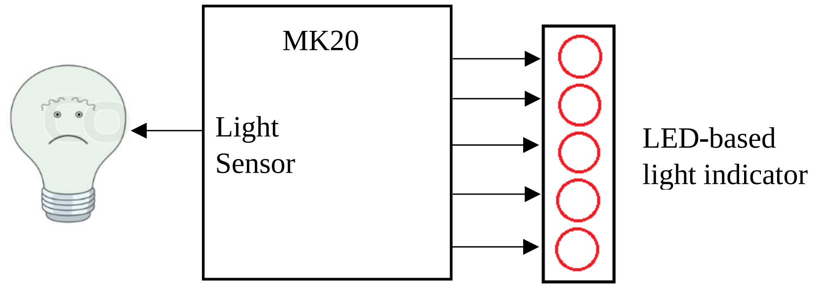
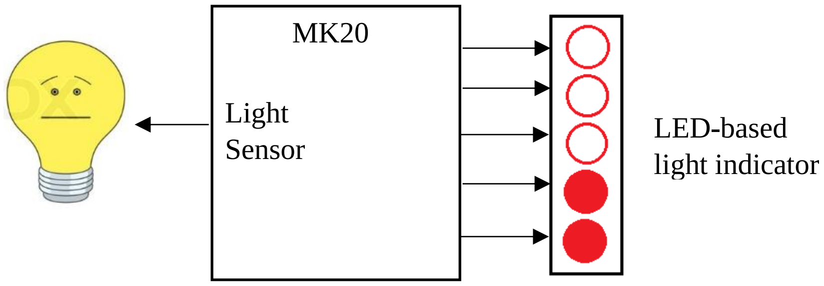
Description of the proposed use case with necessary illustration:

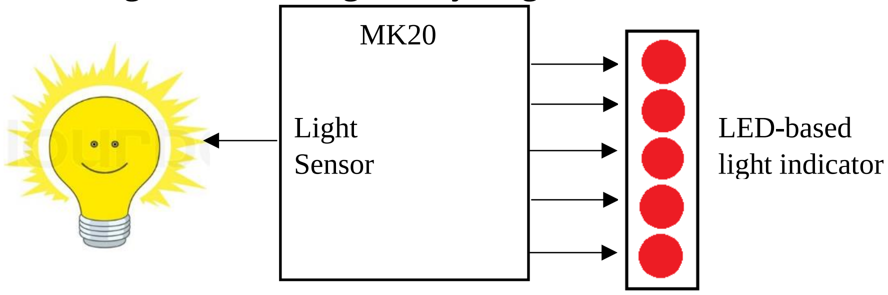
In this assignment, the proposed firmware application is the light intensity indicator using LEDs. Basically, the proposed application works in such a way that the light sensor LDR is used to detect the amount of light intensity in an area. For the hardware part, it consists of 5 LEDs connected to 5 output pins. The amount of LEDs lighting up at a time will be the light intensity surrounding, with no LEDs lighting up as extremely low light intensity and 5 LEDs lighting up as very high light intensity. Illustration as shown below:
When nearly no light surrounding/dim area:



When light surrounding is moderate:



When light surrounding is very bright:



By implementing this LED-based light indicator, it can help us to monitor the light intensity in the surrounding, particular in industrial use, where light intensity plays a crucial factor in some production lines area inside a factory. Low light intensity may cause accidents to the factory operators when working inside the factory.

I/O (Input/Output) table describing the details of every input and output device:

Components:

Components	**Quantity**	**Descriptions**
Light Sensor	1	Directly connected to the MK20 Freedom Board through ADC channel 19, it is used to detect the light intensity surrounding
Red LEDs	5	Displays the light intensity in LED-form. The more the LEDs light up, the higher the light intensity will be.
Resistor Array	1	Directly connected to each LEDs

Uses of the GPIO Port:

Port Name	**Port Type**	**Descriptions**
Channel 19 of ADC	Input	Directly connected to the MK20 Freedom Board through ADC channel 19 without any wiring required, it is used to detect the light intensity surrounding
PTC0-PTC4	Output	Each LEDs is directly connected to each pin from PTC0 to PTC4.

