

PPC2 Report 110062120 高小榛

1. Code explanation

(1) testpreempt.c

First, I create 4 global variable Cur_ID, Word, Buffer and Buffer_full. The meaning of these variable is define in this picture.

```
_data __at (0x35) ThreadID Cur_ID;
_data __at (0x3D) char Word; // The character currently being produced
_data __at (0x3E) char Buffer; // Shared buffer between producer and consumer
_data __at (0x3F) int Buffer_full; // Flag indicating if Buffer is full
```

In the Producer function, I assign first word produce with 'A', then in the while loop, I use the buffer_full to check if the buffer is available or not. If the buffer is full, that means consumer no consume, if the buffer isn't full, that means buffer has consumed, so we can write new data into the buffer. Different with ppc1, I didn't call the ThreadYield.

In Consumer function, first I initialize Tx for polling using the information in class. , change the TMOD to TMOD |= 0x20. Then we should wait for new data from producer, write data to serial port Tx, poll for Tx to finish writing (TI), then finally clear the flag. So I check if the buffer_full is 0, that means producer no produce, if the buffer isn't 0, that means producer has produce something, so we can do the thing I mention before, also do not use ThreadYield which is different from ppc1.

In main function, first I initialize variable buffer_full = 0, buffer = empty character, and use ThreadCreate function to create producer thread and store with cur_id. Finally, use assembly to set up stack pointer, call consumer function.

Additionally, I add timer0_ISR to let ISR call my routine timer.

(2) preemptive.c

First, I create the static global SP0 ~ SP3, Map, Cur_ID, SP_old, SP_new and New_ID. Then I define 2 macro save state and restore state in a similar way. In savestate we should saving the current thread context. So first push ACC, B register, Data pointer registers (DPL, DPH), PSW with assembly. Then save SP into the saved Stack Pointers array with if-else block to check which place we should store the SP into it. In restorestate, I first assign SP to the saved SP from the saved stack pointer array, then pop the registers PSW, data pointer registers, B reg, and ACC.

In bootstrap function, I clear thread bitmap indicating no threads are running and initialize stack pointer. I initialize timer and interrupts for preemption with set TMOD = 0, IE = 0x82 to enable timer 0 interrupt, and set TR0 = 1 to start running timer0. Then I create a thread for main and store with Cur_ID and restore the content to run main function.

In myTimer0Handler function, first I use EA=0 to disable interrupts, then save the current thread state. Next I cycle through thread IDs to find the next valid thread and break out of the loop if a valid thread is found. Finally restore the state, re-enable interrupts then return.

In ThreadCreate function, first disable interrupt, check to see we have not reached the max #threads and return -1 if no available thread ID. Then initialize a New_ID, run a if-else block to check each thread ID (0 to 3) to find an available one. For each thread ID, perform bitwise AND with Map and check if the result is 0. If 0, it means the thread ID is available for use. If Thread i is available: first set New_ID to ‘i’, then update Map to indicate Thread i is now in use, finally set SP_new to the starting stack location for Thread i. Then save the current SP in SP_old, set SP to the new thread's starting stack location. Then push the return address (fp) onto the stack for the new thread. Then initialize the registers to 0 for the new thread by set A to 0 then push into each register. Then set up the PSW for the new thread and save its SP, and PSW setup depends on the thread ID to use the correct register bank. Finally restore the original SP from SP_old, re-enable interrupt, and return new_ID.

In ThreadYield function, I use Round-Robin to find the next thread to run. First cycle through thread IDs and select the next valid thread, then break out of the loop if a valid thread is found, finally restore state.

In ThreadExit function, I clear the current thread from the thread bitmap and set up for context switch, then modify the thread bitmap to indicate the current thread is no longer valid. After dealing with Map, also cycle through thread IDs and select the next valid thread, then break out of the loop if a valid thread is found, finally restore state.

(3) preemptive.h

Given by the document of ppc1.

