FireBird32 Header and Functions
Version 1.04
README
Created By Nick Schulze
17th July 2011
www.HowNotToEngineer.com

DISCLAIMER:

These two files (Firebird.h and Firebird.c) can be included in your Firebird32 project file to hopefully make your life a little easier when coding the Firebird32 board.

I began writing this code for my own use and decided it may come in handy to others. I offer it free of charge for your own use. All I ask is please keep the main headings intact at the top of the files so that others may find their way to my site.

I offer no guaranty or warranty, all I know is the code works for me, I have commented as best I can so hopefully it is easy to follow for anyone reading it.

INSTRUCTIONS:

1. Include the files

Copy the files into your projects source directory, and in CodeWarrior include the files into the project. At the top of your main.c file make sure you include them using the instruction below.

```
#include "Firebird.h"
```

You can delete the default includes "derivative.h" and <hidef.h> as these are included in the "Firebird.h"

Next call the initialise function in your main file BEFORE the infinite while loop, currently this will set the BUSCLOCK of the Firebird32 to 24Mhz by default.

```
Firebird_init();
```

2. Digital IO's

The Firebird has 14 dedicated digital IO pins as shown in the port map picture, the functions of pins 6, 10 and 11 depend on the setting of three jumpers. Currently the jumpers are just set so that the 3 pins are standard GPIO's.

The 14 digital IO's can be set as inputs or outputs using .

```
SET_OUTPUT(D0);

SET_INPUT(D0);
```

You can then set the output or set the direction of the pullup resistor on the input using.

```
DIGITAL_HIGH(D0);

DIGITAL_LOW(D0);
```

You can also toggle the desired digital IO , If it is HIGH it will go LOW if it is LOW it will go HIGH

```
DIGITAL_TOGGLE(D0);
```

Or Pulse the IO HIGH then right away LOW

```
DIGITAL_PULSE(D0);
```

To Read the value of a digital pin

```
DIGITAL_READ(D0);
```

D0 obviously refers to digital IO 0, these functions will work for D0 -> D13.

PWM digital pins 3, 5, 6, 9, 10, 11, set the duty cycle value between 0 (full off) and 256 (full on)

```
PWM(D3, 127);           //50% duty cycle PWM on digital pin 3

PWM_OFF(D3);            //Turn off PWM to use pin as standard digital IO
```

3. Analogue Inputs

For sensor, potentiometer or any analogue voltage reading the Firebird has 8 analogue input pins you can use. To read an analogue value referenced to 0 and 5 volts use the following function.

```
ANALOGUE_read(x);      // x = 0 -> 7 which are the 8 analogue inputs avaliable on the Firebird32
```

By default the ADC does a 10bit conversion i.e. results between 0 and 1023 and takes one reading per call of the above function. You can set the ADC to perform 8bit or 12bit conversions and also to do 2, 4, or 8 averages per call of the read function. An example of how to change these settings is shown below.

```
ANALOGUE_settings(12, 8);      //this will set 12bit conversion mode, and return the average of 8
                               readings each time the read function is called
```

4. RGB LED

The onboard RGB LED is easily controlled using the following functions.

To set the required direction registers use.

```
RGB_ENABLE;

RGB_DISABLE;
```

To apply power to the anode use.

```
ANODE_HIGH;

ANODE_LOW;
```

The red, green and blue cathodes are default low so when you set the anode high the RGB will display white light. To control the red green and blue values use.

```
RED_ON;

RED_OFF;

GREEN_ON;

GREEN_OFF;

BLUE_ON;

BLUE_OFF;
```

A nice feature of RGB LED's is their ability to produce all sorts of different colours by pulse width modulating the three elements. I have written two functions, one which uses the hardware PWM capabilities of the Flexis chip which is nice and efficient and only has to be called once. The other utilises bit angle modulation (BAM) which is essentially software PWM, this function has to be run in the infinite while loop to work, it's not particularly efficient but is a good demonstration of software PWM.

```
RGB_BAM(x, y, z);        //run in while loop

RGB(x, y, z);            //call once to set colour

RGB_off();               //turn off RGB PWM to use pins as standard digital IO's
```

x, y and z are 8 bit numbers (0-255) which correspond to a RGB colour code

5. Buzzer

The buzzer is simple to control, toggle the following functions at whatever frequency you would like to produce.

```
BUZZER_ENABLE;

BUZZER_DISABLE;

BUZZER_HIGH;

BUZZER_LOW;
```

6. 8x2 LCD

The LCD panel is a little more complicated and makes use of several functions including the delay function, using my functions it is very easy to write to the screen though.

To write a string to the screen use LCD print, currently strings longer than 8bits will wrap to the second line.

```
LCD_print("hello");
```

To clear the screen and reset the cursor to the top left position use.

```
LCD_clear();
```

To write to the second line use the new line function before the next write operation.

```
LCD_newLine();
```

7. Serial

The Serial Communication Interface (SCI), offers a convenient way to communicate between devices. The Firebird has two serial ports, but currently only SCI1 is setup in my code. To use the serial interface use the following code.

Initialise the serial port using the following code where BAUD is one of the baud rate constants I have defined in Firebird.h, the currently available baud rates are shown below.

```
SERIAL_begin(BAUD);

BAUD =  B1200

        B4800

        B9600

        B19200

        B115200
```

To use the serial interface I have set up a polling style interface, if you have used the Arduino you will have probably had experience with this. To send data you simply use the write function for a single byte or the send function for a string or array.

```
SERIAL_write(0x41);       //0x41 is the HEX representation of the character 'A'

SERIAL_print("Hello World");
```

To receive you must poll the serial interface for the availability of data and then use the receive function to retrieve the data.

```
char RXdata = 0x00;

if(SERIAL_available()){

        RXdata = SERIAL_receive();

}
```

8. Tactile Pushbutton Input

The Firebird32 has an onboard tactile switch these functions allow you to read user inputs from the switch, I have written a software debouncer to ensure that only one press is registered when you press the button.

```
PRESS_ENABLE;          //Macro to setup input and pullup resistor

READ_PRESS;            //Macro to read value to input

ButtonPressed();       //function to read input presses with debounce
```

9. Useful Functions

- Delay

To interface with the LCD some short delays are required so I wrote the following function, this function will halt ALL operation except interrupts for the duration of the delay. The duration is measured in milliseconds (1000ms = 1s), the following example will delay for 10 milliseconds or 1/100th of a second.

You may notice that the interrupts are underlined in CodeWarriorV10, this is because I have defined them without using the interrupt vector table which CodeWarrior does not seem to like. I will explain how to set up the correct table in one of my tutorials, but until then the way they are defined now should make it easier to include them in your own projects.

```
DelayMillis(10);
```

- Double to String conversion

Convert a double floating point number to a string to send to the LCD screen or as characters via the serial port. This function is completely portable as it has no external dependencies, CodeWarrior does not support the standard c double to string conversion so this should come in handy.

Converts the number to the null terminal string with chosen number of decimal points, for example:

```
dtos(-9.43, 3) = “-9.430”

dtos(432.2999, 1) = “432.3”

dtos(0.43, 4) = “0.4300”
```

- Round number to nearest int

A portable round function which works for both negative and positive numbers

```
round(8.8) = 9

round(8.4) = 8
```

- Map

Map a number in a certain range to a new range, such as mapping a result from a 10 bit ACD read (0-1023) to a voltage level (0-5).

```
double input = ADC_read(1);

double voltage = map(input, 0, 1023, 0, 5);
```