Chen Bai 1560405

EE 474

Lab#5 Report

1.Abstract

In recent lecture, we have discussed about the UART on the Tiva board and its application. UART helps us to build the connection between the board and the outside peripherals, such as computer, and sensors. In this lab, we are going to apply this knowledge and extend our previous lab. The goal is to show the temperature measured and calculated on computer through software named “Putty”. We can do this by configuring the transmitter and assigning data to it. At the same time, we should make the keys on the key board to function as the switches in previous labs to change the system clock frequency, to do this, we should configure the receiver and read the data on it; what is more, because we are going to replace the switches with keyboard, the interrupt of UART is needed. The interrupt does basically the same jobs as the Port F interrupt, while we need to do some small extension to it. I successfully finish the task: the program runs as expectation. When I press “H” on key board, the clock runs at 80MHz, and increase the temperature. When I press any other key on the key board, the clock goes down to 4MHz, and cool the board down. The temperature is shown as integer on screen through Putty. The internal sensor senses the change on temperature as it did in lab before, and LEDs on board changes color accordingly.

2.Introduction

To use Tiva board’s UART, we need to first configure it. But first, we need to know which UART we want to use. In this lab, we are connecting the computer with our board through the USB cable, which means UART0 is needed; for it is designed for the debugging port. That means we need to first activate the UART module and the port A, where the UART0 is. To show the temperature on the screen, we need to transmit things to the peripherals; therefore, we need to do the general configuration for transmitter. And we are using, the interrupt and receiver for the keyboard and board communication, which means we need configuration of receiver for both the general and the interrupt configuration and also modify the startup file as we did for other interrupt. The clock we use for UART is the same as the system clock; so, the rate should be able to change; therefore, the register for setting clock rate should set as variable. Note that we should disable the UART before doing the configuration just as we did for ADC and Timer, and re-enable it after we finish the configuration. The UART handler for the interrupt, basically does the same things as the port F handler, and we need to re-initialize the UART as well, for its clock is changeable.

3.Procedure

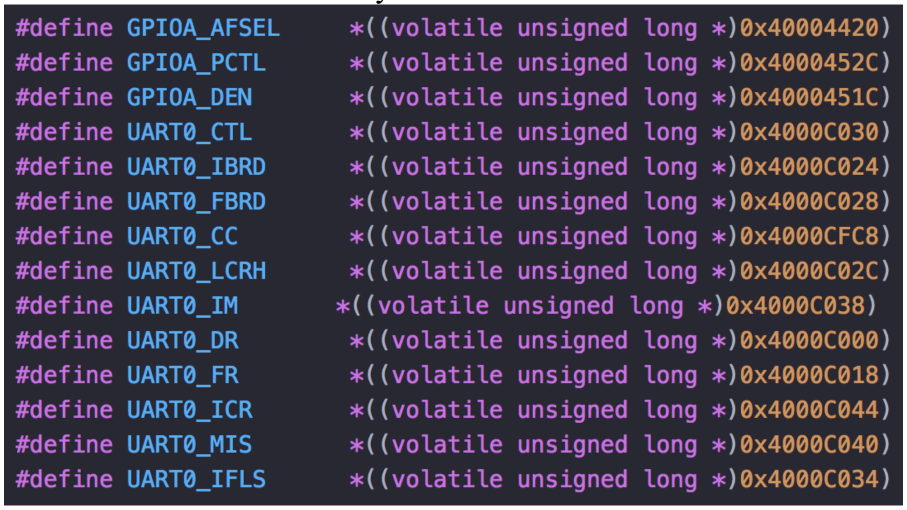
3.1 Checking the data sheet[1]

To use UART we need some need address for the register.

