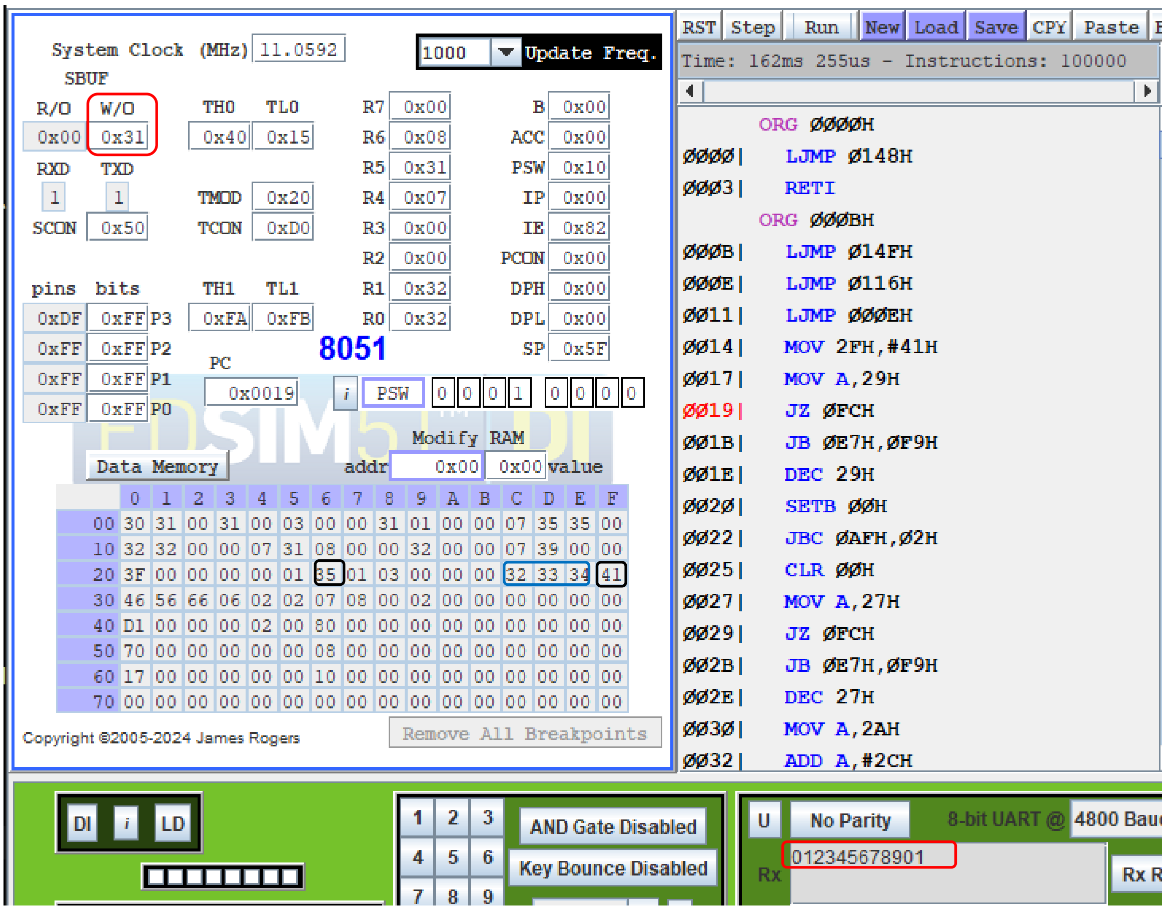
Using the order of spawning as Producer2 and then Producer1, as shown in Fig.4 the output of which can be observed in the Fig. 5. Here the output as from the UART shows execution of only Producer2. Also, the variable sharedBuffer at memory location 0x2C to 0x2E shows the next characters to be transferred are 32H, 33H, 34H (i.e. “2”, “3”, “4”).

The memory location 0x2F used in producer1 with variable name currentChar is used to update the character from “A” to “Z”, is at value 41H (i.e. “A”) which means Producer1 is initiated but this character never got transferred to SBUF to display at UART. And hence never updated later to new values.



```
void main(void)
{
    __critical{
        sharedBuffer[0] = sharedBuffer[1] = sharedBuffer[2] = '\0';
        head = 0;
        tail = 0;
        SemaphoreCreate(mutex, 1);
        SemaphoreCreate(full, 0);
        SemaphoreCreate(empty, BUFFER_SIZE);
    }

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

Fig 4: Screenshot of main Order of spawning Producer2 then Producer1



System Clock (MHz) 11.0592 | 1000 Update Freq.

SBUF R/O 0x00 W/O 0x31 | RXD 1 TXD 1 | SCON 0x50

TH0 0x40 TL0 0x15 | TMOD 0x20 | TCON 0xD0 | TH1 0xFA TL1 0xFB

R7 0x00 | R6 0x08 | R5 0x31 | R4 0x07 | R3 0x00 | R2 0x00 | R1 0x32 | R0 0x32

B 0x00 | ACC 0x00 | PSW 0x10 | IP 0x00 | IE 0x82 | PCON 0x00 | DPH 0x00 | DPL 0x00 | SP 0x5F

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

8051 PC 0x0019 | PSW 0 0 0 1 0 0 0 0

Modify RAM addr 0x00 value 0x00

Data Memory

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	30	31	00	31	00	03	00	00	31	01	00	00	07	35	35	00
10	32	32	00	00	07	31	08	00	00	32	00	00	07	39	00	00
20	3F	00	00	00	00	01	35	01	03	00	00	00	32	33	34	41
30	46	56	66	06	02	02	07	08	00	02	00	00	00	00	00	00
40	D1	00	00	00	02	00	80	00	00	00	00	00	00	00	00	00
50	70	00	00	00	00	00	08	00	00	00	00	00	00	00	00	00
60	17	00	00	00	00	00	10	00	00	00	00	00	00	00	00	00
70	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

Copyright ©2005-2024 James Rogers

Time: 162ms 255us - Instructions: 100000

