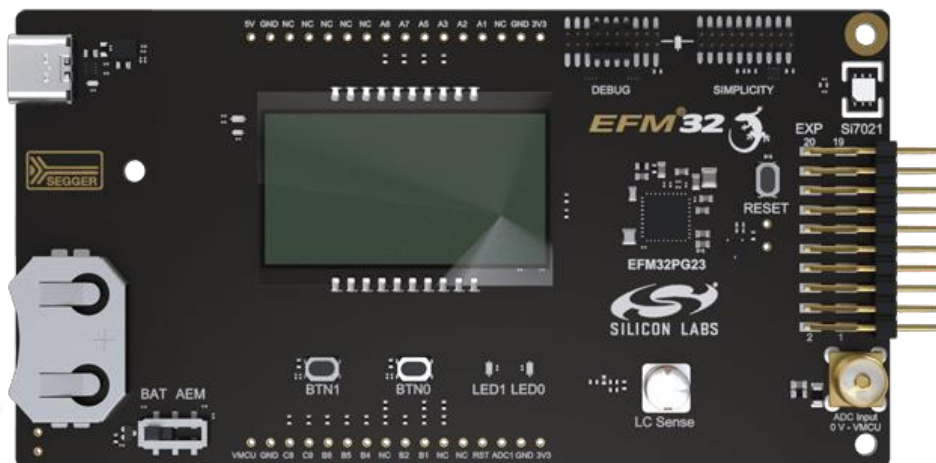
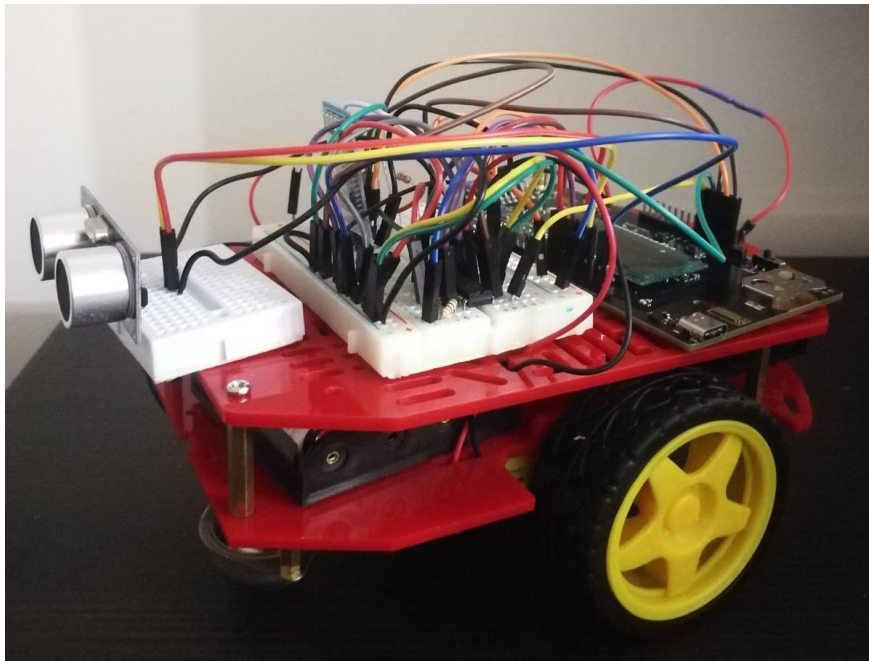


דוח מסכם בקורס מיקרו בקרים

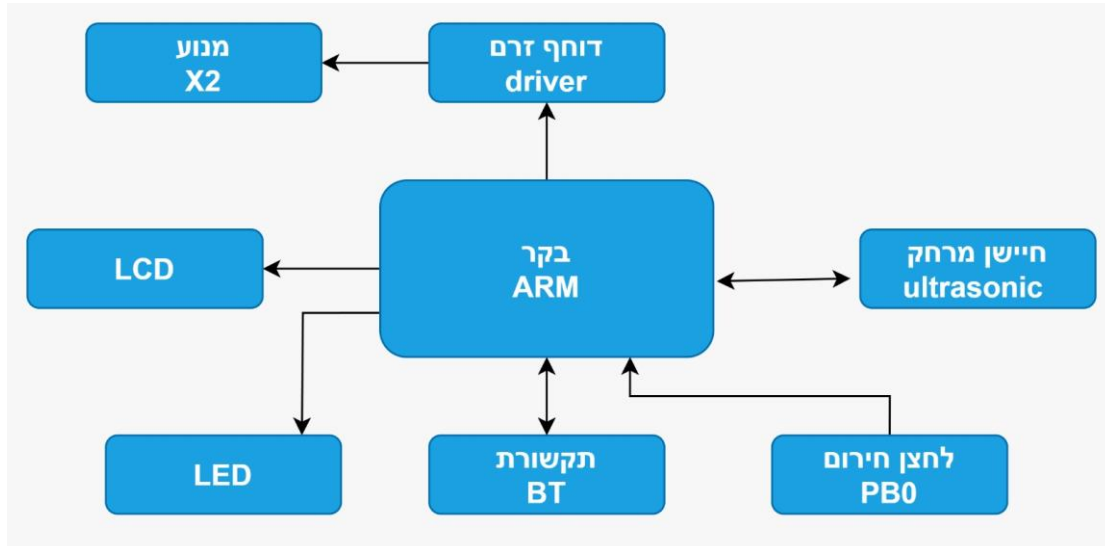




תוכן עניינים

3.....	דיאגרמת מלבנים פונקציונלית:
3.....	מטרת המערכת:
4.....	תרשים זרימה לפעולת המערכת:
5.....	תכנון חלוקת משאבים בבקר:
5.....	הגדרות מערכת:
5.....	מונים:
5.....	ערוצי תקשורת:
6.....	פינים:
6.....	LCD:
7.....	צילומי מסך הסקופ:
7.....	PWM של המנוע:
9.....	צילום של הטריגר של חיישן המרחק:
9.....	צילום של ECHO של חיישן המרחק:
10.....	צילום של הבלוטוס:
11.....	רשימה של כל Prototypes:
12.....	צילום מסך של קוד התוכנית הראשית, פונקציית האתחול, פונקציות נוספות וכן פונקציות פסיקה:
12.....	הגדרת ספריות, קבועים ומשתנים:
13.....	פונקציית בלוטוס:
13.....	פונקציית app init:
16.....	פונקציית measure distance:
16.....	פונקציית speed motor:
17.....	פונקציית app process action:
19.....	הגדרה ראשונית של ה-Software components:
20.....	חישוב ערך ה-TOP של שני הטיימרים:
21.....	חישוב ערך ה-Duty count:
22.....	צילומי מסך רלוונטיים מהדיבגר:
23.....	עבור TIMERO:
24.....	עבור TIMER1:
25.....	עבור EUSART1:
26.....	תקלות ובעיות שנתקלנו בהן תוך כדי הפרויקט:
27.....	קישור לסרטון:

דיאגרמת מלבנים פונקציונלית:



מטרת המערכת:

המערכת מורכבת ממיקרו בקר מסוג ARM EFM32PG28, חיישן מרחק אולטראסוניק, 2 מנועים, דוחף זרם, מסך תצוגה LCD, לחצן חירום, תקשורת בלוטוס ונורת לד.

הרכב נשלט על ידי תקשורת הבלוטוס, אשר היא קובעת את מהירות המכונית ואת הכיוון הנסיעה של המכונית, תקשורת הבלוטוס הינה תקשורת טורית UART אסינכרונית, יש לנו ארבעה אפשרויות למהירות שונות וכיווני נסיעה שונים (יש בנוסף גם מצב ניוטרל, ומצב ERROR).

