Embedded Systems
Lab4. Interrupt

Due date
Report : 11:59:59 PM, May 21, Friday

1. Purpose

Understand how to program the interrupt and timer via signal handler on the AM3359 processor in Beaglebone with Linux.

2. Problem Statement

-Interrupt signal handler for high-resolution timer-

Make an application program named “metronome_hrt” with interrupt from high-resolution timer.

The main program waits for the user input (tempo, time-signature, start/stop) with menus given in the Lab 3. When the run state become 1, it should arm the high resolution timer to activate the signal handler. When the run state become 0, it should dis-arm the high-resolution timer and deactivate the signal handler.

3. Technical Backgrounds

A. Hardware Connection

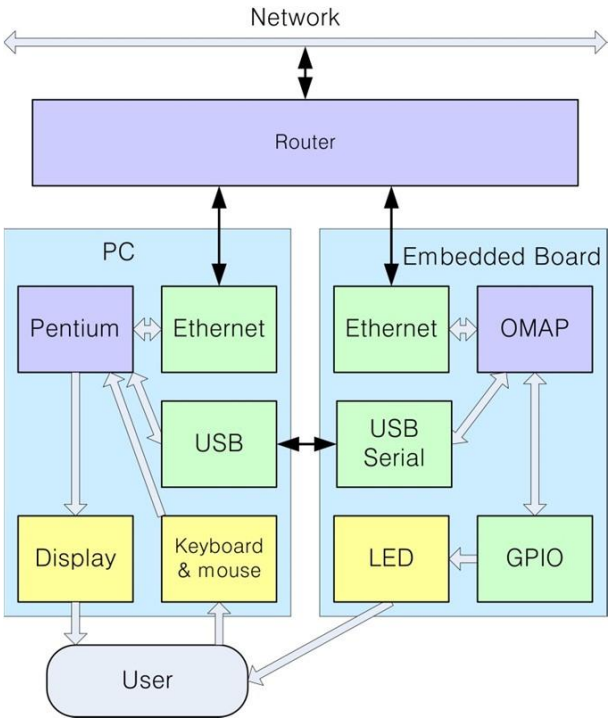

Fig. 1 Block Diagram for Lab 4

B. High Resolution Timers

(Link: http://elinux.org/High_Resolution_Timers)

The objective of the high resolution timers project is to implement the POSIX 1003.1b Section 14(Clocks and Timers) API in Linux. This includes support for high resolution timers – that is, timers with accuracy better than 1 jiffy

*** How to detect if your timer system supports high resolution**

Method 1. Examine kernel startup message

```
# Example 1
$ dmesg | grep resolution
[ 0.000013] sched_clock: 32 bits at 24MHz, resolution 41ns, wraps every 89 s 478 484 971ns
```

Method 2. Examine proc file system (/proc/timer_list)

```
# Example 2
$ cat /proc/timer_list
Timer List Version: v0.3
HRTIMER_MAX_CLOCK_BASES: 2
now at 294115539550 nsecs
```

```
cpu: 0
 clock 0:
  .index:         0
  .resolution: 1 nsecs
  .get_time:     ktime_get_real
  .offset:       0 nsecs
active timers:
clock 1:
  .index:         1
  .resolution: 1 nsecs
  .get_time:     ktime_get
  .offset:       0 nsecs
active timers:
 #0: <c1e39e38>, tick_sched_timer, S:01, tick_nohz_restart_sched_tick, swapper/0
 # expires at 294117187500 nsecs [in 1647950 nsecs]
 #1: <c1e39e38>, it_real_fn, S:01, do_setitimer, syslogd/796
 # expires at 1207087219238 nsecs [in 912971679688 nsecs]
  .expires_next    : 294117187500 nsecs
  .hres_active     : 1
  .nr_events       : 1635
  .nohz_mode       : 2
  .idle_tick       : 294078125000 nsecs
  .tick_stopped    : 0
  .idle_jiffies    : 4294966537
  .idle_calls      : 2798
  .idle_sleeps     : 1031
  .idle_entrytime : 294105407714 nsecs
  .idle_sleeptime : 286135498094 nsecs
  .last_jiffies    : 4294966541
  .next_jiffies    : 4294966555
  .idle_expires    : 294179687500 nsecs
jiffies: 4294966542


Tick Device: mode:     1
Clock Event Device: 32k-timer
 max_delta_ns:   2147483647
 min_delta_ns:   30517
 mult:           140737
 shift:          32
 mode:           3
 next_event:     294117187500 nsecs
 set_next_event: omap_32k_timer_set_next_event
 set_mode:       omap_32k_timer_set_mode
 event_handler:  hrtimer_interrupt
```

