[EE414] Lab4 Report

20180480, Woojin Lee

1. Purpose

In lab3, we implemented metronome using thread with “pthread.h”. In lab4, we are going to implement a metronome that works with signal interrupt, specifically SIGALRM.
Also, you have to know POSIX high resolution timers for this lab. To implement high accurate timers in Linux userspace, this POSIX timer can be a proper choice. You can create a timer with the timer_create() function, and manipulate them with various structures.

2. Experiment Sequence

First, you have to detect if your timer system supports high resolution. You can check them with kernel startup message
$ dmesg | grep resolution
or with proc file system (/proc/timer_list)
$ cat /proc/timer_list
In my case, I could get meaningful output with latter one.

As a second step, to be familiar with POSIX timer, you have to thoroughly dissect “timer_hrtimer.c” file. Note that you have to give -lrt option when you compile. By doing so, you can get hands-on experience with how to signal blocking and handling, many structs like itimerspec, timerspec. Also, you can see how timer_create, timer_settime, clock_gettime functions are used.

Then, you implement metronome_heart.c file to work lab3 metronome in signal handling manner. Here comes the required algorithm:

1. Init GPIO LED
    a. Init HR timer
        i. Create signal handler for HR timer - metronome processing
2. Init key processing
    a. Set termios
    b. Print title and menu
3. Set default values to parameters (TimeSig 3 (¾), Tempo 90, Stop)
    a. Print default values
4. Loop
    a. Get user input key without entering in blocking mode
    b. If ‘q’ == break
    c. else:
        i. ‘z’ -> Time signature
            1. Increase TimeSig
            2. If TimeSig >= 4 -> TimeSig = 1
        ii. ‘c’ -> Decrease Tempo by 5
            1. minimum tempo = 30
        iii. ‘b’ -> Increase Tempo by 5
            1. maximum tempo = 200

iv. ‘m’ -> Start or Stop
1. Start = 1 if stop. Else, start=0
2. Start/Stop HR Timer
d. Print single line message (Input & status: without linefeed)
5. Print quit message
6. Reset termios

3. Experimental Result

I attached a demo video.

I submitted 3 files:
tar file, pdf file, and mp4 file. mp4 file is my demo video, and tar file is my code. Since I splitted modules, I compressed multiple files into one file. I also implemented make, which you can easily use.

Here comes my code architecture:

metronome_hrt - signal handler (SIGALRM) - loop - timer
key_input_fu - init_termios() - reset_termios() - init_key() - key_hit() - getch() - exit_key()
gpio_led_fu - init_LED() - play_LED() - pause_LED() - exit_LED()

key_input_fu and gpio_led_fu are used as utility files, and metronome_hrt file is the main function. For example, when you get SIGALRM, you can turn on and off LED with play_LED() and pause_LED() functions. Also, inside the loop at metronome_hrt file, it indicates key hit with key_hit() and knows what key is pushed with getch(). By using timer, you can periodically turn on and off LED. You set the timer to proper period with tempo.

4. Discussion

- Brief explanation about POSIX timer.
functions in time.h

int **timer_create**(clockid_t clockid, struct sigevent *restrict sevp, timer_t *restrict

timerid);

As the name implies, this function creates new timer with timerid. We are going to use clockid CLOCK_MONOTONIC. This setting is unconfigurable and measures monotonically increased time from specific past time. This past time is constant.

int **timer_settime**(timer_t timerid, int flags, const struct itimerspec *new_value, struct ittimerspec *old_value);

This function starts or stops the timer with a specified timerid. You can set timer properties in new_value. In error case, it returns -1 and sets error value to errno. There are two errnos. “EFAULT” and “EINVAL”. “EFAULT” means new_value or old_value is invalid pointer. “EINVAL” means invalid timerid or time value is negative or too large.

There are many other functions like clock_gettime(), but I won’t explain here since I didn’t use it for my metronome_heart.c

structures in time.h

```
struct timespec
{
  time_t tv_sec; // 초값
  long tv_nsec; // 나노초값
}

struct itimerspec
{
  struct timespec it_interval; // 타이머 주기
  struct timespec it_value;    // 첫 만기 시점
}
```

timespec is for timer settings. There is tv_milisec component, too.

In struct itimerspec, **it_value** sets first expiry time, and **it_interval** is reloading time.
If it_value has 0 value, timer is cleared off. If it_value is not 0, the timer will start, and will expire at time at it_value. At this time, timer will be reloaded to value in it_interval. If it_interval is 0, it means that timer is disposable one.
We can use these struct in a useful manner since the metronome works in a periodic way.

- Search timers on AM3359 and summarize features.
You can get precise information at AM335x ARM Cortex A8 Microprocessors(MPU) Technical Reference Manual, Ch.20(Timers). There are 4 types of timers in AM335x, DMTimer, DMTimer 1ms, RTC_SS, and WATCHDOG.