Circuit diagram of the proposed system:

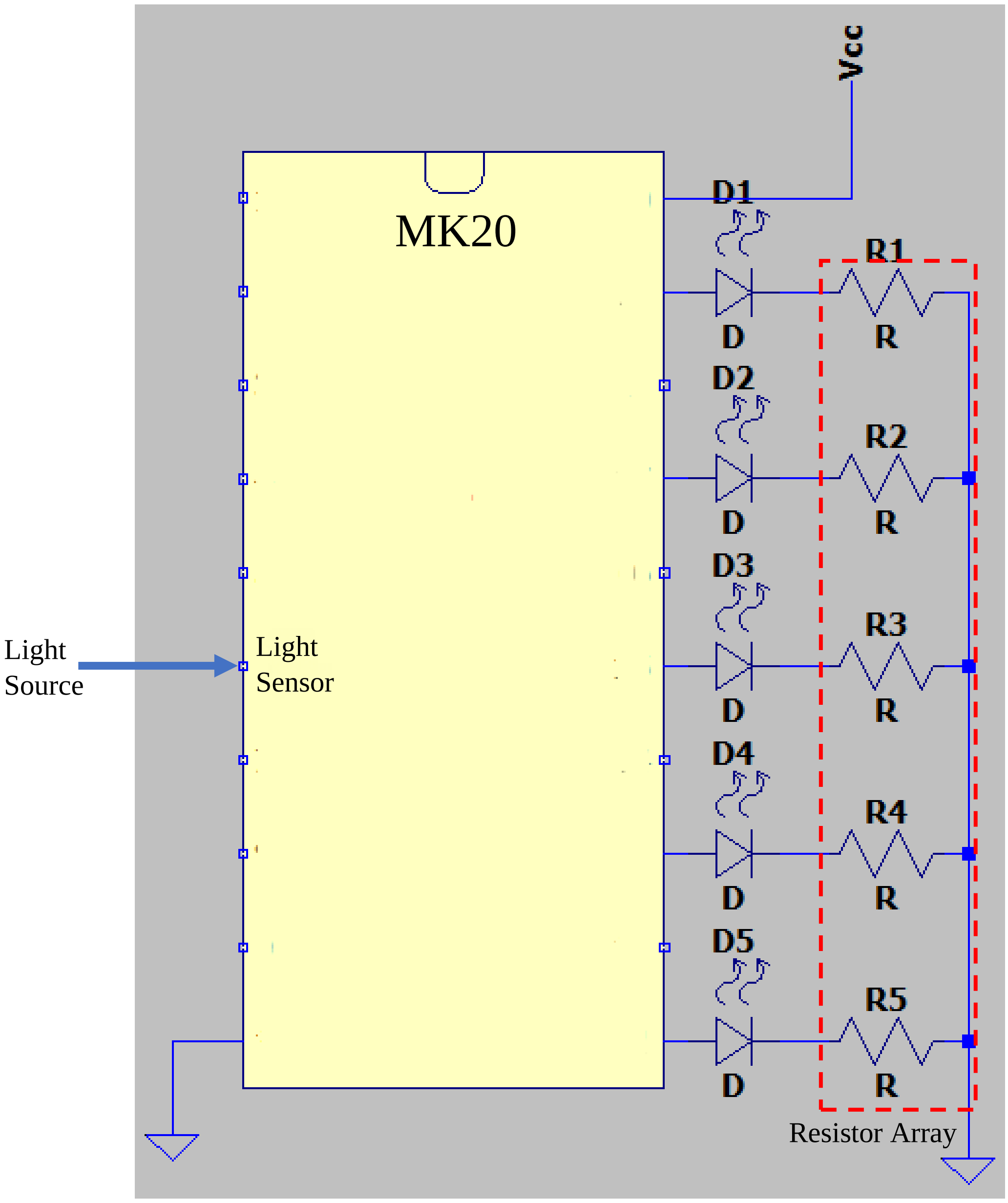


Figure 1: Schematic Circuit of the Proposed Applications

Internal Block Diagram:

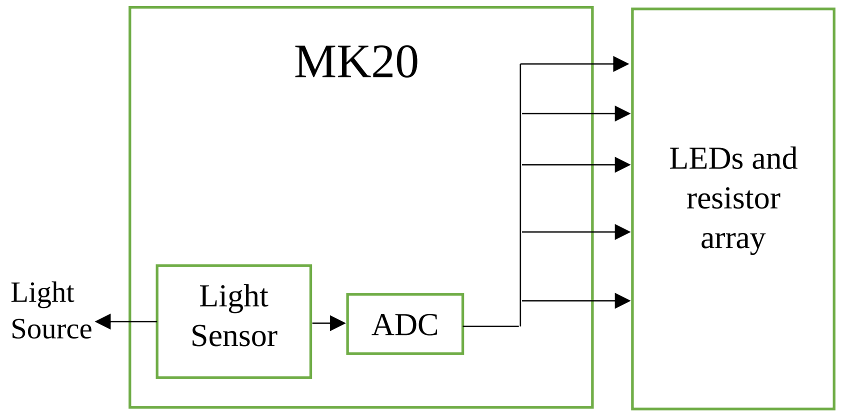


Figure 2: Internal Block Diagram of how the Light Intensity is being detected with the implementation of ADC

Based on the internal block diagram, initially the light source is being detected by the light sensor connected to the ADC channel 19.

ADC Channel #	Channel Name		Input Signal			
			AC1x.DIFF=0		AC1x.DIFF=1	
	CFG2.MUX=0	CFG2.MUX=1	CFG2.MUX=0	CFG2.MUX=1	CFG2.MUX=0	CFG2.MUX=1
0	DAD0		ADC0_DP0		ADC0_DP0 + ADC0_DM0	
1	DAD1					
2	DAD2					
3	DAD3		ADC0_DP3		ADC0_DP3 + ADC0_DM3	
4	AD4a	AD4b		ADC0_SE4b		
5	AD5a	AD5b		ADC0_SE5b		
6	AD6a	AD6b		ADC0_SE6b		
7	AD7a	AD7b		ADC0_SE7b		
8	AD8		ADC0_SE8			
9	AD9		ADC0_SE9			
10	AD10					
11	AD11					
12	AD12		ADC0_SE12			
13	AD13		ADC0_SE13			
14	AD14		ADC0_SE14			
15	AD15		ADC0_SE15			
16	AD16					
17	AD17					
18	AD18					
19	AD19		ADC0_DM0			
20	AD20					
21	AD21		ADC0_DM3			
22	AD22		VREF Output			
23	AD23		/ADC0_SE23			
24	AD24					
25	AD25					
26	AD26		Temperature Sensor SE		Temperature Sensor DIFF	
27	AD27		Bandgap SE		Bandgap DIFF	
28	AD28					
29	AD29		VREFH		-VREFH DIFF	
30	AD30		VREFL			
31	Disabled		Disabled			

Figure 3: ADC Channel being used from the ADC mapping table

Implementation of ADC function inside MK20 Board:

To implement the ADC into the application, first a few registers are required to be considered and configured properly before reading the light intensity data from the light source surrounding.
To use the ADC function inside MK20, first the clock needs to be enabled inside the register SIM_SCGC6:

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	1	RTC	0	ADC0	0	FTM1	FTM0	PIT	PDB	USBDCD	0		CRC	0	
W																
Reset	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	I2S	0	0	SPI0	0		0	0				0	0		DMAMUX	FTFL
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Figure 4: Clock Enable Function at bit 27

Based on the SIM_SCGC6 register, the ADCO represents the Clock Gate Control, which needs to be set to logic '1' in order to enable the clock to the ADC.

