Peer Review #3

This review is for Lab 6: Pulse-width modulation using the programmable timer in C. The goal for this lab is to use what we have previously learned about timer interrupts and apply it to make a pulse width change depending on the value pressed by a keypad. This applies knowledge of normal GPIO interrupts, the using of a keypad, and an interrupt timer, all mixed into one lab to create this function. Used in this lab includes: A breadboard (with studio), a TM32 NUCLEO-L432KC Board, wires, and a .c file. This lab is very similar to last weeks Lab 5, where we first learned to use the timer to count in different increments, however, this time we use that timer not to count, but to instead change the time variation between pulses that can be seen on an oscilloscope. Duty cycle = T1/T. We then use the scope to view and verify our output for each keypad input. We can also view the keypad value pressed on the DIO pins in Static IO of Waveforms. With these two things we can verify the results of our code.

The program written obtains the following steps:

1. Establish variables used, pins set up, and interrupts setup (including normal interrupts and timer)
2. Enter an infinite loop where we do nothing
3. If an interrupt is pressed, we find the value pressed and change the duty cycle output to the corresponding keypad value pressed.
4. Then go back to the loop and wait.

My code for this worked very well and did not have too many problems. The connections I made matched the pinout that I drew for Lab 5, and added the PA0 output pin for testing the output duty cycle on the scope. The code for this lab is shown on the next page.

After setting up the lab and testing it out I had a couple small problems where I had forgot to change TIM7 to TIM2 and this caused a lot of errors in the code. After going through and changing all those, I could not get the ARR to upload right, because I had OR'd the wrong bits into the mode register. However, after getting these figured out the rest of the lab went simple. I learned to also view the input of the keypad after the AND gate to view if the keypad is sending the signal low or not. The scope was easily set up by connecting to any ground on the board, and then probing the output pin set PA0. Screenshots for the Duty cycle of 30% and 50% are shown in the Figures below.

