EE414 Embedded Systems

Lab 4. Interrupt

Due Demo 4 – 6 PM, Nov 16, Thu.
Report 6 PM, Nov 21, Tue.

1. Purpose

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

2. Problem Statement

Problem 4 (Interrupt signal handler for high-resolution timer)

Program 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 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.

The signal handler plays the metronome should keep playing indefinitely (infinite number of measures) as far as the run state is equal to 1. At every call to signal handler, it performs displaying the next beat pattern to user LED lamps.

3. Technical Backgrounds

A. Hardware connection

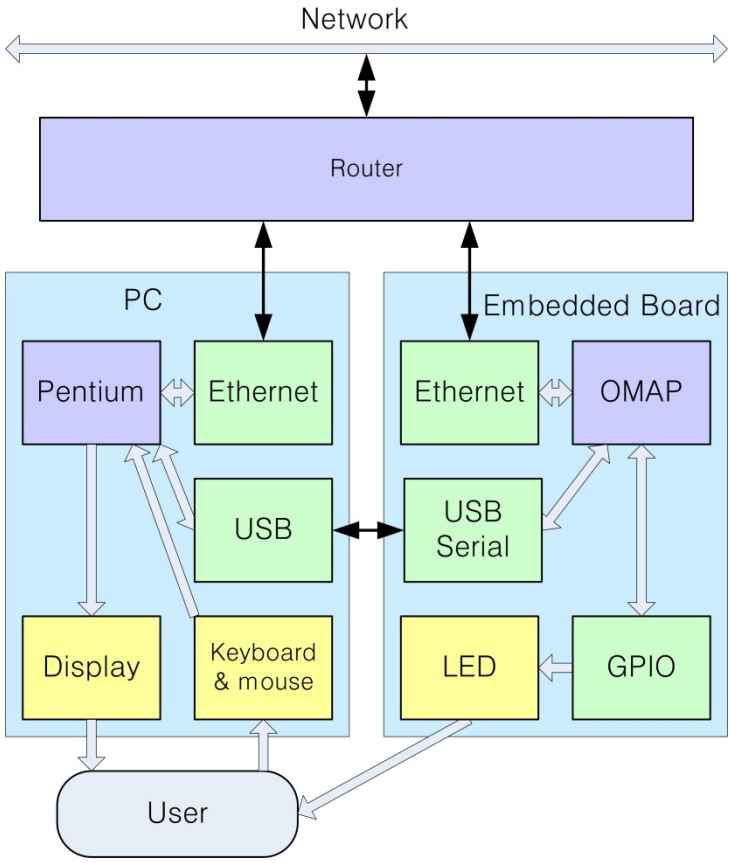

Fig. 4.1 Block Diagram for Lab 4

B. High Resolution Timers
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

Examine kernel startup messages

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

Examine /proc/timer_list

```
# cat /proc/timer_list
Timer List Version: v0.8
HRTIMER_MAX_CLOCK_BASES: 4
now at 20953449498121 nsecs
…....
cpu: 0
 clock 0:
  .base:      cfb67d80
```

```
  .index:     0
  .resolution: 1 nsecs
  .get_time:   ktime_get
  .offset:    0 nsecs
……
  clock 1:
  .base:      cfb67dc0
  .index:     1
  .resolution: 1 nsecs
  .get_time:   ktime_get_real
  .offset:    1470729819881022458 nsecs
 ……
Tick Device: mode:    1
…
 event_handler:  hrtimer_interrupt
```

C. How to implement highly accurate timers in Linux Userspace?
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.
http://open-nandra.com/2008/12/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 userspace 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 userspace.

test_posix_hrtimer1.c
Download and Test first.
Test Posix hrtimer 5 times with 1s 10 us.

4. Design and Preparation

1. What is the difference of thread and signal handler?

2. Combine
   Lab3/c_MetroTUI/metronome_tui_thread.c
   Lab4/a_HRTimer/test_hrtimer.c
   to build
   metronome_hrt.c

Menu

The same as in Lab 3.

Key input

Single key without enter

Use non-blocking key check: Refer test_single_key_nb.c

Algorithm

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

5. Loop
   A. Get user input key without enter in blocking mode.
   B. If 'q' break
   C. Interpret the key
      If 'z' (Time-signature)
         inc TimeSig
         If TimeSig >= 4 TimeSig = 1 (Rotating)
      If 'c' (Dec Tempo)
         Tempo = Tempo – 5
         If Tempo < 30 Tempo = 30
      If 'b' (Inc Tempo)
         Tempo = Tempo + 5
         If Tempo > 200 Tempo = 200
      If 'm' (Start/Stop)
         Start = 1 if stop, 0 else
         start/stop HR timer
   D. Print single line message: Input & Status

8. Print quit message
9. Reset termios

sig_handler

Play LED one-half note (on or off pattern)

Print pattern (at first half) or "." (at second half)

Files

metronome_hrt.c (incl. sig_handler)

gpio_led_fu.c (for Bone) gpio_led_fu_sim.c (for PC)

key_input_fu.c

Make two versions: PC and Bone

5. Experiment Procedures

Step 1. test_posix_hrtimer.c

1.1 How to detect if your timer system supports high resolution

Examine kernel startup messages

```
# dmesg | grep resolution
```

Examine /proc/timer_list

```
# cat /proc/timer_list
```

1.2 test_posix_hrtimer.c

Visit
Testing POSIX high resolution timers
http://open-nandra.com/2008/12/testing-posix-high-resolution-timers/

Download and test test_hrtimer.c.
It uses signal handler with high-resolution POSIX timer called every 1 s & 10 ns.
Compile with -lrt

```
# arm-linux-gnueabihf-gcc -o a_test_posix_hrtimer test_posix_hrtimer.c -lrt
```

Run on Bone

```
# ./a_test_posix_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
```

Step 2. Test metronome_hrt.c

2.1 Debug on PC

2.2 Run on PC

```
$ ./metronome_hrt_sim
Sim user_leds_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       Toggles start and stop
q:  Quit this program

 Key m: TimeSig 4/4, Tempo  90, Run 1.
7.1.1.1.7.1.1.1.7.1.1. Key ,: TimeSig 4/4, Tempo  90, Run 1.
7.1.1.1.7.1.1. 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.
7.1.1.3.1.1.7.1.1.3.1.1.7.1. 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.
71.17.1.7.1. Key m: TimeSig 2/4, Tempo  90, Run 0.
```

2.3 Run on Bone

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

# ./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      Toggles start and stop
q:  Quit this program

  Key m: TimeSig 4/4, Tempo  90, Run 1.
……
  Quit.

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

Which one is better for single key input: blocking or non-blocking?

Is the tempo more accurate than Lab 3? How can you measure accuracy?

6. Demonstration

Demonstrate metronome_hrt.

7. Report

Each student should prepare his own report containing:

Purpose
Experiment sequence
Experimental results
Discussion: should be different even for each member of the same team.
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] AM335x Technical Reference Manual. Texas Instruments.