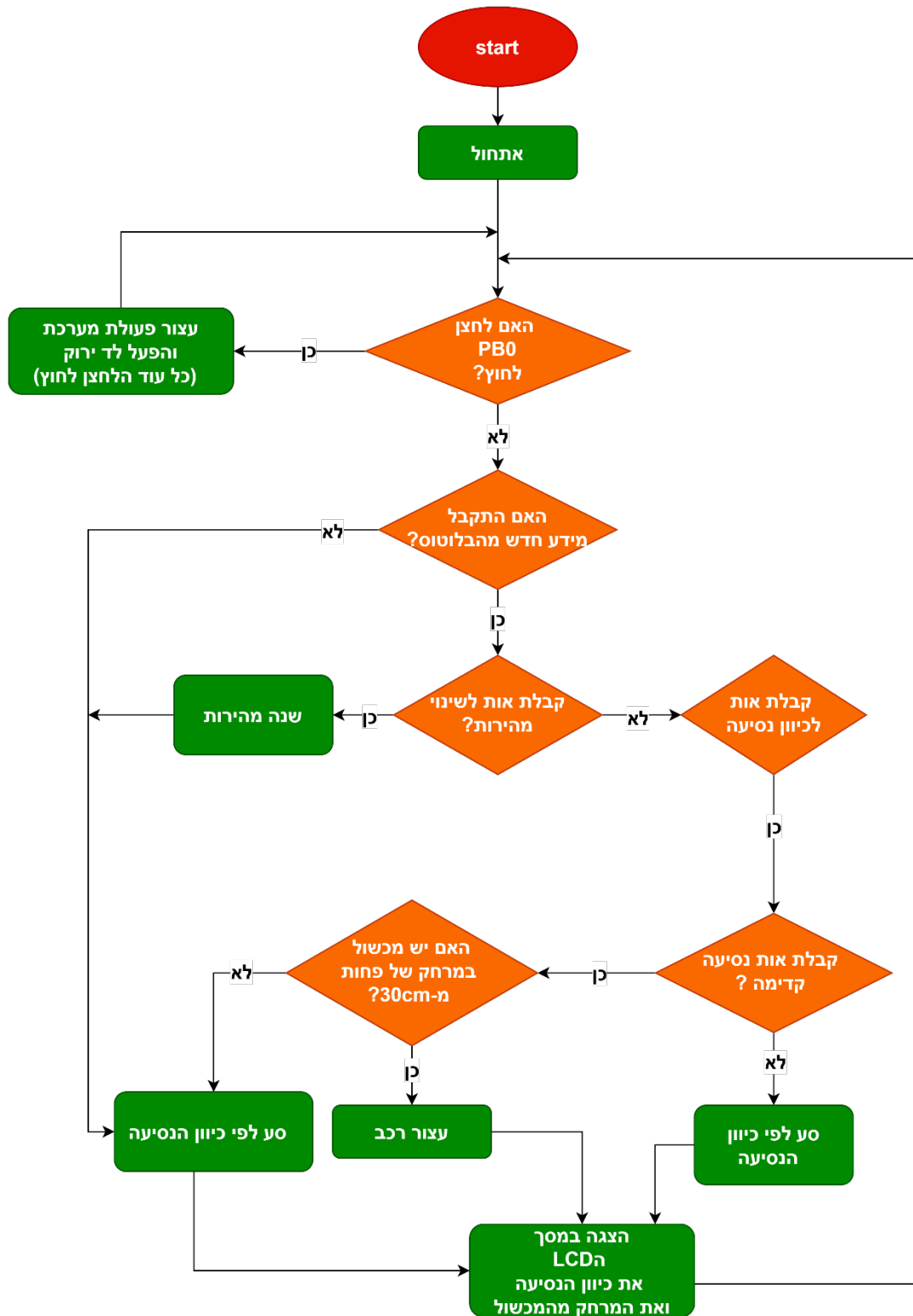
בכל רגע אנו מציגים במסך LCD את כיוון הנסיעה של המכונית, ואת המרחק של המכונית מהמכשול.

כיווני נסיעה שהגדרנו ותוצג במסך ה-LCD :

1. "F" - נסיעה קדימה
2. "B" - נסיעה אחורה
3. "R" - נסיעה ימינה
4. "L" - נסיעה שמאלה
5. "N" - מצב עצירה הרכב לא בנסיעה
6. "S" - מצב עצירה במידה ויש מכשול מלפנים במרחק קטן מ-30 ס"מ

בכדי לא לקחת סיכונים יש לנו לחצן חירום, כאשר אנו לוחצים עליו, המכונית מפסיקה לפעול, תוצג במסך LCD מצב "ERROR" ונדלק לד ירוק עד שאנחנו משחררים את הלחצן, לאחר שיחרור הלחצן הרכב עובר למצב "N".

תרשים זרימה לפעולת המערכת:





תכנון חלוקת משאבים בבקר:

Port	Pin	Description	Mode
B	1	כפתור חירום	Input
C	10,11	אפשר למנוע	Output Pushpull
D	6,7,8,9	Driver inputs	Output Pushpull
D	10	Echo	Input
D	11	Trigger	Output Pushpull
D	13	Tx	Output Pushpull
D	14	Rx	Input
D	15	Green LED	Output Pushpull
Timer 0		Pwm של המנוע	
Timer 1		Pwm של החיישן מרחק	
EUSART1		Enable clock to EUSART1	9600
GPIO		Enable GPIO	
LCD		הצגה של המידע של המסך	

הגדרות מערכת:

- שעון המערכת HFXO 39MHZ
- שעון מאופשר ל-GPIO.
- שעון מאופשר ל-TIMER1 ו-TIMER0
- שעון מאופשר ל-EUSART1.
- שעון מאופשר ל-LCD

מונים:

- 32-bit Timer/Counters (TIMER0): שימוש בערוצים 0 ו-1 ללא Prescale שעובדים במצב PWM בתדר 300HZ עם Duty cycle משתנה במהלך התוכנית (95%,80%,70%,60%).
- 16-bit Timer/Counters (TIMER1): שימוש בערוץ 0 עם Prescale 32 שעובד במצב PWM בתדר 20HZ עם פולס של 10μsec.

ערוצי תקשורת:

EUSART1 עם קצב של 9600 [Baud Rate] לתקשורת עם הפלאפון.

פינים:

- PD6,PD7,PD8,PD9 : לטובת שינוי כיוון סיבוב המנועים ע"י פונקציות API-SET/CLEAR
- PC10,PC11 : לטובת אות ה-PWM לקביעת מהירות המנועים במצב .PUSH PULL

:LCD

אנו מציגים את המידע בתצוגת 14 המקטעים, אשר בעזרתה ניתן לכתוב תו אלפא-נומרי.

התקשורת הינה תקשורת מקבילית, ניגשים דרך ה-GPIO לכל אחד מהפינים המחוברים.



אלו הלדים אותם אנו
מדליקים, למען הסר
ספק, מדובר על השורה
התחתונה.

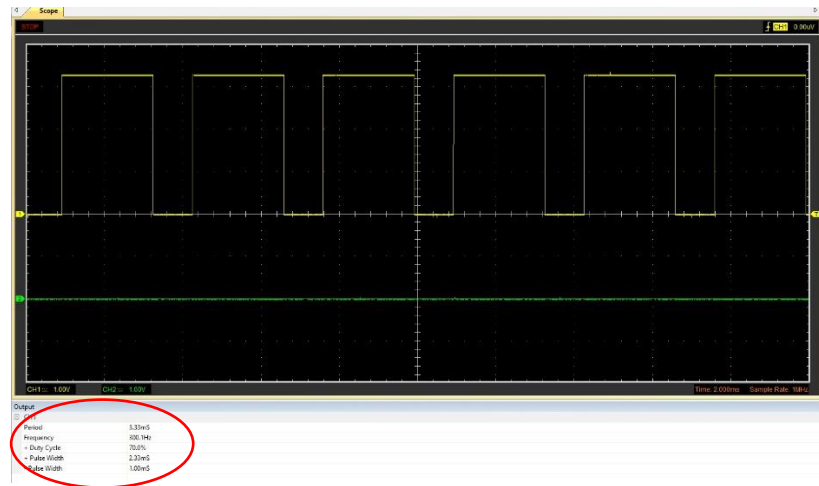
צילומי מסך הסקופ:

PWM של המנוע:

אנו מציגים את ה-PWM עם D.C של 70%.

