John DeBoer
Sereen Benchohra

A7 - Analog to Digital Converter

Sample Time Measurements

Sample Time	Max Voltage (V)	Min Voltage (V)	Average Voltage (V)
2.5 clocks	1.50	1.49	1.49
47.5 clocks	1.50	1.49	1.50
640.5 clocks	1.50	1.50	1.50

C Code

main.c

```c
#include "main.h"
#include "ADC.h"
#include "UART.h"

// value used to store ADC input
volatile uint16_t ADC_sample;
// flag set when new ADC value is converted
uint8_t ADC_flag = 0;

void SystemClock_Config(void);

int main(void)
{
	HAL_Init();
	// 80MHz clock
	SystemClock_Config();
	__enable_irq();
	USART_init();
	ADC_init();

	// display UART interface
	USART_print("Max: 0.00V");
	USART_ESC_code("[2;0H"); // move cursor to beginning of second line
	USART_print("Min: 0.00V");
	USART_ESC_code("[3;0H"); // move cursor to beginning of third line
	USART_print("Avg: 0.00V");

	// initialize array to hold 20 ADC samples
	uint16_t ADC_samples[20];
	// variable holds current ADC sample count
	uint8_t count = 0;

	uint16_t max_sample;
	uint16_t min_sample;
```

John DeBoer
Sereen Benchohra

```
uint16_t avg_sample;

while (1)
{
	// handles ADC sample
	if (ADC_flag) {
		// save current ADC sample to array of samples
		ADC_samples[count] = ADC_sample;
		// increment sample counter
		count++;
		// calculates min, max, avg ADC sample values
		// after 20 samples have been collected
		if (count  == 20) {
			count = 0;
			uint32_t total = 0;
			max_sample = ADC_samples[0];
			min_sample = ADC_samples[0];
			// calculate min, max, avg sample values
			for (uint8_t i = 0; i < 20; i++) {
				// find max sample value
				if (ADC_samples[i] > max_sample)
					max_sample = ADC_samples[i];
				// find min sample value
				if (ADC_samples[i] < min_sample)
					min_sample = ADC_samples[i];
				// add up all samples to average later
				total += ADC_samples[i];
			}
			// calculate average sample value
			avg_sample = total/20;

			// convert digital values to millivolts
			uint16_t max_volts = ADC_volt_conv(max_sample);
			uint16_t min_volts = ADC_volt_conv(min_sample);
			uint16_t avg_volts = ADC_volt_conv(avg_sample);
			// temp char
			char character = 0;

			// print max value
			USART_ESC_code("[1;6H"); // move cursor to max value location
			// calculate volts character
			// adding '0' converts to ASCII digit
			character = ((max_volts & 0xF000) >> 12) + '0';
			// print volts
			USART_print_char(character);
			USART_ESC_code("[1C"); // move cursor right 1 space
			// calculate 100 millivolts character
			character = ((max_volts & 0x0F00) >> 8) + '0';
			// print 100 millivolts
			USART_print_char(character);
			// calculate 10 millivolts character
			character = ((max_volts & 0x00F0) >> 4) + '0';
			// print 10 millivolts
			USART_print_char(character);
```

John DeBoer
Sereen Benchohra

```
                        // print min value
                        USART_ESC_code("[2;6H"); // move cursor to max value location
                        // calculate volts character
                        // adding '0' converts to ASCII digit
                        character = ((min_volts & 0xF000) >> 12) + '0';
                        // print volts
                        USART_print_char(character);
                        USART_ESC_code("[1C"); // move cursor right 1 space
                        // calculate 100 millivolts character
                        character = ((min_volts & 0x0F00) >> 8) + '0';
                        // print 100 millivolts
                        USART_print_char(character);
                        // calculate 10 millivolts character
                        character = ((min_volts & 0x00F0) >> 4) + '0';
                        // print 10 millivolts
                        USART_print_char(character);

                        // print avg value
                        USART_ESC_code("[3;6H"); // move cursor to max value location
                        // calculate volts character
                        // adding '0' converts to ASCII digit
                        character = ((avg_volts & 0xF000) >> 12) + '0';
                        // print volts
                        USART_print_char(character);
                        USART_ESC_code("[1C"); // move cursor right 1 space
                        // calculate 100 millivolts character
                        character = ((avg_volts & 0x0F00) >> 8) + '0';
                        // print 100 millivolts
                        USART_print_char(character);
                        // calculate 10 millivolts character
                        character = ((avg_volts & 0x00F0) >> 4) + '0';
                        // print 10 millivolts
                        USART_print_char(character);
                    }
                    // clear ADC flag
                    ADC_flag = 0;
                    // begin another conversion
                    ADC1->CR |= (ADC_CR_ADSTART);
            }
        }
}

void ADC1_2_IRQHandler (void) {
        if (ADC1->ISR & ADC_ISR_EOC) {
                // save ADC data register value
                ADC_sample = ADC1->DR;
                // set global ADC flag
                ADC_flag = 1;
        }
}
```