```
      ORG ØØØØH
ØØØØ|   LJMP Ø148H
ØØØ3|   RETI
      ORG ØØØBH
ØØØB|   LJMP Ø14FH
ØØØE|   LJMP Ø116H
ØØ11|   LJMP ØØØEH
ØØ14|   MOV 2FH,#41H
ØØ17|   MOV A,29H
ØØ19|   JZ ØFCH
ØØ1B|   JB ØE7H,ØF9H
ØØ1E|   DEC 29H
ØØ2Ø|   SETB ØØH
ØØ22|   JBC ØAFH,Ø2H
ØØ25|   CLR ØØH
ØØ27|   MOV A,27H
ØØ29|   JZ ØFCH
ØØ2B|   JB ØE7H,ØF9H
ØØ2E|   DEC 27H
ØØ3Ø|   MOV A,2AH
ØØ32|   ADD A,#2CH
```

No Parity 8-bit UART @ 4800 Bau

Rx 012345678901

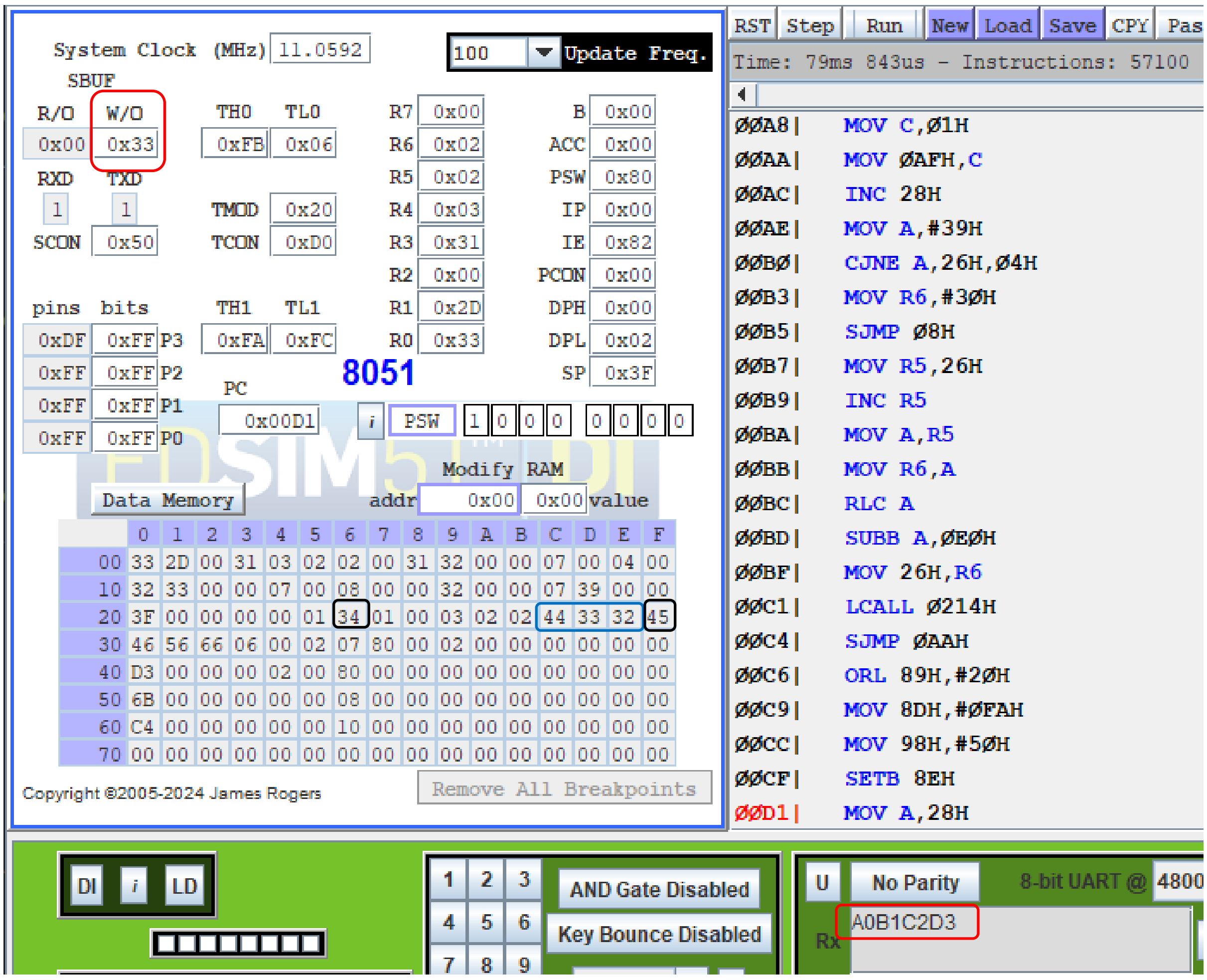
Fig 6: EdSim51 showing result for Order of spawning Producer2 then Producer1

Approach 1: Using ThreadYield()

The method of ThreadYield which has been done in Checkpoint 1 is tried for two producers to be used for one consumer to yield alternatively, the output of which is shown in Fig. 7. UART displays the characters containing alphabets and digits alternatively i.e. “A0B1C2D3” where alphabets are from Producer1, and digits are from Producer2.

Observe variable sharedBuffer at memory location 0x2C to 0x2E which has the value 44H, 33H, 32H (i.e. “D”, “3”, “2”) this are the values used to transfer to SBUF which are displayed at UART. See the value at 0x2F i.e. 45H (i.e. “E”) which indicates new character going to be transferred to sharedBuffer which belongs to Producer1. Similarly, at 0x26 i.e. 34H (i.e. “4”) which is the new digit used to update sharedBuffer.

Overall, this does shows both producers are spawned alternatively, where it updates one location sharedBuffer from one producer and jumps to other producer to update other location of sharedBuffer and the cycle repeats in alternate fashion. But the ISR is unused in this approach.