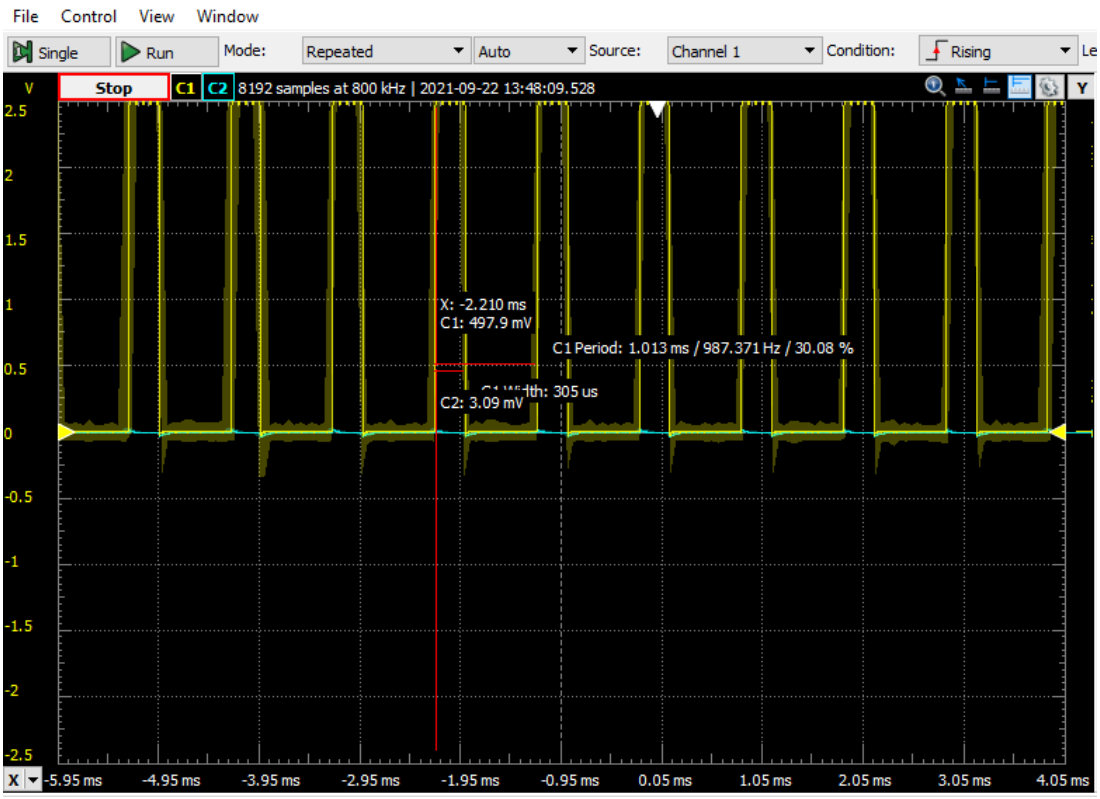

Figure 1

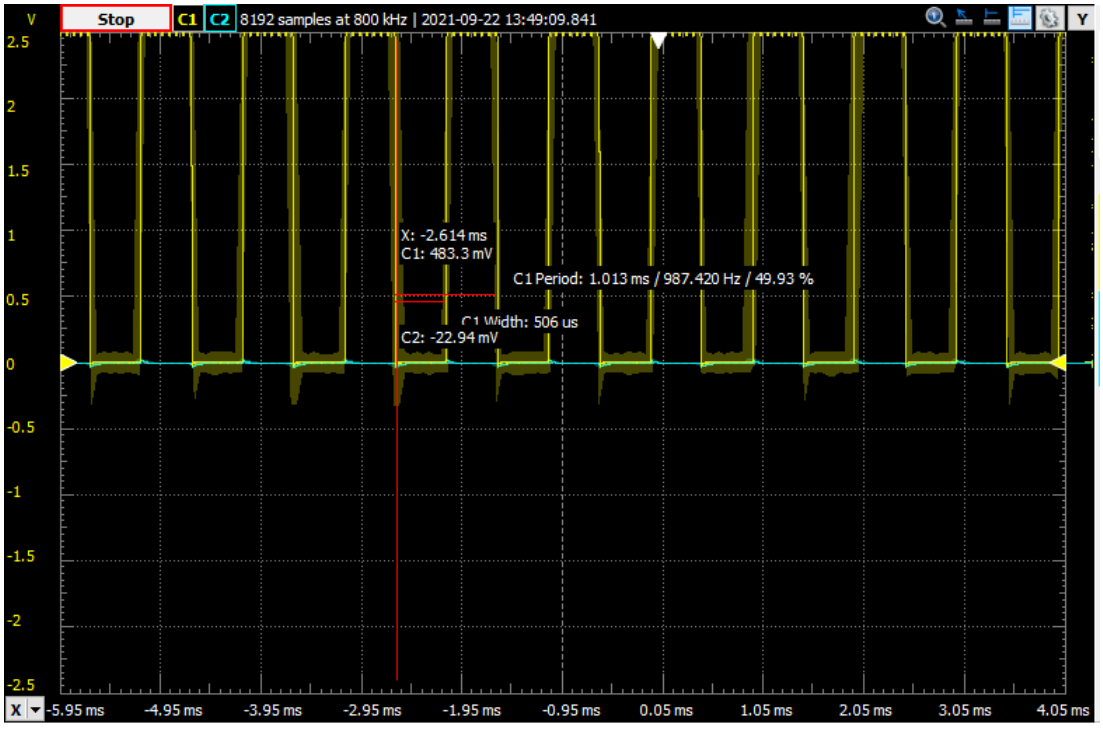

Figure 2

```
1
/*=====================================================*/
/* ELEC 3040 - Lab 6
/*=====================================================*/


#include "stm32l4xx.h" /* microcontroller information */


/* Define global variables */
static unsigned int counter;        //value of count (0-
9)
static unsigned int button;            //value of button
press
unsigned int col;                //what column has been
pressed
unsigned int row;                //what row has been
pressed
static unsigned int colNum;            //what column #
has been pressed
static unsigned int rowNum;            //what row # has
been pressed
unsigned int ccrNum;            //value for duty cycle
static unsigned int go;                    //handler
variable
unsigned int i,j,n,k;                    //delay
variables
static unsigned int keypad_map [4][4] = {        //keypad
matrix, no press = 0xFF
   {0x01,0x02,0x03,0x0A},    //0,0;1st row