הגדרנו את תדר המנוע כ-300 הרץ קבוע.

CH1	
Period	3.33mS
Frequency	300.1Hz
+ Duty Cycle	70.0%
+ Pulse Width	2.33mS
- Pulse Width	1.00mS



$$T_{on} = 2.33[ms]$$

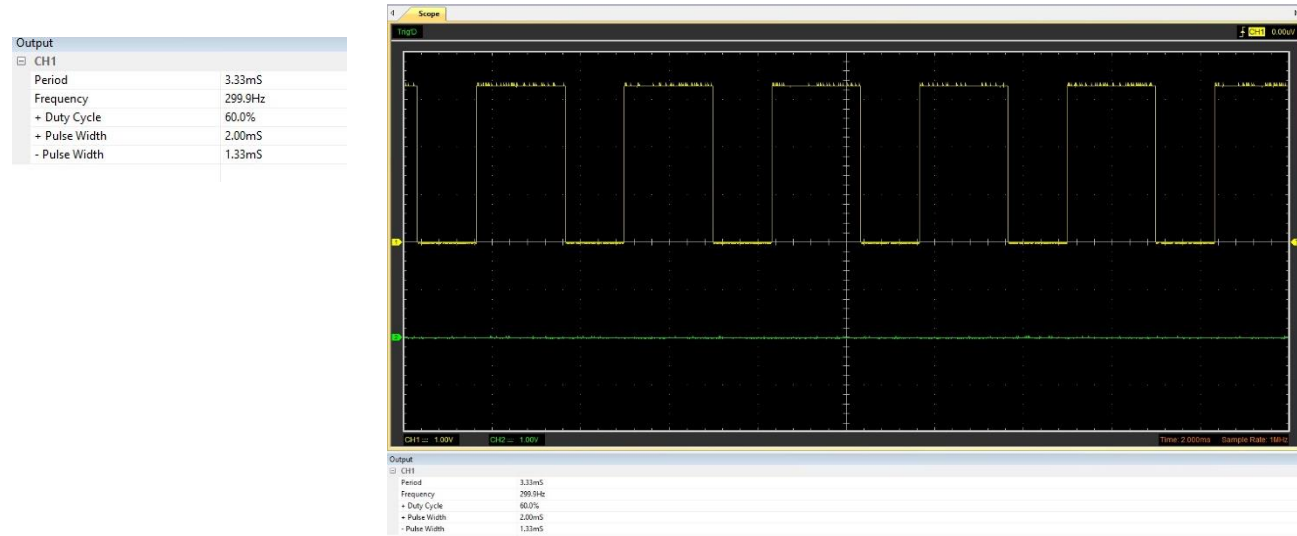
$$T_{off} = 1[ms]$$

$$Duty\ cycle = \frac{T_{on}}{T_{on} + T_{off}} * 100\% = \frac{2.33}{2.33 + 1} * 100\% = 70\%$$

הערה חשובה:

זהו הדיוטי סייקל ההתחלתי, כמובן שכאשר אנו משנים את המהירות ה-D.C משתנה בהתאמה.

כעת נבצע את המדידה למהירות הכי נמוכה, שהיא D.C=60%

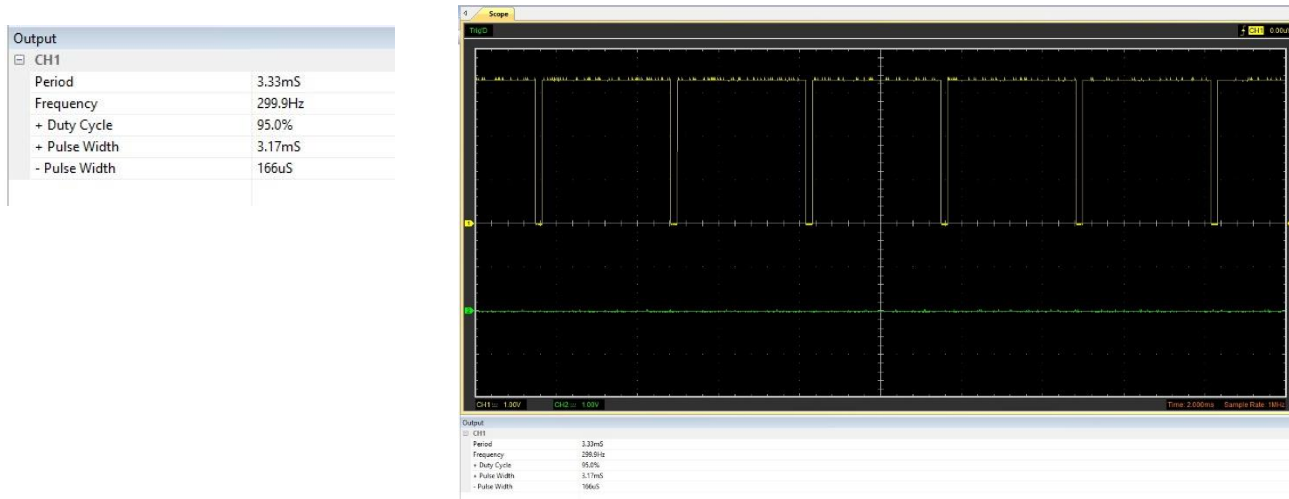


$$T_{on} = 2[ms]$$

$$T_{off} = 1.33[ms]$$

$$Duty\ cycle = \frac{T_{on}}{T_{on} + T_{off}} * 100\% = \frac{2}{2 + 1.33} * 100\% = 60\%$$

כעת נבצע את המדידה למהירות הכי נמוכה, שהיא D.C=95%

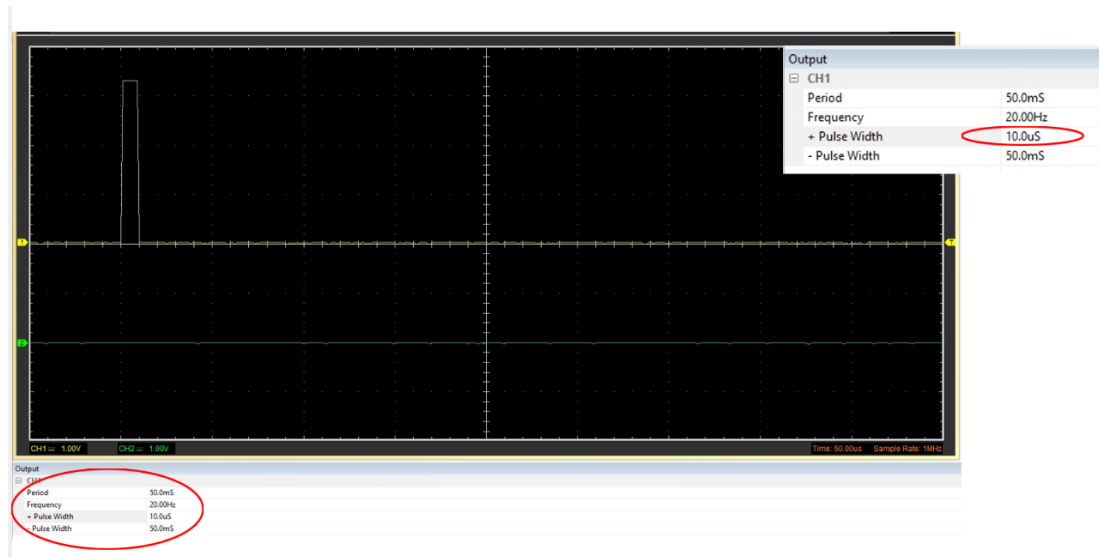


$$T_{on} = 3.17[ms]$$

$$T_{off} = 166[\mu s]$$

$$Duty\ cycle = \frac{T_{on}}{T_{on} + T_{off}} * 100\% = \frac{3.17 * 10^{-3}}{3.17 * 10^{-3} + 166 * 10^{-6}} * 100\% = 95\%$$