```
System Clock (MHz) 11.0592        100   Update Freq.
SBUF
R/O   W/O     TH0   TL0     R7 0x00     B    0x00
0x00  0x33    0xFB  0x06    R6 0x02     ACC  0x00
RXD   TXD                   R5 0x02     PSW  0x80
1     1       TMOD  0x20    R4 0x03     IP   0x00
SCON  0x50    TCON  0xD0    R3 0x31     IE   0x82
                            R2 0x00     PCON 0x00
pins  bits    TH1   TL1     R1 0x2D     DPH  0x00
0xDF  0xFF P3 0xFA  0xFC    R0 0x33     DPL  0x02
0xFF  0xFF P2                           SP   0x3F
0xFF  0xFF P1 PC
0xFF  0xFF P0 0x00D1     PSW 1 0 0 0  0 0 0 0
                         addr 0x00  0x00 value
```

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	33	2D	00	31	03	02	02	00	31	32	00	00	07	00	04	00
10	32	33	00	00	07	00	08	00	00	32	00	00	07	39	00	00
20	3F	00	00	00	00	01	34	01	00	03	02	02	44	33	32	45
30	46	56	66	06	00	02	07	80	00	02	00	00	00	00	00	00
40	D3	00	00	00	02	00	80	00	00	00	00	00	00	00	00	00
50	6B	00	00	00	00	00	08	00	00	00	00	00	00	00	00	00
60	C4	00	00	00	00	00	10	00	00	00	00	00	00	00	00	00
70	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

Copyright ©2005-2024 James Rogers

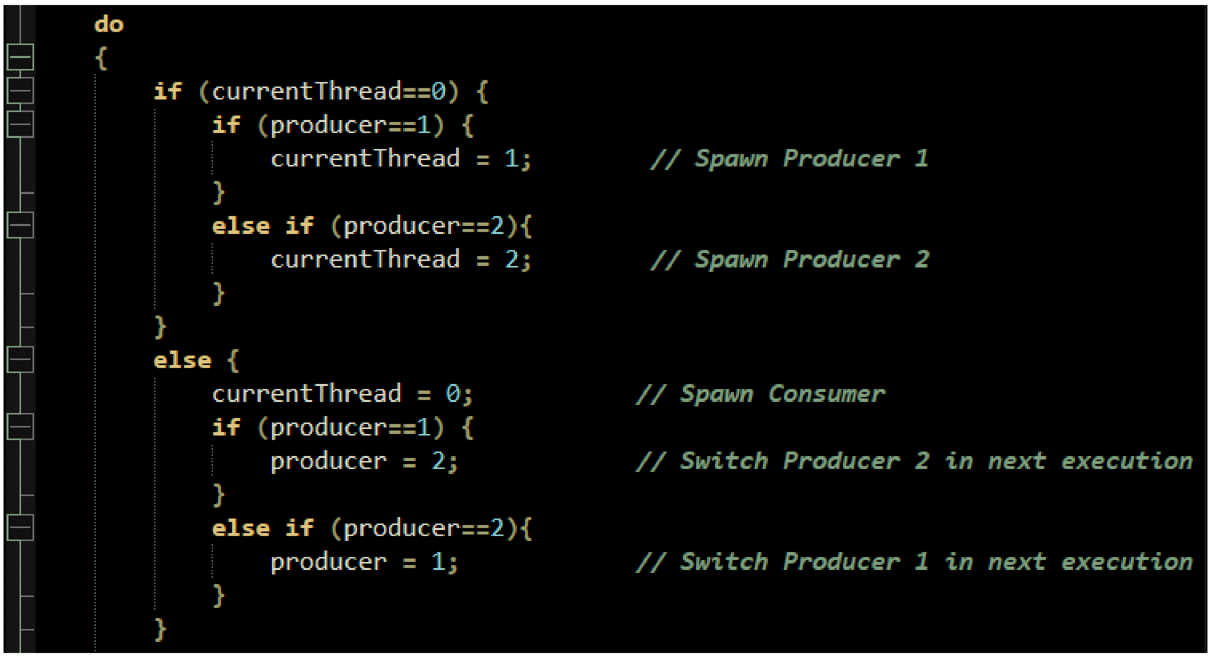
```
Time: 79ms 843us - Instructions: 57100
ØØA8|   MOV C,Ø1H
ØØAA|   MOV ØAFH,C
ØØAC|   INC 28H
ØØAE|   MOV A,#39H
ØØBØ|   CJNE A,26H,Ø4H
ØØB3|   MOV R6,#3ØH
ØØB5|   SJMP Ø8H
ØØB7|   MOV R5,26H
ØØB9|   INC R5
ØØBA|   MOV A,R5
ØØBB|   MOV R6,A
ØØBC|   RLC A
ØØBD|   SUBB A,ØEØH
ØØBF|   MOV 26H,R6
ØØC1|   LCALL Ø214H
ØØC4|   SJMP ØAAH
ØØC6|   ORL 89H,#2ØH
ØØC9|   MOV 8DH,#ØFAH
ØØCC|   MOV 98H,#5ØH
ØØCF|   SETB 8EH
ØØD1|   MOV A,28H
```

