Chen Bai 1560405

EE 474
Lab#4 Report

1.Abstract

In lecture last week, we have talked about the analog to digital converter on Tiva board which can convert the analog input into digital output. In this lab, we will apply this knowledge and make the board show the temperature (analog) by different LED's color (digital programmed). And at early of this quarter, we have discussed about the phase lock loop (PLL) along with the system clock, and how to use the PLL to change the system clock frequency, but we did not apply this skill until this lab. We need to use switch to change the frequency of system clock, which means PLL is necessary. What is more we do not hold the switch this time, so we need only single edge sensitive for switch interrupt trigger. I successfully finished the lab: the program works well. When I press sw1, the clock frequency increases and heat the board, and when I press sw2, the clock frequency decreases and cool the board down. The internal sensor can sense the temperature switch and the board changes LED's color accordingly.

2.Introduction

If we want to do the analog to digital conversion, we need to set the I/O first: the analog input and digital output. In this lab, we use only one input (only one temperature measure), so we use PE3(AIN0) as input, and LEDs as output. Then, we need a sample sequencer to collect data for the ADC to give output, we have only one in and one out, therefore, one step is enough; So, we apply sequencer 3. We also need to use the interrupt for ADC to change the color of LED according to the temperature. And because we need to use our switches to switch system clock's frequency, we need to initialize the PLL. This lab extends the lab before; therefore, we need all the function from previous lab, though the content some of the function swill be changed totally. And we only use ADC0 in this lab because only one ADC is needed.

3.Procedure

3.1 Checking the data sheet[1]

To use interrupts, we should find the address we need:

1. *For GPIO:*

- GPIO Interrupt Event (GPIOIEV): determine which edge trigger the interrupt. [p666]
- GPIO Alternate Function Select (GPIOAFSEL): to make specific pin controlled by peripheral [p671]
- GPIO Analog Mode Select (GPIOAMSEL): to disable or enable the analog function for specific pin [p687]

2. *For ADC:*

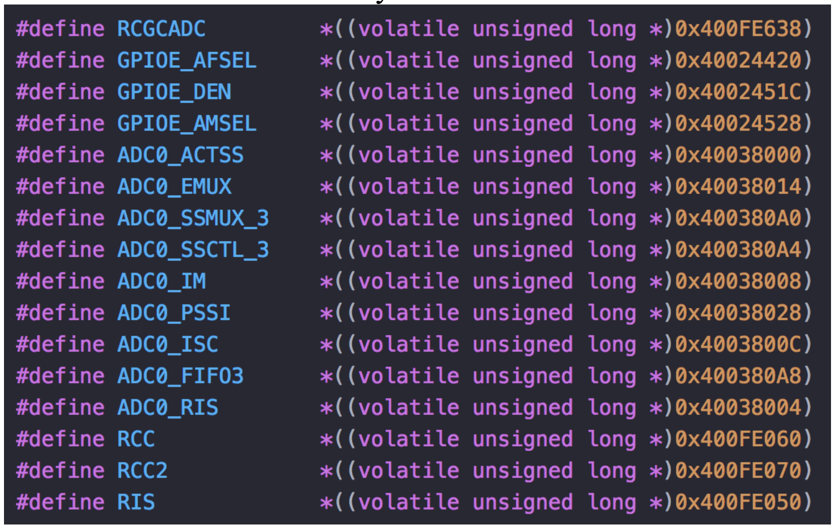
- Analog-to-Digital Converter Run Mode Clock Gating Control(RCGCADC): enable or disable the ADC [p352]
- ADC Active Sample Sequencer (ADCACTSS): enable or disable the sample sequencer [p821]
- ADC Event Multiplexer Select (ADCEMUX): set what even can trigger the ADC [p833]
- ADC Sample Sequence Control 3 (ADCSSCTL3): set sequencer#3's mode: differential or single end / do or do not sense the internal temperature sensor [876]
- ADC Sample Sequence Input Multiplexer Select 3 (ADCSSMUX3): to choose analog input for sequencer3 ADC0 [p875]

- ADC Interrupt Mask (ADCIM): to enable interrupt mask for ADC [p825]
- ADC Interrupt Status and Clear (ADCISC): to clear the interrupt flag for ADC [p828]
- ADC Processor Sample Sequence Initiate (ADCPSSI): to start a new conversion
- ADC Sample Sequence Result FIFO 3 (ADCSSFIFO3): store the converted digital temperature

3. For PLL:

- Run-Mode Clock Configuration (RCC): part 1 of the system clock configuration that will be used to set PLL [p254]
- Run-Mode Clock Configuration 2 (RCC2): part 2 of the system clock configuration that will be used to set PLL [p260]
- Raw Interrupt Status (RIS): to check the interrupt signal, but will not be used in this lab [p244]

