# 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) $<
```

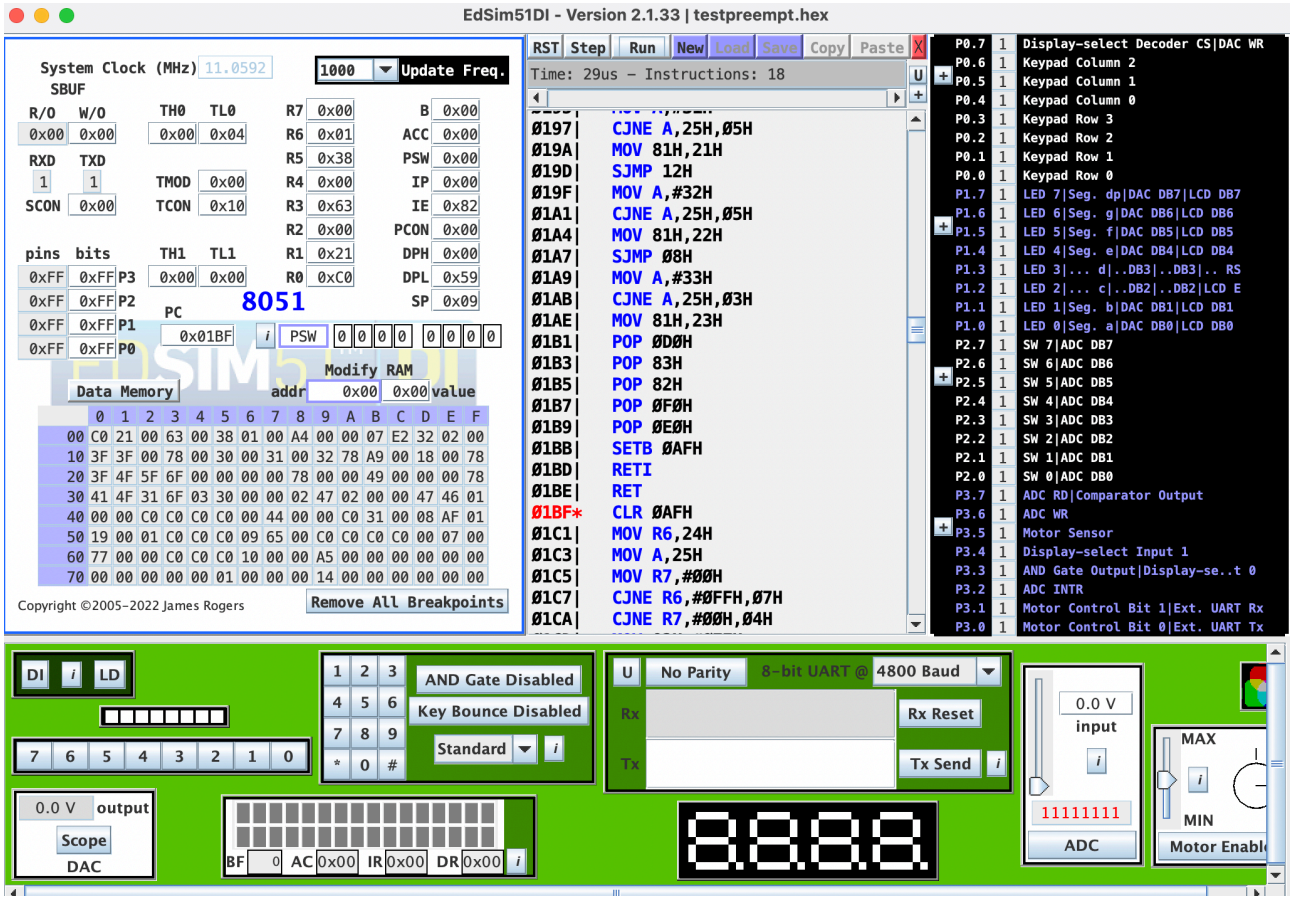
(2) Screenshot

