Table of Contents

Introduction

The build-up of carbon dioxide resulting from poor ventilation in housing can be potentially dangerous to a person's health. Air exchange between 350 and 1000 ppm is the ideal interval range before experiencing drowsiness and difficulty concentrating. In the following design problem, it was required to implement a portable simplified in-home CO_2 monitoring apparatus to prevent the spreading of viruses. A K60 microcontroller, 8-digit 7-segment LCD display unit, warning display, and a green, yellow, and red LED were all used in the hardware implementation of our monitoring apparatus. The default CO_2 monitor is a programmable device that intakes the level of CO_2 in ppm and compares it to the corresponding warning level. Once a warning level is identified, the corresponding LED colour and warning will light up to convey to the user the danger of the CO_2 levels. There is also an updated operation that runs the same as the default, although it allows the user to introduce three new warning levels to the program that the CO_2 levels can be compared to, instead of the predefined.

The benefits of this monitoring apparatus include allowing the user to not have to worry about their air exchange reaching an unhealthy level. The user does not have to constantly monitor their air, and should the ppm be outside a healthy level, the user will be immediately notified so they can resolve the issue safely and quickly. The default operation makes it easier for users who are less qualified on the subject of air exchange and quality by using default settings and pre-defined warning levels. The updated operation gives the option of setting more warnings to allow for those with a deeper understanding of the topic to manage their air quality more specifically.

Hardware Interface Design Solution

In order to solve the design problem, the major hardware components that were implemented and used are detailed in Table 1. All selected IC devices are CMOS compatible as per the specifications of the device.

Table 1: Major Hardware Components of the SIHM_CO2 Apparatus

Hardware	**Description**
K60 Microcontroller [1]	The K60 Microcontroller acts as the brain of the SIHM_CO2 apparatus. Serving as a base for both output control through the LED and LCD displays, and input management from the XENSIV CO_2 sensor and the keypad. The GPIO pins make input and output highly effective because they are bidirectional, as either output or input. The code that would be run on the controller would be responsible for listening for input, and storing data from the 4x4 matrix keypad. It would also have to listen for interrupts from the sensor component. The calculations from these inputs determine the required output to send to the LEDs and LCD displays through the GPIO ports. The data transfers from the sensor and to the LCD controller IC are serial.
XENSIV PAS CO_2 sensor [2]	The CO_2 sensor was implemented in order to read the air for its CO_2 concentration in order to take the average after a minute. The sensor provided multiple functionalities, as its internal hardware can be programmed to fit different configurations. It contains a microcontroller that runs compensation firmware to the SIHM_CO2 module.
LTC-4727JS [3] 4-digit 7-segment LCD display unit	The 7-segment LCD display was implemented so the average CO_2 concentration per minute can be displayed to the user. It uses a common cathode digit selection and the segments are anodes. The 7-segment device selected also has the ability to show decimal points for each digit display, as well as a colon. So while it is optimized for the specifications, it provides extended functionality for improving the overall device.
27899 [4] 4x4 keypad	Keypad was implemented so the user can define whether they are using the default operation mode or the updated operation mode. If update mode is chosen, the

	keypad can be used to set the new CO_2 levels to whatever the user wishes. The row inputs are set high by the MM74C922 IC and when a button is pressed, the column output changes from low to high indicating the selected key.
L-57IID [5] 3 LEDs (green, yellow, red)	The LEDs were implemented in order to show the warning level of the air concentration. For example, if the warning level is green the green LED should light up. The selected LED components are optimized for high efficiency and require a low input voltage with a maximum required of 2.5V.
MM74C922 [6]	The MM74C922 integrated circuit is an encoder chip that works in conjunction with the 4x4 matrix keypad. It uses a depressed button scanning detection implementation and encodes the output into a four bit representation of the depressed button. This IC also uses a Data Available pin that goes high to indicate when a button is pressed. In addition to optimized functionality with the keypad, the chip also allows customization of the scanning frequency and keybounce recovery time by means of changing the capacitance at pins 5 and 6 on the schematic in Figure 1. This allows configuration to fit specifications as required. The current settings with 1µF capacitors provide a button scan frequency of 60Hz and a key debounce time of 10µs, which are ideal for the specifications of the overall device.
A8474 [7]	The A8474 chip is the decoder chip selected for controlling the 7-segment display. This chip uses serial input functionality (into pin 2), where the data is passed into the chip one bit at a time on the system clock (into pin 10). When the data of which segments to switch high is passed, the load pin at pin 9 is set high indicating the data is ready to be released to the LCD display. A singular chip is compatible for a 4-digit display, however this chip also utilizes a Data Out pin so that multiple A8474 chips can be daisy-chained together for larger displays.
SN74LVC1G04 [8]	This IC implements a singular inverter gate to set up the required functionality of the MM73C922 chip. Specifically, the inverter flips the output from the Data Available pin (12) and feeds it into the Output Enable pin (13). Pin 13 is pre-inverted, so when the data from the keypad is ready, the output is enabled and the Data Out pins become respective of the state of the input. This chip is ideal because of the small dimensions, low required voltage and simple utilization.