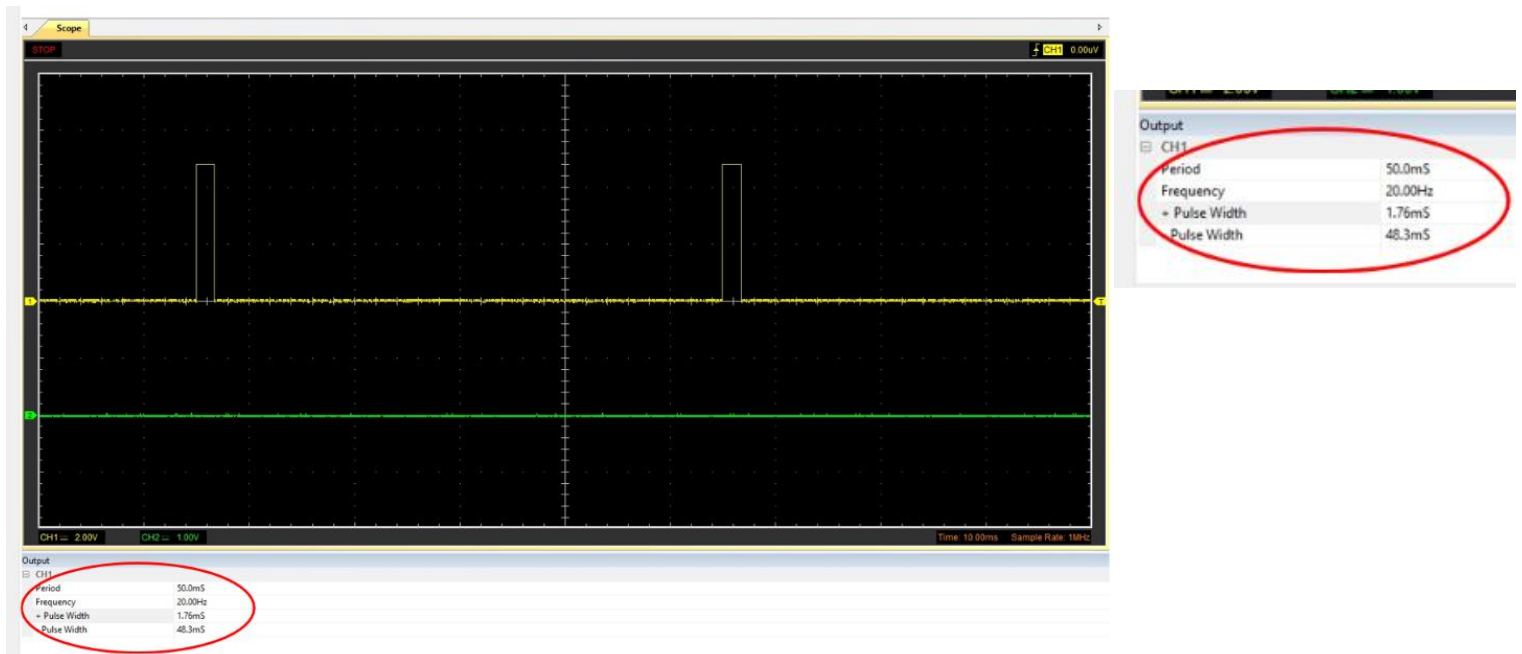
צילום של הטריג'ר של חיישן המרחק:



ניתן לראות בבירור שהזמן הפולס של הטריג'ר הינו 10 מיקרו שניות.

בקוד הגדרנו את תדר העבודה בתור 20 הרץ, על מנת לאפשר לחיישן זמן מחזור של 50 מילי שניות, בכדי לאפשר לחיישן לזהות מרחקים גדולים יותר ולמנוע ערבוב(הפרעה) בין אות הטריג'ר(האות שנשלח) לבין אות ECHO(האות החוזר).

צילום של ECHO של חיישן המרחק:

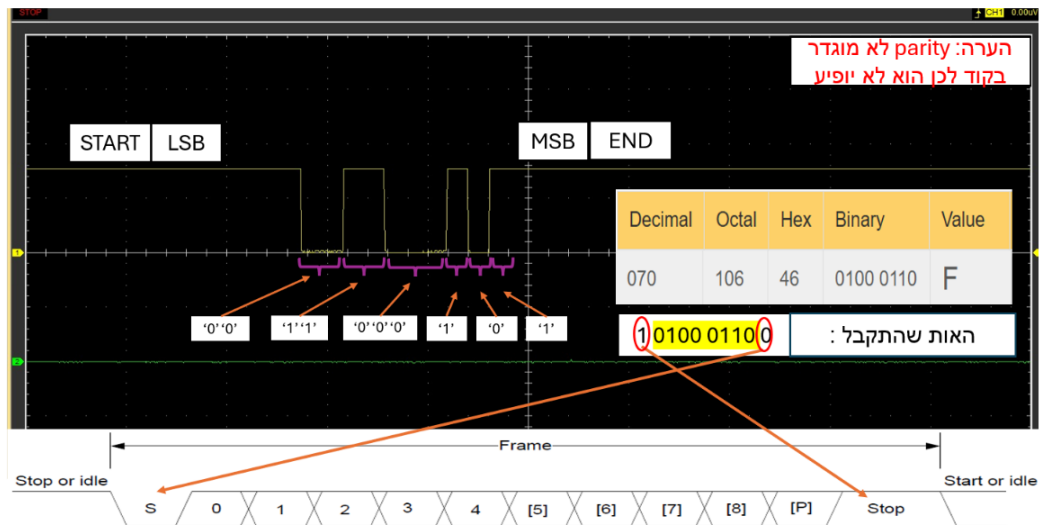


ככל שהמרחק קרוב יותר הפולס קצר יותר, ולהיפך.



צילום של הבלוטוס:

בתמונה הבאה ניתן לראות את שידור של האות F באמצעות הבלוטוס.



בקוד לא הגדרנו את סיבית הזוגיות (P), לכן היא לא מתקיימת פה.



רשימה של כל הPrototypes:

void EUSART1_RX_IRQHandler(void)

פונקציית פסיקה שמטפלת בנתונים המתקבלים מהמודול Bluetooth.

הפונקציה בודקת האם דגל קבלת הנתונים (RXFL) דלוק, כלומר אם התקבל נתון חדש. אם כן, הנתון נקרא מה-EUSART באמצעות EUSART_Rx(EUSART1), נשמר במשתנה .dir.

void app_init(void)

מבצע אתחול של המערכת, כולל הפעלת השעונים, קביעת תצורה לפינים והגדרת רכיבים כמו EUSART, טיימרים ו-LCD.

uint32_t measure_distance(void)

מודדת את המרחק באמצעות חיישן אולטרסוני ומחזירה את המרחק בס"מ.