B. How to implement highly accurate timers in Linux Userspace?

(Link: http://stackoverflow.com/questions/24051863/how-to-implement-highly-accurate-timers-in-linux-userspace)

The POSIX timers (created with timer_create()) are already high-resolution.
clock_gettime() in conjunction with signal handling (handling SIGALRM).

Alternately, you could use a timerfd instead of a POSIX timer (created with timerfd_create())

*** Testing POSIX high resolution timers.**

High resolution timers was implemented in kernel 2.6.16 and provide better resolution like 1 jiffy. Code was created by Ingo Molnar and Thomas Greixner. Implementation should provide high precision timers (with granularity ~1us) for user space applications. Simple code below check what is precise of resolution of high resolution timers implemented in kernel. It is also example program how to create periodic timer with high resolution in user space.

4. Design and Preparation

A. Design Goal

(1) What is the difference of thread and signal handler?

(2) Combine Metronome (metronome_tui_thread.c) from Lab3 and HR timer from Lab 4 (test_hrtimer.c) to build metronome_hrt.c

(*) Understand test_hrtimer.c code to modify the code metronome_tui_thread.c

B. Algorithm

1. Init GPIO LED
   - Init HR timer
     Create signal handler for HR timer - metronome processing
2. Init key processing
   - Set termios
   - Print title and menu
3. Set default values to parameters (TimeSig 3 (3/4) , Tempo 90, Stop)
   - Print default values

4. Loop
   - Get user input key without enter in blocking mode
   - If 'q' = break
   - Interpret the key
     - If 'z' = Time signature
       - Increasing TimeSig
       - If TimeSig >= 4 TimeSig = 1 (Rotating)
     - If 'c' = Decrease Tempo
       - Tempo = Tempo - 5
       - If Tempo < 30; Tempo = 30
     - If 'b' = Increase Tempo
       - Tempo = Tempo + 5
       - If Tempo > 200; Tempo = 200
     - If 'm' = Start or Stop
       - Start = 1 if stop, 0 else
       - Start/Stop HR timer
   - Print single line message: Input & Status (Without linefeed)

5. Print quit message

6. Reset termios

C. Files

- (To make metronome (From lab 3)) metronome_hrt.c (including signal_handler)
- (To make metronome (From lab 3)) gpio_led_fu.c (for bone)
- (To make metronome (From lab 3)) gpio_led_fu_sim.c (for PC)
- (To make metronome (From lab 3)) key_input_fu.c

5. Experiment Procedures

A. Test hrtimer

(1) How to detect if your timer system supports high resolution

Examine kernel startup message

```
$ dmesg | grep resolution
```

Examine proc file system (/proc/timer_list)

```
$ cat /proc/timer_list
```

(2) test_hrtimer.c

Download the c code file "test_hrtimer.c" from KLMS
Compile with -lrt option

```
$ arm-linux-gnueabihf-gcc -o a_test_hrtimer test_hrtimer.c -lrt
```

Run on Bone

```
$ ./a_test_hrtimer
[5]Diff time:1.000043
[4]Diff time:0.999991
[3]Diff time:1.000005
[2]Diff time:1.000008
[1]Diff time:1.000013
```

