

Programming Project Checkpoint 4 Report

Notes

I also updated myTimer0Handler in preemptive.c to handle the unfairness of the output. I hope that is allowed.

Fairness

Yes, with the previous round robin scheduling policy, I can only see the result from one of the producers, changing the order also results in value from the other producer. The Producer1 and Producer2 should have equal chances to output their value to the Consumer. At some time yes, because the SBUF didn't write anything, but now it works. I solved this unfairness problem by tweaking the timer handler, so that it can switch to the correct threads which is Consumer - Producer1 - Consumer - Producer 2 - Consumer and so on.

Typescript for Compiling

```

sdcc -o test3threads.hex test3threads.c preemptive.rel
wilbertallen@MacBook-Air-2016 ppc4 % make clean
rm *.hex *.ihx *.lnk *.lst *.map *.mem *.rel *.rst *.sym
rm: *.ihx: No such file or directory
rm: *.lnk: No such file or directory
make: *** [clean] Error 1
wilbertallen@MacBook-Air-2016 ppc4 % make
sdcc -c test3threads.c
sdcc -c preemptive.c
preemptive.c:223: warning 85: in function ThreadCreate unreferenced function argument : 'fp'
preemptive.c:288: warning 283: function declarator with no prototype
sdcc -o test3threads.hex test3threads.rel preemptive.rel
wilbertallen@MacBook-Air-2016 ppc4 %

```

Fig.1 Typescript for compiling using the given makefile

Producer1 & Producer2 is Running and Changing Semaphore

	Value	Global	Global Defined In Module
C:	00000014	_Producer1	test3threads
C:	00000075	_Producer2	test3threads
C:	000000D6	_Consumer	test3threads
C:	00000129	_main	test3threads
C:	0000015C	__sdcc_gsinit_startup	test3threads
C:	00000160	__mcs51_genRAMCLEAR	test3threads
C:	00000161	__mcs51_genXINIT	test3threads
C:	00000162	__mcs51_genXRAMCLEAR	test3threads
C:	00000163	_timer0_ISR	test3threads
C:	00000167	_Bootstrap	preemptive
C:	00000194	_ThreadCreate	preemptive
C:	00000257	_ThreadYield	preemptive
C:	000002D5	_ThreadExit	preemptive
C:	0000031C	_myTimer0Handler	preemptive
C:	000003A0	__moduint	_moduint
C:	000003ED	__modsint	_modsint

ASxxxx Linker V03.00/V05.40 + sld, page 14.

Fig.2.1 Function & Variable Addresses in .map File

We can see that the Producer1 & Producer2 are running interchangeably by observing their addresses in the assembly.

```

0147| MOV DPTR, #0075H
014A| LCALL 0194H
014D| MOV DPTR, #0014H
0150| LCALL 0194H

```

Fig.2.2 ThreadCreate(Producer2) and ThreadCreate(Producer1)

```

0017*| MOV A, 4DH
0019| JZ 0FCH
001B| JB 0E7H, 0F9H
001E| DEC 4DH
0020| MOV A, 4EH
0022| JZ 0FCH
0024| JB 0E7H, 0F9H
0027| DEC 4EH

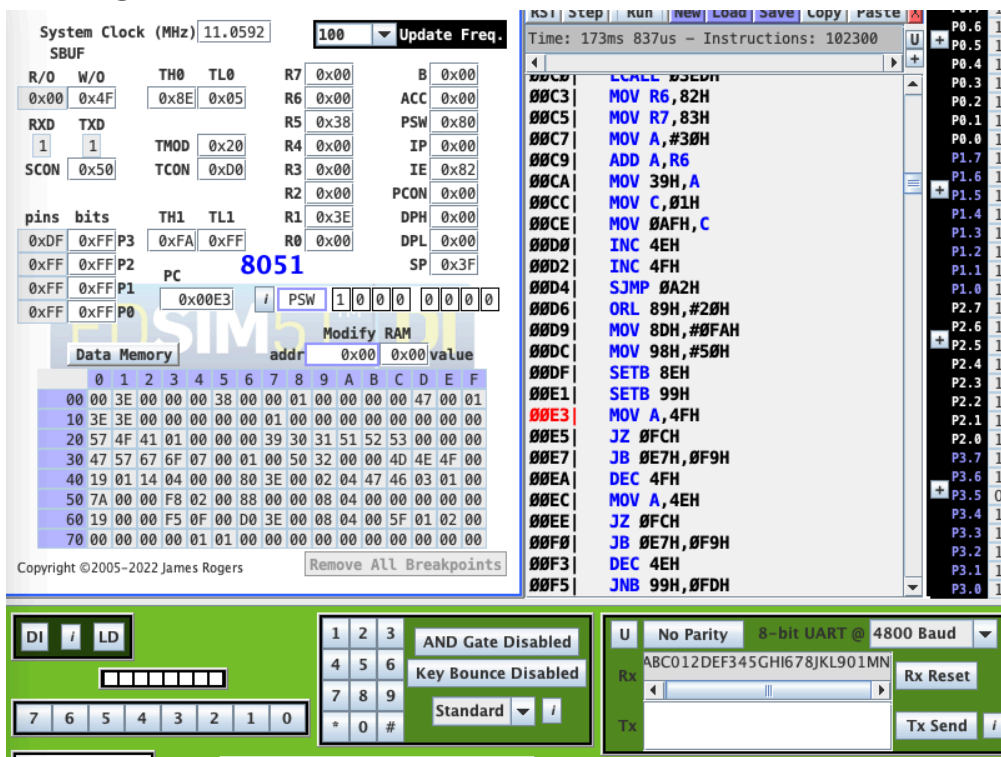
```

Fig 2.3 Producers Calling SemaphoreWait(empty) and SemaphoreWait(mutex)

By figure 2.3, we can observe that the producers call the SemaphoreWait(empty) and SemaphoreWait(mutex). This proves that the Producers and Consumers are communicating through the semaphores, to ensure mutual exclusion so that no one accesses the same variable continuously.