John DeBoer
Sereen Benchohra

ADC.h

```
#ifndef INC_ADC_H_
#define INC_ADC_H_

// 0b000 = 2.5 ADC clock cycles
// 0b100 = 47.5 ADC clock cycles
// 0b111 = 640.5 ADC clock cycles
#define SMP_val 0b000

void ADC_init(void);
uint16_t ADC_volt_conv(uint16_t dig_val);

#endif /* INC_ADC_H_ */
```

ADC.c

```
#include "main.h"
#include "ADC.h"
#include <stdint.h>

void ADC_init(void) {
	// PA0 -> ADC1_INP5

	// enable ADC registers
	RCC->AHB2ENR |= (RCC_AHB2ENR_ADCEN);

	// ADC CLOCK CONFIGURATION *************************
	// change ckmode to run off HCLK/1
	ADC123_COMMON->CCR &= ~(ADC_CCR_CKMODE);
	ADC123_COMMON->CCR |= (0b01 << ADC_CCR_CKMODE_Pos);
	// END ADC CLOCK CONFIGURATION *********************

	// POWER CONFIGURATION *****************************
	// disable ADC deep power down mode
	ADC1->CR &= ~(ADC_CR_DEEPPWD);
	// enable ADC voltage regulator
	ADC1->CR |= (ADC_CR_ADVREGEN);
	// wait at least 20us for vreg start time
	// delay calculated for f_clk = 80MHz
	for (uint16_t i = 0; i < 1600; i++);
	// END POWER CONFIGURATION *************************

	// INPUT MODE CONFIGURATION ************************
	// single-ended mode for PA0 connected to ADC1_INP5
	ADC1->DIFSEL &= ~(ADC_DIFSEL_DIFSEL_5);
	// END INPUT MODE CONFIGURATION ********************

	// CALIBRATION *************************************
	// ensure ADC is disabled
	ADC1->CR &= ~(ADC_CR_ADEN);
	// single-ended mode calibration
	ADC1->CR &= ~(ADC_CR_ADCALDIF);
	// begin calibration
```

John DeBoer
Sereen Benchohra

```c
        ADC1->CR |= (ADC_CR_ADCAL);
        // wait for calibration to complete (ADCAL = 0)
        while (ADC1->CR & ADC_CR_ADCAL);
        // wait at least four clock cycles
        for (uint8_t i = 0; i < 4; i++);
        // END CALIBRATION *********************************

        // ENABLE ADC **************************************
        // clear ADRDY flag (write 1 to clear)
        ADC1->ISR |= (ADC_ISR_ADRDY);
        // set ADC enable bit
        ADC1->CR |= (ADC_CR_ADEN);
        // wait for ADRDY bit to be set
        while (!(ADC1->ISR & ADC_ISR_ADRDY));
        // clear ADRDY bit by writing a 1
        ADC1->ISR |= (ADC_ISR_ADRDY);
        // END ENABLE ADC **********************************

        // SAMPLE TIME CONFIGURATION ***********************
        // sample time for INP5 set to SMP_VAL
        ADC1->SMPR1 &= ~(ADC_SMPR1_SMP5);
        ADC1->SMPR1 |= (SMP_val << ADC_SMPR1_SMP5_Pos);
        // END SAMPLE TIME CONFIGURATION *******************

        // SEQUENCE CONFIGURATION **************************
        // put channel 5 in sequence 1, set length to 1
        ADC1->SQR1 = (ADC1->SQR1 & ~(ADC_SQR1_SQ1 | ADC_SQR1_L)) |
                       (5 << ADC_SQR1_SQ1_Pos);
        // END SEQUENCE CONFIGURATION **********************

        // INTERRUPT CONFIGURATION *************************
        // enable interrupts at end of conversion
        ADC1->IER |= (ADC_IER_EOC);
        ADC1->ISR &= ~(ADC_ISR_EOC);
        // enable ADC1_2 interrupts
        NVIC->ISER[0] = (1 << (ADC1_2_IRQn & 0x1F));
        // END INTERRUPT CONFIGURATION *********************

        // GPIO CONFIGURATION ******************************
        // enable GPIOA registers
        RCC->AHB2ENR |= (RCC_AHB2ENR_GPIOAEN);
        // PA5 analog mode
        GPIOA->MODER |= (GPIO_MODER_MODE0);
        // connect PA5 to ADC INP1 port
        GPIOA->ASCR |= (GPIO_ASCR_ASC0);
        // END GPIO CONFIGURATION **************************

        // begin conversion
        ADC1->CR |= (ADC_CR_ADSTART);
}

// takes digital value from 0-4095 as input
// and outputs hex representation of millivolt value
// ex. 3300 is represented as 0x3300
// this allows for easy bit shifting later on
```

