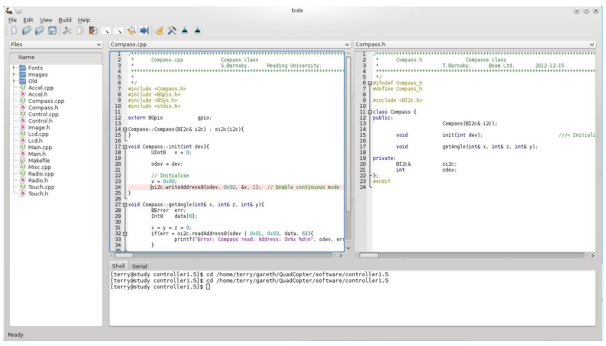
BEAM

Armsys

What Is It ?

Micro-controllers based on the 32bit ARM core provide a lot of power to create sophisticated embedded systems. To help with harnessing this power, in an easy to use way, we have developed the Armsys embedded development environment. This provides a simple to use object orientated, real-time embedded operating system together with a development environment based on the GNU C++ compiler tool set, the FreeRTOS real-time kernel, the LWIP networking stack, DSP functions and the Armsys C++ class library to provide a powerful, but easy to use, development and run-time environment.



```
/*******************************************************************
 *      Compass.cpp             Compass class
 *                              G.Barnaby,      Reading University,
 *******************************************************************
 *
 */
#include <Compass.h>
#include <BGpio.h>
#include <BSys.h>
#include <stdio.h>

extern BGpio            gpio;

Compass::Compass(BI2c& i2c) : oi2c(i2c){
}

void Compass::init(int dev){
        UInt8   v = 0;

        odev = dev;

        // Initialise
        v = 0x00;
        oi2c.writeAddress8(odev, 0x02, &v, 1);  // Enable continuous mode
}

void Compass::getAngle(int& x, int& z, int& y){
        BError  err;
        Int8    data[6];

        x = y = z = 0;
        if(err = oi2c.readAddress8(odev | 0x01, 0x03, data, 6)){
                printf("Error: Compass read: Address: 0x%x %d\n", odev, err
        }
```

```
/*******************************************************************
 *      Compass.h               Compasse class
 *                              T.Barnaby,      Beam Ltd.       2012-12-15
 *******************************************************************
 */
#ifndef Compass_h
#define Compass_h

#include <BI2c.h>

class Compass {
public:
                        Compass(BI2c& i2c);

        void            init(int dev);                          ///< Initiali

        void            getAngle(int& x, int& z, int& y);

private:
        BI2c&           oi2c;
        int             odev;
};
#endif
```

```
[terry@study controller1.5]$ cd /home/terry/gareth/QuadCopter/software/controller1.5
[terry@study controller1.5]$ cd /home/terry/gareth/QuadCopter/software/controller1.5
[terry@study controller1.5]$ 
```

Some core features of the Armsys system include:

- Simple C++ class library to access system hardware including I2C, SPI, RTC, ADC's etc.
- C++ class library provides easy usage of the real time operating system features of FreeRTOS allowing multiple tasks to be run simultaneously.
- LWIP TCP/IP networking with simple to use Ethernet HTTP WEB server, DHCP, NTP, SNMP services.
- Wifi module serial interface classes.
- Real-time DSP functionality.
- Easy to use GUI IDE for development.
- Available for Linux and Microsoft Windows platforms.

Table of Contents

1. Introduction

2. Example Application

3. Interrupts

The STM32 CPU’s have an excellent interrupt system with multiple interrupt vectors, almost one vector per interrupt source. By default these interrupt vectors are set to point to the default_handler() function that aborts operation of the program flashing the systems LED’s at a fast rate and alternately if there are two or more LED’s.

The interrupts can be enabled or disabled using the BInterrupt class. This provides simple access to the CPU’s main interrupt controller and the separated external interrupt controller used for the pin based interrupts. To enable an interrupt you can use the function:

```
        sys.interrupts.config(BUInt channel, BUInt8 priority, Bool on);
```

To enable individual external pin interrupts you can use the functions:

```
        sys.interrupts.extConfig(Pin pin, Bool event, Bool rising, Bool falling, Bool on);
```

When an interrupt is enabled the CPU will call a function in iots interrupt vector list of functions. To override an interrupt you simply provide the function in one of your C++ files with “C” linkage. this will override the standard call to the default handler.

All of the CPU’s external IO port pins can be set to generate an interrupt on their rising or falling edges or both of these. They are multiplexed to a degree to a set of functions, the level of multiplexing dependent on the actual CPU in use. For example on an STM32F405, all of the ports multiplex there pins together. So PinA1 interrupts on the same interrupt as PinB1. Also some of the ports bit numbers are multiplexed to a single interrupt. So interrupts for bits 5,6,7,8,9,10 generate the same interrupt. The STM32F405 provides the following interrupt functions for external interrupts:

```
        void EXT0_IRQHandler();
        void EXT1_IRQHandler();
        void EXT2_IRQHandler();
        void EXT3_IRQHandler();
        void EXT4_IRQHandler();
        void EXT9_5_IRQHandler();
        void EXTI15_10_IRQHandler();
```

To implement an interrupt function:

```
extern “C” {
        void EXTI9_5_IRQHandler(void){
                if(EXTI->PR & EXTI_PR_PR8){
                        myIntFunc8();
                        EXTI->PR = EXTI_PR_PR8;
                }
                if(EXTI->PR & EXTI_PR_PR9){
                        myIntFunc9();
                        EXTI->PR = EXTI_PR_PR9;
                }
        }
}
```

To enable an external interrupt on PinA8 and PinA9 to call these functions you can do the following:

```
        // Configure external interrupt
        sys.interrupts.extConfig(PinA8, 0, 0, 1, 1);   // Sets interrupt on falling edge on PinA8
        sys.interrupts.extConfig(PinA9, 0, 0, 1, 1);   // Sets interrupt on falling edge on PinA9

        // Enable this external interrupt in the main interrupt controller
        sys.interrupts.config(EXTI9_5_IRQn, 9, 1); // Enables interrupt with priority 9
```