2. Typescript

(1) Makefile

```
CC = sdcc
CFLAGS = -c
LDFLAGS =
#--stack-after-data --stack-loc 0x39 --data-loc 0x20

C_OBJECTS = testpreempt.rel preemptive.rel

all: testpreempt.hex

testpreempt.hex: $(C_OBJECTS) $(ASM_OBJECTS)
    $(CC) $(LDFLAGS) -o testpreempt.hex $(C_OBJECTS)

clean:
    rm *.hex *.ihx *.lnk *.lst *.map *.mem *.rel *.rst *.sym

%.rel: %.c    preemptive.h Makefile
    $(CC) $(CFLAGS) $<
```

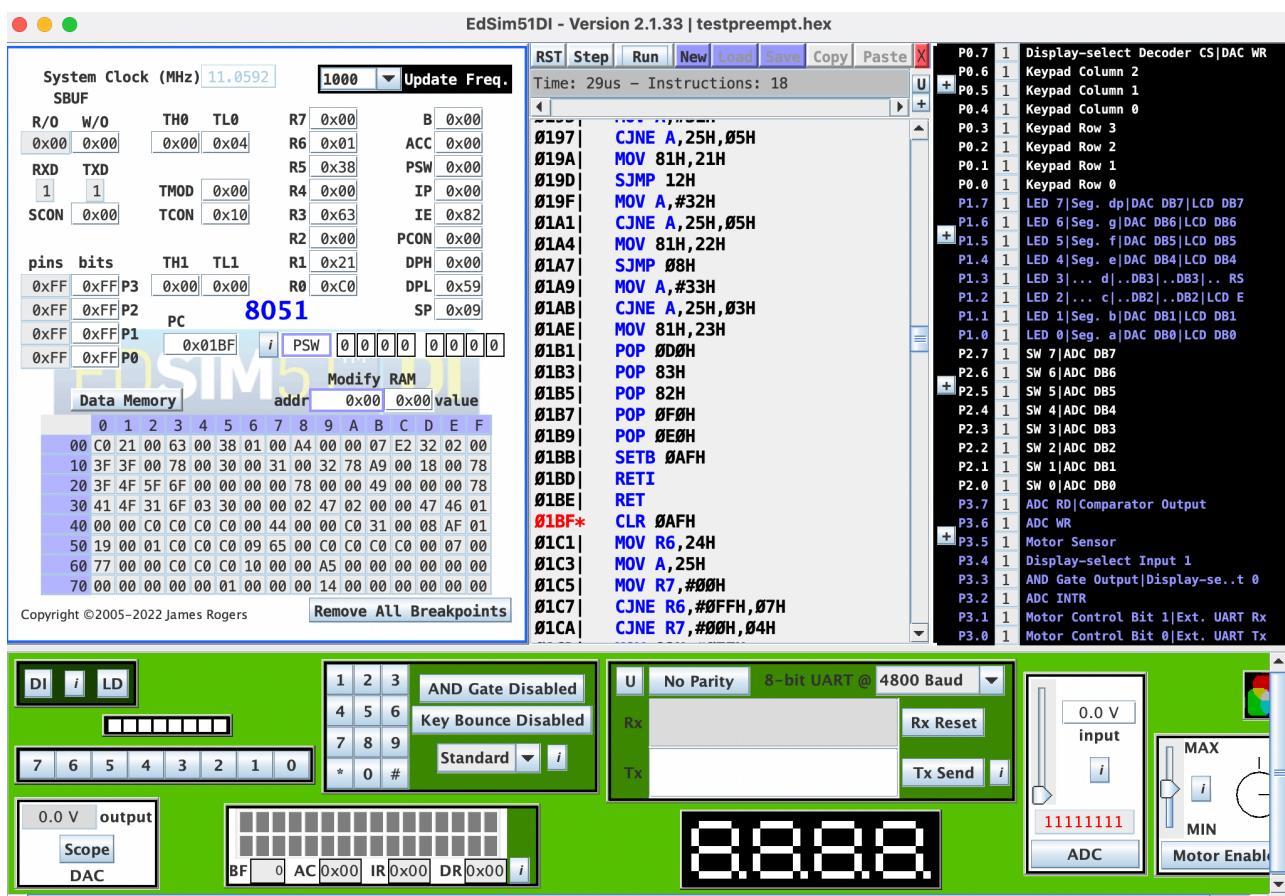
(2) Screenshot

```
sunny@xiaozhendeAir: ~ % cd 110062120-ppc2
sunny@xiaozhendeAir: ~ % make
sdcc -c testpreempt.c
sdcc -c preemptive.c
preemptive.c:281: warning 85: in function ThreadCreate unreferenced function argument : 'fp'
sdcc -o testpreempt.hex testpreempt.rel preemptive.rel
sunny@xiaozhendeAir: ~ % 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
```

