Text

Description automatically generated

**School of Electrical and Computer Engineering**

**Fall 2022**

**ECE 5721**

**Embedded System Design**

**Lab 2: The Use of Audio Output and GPIO Output**

**Lab Report**

**Daniel Funke**

**October 20, 2022**

**Description:**

For this experiment I implemented the “Driving a Speaker” example from the book using the FRDM-KL25Z development board, a bread board, a headphone speaker, a capacitor, a resistor, and some jumpers. This experiment configured a GPIO pin to be an output whose state was toggled between high and low with a delay between state changes. The GPIO pin served as a sink, that is, the speaker turned on when the pin was driven low. The length of the delay between pin transitions altered the frequency of the sound produced by the speaker.

**Source Code:**

1. #include "MKL25Z4.h"
2. #define SPKR\_SHIFT (0)
3. #define MASK(x) (1ul << (x))
4. // Standard delay function
5. static void delay(volatile unsigned int time\_del)
6. {
7. while (time\_del--) {}
8. }
9. int main(void)
10. {
11. // Assign clock to Port B
12. SIM->SCGC5 |= SIM\_SCGC5\_PORTB\_MASK;
13. // Make pin GPIO
14. PORTB->PCR[SPKR\_SHIFT] &= ~PORT\_PCR\_MUX\_MASK;
15. PORTB->PCR[SPKR\_SHIFT] |= PORT\_PCR\_MUX(1);
16. // Set port to outputs
17. PTB->PDDR |= MASK(SPKR\_SHIFT);
19. // Turn on Speaker by clearing the port bit to zeros
20. PTB->PDOR |= MASK(SPKR\_SHIFT);
22. while(1) {
23. PTB->PTOR = MASK(SPKR\_SHIFT);
24. delay(20000);
25. }
26. }

**Hardware Description:**

For this experiment, Port B Pin 0 was used to drive the speaker. The speaker was a 31Ω component I removed from a broken headset and soldered to jumpers for easy connection to the breadboard. In addition to the speaker, I also utilized a 330Ω resistor in series with a 3.3µF capacitor to filter the output waveform. The final schematic is shown below:

A picture containing text, shoji, wall, toilet

Description automatically generated

**Flow diagram:**

**Diagram

Description automatically generated**

**Description:**

For this portion of the lab exercise we implemented a water level control system described by the diagram provided in the lab handout. This sytem included two sensors, Empty Tank (ETS) and Full Tank (FLS), and three output signals, Water in Tank, Tank Full, and Pump Signal, The purpose of this system is to attempt to maintain a state where the tank always has water in it despite constantly draining. To implement this experiment in hardware, we utilized buttons to set the state of the input sensors and LEDs to represent the state of the output signals.