And I add these address to my own header file:



```
#define RCGCADC        *((volatile unsigned long *)0x400FE638)
#define GPIOE_AFSEL    *((volatile unsigned long *)0x40024420)
#define GPIOE_DEN      *((volatile unsigned long *)0x4002451C)
#define GPIOE_AMSEL    *((volatile unsigned long *)0x40024528)
#define ADC0_ACTSS     *((volatile unsigned long *)0x40038000)
#define ADC0_EMUX      *((volatile unsigned long *)0x40038014)
#define ADC0_SSMUX_3   *((volatile unsigned long *)0x400380A0)
#define ADC0_SSCTL_3   *((volatile unsigned long *)0x400380A4)
#define ADC0_IM        *((volatile unsigned long *)0x40038008)
#define ADC0_PSSI      *((volatile unsigned long *)0x40038028)
#define ADC0_ISC       *((volatile unsigned long *)0x4003800C)
#define ADC0_FIFO3     *((volatile unsigned long *)0x400380A8)
#define ADC0_RIS       *((volatile unsigned long *)0x40038004)
#define RCC            *((volatile unsigned long *)0x400FE060)
#define RCC2           *((volatile unsigned long *)0x400FE070)
#define RIS            *((volatile unsigned long *)0x400FE050)
```

3.2 Initialize the GPIO

3.2.1 Port F

We need to use the switches in this lab, but this time, we will not hold the switch to keep it in a specific phase, but only press once, which means will change the both edge-sensitive into single-sensitive. To do this, I need to first disable the IBE and set IEV. And I assign ~0x11 to IEV to make it sense the falling edge; for the default state of switches is 1, and when pressed, it goes to zero, which represents a falling edge. Other settings are the same as labs before.

3.2.2 Port E

The Pin#3 in Port E is analog input no.0, and we will use this one, so we need to first enable E port by making sure the 5th bit of RCGCGPIO is set. Then we need to tell the board this pin will be controlled by peripheral; therefore the register port E's AFSEL's third bit, which is corresponding to PE3, should be set. And because we are using PE3 as the analog input, it is necessary to disable the digital function and make it analog which means we need to clear the register DEN's 3rd bit and enable the AMSEL's 3rd bit for PE3.

3.2.3 Initialize ADC

To use ADC we need first enable its clock. We are using ADC0, so the first bit of RCGCADC should be set. Note that we need a small amount of delay in ADC initialization to make sure the initialization can be finished. I created the delay function by setting a large while loop.

Then we need to do the sequencer configuration; before that, we need to disable it first, as we did for configure the timer, to avoid wired bugs. So first clear the third bit of ADC0's ACTSS register and we can start the configuration:

1. First, we need to select a trigger event of sequencer's sample. In this lab, we need to guarantee taht it samples every second, which means we need timer. So, we put 0x5 to the 12th bit of ADC0's EMUX register to make timer as its trigger event.
2. Second, assign an input to ADC, we have already picked one, PE3, that is corresponding to 0x0 of ADC0's SSMUX register.
3. Third, we will do a series of setting for ADC0's SSCTL3 register. We use SSCTL3 because we use sequencer 3(need only one step) in this lab. We need to set the END bit for the sequencer, and it is the 0th bit. And we want to use the interrupt, which means 1st bit should be set. What is more, we are using the internal temp sensor to measure the temperature of the board; so the corresponding 2nd bit should also be set.
4. Fourth, we need to set up the interrupt of ADC. We need to assign 1 to 3rd bit of ADC0's IM register to set up the mask for it. Then we enable the interrupt by setting 17th(ADC0's position in vector table) of NVIC EN0 register.
5. Finally, we enable the sequencer by recover the 3rd bit of ADC0's ACTSS register.

So, ADC has been configured completely and is ready to go.

3.3 Initialize Timer

The timer initialization in this lab is basically the same the one in previous lab, but we should enable the 5th bit of timer0's GPTMCTL to make its output to trigger ADC, so we can make sure the sample frequency is controlled by the timer. And we need to pass in an integer parameter into the function for the load of the timer need to be changed to make sure the timer's interrupt is triggered even if the clock frequency will be changed.

$$\text{cycles(load)} = \frac{time}{period} = \frac{1s}{\frac{1}{frequency}} = |frequency|$$

3.4 Initialization of PLL

In this lab, we will use change the frequency of system clock; therefore, we need PLL.
we need to configure run mode clock. To do this we want to clear the XTAL field, which is the 6 to 10 bit of the (run mode clock configuration) RCC register. We can do it by clearing 0x7C from 4th bit. Then we choose a crystal. I choose 16MHz, by assigning 0x54 to 4th bit. After that we override RCC with RCC2 by setting the last bit of RCC2 register. And then I will do series of setting to RCC2 but before that RCC2, we need to avoid the access to it, which means we need to set the bypass2. I do this by setting the 11th bit of RCC2 register; so now we can start:

1. First, I should choose an oscillator. I choose the main one, which means I need to clear 4th to 5th bit. I do this by make sure the 0x7 from 4th bit are 0.