John DeBoer
Sereen Benchohra

```c
// when output is sent over UART
uint16_t ADC_volt_conv(uint16_t dig_val) {
	// conversion equation derived from calibration data
	uint32_t uv_dec = 812 * dig_val + 1410;
	// converts uV decimal to mV decimal
	uint16_t mv_dec = uv_dec / 1000;
	// hex representation of decimal mV
	uint16_t mv_hex = 0;
	// store first digit of decimal value in hex value
	mv_hex = mv_dec % 10;
	// remove first digit of decimal value
	mv_dec /= 10;
	mv_hex += (mv_dec % 10) << 4;
	mv_dec /= 10;
	mv_hex += (mv_dec % 10) << 8;
	mv_dec /= 10;
	mv_hex += (mv_dec % 10) << 12;
	return mv_hex;
}
```

John DeBoer
Sereen Benchohra

UART.h

```
#ifndef INC_UART_H_
#define INC_UART_H_

#define TX_pin GPIO_PIN_2
#define RX_pin GPIO_PIN_3

#define BAUD_115200 694

#define CHAR_ESC 0x1B

void USART_init(void);
void USART_print(char string[]);
void USART_print_char(uint8_t frame);
void USART_ESC_code(char string[]);

#endif /* INC_UART_H_ */
```

UART.c

```
#include "main.h"
#include "UART.h"
#include <stdint.h>

void USART_init(void) {
	// TX -> PA2
	// RX -> PA3

	// enable USART2 interrupts
	NVIC->ISER[1] = (1 << (USART2_IRQn & 0x1F));
	// enable GPIOA
	RCC->AHB2ENR   |=  (RCC_AHB2ENR_GPIOAEN);
	// enable USART2 registers
	RCC->APB1ENR1   |=  (RCC_APB1ENR1_USART2EN);
	// initialize PA2 as TX, PA3 as RX
	GPIOA->MODER   &= ~(GPIO_MODER_MODE2  | GPIO_MODER_MODE3);
	GPIOA->MODER   |=  (GPIO_MODER_MODE2_1  |
	               GPIO_MODER_MODE3_1); // alternate function mode
	GPIOA->PUPDR   &= ~(GPIO_PUPDR_PUPD2); // no pull-up/pull-down on TX
	GPIOA->PUPDR   |= (GPIO_PUPDR_PUPD3_1); // pull-up on RX
	GPIOA->OTYPER  &= ~(GPIO_OTYPER_OT2 | GPIO_OTYPER_OT3); // push-pull
	GPIOA->OSPEEDR |= (GPIO_OSPEEDR_OSPEED2 |
	               GPIO_OSPEEDR_OSPEED3); // high speed
	GPIOA->AFR[0] |= ((0x7 << GPIO_AFRL_AFSEL2_Pos) |
	               (0x7 << GPIO_AFRL_AFSEL3_Pos)); // AF7 = USART2

	// frame size 9 to include stop bit
	//USART2->CR1 |= (USART_CR1_M0);
	// oversampling by 16
	//USART2->CR1 &= ~(USART_CR1_OVER8);
	// baud rate = 115.2 kbps
	USART2->BRR = (BAUD_115200);
	// enable USART2
```

John DeBoer
Sereen Benchohra

```
        USART2->CR1 |= (USART_CR1_UE);
        // enable TX and RX
        USART2->CR1 |= (USART_CR1_TE | USART_CR1_RE);
        // enable TX empty interrupt
        // so the program knows when to send the next byte
        //USART2->CR1 |= (USART_CR1_TXEIE);
        // enable RX empty interrupt
        // so the program knows when to receive the next byte
        //USART2->CR1 |= (USART_CR1_RXNEIE);
        USART_ESC_code("[2J"); // clear entire screen
        USART_ESC_code("[H"); // cursor top left
}

void USART_print(char string[]) {
        uint8_t i = 0;
        // send characters in string until null char is reached
        while (string[i] != '\0') {
                USART_print_char(string[i]);
                i++;
        }
}

void USART_ESC_code(char string[]) {
        USART_print_char(CHAR_ESC);
        uint8_t i = 0;
        // send characters in string until null char is reached
        while (string[i] != '\0') {
                USART_print_char(string[i]);
                i++;
        }
}

void USART_print_char(uint8_t frame) {
        // wait for TXE register to be set
        // indicating TDR is ready to queue frame
        while ((USART2->ISR & USART_ISR_TXE)==0);
        //GPIOA->BRR |= (TX_pin);
        // load TDR register with frame to be sent
        USART2->TDR = (frame);
}
```