Text

Description automatically generated

**School of Electrical and Computer Engineering**

**Fall 2022**

**ECE 5721**

**Embedded System Design**

**Lab 4: Analog-to-Digital Converter**

**Lab Report**

**Daniel Funke**

**December 4, 2022**

# Thermistor

**Description:**

This experiment entailed constructing a voltage divider circuit using a static 5k resistor and a negative temperature coefficient thermistor. The goal of this laboratory exercise was to utilize the ADC on the KL25Z to measure the ambient temperature. This was done by measuring the voltage between the thermistor and the static resistor, calculating the resistance of the thermistor using this value, and then converting resistance to temperature using the specifications provided by the thermistor’s manufacturer. Finally, this value was to be displayed on a pair of seven segment displays.

**Source Code:**

**main.c**

1. #include "adc.h"
2. #include "seven\_segment\_display\_encode.h"
3. #include "seven\_segment\_display.h"
4. #include <stdio.h>
5. #include <string.h>
6. #define BUF\_SIZE    (3)
7. int main(void)
8. {
9. adc\_init();
10. seven\_segment\_display\_init();
11. float temp;
12. int temp\_int;
13. char display\_buffer[BUF\_SIZE];
14. memset(display\_buffer, 0, BUF\_SIZE);
15. char encoded\_val\_0;
16. char encoded\_val\_1;
17. while(1)
18. {
19. temp = adc\_read\_temperature();
20. temp\_int = (int)temp;
21. snprintf(display\_buffer, BUF\_SIZE, "%2d", temp\_int);
22. encoded\_val\_0 = seven\_segment\_display\_encode\_character(display\_buffer[1]);
23. encoded\_val\_1 = seven\_segment\_display\_encode\_character(display\_buffer[0]);
24. seven\_segment\_display\_print\_0(encoded\_val\_0);
25. seven\_segment\_display\_print\_1(encoded\_val\_1);
26. }
27. }

**adc.c**

1. #include <stdint.h>
2. float adc\_read\_temperature(void)
3. {
4. // Constants used in conversion
5. static const float A1 = 3.354016e-03;
6. static const float B1 = 3.349290e-04;
7. static const float C1 = 3.683843e-06;
8. static const float D1 = 7.050455e-07;
9. static const float r\_ref = 15.0;
10. static const float r\_bal = 5000.0;
11. static float temp = 0;
12. // Start conversion on channel 0
13. ADC0->SC1[0] = 0x00;
14. // Wait for conversion to finish
15. while(!(ADC0->SC1[0] & ADC\_SC1\_COCO\_MASK)) {}
16. // Read result; convert to floating point
17. //float n = (float)ADC0->R[0];
18. float n = (float)ADC0->R[0];
19. float r\_therm = r\_bal \* ((65535.0 / n) - 1);
20. // Calculate temperature from transfer function
21. temp =   A1 +
22. (B1 \* (log(r\_therm/r\_ref))) +
23. (C1 \* (log(r\_therm/r\_ref) \* log(r\_therm/r\_ref))) +
24. (D1 \* (log(r\_therm/r\_ref) \* log(r\_therm/r\_ref) \* log(r\_therm/r\_ref)));
25. temp = (1 / temp) - 273 + 20;
27. return temp;
28. }

**seven\_segment\_display.c**

1. #include "seven\_segment\_display.h"
2. #include <MKL25Z4.h>
3. #define MASK(x) (1ul << (x))
4. enum seven\_segment\_leds {
5. SEVEN\_SEGMENT\_LED\_A,
6. SEVEN\_SEGMENT\_LED\_B,
7. SEVEN\_SEGMENT\_LED\_C,
8. SEVEN\_SEGMENT\_LED\_D,
9. SEVEN\_SEGMENT\_LED\_E,
10. SEVEN\_SEGMENT\_LED\_F,
11. SEVEN\_SEGMENT\_LED\_G,
12. SEVEN\_SEGMENT\_LED\_DP,
13. NUM\_SEVEN\_SEGMENT\_LEDS
14. };
15. void seven\_segment\_display\_init(void)
16. {
17. // Enable clock to PORTC and PORTD
18. SIM->SCGC5 |= SIM\_SCGC5\_PORTC\_MASK | SIM\_SCGC5\_PORTD\_MASK;
19. // Configure PORTC led pins as output GPIO, initialize low
20. for (int i = 0; i < NUM\_SEVEN\_SEGMENT\_LEDS; ++i) {
21. PORTC->PCR[i] &= ~PORT\_PCR\_MUX\_MASK;
22. PORTC->PCR[i] |= PORT\_PCR\_MUX(1);
23. PTC->PDDR |= MASK(i);
24. PTC->PCOR |= MASK(i);
25. }
26. // Configure PORTD led pins as output GPIO, initialize low
27. for (int i = 0; i < NUM\_SEVEN\_SEGMENT\_LEDS; ++i) {
28. PORTD->PCR[i] &= ~PORT\_PCR\_MUX\_MASK;
29. PORTD->PCR[i] |= PORT\_PCR\_MUX(1);
30. PTD->PDDR |= MASK(i);
31. PTD->PCOR |= MASK(i);
32. }
33. }
34. void seven\_segment\_display\_print\_0(uint8\_t encoded\_val)
35. {
36. // Clear old contents
37. PTC-> PCOR |= 0xFF;
38. // Write new contents
39. PTC-> PSOR |= encoded\_val;
40. }
41. void seven\_segment\_display\_print\_1(uint8\_t encoded\_val)
42. {
43. // Clear old contents
44. PTD-> PCOR |= 0xFF;
45. // Write new contents
46. PTD-> PSOR |= encoded\_val;
47. }