```

```
 37 □□□   {0x04,0x05,0x06,0x0B},    //1,0;2nd row
 38 □□□
 39 □□□   {0x07,0x08,0x09,0x0C},    //2,0;3rd row
 40 □□□
 41 □□□   {0x0E,0x00,0x0F,0x0D}     //3,0;4th row
 42 □□□
 43 □□□};//0,0; 0,1; 0,2; 0,3;
 44 □□□
 45 □□□static unsigned int ccr_value [11] = {        //CCRy
values according to button press
 46 □□□
 47 □□□   0, 399, 799, 1199, 1599, 1999,
 48 □□□
 49 □□□   2399, 2799, 3199, 3599, 4001
 50 □□□
 51 □□□};
 52 □□□
 53 □□□
 54 □□□
 55 □□□/*-----------------------------------------------------*/
 56 □□□
 57 □□□/* initialize clocks used in the program */
 58 □□□
 59 □□□/* initialize GPIOB pins used in the program */
 60 □□□
 61 □□□/* PB[0] = interrupt trigger, output of AND gate (row
signals) */
 62 □□□
 63 □□□/* PB[6:3] = displayed value, counter or button */
 64 □□□
 65 □□□/*-----------------------------------------------------*/
 66 □□□
 67 □□□□static void Setup() {
 68 □□□
 69 □□□
 70 □□□
 71 □□□   /* enable clocks */
 72 □□□
 73 □□□□□□RCC->AHB2ENR |= 0x03;        //enable GPIOA clock
(bit 0) and GPIOB clock (bit 1)
 74 □□□
 75 □□□
 76 □□□
 77 □□□   /* configure GPIO pins */
 78 □□□
```

```
 79 □□□□□□GPIOA->MODER &= (0xFFFFFFF0);
//PA0 = 00 and PA1 = 00
 80 □□□
 81 □□□□□□GPIOA->MODER |=    (0x00000006);
//PA0 = 10 and PA1 = 01
 82 □□□
 83 □□□
 84 □□□
 85 □□□□□□GPIOB->MODER &= (0xFFFFC03C);
//inputs// display and AND, PB[6:3,0] = 00
 86 □□□
 87 □□□□□□GPIOB->MODER |= (0x00001540);
//outputs// display, PB[6:3] = 01
 88 □□□
 89 □□□
 90 □□□
 91 □□□}
 92 □□□
 93 □□□
 94 □□□
 95 □□□/*-----------------------------------------------------*/
 96 □□□
 97 □□□/* initialize GPIO pins used in the program */
 98 □□□
 99 □□□/*-----------------------------------------------------*/
100 □□□
101 □□□□static void PinSetup1() {
102 □□□
103 □□□
104 □□□
105 □□□□□□Setup();
106 □□□
107 □□□   /* configure GPIOA pins */
108 □□□
109 □□□□□□GPIOA->MODER &= (0xFF00F00F);        //inputs//
column and row, PA[11:8,5:2] = 00
110 □□□
111 □□□□□□GPIOA->MODER |= (0x00550000);        //outputs//
column, PA[11:8] = 01
112 □□□
113 □□□
114 □□□
115 □□□   /* configure push-pull pins */
116 □□□
117 □□□□□□GPIOA->PUPDR &= (0xFFFFF00F);        //pull-reset//
row, PA[5:2] = 00
```

```
118 □□□
119 □□□□□□GPIOA->PUPDR |= (0x00000550);        //pull-up// row,
PA[5:2] = 01
120 □□□
121 □□□
122 □□□
123 □□□
124 □□□
125 □□□}
126 □□□
127 □□□
128 □□□
129 □□□/*-----------------------------------------------------*/
130 □□□
131 □□□/* initialize GPIO pins used in the program */
132 □□□
133 □□□/*-----------------------------------------------------*/
134 □□□
135 □□□□static void PinSetup2() {
136 □□□
137 □□□
138 □□□
139 □□□□□□Setup();
140 □□□
141 □□□   /* configure GPIOA pins */
142 □□□
143 □□□□□□GPIOA->MODER &= (0xFF00F00F);        //inputs//
column and row, PA[11:8,5:2] = 00
144 □□□
145 □□□□□□GPIOA->MODER |= (0x0550);               //outputs//
row, PA[5:2] = 01
146 □□□
147 □□□
148 □□□
149 □□□   /* configure push-pull pins */
150 □□□
151 □□□□□□GPIOA->PUPDR &= (0xFF00FFFF);        //pull-reset//
row, PA[11:8] = 00
152 □□□
153 □□□□□□GPIOA->PUPDR |= (0x00550000);        //pull-up// row,
PA[11:8] = 01
154 □□□
155 □□□
156 □□□
157 □□□}
```

```
158 □□□
159 □□□
160 □□□
161 □□□/*----------------------------------------------------*/
162 □□□
163 □□□/* enable PWM used in the program*/
164 □□□
165 □□□/*----------------------------------------------------*/