מוציאה את הערך של ה cnt מתי שיש rising edge ולוקחת את הערך מתי שיש falling edge לאחר מכאן, בודקת את ההפרש בניהם ומחשבת את הזמן שהתקבל כתוצאה מכך.

לאחר מכאן מחשבת את המרחק של המכונית מהאובייקט.

void speed_motor(char speed)

משנה את מהירות המנוע בהתאם לקלט שהתקבל, שינוי המהירות נעשה באמצעות שינוי D.C.

void app_process_action(void)

מנהל את פעולת המכונית בהתאם לקלט המתקבל מהבלוטוס ומציג את הנתונים על המסך.

במידה ולחצן החירום לחוץ, נדלק לד ירוק ומוצג במסך הודעת שגיאה (כל עוד הלחצן לחוץ).

בכל מקרה אחר פעולת המכונית ממשיכה את פעולתה רגיל, רק עם התייחסות למרחק האובייקט מהרכב.



צילום מסך של קוד התוכנית הראשית, פונקציית האתחול,
פונקציות נוספות וכן פונקציות פסיקה:

הגדרת ספריות, קבועים ומשתנים:

```
12 #include "em_device.h"
13 #include "em_chip.h"
14 #include "em_cmu.h"
15 #include "em_gpio.h"
16 #include "em_timer.h"
17 #include "sl_segmentlcd.h"
18 #include "em_eusart.h"
19 #include <stddef.h>
20 #include <stdio.h>
21 #include <string.h>
22
23
24
25 // Desired PWM frequency and initial duty cycle
26 #define PWM_FREQ          300 // Frequency for motor PWM
27 #define INITIAL_DUTY_CYCLE 70 // Initial duty cycle percentage
28 #define PWM_FREQ_us       20 // Frequency for ultrasonic sensor PWM
29 #define INITIAL_DUTY_CYCLE_us 0.0002 // duty cycle for 10 micro pulse
30
31 // Duty cycle global variable
32 static volatile float dutyCycle;
33
34 // Timer and compare values for motor and ultrasonic sensor
35 uint32_t timerFreq, topValue, dutyCount;
36 uint32_t timerFreq_us, dutyCount_us, topValue_us;
37
38 // Global variables for Bluetooth communication and LCD
39 volatile char dir = 'N';
40 char* print_dir="N";
41 volatile bool newDataReceived = false;
42 char display[10];
43 int distance;
```



פונקצית בלוטוס:

זוהי פונקצית הפסיקה שלנו.

```
50 // EUSART1 Interrupt Handler
51 //Handles received data from Bluetooth module
52 void EUSART1_RX_IRQHandler(void)
53 {
54     uint32_t flags = EUSART_IntGet(EUSART1); // Get interrupt flags
55
56     if (flags & EUSART_IF_RXFL) // Check if RX interrupt occurred
57     {
58         // Read received data and echo it back
59         dir = EUSART_Rx(EUSART1); // Read received data
60         EUSART_Tx(EUSART1, dir); // Echo received data back
61
62         newDataReceived = true; // Set flag when new data is received
63     }
64
65     EUSART_IntClear(EUSART1, flags); // Clear all interrupt flags
66 }
```

פונקצית app init:

```
--
70 void app_init(void)
71 {
72     //Enable clocks for peripherals
73     CMU_ClockEnable(cmuClock_GPIO, true); // Enable clock for GPIO
74     CMU_ClockEnable(cmuClock_TIMER0, true); // Enable clock for TIMER0 (Motor PWM)
75     CMU_ClockEnable(cmuClock_TIMER1, true); // Enable clock for TIMER1 (Ultrasonic PWM)
76     CMU_ClockEnable(cmuClock_LCD, true); // Enable clock for LCD display
77     sl_segment_lcd_init(false); // Initialize segment LCD
78     CMU_ClockEnable(cmuClock_EUSART1, true); // Enable clock for EUSART1 (Bluetooth communication)
79
80     // Configure GPIO pins
81     GPIO_PinModeSet(gpioPortD, 15, gpioModePushPull, 0); // Green LED
82     // Configure push button
83     GPIO_PinModeSet(gpioPortB, 1, gpioModeInput, 0); // Configure PTN0 as inputs (Push button)
84     GPIO_PinModeSet(gpioPortC, 11, gpioModePushPull, 0); // Motor DRIVER Enable A
85     GPIO_PinModeSet(gpioPortC, 10, gpioModePushPull, 0); // Motor DRIVER Enable B
86     GPIO_PinModeSet(gpioPortD, 6, gpioModePushPull, 0); // DIR CONTROL 1A to Motor DRIVER
87     GPIO_PinModeSet(gpioPortD, 7, gpioModePushPull, 0); // DIR CONTROL 2A to TO Motor DRIVER
88     GPIO_PinModeSet(gpioPortD, 8, gpioModePushPull, 0); // DIR CONTROL 4A to TO Motor DRIVER
89     GPIO_PinModeSet(gpioPortD, 9, gpioModePushPull, 0); // DIR CONTROL 3A to TO Motor DRIVER
90     //installize pins for Ultrasonic sensor
91     GPIO_PinModeSet(gpioPortD, 11, gpioModePushPull, 0); //trigger
92     GPIO_PinModeSet(gpioPortD, 10, gpioModeInput, 0); //echo
93     // Configure UART pins
94     GPIO_PinModeSet(gpioPortD, 13, gpioModePushPull, 1); // TX
95     GPIO_PinModeSet(gpioPortD, 14, gpioModeInputPull, 1); // RX with pull-up
96
97
98     // Configure EUSART1
99     EUSART_UartInitTypeDef init = EUSART_UART_INIT_DEFAULT_HF; // Initialize EUSART1
100     init.baudrate = 9600; // Set baud rate
101     init.oversampling = eusartOVS4; // Set oversampling
102     init.enable = eusartEnable; // Enable EUSART
103
104
105     //Define local variables for timer0
106     TIMER_InitTypeDef timerInit = TIMER_INIT_DEFAULT;
107     TIMER_InitCC_TypeDef timerCCInit = TIMER_INITCC_DEFAULT;
108 }
```




```

109 //Define local variables for timer1
110 TIMER_Init_TypeDef timer1Init = TIMER_INIT_DEFAULT;
111 TIMER_InitCC_TypeDef timer1CCInit = TIMER_INITCC_DEFAULT;
112
113 timerInit.enable = false; // Don't start counter on initialization
114 timer1Init.enable = false;
115 timerCCInit.mode = timerCCModePWM; // Set PWM mode
116 timer1CCInit.mode = timerCCModePWM;
117 timer1Init.prescale = timerPrescale32; // Set prescaler 32 for TIMER1
118
119 // Initialize TIMER0 (Channel 0 and Channel 1)
120 TIMER_Init(TIMER0, &timerInit);
121 TIMER_InitCC(TIMER0, 0, &timerCCInit);
122 TIMER_InitCC(TIMER0, 1, &timerCCInit);
123
124 // Initialize timer1 with configuration for ultrasonic (channel 0)
125 TIMER_Init(TIMER1, &timer1Init);
126 TIMER_InitCC(TIMER1, 0, &timer1CCInit);
127
128
129 // (TIMER0) Route CC0 and CC1 outputs
130 GPIO->TIMERROUTE[0].ROUTEEN = GPIO_TIMER_ROUTEEN_CC0PEN | GPIO_TIMER_ROUTEEN_CC1PEN;
131
132 GPIO->TIMERROUTE[0].CC0ROUTE = (gpioPortC << _GPIO_TIMER_CC0ROUTE_PORT_SHIFT)
133 | (11 << _GPIO_TIMER_CC0ROUTE_PIN_SHIFT);
134
135 GPIO->TIMERROUTE[0].CC1ROUTE = (gpioPortC << _GPIO_TIMER_CC1ROUTE_PORT_SHIFT)
136 | (10 << _GPIO_TIMER_CC1ROUTE_PIN_SHIFT);
137
138
139 // (TIMER1) Route CC0 output to PD11 (for ultrasonic)
140 GPIO->TIMERROUTE[1].ROUTEEN = GPIO_TIMER_ROUTEEN_CC0PEN; // Changed index to 1 for TIMER1
141 GPIO->TIMERROUTE[1].CC0ROUTE = (gpioPortD << _GPIO_TIMER_CC0ROUTE_PORT_SHIFT)
142 | (11 << _GPIO_TIMER_CC0ROUTE_PIN_SHIFT);
143
144 //bluetooth*****
145 // Route EUSART1 TX and RX
146 GPIO->EUSARTROUTE[1].TXROUTE = (gpioPortD << _GPIO_EUSART_TXROUTE_PORT_SHIFT)
147 | (13 << _GPIO_EUSART_TXROUTE_PIN_SHIFT);
148 GPIO->EUSARTROUTE[1].RXROUTE = (gpioPortD << _GPIO_EUSART_RXROUTE_PORT_SHIFT)
149 | (14 << _GPIO_EUSART_RXROUTE_PIN_SHIFT);
150 GPIO->EUSARTROUTE[1].ROUTEEN = GPIO_EUSART_ROUTEEN_RXPEN | GPIO_EUSART_ROUTEEN_TXPEN;
151