```
A0B1C2D3
```

Fig 7: EdSim51 showing result for spawning using Threadyield().

Approach 2: Using ISR (myTimer0Handler) with necessary modification

Since myTimer0Handler i.e. ISR is used to handle Threadyield (reference from Checkpoint 2) but is unable to cope up with two producers. Here some modifications have been made so that it can handle the spawning of two producers alternately. Figure 8 shows the section of myTimer0Handler where modifications added to handle switching between Producers and Consumer.



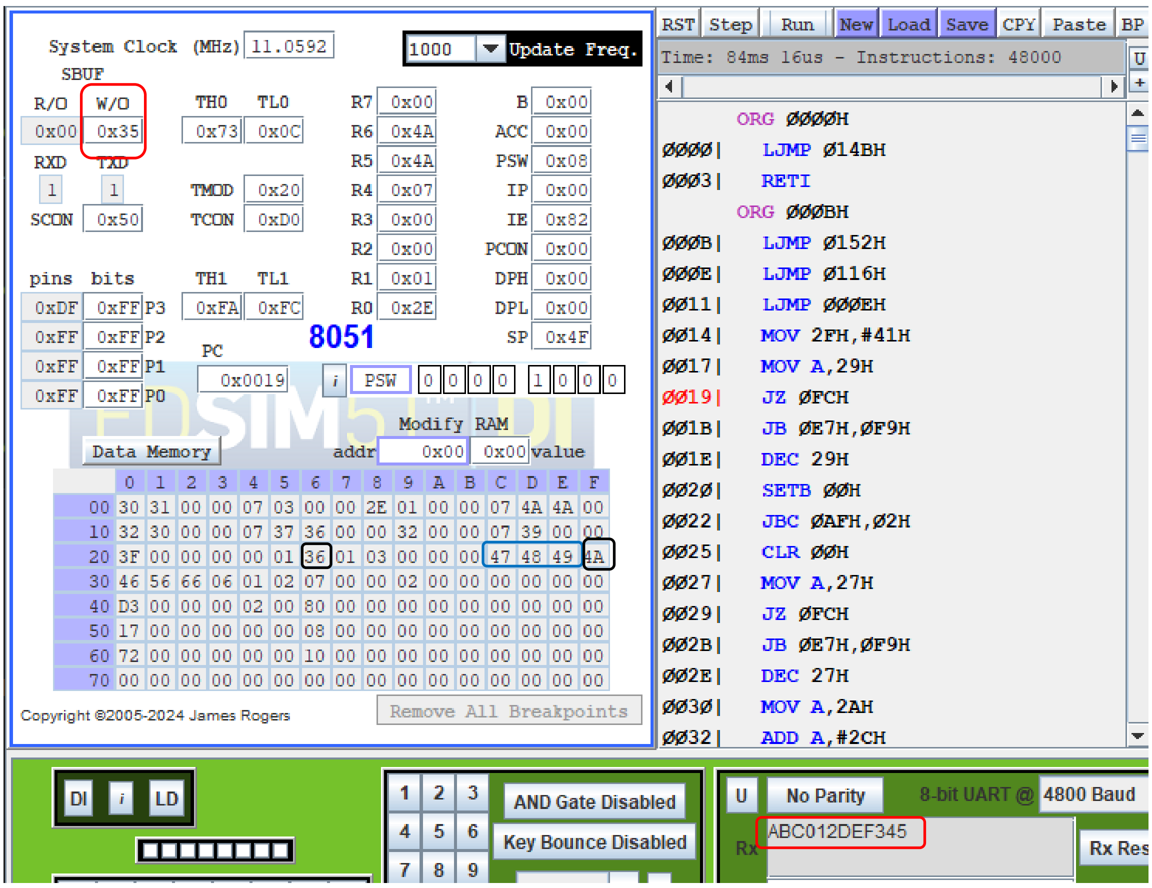
```
do
{
    if (currentThread==0) {
        if (producer==1) {
            currentThread = 1;          // Spawn Producer 1
        }
        else if (producer==2){
            currentThread = 2;          // Spawn Producer 2
        }
    }
    else {
        currentThread = 0;            // Spawn Consumer
        if (producer==1) {
            producer = 2;             // Switch Producer 2 in next execution
        }
        else if (producer==2){
            producer = 1;             // Switch Producer 1 in next execution
        }
    }
```

Fig 8: section of myTimer0Handler indicating modified part

Variable name “producer” is used to indicate which producer is been triggered to spawned, and Thread is triggered accordingly. In sequence one producer spawned by ISR then consumer is operated to display at UART after which another producer and again the consumer. Here we need to keep the track producer which was used before in order to switch to other producer to keep the alternative order. For which variable “producer” is switched between “1” and “2” indicating the Producer1 and Producer2 respectively. When consumer is writing the character to UART, the variable “producer” is switched to other (i.e. “2” if previously “1” and vice versa). This make sure each producer is spawned in next execution.

Here one producer is taking all the sharedBuffer space (i.e. 3) till consumer used it display at UART later all of sharedBuffer space is updated by other producer. Which means the display of character is 3 characters from one producer and then switched to 3 characters from other producer and repeats in alternate fashion. Fig. 9 shows the results of this approach during Producer1 in run and Fig. 10 shows the same but during Producer2 in run.

Figure 11 shows the sequence of consumer in running to transfer characters from sharedBuffer (0x2C to 0x2E) to SBUF and to UART with corresponding changes in semaphore at memory locations 0x27, 0x28 and 0x29 (highlighted in green) indicating mutex, full and empty respectively.