Next, the mode of operation and the low/high power configuration are considered using the register ADC configuration register 1 (ADC*x*_CFG1):

Bit	7	6	5	4	3	2	1	0
R W	ADLPC	ADIV		ADLSMP	MODE		ADICLK	
Reset	0	0	0	0	0	0	0	0

Figure 5: ADC_CFG1 bit configuration

In this register, the MODE and ADLPC are required to be configured. For the MODE, a 10-bit conversion will be used for the ADC conversion process. For the ADLPC bit, normal power is used instead of low power mode. This is due to the fact that low power is not suitable for the ADC implementation as it can affect the sensitivity of the light sensor at the middle range of values, hence ADLPC bit will be set to logic '0'.

Next, the input channel is configured using the ADCH bit from the register ADC status and control registers 1 (ADC*x*_SC1*n*), along with the COCO bit at bit 7 of the same register.

Bit	7	6	5	4	3	2	1	0
R	COCO	AIEN	DIFF	ADCH				
W								
Reset	0	0	0	1	1	1	1	1

Figure 6: Configuration bit of the register ADC*x*_SC1

For the ADCH at bit 0-3, it specifies which input channel will be used and select it by defining the input channel value at the ADCH register. In this case, by default light sensor is located at channel 19 of the MK20 Freedom Board. Hence, the configuration statement will be: ADC0_SC1A = ADC_SC1_ADCH(19);
For the COCO bit, it works by setting it to logic ‘1’. By doing so, the program will wait for the ADC conversion process to be completed before performing the next action, in order to avoid any disturbance and interfering to the conversion process.

The last register used for the conversion process will be the Result Register, (ADC*x*_R*n*). This register contains the results of an ADC conversion of channel 19 which is selected by the channel control register (SC1A:SC1n). Since at the CFG1 register, 10-bit single-ended conversion mode is used. Hence, the data inside the ADCH will have unsigned 10-bit representation.

Conversion mode	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0	Format
13-bit differential	S	S	S	S	D	D	D	D	D	D	D	D	D	D	D	D	Sign extended 2's complement
12-bit single-ended	0	0	0	0	D	D	D	D	D	D	D	D	D	D	D	D	Unsigned right justified
11-bit differential	S	S	S	S	S	S	D	D	D	D	D	D	D	D	D	D	Sign extended 2's complement
10-bit single-ended	0	0	0	0	0	0	D	D	D	D	D	D	D	D	D	D	Unsigned right justified
9-bit differential	S	S	S	S	S	S	S	S	D	D	D	D	D	D	D	D	Sign extended 2's complement
8-bit single-ended	0	0	0	0	0	0	0	0	D	D	D	D	D	D	D	D	Unsigned right justified

