TA Andrew Gerber 11/10/2023

Lab 8 Report

Benjamin Meads
A02323437
Nick Templeton
A02268123
Richard Snider
A02298443

Objectives

The purpose of this lab is more to familiarize ourselves with ADC's and the UART protocol and put our knowledge to use with a very applicable example, such as using the temperature sensor on the board.

Procedure

This lab started with us scratching our heads while reading the textbook trying to find a good starting point. Our lab TA guided us to the beginning code snippets found in the book in order to get the UART working. We decided that it was best to just attempt to build a working UART system first that took our input and spat it back out to us. We took the given code in the book and modify it to the pins described in the lab instructions. This included changing the pins to PA2 and PA3. This didn't take much time. And with a little more logic and editing. We were able to get the UART system talking to the microcontroller. Just repeating what input we put in. We then made a quick modification to only accept the letters "t" and "T".

The next part was to figure out how to get the ADC to work correctly. This took a lot more reading of the book and searching the technical document. Eventually, we found an 18+ step process on how to get the ADC to set up and read from the ADC. We had to read what the instructions were, and understand what they were asking for. Then to find what register corresponded with the instruction. This was what I would consider to be the bulk amount of time spent on the lab. It wasn't easy to correctly assign each register. We also found out after the fact that a lot of these instructions were unnecessary due to the board already being configured this way.

Once we got the instructions complete and held the value returned by the ADC in a register. We quickly integrated it to be part of our UART function which made it easier to see the value without stepping through the program. Our issue was that the value wasn't correct. It was responding with a negative value. While the lab room is cooler than most rooms, it wasn't that low. After a bit more digging, we found that our program was correct. However, the book didn't keep in mind that the calibration values used in the equation were different, and could be off on each board. So after adjusting the equation, we were able to read reasonable values from the ADC that would change when you put your finger on the chip. This completed our lab and we were able to pass off the lab with Dr. Phillips.

Results

Everything worked as intended. We were able to read values from the ADC starting at around 20 degrees Celsius. Which corresponds with the average room temperature. Then we placed our finger on the chip, and the ADC responded with values that reflected that it was getting warmer. We could only read this data by pressing either "t" or "T" on the keyboard to allow UART to display the data.

Figures

Our code below is our only figure.

Conclusion

In this lab, we figured out how to correctly initialize and set up both a functioning UART system, as well as a ADC temperature value. It was helpful to understand these communication protocols so we can use them on future projects. As well as understanding beyond the theory of an Analog to Digital Converter. To be able to see these things work in practice allows us to more fully understand the process of how these things work.

Programs

Main.c

```
#include "stm32l412xx.h"
```

```
#include "USART.h"
#include <stdio.h>
#include <string.h>

#define TS_CAL1 *((uint16_t*)0x1FFF75A8)
#define TS_CAL2 *((uint16_t*)0x1FFF75CA)

int main(void)
{
	//enable clock for GPIOA
	RCC->AHB2ENR |= RCC_AHB2ENR_GPIOAEN;

	//maybe change to PA2 and PA3
	GPIOA->MODER &= ~(0xFu << (2*2)); //clear mode bits for pin 6 and 7
	GPIOA->MODER |= 0xA << (2*2); //Select alternate function mode

	//Alternative function 7 = USART1
	//Appendix I (in book) shows all alternate functions
	GPIOA->AFR[0] |= 0x77 << (4*2); //set pin 6 and 7 to AF7

	//GPIO speed: 00=low, 01=medium, 10=fast, 11=high speed
	GPIOA->OSPEEDR |= 0xF<<(2*2);

	//GPIO push-pull: 00=no pull, 01=pullup, 10= pulldown, 11=reserved
	GPIOA->PUPDR &= ~(0xFu<<(2*2));
	GPIOA->PUPDR |= 0x5<<(2*2); //select pull-up

	//GPIO Output type: 0 = push-pull, 1 = open drain
	GPIOA->OTYPER &= ~(0x3u<<2);

	//some stuff look @ pg535 of book

	RCC->APB1ENR1 |= RCC_APB1ENR1_USART2EN; //enable USART2 clock

	//Select system clock (SYSCLK) USART clock source of UART 1 and 4
	//00 = PCLK, 01 = SYSCLK, 10 = HSI16, 11 = LSE
	RCC->CCIPR &= ~ (RCC_CCIPR_USART2SEL);
	RCC->CCIPR |= (RCC_CCIPR_USART2SEL_0);

	USART_init(USART2);
	ADC_init();

	uint8_t input[10];
	char buffer[10];
	while(1) {
		USART_Read(USART2, input, 1);
		if (input[0] == 't' | input[0] == 'T')
```

```
		{
			uint32_t idata = ADC_Read();
			double data = (double) idata;
			data *= 3.3/3.0;

			data = (110.0 - 30.0)/(TS_CAL2 - TS_CAL1) * (data -
TS_CAL1) + 52;

			snprintf(buffer, 10, "%2.2f\r\n", data);
			USART_Write(USART2, (uint8_t*) buffer, strlen(buffer));
		}
	}
}
```