**seven\_segment\_display\_encode.c**

1. /\*\*
2. \* @file seven\_segment\_display\_encode.c
3. \* @author D. Funke
4. \* @brief A module to encode ascii characters into 7 segment led arrays
5. \* @date 2022-11-16
6. \*
7. \*/
8. #include "seven\_segment\_display\_encode.h"
9. /\*\*
10. \* @brief   Converts numerical ASCII character value to hexidecimal
11. \*          representation of 7 segment display bitmap
12. \*
13. \* @param c Numerical ASCII Character value to be encoded
14. \*
15. \* @return  uint8\_t Hexidecimal bitmap of 7 segment display LED array
16. \*/
17. static uint8\_t numeral\_to\_bitmap(char c) {
18. static uint8\_t numerals[10] = {
19. 0x3F, /\* 0 \*/
20. 0x06, /\* 1 \*/
21. 0x5B, /\* 2 \*/
22. 0x4F, /\* 3 \*/
23. 0x66, /\* 4 \*/
24. 0x6D, /\* 5 \*/
25. 0x7D, /\* 6 \*/
26. 0x07, /\* 7 \*/
27. 0x7F, /\* 8 \*/
28. 0x6F  /\* 9 \*/
29. };
30. return numerals[c - '0'];
31. };
32. /\*\*
33. \* @brief   Converts lower case ASCII character value to hexidecimal
34. \*          representation of 7 segment display bitmap
35. \*
36. \* @param c Lower case ASCII Character value to be encoded
37. \*
38. \* @return  uint8\_t Hexidecimal bitmap of 7 segment display LED array
39. \*/
40. static uint8\_t lower\_case\_to\_bitmap(char c) {
41. static uint8\_t lower\_case\_alpha[26] = {
42. 0x7D, /\* a \*/
43. 0x1F, /\* b \*/
44. 0x0D, /\* c \*/
45. 0x3D, /\* d \*/
46. 0x6F, /\* e \*/
47. 0x47, /\* f \*/
48. 0x7B, /\* g \*/
49. 0x17, /\* h \*/
50. 0x10, /\* i \*/
51. 0x18, /\* j \*/
52. 0x57, /\* k \*/
53. 0x06, /\* l \*/
54. 0x54, /\* m \*/
55. 0x15, /\* n \*/
56. 0x1D, /\* o \*/
57. 0x67, /\* p \*/
58. 0x73, /\* q \*/
59. 0x05, /\* r \*/
60. 0x5B, /\* s \*/
61. 0x0F, /\* t \*/
62. 0x1C, /\* u \*/
63. 0x1C, /\* v \*/
64. 0x2A, /\* w \*/
65. 0x37, /\* x \*/
66. 0x3B, /\* y \*/
67. 0x5B  /\* z \*/
68. };
70. return lower\_case\_alpha[c - 'a'];
71. };
72. /\*\*
73. \* @brief   Converts upper case ASCII character value to hexidecimal
74. \*          representation of 7 segment display bitmap
75. \*
76. \* @param c Upper case ASCII Character value to be encoded
77. \*
78. \* @return  uint8\_t Hexidecimal bitmap of 7 segment display LED array
79. \*/
80. static uint8\_t upper\_case\_to\_bitmap(char c) {
81. static uint8\_t upper\_case\_alpha[26] = {
82. 0x77, /\* A \*/
83. 0x7F, /\* B \*/
84. 0x4E, /\* C \*/
85. 0x7E, /\* D \*/
86. 0x4F, /\* E \*/
87. 0x47, /\* F \*/
88. 0x5E, /\* G \*/
89. 0x37, /\* H \*/
90. 0x30, /\* I \*/
91. 0x3C, /\* J \*/
92. 0x57, /\* K \*/
93. 0x0E, /\* L \*/
94. 0x54, /\* M \*/
95. 0x76, /\* N \*/
96. 0x7E, /\* O \*/
97. 0x67, /\* P \*/
98. 0x6B, /\* Q \*/
99. 0x46, /\* R \*/
100. 0x5B, /\* S \*/
101. 0x0F, /\* T \*/
102. 0x3E, /\* U \*/
103. 0x3E, /\* V \*/
104. 0x2A, /\* W \*/
105. 0x37, /\* X \*/
106. 0x3B, /\* Y \*/
107. 0x5B  /\* Z \*/
108. };
109. return upper\_case\_alpha[c - 'A'];
110. };
111. // Special character bitmaps
112. #define HYPHEN\_BIMAP (0x01)
113. #define DECIMAL\_POINT\_BITMAP (0x80)
114. uint8\_t seven\_segment\_display\_encode\_character(char c) {
115. // Initialize output value
116. uint8\_t ret = 0;
118. // Parse input value and assign output
119. if (c >= '0' && c <= '9') {
120. ret = numeral\_to\_bitmap(c);
121. } else if (c >= 'a' && c <= 'z') {
122. ret = lower\_case\_to\_bitmap(c);
123. } else if (c >= 'A' && c <= 'Z') {
124. ret = upper\_case\_to\_bitmap(c);
125. } else if (c == '-') {
126. ret = HYPHEN\_BIMAP;
127. } else if (c == '.') {
128. ret = DECIMAL\_POINT\_BITMAP;
129. }
130. return ret;
131. }
132. size\_t seven\_segment\_display\_encode\_string(const char \*in, uint8\_t \*out, size\_t in\_size) {
133. // Initialize counter/return variable
134. size\_t count = 0;
135. // Obtain first character of input array
136. char current\_char = \*in;
137. // Iterate through input array, encode value, assign to output array
138. while (current\_char && (count < in\_size)) {
140. // If this is not the first character to encode and it is a DP
141. if ((count != 0) && (current\_char == '.')) {
142. // Check if DP is asserted on previous encoded character
143. uint8\_t dp\_asserted = \*(out - 1) & DECIMAL\_POINT\_BITMAP;
144. // Assert DP if not asserted on previous encoded character
145. if (!dp\_asserted) {
146. \*(out - 1) |= DECIMAL\_POINT\_BITMAP;
147. current\_char = \*++in;
148. \*out = 0;
149. count--;
150. }
151. } else {
152. // Encode current character to bitmap and assign to output array, then increment output pointer
153. \*out++ = seven\_segment\_display\_encode\_character(current\_char);
155. // Increment input pointer and assign the value stored at its location to current\_char
156. current\_char = \*++in;
157. }
158. ++count;
159. }
160. return count;
161. }