```



```
151 //*****
152
153 //for motor*****
154 // Set top value to overflow at the desired PWM frequency
155 timerFreq = CMU_ClockFreqGet(cmuClock_TIMER0) / (timerInit.prescale + 1);
156 topValue = (timerFreq / PWM_FREQ);
157 TIMER_TopSet(TIMER0, topValue);
158
159 // Set dutyCycle global variable and compare values for initial duty cycle
160 dutyCycle = INITIAL_DUTY_CYCLE;
161 dutyCount = (topValue * INITIAL_DUTY_CYCLE) / 100;
162
163 TIMER_CompareSet(TIMER0, 0, dutyCount); // Set compare value for CC0
164 TIMER_CompareSet(TIMER0, 1, dutyCount); // Set compare value for CC1
165 //end for motor*****
166
167 //for ultrasonic*****
168 // Calculate timer frequency
169 timerFreq_us = CMU_ClockFreqGet(cmuClock_TIMER1) / (timer1Init.prescale + 1);
170 // Calculate top value for desired PWM frequency
171 topValue_us = timerFreq_us / PWM_FREQ_us;
172 TIMER_TopSet(TIMER1, topValue_us);
173 // Calculate compare value for desired duty cycle
174 dutyCount_us = (uint32_t)(topValue_us * INITIAL_DUTY_CYCLE_us);
175 TIMER_CompareSet(TIMER1, 0, dutyCount_us);
176 //end for ultrasonic*****
177 // Start TIMER0 and TIMER1
178 TIMER_Enable(TIMER0, true);
179 TIMER_Enable(TIMER1, true);
180
181
182 // Initialize EUSART
183 EUSART_UartInitHf(EUSART1, &init);
184 // Clear pending interrupts
185 EUSART_IntClear(EUSART1, _EUSART_IF_MASK);
186 // Enable RX interrupts
187 EUSART_IntEnable(EUSART1, EUSART_IEN_RXFL);
188 // Enable EUSART interrupts in NVIC
189 NVIC_ClearPendingIRQ(EUSART1_RX_IRQn);
190 NVIC_EnableIRQ(EUSART1_RX_IRQn);
191 }
```



פונקציית measure distance:

```
196@uint32_t measure_distance(void)
197 {
198     // Ensure the echo pin is ready (low)
199     while (GPIO_PinInGet(gpioPortD, 10)) {}
200
201     // Wait for the rising edge
202     while (!GPIO_PinInGet(gpioPortD, 10)) {}
203     uint32_t start = TIMER_CounterGet(TIMER1);
204
205     // Wait for the falling edge
206     while (GPIO_PinInGet(gpioPortD, 10)) {}
207     uint32_t end = TIMER_CounterGet(TIMER1);
208
209     // Calculate the distance in centimeters
210     uint32_t timerFreq = CMU_ClockFreqGet(cmuClock_TIMER1);
211     float timeElapsed = abs(end - start) / (float)timerFreq; // Time
212     uint32_t distance = ((timeElapsed*3.4) * 343000) / 2; // Distance in cm
213
214     return distance;
215 }
216 //*****
```

בעזרת TIMER_CounterGet() ניתן לקבל את ערכי ה-Cnt במצבי Rising edge
וFalling edge.

פונקציית speed motor:

```
217@void speed_motor(char speed)
218 {
219     if(speed=='1')
220     {
221         dutyCycle = 60;
222     }
223     else if(speed=='2')
224     {
225         dutyCycle = 70;
226     }
227     else if(speed=='3')
228     {
229         dutyCycle = 80;
230     }
231     else if(speed=='4')
232     {
233         dutyCycle = 95;
234     }
235     dutyCount = (topValue * dutyCycle) / 100;
236     TIMER_CompareSet(TIMER0, 0, dutyCount); // Set compare value for CC0
237     TIMER_CompareSet(TIMER0, 1, dutyCount); // Set compare value for CC1
238 }
```