**Source Code:**

1. #include "MKL25Z4.h"
2. #include <stdbool.h>
3. #define GREEN\_LED\_SHIFT (0)
4. #define RED\_LED\_SHIFT (1)
5. #define YELLOW\_LED\_SHIFT (2)
6. #define BLUE\_LED\_SHIFT (3)
7. #define ETS\_SHIFT (1)
8. #define FLS\_SHIFT (2)
9. #define MASK(x) (1ul << (x))
10. enum state {
11. STATE\_START,
12. STATE\_TANK\_EMPTY,
13. STATE\_WATER\_IN\_TANK,
14. STATE\_TANK\_FULL,
15. STATE\_TANK\_OVERFLOW
16. };
17. // Sensor states
18. static bool ETS; // Empty Tank Sensor
19. static bool FLS; // Full Tank Sensor
20. // System state
21. static int current\_state = STATE\_START;
22. static void init\_output(void)
23. {
24. // Assign clock to Port B
25. SIM->SCGC5 |= SIM\_SCGC5\_PORTB\_MASK;
26. // Make Port B Pins(0:3) GPIO
27. PORTB->PCR[GREEN\_LED\_SHIFT] &= ~PORT\_PCR\_MUX\_MASK;
28. PORTB->PCR[GREEN\_LED\_SHIFT] |= PORT\_PCR\_MUX(1);
29. PORTB->PCR[RED\_LED\_SHIFT] &= ~PORT\_PCR\_MUX\_MASK;
30. PORTB->PCR[RED\_LED\_SHIFT] |= PORT\_PCR\_MUX(1);
31. PORTB->PCR[YELLOW\_LED\_SHIFT] &= ~PORT\_PCR\_MUX\_MASK;
32. PORTB->PCR[YELLOW\_LED\_SHIFT] |= PORT\_PCR\_MUX(1);
33. PORTB->PCR[BLUE\_LED\_SHIFT] &= ~PORT\_PCR\_MUX\_MASK;
34. PORTB->PCR[BLUE\_LED\_SHIFT] |= PORT\_PCR\_MUX(1);
35. // Set Port B Pins(0:3) as outputs
36. PTB->PDDR |= MASK(GREEN\_LED\_SHIFT) | MASK(RED\_LED\_SHIFT) | MASK(YELLOW\_LED\_SHIFT) | MASK(BLUE\_LED\_SHIFT);
37. }
38. static void init\_sensors(void)
39. {
40. // Assign clock to Port C
41. SIM->SCGC5 |= SIM\_SCGC5\_PORTC\_MASK;
42. // Make Port C Pins(1:2) GPIO
43. PORTC->PCR[ETS\_SHIFT] &= ~PORT\_PCR\_MUX\_MASK;
44. PORTC->PCR[ETS\_SHIFT] |= PORT\_PCR\_MUX(1);
45. PORTC->PCR[FLS\_SHIFT] &= ~PORT\_PCR\_MUX\_MASK;
46. PORTC->PCR[FLS\_SHIFT] |= PORT\_PCR\_MUX(1);
47. // Set Port B Pins (1:2) as inputs
48. PTC->PDDR &= ~(MASK(ETS\_SHIFT) | MASK(FLS\_SHIFT));
49. }
50. static void init\_system(void)
51. {
52. // Set up GPIO ports
53. init\_output();
54. init\_sensors();
55. // Set initial sensor states
56. ETS = false;
57. FLS = false;
58. OFS = false;
59. }
60. static void update\_sensors(void)
61. {
62. if (!(PTC->PDIR & MASK(ETS\_SHIFT))) {
63. ETS = !ETS;
64. }
65. if (!(PTC->PDIR & MASK(FLS\_SHIFT))) {
66. FLS = !FLS;
67. }
68. }
69. static void update\_output(void)
70. {
71. switch(current\_state) {
72. case STATE\_START:
73. // All LEDs = OFF
74. PTB->PSOR |= MASK(GREEN\_LED\_SHIFT)| MASK(BLUE\_LED\_SHIFT) | MASK(YELLOW\_LED\_SHIFT) | MASK(RED\_LED\_SHIFT);
75. break;
76. case STATE\_TANK\_EMPTY:
77. // Blue LED = ON, others = OFF
78. PTB->PCOR |= MASK(BLUE\_LED\_SHIFT);
79. PTB->PSOR |= MASK(GREEN\_LED\_SHIFT)| MASK(YELLOW\_LED\_SHIFT) | MASK(RED\_LED\_SHIFT);
80. break;
81. case STATE\_WATER\_IN\_TANK:
82. // Green and Blue LEDs = ON, Yellow and Red LEDs = OFF
83. PTB->PCOR |= MASK(BLUE\_LED\_SHIFT) | MASK(GREEN\_LED\_SHIFT);
84. PTB->PSOR |= MASK(YELLOW\_LED\_SHIFT) | MASK(RED\_LED\_SHIFT);
85. break;
86. case STATE\_TANK\_FULL:
87. // Yellow and Blue LEDs = ON, Green and Red LEDs = OFF
88. PTB->PCOR |= MASK(BLUE\_LED\_SHIFT) | MASK(YELLOW\_LED\_SHIFT);
89. PTB->PSOR |= MASK(GREEN\_LED\_SHIFT)| MASK(RED\_LED\_SHIFT);
90. break;
91. case STATE\_TANK\_OVERFLOW:
92. // Red LED ON, others OFF
93. PTB->PCOR |= MASK(RED\_LED\_SHIFT);
94. PTB->PSOR |= MASK(GREEN\_LED\_SHIFT)| MASK(BLUE\_LED\_SHIFT) | MASK(YELLOW\_LED\_SHIFT);
95. break;
96. }
97. }
98. static void update\_state(void)
99. {
100. switch(current\_state) {
101. case STATE\_START:
102. if (ETS) {
103. current\_state = STATE\_TANK\_EMPTY;
104. } else {
105. current\_state = STATE\_START;
106. }
107. break;
108. case STATE\_TANK\_EMPTY:
109. if (ETS) {
110. current\_state = STATE\_TANK\_EMPTY;
111. } else {
112. current\_state = STATE\_WATER\_IN\_TANK;
113. }
114. break;
115. case STATE\_WATER\_IN\_TANK:
116. if (ETS) {
117. current\_state = STATE\_TANK\_EMPTY;
118. } else if (FLS) {
119. current\_state = STATE\_TANK\_FULL;
120. } else {
121. current\_state = STATE\_WATER\_IN\_TANK;
122. }
123. break;
124. case STATE\_TANK\_FULL:
125. if (!FLS) {
126. current\_state = STATE\_WATER\_IN\_TANK;
127. } else if (OFS) {
128. current\_state = STATE\_TANK\_OVERFLOW;
129. } else {
130. current\_state = STATE\_TANK\_FULL;
131. }
132. break;
133. case STATE\_TANK\_OVERFLOW:
134. if (!OFS) {
135. current\_state = STATE\_TANK\_FULL;
136. } else {
137. current\_state = STATE\_TANK\_OVERFLOW;
138. }
139. break;
140. }
141. }
142. int main(void)
143. {
144. init\_system();
145. while(1)
146. {
147. update\_sensors();
148. update\_state();
149. update\_output();
150. }
151. }

