CE2810: Embedded Systems Software 2

A Poor Man's Light Dimmer

"Experience is a dim lamp, which only lights the one who bears it." - Louis-Ferdinand Celine

Due: Tuesday, May 1, 2012 23:00

1 Objectives

- Implement an interrupt driven Device driver which will allow the user to sample up to 8 analog channels
- Implement Interrupt Service Routines in C
- Implement a device driver that will generate a PWM signal with a given duty cycle.
- Practice integer mathematics on the ATMEGA32 device.

2 Introduction

According to InterNACH, lighting uses 11% of a home's energy budget. That's a lot of energy. And that is an area which is ripe for savings.

While there are many ways to go about reducing energy, one way which can prove successful is to alter the amount of light produced based on the task being performed. For example, reading a book requires bright lighting, and studying at the kitchen table also requires bright lighting. But, if one is eating dinner, the light may not need to be as bright. Thus, energy can be conserved by using dimmers.

One way to dim lighting is to use a pulse width modulation scheme. With this scheme, the light bulb is rapidly turned on and off many hundreds of times a second based upon the required brightness. If more light is required, the bulb is on more whereas if less light is required the bulb is turned on less.

In this lab, you are to construct a device driver for the A/D device in C. The device shall have the capability of reading all 8 AD channels and storing them into an array of uint16_t variables. Externally, through an accessor operation, the application can access any one of the 8 A/D readings.

Additionally, you will add to this project to control the brightness of the LEDs through pulse width modulation. LED 3 will directly be controlled by the PWM peripheral on the ATMEGA32. The other LEDS will be controlled via turning on and off in the interrupt service routines associated with the PWM cell firing.

Your instructor will aid you with the wiring of a potentiometer to set the voltage to the A/D converter if you feel uncomfortable doing it yourself.

