// Lab 10: Analog to Digital Converter (ADC)

// Lab Purpose 🡪 Familiarizing the user with Analog to Digital Converter (ADC)!

// ADC + Successive Approximation Register (SAR) with Charge Distribution

// Using ADC 🡪 We can familiarize the user in the user in the usage of two-dimensional joystick on the Educational BoosterPack!

// **10.1: Using the ADC SAR-Type.**

// Learning about an ADC!

// ADC 🡪 A component of the MCU that can convert an analog input signal to a binary number!

// Calculating the result into a 10-bit binary number 🡪 Using the input signal that falls between two reference voltages (i.e. one upper and one lower)!

// Learning the various configuration with the aforementioned voltages that ends up being controlled by Equation 1.



// N 🡪 the full range value (10-bit) that happens when Vin is equal to Vr+.

// SAR ADC 🡪 It produces the n-bit result by doing voltage comparisons.

// Finding a sample-an-hold period that is usually followed by the conversion.

// Our ranges for CI 🡪 10pF to 15pF!

// Achieving 3us | choosing upper values (CI = 15pF RI = 10Kohm)

// Opted for fastest conversion possible!

// Obtaining by dividing the MODOSC clock signal by one!

// Multiplying the MODOSC clock frequency (ranging between 4 to 5.4 MHz)!

// Multiplying 5.4 MHz by 3 us + the result of 16n clock cycles!

// Completing the provided skeleton of initializing ADC function and using the UART through Tera Term 🡪 Displaying the values of the horizontal axis of the Joystick.

// Toggling the red LED to indicate continuous activity!

// Setting a delay loop that sets an interval between the readings of about 0.5 seconds.

// Starting the ADC conversion 🡪 Setting the ADC12SC bit inside the for loop and waiting for the ADC12BUSY bit to get clear!

// ADC12SC bit 🡪 Setting inside the ADC12CTL0 register variable! + The ADC12BUSY is checked inside the ADC12CTL1 register variable!

// Using the ADC12MEM0 and storing it in an integer data type variable.

#include <msp430.h>

#include <stdio.h>

#define redLED BIT0 // red LED location of the MCU P1.0

#define FLAGS UCA1IFG // Variable that possesses Transmit & Receive Flags of UART

#define RXFLAG UCRXIFG // Receive flag of UART

#define TXFLAG UCTXIFG // Transmit flag of UART

#define TXBUFFER UCA1TXBUF // Transmit buffer contains the transmit byte of UART

#define RXBUFFER UCA1RXBUF // Receive buffer contains the received byte of UART

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Reconfigures ACLK to be rerouted to the 32 KHz crystal on the LaunchPad

void config\_ACLK\_to\_32KHz\_crystal() {

// The default mode of the ACLK is a built-in oscillator at a frequency of 39KHz normally.

// Rerouted the pins to LFXIN/LFXOUT functionality so that the ACLK can be routed to the 32KHz crystal.

// This information can be found using the LaunchPad user's guide (page 29) and the chip's data sheet (page 123).

PJSEL1 &= ~BIT4;

PJSEL0 |= BIT4;

// We need to for the crystal to settle, once it has started.

// Therefore, we will wait until the local and global oscillator fault flags are cleared and remain cleared.

CSCTL0 = CSKEY; // Unlock CS registers, to divert the pins for the crystal functionality.

//Clears the flag and will do so until they remain cleared.

do {

CSCTL5 &= ~LFXTOFFG; // Local oscillator fault flag

SFRIFG1 &= ~OFIFG; // Global oscillator fault flag

} while((CSCTL5 & LFXTOFFG) != 0);

CSCTL0\_H = 0; // Lock CS registers, returns the pins.

return;

}

// Popular Configuration of UART

// 9600 baud, 8-bit data, LSB first, no parity bits, 1 stop bit

// no flow control

// Initial clock: SMCLK @ 1.048 MHz with oversampling

void Initialize\_UART(void){

// Divert pins to UART functionality to enable transmission/reception of data between PC and MCU

P3SEL1 &= ~(BIT4|BIT5);

P3SEL0 |= (BIT4|BIT5);

// Sets the clock source to SMCLK

UCA1CTLW0 |= UCSSEL\_2;

// Configures the dividers and modulators of the clock with the popular configuration of UART

// Configurations can be found in the Family User's Guide Page 783

UCA1BRW = 6;

UCA1MCTLW = UCBRS5|UCBRS1|UCBRF3|UCBRF2|UCBRF0|UCOS16;

// Enables transmission/reception to start

UCA1CTLW0 &= ~UCSWRST;

}

void Initialize\_ADC() {

// Divert the pins to analog functionality for the horizontal axis of the Joystick.

