To: John Hung, Victor Nelson

From: Jake Neal

Section: Wednesday, 3:00 p.m.

Date: 22 March 2019

Subject: Experimentation Report

Experimentation Report: PWM Waveform Generator

This report contains experimental proof for the proper function of the pulse-width modulated (PWM) waveform generator as well as the methods for how these experiments were conducted. Ultimately, it will be seen through experimental data that the design operated as described in the 3040/3050 lab manual [1].

The performance expected of the design is to generate PWM for various duty cycles. These duty cycles, as explained in the manual, are to range from 10% to 100% modulation. Further, it is expected that this range is achievable both in the original 1kHz frequency as well as other frequencies—100kHz was chosen as the alternate frequency.

The main data required to prove the functionality of the program designed for Lab 7 is the output from pin PA6 on the Discovery Board [2]. This pin is the output pin linked to timer 10, the component responsible for generating the PWM waveforms. From the designers’ point of view, the debugging software, Keil, presented important data for proper program functionality; specifically, the watch window served as a method to watch variables being passed between the user-defined functions.

As mentioned above, the output from pin PA6 contained the information necessary in proving proper PWM generation. To experimentally confirm the operation of the design, PA6 was connected to the oscilloscope that is built into the EE Board [3]. To interface with the oscilloscope, Waveform’s “scope” window was used. Through this software interface, the PWM output was directly observable as seen in Figures 1 and 2, and data such as duty cycle, frequency, and period could be accurately measured. To observe the values being held by the variables in the code, the watch window was opened and the variable “key” was displayed in that window. The variable “key” was chosen due to its use in the code [appendix 1]: when the duty cycles are being set in “EXTI1\_IRQHandler ()” starting on line 161, each if statement also assigns the percentage value to the variable “key.” Monitoring the value of key directly confirmed that the proper duty cycles were being set.

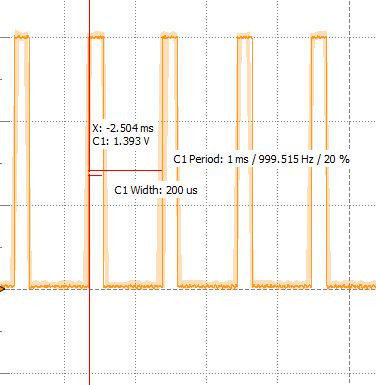


Figure 1.

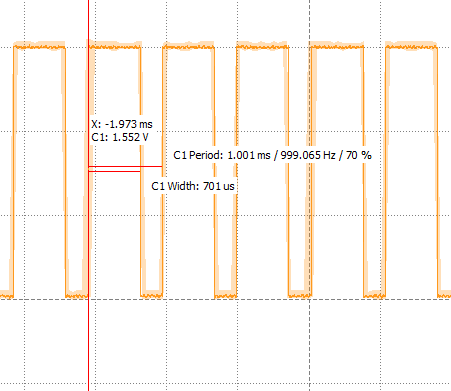


Figure 2.  
  
The figures above are two sample readings from the scope window. As seen in the on-screen measurements, Figure one shows a waveform with 20% modulation while Figure 2 contains a 70% modulated waveform. Both waveforms can also be seen to have a frequency of approximately 1kHz.

To further prove proper program functionality, the data recorded in the experiment has been compiled in Table 1. As seen in Figures 1 and 2, the frequency was consistently within 1Hz of the desired frequency, so its exact value was not recorded and is therefore omitted from Table 1.

Table 1. The duty cycles are listed for each of their corresponding buttons

|  |  |
| --- | --- |
| **Button** | **Duty Cycle Observed** |
| 0 | 0% |
| 1 | 10% |
| 2 | 20% |
| 3 | 30% |
| 4 | 40% |
| 5 | 50% |
| 6 | 60% |
| 7 | 70% |
| 8 | 80% |
| 9 | 90% |
| A | 100% |

The information found in the table may seem to be merely a copy of the requirements from the manual, but these were the actual measurements recorded from the oscilloscope. The realization of the exact duty cycle percentages serves to prove program functionality. Furthermore, the functionality can be seen to be precise rather than just approximately correct. When the experiment was conducted with an alternate frequency of 100Hz, the exact same results were observed: all the duty cycles were exactly equal to the desired values (precisely 20%, 70%, etc.). These results confirm that the program can create accurate PWM waveforms for any frequency rather than just 1kHz.