**Hardware Description:**

The voltage divider circuit used in this experiment is shown in Figure 1. The node labeled “To FRDM MKL25Z4” was connected to Pin 20 of Port E, which corresponded to ADC0. “VDD” was connected to pin 8 on the board, and “GND” was connected to pin 12. The 8 LEDs of each seven-segment display device were connected to pins 0-7 of Ports C and D, respectively. The common cathode of both displays was connected to the board’s ground pin.

Diagram, schematic

Description automatically generated

Figure : Thermistor voltage divider circuit

**Flow Diagram:**

See Figure 2 for a top-level flow diagram that describes the execution of the code used in Part 1 of this laboratory exercise.

**Diagram

Description automatically generated**

Figure : Part 1 code flow diagram

# IR Sensor

**Description:**

For this laboratory exercise, we constructed a proximity sensor using an IR transmitter and a photodiode. The photodiode was sensitive to the infrared light emitted by the IR LED. The IR LED was flashed on and off repeatedly. The voltage level at the anode of the photodiode was measured using the ADC on the KL25z while the emitter was on, and then again when it was turned off. When an object was near the photodiode, IR light bounced off the object, thereby increasing the total IR light that was absorbed by the photodiode. The increase in IR light resulted in greater conductivity through the diode, thereby lowering the voltage at the anode, and increasing the difference between the on and off states of the emitter.

The threshold values provided in the example proved to be incorrect for the IR emitter and photodiode used for this exercise. New thresholds were determined by recoding the voltage differential with varying degrees of object proximity using the debugger. Once the new thresholds were determined, the circuit behaved as expected, albeit with perhaps a smaller range than was anticipated.