```
System Clock (MHz) 11.0592        1000   Update Freq.
SBUF
R/O   W/O     TH0   TL0     R7 0x00     B    0x00
0x00  0x35    0x73  0x0C    R6 0x4A     ACC  0x00
RXD   TXD                   R5 0x4A     PSW  0x08
1     1       TMOD  0x20    R4 0x07     IP   0x00
SCON  0x50    TCON  0xD0    R3 0x00     IE   0x82
                            R2 0x00     PCON 0x00
pins  bits    TH1   TL1     R1 0x01     DPH  0x00
0xDF  0xFF P3 0xFA  0xFC    R0 0x2E     DPL  0x00
0xFF  0xFF P2               8051        SP   0x4F
0xFF  0xFF P1   PC
0xFF  0xFF P0   0x0019   i  PSW  0 0 0 0  1 0 0 0
                              Modify RAM
Data Memory                 addr 0x00  0x00 value
```

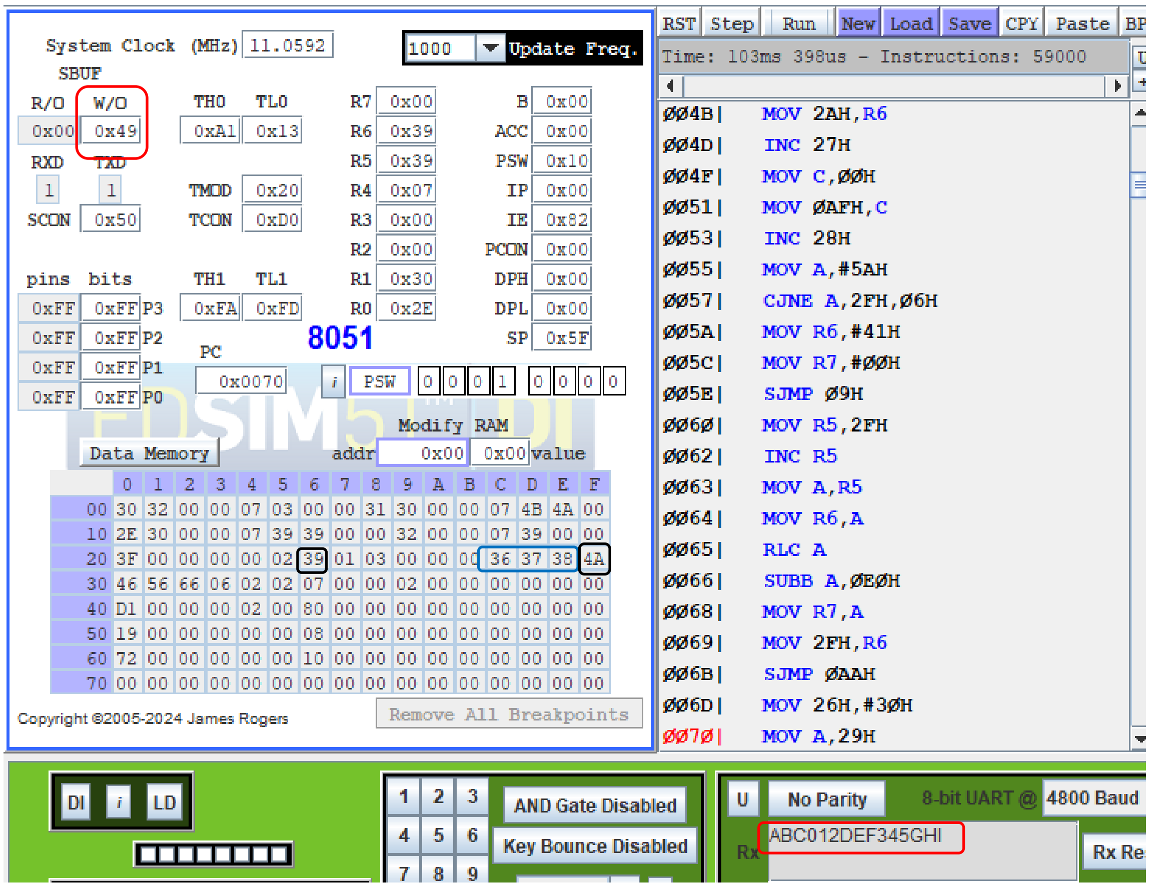
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	30	31	00	00	07	03	00	00	2E	01	00	00	07	4A	4A	00
10	32	30	00	00	07	37	36	00	00	32	00	00	07	39	00	00
20	3F	00	00	00	00	01	36	01	03	00	00	00	47	48	49	4A
30	46	56	66	06	01	02	07	00	00	02	00	00	00	00	00	00
40	D3	00	00	00	02	00	80	00	00	00	00	00	00	00	00	00
50	17	00	00	00	00	00	08	00	00	00	00	00	00	00	00	00
60	72	00	00	00	00	00	10	00	00	00	00	00	00	00	00	00
70	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

Copyright ©2005-2024 James Rogers

```
Time: 84ms 16us - Instructions: 48000

      ORG ØØØØH
ØØØØ|  LJMP Ø14BH
ØØØ3|  RETI
      ORG ØØØBH
ØØØB|  LJMP Ø152H
ØØØE|  LJMP Ø116H
ØØ11|  LJMP ØØØEH
ØØ14|  MOV 2FH,#41H
ØØ17|  MOV A,29H
ØØ19|  JZ ØFCH
ØØ1B|  JB ØE7H,ØF9H
ØØ1E|  DEC 29H
ØØ2Ø|  SETB ØØH
ØØ22|  JBC ØAFH,Ø2H
ØØ25|  CLR ØØH
ØØ27|  MOV A,27H
ØØ29|  JZ ØFCH
ØØ2B|  JB ØE7H,ØF9H
ØØ2E|  DEC 27H
ØØ3Ø|  MOV A,2AH
ØØ32|  ADD A,#2CH
```

```
8-bit UART @ 4800 Baud
Rx ABC012DEF345
```

Fig 9: EdSim51 showing result for spawning using ISR, during Producer1 in run.



