EE414 Embedded Systems

Lab 3. Serial Input

Due ***11:59 PM, May 7. Fri.*** ***via KLMS***

1. Purpose

In order to grab a firm concept on serial communication, a serial input for the metronome will be programmed. This program will be used to send a command string from the PC to the embedded board via RS-232C serial cable (actually USB cable).

2. Problem Statement

Problem 3 (Serial Input)

Write an application program named "Metronome_tui" on the embedded board, which gets user commands – tempo, time-signature, start, and stop - from the PC keyboard via USB serial cable.

Instead of string command input, we use single keys without Enter key with menus as follows:

Menu

Use five single key inputs without Enter key.

'z'	Time signature	Rotates as 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 run state
'q'	Quit this program	

The Metronome_tui should play with user LEDs on the embedded board according to the Tempo, Time-signature, and Start commands.

Time signatures
2/4 : 71717171
3/4 : 711711711711
4/4 : 7131713171317131
6/8 : 711311711311711311711311

3. Technical Backgrounds

A. Hardware Setup

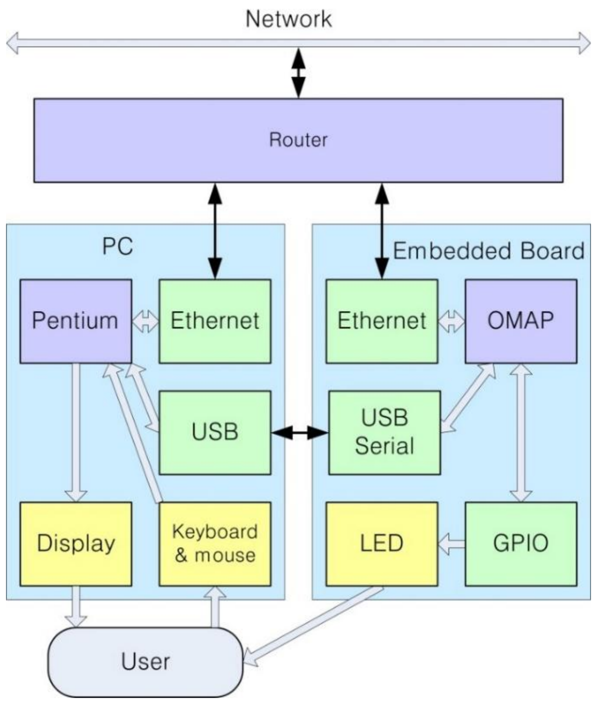

Fig. 3.1 Block Diagram for Lab 3

User I/O ←→ PC ←→ USB ←→ USB serial cable ←→ Embedded processor → GPIO → LEDs.
(keyboard and display)

B. System Block Diagram [1]

High level system block diagram of the Beaglebone Black.

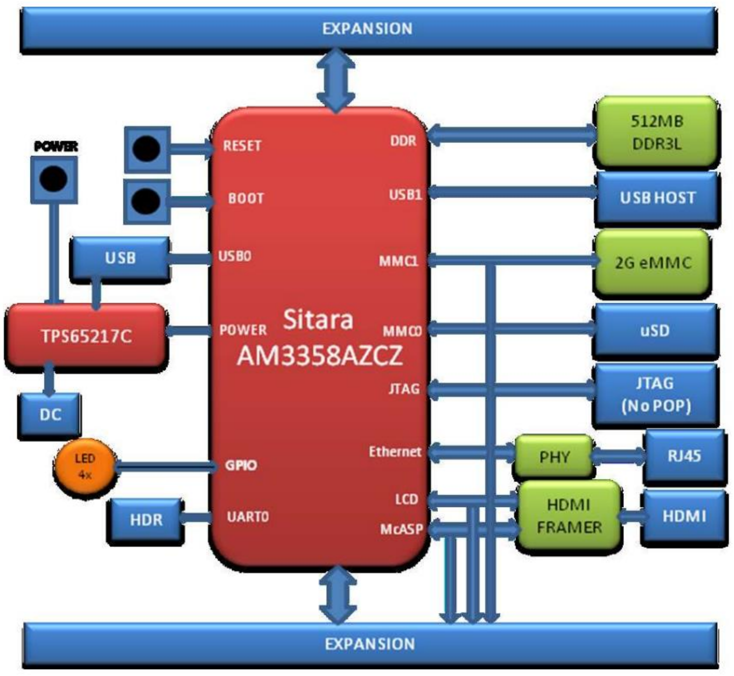

Fig 3.2. Beaglebone Black System Block Diagram

PC USB Interface [1]

The BeagleBone Black board has a miniUSB connector that connects the USB0 port to the processor. This is the same connector as used on the original BeagleBone.