In conclusion, the oscilloscope was used to observe and measure the PWM waveform output. After taking these measurements, precise functionality for the program was confirmed for any desired duty cycle operating at any desired frequency.

**References**

[1] Nelson, V. P. (2015, September 8). LAB #7: PWM WAVEFORM GENERATION. Retrieved March 21, 2019, from [http://www.eng.auburn.edu/~nelson/courses/elec3040\_3050/LabWriteups/ELEC30x0 Lab7.pdf](http://www.eng.auburn.edu/~nelson/courses/elec3040_3050/LabWriteups/ELEC30x0%20Lab7.pdf)

[2] STMicroelectronics NV. (2015, March 9). STM32L100 Microcontroller Data Sheet. Retrieved March 21, 2019, from [http://www.eng.auburn.edu/~nelson/courses/elec3040\_3050/ARM\_Keil Documents/STM32L100\_DataSheet.pdf](http://www.eng.auburn.edu/~nelson/courses/elec3040_3050/ARM_Keil%20Documents/STM32L100_DataSheet.pdf)

[3] Digilent. (n.d.). Electronics Explorer Board - Reference Manual. Retrieved March 21, 2019, from <https://reference.digilentinc.com/reference/instrumentation/electronics-explorer/reference-manual?redirect=1>

**Appendix**

1. /\*====================================================\*/
2. /\* Jake Neal and Cameron Shea \*/
3. /\* ELEC 3040/3050 - Lab 7 \*/
4. /\*====================================================\*/
5. #include "STM32L1xx.h" /\* Microcontroller information \*/
6. /\* Define global variables \*/
7. int state=0; //current state of the LEDs in tenths
8. int key=0; //key that was pressed
9. unsigned char led1=0; //state of LED1
10. unsigned char led2=0; //state of LED2
11. unsigned char led3=0; //state of LED3
12. unsigned char led4=0; //state of LED4
13. /\*---------------------------------------------------\*/
14. /\* Initialize GPIO pins used in the program \*/
15. // PA1 input IRQ
16. // PB0-PB3 input/output keypad rows
17. // PB4-PB7 input/output keypad columns
18. // PC0-PC3 output counter LEDs
19. /\*---------------------------------------------------\*/
20. void PinSetup () {
21. /\* Configure PA1 as input for IRQ \*/
22. RCC->AHBENR |= 0x01; // Enable GPIOA clock (bit 0)
23. GPIOA->MODER &= ~(0x0000000C); // Clear PA1
24. GPIOA->MODER |= 0x00000000; // General purpose input mode
25. GPIOA->MODER &= ~(0x00003000); // Clear PA6
26. GPIOA->MODER |= 0x00002000; // Sets PA6 to AF mode
27. GPIOA->AFR[0] &= ~(0x0F000000); // Clear AFRL6
28. GPIOA->AFR[0] |= 0x03000000; // PA6 = AF3
29. /\* Configure PC0-PC3 as output pins to drive LEDs \*/
30. RCC->AHBENR |= 0x04; // Enable GPIOC clock (bit 2)
31. GPIOC->MODER &= ~(0x000000FF); // Clear PC0-PC3 mode bits
32. GPIOC->MODER |= (0x00000055); // General purpose output mode for PC0-PC7
33. RCC->AHBENR |= 0x02; // Enable GPIOB clock (bit 0)
34. GPIOB->MODER &= ~(0x0000FF00); // PB4-PB7 output keypad rows
35. GPIOB->MODER |= (0x00005500); // ^^^^
36. GPIOB->PUPDR &= ~(0x0000FFFF); //clear bits 0-15 for PB0-PB7 \*HERE I AM MAKING SURE THE AND GATE READS LOW\*
37. GPIOB->PUPDR |= 0x00000055; //set bits 0-7 to 01 for PB0-PB3 pull-up resistors, \*CHECK THIS STEP\*
39. GPIOB->BSRR = 0x0010 << 16; // send 0 to pin 4
40. GPIOB->BSRR = 0x0020 << 16; // send 0 to pin 5