פונקציית app_process_action:

```
240=void app_process_action(void)
241 {
242     GPIO_PinOutClear(gpioPortD, 6);
243     GPIO_PinOutClear(gpioPortD, 7);
244     GPIO_PinOutClear(gpioPortD, 8);
245     GPIO_PinOutClear(gpioPortD, 9);
246     while(!(GPIO_PinInGet(gpioPortB, 1)))// wait till PTN0 is released
247     {
248         GPIO_PinOutClear(gpioPortD, 6);
249         GPIO_PinOutClear(gpioPortD, 7);
250         GPIO_PinOutClear(gpioPortD, 8);
251         GPIO_PinOutClear(gpioPortD, 9);
252         print_dir="ERROR";
253         sl_segment_lcd_write(print_dir);
254         GPIO_PinOutSet(gpioPortD, 15);
255     }
256     while((GPIO_PinInGet(gpioPortB, 1)) )
257     {
258         GPIO_PinOutClear(gpioPortD, 15);
259         while (newDataReceived==true)
260         {
261             speed_motor(dir);
262             // Measure the distance and output it
263             distance = measure_distance();
264             if((distance<30) && (dir=='F'))
265             {
266                 print_dir="S";
267                 GPIO_PinOutClear(gpioPortD, 6);
268                 GPIO_PinOutClear(gpioPortD, 7);
269                 GPIO_PinOutClear(gpioPortD, 8);
270                 GPIO_PinOutClear(gpioPortD, 9);
271                 // Format the string using sprintf
272                 sprintf(display, "%d%s", distance, print_dir);
273                 // Use sl_segment_lcd_write to display the formatted string
274                 sl_segment_lcd_write(display);
275             }
276             else if(dir=='F')
277             {
278                 GPIO_PinOutClear(gpioPortD, 7);
279                 GPIO_PinOutClear(gpioPortD, 9);
280                 GPIO_PinOutSet(gpioPortD, 6);
281                 GPIO_PinOutSet(gpioPortD, 8);
282                 // Format the string using sprintf
283                 print_dir="-F";
284                 sprintf(display, "%d%s", distance, print_dir);
285                 // Use sl_segment_lcd_write to display the formatted string
286                 sl_segment_lcd_write(display);
287             }
288             else if(dir=='B')
289             {
290                 GPIO_PinOutClear(gpioPortD, 6);
291                 GPIO_PinOutClear(gpioPortD, 8);
292                 GPIO_PinOutSet(gpioPortD, 7);
293                 GPIO_PinOutSet(gpioPortD, 9);
294                 // Format the string using sprintf
295                 print_dir="-B";
296                 sprintf(display, "%d%s", distance, print_dir);
297                 // Use sl_segment_lcd_write to display the formatted string
298                 sl_segment_lcd_write(display);
299             }
300             else if(dir=='R')
301             {
302                 GPIO_PinOutClear(gpioPortD, 7);
303                 GPIO_PinOutClear(gpioPortD, 8);
304                 GPIO_PinOutSet(gpioPortD, 6);
305                 GPIO_PinOutSet(gpioPortD, 9);
306                 // Format the string using sprintf
307                 print_dir="-R";
308                 sprintf(display, "%d%s", distance, print_dir);
309                 // Use sl_segment_lcd_write to display the formatted string
310                 sl_segment_lcd_write(display);
311             }
312             else if(dir=='I')
313             {
```



```
313     else if(dir=='L')
314     {
315         GPIO_PinOutClear(gpioPortD, 6);
316         GPIO_PinOutClear(gpioPortD, 9);
317         GPIO_PinOutSet(gpioPortD, 7);
318         GPIO_PinOutSet(gpioPortD, 8);
319         // Format the string using sprintf
320         print_dir="-L";
321         sprintf(display, "%d%s", distance, print_dir);
322         // Use sl_segment_lcd_write to display the formatted string
323         sl_segment_lcd_write(display);
324     }
325     else
326     {
327         print_dir="-N";
328         GPIO_PinOutClear(gpioPortD, 6);
329         GPIO_PinOutClear(gpioPortD, 7);
330         GPIO_PinOutClear(gpioPortD, 8);
331         GPIO_PinOutClear(gpioPortD, 9);
332         // Format the string using sprintf
333         sprintf(display, "%d%s", distance, print_dir);
334         // Use sl_segment_lcd_write to display the formatted string
335         sl_segment_lcd_write(display);
336     }
337     newDataReceived = false;
338 }
339 if(newDataReceived==false)
340 {
341     // Measure the distance and output it
342     distance = measure_distance();
343     if(distance<30 && dir=='F')
344     {
345         print_dir="-S";
346         GPIO_PinOutClear(gpioPortD, 6);
347         GPIO_PinOutClear(gpioPortD, 7);
348         GPIO_PinOutClear(gpioPortD, 8);
349         GPIO_PinOutClear(gpioPortD, 9);
350         // Format the string using sprintf
351         sprintf(display, "%d%s", distance, print_dir);
352         // Use sl_segment_lcd_write to display the formatted string
353         sl_segment_lcd_write(display);
354     }
355     else
356     {
357         if(strcmp(print_dir, "ERROR") == 0)//if print_dir=="ERROR"
358         {
359             print_dir="-N";
360         }
361         sprintf(display, "%d%s", distance, print_dir);
362         sl_segment_lcd_write(display);
363     }
364 }
365 }
366 }
367
```



הגדרה ראשונית של ה-Software components:

final_project OVERVIEW **SOFTWARE COMPONENTS** CONFIGURATION T

Filter components by Configurable ☐ Installed ☐ Installed by you ☒

▼ Platform

▼ Board

- ▶ Starter Kit

▼ Board Drivers

- ☒ Segment LCD

▼ Device

- ▼ EFM32PG28
 - ☒ EFM32PG28B310F1024IM68

▼ Driver

- ▼ Button
 - ▶ Simple Button
 - ☒ Generic Button API

▼ Peripheral

- ☒ EUSART
- ☒ TIMER

☒ GPIOINT

☒ TEMPDRV

☒ USTIMER

▼ Peripheral

- ▶ Init
 - ACMP
 - BURTC
- ☒ CHIP
- ☒ CMU
- ☒ CORE
- ☒ Common Headers
- DBG
- EMLIB Peripheral HAL
- ☒ EMU
- ☒ EUSART
- GPCRC
- ☒ GPIO
- I2C
- IADC
- KEYSCAN

GPIO

Description

General Purpose IO (GPIO) peripheral API

Quality

PRODUCTION

Dependencies

emlib_gpio requires 3 components

▶ Platform

Dependents

5 components require emlib_gpio

✖ Uninstall



חישוב ערך ה-TOP של שני הטיימרים:

עבור TIMER0:

אנו עובדים עם תדר [MHZ] 39, ואנו צריכים לממש אות ריבועי בעל תדר של 300[HZ].

החישוב יהיה כדלקמן:

$$TOP_0 = \frac{F_{sys}}{F_{sig}}$$

$$TOP_0 = \frac{39 \cdot 10^6}{300}$$

$$TOP_0 = 130,000(DECIMAL) = 1FBD0(HEX)$$

עבור TIMER1:

אנו עובדים עם תדר [MHZ] 39, יש לציין מכיוון שטיימר 1 הוא בעל 16 ביט לכן אם נחלק ללא חלוקת תדר נקבל ערך גדול מ-16 ביט כלומר גדול מ- 2^{16} , לכן נבצע לו חלוקת תדר ב-32 ואז ניתן נחלק בתדר הרצוי שהוא בעל תדר של 20[HZ].

החישוב יהיה כדלקמן:

$$TOP_1 = \frac{F_{sys}}{32 * F_{sig}}$$

$$TOP_1 = \frac{39 \cdot 10^6}{32 * 20} = 60,937$$

$$TOP_1 = 60,937(DECIMAL) = EE09(HEX)$$



חישוב ערך ה-Duty count:

TIMER 0

החישוב מתבצע בעזרת הנוסחה הבאה:

$$\text{duty count} = \frac{\text{top value} * \text{duty cycle}}{100} = \frac{130,000 * \text{duty cycle}}{100}$$

עבור duty Cycle = 60

$$\text{duty count} = \frac{130,000 * 60}{100} = 78,000(\text{DECIMAL}) = 130B0(\text{HEX})$$

עבור duty Cycle = 70

$$\text{duty count} = \frac{130,000 * 70}{100} = 91,000(\text{DECIMAL}) = 16378(\text{HEX})$$

עבור duty Cycle = 80

$$\text{duty count} = \frac{130,000 * 80}{100} = 104,000(\text{DECIMAL}) = 19640(\text{HEX})$$

עבור duty Cycle = 95

$$\text{duty count} = \frac{130,000 * 95}{100} = 123,500(\text{DECIMAL}) = 1E26C(\text{HEX})$$

TIMER 1

החישוב מתבצע בעזרת הנוסחה הבאה:

$$\text{duty count} = \text{top value} * \text{duty cycle} = 60,937 * \text{duty cycle}$$

לכן, $\text{duty cycle} = 0.0002\%$

$$\text{duty count} = 60,937 * 0.0002$$

$$\text{duty count} = 12(\text{DECIMAL}) = C(\text{HEX})$$

צילומי מסך רלוונטיים מהדיבגר:

אתחולים של ה-TIMER0,GPIO,TIMER1,EUSART1,LCD-

```

72 //Enable clocks for peripherals
73 CMU_ClockEnable(cmuClock_GPIO, true); // Enable clock for GPIO
74 CMU_ClockEnable(cmuClock_TIMER0, true); // Enable clock for TIMER0 (Motor PWM)
75 CMU_ClockEnable(cmuClock_TIMER1, true); // Enable clock for TIMER1 (Ultrasonic PWM)
76 CMU_ClockEnable(cmuClock_LCD, true); // Enable clock for LCD display
77 sl_segment_lcd_init(false); // Initialize segment LCD
78 CMU_ClockEnable(cmuClock_EUSART1, true); // Enable clock for EUSART1 (Bluetooth communication)
79

```

Register	Address	Value
CALCNT	0x50008058	0x00000000
CLKEN0	0x50008064	0x84DE0030
LDMA	[0]	0x0
LDMAxBAR	[1]	0x0
GPCRC	[3]	0x0
TIMER0	[4]	0x1
TIMER1	[5]	0x1
GPIO	[26]	0x1

Register	Address	Value
CALCCTRL	0x50008054	0x00000000
CALCNT	0x50008058	0x00000000
CLKEN0	0x50008064	0x84DE0030
CLKEN1	0x50008068	0x00815000
HOSTMAILBOX	[8]	0x0
SEMAILBOXHOST	[10]	0x0
LCD	[12]	0x1
EUSART1	[23]	0x1

עבור TIMER0:

```
156   topValue = (timerFreq / PWM_FREQ);
157   TIMER_TopSet(TIMER0, topValue);
```

TIMER0_NS: 0x50048000		
Register	Address	Value
TOP	[31:0]	0x1FBD0

כפי שניתן לראות, החישוב של ה-TOP בתיאוריה ובפועל שווים אחד לשני.