USB1 Port

On the board is a single USB Type A female connector with full LS/FS/HS Host support that connects to USB1 on the processor. The port can provide power on/off control and up to 500mA of current at 5V. Under USB power, the board will not be able to supply the full 500mA, but should be sufficient to supply enough current for a lower power USB device supplying power between 50 to 100mA.

You can use a wireless keyboard/mouse configuration or you can add a HUB for standard keyboard and mouse interfacing.

Serial Debug Port

Serial debug is provided via UART0 on the processor via a single 1x6 pin header. In order to use the interface a USB to TTL adapter will be required. The header is compatible with the one provided by FTDI and can be purchased for about $12 to $20 from various sources. Signals supported are TX and RX. None of the handshake signals are supported.

C. Get single key without Enter key

What is equivalent to getch() & getche() in Linux?

http://stackoverflow.com/questions/7469139/what-is-equivalent-to-getch-getche-in-linux

On Linux (and other unix-like systems) this can be done in following way:

```
#include <unistd.h>
#include <termios.h>

char getch() {
        char buf = 0;
        struct termios old = {0};
        if (tcgetattr(0, &old) < 0)
                perror("tcsetattr()");
        old.c_lflag &= ~ICANON;
        old.c_lflag &= ~ECHO;
        old.c_cc[VMIN] = 1;
        old.c_cc[VTIME] = 0;
        if (tcsetattr(0, TCSANOW, &old) < 0)
                perror("tcsetattr ICANON");
        if (read(0, &buf, 1) < 0)
                perror ("read()");
        old.c_lflag |= ICANON;
        old.c_lflag |= ECHO;
        if (tcsetattr(0, TCSADRAIN, &old) < 0)
                perror ("tcsetattr ~ICANON");
        return (buf);
}
```

Basically you have to turn off canonical mode (and echo mode to suppress echoing).

We modify this program into several useful functions as test_single_key.c, which can get single keys until ‘q’ key:

```
/*
       Program test_single_key.c

       Get a key input without hitting Enter key
       using termios

   Modified from
http://stackoverflow.com/questions/7469139/what-is-equivalent-to-getch-getche-in-
linux
   Global old_termios and new_termios for efficient key inputs.
*/

#include <termios.h>
#include <stdio.h>
#include <unistd.h>        // read()

// GLobal termios structs
static struct termios old_tio;
static struct termios new_tio;

// Initialize new terminal i/o settings
void init_termios(int echo)
{
  tcgetattr(0, &old_tio);         // Grab old_tio terminal i/o setting
  new_tio = old_tio;              // Copy old_tio to new_tio
  new_tio.c_lflag &= ~ICANON;     // disable buffered i/o
  new_tio.c_lflag &= echo? ECHO : ~ECHO; // Set echo mode
  if (tcsetattr(0, TCSANOW, &new_tio) < 0)  perror("tcsetattr ~ICANON");
                                  // Set new_tio terminal i/o setting
}

// Restore old terminal i/o settings
void reset_termios(void)
{
  tcsetattr(0, TCSANOW, &old_tio);
}

// Read one character without Enter key: Blocking
char getch(void)
{
  char ch = 0;

  if (read(0, &ch, 1) < 0)  perror ("read()");        // Read one character

  return ch;
}
```

```
int main(void)
{
  char c;
  int echo;

  // Init termios: Disable buffered IO with arg 'echo'
  echo = 0;                        // Disable echo
  init_termios(echo);

  // Test loop
  printf("  Test_single_key\n");
  printf("single key input until 'q' key\n");
  while (1) {
       c = getch();
       printf("%c", c);
       //printf("%c %2x ", c, c);
       fflush(stdout);

       if (c == 'q') break;
  }
  printf("  Quit!\n");
  fflush(stdout);

  // Reset termios
  reset_termios();

  return 0;
}
```

