MSP430 Hardware Porting Guidelines

Introduction

This document will explain the process to port the Bluetopia stack to run on a different hardware platform. Currently the MSP430 contains support for the EZ430-256x, the MSP430F5438A Experimenter Board and the MSP430F5529 Experimenter Board. This will document will show the process of porting to the MSP430F5529 Experimenter Board to serve as a guideline for porting to other hardware platforms.

Creating the Starting Point

The Bluetopia stack uses the functions exposed in HCITRANS.c to communicate with the Bluetooth chip. The HCITRANS.c contains a UART driver that relies on C macros to configure the UART correctly to communicate with the Bluetooth chip. These macros are contained in HRDWCFG. We will use the EZ430-RF256x platform files are the starting point for porting to the 5529 Experimenter Board. First under the "MSP430_Experimenter\Hardware" directory we will create a new folder called "MSP430_EXP5529". Then copy the three files under the "MSP430_Experimenter\Hardware\ez430" into the directory that was just created. The "MSP430_EXP5529" will look like the following:

- MSP430_EXP5529\
  - **HAL.c** This is used for setting up the system clocks of the MSP430, for configuring the debug UART and for setting up the processor pins to a default configuration (to ensure no I/Os are left floating).
  - **HAL.h** This contains the protoypes of the functions that are exposed in HAL.c
  - **HRDWCFG.h** This contains the macros that are used to configure the system.

Modifying HRDWCFG.h

First we will configure the mapping for the Slow Clock line for the Bluetooth chip. The CC256x requires a 32.768 KHz signal be driven to it for internal timing. For this purpose we will use a pin on the 5529 that the inter MSP430 ACLK signal can be driven to. The ACLK will be driven by the 32.768 KHz crystal that is attached to XT1. We will use Port 1 Pin 0 as the ACLK output for the CC256x. In HRDWCFG.h we will make the following change to configure the mapping:

```
#define BT_SLOW_CLOCK_PORT_BASE          ((unsigned int)&P1IN)
#define BT_SLOW_CLOCK_PORT_PIN           (BIT0)
```

Next we will configure a GPIO for use as a reset line for the CC256x. This is used to reset the CC256x to a known state before we initialize the Bluetopia stack. We will use Port 7 Pin 5 for this purpose. In HRDWCFG.h we will make the following change to configure the mapping:

```
#define BT_DEVICE_RESET_PORT_BASE        ((unsigned int)&P7IN)
#define BT_DEVICE_RESET_PORT_PIN         (BIT5)
```

Next we will configure a GPIO for use as the RTS line for the Bluetooth UART. The UART module on the 5529 does not support hardware flow control so this line is necessary to provide emulated hardware flow control in software. We will use Port 2 Pin 1 for this purpose. In HRDWCFG.h we will make the following change to configure the mapping:

```
#define BT_UART_FLOW_RTS_PIN_BASE        ((unsigned int)&P2IN)
#define BT_UART_RTS_PIN                  (BIT1)
```

Next we will configure a GPIO for use as the CTS line for the Bluetooth UART. The UART module on the 5529 does not support hardware flow control so this line is necessary to provide emulated hardware flow control in software and this will also be used to provide a mechanism for the CC256x to wake up the 5529 from low power modes when some Bluetooth event needs to be processed. We will use Port 2 Pin 4 for this purpose. In HRDWCFG.h we will make the following change to configure the mapping:

```
#define BT_UART_FLOW_CTS_PIN_BASE        ((unsigned int)&P2IN)
#define BT_UART_CTS_IV                   (PORT2_VECTOR)
#define BT_UART_CTS_IVR                  (P2IV)
#define BT_UART_CTS_PIN                  (BIT4) // no change necessary!
#define BT_UART_CTS_INT_NUM              (P2IV_P2IFG4)
```

Finally we will configure the UART that is used to communicate with the Bluetooth chip. The UART that we will use is USCI_A1 which uses Pin 4 for transmitting and Pin 5 on Port 4 for reception of UART characters to and from the Bluetooth chip. In HRDWCFG.h we will make the following change to configure the mapping:

```
#define BT_UART_MODULE_BASE                ((unsigned int)&UCA1CTLW0)
#define BT_UART_IV                         (USCI_A1_VECTOR)
#define BT_UART_IVR                        (UCA1IV)
#define BT_UART_PIN_PORT_BASE              ((unsigned int)&P4IN)
#define BT_UART_PIN_TX                     (BIT4) //no change necessary!
#define BT_UART_PIN_RX                     (BIT5) //no change necessary!
```

Modifying HAL.c

Since this HAL will only run on 1 chip version (MSP430F5529) there is no reason to determine the processor version. Therefore change the **DetermineProcessorType**() function to always return TRUE (any non-zero value).