Figure 1 shows the complete hardware implementation of the SIHM_CO2 apparatus. Running off a 12V power supply, which can be a battery, or configured to an AC source with extended hardware to meet user specifications. The main supply voltage passes through a set of voltage divider resistors to ensure each individual component works under design criteria in the datasheets. The overall device can be separated into 4 operating components. The matrix keypad and MM74C922 chip encode user input and send the data to the K60 microcontroller. The 7-segment display and the A8474 decoder chip work to display a four-digit number to the user. The digits are shown one by one, but the frequency at which the digits are switched is high, so the display appears coherent [9]. The sensor measures the CO2 level via a gas inlet on its microcontroller board, and the data is transferred to the K60 board. And the three LEDs are controlled by the GPIO ports on the K60 board to display the CO2 level measured by the sensor. Voltage inputs into the sensor make use of various decoupling capacitors that act as a noise reduction for the voltage. These are applied to both the 12V and 3V input signals. Resistors are applied to multiple wires in the circuit as pull-up resistors such as to the clock and serial data connections for the sensor in order to bring the voltage up when high. There are also resistors that are pull-down components and are applied to the PSEL and PWM_DIS pins on the sensor to keep them at a logical 0 [10].

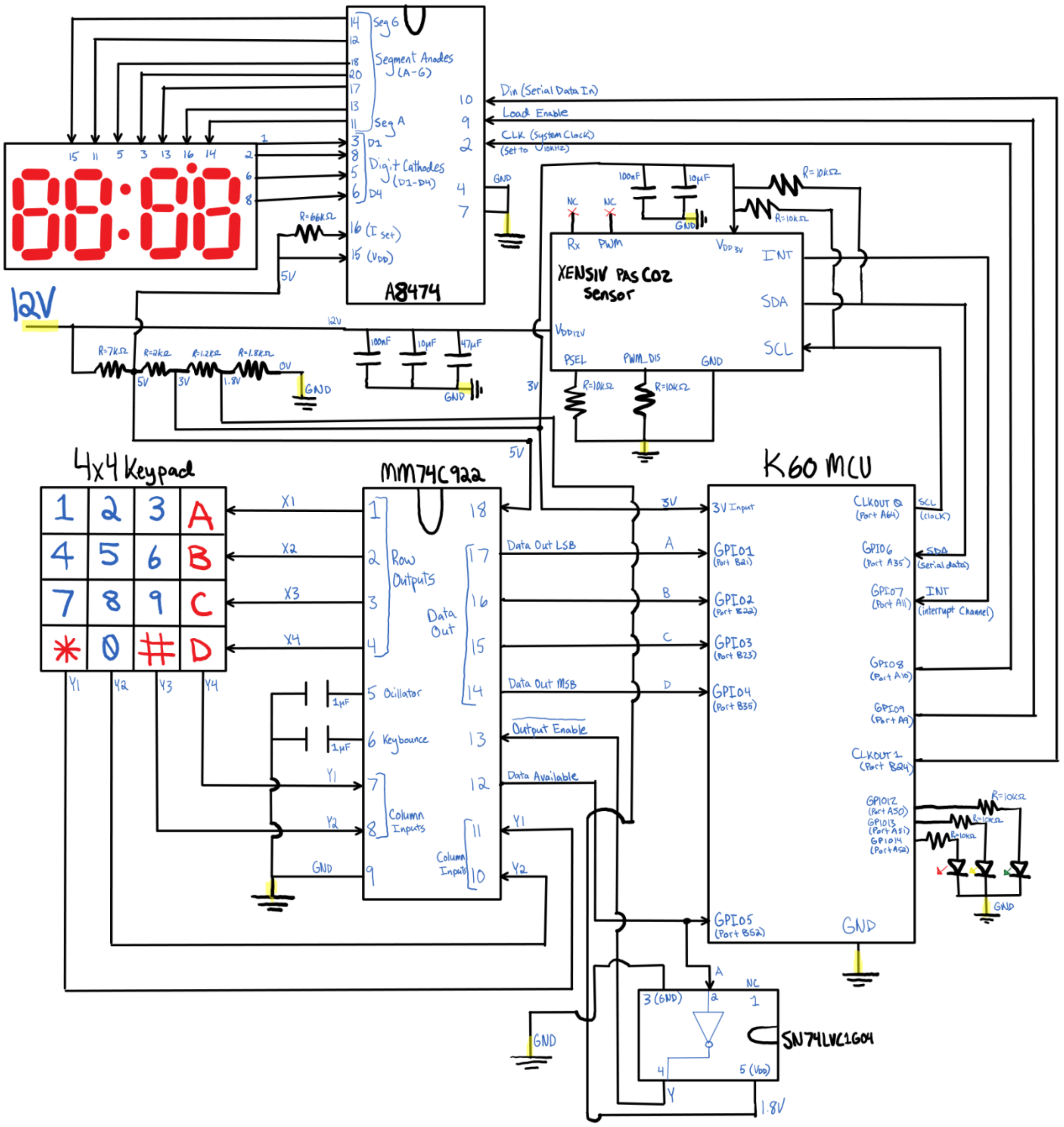


Figure 1: Hardware Implementation for Design Problem

Software Interface Design Solution

Flowchart

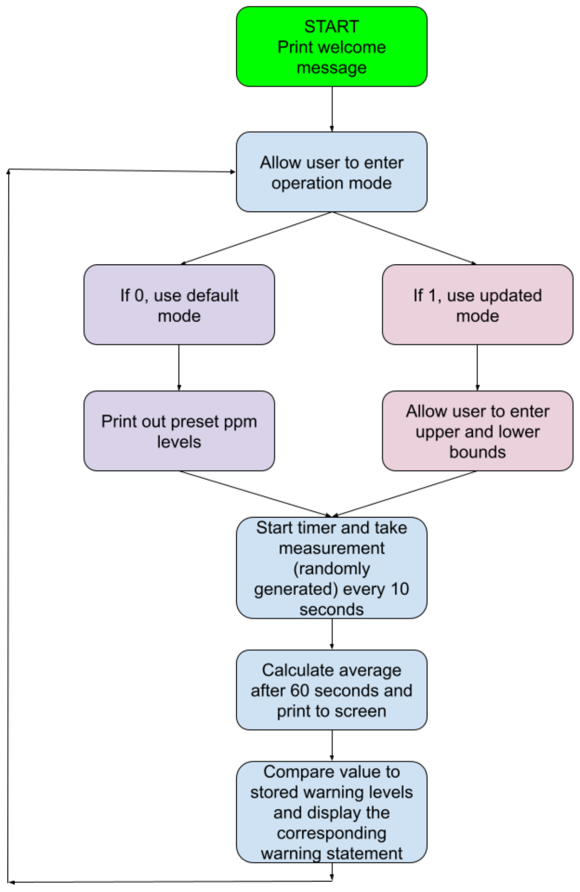

START
Print welcome message
Allow user to enter operation mode
If 0, use default mode
If 1, use updated mode
Print out preset ppm levels
Allow user to enter upper and lower bounds
Start timer and take measurement (randomly generated) every 10 seconds
Calculate average after 60 seconds and print to screen
Compare value to stored warning levels and display the corresponding warning statement

Figure 2: Flowchart for Software Implementation

Default Operation

In the default operating mode, the CO_2 level is read every 10 seconds and using the predefined intervals, compares the average level and outputs the corresponding warning and LED colour every minute. To access the default operation, the user must input 0 before the CO_2 sensor starts to measure values, and the code uses the subroutine setupDef and stores the lower register as 800 and the upper register as 1000, before running.

The ppm concentrations for each level are: <800 ppm lights the green LED, 800-1000 ppm lights the yellow LED, and >1000 ppm lights the red LED. The warnings for each level are: ‘CO2 Normal’ for green, ‘CO2 increasing, turn on house fan’ for yellow, and ‘CO2 high, turn on house fan and open windows’ for red.

