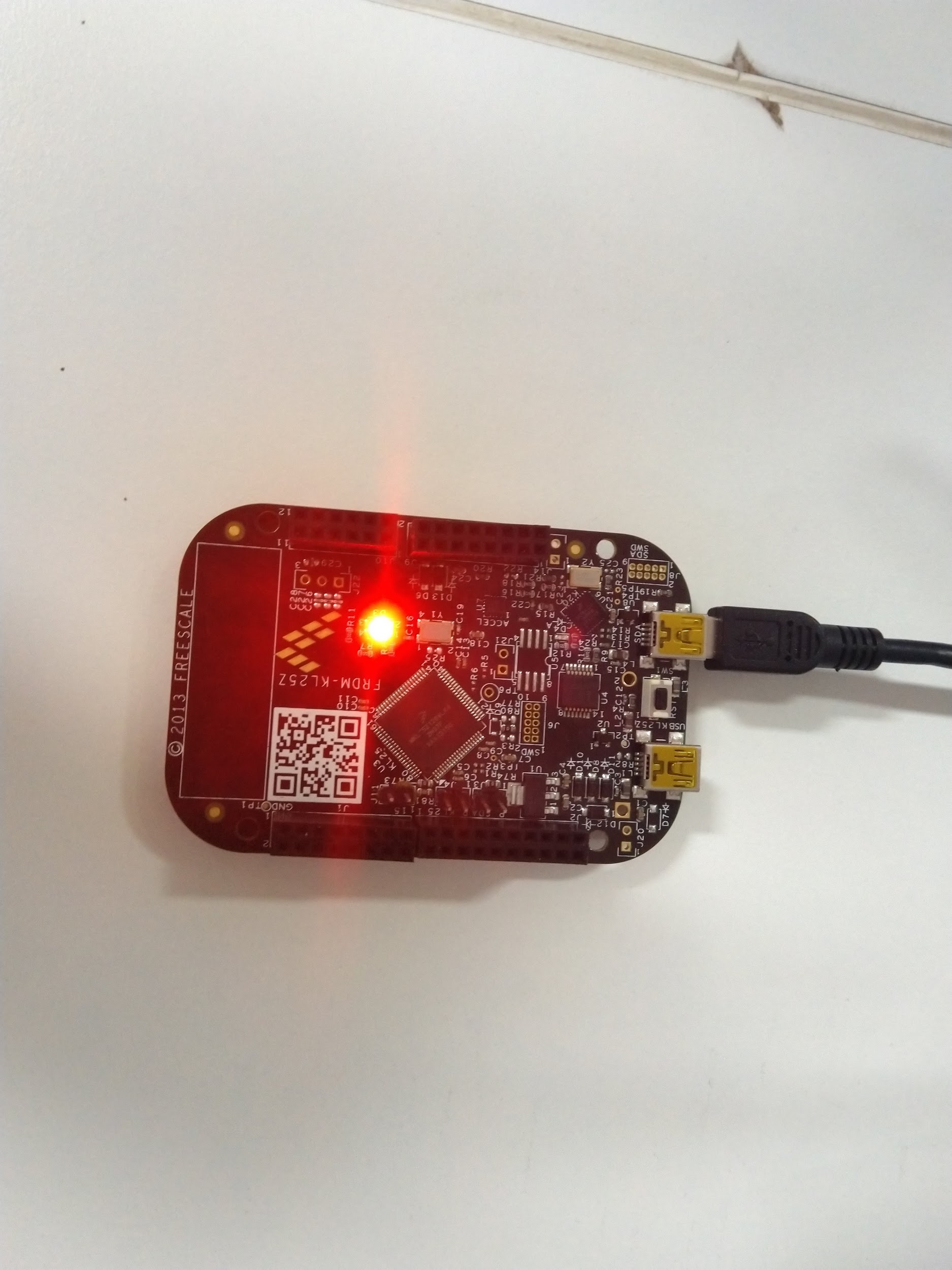
**Lab Report**

****

**Embedded Systems**

**Himanshu Rai**

**111601032 27th September, 2019**

**Objective**

To light up RGB lights on FRDM-KL25Z board, with LED colour determined by input from a potentiometer through ADC.