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

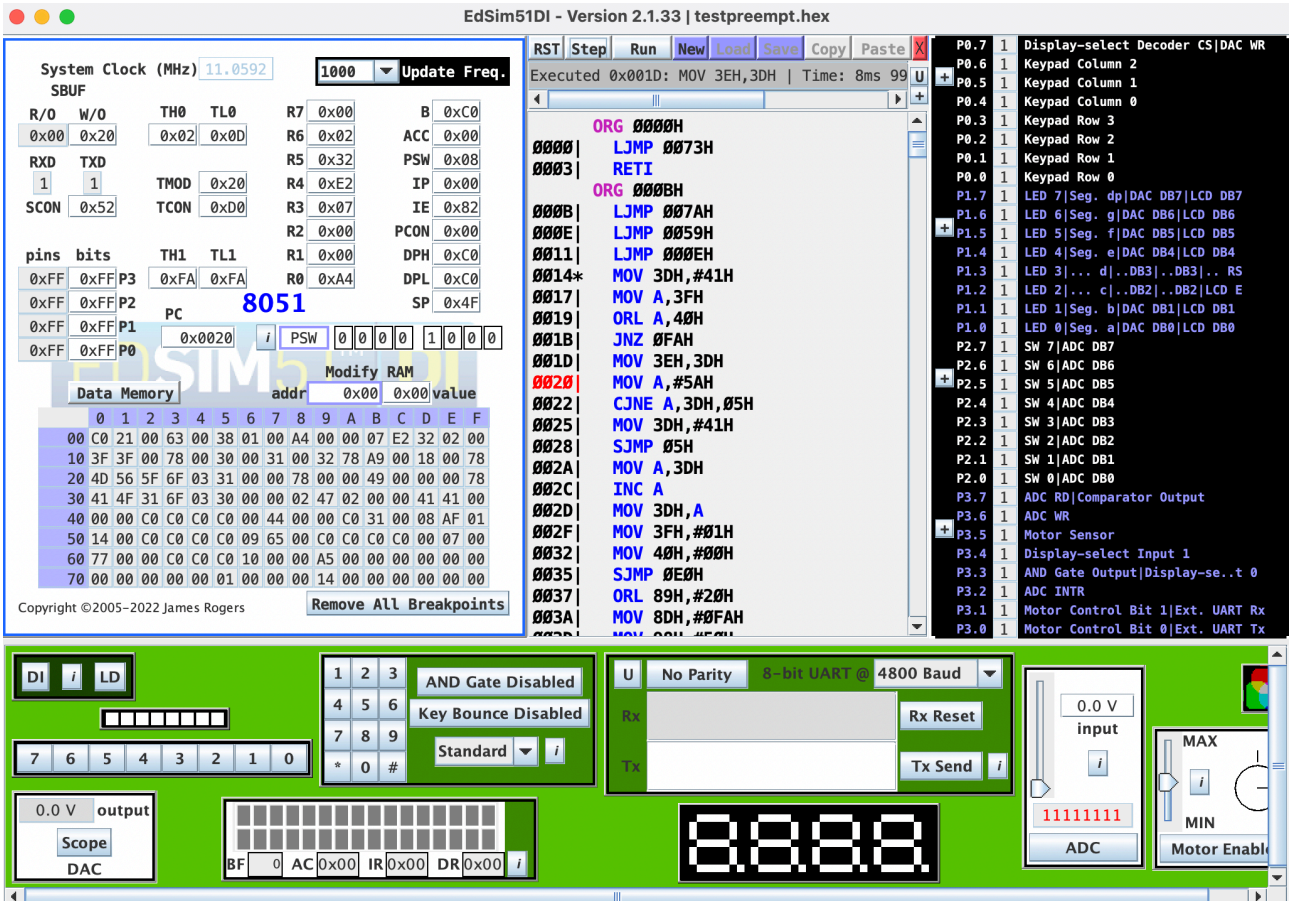
## (1) before ThreadCreate

ThreadCreate is in 01BF according to testcoop.map.