Updated Operation

In the updated operating mode, the CO_2 level is read every 10 seconds and using the user defined intervals, compares the average level and outputs the corresponding warning and LED colour every minute. To access the updated operation, the user must input 1 before the CO_2 sensor starts to measure values. The code uses the subroutine setupUpd and stores the lower and upper register as whatever the user inputs before running.

The ppm concentrations are: <green Level Lg, green Level Lg - red Level Lg, and >red Level Lg. The warning levels for each level are: the same as those in the default operation

Assumptions

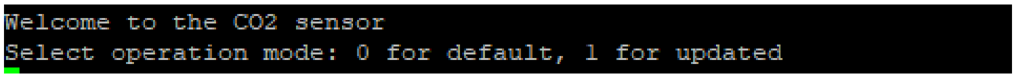
Some assumptions were made while developing the software implementation. One assumption was that the user only had to enter 2 numbers for updated mode, as the green warning would occur when below the lower bound, the yellow warning would occur when between the lower bound and the upper bound, and the red warning would occur when above the upper bound. Another assumption was that simulation helping functions (ie. printing, scanf random number generation) could be programmed in C.

Subroutine Breakdown

Table 2: Subroutine Breakdown of the SIHM_CO2 Apparatus

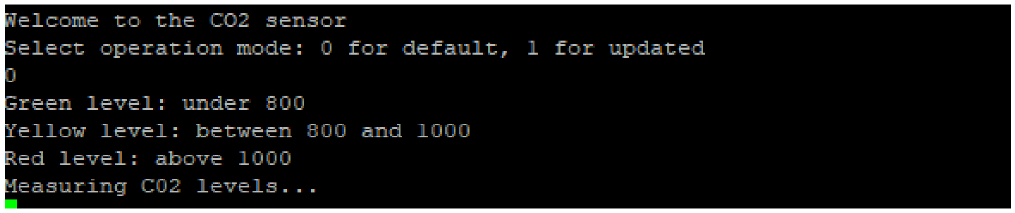
Subroutine	**Description**
asmmain	Prints opening message and moves seed for random number generation into r4. Branches to intro
intro	Asks the user which operating mode to enter, accepts input from the user and branches to setupDef if they enter 0, or setupUpd if they enter 1. Also resets all registers but r4.
timerStuff	Sets values for the LPTMR (interrupts every 10 seconds). Branches to enable_irq
enable_irq	Enables the timer. Branches to loop
loop	Generates a random number at every interrupt and adds this number to r6. When irqcounter = 6, branches to NotLoop, otherwise branches to loop.
NotLoop	Divides r6 by 6 to obtain the average C02 measurement for the minute and prints to screen. Branches to checkRed if average is greater than r9 (upper warning level), checkyellow if average is greater than r8 (lower warning level), or checkGreen if average is less than r8 (lower warning level).
lptmr_disable	Disables the LPTMR (so interrupts do not occur while waiting for the user to reenter operating mode).
setupDef	Stores 800 in r8, and 1000 in r9. Prints a message to the terminal to tell the user what the default warning levels are. Branches to timerStuff.
setupUpd	Prompts for and accepts input from the user for the lower warning level and the higher warning level, storing them in r8 and r9 respectively. Branches to timerStuff
checkRed	Prints warning for the red level, branches to lptmr_disable.
checkYellow	Prints warning for the yellow level, branches to lptmr_disable.
checkGreen	Prints warning for the green level, branches to lptmr_disable.
asm_lptmr_irq	Interrupt service routine, adds 1 to irqcounter at every interrupt.

Simulation Results



```
Welcome to the CO2 sensor
Select operation mode: 0 for default, 1 for updated
```

Figure 3: Simulation Result 1

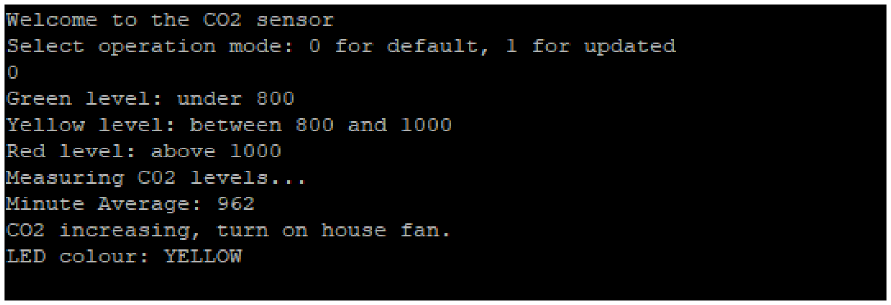


```
Welcome to the CO2 sensor
Select operation mode: 0 for default, 1 for updated
0
Green level: under 800
Yellow level: between 800 and 1000
Red level: above 1000
Measuring C02 levels...
```