**Theory**

**Analog to Digital converter**

An analog-to-digital converter (ADC) is a system that converts an analog signal, such as a sound picked up by a microphone or light entering a digital camera, into a digital signal. An ADC may also provide an isolated measurement such as an electronic device that converts an input analog voltage or current to a digital number representing the magnitude of the voltage or current. The conversion involves quantization of the input, so it necessarily introduces a small amount of error or noise. Furthermore, instead of continuously performing the conversion, an ADC does the conversion periodically, sampling the input, limiting the allowable bandwidth of the input signal.

**Using ADC on FRDM-KL25Z**

At first, we need to setup the ADC on the FRDM board to start using it. The following steps describes the setup process.

* First of all we need to disable the clock gating (enable the clock) for the *ADC0* module. For this, the 27th bit in the *SCGC\_6 (system clock gating control register 6)* must be set.
* We would be using *ADC0\_SE9 (single ended analog channel inputs),* which wecan access through *PORT B1.* So, we must also disable the clock gating to *PORT B1,* which can be done by setting the 10th bit in *SCGC\_5* register.
* To configure *PORT B1* for *ADC0\_SE9,* we must set the bit positions 10 to 8 to 001 in its *pin control register (PCR).*
* Next, set ADC0 for software trigger, by setting the 6th bit in *ADC0\_SC2 (status and control register)* to 0. Also, select the voltage reference source used for conversions to default voltage reference pin pair, that is, external pins VREFH and VREFL, by setting its 0th and 1th bits to 0.
* Next, set the *ADC0\_CFG (configuration register)* as *0b01000100.*
  + *\_\_\_\_\_\_00 -* Use bus clock as the input clock
  + \_\_\_\_01\_\_ - Selects the ADC resolution. When DIFF=0, it is single ended 12-bit conversion and when DIFF=1, it is differential 13-bit conversion with 2's complement output.
  + \_\_\_0\_\_\_\_ - Small sample time
  + \_10\_\_\_\_\_ - Selects the divide ratio used by the ADC to generate the internal clock ADCK as 4 and the clock rate is (input clock)/4.
  + 0\_\_\_\_\_\_\_ - Normal power configuration

Now, after the initial setup, we can start reading the input from ADC as follows -

* Set input channel as AD9 for ADC0 by setting *ADC0\_SC1* to 9. This also set the 7th bit which the *COCO flag (conversion complete flag)* to 0.
* Now, wait while the *COCO* flag is not set (ie. no input has been generated).
* When *COCO flag* is set, read the input from from twelve less significant bits in *ADC0\_RA* register (unsigned right justified data).

**Lighting up the LEDs**

The following tables give the correspondence between the LEDs and port pins associated with them

|  |  |
| --- | --- |
| **LED** | **GPIO PIN** |
| Red | PortB18 |
| Green | PortB19 |
| Blue | PortD1 |

First we will have to follow the following procedure to configure the ports/pins corresponding to the LEDs, before using them

* By default, GPIO modules are disabled to save power. So, we should first enable the clocks to the ports corresponding to the LEDs to prevent hardware fault. The *system clocking gate control register 5 (SIM\_SCGC5)* gates the clocks to GPIO ports. We can enable the clock for these ports by setting the respective bits in *SIM\_SCGC5* control registers - which are bit 10 (for port B - red and green LEDs) and bit 12 (for port D - blue LED).
* Next configure the pins for GPIO . For this we must write 001 in the corresponding *program control registers (PCR)* at bit positions 10 to 8.
* Now set the pins as output pins by setting the corresponding pin position in the *port data direction register (PDDR)* of the corresponding ports.

After setting up the pins, we can turn the LEDs on and off by setting the corresponding bits in *port clear output register (PCOR)* and *port toggle output register (PTOR).*

**Procedure**

According to the steps mentioned above, it would suffice to write a function for setting up ADC0, one other for reading ADC outputand three other functions for each LEDs - one to initialise the ports and the other two to switch the LEDs on and off.

The initialisation functions for each LEDs and ADC0 need to be called only once, at the beginning of the program. Also it is safe to call off function for each LED at the beginning.

Next in a loop we would read the data from ADC using function mentioned above and light up the respective LEDs, depending upon the ADC output. Since, ADC output is 12 bits, we can divide it into three intervals and light up an LED corresponding to the interval in which the magnitude of ADC output lies.

The source code is at the end of the report.

**Results**

The desired outputs were obtained and demonstrated during the lab.