```
System Clock (MHz) 11.0592        1000   Update Freq.
SBUF
R/O   W/O     TH0   TL0     R7 0x00     B    0x00
0x00  0x49    0xA1  0x13    R6 0x39     ACC  0x00
RXD   TXD                   R5 0x39     PSW  0x10
1     1       TMOD  0x20    R4 0x07     IP   0x00
SCON  0x50    TCON  0xD0    R3 0x00     IE   0x82
                            R2 0x00     PCON 0x00
pins  bits    TH1   TL1     R1 0x30     DPH  0x00
0xFF  0xFF P3 0xFA  0xFD    R0 0x2E     DPL  0x00
0xFF  0xFF P2               8051        SP   0x5F
0xFF  0xFF P1   PC
0xFF  0xFF P0   0x0070   i  PSW  0 0 0 1  0 0 0 0
                              Modify RAM
Data Memory                 addr 0x00  0x00 value
```

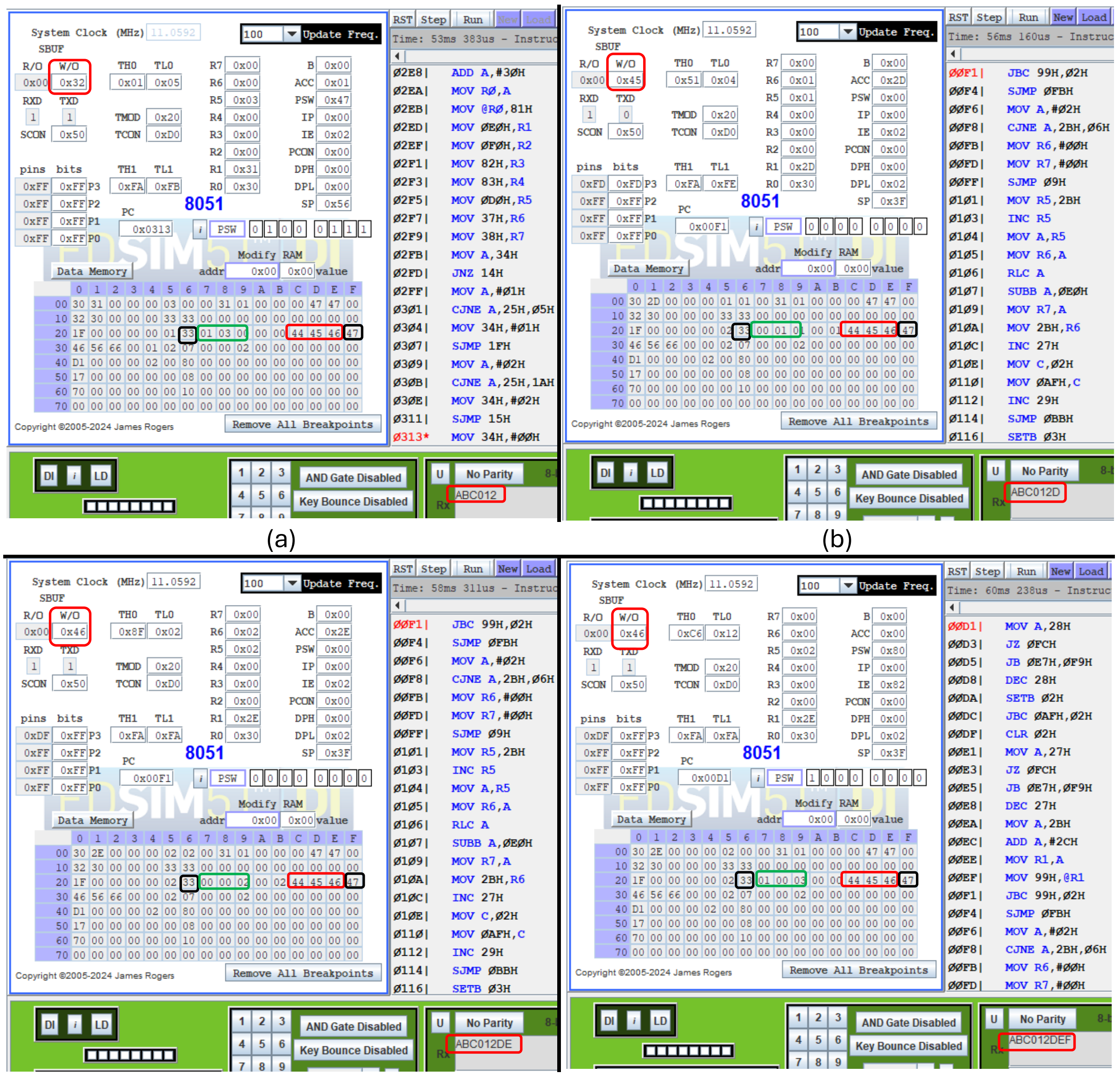
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	30	32	00	00	07	03	00	00	31	30	00	00	07	4B	4A	00
10	2E	30	00	00	07	39	39	00	00	32	00	00	07	39	00	00
20	3F	00	00	00	00	02	39	01	03	00	00	00	36	37	38	4A
30	46	56	66	06	02	02	07	00	00	02	00	00	00	00	00	00
40	D1	00	00	00	02	00	80	00	00	00	00	00	00	00	00	00
50	19	00	00	00	00	00	08	00	00	00	00	00	00	00	00	00
60	72	00	00	00	00	00	10	00	00	00	00	00	00	00	00	00
70	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

Copyright ©2005-2024 James Rogers

```
Time: 103ms 398us - Instructions: 59000

ØØ4B|  MOV 2AH,R6
ØØ4D|  INC 27H
ØØ4F|  MOV C,ØØH
ØØ51|  MOV ØAFH,C
ØØ53|  INC 28H
ØØ55|  MOV A,#5AH
ØØ57|  CJNE A,2FH,Ø6H
ØØ5A|  MOV R6,#41H
ØØ5C|  MOV R7,#ØØH
ØØ5E|  SJMP Ø9H
ØØ6Ø|  MOV R5,2FH
ØØ62|  INC R5
ØØ63|  MOV A,R5
ØØ64|  MOV R6,A
ØØ65|  RLC A
ØØ66|  SUBB A,ØEØH
ØØ68|  MOV R7,A
ØØ69|  MOV 2FH,R6
ØØ6B|  SJMP ØAAH
ØØ6D|  MOV 26H,#3ØH
ØØ7Ø|  MOV A,29H
```

```
8-bit UART @ 4800 Baud
Rx ABC012DEF345GHI
```

Fig 10: EdSim51 showing result for spawning using ISR, during Producer2 in run.