P9SEL1 |= BIT2;

P9SEL0 |= BIT2;

// Turn on the ADC module

ADC12CTL0 |= ADC12ON;

// Turn off ENC (Enable Conversion) bit while modifying the ADC module

ADC12CTL0 &= ~ADC12ENC;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ADC12CTL0 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// We set the cycles to 16 cycles

ADC12CTL0 |= ADC12SHT0\_3;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ADC12CTL1 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Set ADC12SHS to 0 since we are using the ADC12SC bit as the trigger

// Set ADC12SHP to 1 since we are using the SAMPCON signal as sourced from the sampling timer

// ADC12DIV to 0 since we decided to divide the MODOSC clock signal by 1

// ADC12SSEL to 0 since we are using the MODOSC clock signal

ADC12CTL1 |= ADC12SHS\_0 | ADC12SHP | ADC12DIV\_0| ADC12SSEL\_0;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ADC12CTL2 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Set ADC12RES to 2 since we are converting for a 12-bit result

ADC12CTL2 |= ADC12RES\_2;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ADC12MCTL0 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Set ADC12VRSEL to 0 since we are using the VR+ as AVCC and VR- as AVSS

// Set ADC12INCH to 10 since we are using the analog channel A10

ADC12MCTL0 |= ADC12VRSEL\_0 | ADC12INCH\_10;

// Turn on ENC (Enable Conversion) bit at the end of the sconfiguration

ADC12CTL0 |= ADC12ENC;

return;

}

void uart\_write\_char(unsigned char ch){

// Ongoing transmission to be over

while ( (FLAGS & TXFLAG)==0 ) {}

// Write the input byte into the buffer to display in the terminal

TXBUFFER = ch;

}

// Function that takes a 16-bit unsigned integer and transmit it through UART communication

void uart\_write\_uint16(unsigned int n){

volatile unsigned int digit = 0;

int k = 0,l = 0;

int array[5];

// Extracts digit by digit of the input number and stores it in an array

do{

digit = n % 10; // Extracts the digit from the input number

array[k] = digit; // Stores it into an array

n = n/10; // Truncates the input number

k++;

l++;

}while(n != 0);

// Extracts from the array backwardly to transmit the each digit through UART

for(k = l - 1 ; k >= 0;k--)

uart\_write\_char(array[k] + '0');

// New line

uart\_write\_char('\n');

// Carry return

uart\_write\_char('\r');

}

/\*\*

\* main.c

\*/

int main(void)

{

WDTCTL = WDTPW | WDTHOLD; // Stops the watchdog timer. We do this so the MCU doesn't reset itself periodically.

PM5CTL0 &= ~LOCKLPM5; // We enable the general purpose I/O pins.

P1DIR |= redLED; // Sets the LED as an output

P1OUT &= ~redLED; // Starts the LED as off

unsigned int temp; //variable used to store the value of the horizontal axis of the Joystick

// ADC configuration to 16 cycles of sample-and-hold time and MODOSC clock signal @ [4 to 5.4 MHz]

Initialize\_ADC();

// Initializes the UART communication using SMCLK @ 1.048 MHz with oversampling at a baud rate of 9600

Initialize\_UART();

// Configures the ACLK to the 32KHz crystal

config\_ACLK\_to\_32KHz\_crystal();

// Configures the timer to ACLK, up mode and clear TAR.