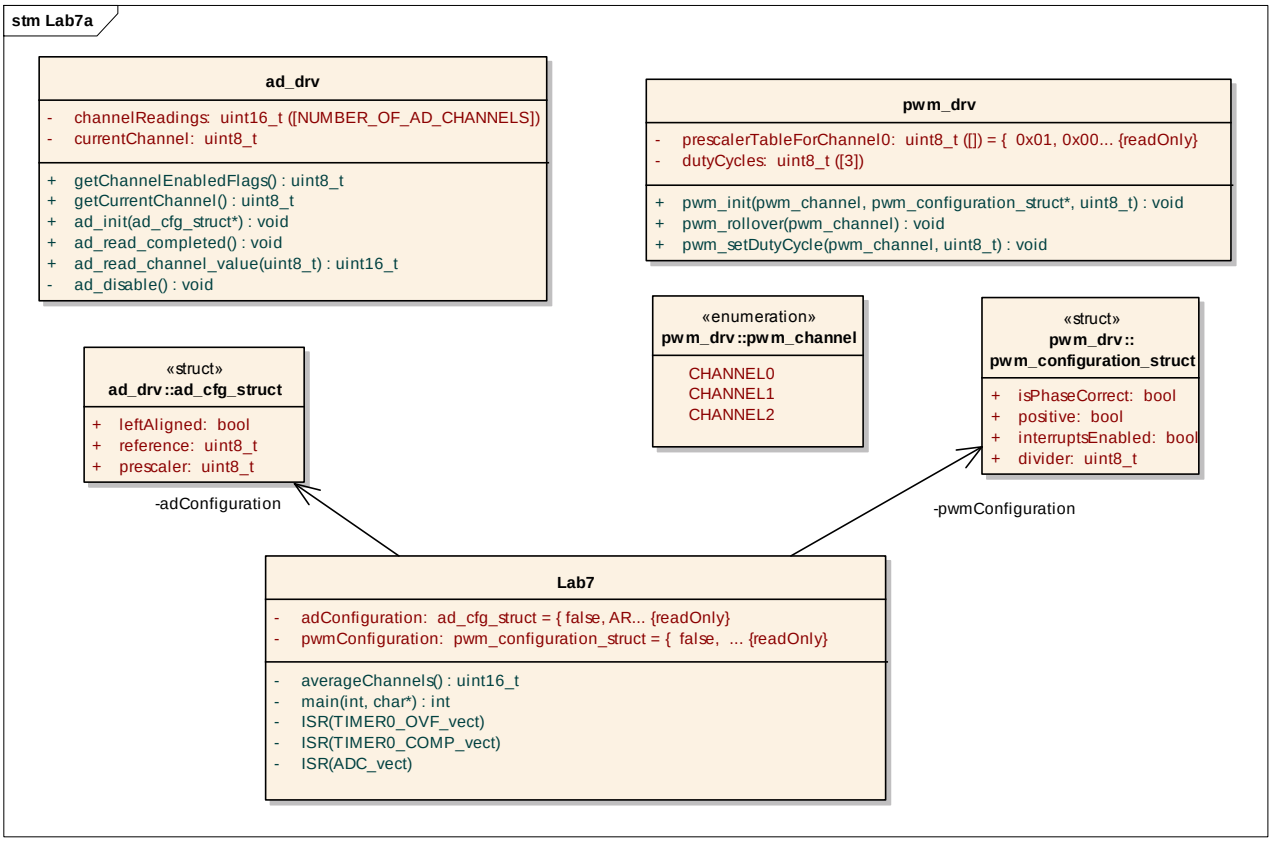

Figure 1 Suggested UML Class diagram for project.

3 Lab process

In addition to implementing the code, you are to follow the following, simple lab process.

1. Estimate how many lines of code you expect this project to be from start to finish.
2. Estimate how long you expect it to take you to complete this project.
3. Implement the solution, keeping track of how long it took to complete the project. (While it is not required that you use project dashboard, it is certainly permitted.)
4. At project completion, compare your line of code count and your time estimates with the actual values.

4 Lab Assignment

4.1 Design Requirements

- Requirement R1: PWM channel 0 shall be configured to drive an LED using fast PWM mode.
- Requirement R2: PWM Channel 0 duty cycle (which ranges between 0 and 100%) shall be set using the equation to 100% if the average of the values read from the A/D on

channel 0 and channel 1 is 1023 and 0 if the value read is 0, with intermediate values in between.

- Requirement R3: LEDS 0 and 7 shall be controlled via interrupt and shall have a duty cycle of 100% if the PWM duty cycle is set to 100% and 0% if the PWM duty cycle is set to 0%.
- Requirement R4: LEDs 1, 2, 4, 5, and 6 shall have a duty cycle of 100% if the PWM duty cycle on the PWM peripheral is set to 0% and 0% if the PWM duty cycle is set to 100%.

4.2 Implementation Constraints

- Constraint C1: The definitions for ports shall be defined in PortDef.h.
- Constraint C2: The only allowable types within this program shall be the C99 types.
- Constraint C3: The source code shall compile cleanly without any warnings or error messages.
- Constraint C4: Implementation of the divider within the PWM init routine shall be accomplished through a table lookup operation.

5 Deliverables

5.1 Lab Report

Now that you have completed your lab assignment, submit the following lab report detailing your experiences. The lab report should be submitted electronically through the online submission form on the course website.

1. Introduction -> What did you accomplish with this lab?
2. Time Analysis and code analysis
   a. How long did you plan on spending on this project?
   b. How many lines of code did you estimate for this project?
   c. How long did this project actually take?
   d. What is the count of new and changed code for this project versus last week's project? (Note: You can measure this using Project Dashboard or RSM metrics, which ever you desire.)
   e. How much code space does your program use? How many bytes are used per line of C code? How does this compare with the last lab? What has the impact of adding the string library to your code had on this metric?
   f. How does this compare with programs from CE2800?
3. Questions
   a. How did you implement the scaling between the duty cycle passed in, which varies between 0 and 100, and the configuration of the timer? How did the compiler implement this logic? (Hint: You'll need to look at the generated assembly to answer this question.)
   b. How does the compiler handle multiply within your code? Does it use the built in multiply instructions of the ATMEGA32? (Hint: You'll need to look at the generated assembly to answer this question.)
4. Compile log

   a. Copy and paste the output of the compiler in the message window indicating that the code compiles without warnings or errors.
5. Things Gone Right
   a. What things went right in performing this lab?
6. Things Gone Wrong
   a. What problems did you have?
   b. What mistakes did you make?
7. Conclusions
   a. This section shall discuss what has been learned from this laboratory experience.
8. Source Code. Submit your well commented source code.

5.2 Project Demo

In addition to the lab report, you are required to demonstrate correct functionality. This can be accomplished prior to lab on May 2, 2012 or during the time period immediately following the lab quiz on May 2, 2012.

If you have any questions, consult your instructor.