- Universal Asynchronous Receiver/Transmitter Run Mode Clock Gating Control (RCGCUART): for enable or disable the UART module in a whole picture [p344]
- GPIO Port Control (GPIOPCTL): make specific pins to be controlled by peripherals [p688]
- UART Control (UARTCTL): disable or enable certain UART [918]
- UART Integer Baud-Rate Divisor (UARTIBRD): the integer part of baud rate divisor value [914]
- UART Fractional Baud-Rate Divisor (UARTFBRD): the fraction part of baud rate divisor value [915]
- UART Clock Configuration (UARTCC): set the source for setting the UART clock. [939]
- UART Interrupt Mask (UARTIM): set the interrupt mask for UART [p924]
- UART Interrupt Clear (UARTICR): for clearing the interrupt signal [p933]
- UART Line Control (UARTLCRH): for setting the data length, parity, and stop bit. [p916]

- UART Masked Interrupt Status (UARTMIS): make UART to be interrupted by FIFO level [p930]
- UART Interrupt FIFO Level Select (UARTIFLS): to define the FIFO level to trigger the interrupt [p922]

And I add these address to my own header file:



```
#define GPIOA_AFSEL     *((volatile unsigned long *)0x40004420)
#define GPIOA_PCTL      *((volatile unsigned long *)0x4000452C)
#define GPIOA_DEN       *((volatile unsigned long *)0x4000451C)
#define UART0_CTL       *((volatile unsigned long *)0x4000C030)
#define UART0_IBRD      *((volatile unsigned long *)0x4000C024)
#define UART0_FBRD      *((volatile unsigned long *)0x4000C028)
#define UART0_CC        *((volatile unsigned long *)0x4000CFC8)
#define UART0_LCRH      *((volatile unsigned long *)0x4000C02C)
#define UART0_IM       *((volatile unsigned long *)0x4000C038)
#define UART0_DR        *((volatile unsigned long *)0x4000C000)
#define UART0_FR        *((volatile unsigned long *)0x4000C018)
#define UART0_ICR       *((volatile unsigned long *)0x4000C044)
#define UART0_MIS       *((volatile unsigned long *)0x4000C040)
#define UART0_IFLS      *((volatile unsigned long *)0x4000C034)
```

3.2 Initialize the GPIO

Port A

We are using the UART0, whose receiver is on PA0, and the transmitter is PA1. So, I follow the steps below to configure port A:

1. Assign 1 to 0th bit of RCGCGPIO to activate the clock of port A to be RUN mode.
2. Assign 1 to 0th bit (PA0), and 1st bit (PA1) of the GPIOA_AFSEL to make these pins to be controlled by the peripherals
3. After setting the AFSEL, PCTL show be set as well. PCTL selects one out of a set of peripheral functions for each GPIO, providing additional flexibility in signal definition [1]. I assign 1 to the 4th and 0th bits of this register as 4th is corresponding to pin1 and 0th is corresponding to pin2.
4. We also need to set DEN register for GPIOA; for in this lab, we are transmitting and receiving digital information.

3.3 Initialize UART

If we want to use UART, we should first activate the UART module in the whole picture by enabling the clock of UART. To do this we assign 1 to the 0th bit of RCGCUART, which is corresponding to the UART0, which we are using in this lab. Note that the system clock frequency will be changed, which means we also need parameters passed in to indicate baud rate, whose setting is based on system clock frequency. Then we will do a series of specific setting for UART0 (make sure the UART is disabled before further configuration, by clearing the 0th bit of UART0_CTL):

1. We need to set the baud rate for UART. And we need to use two registers to configure the baud rate, one is for integer part and one is for fraction part. And because the system clock frequency is can be changed, the value we assign to baud rate registers should also be variable. I assign UART0_IBRD with “int ibrd” and UART0_FBRD with “int fbrd”.
2. Then we will set the data length, parity and number of stop bits. Information about them are all from the device manager: data length-8, parity-none, and stop bit-1. So, I assign

0x5 to 5th bit of UART0_LCRH to set the data length to be 8, and clear the 3rd bit to make stop bit number to be 1. The parity is none by default.

3. We need to give a source of clock to UART. As I mentioned before, we use system clock as the source. I do this by assign 0x0 to UART0_CC.

