Green Electronics Library Documentation

Ned Danyliw

September 30, 2016

1 Introduction

The Green Electronics libraries provide a simplified interface to the STM32F3 microcontroller for the labs in this class. The libraries wrap around ST's own libraries which provide much more functionality but require a much deeper understanding of the microcontroller which is outside the scope of this course. The Green Electronics libraries allow you to use much of the STM32F3's functionality including its ADCs, timers, PWM, GPIO, input capture, and UART.

This handout will discuss the basics of using the libraries and programming the STM32F3 microcontroller.

2 Updating the Libraries

The library code is hosted on Github at https://github.com/ndanyliw/green-electronics. Whenever using the library, make sure to pull the latest code from the repository to get any bugfixes or upgrades to the code.

To do this, first navigate to your local repository directory (`~/green-electronics` in the provided virtual machine). Then pull the latest code using Git:

```
git pull origin master
```

or if you have re-mapped your Git remotes (i.e. mapped to a private repository) you may have to use the following:

```
git pull upstream master
```

If there is updated code, make sure to clean out the old generated library files using `make reallyclean` within a project directory (i.e. project-template or a lab folder). Then the next time `make` is called, it will recompile the library.

3 Navigating the Repository

The Green Electronics repository contains several folder whose purposes are listed below.

- *datasheets* - Contains datasheets and reference manuals for the STM32F3 microcontroller.
- *hardware* - Hardware design documents for the Green Electronics breakout board.
- *labs* - Starter code for all the labs.
- *libraries* - The custom Green Electronics libraries.
- *misc* - Miscellaneous files for the repository. Currently the shared linker scripts for the microcontroller.
- *project-template* - A blank project template for the microcontroller. Copy this folder to a new location to setup a new project.
- *scripts* - Helper scripts for the repository. The board script simplifies connecting the UART.
- *init_bitbucket.sh* - Script that remaps the Git remote to point to a private Bitbucket repository.
- *setup.sh* - Handle installing all the library dependencies on a new machine.

4 Creating a New Project

To create a new project you simply need to copy the `project-template` folder to your new project directory. The new project starts off with the Hello World demo code. Just replace the code in `src/main.c` with your own.

To copy the project template via the command line use the command:

```
cp -r ~/green-electronics/project-template /path/to/new/project
```

5 Library Basics

When using the Green Electronics libraries make sure to include `ge_libs.h` which includes all the necessary Green Electronics library files.

The first step is to initialize the libraries which is done through the `ge_init()` function. This handles setting up all the common data structures, timers, ADCs, etc. that are used by the libraries. This method handles calling all of the libraries' init functions.

Each library also has its own runtime specific setup code which is detailed below. For more detailed information on the libraries look in the `src/` and `inc/` folders in the `libraries` directory in the repository.

ADC1 Channel	Pin	ADC2 Channel	Pin	ADC3 Channel	Pin	ADC4 Channel	Pin
ADC1_1	PA0	ADC2_1	PA4	ADC3_1	PB1	ADC4_1	PE14
ADC1_2	PA1	ADC2_2	PA5	ADC3_2	PE9	ADC4_2	PE15
ADC1_3	PA2	ADC2_3	PA6	ADC3_3	PE13	ADC4_3	PB12
ADC1_4	PA3	ADC2_4	PA7	ADC3_5	PB13	ADC4_4	PB14
ADC1_5	PF4	ADC2_5	PC4	ADC3_12	PB0	ADC4_5	PB15
ADC12_6	PC0	ADC2_11	PC5	ADC3_13	PE7	ADC4_12	PD8
ADC12_7	PC1	ADC2_12	PB2	ADC3_14	PE10	ADC4_13	PD9
ADC12_8	PC2			ADC3_15	PE11		
ADC12_9	PC3			ADC3_16	PE12		
ADC12_10	PF2			ADC34_6	PE8		
				ADC34_7	PD10		
				ADC34_8	PD11		
				ADC34_9	PD12		
				ADC34_10	PD13		
				ADC34_11	PD14		

Table 1: ADC channels and corresponding pins.

5.1 ADC

The ADC library is found in `ge_adc.h/c`. The library configures the internal ADCs on the STM32F3 to perform single-ended conversions at the specified sampling rate and then call a user-specified callback function with all the conversion results.

The Green Electronics breakout board exposes four of the ADC channels. These channels have specific definitions that correspond to the labeled ADC channel (i.e. GE_ADC1, GE_ADC2, GE_ADC3, or GE_ADC4). GE_ADC3 and GE_ADC4 are connected directly to the A3 and A4 pins respectively. They will convert an input voltage between 0-3V to a 12-bit value. The GE_ADC1 and GE_ADC2 channels are connected to instrumentation amplifiers which have internal gains (default is 5) and a reference voltage of 1.5V. This means that you may need to add attenuation to avoid saturation. Additionally the reference voltage of 1.5V means that a differential input of 0V will correspond to 1.5V on the ADC pin meaning you may need to subtract a fixed offset from the result to properly map to the input voltage.