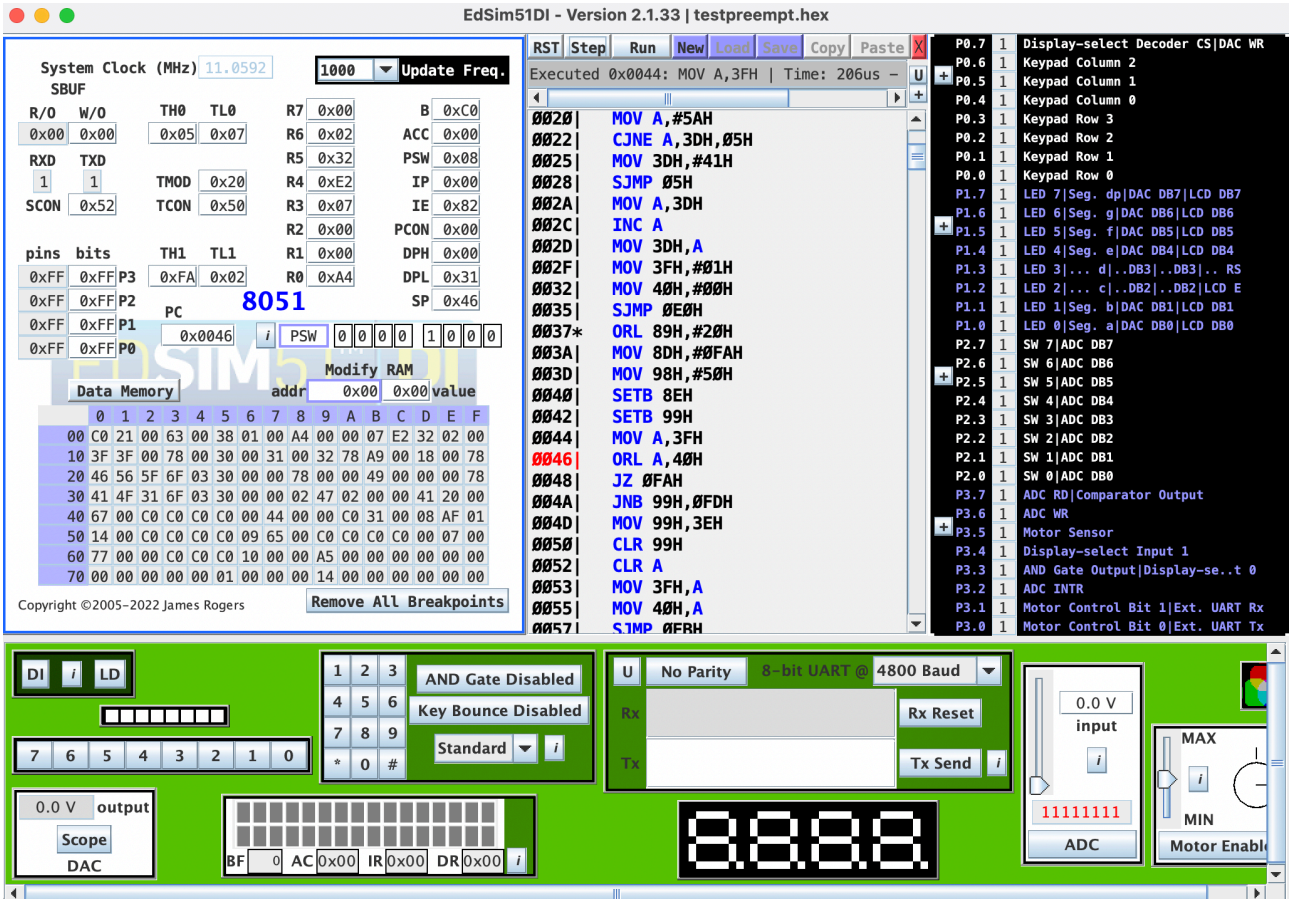
## (2) Producer running

Producer is at 0014 according to map.




## (3) Consumer running

Consumer is at 0037 according to map.



(4) Interrupt

_timer0_ISR is at 007A according to map.