41. GPIOB->BSRR = 0x0040 << 16; // send 0 to pin 6
42. GPIOB->BSRR = 0x0080 << 16; // send 0 to pin 7
43. //counter setup
44. RCC->CR |= RCC\_CR\_HSION; //Turn on high speed (16MHz)
45. while((RCC->CR & RCC\_CR\_HSIRDY)==0); //wait until HSI is ready
46. RCC->CFGR |= RCC\_CFGR\_SW\_HSI; //select HSI as clock
47. RCC->APB2ENR |= 0x00000008; //TIM10EN is enabled
48. TIM10->PSC = 0x13; //enable prescale register
49. TIM10->ARR = 0x1F3F; //enable auto reload register
50. TIM10->DIER |= 0x03; //enable interrupt from counter
51. TIM10->CCR1 = 0x00; //starts the PWM as always off
52. TIM10->CCMR1 |= 0x0060; //PWM mode 1, and output compare and select
53. TIM10->CCER |= 0x0001; //output will drive pin and is active high
54. TIM10->CNT; //enable counter
55. TIM10->SR &= ~0x01;
57. //EXTI SECTION
58. SYSCFG->EXTICR[0] &= 0xFF0F; //clears EXTI1 bit
59. SYSCFG->EXTICR[0] |= 0x0000; //set EXTI1 = 0 to select PA1
60. EXTI->FTSR |= 0x0002; //Bit0=1 to make EXTI1 falling-edge trig.
61. EXTI->IMR |= 0x0002; //Bit0=1 to enable EXTI1
62. EXTI->PR |= 0x0002; //Bit0=1 to clear EXTI1 pending status
63. //NVIC SECTION
64. NVIC\_EnableIRQ(EXTI1\_IRQn); //set bit n to enable IRQ7
65. NVIC\_ClearPendingIRQ(EXTI1\_IRQn); // clears pending status
66. NVIC\_EnableIRQ(7); //set bit n to enable TIM10 IRQ
67. NVIC\_ClearPendingIRQ(7); // clears pending status
69. //CPU SECTION
70. \_\_enable\_irq(); //enable interrupts
71. TIM10->CR1 |=0x01; //enable counting
72. }
73. /\*----------------------------------------------------------\*/
74. /\* TIM10\_IRQ26 Interrupt Function (signals the pressing of a keyboard button
75. /\*----------------------------------------------------------\*/
76. void TIM10\_IRQHandler ()
77. {
78. TIM10->SR &= ~0x01;
79. NVIC\_ClearPendingIRQ (TIM10\_IRQn); // clears pending status
80. }
81. /\*----------------------------------------------------------\*/
82. /\* EXTI1 Interrupt Function (signals the pressing of a keyboard button
83. /\*----------------------------------------------------------\*/
84. void EXTI1\_IRQHandler ()
85. {
86. int pb0=1; //reading from PB0
87. int pb1=1; //reading from PB1
88. int pb2=1; //reading from PB2
89. int pb3=1; //reading from PB3
90. int pb4=1; //reading from PB4
91. int pb5=1; //reading from PB5
92. int pb6=1; //reading from PB6
93. int pb7=1; //reading from PB7
94. int i,j,n;
95. // wait 1 ms
96. for (i=0; i<40; i++) //outer loop
97. {
98. for (j=0; j<18; j++)
99. { //inner loop
100. n = j; //dummy operation for single-step test
101. } //do nothing
102. }
104. //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//reading columns \*CHECK THIS STEP\*
106. GPIOB->MODER &= ~(0x000000FF); // PB0-PB3 input keypad columns
107. GPIOB->MODER |= (0x00000000); // ^^^^
109. GPIOB->MODER &= ~(0x0000FF00); // PB4-PB7 output keypad rows
110. GPIOB->MODER |= (0x00005500); // ^^^^

113. //Pull Up Pull Down Section \*CHECK THIS STEP\*
114. GPIOB->PUPDR &= ~0x0000FFFF; //clear bits 0-15 for PB0-PB7
115. GPIOB->PUPDR |= 0x00000055; //set bits 0-7 to 01 for PB0-PB3 pull-up resistors, \*CHECK THIS STEP\*