166 □□□
167 □□□□static void PulseSetup() {
168 □□□
169 □□□
170 □□□
171 □□□   /* enable clock */
172 □□□
173 □□□□□□RCC->AHB2ENR |= 0x01;             // Enable GPIOA
clock (bit 0)
174 □□□
175 □□□
176 □□□
177 □□□   /* configure pins */
178 □□□
179 □□□□□□GPIOA->MODER &= 0xFFFFFFFC; // PA0 = 00, clear
180 □□□
181 □□□□□□GPIOA->MODER |= 0x0002;         // PA0 = 01,
alternative function mode
182 □□□
183 □□□
184 □□□
185 □□□   /* select desired AF (timer) */
186 □□□
187 □□□□□□GPIOA->AFR[0] &= (0xFFFFFFF0);                //mask
bit[3:0]=00
188 □□□
189 □□□□□□GPIOA->AFR[0] |= (0x00000001);
//(0x0002);//configure bit[3:0]=0010, AF1 selected
190 □□□
191 □□□
192 □□□
193 □□□   /* configure timer */
194 □□□
195 □□□□□□RCC->APB1ENR1 |= RCC_APB1ENR1_TIM2EN;
//(0x01);//enable timer module
196 □□□
197 □□□□□□TIM2->CR1 |= 0x01;
//enable timer counter
```

```
198 □□□
199 □□□□□□TIM2->CCMR1 &= (0xFFFFFF8C);        //mask channel
one (bit[6:4]=00), output mode
200 □□□
201 □□□□□□TIM2->CCMR1 |= (0x00000060);
//configure output mode for PWM mode 1
202 □□□
203 □□□□□□TIM2->CCER &= (0xFFFC);
//bit[1:0]=00, clear timer channel 1 output
204 □□□
205 □□□□□□TIM2->CCER |= (0x0001);
//bit[1:0]=01, enable timer channel 1 output (active high)
206 □□□
207 □□□
208 □□□
209 □□□   /* configure pulse */
210 □□□
211 □□□□□□TIM2->PSC = 0;
212 □□□
213 □□□□□□TIM2->ARR = 4000;
214 □□□
215 □□□□□□TIM2->CCR1 = 0;
216 □□□
217 □□□□□□NVIC_EnableIRQ(EXTI0_IRQn);    /* Enable IRQ */
218 □□□
219 □□□}
220 □□□
221 □□□
222 □□□
223 □□□/*-----------------------------------------------------*/
224 □□□
225 □□□/* initialize interrupts used in the program */
226 □□□
227 □□□/* EXTI1 = external interrupt one */
228 □□□
229 □□□/* EXTI2 = external interrupt two */
230 □□□
231 □□□/*-----------------------------------------------------*/
232 □□□
233 □□□□static void InterruptSetup() { //maybe void in ()
234 □□□
235 □□□
236 □□□
237 □□□   /* enable clocks */
238 □□□
```

```
239 □□□□□□RCC->APB2ENR |= 0x01;        //enable interrupt clock
SYSCFG
240 □□□
241 □□□
242 □□□
243 □□□   /* configure port PA0 as input source of EXTI0 */
244 □□□
245 □□□□□□SYSCFG->EXTICR[0] &= 0xFFF0;            //clear EXTI1
bit in config reg ~(0xF)
246 □□□
247 □□□□□□SYSCFG->EXTICR[0] |= 0x0001;          //PB
configuration in EXTI0
248 □□□
249 □□□
250 □□□
251 □□□
252 □□□
253 □□□ /* configure and enable EXTI0 as falling-edge triggered
*/
254 □□□
255 □□□□□□EXTI->FTSR1 |= 0x0001;
//falling edge trigger enabled
256 □□□
257 □□□□□□EXTI->IMR1 |= 0x0001;
//enable (unmask) EXTI0
258 □□□
259 □□□□□□EXTI->PR1 |= 0x0001;
//clear EXTI0 pending bit for line 1
260 □□□
261 □□□□□□NVIC_ClearPendingIRQ(EXTI0_IRQn);
////////* Clear NVIC pending bit */
262 □□□
263 □□□□□□NVIC_EnableIRQ(EXTI0_IRQn);
//enable IRQ with EXTI line 0 interrupt
264 □□□
265 □□□
266 □□□
267 □□□
268 □□□
269 □□□}
270 □□□
271 □□□/*-------------------------------------------------------
---*/
272 □□□
273 □□□/* debounce delay function - do nothing for about 0.001
second */
```

```
274 □□□
275 □□□/*-------------------------------------------------------
---*/
276 □□□
277 □□□□static void debounce() {                                //
278 □□□
279 □□□□□□for (i=0; i<15; i++) {                 //outer loop
280 □□□□□□
281 □□□□□□□□□for (j=0; j<40; j++) {             //inner loop
282 □□□□□□□□□
283 □□□□□□□□□□□□n = j;
