ECEN301 Embedded Systems Lab 5
Bare-metal ARM programming

Daniel Eisen 300447549

September 22, 2020

1 Objectives

This lab marks the beginning of the use of 32bit, ARM based microprocessors and compared to the previous 8-bit 8051, this is the step up into high performance embedded development. For this we used the Texas Instruments Beaglebone-Black dev board, based around the Cortex-A8 AM 3358 and so support both a bare-metal and OS development environments.
This session focused on familiarising ourselves with interfacing with the board via JTAG and UART serial as well as using Code Composer Studio to code, program, and debug the uC.

2 Methodology

2.1 Setup and Introduction

To setup the hardware in bare metal mode, the Linux OS auto-boot must be bypassed and this is just done be holding down the S2 as the board is powered on. To enable us to program and receive debugging information during development the board was connected to the computer via a JTAG to USB debugging module this allowed us to write, upload, and run code in a debug session with the Segger debugger giving us complete control and access via the JTAG port.
Setting up Code Composer as the programming environment for the board was straightforward, and just required setting the hardware that was being used and selecting the hello_world C code template. With everything configured and setup the code was compiled, uploaded and run in debug mode and the internal console displayed the print message. In addition we used the Disassembly window to view the generated ARM assembler code, seeing the main entry point (seeing the PC set) and following instructions.

2.2 Flashing Lights

As is always necessary with kit that has lights we must make them flash. Code Composer allows for projects stored as archived to be easily unpackaged and imported, so using a provided archive the board was programmed to light up the 4 on board blue LEDs.
This was controlled in code with the `GPIO_setPin/clrPin` functions to individually set the LEDs on/off so I modified the code using simple blocking delays and loops to display a little animation that looked like the light was bouncing back and forth from LED0 to LED3.

2.3 Serial and LEDs

Now to send information back and forth between the uC and the computer during runtime there is the on bouard UART comms port that we can open a serial connection in a terminal to interface with.
I used Putty and modified to provided code (see modified below) to receive a char from the COMM ports, send it back and utilise a provided LED_setValue function to display the 4 least significant bits of that char on the LEDs.
This function masks the 4 LSB, clears the port (the 4 LEDs) and pushing the masked value to them, resulting in the bit values of the passed char displayed on the 4 LEDs.

```
1
2  LED_init();
3
4  UART_initUART(UART0,115200,STOP1,PARITY_NONE,FLOW_OFF);
5  UART_putString(UART0,"UART0 Initialized...\n",21);
6  char u_char = 0;
7  while(1){
8      u_char = UART_getC(UART0);
9      UART_putC(UART0, u_char);
10     LED_setValue(u_char);
11 }
12 return 0;
```