Appendix A: Function Documentation

Pwm drv	`void pwm_init(pwm_channel channel, const pwm_configuration_struct* configuration, uint8_t dutyCycle);`	This method will initialize the PWM channel. The initialization will be based on the configuration passed in the structure. Parameters: - `pwm_channel channel` This is the channel that is to be setup. - `pwm_configuration_struct* configuration` This is a pointer to the configuration structure that is to be used. - `Uint8_t dutyCycle` This si the initial duty cycle, between 0 and 100.
	`void pwm_rollover(pwm_channel channel);`	This method shall be invoked whenever a timer rollover occurs. It shall update the value in the PWM register for the given channel. Internal to the file there is an array of PWM values, one for each channel. When this method is called, the value in the array shall be scaled appropriately for the timer/counter and shall be placed in the control register. Parameters: - `pwm_channel` channel This is the channel which has rolled over, either CHANNEL0, CHANNEL1, or CHANNEL2.
	`void pwm_setDutyCycle(pwm_channel channel, uint8_t dutyCycle);`	This method will set the duty cycle for the given channel. The duty cycle ranges between 0 and 100. Parameters: - `pwm_channel` channel – This is the channel for which the PWM duty cycle is to be set. - Uint8_t dutyCycle – This is the duty cycle that is to be set, between 0 and 100.
	`typedef struct { bool isPhaseCorrect; bool positive; bool interruptsEnabled; uint8_t divider; } pwm_configuration_struct;`	bool isPhaseCorrect – This will be true if phase correct PWM is to be used. Otherwise, it will be false. bool positive – This will be true if positive PWM is to be used, whereby the signal will rise on an overflow and fall on a compare. bool interruptsEnabled – This will be true if the interrupts are to be enabled. uint8_t divider – This is theclock divider for pwm, and it is expressed as the power of 2 for the divider. For example, a value of 0 results in a division by 1 (2^0), a value of 1 represent division by 2, etc. This shall result in a table lookup for the correct configuration bits. If the division factor is invalid, NO clock shall be used and the cell shall effectivly be disabled. } pwm_configuration_struct;
main	`int main(int argc, char*`	This is the main method for the program. It initializes all devices, enabled the global interrupt flag, initializes memory

	`argv[])`	buffers, and then enters into an infinite loop. This loop reads all 8 channels into a local array and converts them into a textual string. Once this is completed, the top and bottom lines are written out to the lcd display. It then updates the duty cycle for the pwm signal by calling the appropriate methods.
	`ADC_vect interrupt`	This interrupt service routine will be called whenever an a/d reading is completed. In short, it will call the ad_read_completed function.
	`TIMER0_OVF_vect`	This interrupt service routine will be called whenever a rollover occurs of the PWM signal. This ISR will call the method pwm_rollover with the channel as 0, indicating that channel 0 pwm has rolled over. It will also adjust the LEDs properly to obtain the proper dimming of the NON-peripheral driven LEDs.
A/D Driver	`void ad_init(const ad_cfg_struct* configuration);`	This function will initialize the A/D converted. It will initialize all data values to 0, determine the first channel to convert, and set up the appropriate registers for the A/D converter to convert in an interrupt driven mode. The system shall operate in free running mode, always converting channels. Parameters - `const ad_cfg_struct* configuration` This is a pointer to the configuratioon structure that is to be used to setup this peripheral.
	`void ad_read_completed()`	This function will be called whenever the interrupt fires indicating that an A/D conversion has completed. It is responsible for reading the values from the register (ADC), storing the value into the appropriate variable, determining the next channel to read, updating the mux, and initiating a new conversion. Effeciency is important within this routine, as it is called from an ISR.
	`uint16_t ad_read_channel_value(uint8_t channel)`	This function will return the last A/D reading for the given channel. Parameters: uint8_t channel – This is the channel that is to be retruned between 0 and 7. Return: The last A/D reading for the given channel will be returned. If the channel is disabled, the reading shall be 0. If the channel is out of range, the value returned shall be 0.
	`Structure ad_cfg_struct`	bool leftAligned This parameter will be true if the values are to be left aligned and false if they are to be right aligned. (Hint: ADLAR bit.) uint8_t reference This parameter will select the given reference voltage. Potential values are AREF, AVCC, and INTERNAL. uint8_t prescaler This is the prescaler division factor, effectivly setting the frequency for A/D conversions, and it ranges between 1 and 128 based on powers of 2.