Table 1: Bit status inside the Data Result Register

Complete source code of the Firmware Application:

```
#include "derivative.h" /* include peripheral declarations */
#include <stdio.h>
#include "clock.h"

#define lightintensityADC 19//Channel 19 of ADC
#define ADC_MAX_VALUE ((1<<10)-1)//10-bit conversion
#define LIGHTMAX_MASK 15
#define LIGHTMIN_MASK 0
#define LED_MASK (0x1F)//LED on/off mask

 void delay(){//delay function
	int delayCount;
	for(delayCount = 0; delayCount < 200000; delayCount++){
		asm("nop");
	}
 }

 unsigned int getLightIntensity(){//function to get the light intensity
value
	double lightIntensity;
	ADC0_SC1A=ADC_SC1_ADCH(lightintensityADC);//Trigger conversion on
ADC Ch 19
	while((ADC0_SC1A && ADC_SC1_COCO_MASK)==0);//Wait for conversion
tom complete
	lightIntensity = LIGHTMIN_MASK + ((int)ADC0_RA * (LIGHTMAX_MASK-
LIGHTMIN_MASK))/ADC_MAX_VALUE;
	lightIntensity = LIGHTMAX_MASK - lightIntensity;
	return lightIntensity;
 }

int main(void)
{
	clock_initialise();
	SIM_SCGC5 |= SIM_SCGC5_PORTC_MASK;
	SIM_SCGC6 |= SIM_SCGC6_ADC0_MASK;// Enable clock to ADC
	PORTC_PCR0 = PORT_PCR_MUX(1)|PORT_PCR_DSE_MASK|
			PORT_PCR_PE_MASK|PORT_PCR_PS_MASK;//Set high drive
strength and pull-down and pull-up function
	PORTC_PCR1 = PORT_PCR_MUX(1)|PORT_PCR_DSE_MASK|
			PORT_PCR_PE_MASK|PORT_PCR_PS_MASK;
	PORTC_PCR2 = PORT_PCR_MUX(1)|PORT_PCR_DSE_MASK|
			PORT_PCR_PE_MASK|PORT_PCR_PS_MASK;
	PORTC_PCR3 = PORT_PCR_MUX(1)|PORT_PCR_DSE_MASK|
			PORT_PCR_PE_MASK|PORT_PCR_PS_MASK;
	PORTC_PCR4 = PORT_PCR_MUX(1)|PORT_PCR_DSE_MASK|
			PORT_PCR_PE_MASK|PORT_PCR_PS_MASK;
	GPIOC_PDDR |= LED_MASK;

	int counter = 0;
	ADC0_CFG1 = ADC_CFG1_MODE(2);//Set the mode operation to 10-bit
conversion mode

	double light = 0;//convert to decimal place for the light intensity
value
```

```c
	for(;;) {
		counter++;
		light = getLightIntensity();//Calling the function to get the
light intensity value
	   if(light<1.1){
		GPIOC_PDOR = 0x00;//None of the LEDs will light up
		printf("Extremely low light intensity!\n");
		delay();
	   }
	   else if(light<2.5){
		GPIOC_PDOR = 0x01;//One LED lights up
		printf("Very low light intensity!\n");
		delay();
	   }
	   else if(light<3.5){
		GPIOC_PDOR = 0x03;//Two LEDs will light up
		printf("Low light intensity!\n");
		delay();
	   }
	   else if(light<4.5){
		GPIOC_PDOR = 0x07;//Three LEDs will light up
		printf("Average Light intensity!\n");
		delay();
	   }
	   else if(light<6.5){
		GPIOC_PDOR = 0x0F;//Four LEDs will light up
		printf("High light intensity!\n");
		delay();
	   }
	   else{
		GPIOC_PDOR = 0x1F;//All LEDs will light up
		printf("Very high light intensity!\n");
		delay();
	   }
	}

	return 0;
}
```

Key Components of the Program Coding:

(a) Delay function

```
void delay(){//delay function
    int delayCount;
    for(delayCount = 0; delayCount < 200000; delayCount++){
        asm("nop");
    }
}
```

Figure 7: Code Snippet of the delay function

The above function performs a short delay where the microcontroller does nothing for 200000 times. The asm("nop"); statement was included to prevent the microcontroller from finishing the delay very quickly with its high clock speed.

(b) getLightIntensity function

```
unsigned int getLightIntensity(){//function to get the light intensity value
    double lightIntensity;
    ADC0_SC1A=ADC_SC1_ADCH(lightintensityADC);//Trigger conversion on ADC Ch 19
    while((ADC0_SC1A && ADC_SC1_COCO_MASK)==0);//Wait for conversion to complete
    lightIntensity = LIGHTMIN_MASK + ((int)ADC0_RA * (LIGHTMAX_MASK-LIGHTMIN_MASK))/ADC_MAX_VALUE;
    lightIntensity = LIGHTMAX_MASK - lightIntensity;
    return lightIntensity;
}
```

Figure 8: Code Snippet of the getLightIntensity function

The above function is the function which is used to get the light intensity data from outside surrounding. The light sensor then detects the light intensity and trigger the conversion on ADC channel 19 by writing 10011 in binary into the ADCH inside the SC1A register. The microcontroller will wait for the conversion process to complete by writing the statement while((ADC0_SC1A && ADC_SC1_COCO_MASK)==0); where COCO bit is a bit used to check whether the conversion has been completed or not. The function calculates the light intensity result with the formula above, where ADC0_RA is a result register which contains the results of the ADC channel 19.