**Source Code**

#include<MKL25Z4.h>

**static** **volatile** **short** data = 0;

**void** ADC0\_init() {

// disable clock gating to ADC0 to activate it

SIM->SCGC6 |= (1 << 27);

// activate port B

SIM->SCGC5 |= 1 << 10;

// set the pin as ADC0\_SE9 (PCR[10-8] = 001)

PORTB->PCR[1] &= 0xFFFFF8FF; // set 8th, 9th and 10th bit of PCR[1] to 0

PORTB->PCR[1] |= 1 << 8; // set 8th bit of PCR[1] to 1

// configuring ADC0 for software trigger and default voltage reference pin pair, that is,

//external pins VREFH and VREFL

ADC0->SC2 &= 0xFFFFFFBC; // reset the 6th bit and 0th, 1th bit in ADC0\_SC2 register

// setting configuration register for ADC0 (see manual for details about ADC0\_CFG1)

ADC0->CFG1 = ***0b01000100***;

}

**void** read() {

// setting input channel as AD9 for ADC0

ADC0->SC1[0] = 9;

// check conversion complete (COCO) flag of ADC0\_SC1A to see if data is read

**while**(!(ADC0->SC1[0] & (1 << 7))); // 0 for A

// read the data from ADC0\_RA register (only first 12 bits)

data = ADC0->R[0] & 0xFFF; // 0 of A

}

**void** led\_red\_init() {

SIM->SCGC5 |= 1 << 10; // will activate the port B

// set the pin as gpio

PORTB->PCR[18] &= 0xFFFFF8FF; // set 8th, 9th and 10th bit of PCR[18] to 0

PORTB->PCR[18] |= 1 << 8; // set 8th bit of PCR[18] to 1

// set the port as output port

PTB->PDDR |= 1 << 18; // set the 18th bit of PDDR to 1 for output

}

**void** led\_red\_on() {

// clear the 18th bit of PDOR register

PTB->PCOR |= 1 << 18; // set the 18th bit of PCOR to 1

}

**void** led\_red\_off() {

// toggle the 18th bit of PDOR register

PTB->PTOR |= 1 << 18;

}

**void** led\_green\_init() {

SIM->SCGC5 |= 1 << 10; // will activate the port B

// set the pin as gpio

PORTB->PCR[19] &= 0xFFFFF8FF; // set 8th, 9th and 10th bit of PCR[19] to 0

PORTB->PCR[19] |= 1 << 8; // set 8th bit of PCR[19] to 1

// set the port as output port

PTB->PDDR |= 1 << 19; // set the 19th bit of PDDR to 1 for output

}

**void** led\_green\_on() {

// clear the 18th bit of PDOR register

PTB->PCOR |= 1 << 19; // set the 19th bit of PCOR to 1

}

**void** led\_green\_off() {

// toggle the 18th bit of PDOR register

PTB->PTOR |= 1 << 19;

}

**void** led\_blue\_init() {

SIM->SCGC5 |= 1 << 12; // will activate the port D

// set the pin as gpio

PORTD->PCR[1] &= 0xFFFFF8FF; // set 8th, 9th and 10th bit of PCR[1] to 0

PORTD->PCR[1] |= 1 << 8; // set 8th bit of PCR[1] to 1

// set the port as output port

PTD->PDDR |= 1 << 1; // set the 1th bit of PDDR to 1 for output

}

**void** led\_blue\_on() {

// clear the 18th bit of PDOR register

PTD->PCOR |= 1 << 1; // set the 1th bit of PCOR to 1

}

**void** led\_blue\_off() {

// toggle the 18th bit of PDOR register

PTD->PTOR |= 1 << 1;

}

**void** delay(**int** t) {

**while**(t--);

}

**int** main() {

SystemCoreClockUpdate(); // updating the clock from PLL

ADC0\_init();

led\_red\_init(); led\_red\_off();

led\_blue\_init(); led\_blue\_off();

led\_green\_init(); led\_green\_off();

**while**(1) {

read();

**if**(data > (1 << 13) / 3) led\_red\_on(), delay(100000), led\_red\_off();

**else** **if**(data > (1 << 12) / 3) led\_blue\_on(), delay(100000), led\_blue\_off();

**else** led\_green\_on(), delay(100000), led\_green\_off();

}

}