Figure 4: Simulation Result 2

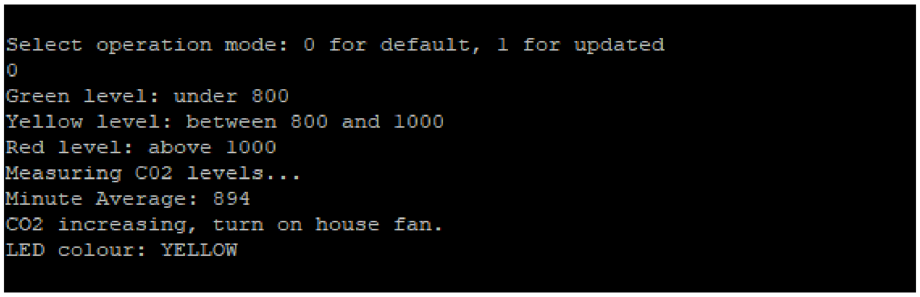
The first thing that is printed to the output terminal is a command asking for which operation mode is going to be used (0 for default mode or 1 for updated) and then displays what the respective values for each warning level is. After 1 minute the average level of CO_2 is displayed along with the respective warning and LED colour. If the user chooses 1, the input of the upper and lower ppm concentration values need to be input to be used as the new levels and then continues on in the same fashion of calculating the average after a minute.

Results for Default Mode:



```
Welcome to the CO2 sensor
Select operation mode: 0 for default, 1 for updated
0
Green level: under 800
Yellow level: between 800 and 1000
Red level: above 1000
Measuring C02 levels...
Minute Average: 962
CO2 increasing, turn on house fan.
LED colour: YELLOW
```

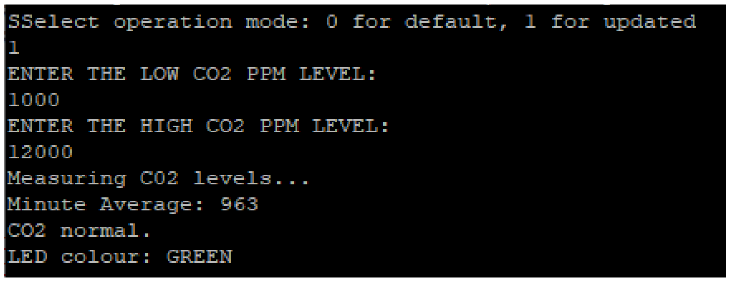
Figure 5: Simulation Result 3



```
Select operation mode: 0 for default, 1 for updated
0
Green level: under 800
Yellow level: between 800 and 1000
Red level: above 1000
Measuring C02 levels...
Minute Average: 894
CO2 increasing, turn on house fan.
LED colour: YELLOW
```

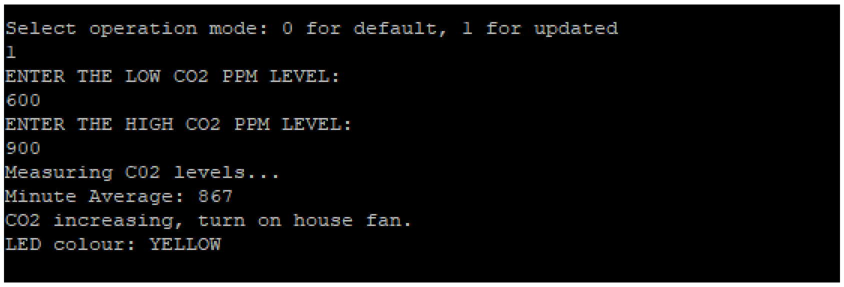
Figure 6: Simulation Result 4

Results for Updated Mode:



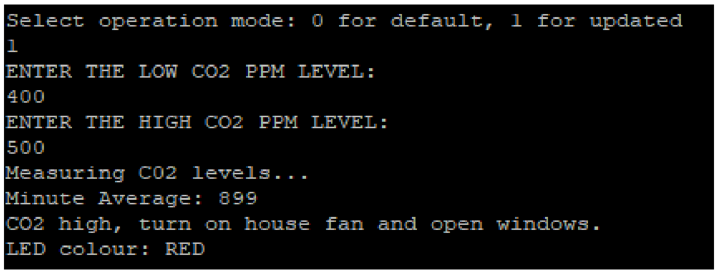
```
SSelect operation mode: 0 for default, 1 for updated
1
ENTER THE LOW CO2 PPM LEVEL:
1000
ENTER THE HIGH CO2 PPM LEVEL:
12000
Measuring C02 levels...
Minute Average: 963
CO2 normal.
LED colour: GREEN
```

Figure 7: Simulation Result 5



```
Select operation mode: 0 for default, 1 for updated
1
ENTER THE LOW CO2 PPM LEVEL:
600
ENTER THE HIGH CO2 PPM LEVEL:
900
Measuring C02 levels...
Minute Average: 867
CO2 increasing, turn on house fan.
LED colour: YELLOW
```

Figure 8: Simulation Result 6



```
Select operation mode: 0 for default, 1 for updated
1
ENTER THE LOW CO2 PPM LEVEL:
400
ENTER THE HIGH CO2 PPM LEVEL:
500
Measuring C02 levels...
Minute Average: 899
CO2 high, turn on house fan and open windows.
LED colour: RED
```

Figure 9: Simulation Result 7

References

[1] Freescale, "*K60 Sub-Family Reference Manual*", TWR-K60D100M. Oct, 2011.

[2] Infineon, "*Preliminary Datasheet of XENSIV PAS CO2*", XENSIV PAS CO2. June, 2021.

[3] LiteOn Electronics Inc., "*7-Segment Display Datasheet*", C4727JS.

[4] Parallax Inc., "*4x4 Matrix Membrane Keypad*", 27899. Dec, 2011.

[5] Kingbright, "*Solid State Lamp*", L-57IID. Mar, 2018.

[6] Fairchild Semiconductor, "*16-Key Encoder*", MM74C922. Oct, 1987.

[7] Advanced Innovation Technology Corp., "*Up to 4 Digit, 7 Segment Numeric Display Driver*", A8474.

[8] Texas Instruments, "*SN74LVC1G04 Datasheet*", SN74LVC1G04. Apr, 1999.

[9] R. Bhatt, *Serial 4-Digit Seven Segment LED Display*. Accessed on: Nov 4, 2021. Available: www.electronics-lab.com

[10] G. E. Ngondi. *A Dictionary of Computer Science*, 7th ed. Oxford University Press, 2016.