**Hardware Description:**

As states in the descriptions, buttons were used to toggle the state of the ETS and FLS sensor states. The figure below describes how these buttons were connected to the development board. For each button, 100kΩ pull up resistor was connected to its respective pin on the FRDM-KL25Z target board as well as one side of the single pull, single throw switch. The other side of the switch was connected to ground. When the switch was toggled, the state of the sensor was inverted in the code.

**Diagram, schematic

Description automatically generated**

The output signals were implemented using different color LEDs. The different signals and their respective LED colors are described in the table below. The software turned on these sensors by writing a zero to (grounding) the GPIO pin connected to the desired LED. This caused current to flow from the 3.3VDD source and through the LED. The circuit diagram depicting this set up is also shown below.

|  |  |
| --- | --- |
| Output Signal | LED Color |
| Pump Signal | Blue |
| Empty Tank Signal (ETS) | Green |
| Full Tank Level Sensor (FLS) | Yellow |

**Diagram

Description automatically generated**

**Flow Diagram:**

**Diagram

Description automatically generated**

**Description:**

For Part Three of this laboratory, the system described in Part Two was extended to include another sensor and output signal. This new sensor indicated whether the tank was in an overflow state. While in this state, the three output signals described in Part Two would be turned off, and a new output signal (OFS) would turn on, indicating that the pump was off while the system was in overflow. Once enough water had drained from the tank, thereby causing the system to exit the overflow state, the pump and full tank signals would be reasserted, and the overflow signal would be turned off. The following section provides the code for how this was implemented in C.