```
System Clock (MHz) 11.0592          100  Update Freq.
SBUF
R/O   W/O      TH0   TL0      R7 0x00      B 0x00
0x00  0x32     0x01  0x05     R6 0x00    ACC 0x01
RXD   TXD                     R5 0x03    PSW 0x47
1     1        TMOD  0x20     R4 0x00     IP 0x00
SCON  0x50     TCON  0xD0     R3 0x00     IE 0x02
                              R2 0x00   PCON 0x00
pins  bits     TH1   TL1      R1 0x31    DPH 0x00
0xFF  0xFF P3  0xFA  0xFB     R0 0x30    DPL 0x00
0xFF  0xFF P2                 8051        SP 0x56
0xFF  0xFF P1  PC
0xFF  0xFF P0  0x0313    i  PSW  0 1 0 0  0 1 1 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 30 31 00 00 00 03 00 00 31 01 00 00 00 47 47 00
10 32 30 00 00 00 33 33 00 00 00 00 00 00 00 00 00
20 1F 00 00 00 00 01 33 01 03 00 00 00 44 45 46 47
30 46 56 66 00 01 02 07 00 00 02 00 00 00 00 00 00
40 D1 00 00 00 02 00 80 00 00 00 00 00 00 00 00 00
50 17 00 00 00 00 00 08 00 00 00 00 00 00 00 00 00
60 70 00 00 00 00 00 10 00 00 00 00 00 00 00 00 00
70 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Copyright ©2005-2024 James Rogers      Remove All Breakpoints
```

```
RST  Step  Run  New  Load
Time: 53ms 383us - Instruc
Ø2E8|   ADD A,#3ØH
Ø2EA|   MOV RØ,A
Ø2EB|   MOV @RØ,81H
Ø2ED|   MOV ØEØH,R1
Ø2EF|   MOV ØFØH,R2
Ø2F1|   MOV 82H,R3
Ø2F3|   MOV 83H,R4
Ø2F5|   MOV ØDØH,R5
Ø2F7|   MOV 37H,R6
Ø2F9|   MOV 38H,R7
Ø2FB|   MOV A,34H
Ø2FD|   JNZ 14H
Ø2FF|   MOV A,#Ø1H
Ø3Ø1|   CJNE A,25H,Ø5H
Ø3Ø4|   MOV 34H,#Ø1H
Ø3Ø7|   SJMP 1FH
Ø3Ø9|   MOV A,#Ø2H
Ø3ØB|   CJNE A,25H,1AH
Ø3ØE|   MOV 34H,#Ø2H
Ø311|   SJMP 15H
Ø313*   MOV 34H,#ØØH
```

```
DI  i  LD      1 2 3   AND Gate Disabled    U  No Parity
               4 5 6   Key Bounce Disabled  Rx ABC012
```

(a)

```
System Clock (MHz) 11.0592          100  Update Freq.
SBUF
R/O   W/O      TH0   TL0      R7 0x00      B 0x00
0x00  0x45     0x51  0x04     R6 0x01    ACC 0x2D
RXD   TXD                     R5 0x01    PSW 0x00
1     0        TMOD  0x20     R4 0x00     IP 0x00
SCON  0x50     TCON  0xD0     R3 0x00     IE 0x02
                              R2 0x00   PCON 0x00
pins  bits     TH1   TL1      R1 0x2D    DPH 0x00
0xFD  0xFD P3  0xFA  0xFE     R0 0x30    DPL 0x02
0xFF  0xFF P2                 8051        SP 0x3F
0xFF  0xFF P1  PC
0xFF  0xFF P0  0x00F1    i  PSW  0 0 0 0  0 0 0 0
                         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 30 2D 00 00 00 01 01 00 31 01 00 00 00 47 47 00
10 32 30 00 00 00 33 33 00 00 00 00 00 00 00 00 00
20 1F 00 00 00 00 02 33 00 01 01 00 01 44 45 46 47
30 46 56 66 00 00 02 07 00 00 02 00 00 00 00 00 00
40 D1 00 00 00 02 00 80 00 00 00 00 00 00 00 00 00
50 17 00 00 00 00 00 08 00 00 00 00 00 00 00 00 00
60 70 00 00 00 00 00 10 00 00 00 00 00 00 00 00 00
70 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Copyright ©2005-2024 James Rogers      Remove All Breakpoints
```

```
RST  Step  Run  New  Load
Time: 56ms 160us - Instruc
ØØF1|   JBC 99H,Ø2H
ØØF4|   SJMP ØFBH
ØØF6|   MOV A,#Ø2H
ØØF8|   CJNE A,2BH,Ø6H
ØØFB|   MOV R6,#ØØH
ØØFD|   MOV R7,#ØØH
ØØFF|   SJMP Ø9H
Ø1Ø1|   MOV R5,2BH
Ø1Ø3|   INC R5
Ø1Ø4|   MOV A,R5
Ø1Ø5|   MOV R6,A
Ø1Ø6|   RLC A
Ø1Ø7|   SUBB A,ØEØH
Ø1Ø9|   MOV R7,A
Ø1ØA|   MOV 2BH,R6
Ø1ØC|   INC 27H
Ø1ØE|   MOV C,Ø2H
Ø11Ø|   MOV ØAFH,C
Ø112|   INC 29H
Ø114|   SJMP ØBBH
Ø116|   SETB Ø3H
```