USART.h

```
#ifndef USART_H
#define USART_H
#include "stm32l412xx.h"

void USART_init(USART_TypeDef * USARTx);
void USART_Read (USART_TypeDef *USARTx, uint8_t *buffer, uint32_t nBytes);
void USART_Write (USART_TypeDef *USARTx, uint8_t *buffer, uint32_t
nBytes);
void ADC_init(void);
uint32_t ADC_Read(void);
#endif
```

USART_init.c

```
#include "stm32l412xx.h"

#include "USART.h"

void USART_init(USART_TypeDef * USARTx)
{
	//disable USART
	USARTx->CR1 &= ~USART_CR1_UE;

	//Set data length to 8 bits
	//00 = 8 data bits, 01 = 9 data bits, 10 = 7 data bits
	USARTx->CR1 &= ~USART_CR1_M;

	//Select 1 stop bit
```

```
	//00 = 1 stop bit, 01 = 0.5 stop bit, 10 = 2 stop bit, 11 = 1.5 stop
bit
	USARTx->CR2 &= ~USART_CR2_STOP;

	//Set parity control as no parity
	//0 = no parity, 1 = parity
	USARTx->CR1 &= ~USART_CR1_PCE;

	//Oversampling by 16
	//0 = oversampling by 16, 1 = oversampling by 8
	USARTx->CR1 &= ~USART_CR1_OVER8;

	//Set Baud rate to 9600 by using APB freq (80MHz)
	USARTx->BRR = 417;

	//enable transmission and reception
	USARTx->CR1 |= (USART_CR1_TE | USART_CR1_RE);

	//enable USART
	USARTx->CR1 |= USART_CR1_UE;

	//Verify that USART is ready for transmission
	while ((USARTx->ISR & USART_ISR_TEACK) == 0);

	//Verify that USART is ready for reception
	while ((USARTx->ISR & USART_ISR_REACK) == 0);

}

void USART_Read (USART_TypeDef *USARTx, uint8_t *buffer, uint32_t nBytes)
{
	for(uint32_t i = 0; i < nBytes ; i++)
	{
		while(!(USARTx->ISR & USART_ISR_RXNE));
		buffer[i] = (USARTx->RDR) & 0xFF;
	}
}

void USART_Write (USART_TypeDef *USARTx, uint8_t *buffer, uint32_t nBytes)
{
	for(uint32_t i = 0; i < nBytes ; i++)
	{
		while(!(USARTx->ISR & USART_ISR_TXE));
		USARTx->TDR = buffer[i] & 0xFF;
	}

	while (!(USARTx->ISR & USART_ISR_TC));
```

```
	USARTx->ICR |= USART_ICR_TCCF;
}


//incomplete from here down
void ADC_init()
{
	// 1.
	RCC->AHB2ENR |= RCC_AHB2ENR_ADCEN;
	RCC->APB2ENR |= RCC_APB2ENR_SYSCFGEN;
	// 2.
	ADC1->CR &= ~ADC_CR_ADEN;

	// 3.
	SYSCFG->CFGR1 |= SYSCFG_CFGR1_BOOSTEN;

	// 4.
	ADC12_COMMON->CCR |= ADC_CCR_VREFEN;

	// 5.
	ADC12_COMMON->CCR &= ~ADC_CCR_PRESC;

	// 6.
	ADC12_COMMON->CCR &= ~ADC_CCR_CKMODE;
	ADC12_COMMON->CCR |= ADC_CCR_CKMODE_0;

	// 7.
	ADC12_COMMON->CCR &= ~ADC_CCR_DUAL;

	// 8.
	// exit deep pwd
	if ((ADC1->CR & ADC_CR_DEEPPWD) == ADC_CR_DEEPPWD)
	{
		ADC1->CR &= ~ADC_CR_DEEPPWD;
	}

	//Set the CH17SEL bit in the ADCx_CCR register to wake up temp
sensor
	ADC1->CR |= ADC_CR_ADVREGEN;
	int wait_time = 80;
	while (wait_time != 0)
	{
		wait_time--;
	}

	ADC12_COMMON->CCR |= ADC_CCR_TSEN;
```

```
	// 9.
	ADC1->CFGR &= ~ADC_CFGR_RES;

	// 10.
	ADC1->CFGR &= ~ADC_CFGR_ALIGN;

	// 11.
	ADC1->SQR1 &= ~ADC_SQR1_L;

	// 12.
	ADC1->SQR1 |= (17U << 6);

	// 13.
	ADC1->DIFSEL &= ~ADC_DIFSEL_DIFSEL_17;

	// 14.
	ADC1->SMPR2 &= ~ADC_SMPR2_SMP17;
	ADC1->SMPR2 |= ADC_SMPR2_SMP17_1 | ADC_SMPR2_SMP17_0;

	// 15.
	ADC1->CFGR &= ~ADC_CFGR_CONT;

	// 16.
	ADC1->CFGR &= ~ADC_CFGR_EXTEN;

	// 17.
	ADC1->CR |= ADC_CR_ADEN;

	// 18.
	while ((ADC1->ISR & ADC_ISR_ADRDY) != ADC_ISR_ADRDY);
}

uint32_t ADC_Read()
{
	ADC1->CR |= ADC_CR_ADSTART;

	while (!(ADC12_COMMON->CSR & ADC_CSR_EOC_MST));

	return ADC1->DR;
}
```