We can also notice that Producer1 and Producer2 are running by observing the *az* and *onine* which change interchangeably according to the semaphores. Video can be seen [here](#).

Running Consumer



The screenshot shows the Proteus 8.09 SP5 IDE. The main window displays the assembly code for the consumer thread. The code includes instructions for moving data to registers, jumping, and decrementing. The I/O window shows the output of the consumer thread, which is 'ABC012DEF345GHI678JKL901MN'.

Figure 3

With similar logic, we can prove that the semaphore empty, mutex, and full is changing to accommodate the communication between Producers and Consumers. We can then further prove that the Consumer is

running by observing the value of SBUF, which in this case is 0x4F ‘O’, being written to the received_data. It’s currently only showing ‘M’ in the received data, but it means that SBUF is being written.

Outputs

The screenshot displays the Proteus ISIS simulation environment. The top window shows the assembly code for an AVR microcontroller. The code includes instructions such as MOV, JZ, JB, DEC, SETB, JBC, CLR, and LCALL. The bottom window shows the UART terminal output, which displays the received data '012345678901'.

Assembly Code Window:

```

0017| MOV A,4DH
0019| JZ 0FCH
001B| JB 0E7H,0F9H
001E| DEC 4DH
0020| MOV A,4EH
0022| JZ 0FCH
0024| JB 0E7H,0F9H
0027| DEC 4EH
0029| SETB 00H
002B| JBC 0AFH,02H
002E| CLR 00H
0030| MOV A,3AH
0032| ADD A,#3CH
0034| MOV R0,A
0035| MOV @R0,38H
0037| MOV R6,3AH
0039| MOV R7,#00H
003B| MOV 82H,R6
003D| MOV 83H,R7
003F| INC DPTR
0040| MOV 08H,#03H
0043| MOV 09H,R7
0045| LCALL 03E1H
0048| MOV R6,82H
004A| MOV 2AH,R6
  
```

UART Terminal Window:

```

U No Parity 8-bit UART @ 4800 Baud
Rx 012345678901
Tx
  
```

Fig. 4.1 Unfair output

This is the output of the unfair version. By changing the order to CreateThread(Producer2) first, the output will always come from Producer 2.

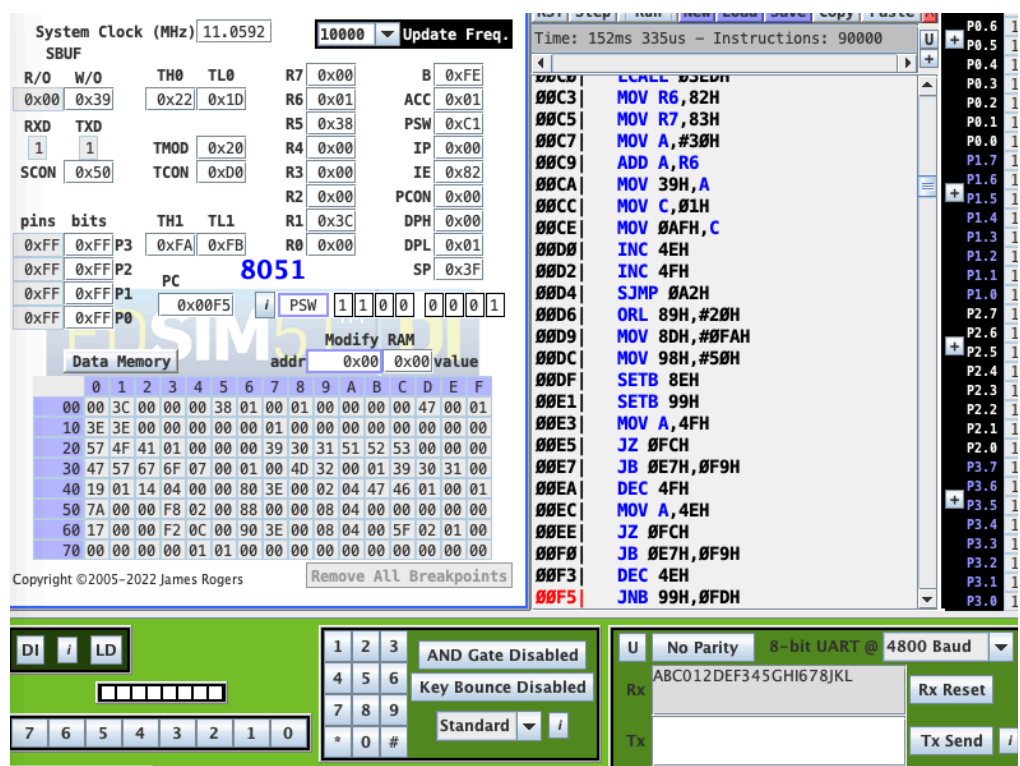


Fig.4.2 Fair Output

This is the output of the fair version, where each producer takes turns to feed the value into the consumer.