TA0CTL = TASSEL\_1|MC\_1|TACLR;

// Sets the upperbound of TAR to 16383. This generates a delay of 0.5 seconds with the ACLK @ 32KHz

TA0CCR0 = 16383;

for(;;){

ADC12CTL0 |= ADC12SC; // Starts conversion

while((ADC12CTL1 & ADC12BUSY) != 0){} // Waits for conversion to end

temp = ADC12MEM0; // stores result inside the variable

uart\_write\_uint16(temp); // transmits the result to Tera Term via UART

P1OUT ^= redLED; // Toggles the LED

while((TA0CTL & TAIFG) == 0){} // waits for 0.5 seconds until next reading

TA0CTL &= ~TAIFG; // clears the timer flag for repetition

}

return 0;

}

// **10.2: Reading the Measurements from the Light Sensor**

#include <msp430.h>

#include <stdio.h>

#define redLED BIT0 // red LED location of the MCU P1.0

#define FLAGS UCA1IFG // Variable that possesses Transmit & Receive Flags of UART

#define RXFLAG UCRXIFG // Receive flag of UART

#define TXFLAG UCTXIFG // Transmit flag of UART

#define TXBUFFER UCA1TXBUF // Transmit buffer contains the transmit byte of UART

#define RXBUFFER UCA1RXBUF // Receive buffer contains the received byte of UART

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Reconfigures ACLK to be rerouted to the 32 KHz crystal on the LaunchPad

void config\_ACLK\_to\_32KHz\_crystal() {

// The default mode of the ACLK is a built-in oscillator at a frequency of 39KHz normally.

// Rerouted the pins to LFXIN/LFXOUT functionality so that the ACLK can be routed to the 32KHz crystal.

// This information can be found using the LaunchPad user's guide (page 29) and the chip's data sheet (page 123).

PJSEL1 &= ~BIT4;

PJSEL0 |= BIT4;

// We need to for the crystal to settle, once it has started.

// Therefore, we will wait until the local and global oscillator fault flags are cleared and remain cleared.

CSCTL0 = CSKEY; // Unlock CS registers, to divert the pins for the crystal functionality.

//Clears the flag and will do so until they remain cleared.

do {

CSCTL5 &= ~LFXTOFFG; // Local oscillator fault flag

SFRIFG1 &= ~OFIFG; // Global oscillator fault flag

} while((CSCTL5 & LFXTOFFG) != 0);

CSCTL0\_H = 0; // Lock CS registers, returns the pins.

return;

}

// Popular Configuration of UART

// 9600 baud, 8-bit data, LSB first, no parity bits, 1 stop bit

// no flow control

// Initial clock: SMCLK @ 1.048 MHz with oversampling

void Initialize\_UART(void){

// Divert pins to UART functionality to enable transmission/reception of data between PC and MCU

P3SEL1 &= ~(BIT4|BIT5);

P3SEL0 |= (BIT4|BIT5);

// Sets the clock source to SMCLK

UCA1CTLW0 |= UCSSEL\_2;

// Configures the dividers and modulators of the clock with the popular configuration of UART

// Configurations can be found in the Family User's Guide Page 783

UCA1BRW = 6;

UCA1MCTLW = UCBRS5|UCBRS1|UCBRF3|UCBRF2|UCBRF0|UCOS16;

// Enables transmission/reception to start

UCA1CTLW0 &= ~UCSWRST;

}

void Initialize\_ADC() {

// Divert the pins to analog functionality for the horizontal axis of the Joystick.

P9SEL1 |= BIT2;

P9SEL0 |= BIT2;

// Divert the pins to analog functionality for the vertical axis of the Joystick.

P8SEL1 |= BIT7;

P8SEL0 |= BIT7;

// Turn on the ADC module

ADC12CTL0 |= ADC12ON;

// Turn off ENC (Enable Conversion) bit while modifying the

ADC12CTL0 &= ~ADC12ENC;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ADC12CTL0 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// We set the cycles to 16 cycles

// Set the bit ADC12MSC (Multiple Sample and Conversion)

ADC12CTL0 |= ADC12SHT0\_3 | ADC12MSC;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ADC12CTL1 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Set ADC12SHS to 0 since we are using the ADC12SC bit as the trigger

// Set ADC12SHP to 1 since we are using the SAMPCON signal as sourced from the sampling timer

// ADC12DIV to 0 since we decided to divide the MODOSC clock signal by 1

// ADC12SSEL to 0 since we are using the MODOSC clock signal

// Set ADC12CONSEQ (select sequence-of-channels)

ADC12CTL1 |= ADC12SHS\_0 | ADC12SHP | ADC12DIV\_0| ADC12SSEL\_0| ADC12CONSEQ\_1;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ADC12CTL2 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Set ADC12RES to 2 since we are converting for a 12-bit result

ADC12CTL2 |= ADC12RES\_2;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ADC12CTL3 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Set ADC12CSTARTADD to 0 (first conversion in ADC12MEM0)