Appendix

A.1 Source Code

my_main.s

```
		PRESERVE8
		AREA	SCopy, CODE, READONLY
my_LPTMR0_base EQU 0x40040000
mySIM_SCGC5 EQU 0x40048038
my_PLTMR0_mask EQU 0x00000001  ;need to set bit 0 (LPTIMER) in SIM_SCGC5
my_LPTMR0_CSR EQU 0x40040000  ;CSR register address
my_LPTMR0_PSR EQU 0x40040004  ;PSR register address
my_LPTMR0_CMR EQU 0x40040008  ;CMR register address
my_LPTMR0_CNR EQU 0x4004000C  ;CNR register address
my_CSR_value EQU 0x00000040; TIE set and TEN clear
my_CSR_TEN1 EQU 0x00000001 ;mask for TEN set to 1
my_CSR_TEN0 EQU 0xFFFFFFFE  ;mask for TEN set to 0
my_PSR_value EQU 0x00000005;set prescale value
my_CMR_value EQU 0x00001388;set compare value
BITBAND_base EQU 0x40000000
ALIASED_base EQU 0x42000000
CSR_TEN EQU 0 ;position bit
CSR_TCF EQU 7 ;position bit for TCV
my_NVIC_value EQU 0x00200000 ; enable the IRQ 0x55
my_NVIC_addr  EQU 0xE000E100 ;NVIC address

  EXPORT asmmain
		  ;import c functions used for printing and getting new values
		IMPORT my_fprint
		IMPORT int_fprint
		import highScan
		import lowScan
		import newVal
		import my_getChar
		import my_putChar
		import defaultSetup
		import greenWarning
		import yellowWarning
		import redWarning

asmmain
		LDR r0, =message2;print open message
		BL my_fprint
		MOV r4, #4
		B intro

intro
  ALIGN 4

		LDR r0, =message1
		BL my_fprint
		;reset all the registers except for r4
		MOV r1, #0
		MOV r2, #0
		MOV r3, #0
		MOV r5, #0
		MOV r6, #0
		MOV r7, #0
		MOV r8, #0
		MOV r9, #0
		MOV r10, #0
		MOV r11, #0
		MOV r12, #0

		BL my_getChar;get the operation select character

		CMP r0, #48;if 0, branch to default mode
		BEQ setupDef
		CMP r0, #49;if 1, branch to updated mode
		BEQ setupUpd

		BL my_putChar;print the operation select character
		B intro

timerStuff
		LDR r0, =message3
		BL my_fprint
		;set up timer values
  LDR r2,=mySIM_SCGC5
		MOV r1,#my_PLTMR0_mask
		LDR r0,[r2]
		ORR r0,r0,r1
		STR r0,[r2]

		LDR r2,=my_LPTMR0_CSR
		MOV r1,#my_CSR_value
		STR r1,[r2]
		ADD r2,#0x4
		MOV r1,#my_PSR_value
		STR r1,[r2]
		ADD r2,#0x4
		MOV r1,#my_CMR_value
		STR r1,[r2]
		SUB r2,#0x8
		MOV r1,#my_CSR_TEN1
		LDR r0,[r2]
		ORR r0,r0,r1
		STR r0,[r2]  ;start the timer now

enable_irq;enables interrupts
		LDR r2,=my_NVIC_value ;this value sets IRQ 85 or 0x55
		LDR r1,=my_NVIC_addr
		STR r2,[r1,#0x8]
		MOV r5,#0

		loop ;get new co2 values: simulates the sensor functionality
  LDR r1, =irqcounter
		LDRB r2,[r1]
		CMP r2,#11
		BEQ lptmr_disable
		CMP r5,r2
		BEQ loop
		MOV r5,r2
		ADD r4,r4,#1
		MOV r0, r4
		BL newVal;gets a new co2 reading
		ADD r6,r6,r0;add to running total for the average co2 per minute calculation
		LDR r1, =irqcounter
		LDR r0, [r1]
		LDR r1, =irqcounter
		LDR r0, [r1]
		CMP r0,#6
		BEQ NotLoop
		B loop

NotLoop
		LDR r1, =irqcounter
		LDRB r0,[r1]
		SDIV r6,r6,r0;divide the running total of the co2 readings by the irq counter (6) taking the average
		MOV r0,#1
		STRB r0,[r1]
		MOV r0, r6
		BL int_fprint;print the average co2 for the mminute
		;perform the check on the co2 average
		CMP r6, r9
		BGE checkRed
		CMP r6, r8
		BGE checkYellow
		CMP r6, r8
		BLT checkGreen

		MOV r6, #0
		B intro

lptmr_disable;disable the low power timer by setting the timer enable bit to 0
		LDR r2,=my_LPTMR0_CSR
		LSL r2,r2, #5
		LDR r3,=ALIASED_base
		ADD r2,r2,r3
		MOV r3,#CSR_TEN
		LSL r3,r3,#2  ;bit number times 4
		ADD r2,r2,r3
		MOV r1, #0x0
		STR r1,[r2]
		BL intro

setupDef
		BL my_putChar
		LDR r0, =spcc
		BL my_fprint

		MOV r8, #800;store 800 and 1000 as default settings for the co2 sensor
		MOV r9, #1000

		BL defaultSetup
		B timerStuff

setupUpd
		BL my_putChar
		LDR r0, =spcc
		BL my_fprint

		BL lowScan;get the low value for the co2 range
		MOV r8, r0
		BL highScan;get the high value for the co2 range
		MOV r9, r0

		B timerStuff

checkRed;check for red range
		BL redWarning
		MOV r6, #0
		LDR r0, =spcc
		BL my_fprint
```

```
        B lptmr_disable

checkYellow;check for yellow range
        BL yellowWarning
        MOV r6, #0
        LDR r0, =spcc
        BL my_fprint
        B lptmr_disable

checkGreen;check for green range
        BL greenWarning
        MOV r6, #0
        LDR r0, =spcc
        BL my_fprint
        B lptmr_disable

stop
        B stop

        ;interrupt service routine: gets called when there is an timer interrrupt
        PRESERVE8
                ALIGN 4
        AREA    LPTMRint, CODE, READONLY
        EXPORT asm_lptmr_irq

LPTMR0_IRQHandler EQU asm_lptmr_irq+1
        EXPORT LPTMR0_IRQHandler  ;the vector table must contain odd addresses for the Cortex processor
asm_lptmr_irq
        PUSH {lr}  ;store LR
        LDR r2,=my_LPTMR0_CSR
        LSL r2,r2, #5
        LDR r3,=ALIASED_base
        ADD r2,r2,r3
        MOV r3,#CSR_TCF
        ADD r2,r2,r3,LSL #2
        MOV r1, #0x1
        STR r1,[r2]
        LDR r1, =irqcounter
        LDRB r0,[r1]
        ADD r0,#1;increment the irq counter and store in register
        STRB r0,[r1]
        POP {pc}

;create all the print data
  ALIGN 4
  AREA MyData, DATA, READWRITE
irqcounter DCB 0x0,0x0,0x0,0x0
message1 DCB "Select operation mode: 0 for default, 1 for updated",0
message2 DCB "Welcome to the CO2 sensor",0
message3 DCB "Measuring C02 levels...",0
spcc DCB " ",0
        END
```