CNT	0x50048024	0x00016EFD
CNT	[31:0]	0x16EFD

הערך של ה CNT במהלך הרצת התוכנית.

```
119 // Initialize TIMER0 (Channel 0 and Channel 1)
120 TIMER_Init(TIMER0, &timerInit);
```

EN	0x50048030	0x00000001
EN	[0]	0x1
DISABLING	[1]	0x0

משמש להפעיל את הטיימר.

Duty count:

```
161   dutyCount = (topValue * INITIAL_DUTY_CYCLE) / 100;
162
163   TIMER_CompareSet(TIMER0, 0, dutyCount); // Set compare value for CC0
164   TIMER_CompareSet(TIMER0, 1, dutyCount); // Set compare value for CC1
```

עבור duty Cycle = 60:

CC0_OC	0x50048068	0x000130B0
OC	[31:0]	0x130B0

כפי שניתן לראות, החישוב של ה- duty count בתיאוריה ובפועל שווים אחד לשני.

עבור duty Cycle = 70:

CC0_CTRL	0x50048064	0x00000000
CC0_OC	0x50048068	0x00016378
OC	[31:0]	0x16378

כפי שניתן לראות, החישוב של ה- duty count בתיאוריה ובפועל שווים אחד לשני.

עבור duty Cycle = 80:

CC0_OC	0x50048068	0x00019640
OC	[31:0]	0x19640

כפי שניתן לראות, החישוב של ה- duty count בתיאוריה ובפועל שווים אחד לשני.









עבור duty Cycle = 95:

CC0_OC	0x50048068	0x0001E26C
OC	[31:0]	0x1E26C

כפי שניתן לראות, החישוב של ה- duty count בתיאוריה ובפועל שווים אחד לשני.

עבור TIMER1:

```
171     topValue_us = timerFreq_us / PWM_FREQ_us;
172     TIMER_TopSet(TIMER1, topValue_us);
```

 TIMER0_NS	Register	Address	Value
 TIMER1_NS	 IF	0x5004C014	0x00000011
 EUSART1_NS	 IEN	0x5004C018	0x00000000
 CMU_NS	 TOP	0x5004C01C	0x0000EE09
	 TOP	[15:0]	0xEE09

כפי שניתן לראות, החישוב של ה-TOP בתיאוריה ובפועל שווים אחד לשני.

▼ CNT	0x5004C024	0x00006AAB
⊞ CNT	[15:0]	0x6AAB

הערך של ה CNT במהלך הרצת התוכנית.

EN	0x5004C030	0x00000001
EN	[0]	0x1

משמש להפעיל את הטיימר.

Duty count:

```
173     // Calculate compare value for desired duty cycle
174     dutyCount_us = (uint32_t)(topValue_us * INITIAL_DUTY_CYCLE_us);
175     TIMER_CompareSet(TIMER1, 0, dutyCount_us);
176     //end for ultrasonic*****
```

> 0101	CC0_CTRL	0x5004C064	0x00000000
✓ 1010 0101	CC0_OC	0x5004C068	0x0000000C
001	OC	[15:0]	0xC
> 1010 0101	CC0_OCB	0x5004C070	0x00000000

כפי שניתן לראות, החישוב של ה- duty count בתיאוריה ובפועל שווים אחד לשני.



עבור EUSART1:

```
52 void EUSART1_RX_IRQHandler(void)
53 {
54     uint32_t flags = EUSART_IntGet(EUSART1); // Get interrupt flags
55
56     if (flags & EUSART_IF_RXFL) // Check if RX interrupt occurred
```

Register	Address	Value
IF	0x500A004C	0x00000002
TXC	[0]	0x0
TXFL	[1]	0x1
RXFL	[2]	0x0

לפני שהתקבל המידע מהבלוטוס

Register	Address	Value
IF	0x500A004C	0x00002007
TXC	[0]	0x1
TXFL	[1]	0x1
RXFL	[2]	0x1

ברגע שהתקבל מידע מהבלוטוס -

כשמתקבלת פסיקה זו, המעבד פונה למערך (NVIC - Nested Vectored Interrupt Controller), אשר מזהה את מקור הפסיקה ומפנה את השליטה לפונקציית (ISR - Interrupt Service Routine) המתאימה. במצב זה הפסיקה מבוצעת – הקוד בתוך הפונקציה `EUSART1_RX_IRQHandler()` רץ ומבצע את כל שלבי הטיפול בנתונים. לאחר שהפסיקה מסתיימת וכל דגלי הפסיקה מנוקים, המעבד חוזר להמשך ריצת התוכנית מהמקום שבו הופסקה.

תקלות ובעיות שנתקלנו בהן תוך כדי הפרויקט:

1. בעת שליחת נתונים למיקרו-בקר באמצעות Bluetooth, האותות לא נקלטו תמיד בניסיון הראשון, ונדרש היה לשלוח אותם מספר פעמים עד שהתקבלו.

פתרון: שילבנו מנגנון Interrupt לקריאת הנתונים המתקבלים מה-Bluetooth, כך שהמיקרו-בקר יוכל לזהות ולטפל באותות בזמן אמת, זה שיפר משמעותית את זמן התגובה של המערכת.

2. רק המנוע הימני הסתובב, לאחר ששתי המנועים עבדו בעבר, כלומר רק מנוע אחד עבד, בעוד שהשני נותר דומם.

פתרון: עברנו על כל החיבורים של הדוחף זרם, וכל החיבורי החומרה האפשריים, לבסוף גילינו שרגל הGND של המנוע השמאלי הייתה מנותקת מהמנוע, לכן ביצענו פעולת הלחמה בין רגל הGND למנוע, לאחר חידוש ההלחמה שני המנועים החלו לפעול כמצופה, ובכך נפתרה לנו הבעיה.



רגל הGND הייתה
מנותקת, זו
תמונה אחרי
ההלחמה

3. במהלך הפיתוח, ניסינו להשתמש בטיימר יחיד (Timer) כדי לשלוט על מספר רכיבים שונים, כאשר כל רכיב נדרש לפעול בתדר שונה.

פתרון: כדי לעקוף את המגבלה הזו, הוספנו טיימר נוסף (TIMER 1) והגדרנו אותו כך שיעבוד עם תדר שונה ועצמאי. בנוסף, התאמנו את חלוקת תדר השעון (Prescaler) כך שיתאפשר שימוש בערכים מדויקים בהתאם לרזולוציית 16 ביט, מה שאפשר לנו להפעיל כל רכיב בתדר המתאים לו.



קישור לסרטון:

<https://youtu.be/edGz5eK-1Wk>