1. DMTimer / DMTimer 1ms
   Both are almost identical. These are for free running upward counters with auto reload capability on overflow. The timer counter value can be read and written in real-time. This timer includes compare logic so that you can interrupt events at the time you want. These timers support 3 functional modes: Timer mode/Capture mode/Compare mode. By default, after core reset, capture mode and compare modes are disabled.

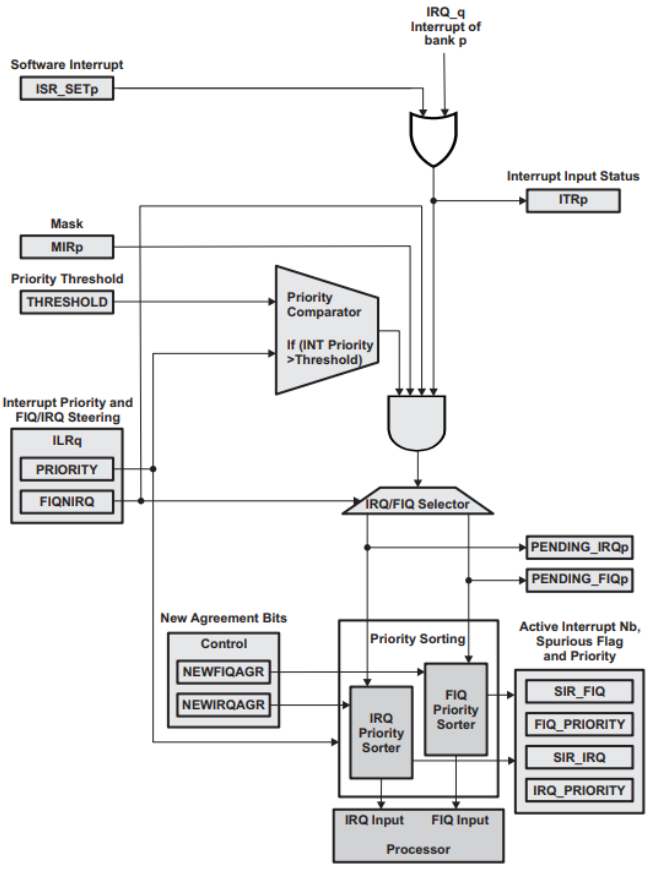
Difference between DMTimer and DMTimer 1ms is that DMTimer 1ms generate 1 ms tick with 32768-Hz functional clock. This is for minimizing the error between a true 1ms tick and the tick generated by the 32768Hz timer. Additional block, 1ms block is used to correct this error.

2. RTC_SS
   This timer is used to generate interrupts on intervals specified by the user. The basic purpose for RTC_SS is keeping time of the day. RTC timer is also used for Digital Rights management. The final purpose of RTC is to wake the rest of the chip up from a power down state. I think we can utilize this timer for our lab.

3. WATCHDOG
   The watchdog timer is an upward counter capable of generating a pulse on the reset pin and an interrupt to the device system modules following an overflow condition(abnormal, infinite loop, etc...). The default state of the watchdog timer is enabled and not running. Watchdog timer periodically inspect hardware. If you don't get kick from the device for certain time, it regards the system error and reset it.

- Search how the interrupt on AM3359-interrupt request, Acknowledge, Priority, NMI - can be handled in kernel space.

Figure 6-1. Interrupt Controller Block Diagram



This figure which is from the AM335x reference manual illustrates how the interrupt controller works.

Interrupts from inside/outside of MCU are handled at NVIC(Nested Vectored Interrupt Controller). Each interrupt has their own priority. NMI(Non-maskable Interrupt) has relatively high priority. Priorities are managed at IPR register as Preemption Priority and Sub Priority. Preemption priority is applied for inter-ISR preemption, and sub priority determines priority of pended ISR operation order. IPR register can be configured to 5 Priority groups, and the number of preemption priority and sub priority is modified according to this configuration.

When the device requests interrupt, the interrupt controller will generate IRQ, and IRQ Exception will change IRQ mode and branch into IRQ Exception Vector. Following key-pair in IRQ Exception Vector Table, it will branch into IRQ handler and conduct prologue(saving register set. CPSR->SPSR). When you get the interrupt number, you will call the corresponding ISR(Interrupt Service Routine). After handling interrupt, it will return from exception after epilogue(restoring stored register set. SPSR->CPSR).

- Difference between thread and signal handler

They are totally different concepts. Threads execute concurrently, while signals interrupt(works exclusively). Signal handler is event-driven programming and can be used to influence threads. Also, signals can be used in single-thread programs.

5. References

https://tttsss77.tistory.com/205

https://www.ic.unicamp.br/~celio/mc404-2013/arm-manuals/ARM_exception_slides.pdf

https://m.blog.naver.com/PostView.nhn?blogId=eziya76&logNo=221428695204&proxyReferer=https:%2F%2Fwww.google.com%2F

AM335x ARM Cortex-A8 reference manual