**Source Code:**

**main.c**

1. #include "adc.h"
2. #include "display.h"
3. #include "ir\_led.h"
4. #define NUM\_SAMPLES\_TO\_AVG  (10)
5. #define T\_DELAY\_ON  (1000)
6. #define T\_DELAY\_OFF (1000)
7. static void delay(unsigned int time\_del)
8. {
9. time\_del = 10 \* time\_del;
10. while (time\_del--) {}
11. }
12. int main(void)
13. {
14. static int on\_brightness = 0;
15. static int off\_brightness = 0;
16. static int avg\_diff;
17. static int diff;
18. unsigned n;
19. adc\_init();
20. display\_init();
21. ir\_led\_init();
22. while (1)
23. {
24. diff = 0;
25. for (n = 0; n < NUM\_SAMPLES\_TO\_AVG; n++) {
26. // Measure IR level with IRLED off
27. ir\_led\_write\_state(IR\_LED\_OFF);
28. delay(T\_DELAY\_OFF);
29. off\_brightness = adc\_measure\_IR();
30. // Measure IR level with IRLED on
31. ir\_led\_write\_state(IR\_LED\_ON);
32. delay(T\_DELAY\_ON);
33. on\_brightness = adc\_measure\_IR();
34. // Calculate difference
35. diff += on\_brightness - off\_brightness;
36. }
37. // Calculate average difference
38. avg\_diff = diff / NUM\_SAMPLES\_TO\_AVG;
39. // Light RGB LED according to range
40. display\_range(avg\_diff);
41. }
42. }

**adc.c**

1. #include "adc.h"
2. #include "MKL25Z4.h"
3. #define IR\_PHOTOTRANSISTOR\_CHANNEL  (8)
4. #define NUM\_SAMPLES\_TO\_AVG          (10)
5. void adc\_init(void)
6. {
7. // Enable clock to ADC0
8. SIM->SCGC6 |= (1UL << SIM\_SCGC6\_ADC0\_SHIFT);
10. // Configure ADC
11. ADC0->CFG1 =    ADC\_CFG1\_ADLPC\_MASK     |   // Enable low-power configuration
12. ADC\_CFG1\_ADIV(0)        |   // Set clock divide ratio to 1
13. ADC\_CFG1\_ADLSMP\_MASK    |   // Long sample time
14. ADC\_CFG1\_MODE(3)        |   // 16-bit conversion mode
15. ADC\_CFG1\_ADICLK(0);         // Input clock = bus clock
16. // Select default voltage reference pins
17. ADC0->SC2 = ADC\_SC2\_REFSEL(0);
18. }
19. unsigned adc\_measure\_IR(void)
20. {
21. volatile unsigned res = 0;
22. // Start Conversion
23. ADC0->SC1[0] = IR\_PHOTOTRANSISTOR\_CHANNEL;
24. // Wait until conversion is complete
25. while (!(ADC0->SC1[0] & ADC\_SC1\_COCO\_MASK)) {}
26. res = ADC0->R[0];
27. // Complement result since voltage falls with increasing IR level
28. return (0xffff - res);
29. }

**ir\_led.c**

1. #include "ir\_led.h"
2. #include "MKL25Z4.h"
3. #define IR\_LED\_POS  (1)
4. #define MASK(x) (1ul << (x))
5. void ir\_led\_init(void)
6. {
7. // Enable clock to PORTB
8. SIM->SCGC5 |= SIM\_SCGC5\_PORTB\_MASK;
9. // Set IR pin to GPIO
10. PORTB->PCR[IR\_LED\_POS] &= ~PORT\_PCR\_MUX\_MASK;
11. PORTB->PCR[IR\_LED\_POS] |= PORT\_PCR\_MUX(1);
12. // Set pin as output
13. PTB->PDDR |= MASK(IR\_LED\_POS);
14. // Initialize state to off
15. ir\_led\_write\_state(IR\_LED\_OFF);
16. }
17. void ir\_led\_write\_state(int state)
18. {
19. if (state == IR\_LED\_OFF) {
20. PTB->PSOR = MASK(IR\_LED\_POS);
21. } else {
22. PTB->PCOR = MASK(IR\_LED\_POS);
23. }
24. }