2. Second, I set 0x4 to 28th bit of RCC2 to apply 400 PLL output.
3. Third, I need to activate PLL by clearing the 13th bit of RCC2
4. Fourth, we need to set our desired clock frequency by resetting the divider. To do this, we need first clear 22th to 23th bit on RCC2 and then assign a parameter to 22th bit. The parameter should be passed into the PLL initialization function as an integer. The two possible number it can be are 4(for 80MHz) and 99(for 4MHz). The equation is as below:

$$frequency = \frac{400MHz}{parameter + 1}$$

5. Finally, we need to clear up the bypass2 to access the PLL. Note that we also need to wait for the PLL to complete the frequency changing. I do this by adding a empty while loop checking the RIS register until I get a 1 on it.

So, now, the PLL has been configured completely and is ready to be used.

3.5 Modification of startup file

We need to use interrupt for ADC, so we need an extra handler for it, and before using it, we need to modify the startup file by adding a handler in vector table, and the corresponding weak and external function for it just as we did in previous labs. The ADC0 handler should be in 17th position in vector table (after system clk).

3.7 Behavior of interrupt

3.7.1 Port F Handler

In this function, I first clear the bit as I did in previous lab. Then I created two variables one is for divider in PLL, one is for the load in timer. Then, using mask, and if else, I set the value for each situation. When sw1 is pressed, I give divider 4 and load 80M's hexadecimal. And else I give divider 99 and the load 4M's hexadecimal. The equations for calculating these numbers are mentioned in section 3.3 and 3.4. Then I call the timer and PLL initialization function to re-initialize them. And pass the variables into them. So, while the system clock frequency is changed, the period for each timer interrupt should be kept as 1s.

3.7.2 Time Handler

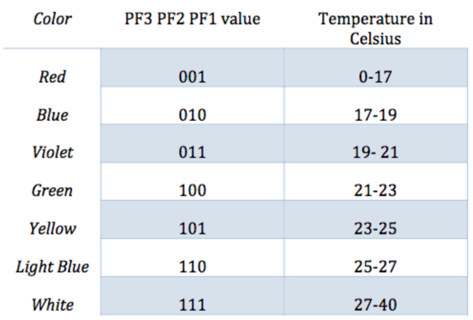
We do not need it any more in this lab.

3.7.3 ADC Handler

When ADC interrupt is trigged, the temperature will be calculated in degree Celsius based on the digital data collected by the ADC, which is stored in FIFO. The equation is shown as below:

$$temperature = 147.5 - 247.5 * \frac{ADC0\_FIFO3}{4096.0}$$

Then, we assign color to LEDs according to the specification, using a series of if else statement. And finally clear the interrupt flag.



Color	PF3 PF2 PF1 value	Temperature in Celsius
Red	001	0-17
Blue	010	17-19
Violet	011	19- 21
Green	100	21-23
Yellow	101	23-25
Light Blue	110	25-27
White	111	27-40

3.8 Main

In my main function, I called all the initialization function once, and add a infinite while loop to keep it working all the time. Note that I choose the 4MHz system clock as my default, so the parameter I put in PLL initialization function is 99, and the one I put in timer initialization function is 4M's hexadecimal number.

4. Result

4.1 Outcomes

The code operates as expected: when switch 1 is pressed, the system clock runs in 80MHz and will heat the board up. When switch 2 is pressed, the system clock runs in 4MHz and will cool board down. The internal temperature sensor senses the temperature change and the LEDs on board show different colors according to the temperature range the temperature calculated lays on. From 0°C to 17 °C, red is on, from 18°C to 19°C, blue is on, from 20°C to 21°C, the blue and red are on, from 22°C to 23°C, the green is on, from 24°C to 25°C, the green and red are on, from 26°C to 27°C, green and blue are on, and from 28°C to 40°C, all LEDs are on.

5.Conclusion

In this lab, I understood what PLL and ADC mean, and how to use them: changing the system clock's running frequency and the collecting analog data, at the same time, converting it into digital result. And I enhance my programming skill on using the interrupt and switches and the ability on digging information from the datasheet. This lab is really rewarding: The outside world is almost all about analog, but I do need to do the digital analysis. This makes ADC a necessary part of my future works. And PLL allows me to get clock frequency more than default ones, which increases my flexibility on approaching my goal.

6.Reference

[1] Tiva™ TM4C123GH6PM Microcontroller DATA SHEET [Online] Available: canvas.uw.edu/courses/1116191/files/folder/Ek-TM4C123GXL?preview=43966944 [Accessed: Nov 6, 2017]