**Source Code:**

1. #include "MKL25Z4.h"
2. #include <stdbool.h>
3. #define GREEN\_LED\_SHIFT (0)
4. #define RED\_LED\_SHIFT (1)
5. #define YELLOW\_LED\_SHIFT (2)
6. #define BLUE\_LED\_SHIFT (3)
7. #define ETS\_SHIFT (1)
8. #define FLS\_SHIFT (2)
9. #define OFS\_SHIFT (9)
10. #define MASK(x) (1ul << (x))
11. enum state {
12. STATE\_START,
13. STATE\_TANK\_EMPTY,
14. STATE\_WATER\_IN\_TANK,
15. STATE\_TANK\_FULL,
16. STATE\_TANK\_OVERFLOW
17. };
18. // Sensor states
19. static bool ETS; // Empty Tank Sensor
20. static bool FLS; // Full Tank Sensor
21. static bool OFS; // Over Flow Sensor
22. // System state
23. static int current\_state = STATE\_START;
24. static void init\_output(void)
25. {
26. // Assign clock to Port B
27. SIM->SCGC5 |= SIM\_SCGC5\_PORTB\_MASK;
28. // Make Port B Pins(0:3) GPIO
29. PORTB->PCR[GREEN\_LED\_SHIFT] &= ~PORT\_PCR\_MUX\_MASK;
30. PORTB->PCR[GREEN\_LED\_SHIFT] |= PORT\_PCR\_MUX(1);
31. PORTB->PCR[RED\_LED\_SHIFT] &= ~PORT\_PCR\_MUX\_MASK;
32. PORTB->PCR[RED\_LED\_SHIFT] |= PORT\_PCR\_MUX(1);
33. PORTB->PCR[YELLOW\_LED\_SHIFT] &= ~PORT\_PCR\_MUX\_MASK;
34. PORTB->PCR[YELLOW\_LED\_SHIFT] |= PORT\_PCR\_MUX(1);
35. PORTB->PCR[BLUE\_LED\_SHIFT] &= ~PORT\_PCR\_MUX\_MASK;
36. PORTB->PCR[BLUE\_LED\_SHIFT] |= PORT\_PCR\_MUX(1);
37. // Set Port B Pins(0:3) as outputs
38. PTB->PDDR |= MASK(GREEN\_LED\_SHIFT) | MASK(RED\_LED\_SHIFT) | MASK(YELLOW\_LED\_SHIFT) | MASK(BLUE\_LED\_SHIFT);
39. }
40. static void init\_sensors(void)
41. {
42. // Assign clock to Port C
43. SIM->SCGC5 |= SIM\_SCGC5\_PORTC\_MASK;
44. // Make Port C Pins(1,2,9) GPIO
45. PORTC->PCR[ETS\_SHIFT] &= ~PORT\_PCR\_MUX\_MASK;
46. PORTC->PCR[ETS\_SHIFT] |= PORT\_PCR\_MUX(1);
47. PORTC->PCR[FLS\_SHIFT] &= ~PORT\_PCR\_MUX\_MASK;
48. PORTC->PCR[FLS\_SHIFT] |= PORT\_PCR\_MUX(1);
49. PORTC->PCR[OFS\_SHIFT] &= ~PORT\_PCR\_MUX\_MASK;
50. PORTC->PCR[OFS\_SHIFT] |= PORT\_PCR\_MUX(1);
51. // Set Port C Pins (1,2,9) as inputs
52. PTC->PDDR &= ~(MASK(ETS\_SHIFT) | MASK(FLS\_SHIFT) | MASK(OFS\_SHIFT));
53. }
54. static void init\_system(void)
55. {
56. // Set up GPIO ports
57. init\_output();
58. init\_sensors();
59. // Set initial sensor states
60. ETS = false;
61. FLS = false;
62. OFS = false;
63. }
64. static void update\_sensors(void)
65. {
66. if (!(PTC->PDIR & MASK(ETS\_SHIFT))) {
67. ETS = !ETS;
68. }
69. if (!(PTC->PDIR & MASK(FLS\_SHIFT))) {
70. FLS = !FLS;
71. }
72. if (!(PTC->PDIR & MASK(OFS\_SHIFT))) {
73. OFS = !OFS;
74. }
75. }
76. static void update\_output(void)
77. {
78. switch(current\_state) {
79. case STATE\_START:
80. // All LEDs = OFF
81. PTB->PSOR |= MASK(GREEN\_LED\_SHIFT)| MASK(BLUE\_LED\_SHIFT) | MASK(YELLOW\_LED\_SHIFT) | MASK(RED\_LED\_SHIFT);
82. break;
83. case STATE\_TANK\_EMPTY:
84. // Blue LED = ON, others = OFF
85. PTB->PCOR |= MASK(BLUE\_LED\_SHIFT);
86. PTB->PSOR |= MASK(GREEN\_LED\_SHIFT)| MASK(YELLOW\_LED\_SHIFT) | MASK(RED\_LED\_SHIFT);
87. break;
88. case STATE\_WATER\_IN\_TANK:
89. // Green and Blue LEDs = ON, Yellow and Red LEDs = OFF