118. GPIOB->BSRR = 0x0010 << 16; // send 0 to pin 4
119. GPIOB->BSRR = 0x0020 << 16; // send 0 to pin 5
120. GPIOB->BSRR = 0x0040 << 16; // send 0 to pin 6
121. GPIOB->BSRR = 0x0080 << 16; // send 0 to pin 7
123. for (int x=0;x<4;x++); // just making sure it has time
125. pb0 = GPIOB->IDR & 0x0001; //reading PB0
126. pb1 = GPIOB->IDR & 0x0002; //reading PB1
127. pb2 = GPIOB->IDR & 0x0004; //reading PB2
128. pb3 = GPIOB->IDR & 0x0008; //reading PB3
129. //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//reading rows
131. GPIOB->MODER &= ~(0x0000FF00); // PB4-PB7 input keypad rows
132. GPIOB->MODER |= (0x00000000); // ^^^^
134. GPIOB->MODER &= ~(0x000000FF); // PB0-PB3 output keypad columns
135. GPIOB->MODER |= (0x00000055); // ^^^^

138. GPIOB->PUPDR &= ~0x0000FFFF; //clear bits 0-15 for PB0-PB7
139. GPIOB->PUPDR |= 0x00005500; //set bits 08-15 to 01 for PB4-PB7 pull-up resistors, \*CHECK THIS STEP\*