(c) Enable Clock to GPIO port and ADC

```
clock_initialise();
SIM_SCGC5 |= SIM_SCGC5_PORTC_MASK;
SIM_SCGC6 |= SIM_SCGC6_ADC0_MASK;// Enable clock to ADC
```

Figure 9: Code Snippet for Enable Clock Function to GPIOC and ADC

To use the GPIO and ADC function, first clock enabling must be done by carrying out bit masking to the SIM_SCGC5 register for GPIO and to SIM_SCGC6 for ADC.

(d) Checking Light Intensity Value

```
for(;;) {
    counter++;
    light = getLightIntensity();//Calling the function to get the light intensity value
    if(light<1.1){
        GPIOC_PDOR = 0x00;//None of the LEDs will light up
        printf("Extremely low light intensity!\n");
        delay();
    }
    else if(light<2.5){
        GPIOC_PDOR = 0x01;//One LED lights up
        printf("Very low light intensity!\n");
        delay();
    }
    else if(light<3.5){
        GPIOC_PDOR = 0x03;//Two LEDs will light up
        printf("Low light intensity!\n");
        delay();
    }
    else if(light<4.5){
        GPIOC_PDOR = 0x07;//Three LEDs will light up
        printf("Average Light intensity!\n");
        delay();
    }
    else if(light<6.5){
        GPIOC_PDOR = 0x0F;//Four LEDs will light up
        printf("High light intensity!\n");
        delay();
    }
    else{
        GPIOC_PDOR = 0x1F;//All LEDs will light up
        printf("Very high light intensity!\n");
        delay();
    }
}
```

Figure 10: Code Snippet for the Checking Light Intensity statement

To check the range of values of the light intensity value, if-else statements are implemented with an infinite for loop enclosing it. First, the program calls the function *getLightIntensity()* to get the resultant converted light intensity value. Then the program will check each light intensity value to display the required amount of LEDs.

Flowcharts:

(a) Main Program

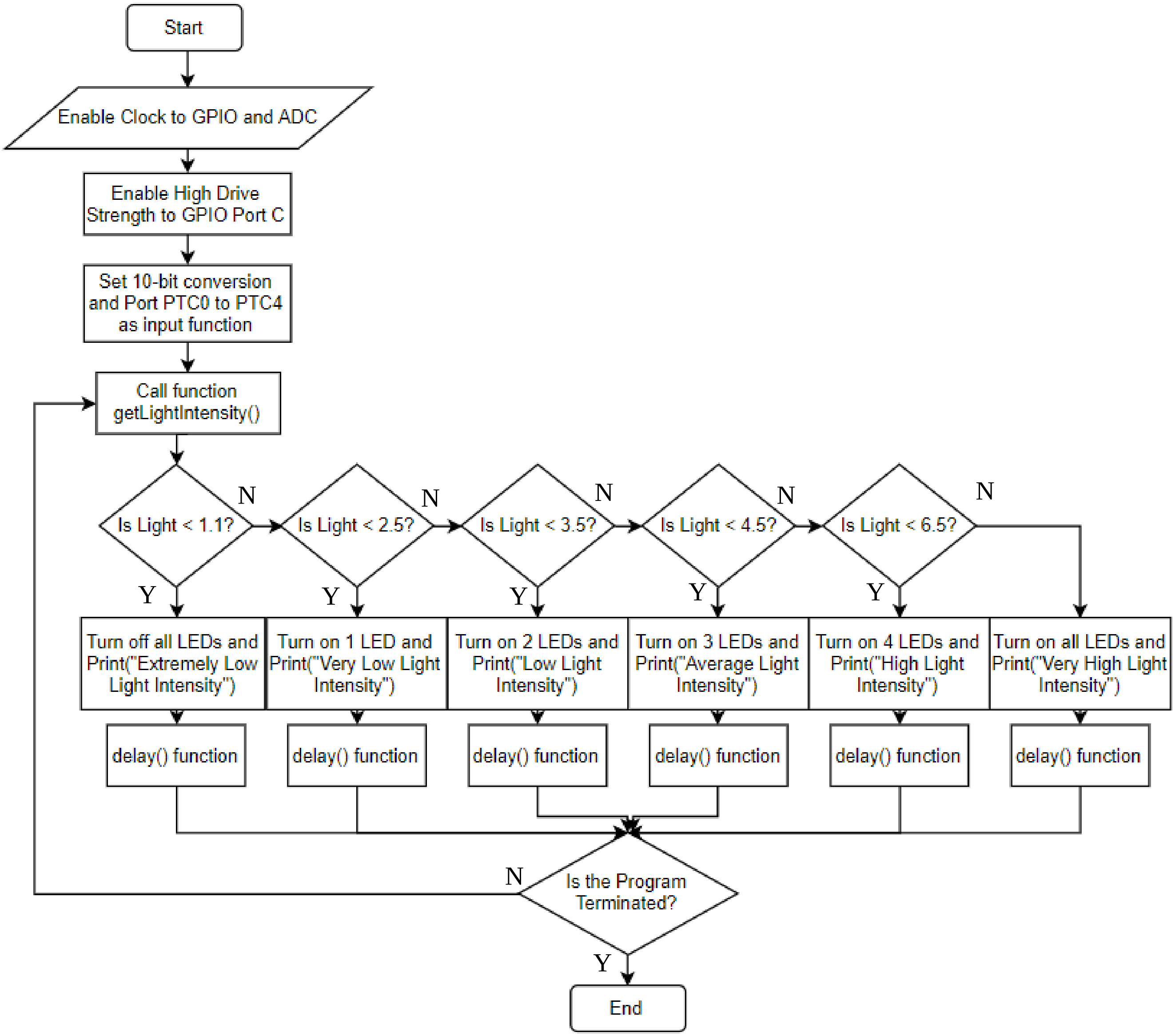


Figure 11: Flowchart of the Main Program

(b) getLightIntensity function

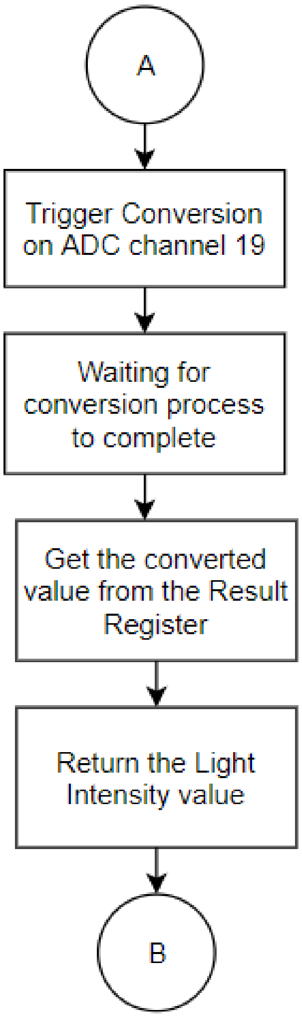


Figure 12: Flowchart of the getLightIntensity function

(c) Delay function

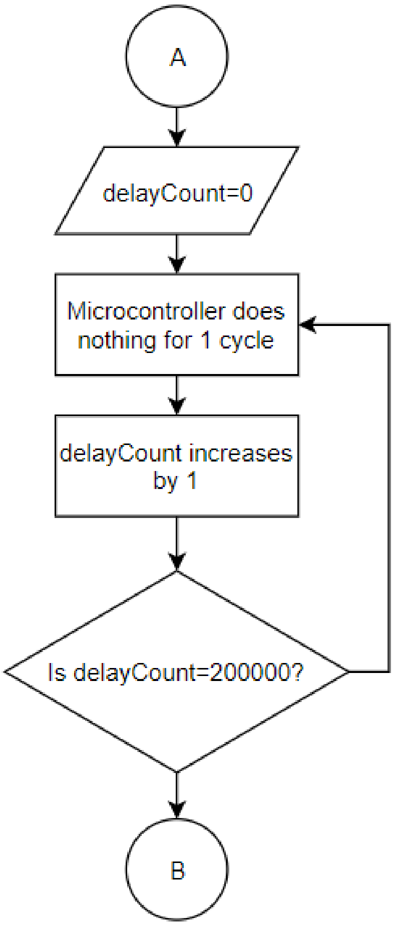


Figure 13: Flowchart of the Delay function