mains.c

```
 #define IO_MAXLINE 20
 #if (defined (FSL_RTOS_MQX) && (MQX_COMMON_CONFIG != MQX_LITE_CONFIG))
 #define PRINTF      printf
 #define SCANF scanf
 #define PUTCHAR putchar
 #define GETCHAR getchar
 #else
 #define PRINTF debug_printf
 #define SCANF debug_scanf
 #define PUTCHAR debug_putchar
 #define GETCHAR debug_getchar
 #endif


 ////////////////////////////////////////////////////////////////////////////////
//  Includes
////////////////////////////////////////////////////////////////////////////////

// Standard C Included Files
#include <stdio.h>
#include <stdlib.h>
// SDK Included Files
#include "board.h"
#include "gpio_pins.h"
#include "fsl_debug_console.h"

////////////////////////////////////////////////////////////////////////////////
// Variables
////////////////////////////////////////////////////////////////////////////////



////////////////////////////////////////////////////////////////////////////////
// Code
////////////////////////////////////////////////////////////////////////////////

extern void asmmain(void);


int my_getChar(){//get a character from the terminal
		int c = GETCHAR();
		return c;
}

void my_putChar(char c){//print character
		PUTCHAR(c);
}

void my_fprint(char *d)//print function
{
		PRINTF("%s\r\n", d);
}

void int_fprint(int d)//print the minute average
{
		PRINTF("Minute Average: %d\r\n", d);
}

int lowScan(int c) {//use the scanf to ge the low value for the co2 sensor warnings
		PRINTF("ENTER THE LOW CO2 PPM LEVEL: \r\n");
		SCANF("%d", &c);
		PRINTF("%d\r\n",c);
		return c;
}

int highScan(int c) {//use scanf to get the high value for the co2 sensor warnings
```

```
		PRINTF("ENTER THE HIGH CO2 PPM LEVEL: \r\n");
		SCANF("%d", &c);
		PRINTF("%d\r\n",c);
		return c;
}

void defaultSetup() {//print info message
		PRINTF("Green level: under 800 \r\n");
		PRINTF("Yellow level: between 800 and 1000 \r\n");
		PRINTF("Red level: above 1000 \r\n");
}

void greenWarning() {//printf the warning if  in the green warning if in the greenWarning range
		PRINTF("CO2 normal.\r\n");
		PRINTF("LED colour: GREEN \r\n");
}

void yellowWarning() {//print the warning if in yellow range
		PRINTF("CO2 increasing, turn on house fan.\r\n");
		PRINTF("LED colour: YELLOW \r\n");
}

void redWarning() {//print the warning if in red range
		PRINTF("CO2 high, turn on house fan and open windows.\r\n");
		PRINTF("LED colour: RED \r\n");
}

int newVal(int seed){//get a random value to simulate the co2 sensor functionality
  srand(seed);
  int x = (rand()%600)+600;
		return x;
}

int main(void)
{
  hardware_init();
		asmmain();
		return(0);
}

/*******************************************************************************
 * EOF
 ******************************************************************************/
```