```
DI  i  LD      1 2 3   AND Gate Disabled    U  No Parity
               4 5 6   Key Bounce Disabled  Rx ABC012D
               7 8 9
```

(b)

```
System Clock (MHz) 11.0592          100  Update Freq.
SBUF
R/O   W/O      TH0   TL0      R7 0x00      B 0x00
0x00  0x46     0x8F  0x02     R6 0x02    ACC 0x2E
RXD   TXD                     R5 0x02    PSW 0x00
1     1        TMOD  0x20     R4 0x00     IP 0x00
SCON  0x50     TCON  0xD0     R3 0x00     IE 0x02
                              R2 0x00   PCON 0x00
pins  bits     TH1   TL1      R1 0x2E    DPH 0x00
0xDF  0xFF P3  0xFA  0xFA     R0 0x30    DPL 0x02
0xFF  0xFF P2                 8051        SP 0x3F
0xFF  0xFF P1  PC
0xFF  0xFF P0  0x00F1    i  PSW  0 0 0 0  0 0 0 0
                         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 30 2E 00 00 00 02 02 00 31 01 00 00 00 47 47 00
10 32 30 00 00 00 33 33 00 00 00 00 00 00 00 00 00
20 1F 00 00 00 00 02 33 00 00 02 00 02 44 45 46 47
30 46 56 66 00 00 02 07 00 00 02 00 00 00 00 00 00
40 D1 00 00 00 02 00 80 00 00 00 00 00 00 00 00 00
50 17 00 00 00 00 00 08 00 00 00 00 00 00 00 00 00
60 70 00 00 00 00 00 10 00 00 00 00 00 00 00 00 00
70 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Copyright ©2005-2024 James Rogers      Remove All Breakpoints
```

```
RST  Step  Run  New  Load
Time: 58ms 311us - Instruc
ØØF1|   JBC 99H,Ø2H
ØØF4|   SJMP ØFBH
ØØF6|   MOV A,#Ø2H
ØØF8|   CJNE A,2BH,Ø6H
ØØFB|   MOV R6,#ØØH
ØØFD|   MOV R7,#ØØH
ØØFF|   SJMP Ø9H
Ø1Ø1|   MOV R5,2BH
Ø1Ø3|   INC R5
Ø1Ø4|   MOV A,R5
Ø1Ø5|   MOV R6,A
Ø1Ø6|   RLC A
Ø1Ø7|   SUBB A,ØEØH
Ø1Ø9|   MOV R7,A
Ø1ØA|   MOV 2BH,R6
Ø1ØC|   INC 27H
Ø1ØE|   MOV C,Ø2H
Ø11Ø|   MOV ØAFH,C
Ø112|   INC 29H
Ø114|   SJMP ØBBH
Ø116|   SETB Ø3H
```

```
DI  i  LD      1 2 3   AND Gate Disabled    U  No Parity
               4 5 6   Key Bounce Disabled  Rx ABC012DE
               7 8 9
```

(c)

```
System Clock (MHz) 11.0592          100  Update Freq.
SBUF
R/O   W/O      TH0   TL0      R7 0x00      B 0x00
0x00  0x46     0xC6  0x12     R6 0x00    ACC 0x00
RXD   TXD                     R5 0x02    PSW 0x80
1     1        TMOD  0x20     R4 0x00     IP 0x00
SCON  0x50     TCON  0xD0     R3 0x00     IE 0x82
                              R2 0x00   PCON 0x00
pins  bits     TH1   TL1      R1 0x2E    DPH 0x00
0xDF  0xFF P3  0xFA  0xFA     R0 0x30    DPL 0x02
0xFF  0xFF P2                 8051        SP 0x3F
0xFF  0xFF P1  PC
0xFF  0xFF P0  0x00D1    i  PSW  1 0 0 0  0 0 0 0
                         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 30 2E 00 00 00 02 00 00 31 01 00 00 00 47 47 00
10 32 30 00 00 00 33 33 00 00 00 00 00 00 00 00 00
20 1F 00 00 00 00 02 33 01 00 03 00 00 44 45 46 47
30 46 56 66 00 00 02 07 00 00 02 00 00 00 00 00 00
40 D1 00 00 00 02 00 80 00 00 00 00 00 00 00 00 00
50 17 00 00 00 00 00 08 00 00 00 00 00 00 00 00 00
60 70 00 00 00 00 00 10 00 00 00 00 00 00 00 00 00
70 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Copyright ©2005-2024 James Rogers      Remove All Breakpoints
```

```
RST  Step  Run  New  Load
Time: 60ms 238us - Instruc
ØØD1|   MOV A,28H
ØØD3|   JZ ØFCH
ØØD5|   JB ØE7H,ØF9H
ØØD8|   DEC 28H
ØØDA|   SETB Ø2H
ØØDC|   JBC ØAFH,Ø2H
ØØDF|   CLR Ø2H
ØØE1|   MOV A,27H
ØØE3|   JZ ØFCH
ØØE5|   JB ØE7H,ØF9H
ØØE8|   DEC 27H
ØØEA|   MOV A,2BH
ØØEC|   ADD A,#2CH
ØØEE|   MOV R1,A
ØØEF|   MOV 99H,@R1
ØØF1|   JBC 99H,Ø2H
ØØF4|   SJMP ØFBH
ØØF6|   MOV A,#Ø2H
ØØF8|   CJNE A,2BH,Ø6H
ØØFB|   MOV R6,#ØØH
ØØFD|   MOV R7,#ØØH
```

```
DI  i  LD      1 2 3   AND Gate Disabled    U  No Parity
               4 5 6   Key Bounce Disabled  Rx ABC012DEF
               7 8 9
```

(d)

Fig. 11: Consumer in run in order from (a) to (d) to transfer “D”, “E”, “F” to UART.