Program test_single_key_nb.c

Test checking single key without enter key, with non-blocking mode

Add the following function:

```
#include <sys/select.h>
int key_hit()
{
   struct timeval tv = { 0L, 0L };
   fd_set fds;
   FD_ZERO(&fds);
   FD_SET(0, &fds);
   return select(1, &fds, NULL, NULL, &tv);
}
```

Change test loop in main() as follows:

```
// Test loop
  int i = 0;
  char waitchar[] = “|/-\\”;

  printf("  Test_single_key_nb\n");
  printf("single key input in non-blocking mode until 'q' key\n");
  while (1) {
       while (!key_hit()) {
         i = ++i % 4;
         printf("%c", waitchar[i]);
         fflush(stdout);
         usleep(250000);          // 0.25 s
       }
       c = getch();
       ……
```

D. Thread

A process can have multiple threads of execution which are executed asynchronously.

This asynchronous execution brings in the capability of each thread handling a particular work or service independently. Hence multiple threads running in a process handle their services which overall constitutes the complete capability of the process.

Read the following articles on thread in C:

Introduction to Linux Threads – Part I
https://www.thegeekstuff.com/2012/03/linux-threads-intro/

How to Create Threads in Linux (With a C Example Program)
https://www.thegeekstuff.com/2012/04/create-threads-in-linux/

How to Terminate a Thread in C Program (pthread_exit Example)
https://www.thegeekstuff.com/2012/04/terminate-c-thread/

4. Design and Preparation

1) What is difference of single key input in blocking and non-blocking mode?

2) Design **algo_metronome_tui.c** utilizing test_single_key.c

Algorithm for algo_metronome_tui can be constructed as follows

Algorithm for algo_metronome_tui

```
0. Set termios
1. Print title & menu.
2. Set default values to parameters (TimeSig 3 (3/4), Tempo 90, Stop)
   Print default values
3. Loop
        A. Get user input key without enter.
        B. If 'q' , break
        C. Interpret the key
            If 'z' (Time-signature)
            inc TimeSig
            Make TimeSig rotating: 1, 2, 3, 4, 1, 2, 3, 4, 1, …
        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)
            Run = 1 if run == 0. Run = 0 if run == 1 (toggle 0 and 1)
    D. Print single line message: Input & Status (Time-sig, tempo, and run)

8. Print quit message
9. Reset termios.
```

3) Design metronome_tui_thread.c using **thread.**

Why thread?

We require two wait loops:

1) Key input from user
2) Wait half periods for user LED display (Either on and off for one period)

We divide functionality as:

Main:	Key input from user
Thread:	Wait half periods for user LED display

Files

userLEDmmap.h	// Header file for memory map
metronome_tui_thread.c	// Main including thread
gpio_led_fu.c	// User LEDs output functions with mmap (modify lab2 source code)
key_input_fu.c	// Single key input functions (using test_single_key_nb.c)

Algorithm for metronome_tui_thread

Main:

```
1. Init GPIO LED
    Create thread for metronome processing
2. Init key processing
    Set the signal callback for Ctrl+C
    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
            Set TimeSig 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)
            Run = 1 if run == 0. Run = 0 if run == 1 (toggle 0 and 1)
    D. Print single line message: Input & Status
```

8. Print quit message
9. Reset termios

Thread:

```
Loop
    If Run == 0 sleep 0.1 s
    Else
        Play LED for each quarter or eighth note
        Print a character corresponding to Led pattern
```

Note

When you compile, put option **-pthread**:

```
ex) arm-linux-gnueabihf-gcc -pthread -o metronome_tui_thread metronome_tui_thread.c
```

Two Outputs are interlaced to one terminal:

```
Metronome thread:       A character
Key input & state: bKey n: TimeSig ccc, Tempo nnn, Run n
```

