Lab Assignment: ADC + PWM

Objective

Implement an ADC driver, implement a PWM driver, and design and implement an embedded application, which uses both drivers.

This lab will utilize:

- ADC Driver
- PWM Driver
- FreeRTOS Tasks
- A potentiometer
- An RGB LED

Assignment

Part 0: Implement basic ADC Driver and read Light Sensor Values

- Channel 2 (Pin P0.25) already has Light Sensor connected to it.
- Create just 1 task which reads the Light sensor value and prints it periodically.
- While the task is running cover the light sensor and your task should print values <50.
- Use the flash light on your phone on the light sensor and your task should print values >3500.

```
void light_sensor_print_task(void *p)
{
	/*
	 * 1) Initial ADC setup (Power, clkselect, pinselect, clkdivider)
	 * 2) Select ADC channel 2
	 * 3) Enable burst mode
	 */

	while(1) {
		uint16_t ls_val = adc_read_channel(2);
		printf("Light Sensor value is %d\n", ls_val);
```

```
    delay_ms(100);
}
```

Part 1: Implement an ADC Driver

Using the following header file,

- Implement adcDriver.cpp such that it implements all the methods in adcDriver.h below.
- Every method must accomplish its task as indicated in the comments.
- You may add any other methods to enhance the functionality of this driver.
- It is recommended that you test your ADC driver with ADC_PIN_0_25 because it is connected to the analog light sensor and this is probably the easiest way to test your driver.

> For proper operation of the SJOne board, do NOT configure any pins as ADC except for 0.26, 1.30, 1.31

> While in burst mode, do not wait for the "DONE" bit to get set.

```
#include <stdio.h>
#include "io.hpp"
class LabAdc
{
public:
    enum Pin
    {
        k0_25,       // AD0.2 <-- Light Sensor -->
        k0_26,       // AD0.3
        k1_30,       // AD0.4
        k1_31,       // AD0.5

        /* These ADC channels are compromised on the SJ-One,
         * hence you do not need to support them
         */
        // k0_23 = 0,   // AD0.0
        // k0_24,       // AD0.1
        // k0_3,        // AD0.6
        // k0_2         // AD0.7
    };
```

```
    // Nothing needs to be done within the default constructor
    LabAdc();

    /**
    * 1) Powers up ADC peripheral
    * 2) Set peripheral clock
    * 2) Enable ADC
    * 3) Select ADC channels
    * 4) Enable burst mode
    */
    void AdcInitBurstMode();
    /**
    * 1) Selects ADC functionality of any of the ADC pins that are ADC capable
    *
    * @param pin is the LabAdc::Pin enumeration of the desired pin.
    *
    * WARNING: For proper operation of the SJOne board, do NOT configure any pins
    *           as ADC except for 0.26, 1.31, 1.30
    */
    void AdcSelectPin(Pin pin);

    /**
    * 1) Returns the voltage reading of the 12bit register of a given ADC channel
    * You have to convert the ADC raw value to the voltage value
    * @param channel is the number (0 through 7) of the desired ADC channel.
    */
    float ReadAdcVoltageByChannel(uint8_t channel);};
```

Part 2: Implement a PWM Driver

Using the following header file,

- Implement pwmDriver.cpp such that it implements all the methods in pwmDriver.h below.
- Every method must accomplish its task as indicated in the comments.
- You may add any other methods to enhance the functionality of this driver.
- **It may be best to test the PWM driver by using a logic analyzer**

```
#include <stdint.h>
```

```
class LabPwm
{
 public:
    enum Pin
    {
        k2_0,    // PWM1.1
        k2_1,    // PWM1.2
        k2_2,    // PWM1.3
        k2_3,    // PWM1.4
        k2_4,    // PWM1.5
        k2_5,    // PWM1.6
    };
    /// Nothing needs to be done within the default constructor
    LabPwm() {}
    /**
    * 1) Select PWM functionality on all PWM-able pins.
    */
    void PwmSelectAllPins();

□/**
    * 1) Select PWM functionality of pwm_pin_arg
    *
    * @param pwm_pin_arg is the PWM_PIN enumeration of the desired pin.
    */
    void PwmSelectPin(PWM_PIN pwm_pin_arg);

    /**
    * Initialize your PWM peripherals.  See the notes here:
    * http://books.socialledge.com/books/embedded-drivers-real-time-operating-systems/page/pwm-%28puls
    *
    * In general, you init the PWM peripheral, its frequency, and initialize your PWM channels and set
    *
    * @param frequency_Hz is the initial frequency in Hz.
    */
    void PwmInitSingleEdgeMode(uint32_t frequency_Hz);
```

```
/**
    * 1) Convert duty_cycle_percentage to the appropriate match register value (depends on current fre
    * 2) Assign the above value to the appropriate MRn register (depends on pwm_pin_arg)
    *
    * @param pwm_pin_arg is the PWM_PIN enumeration of the desired pin.
    * @param duty_cycle_percentage is the desired duty cycle percentage.
    */
void SetDutyCycle(PWM_PIN pwm_pin_arg, float duty_cycle_percentage);


/**
    * Optional:
    * 1) Convert frequency_Hz to the appropriate match register value
    * 2) Assign the above value to MR0
    *
    * @param frequency_hz is the desired frequency of all pwm pins
    */
void SetFrequency(uint32_t frequency_Hz);};
```

Part 3: Application

In order to demonstrate that both drivers function, you are required to interface a potentiometer and an RGB LED to the SJOne board. The potentiometer ADC input shall control the duty cycle of the RGB LED pwm outputs. Note that an RGB LED has three input pins that you will connect to three different PWM output pins. You must use your own ADC and PWM drivers, as well as your own FreeRTOS task.

> **Extra credit** can be earned with an interesting/cool/creative RGB output.

Requirements

- Using your own ADC Driver, read input voltage from a potentiometer
  - Print the voltage reading every 1s.
- Using your own PWM Driver, drive an RGB LED.
  - Print the duty cycle of all three RGB pins every 1s.
- The PWM output to the RGB LED must be dependent on the ADC input from the potentiometer.
- By varying the potentiometer, you should be able to see changes in the color of the RGB Led.

> ? You don't need a periodic task for the PWM to work. Initialize the driver, set period and duty cycle. PWM will start generating pulses immediately. You can vary the duty cycle of PWM inside the ADC task.

Revision #27
Created 6 years ago by Admin

Updated 3 years ago by Preet Kang