142. GPIOB->BSRR = 0x0001 << 16; // send 0 to pin 0
143. GPIOB->BSRR = 0x0002 << 16; // send 0 to pin 1
144. GPIOB->BSRR = 0x0004 << 16; // send 0 to pin 2
145. GPIOB->BSRR = 0x0008 << 16; // send 0 to pin 3
147. for (int x=0;x<4;x++); // just making sure it has time
149. pb4 = GPIOB->IDR & 0x0010; //reading PB4
150. pb5 = GPIOB->IDR & 0x0020; //reading PB5
151. pb6 = GPIOB->IDR & 0x0040; //reading PB6
152. pb7 = GPIOB->IDR & 0x0080; //reading PB7
154. //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//decide key
155. if((pb4==0) && (pb0==0)) // button 1
156. {
157. key=1;
158. TIM10->CCR1 = 0x320; //makes it 10% PWM (8000 in decimal)
159. }
160. else if((pb4==0) && (pb1==0)) // button 2
161. {
162. key=2;
163. TIM10->CCR1 = 0x640; //makes it 20% PWM (16000 in decimal)
164. }
165. else if((pb4==0) && (pb2==0)) // button 3
166. {
167. key=3;
168. TIM10->CCR1 = 0x960; //makes it 30% PWM (24000 in decimal)
169. }
170. else if((pb5==0) && (pb0==0)) // button 4
171. {
172. key=4;
173. TIM10->CCR1 = 0xC80; //makes it 40% PWM (32000 in decimal)
174. }
175. else if((pb5==0) && (pb1==0)) // button 5
176. {
177. key=5;
178. TIM10->CCR1 = 0xFA0; //makes it 50% PWM (4000 in decimal)
179. }
180. else if((pb5==0) && (pb2==0)) // button 6
181. {
182. key=6;
183. TIM10->CCR1 = 0x12C0; //makes it 60% PWM (4800 in decimal)
184. }
185. else if((pb6==0) && (pb0==0)) // button 7
186. {
187. key=7;
188. TIM10->CCR1 = 0x15E0; //makes it 70% PWM (5600 in decimal)
189. }
190. else if((pb6==0) && (pb1==0)) // button 8
191. {
192. key=8;
193. TIM10->CCR1 = 0x1900; //makes it 80% PWM (6400 in decimal)
194. }
195. else if((pb6==0) && (pb2==0)) // button 9
196. {
197. key=9;
198. TIM10->CCR1 = 0x1C20; //makes it 90% PWM (7200 in decimal)
199. }
200. else if((pb4==0) && (pb3==0)) // button A
201. {
202. key=10;
203. TIM10->CCR1 = 0x1F40; //makes it 100% PWM (8000 in decimal)
204. }
205. else if((pb5==0) && (pb3==0)) // button B
206. {
207. //key=11;
208. }
209. else if((pb6==0) && (pb3==0)) // button C
210. {
211. //key=12;
212. }
213. else if((pb7==0) && (pb3==0)) // button D
214. {
215. //key=13;
216. }
217. else if((pb7==0) && (pb0==0)) // button \* (treat like E)
218. {
219. //key=14;
220. }
221. else if((pb7==0) && (pb2==0)) // button # (treat like F)
222. {
223. //key=15;
224. }
225. else if((pb7==0) && (pb1==0)) // button 0
226. {
227. key=0;
228. TIM10->CCR1 = 0x000; //makes it 0% PWM
229. }
230. else{}
231. //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//display key on LEDs
232. // wait 1 ms
233. for (i=0; i<200; i++) //outer loop
234. {
235. for (j=0; j<18; j++)
236. { //inner loop
237. n = j; //dummy operation for single-step test
238. } //do nothing
239. }
241. GPIOB->MODER &= ~(0x000000FF); // PB0-PB3 input keypad columns
242. GPIOB->MODER |= (0x00000000); // ^^^^
244. GPIOB->MODER &= ~(0x0000FF00); // PB4-PB7 output keypad rows
245. GPIOB->MODER |= (0x00005500); // ^^^^
246. GPIOB->PUPDR &= ~0x0000FFFF; //clear bits 0-15 for PB0-PB7 \*HERE I AM MAKING SURE THE AND GATE READS LOW\*
247. GPIOB->PUPDR |= 0x00000055; //set bits 0-7 to 01 for PB0-PB3 pull-up resistors, \*CHECK THIS STEP\*
249. GPIOB->BSRR = 0x0010 << 16; // send 0 to pin 4
250. GPIOB->BSRR = 0x0020 << 16; // send 0 to pin 5
251. GPIOB->BSRR = 0x0040 << 16; // send 0 to pin 6
252. GPIOB->BSRR = 0x0080 << 16; // send 0 to pin 7 resistors, \*CHECK THIS STEP\*
254. NVIC\_ClearPendingIRQ (7); // clears pending status
255. EXTI->PR |= 0x0002; //Bit0=1 to clear EXTI1 pending status
256. }
257. /\*---------------------------------------------------------------\*/
258. /\* Count - counts up or down based on value passed into function \*/
259. /\*---------------------------------------------------------------\*/
260. void count (a)
261. {
262. //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//updating tenths LEDs
263. switch(key)
264. {
265. case 0:
266. led1=0;
267. led2=0;
268. led3=0;
269. led4=0;
270. break;
271. case 1:
272. led1=1;
273. led2=0;
274. led3=0;
275. led4=0;
276. break;
277. case 2:
278. led1=0;
279. led2=1;
280. led3=0;
281. led4=0;
282. break;
283. case 3:
284. led1=1;
285. led2=1;
286. led3=0;
287. led4=0;
288. break;
289. case 4:
290. led1=0;
291. led2=0;
292. led3=1;
293. led4=0;
294. break;
295. case 5:
296. led1=1;
297. led2=0;
298. led3=1;
299. led4=0;
300. break;
301. case 6:
302. led1=0;
303. led2=1;
304. led3=1;
305. led4=0;
306. break;
307. case 7:
308. led1=1;
309. led2=1;
310. led3=1;
311. led4=0;
312. break;
313. case 8:
314. led1=0;
315. led2=0;
316. led3=0;
317. led4=1;
318. break;
319. case 9:
320. led1=1;
321. led2=0;
322. led3=0;
323. led4=1;
324. break;
325. case 10:
326. led1=0;
327. led2=1;
328. led3=0;
329. led4=1;
330. break;
331. }
333. if (led1 == 0)
334. GPIOC->BSRR = 0x0001 << 16;
335. else
336. GPIOC->BSRR = 0x0001;
337. if (led2 == 0)
338. GPIOC->BSRR = 0x0002 << 16;
339. else
340. GPIOC->BSRR = 0x0002;
341. if (led3 == 0)
342. GPIOC->BSRR = 0x0004 << 16;
343. else
344. GPIOC->BSRR = 0x0004;
345. if (led4 == 0)
346. GPIOC->BSRR = 0x0008 << 16;
347. else
348. GPIOC->BSRR = 0x0008;
349. }
350. /\*------------------------------------------------\*/
351. /\* Main program \*/
352. /\*------------------------------------------------\*/
353. int main(void)
354. {
355. PinSetup(); //Configure GPIO pins
357. //the infinite loop will begin by counting up from zero once SW1
358. while(1)
359. {
360. count(0);
361. }
362. }