5. Experiment Procedures

Step 1. Test single key input

Test test_single_key_nb.c, which gets single keys without enter in non-blocking mode

Run on PC

```
$ ./test_single_key_nb_pc
single key input in non-blocking mode until 'q' key
/-\|/-\|/-\|/-\|/-\|/-\|/-\|/-\|/-\|/-\|/-\|/-\|/-\|/-\|/-\|/-\|/-\|/-\|/-
\|/-\|/-\|/-\|/-\a|/s-d\f|g/-\h|/hhhh-hhhhh\|/-\|/u-\i|/-k\|/-\j|/-\|/-\|/-
\|/-\|/-\|/-\|/-\|/-\|/-\|z/-\|q  Quit!
```

Step 2. Test Algorithm of Metronome_TUI.c

Test algorithmic part algo_metronome_tui.c of metronome_tui.c on PC, since PC is more convenient to debug.

No output to user LEDs, but print messages.

Can be tested on the PC using native gcc! This process will shorten your debugging time.

Run on PC

```
$ ./algo_metro_tui_pc
Menu for Metronome TUI:
'z':  Time signature 2/4 > 3/4 > 4/4 > 6/8 > 2/4 ...
'c':  Dec tempo              Dec tempo by 5
'b':  Inc tempo              Inc tempo by 5
'm':  Start/Stop             Toggles start and stop
'q':  Quit this program

  Key m: TimeSig 4/4, Tempo  80, Run 0.
……
  q: Quit!
```

Step 3. Test thread example programs

Example programs are located in:

How to Create Threads in Linux (With a C Example Program)
https://www.thegeekstuff.com/2012/04/create-threads-in-linux/

How to Terminate a Thread in C Program (pthread_exit Example)
https://www.thegeekstuff.com/2012/04/terminate-c-thread/

Step 4. Test metronome_tui_thread.c on Beaglebone

4.1 Debug on PC first!

Run on PC **('7' means 3 LEDs, '3' means 2 LEDs, and '1' means 1 LED.)**

```
$ ./metronome_tui_thread_sim
Menu for Metronome_TUI_thread:
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

This is a thread!
 Key m: TimeSig 4/4, Tempo  90, Run 1.
7111711171 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.
711311711311171131
  Quit.
```

4.2 Test on Bone finally!

Run on Bone

```
# ./metronome_tui_thread
Menu for Metronome_TUI_thread:
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

This is a thread!
 Key z: TimeSig 6/8, Tempo  90, Run 0.
 Key m: TimeSig 6/8, Tempo  90, Run 1.
71131171131171 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.
71717171717 Key m: TimeSig 2/4, Tempo  90, Run 0.
 Key b: TimeSig 2/4, Tempo  95, Run 0.
 Key b: TimeSig 2/4, Tempo 100, Run 0.
 Key b: TimeSig 2/4, Tempo 105, Run 0.
 Key b: TimeSig 2/4, Tempo 110, Run 0.
 Key b: TimeSig 2/4, Tempo 115, Run 0.
 Key b: TimeSig 2/4, Tempo 120, Run 0.
 Key b: TimeSig 2/4, Tempo 125, Run 0.
 Key b: TimeSig 2/4, Tempo 130, Run 0.
 Key m: TimeSig 2/4, Tempo 130, Run 1.
7171 Key m: TimeSig 2/4, Tempo 130, Run 0.
 Key z: TimeSig 3/4, Tempo 130, Run 0.
 Key m: TimeSig 3/4, Tempo 130, Run 1.
71171171171171
  Quit.
```

Are outputs (text and LED display) works as expected?

6. Demonstration

Demonstrate the result of metronome_tui_thread!

7. Report

Each student should prepare his own report containing:

Purpose
Experiment sequence
Experimental results
Discussion
References.

Discussion items

1) Explain how the console serial input/output is routed to USB in the Beaglebone.
2) Compare single key input methods: blocking and non-blocking
3) Explain why you turn of canonical and echoing modes.
4) Explain how you apply *pthread* in your lab briefly. Find the summary of functions what you used.

8. References

[1] BeagleBone Black System Reference Manual, on KLMS