Homework 3/4 Report  
 (Remote LED Ascii display, using UART

pyOCD and gdb)

Submitted by: Kripa Varma  
Date : 4/27/2015

# Summary

* Files : main.c, makefile, Homework2\_Report.docx  
   Combined the **UART example with the Hex LED display** program.
* The program is an infinite loop of 2 non blocking reads - one from pc serial and next from UART serial. If user inputs an hexadecimal digit, it is sent to remote mbed via UART. If a character is read from from UART, the corresponding hex digit is displayed on LEDs.
* Installed **pyOCD Windows** version.
* Used **arm-none-eabi-gdb** of windows based toolchain to download and debug the program on two mbeds.
* Also used **LPCXpresso IDE** to achieve the same goals – single stepping, setting breakpoints, halting execution etc.

# Learned:

* Using **pyOCD**

PyOCD or Python On Chip Debugger is an open source GNU Debugger Server software. When PyOCD is run on the pc, it opens a port, listens by default on port 3333, for incoming connections from gdb clients.

arm-none-eabi-gdb is used as the gdb client and it connects to pyOCD server and downloads program to the remote target via the server.

PC mbed

[ gdb client <===> pyOCD gdb server ] <===> [cmsis dap <===> mcu ]

* Using **gdb**

Learned usage of gdb commands ( break, step, continue, clear, print, info etc.)

* **UART communication** api for mbed.

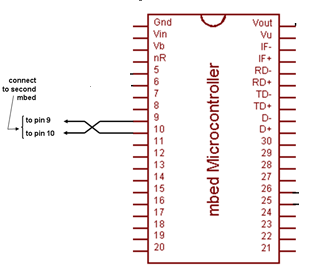
Used the apis baud(), putch(), getch(), readable()

* Using **LPCXpresso** IDE to download, run, debug and set breakpoints, single stepping.
* Non blocking read from pc/uart serial ports

Using readable() to perform non blocking reads from uart as well as serial port connection to pc.

# Procedure:

1. Downloaded pyOCD\_win.exe. When this binary is executed, it launches the OpenOCD gdb server and listens on port 3333 of local host.
2. Connected the two mbeds’s uart to each other and connected together the ground pins.



To mbed2 Gnd

1. Combined the UART program from HW 3 and Hex LED display from gram of HW4.

**Send\_word()** function performs **non-blocking read of pc serial** and if hex digit is read, sends it over UART.

void send\_word()

{

char ch;

if(pc.readable()==1)

{

led1=0; led2=0; led3=0; led4=0;

ch = pc.getc();

pc.putc(ch);

if( (ch>='0' && ch<='9') || (ch>='A' && ch<='F') || (ch>='a' && ch<='f') )

{

async\_port.putc(ch);

}

pc.printf("\r\nEnter:");

}

}

**Recv\_word()** performs **non blocking read** of character from **UART**. If char is read, it is sent to Hex LED display function.

void recv\_word()

{

//set leds according to incoming word from slave

char recd\_val;

if (async\_port.readable()==1) //is there a character to be read?

{

recd\_val=async\_port.getc(); //if yes, then read it

DisplayLed(recd\_val);

}

}

**Main()** program loops infinitely, calling send() /recv() functions alternatingly.

1. Used arm-none-eabi-gdb of windows based gnu arm toolchain to download the program to the mbed board, set break point in main(),read(),send() functions. Following is a short list of commands used during the testing.

<gdb> target remote localhost:3333 // to connect to the pyocd server

<gdb> file homework4.elf // to load symbols

<gdb> b main // to set breakpoint at main

<gdb> info breakpoints // to view currently set breakpoints

<gdb> load homework4.elf // to download and run the program

<gdb> c // to continue running, from current breakpoint stop

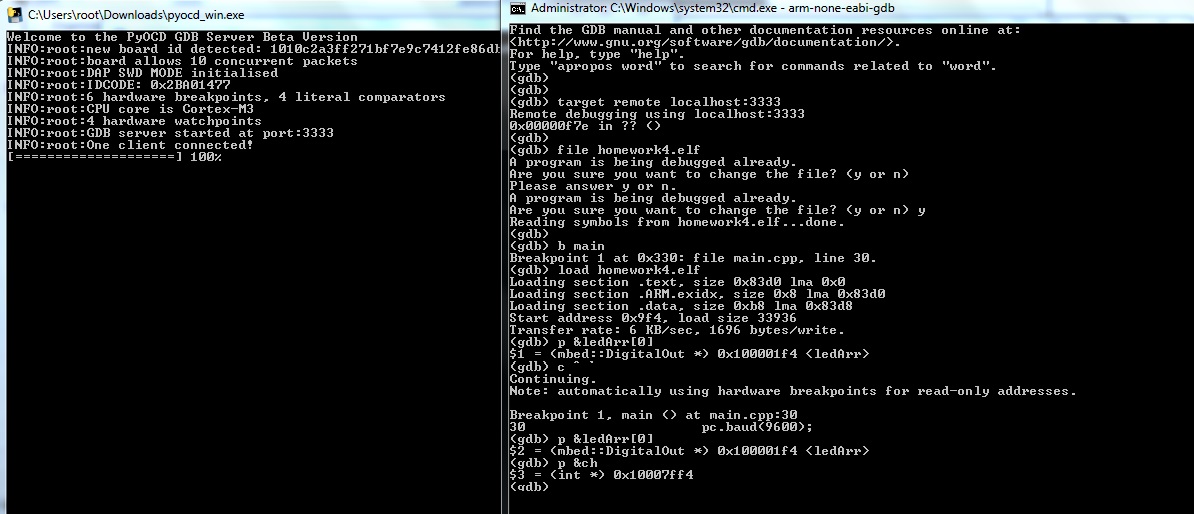
<gdb> ^Ctrl+C // to stop/halt the program at current instruction

<gdb> c // to continue execution again

<gdb> p &ch // to view address of variable ‘ch’

# Result:

1. If user inputs an hexadecimal digit on either one of the mbed modules, it is sent to remote mbed via UART. If a character is read from from UART, the corresponding hex digit is displayed on LEDs. This feature was verified using two mbeds.
2. The address of variables ledArr[0] and ch were displayed on gdb, via pyOCD as shown in the screenshot below.



# Hours:

2-3 Hrs : Coding.  
3 hrs : Installing Keil, LPCXpresso, running and debugging the program using pyOCD, gdb, LPCXpresso.

2-3 Hrs : Preparing the document

# References:

1. Gdb man pages