Here we finish the general configuration of UART0. Then we will configure the interrupt part:

1. Set up the interrupt mask for UART0's receiver. I do this by sending 1 to the 4th bit of UART0_IM register.
2. Make UART interrupt is trigged by FIFO level by setting the 4th bit of UART0_MIS.
3. Set the FIFO level that will trigger the interrupt. I set it as default (1/2 full) by sending 0x2 to 3rd bit.
4. Setting NVIC_EN0's 5th bit to enable the interrupt.

And now we finish the configuration of UART0. Now we can enable the UART0 to use it. To do this, I assign 1 to 0th bit (enable), to 8th bit (enable receiver), and to 9th bit (enable transmitter).

3.4 Modification of startup file

We need to use interrupt for UART0, 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 UART0 handler should be in 5th position in vector table (after system clk).

3.5Behavior of interrupt

3.5.1 ADC Handler

This handler will do the same things as previous lab: collect the analog data, and calculate the temperature. And I extend this function, made it print the temperature on the putty. I do this by calling my "prints" function. In my prints function: I passed in a char pointer as parameter and check the pointer. While the pointer is not pointing to nothing, I will print the char by calling my "printc" function. And increment the pointer to make it point to the next char. In my printc function: I pass in a char as the parameter. I first check if the transmitter is full by masking the UART0_FR with 1 on the 5th bit. If it is full, I will make it do the empty while loop until there is nothing in it, so now I can assign the desired char(parameter) to the UART0_DR(data register). In summary, my prints and printc functions works together to print the chars one by one to the putty. So, they will help me to put the temperature on putty. Note that we need to cast the double temperature into integer and convert it into an array of chars. I do the conversion by using the "sprintf" function.

3.5.2UART0 handler

This handler replaces the PF handler and does most of its jobs. But we need to do some changes to it:

1. First, create a variable named "mode" in type of char for reading the data from UART0_DR.
2. Second, create two variables named "ibrd" and "fbrd", which will be used for re-initialize the UART. The values that will be assigned in each situation are calculated as below:

BRD = BRDI + BRDF = UARTSysClk / (ClkDiv * Baud Rate)

UARTSysClk means the clock rate we want to use for UART (one is 80MHz and one is 4MHz), and ClkDiv is one of the constants provided, and I choose 16. Baud Rate, according to the device manager, is 9600. My calculation processes are shown below:
For 4MHz:
BRD = 4,000,000 / (16 * 9600) = 26.0417
UARTFBRD[DIVFRAC] = integer(0.417 * 64 + 0.5) = 3

For 80MHz:
BRD = 80,000,000 / (16 * 9600) = 520.833
UARTFBRD[DIVFRAC] = integer(0.417 * 64 + 0.5) = 54
Therefore, the possible value for ibrd are 26 and 520, and the possible value for fbrd are 3 and 54, corresponding to 4MHz and 80MHz system clock frequency respectively.

3. Change the condition for from checking sw1 to check if "mode" contains the char we want. I set "h" as the char for increase the frequency and any other key as the char for decrease the frequency.
4. And we do not need clear the PF handler interrupt, instead, we need to clear the UART0 interrupt by assigning 1 to the 4th bit (receiver) of UART0_ICR.

3.7 Main

In my main function, I called all the initialization function once, and add an 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 UART initialization function are 26(ibrd) and 3(fbrd).

4. Result

4.1 Outcomes

The code operates as expected: when "H" on keyboard is pressed, the system clock runs in 80MHz and will heat the board up. When any other key on keyboard 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. And the temperature is shown on putty and update every second in integer.

4.2 Problems

I had a problem that, it is impossible for me to send a whole sentence to the putty, because in every sample(second), only first two chars of what I send will be read.

5.Conclusion

In this lab, I understood what is UART, and the basic application of it, building communication between board and PC. This is a really rewarding lab because in embedded system, it is impossible to avoid communicate with and control the outside peripherals. By finishing this lab, I grasp the basic skill to operate with UART which will definitely help me in my upcoming project because we will apply Bluetooth module, for which UART1 will be used. What is more, I enhanced my understanding on interrupt setting and digging information from datasheet further more.

<u>6.Reference</u>

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