The other ADC channel mappings are shown in table 1.

When using the ADC library, you will need to specify the sampling rate of the ADC, what channels are being converted, and the callback function to handle the ADC results. An example is shown in Listing 1.

Listing 1: ADC Example Code

```
ADC_CHAN_Type chan_to_conv[4] = {GE_ADC1, GE_ADC2,
  GE_ADC3, GE_ADC4};

void my_adc_callback(uint16_t *data) {
  // handle data here
}

int main() {
  // ...
```

```
10
  adc_set_fs(10000); // set Fs to 10kHz
  adc_callback(&my_adc_callback);
  adc_enable_channels(chan_to_conv, 4);
  adc_initialize_channels();
15 adc_start();

  // ...
}
```

5.2 Timer

The Green Electronics library provides a simplified timer interface in `ge_timer.h/c`. This library allows you to register periodic or one-shot callback functions to the hardware timers on the STM32F3 board.

The library requires you to specify the minimum timestep for the timers and then register the callbacks and number of timesteps per call.

Listing 2: Timer Example Code

```
void toggle_led() {
  // do something
}

5 int main() {
  // ...
  // Set minimum timestep to 1ms (number of counts
  // referenced to a 72MHz clock)
  timer_set_timestep(72000);
10 // register callback for toggling LEDs every 500ms
  // use GE_SINGLESHOT if only happening once
  led_timer = timer_register(500, &toggle_led, GE_PERIODIC);
  timer_start(led_timer);
  // ...
15 }
```

5.3 UART

The provided UART library allows the STM32F3 to communicate over the serial port using the `printf()` function. This port is initialized to communicate at a 115200 baud rate in the `ge_init()` function. The library code can be found in `ge_uart.h/c`.

5.4 PWM

There are three PWM pins available on the breakout board. The library code is found in `ge_pwm.h/c`. The three external PWM pins are connected to PA8, PA9, and PA10 (labeled PWM1, PWM2, and PWM3 respectively).

To setup the PWM pins, first set the PWM frequency using the `pwm_freq()` function. Then enable the pin using `pwm_set_pin()`. Finally to set the duty cycle, use the `pwm_set()` function and specify the appropriate PWM channel (`PWM_CHAN1`, `PWM_CHAN2`, `PWM_CHAN3`, or `PWM_CHAN4`).

Listing 3: PWM Example Code

```
int main() {
  // ...
  pwm_freq(25000); // set to 25 kHz
  pwm_set_pin(PA8);
  pwm_set(PWM_CHAN1, 0.3); // set to 30% duty cycle
  // ...
}
```

5.5 LCD

The LCD library provides an easy way to print to the on-board LCD. The LCD code is found in `ge_lcd.h/c`. When printing the the LCD, specify the starting location using `lcd_goto(x,y)` and then the string to print using `lcd_puts()`. Note that if your string exceeds the spaces available in the line, the string will wrap to another line due to the addressing scheme of the LCD.

5.6 Input Capture

The input capture library allows you to read in the frequency of a waveform. The input capture pin is labeled on the breakout board. To read in the last measured frequency use the command `ic_int_read_freq()` which returns the frequency in hertz.

5.7 GPIO

The GPIO library allows you to setup any of the GPIO available on the STM32F3 as well as read/write to them. The code is located in `ge_gpio.h/c`. To setup the pin use `gpio_setup_pin()`. This function allows you to set the direction of the pin as well as if it is open drain/push-pull and has internal pull-up resistors. To write to the pin use `gpio_write_pin()`. To read use `gpio_read_pin()`.

Listing 4: GPIO Example Code

```
int main() {
  // ...
  // set PA5 as output, push-pull, no pull-up
  gpio_setup_pin(PA5, GPIO_OUTPUT, false, false);
  gpio_write_pin(PA5, GPIO_HIGH);
  // set PA6 as input, open-drain, internal pull-up
  gpio_setup_pin(PA6, GPIO_INPUT, true, true);
  int res = gpio_read_pin(PA6);
```

```
  // ...
}
```

5.8 EEPROM

The EEPROM library allows the user to store values in non-volatile memory allowing you to save things like calibration constants. Use the `eeprom_read()` and `eeprom_write()` methods to use the EEPROM. The code is found in `ge_eeprom.h/c`

5.9 Pin Definitions

There are many pin definitions available to simplify the naming of resources being used with the Green Electronics libraries. These definitions can be found in `ge_pins.h`.