The next thing that we need to do is ensure that the pins used by the crystal are configured as peripheral pins. To do this in the **StartCrystalOscillator**() function changed the following line as follows:

```
   P7SEL   |= (BIT1 | BIT0);
```

To

```
   P5SEL   |= (BIT5 | BIT4);
```

This oscillator is used to generate the system clock so we need to ensure that the proper pins are configured for use by the clock system.

Next we need to change **ConfigureBoardDefaults**() to configure all of the port pins to a known state. In this function we will delete all of the code already there and add the following code:

```
static void ConfigureBoardDefaults(void)
{
   /* Configure the default GPIO configuration.
*/
   P1OUT = 0;
   P1DIR = 0xFF;
   P2OUT = 0;
   P2DIR = (unsigned char)
~BIT0;
   P3OUT = 0;
   P3DIR = 0xFF;
   P4OUT = 0;
   P4DIR = (unsigned char)
~(BIT0 | BIT2 | BIT6 | BIT7);
   P5OUT = 0;
   P5DIR = 0xFF;

   /* P6.5 is ADC input from put show we should configure as an input
*/
   /* with a pulldown resistor.
*/
   P6OUT = 0;
   P6DIR = 0xDF;
   P6REN = BIT5;

   P7OUT = 0;
   P7DIR = 0xFF;
   P8OUT = 0;
   P8DIR = 0xFF;

   /* The following pins are jumpered to other pins on J12 so need to
be*/
   /* configured as inputs with pulldowns.
*/
   P2REN |= BIT0;
   P4REN |= (BIT0 | BIT6 | BIT7);
}
```

Next we will setup the LEDs that will be used by our application. We will use LEDs 2 and 3 on the board. These are connected to Port 8 Pins 1 and 2. We will change the **ConfigureLEDs**() to the following to configure the pins.

```
static void ConfigureLEDs(void)
{
   P8SEL &= ~(BIT1 | BIT2);
   P8DIR |= (BIT1 | BIT2);
}
```

Similarly we need to change the **ToggleLED**() and **SetLED()** functions as follows:

```
static void ToggleLED(int LEDID)
{
   if(P8OUT & BIT2)
   {
      P8OUT &= ~BIT2;
      P8OUT |= BIT1;
   }
   else
   {
      P8OUT |= BIT2;
      P8OUT &= ~BIT1;
   }
}

static void SetLED(int LED_ID, int State)
{
   Byte_t Mask;

  if(LED_ID)
     Mask = BIT2;
  else
     Mask = BIT1;
  if(State)
     P8OUT |= Mask;
  else
     P8OUT &= ~Mask;
}
```

Since the UART normally used for PC communication on the 5529 is used to communicate with the Bluetooth chip we have no need to configure the debug UART in the **HAL_ConfigureHardware**() function. Therefore we will delete the following lines:

```
   /* Configure the UART-USB Port for its default configuration
*/
   HAL_CommConfigure(BT_DEBUG_UART_BASE, BT_DEBUG_UART_BAUDRATE, 0);
   GPIOPinTypeUART(BT_DEBUG_UART_PIN_BASE, BT_DEBUG_UART_PIN_TX_MASK,
BT_DEBUG_UART_PIN_RX_MASK);

   /* Enable Debug UART Receive Interrupt.
*/
   UARTIntEnableReceive(BT_DEBUG_UART_BASE);
```

We also need to delete the debug UART interrupt handler. Therefore delete the **DEBUG_UART_INTERRUPT**() interrupt function. The **HAL_ConsoleRead**() and **HAL_ConsoleWrite**() functions should be stubbed out as there is no console to write to.

In the **CTS_ISR**() interrupt function the **P1IV_P1IFG2** case should be deleted as this was specific to the ez430 platform.