90. PTB->PCOR |= MASK(BLUE\_LED\_SHIFT) | MASK(GREEN\_LED\_SHIFT);
91. PTB->PSOR |= MASK(YELLOW\_LED\_SHIFT) | MASK(RED\_LED\_SHIFT);
92. break;
93. case STATE\_TANK\_FULL:
94. // Yellow and Blue LEDs = ON, Green and Red LEDs = OFF
95. PTB->PCOR |= MASK(BLUE\_LED\_SHIFT) | MASK(YELLOW\_LED\_SHIFT);
96. PTB->PSOR |= MASK(GREEN\_LED\_SHIFT)| MASK(RED\_LED\_SHIFT);
97. break;
98. case STATE\_TANK\_OVERFLOW:
99. // Red LED ON, others OFF
100. PTB->PCOR |= MASK(RED\_LED\_SHIFT);
101. PTB->PSOR |= MASK(GREEN\_LED\_SHIFT)| MASK(BLUE\_LED\_SHIFT) | MASK(YELLOW\_LED\_SHIFT);
102. break;
103. }
104. }
105. static void update\_state(void)
106. {
107. switch(current\_state) {
108. case STATE\_START:
109. if (ETS) {
110. current\_state = STATE\_TANK\_EMPTY;
111. } else {
112. current\_state = STATE\_START;
113. }
114. break;
115. case STATE\_TANK\_EMPTY:
116. if (ETS) {
117. current\_state = STATE\_TANK\_EMPTY;
118. } else {
119. current\_state = STATE\_WATER\_IN\_TANK;
120. }
121. break;
122. case STATE\_WATER\_IN\_TANK:
123. if (ETS) {
124. current\_state = STATE\_TANK\_EMPTY;
125. } else if (FLS) {
126. current\_state = STATE\_TANK\_FULL;
127. } else {
128. current\_state = STATE\_WATER\_IN\_TANK;
129. }
130. break;
131. case STATE\_TANK\_FULL:
132. if (!FLS) {
133. current\_state = STATE\_WATER\_IN\_TANK;
134. } else if (OFS) {
135. current\_state = STATE\_TANK\_OVERFLOW;
136. } else {
137. current\_state = STATE\_TANK\_FULL;
138. }
139. break;
140. case STATE\_TANK\_OVERFLOW:
141. if (!OFS) {
142. current\_state = STATE\_TANK\_FULL;
143. } else {
144. current\_state = STATE\_TANK\_OVERFLOW;
145. }
146. break;
147. }
148. }
149. int main(void)
150. {
151. init\_system();
152. while(1)
153. {
154. update\_sensors();
155. update\_state();
156. update\_output();
157. }
158. }

**Hardware Description:**

The circuit described in the previous portion of this lab was extended to include another “sensor.” This required another 100kΩ pull-up resistor and another single-pull, single-throw switch. Once again, the pull-up resistor was connected to an input GPIO pin (Port C, Pin 9), which was pulled low (grounded) when the switch was pressed. The diagram below describes this circuit.

Diagram, schematic

Description automatically generated

A fourth LED was added to the output signal circuit described in the previous section of this laboratory. This LED was red and represented the Overflow Output Signal. It was wired up in a similar manner as the LEDs from Part Two of this lab, and is depicted in the diagram below.

Diagram, schematic

Description automatically generated

**Flow Diagram:**

The flow diagram for this portion of the lab is the exact same as the previous portion, as the workflow remains the same. That is, initialize the system (GPIO pins and “system state”), and then constantly poll the sensors, update the system state based on the sensor values, update the output signals based on system state, and then begin polling the sensors again.

Diagram

Description automatically generated