**display.c**

1. #include "display.h"
2. #include "MKL25Z4.h"
3. #define MASK(x) (1ul << (x))
4. #define RED\_POS     (18)
5. #define GREEN\_POS   (19)
6. #define BLUE\_POS    (1)
7. enum led\_colors {
8. RED\_LED,
9. GREEN\_LED,
10. BLUE\_LED,
11. NUM\_LED\_COLORS
12. };
13. enum color\_names {
14. WHITE,
15. MAGENTA,
16. RED,
17. YELLOW,
18. BLUE,
19. GREEN,
20. NUM\_COLORS
21. };
22. static int color\_thresholds[NUM\_COLORS] = {
23. [WHITE]     = 0x20D,
24. [MAGENTA]   = 0x1E1,
25. [RED]       = 0x1B5,
26. [YELLOW]    = 0x189,
27. [BLUE]      = 0x15D,
28. [GREEN]     = 0x102
29. };
30. static int colors[NUM\_COLORS][NUM\_LED\_COLORS] = {
31. [WHITE]     = {[RED\_LED]= 1, [GREEN\_LED]= 1, [BLUE\_LED]= 1},
32. [MAGENTA]   = {[RED\_LED]= 1, [GREEN\_LED]= 0, [BLUE\_LED]= 1},
33. [RED]       = {[RED\_LED]= 1, [GREEN\_LED]= 0, [BLUE\_LED]= 0},
34. [YELLOW]    = {[RED\_LED]= 1, [GREEN\_LED]= 1, [BLUE\_LED]= 0},
35. [BLUE]      = {[RED\_LED]= 0, [GREEN\_LED]= 0, [BLUE\_LED]= 1},
36. [GREEN]     = {[RED\_LED]= 0, [GREEN\_LED]= 1, [BLUE\_LED]= 0}
37. };
38. static void control\_rgb\_led(int red, int green, int blue)
39. {
40. // Clear previous
41. PTB->PSOR |= MASK(RED\_POS) | MASK(GREEN\_POS);
42. PTD->PSOR |= MASK(BLUE\_POS);
43. // Set current
44. PTB->PCOR |= (red << RED\_POS) | (green << GREEN\_POS);
45. PTD->PCOR |= (blue << BLUE\_POS);
46. }
47. void display\_init(void)
48. {
49. // Enable clock to PORTB and PORTD
50. SIM->SCGC5 |= SIM\_SCGC5\_PORTB\_MASK | SIM\_SCGC5\_PORTD\_MASK;
51. // Set RBG pins as GPIO
52. PORTB->PCR[RED\_POS] &= ~PORT\_PCR\_MUX\_MASK;
53. PORTB->PCR[RED\_POS] |= PORT\_PCR\_MUX(1);
54. PORTB->PCR[GREEN\_POS] &= ~PORT\_PCR\_MUX\_MASK;
55. PORTB->PCR[GREEN\_POS] |= PORT\_PCR\_MUX(1);
56. PORTD->PCR[BLUE\_POS] &= ~PORT\_PCR\_MUX\_MASK;
57. PORTD->PCR[BLUE\_POS] |= PORT\_PCR\_MUX(1);
58. // Set pins as outputs
59. PTB->PDDR |= MASK(RED\_POS) | MASK(GREEN\_POS);
60. PTD->PDDR |= MASK(BLUE\_POS);
61. // Initialize as off
62. PTB->PSOR |= MASK(RED\_POS) | MASK(GREEN\_POS);
63. PTD->PSOR |= MASK(BLUE\_POS);
64. }
65. void display\_range(int b)
66. {
67. unsigned i;
68. for (i = 0; i < NUM\_COLORS; ++i) {
69. if (b > color\_thresholds[i]) {
70. break;
71. }
72. }
73. control\_rgb\_led(colors[i][RED\_LED], colors[i][GREEN\_LED], colors[i][BLUE\_LED]);
74. }

**Hardware Description:**

The general layout of the circuit used in this laboratory exercise is shown in Figure 3. The main difference between the circuit shown and the one used in the exercise was that the onboard RBG LED was used instead of an external component. As such, no external resistor was required for that leg of the circuit. Instead, 3 GPIO pins (Pins 18 and 19 on Port B, and Pin 1 on Port D) were used to control the red, green, and blue elements of the RGB led. The current flowing through the IR emitter was controlled by selectively pulling Pin 1 on Port B low. The voltage between R2 and the photo transistor was measured using channel 8 on ADC 0, which corresponded to Pin 0 on Port B. The photo transistor and IR emitter used in this lab differed from the one shown in the diagram due to availability issues. Instead, a 5mm 940nm LED/emitter pair from HiLetGo was used.

**Diagram, schematic

Description automatically generated**

Figure : IR Sensor Circuit Diagram

**Flow Diagram:**

The control flow of the program used to drive this circuit is shown in Figure 4. Delays were implemented using the delay() function described in the text, and therefore are imprecise and susceptible to interruptions. This program could be improved through the utilization of a timer peripheral.

Diagram

Description automatically generated

Figure : Flow diagram of IR proximity sensor program