B. Test metronome_hrt.c

(1) Debug on PC

(2) Run on PC

```
$ ./metronome_hrt_sim
Sim user_led_setup
    Menu for Metronome_hrt:
z: Time signature 2/4 > 3/4 > 4/4 > 6/8 / 2/4 ...
c: Dec tempo          Dec tempo by 5 (min tempo 30)
b: Inc tempo          Inc tempo by 5 (max tempo 200)
m: Start/Stop         Toggle start and stop
```

```
q: Quit this program
    Key m: TimeSig 4/4, Tempo 90, Run 1.

311 312 321 322 331 332 341 342 311 312 321 322 Key,: TimeSig 4/4, Tempo 90, Run 1.
311 332 341 342 311 312 321 322 331 332 341 342 311 312 321 322 331 332 341 342 311
Key m: TimeSig 4/4, Tempo 90, Run 0.
342 311 Key m: TimeSig 4/4, Tempo 90, Run 0.
  Key z: TimeSig 6/8, Tempo 90, Run 0.
  Key m: TimeSig 6/8, Tempo 90, Run 1.
412 421 422 431 432 441 442 451 452 461 462 411 412 421 422 431 432 441 442 451 452
461 462 411 412 421 Key m: TimeSig 6/8, Tempo 90, Run 0.
  Key z: TimeSig 2/4, Tempo 90, Run 0.
  Key m: TimeSig 2/4, Tempo 90, Run 1.
112 121 122 111 112 121 122 111 112 Key m: TimeSig 2/4, Tempo 90, Run0.
```

(3) Run on Bone

Stop LED

```
$ ./stop_user_leds.sh
Stop user LED0
Stop user LED1
Stop user LED2
Stop user LED3
```

Run Metronome

```
$ ./metronome_hrt
GPIO1 at 4804c000 is mapped to 0xb6fd7000

    Menu for Metronome_hrt:
z: Time signature 2/4 > 3/4 > 4/4 > 6/8 / 2/4 ...
c: Dec tempo            Dec tempo by 5 (min tempo 30)
b: Inc tempo            Inc tempo by 5 (max tempo 200)
m: Start/Stop           Toggle start and stop
q: Quit this program
    Key m: TimeSig 4/4, Tempo 90, Run 1.

311 312 321 322 331 332 341 342 311 312 321 322 331 332 341 342 311 Key m: TimeSig 4/4,
Tempo 90, Run 0.
        Key z: TimeSig 6/8, Tempo 90, Run 0.
        Key b: TimeSig 6/8, Tempo 95, Run 0.
        Key b: TimeSig 6/8, Tempo 100, Run 0.
        Key b: TimeSig 6/8, Tempo 105, Run 0.
        Key b: TimeSig 6/8, Tempo 110, Run 0.
412 421 422 431 432 441 442 451 452 461 462 411 412 421 422 431 432 441 442 451
452 461 462 411 412 421 422 431 432 441 442 451 452 461 462 411 412 421 Key m: TimeSig
6/8, Tempo 110, Run 0.
        Key z: TimeSig 2/4, Tempo 90, Run 0.
        Key m: TimeSig 2/4, Tempo 95, Run 1.
112 121 122 111 112 121 122 111 112 121 122 111 112 121 122 111 112 121 122 111
112 121 122 111 112 121 122 Key m: TimeSig 2/4, Tempo 110, Run 0.
Quit
```

Restore user LEDs

```
$ ./restore_user_leds.sh
Restore user LED0
Restore user LED1
Restore user LED2
Restore user LED3
```

6. Demonstration

Demonstrate metronome_hrt

7. Report

Each student should prepare his own report containing:

- Purpose
- Experiment Sequence
- Experimental Result
- Discussion
- References

Discussion items

A. Search timers on AM3359 and summarize features.

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

8. References

[1] Getting Started with Beaglebone and Beaglebone Black, http://www.beagleboard.org//Getting%20Started

[2] Beaglebone Rev. A5. System Reference Manual, http://circuitco.com/support/index.php?title=Beaglebone#Rev_A5.

[3] "Embedded Linux Priver", C.Hallinan, Prentice Hall.