ADC12CTL3 |= ADC12CSTARTADD\_0;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ADC12MCTL0 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Set ADC12VRSEL to 0 since we are using the VR+ as AVCC and VR- as AVSS

// Set ADC12INCH to 10 since we are using the analog channel A10

ADC12MCTL0 |= ADC12VRSEL\_0 | ADC12INCH\_10;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ADC12MCTL1 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Set ADC12VRSEL to 0 since we are using the VR+ as AVCC and VR- as AVSS

// Set ADC12INCH to 4 since we are using the analog channel A4

// Set ADC12EOS (last conversion in ADC12MEM1)

ADC12MCTL1 |= ADC12VRSEL\_0 | ADC12INCH\_4 | ADC12EOS;

// Turn on ENC (Enable Conversion) bit at the end of the sconfiguration

ADC12CTL0 |= ADC12ENC;

return;

}

void uart\_write\_char(unsigned char ch){

// Ongoing transmission to be over

while ( (FLAGS & TXFLAG)==0 ) {}

// Write the input byte into the buffer to display in the terminal

TXBUFFER = ch;

}

// Function that takes a 16-bit unsigned integer and transmit it through UART communication

void uart\_write\_uint16(unsigned int n){

volatile unsigned int digit = 0;

int k = 0,l = 0;

int array[5];

// Extracts digit by digit of the input number and stores it in an array

do{

digit = n % 10; // Extracts the digit

array[k] = digit; // Stores it

n = n/10; // Truncates the input number

k++;

l++;

}while(n != 0);

// Extracts from the array backwardly to transmit the each digit through UART

for(k = l - 1 ; k >= 0;k--)

uart\_write\_char(array[k] + '0');

// New line

uart\_write\_char('\n');

// Carry return

uart\_write\_char('\r');

}

// Function that transmit a string (character array) through UART communication

void uart\_write\_string(char \*str){

unsigned int i = 0;

// Transmits character by character in the UART communication until it reaches the NULL terminated

while( str[i] != '\0'){

uart\_write\_char(str[i]);

i++;

}

// New Line

uart\_write\_char('\n');

// Carry Return

uart\_write\_char('\r');

}

/\*\*

\* main.c

\*/

int main(void)

{

WDTCTL = WDTPW | WDTHOLD; // Stops the watchdog timer. We do this so the MCU doesn't reset itself periodically.

PM5CTL0 &= ~LOCKLPM5; // We enable the general purpose I/O pins.

P1DIR |= redLED; // Sets the LED as an output

P1OUT &= ~redLED; // Starts the LED as off

unsigned int temp,temp2; // variable used to store the value of the horizontal axis and vertical axis of the Joystick

char mystring[] = "x-coordinate:"; // indicates the x-coordinate of the joystick

char mystring2[] = "y-coordinate:"; // indicates the y-coordinate of the joystick

// ADC configuration to 16 cycles of sample-and-hold time and MODOSC clock signal @ [4 to 5.4 MHz]

Initialize\_ADC();

// Initializes the UART communication using SMCLK @ 1.048 MHz with oversampling at a baud rate of 9600

Initialize\_UART();

// Configures the ACLK to the 32KHz crystal

config\_ACLK\_to\_32KHz\_crystal();

// Configures the timer to ACLK, up mode and clear TAR.

TA0CTL = TASSEL\_1|MC\_1|TACLR;

// Sets the upperbound of TAR to 16383. This generates a delay of 0.5 seconds with the ACLK @ 32KHz

TA0CCR0 = 16383;

for(;;){

ADC12CTL0 |= ADC12SC; // Starts conversion

while((ADC12CTL1 & ADC12BUSY) != 0){} // Waits for conversion to end

temp = ADC12MEM0; // Stores the horizontal axis value of the Joystick

temp2 = ADC12MEM1; // Stores the vertical axis value of the Joystick

uart\_write\_string(mystring); // Transmit x-coordinate

uart\_write\_uint16(temp); // Transmit the horizontal axis value

uart\_write\_string(mystring2); // Transmit y-coordinate

uart\_write\_uint16(temp2); // Transmite the vertical axis value

P1OUT ^= redLED; // Toggles the LED

while((TA0CTL & TAIFG) == 0){} // waits for 0.5 seconds until next reading

TA0CTL &= ~TAIFG; // clears the timer flag for repetition

}

return 0;

}