3. Screenshots and explanation

(1) before ThreadCreate

ThreadCreate is in 01BF according to testcoop.map.



(2) Producer running

Producer is at 0014 according to map.

EdSim51DI - Version 2.1.33 | testpreempt.hex

System Clock (MHz) 11.0592 1000 Update Freq.

RST Step Run New Load Save Copy Paste

Executed 0x001D: MOV 3EH,3DH | Time: 8ms 99

0.7 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

ORG 0000H
0000| LJMP 0073H
0003| RETI
ORG 000BH
000B| LJMP 007AH
000E| LJMP 0059H
0011| LJMP 000EH
0014*| MOV 3DH,#41H
0017| MOV A,3FH
0019| ORL A,40H
001B| JNZ 0FAH
001D| MOV 3EH,3DH
0020| MOV A,#5AH
0022| CJNE A,3DH,05H
0025| MOV 3DH,#41H
0028| SJMP 05H
002A| MOV A,3DH
002C| INC A
002D| MOV 3DH,A
002F| MOV 3FH,#01H
0032| MOV 40H,#00H
0035| SJMP 0E0H
0037| ORL 89H,#20H
003A| MOV 8DH,#0FAH
003B| SJMP 0000H

Copyright ©2005-2022 James Rogers

Remove All Breakpoints

Digital I/O: DI, LD, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F
Analog I/O: 0.0 V input, 0.0 V output, Scope, DAC
UART: No Parity, 8-bit UART @ 4800 Baud, Rx, Tx, Rx Reset, Tx Send
ADC: MAX, MIN, Motor Enable
Motor Control: Motor Control Bit 1|Ext. UART Rx, Motor Control Bit 0|Ext. UART Tx

(3) Consumer running

Consumer is at 0037 according to map.

EdSim51DI - Version 2.1.33 | testpreempt.hex

System Clock (MHz) 11.0592 1000 Update Freq.

RST Step Run New Load Save Copy Paste

Executed 0x0044: MOV A,3FH | Time: 206us -

0.7 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

0020| MOV A,#5AH
0022| CJNE A,3DH,05H
0025| MOV 3DH,#41H
0028| SJMP 05H
002A| MOV A,3DH
002C| INC A
002D| MOV 3DH,A
002F| MOV 3FH,#01H
0032| MOV 40H,#00H
0035| SJMP 0E0H
0037*| ORL 89H,#20H
003A| MOV 8DH,#0FAH
003D| MOV 98H,#50H
0040| SETB 8EH
0042| SETB 99H
0044| MOV A,3FH
0046| ORL A,40H
0048| JZ 0FAH
004A| JNB 99H,0FDH
004D| MOV 99H,3EH
0050| CLR 99H
0052| CLR A
0053| MOV 3FH,A
0055| MOV 40H,A
0057| SJMP 0000H

Copyright ©2005-2022 James Rogers

Remove All Breakpoints

Digital I/O: DI, LD, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F
Analog I/O: 0.0 V input, 0.0 V output, Scope, DAC
UART: No Parity, 8-bit UART @ 4800 Baud, Rx, Tx, Rx Reset, Tx Send
ADC: MAX, MIN, Motor Enable
Motor Control: Motor Control Bit 1|Ext. UART Rx, Motor Control Bit 0|Ext. UART Tx

(4) Interrupt

_timer0_ISR is at 007A according to map.