//dummy operation for single-step test
284 □□□□□□□□□
285 □□□□□□□□□}
//do nothing
286 □□□□□□
287 □□□□□□}
288 □□□
289 □□□}
290 □□□
291 □□□
292 □□□
293 □□□/*-------------------------------------------------------
---*/
294 □□□
295 □□□/* delay function - do nothing for about 1 second */
296 □□□
297 □□□/*-------------------------------------------------------
---*/
298 □□□
299 □□□□static void delay() {
//
300 □□□
301 □□□□□□for (i=0; i<15; i++) {                 //outer loop
302 □□□□□□
303 □□□□□□□□□for (j=0; j<10000; j++) {        //inner loop
*4000*
304 □□□□□□□□□
305 □□□□□□□□□□□□n = j;
//dummy operation for single-step test
306 □□□□□□□□□
307 □□□□□□□□□}
//do nothing
308 □□□□□□
309 □□□□□□}
310 □□□
```

```
311 □□□}
312 □□□
313 □□□
314 □□□
315 □□□/*-------------------------------------------------------
---*/
316 □□□
317 □□□/* keypad function - find which button has been pressed
*/
318 □□□
319 □□□/*-------------------------------------------------------
---*/
320 □□□
321 □□□□static void keypad() {
322 □□□
323 □□□
324 □□□
325 □□□□□□button = 0;
326 □□□
327 □□□□□□rowNum = 0;
328 □□□
329 □□□□□□colNum = 0;
330 □□□
331 □□□□□□ccrNum = 0;
332 □□□
333 □□□□□□col = 0;                                  //initialize
col
334 □□□
335 □□□□□□row = 0;                                  //initialize
row
336 □□□
337 □□□
338 □□□
339 □□□   /* clear unwanted values */
340 □□□
341 □□□□□□GPIOB->ODR &= 0xFF87;                       //mask
PB[6:3] to 0
342 □□□
343 □□□   /* columns output 0 and find which row is 0*/
344 □□□
345 □□□   //PinSetup1();
346 □□□
347 □□□□□□row=0;
348 □□□
349 □□□□□□GPIOA->ODR &= (0xF0FF);                   //set column
to output 0, PA[11:8] = 0
```

```
350 □□□
351 □□□□□□for(k=0; k<4; k++);                          //delay
for values to load
352 □□□
353 □□□□□□row = (~GPIOA->IDR & 0x003C);    //get row inputs,
PA[5:2] = 0***check~
354 □□□
355 □□□□□□row = row >> 2;
//shift right by 2
356 □□□
357 □□□□□□do {
358 □□□□□□
359 □□□□□□□□□row = row << 1;
//shift left by 1 to find row count
360 □□□□□□
361 □□□□□□□□□rowNum++;
//add to row count
362 □□□□□□
363 □□□□□□} while(row < 0x10) ;                    //can only
shift four times
364 □□□
365 □□□
366 □□□
367 □□□   /* rows output 0 and find which column is 0 */
368 □□□
369 □□□□□□PinSetup2();
370 □□□
371 □□□□□□debounce();
372 □□□
373 □□□□□□col=0;
374 □□□
375 □□□□□□GPIOA->ODR &= (0xFFC3);                //set row to
output 0, PA[5:2] = 0
376 □□□
377 □□□□□□for(k=0; k<4; k++);                          //delay
for values to load
378 □□□
379 □□□□□□col = (~GPIOA->IDR & 0xF00);    //get column inputs,
PA[11:8] = 0
380 □□□
381 □□□□□□col = col >> 8;
//shift right by 8
382 □□□
383 □□□□□□do {
384 □□□□□□
385 □□□□□□□□□col = col << 1;
```

```
//shift left by 1 to find column count
386 □□□□□□
387 □□□□□□□□□colNum++;
//add to column count
388 □□□□□□
389 □□□□□□} while(col < 0x10) ;                    //can only
shift four times
390 □□□
391 □□□
392 □□□
393 □□□□□□button = keypad_map[--rowNum][--colNum];    //test
and see if works****
394 □□□
395 □□□
396 □□□
397 □□□□□□ccrNum = ccr_value[button];
398 □□□
399 □□□□□□TIM2->CCR1 = ccrNum;
400 □□□
401 □□□
402 □□□
403 □□□□□□button = button << 3;
404 □□□
405 □□□□□□GPIOB->ODR &= 0xFF87;                    //mask
PB[6:3] to 0
406 □□□
407 □□□□□□GPIOB->ODR |= button;        //output button value,
PB[6:3]
408 □□□
409 □□□
410 □□□
411 □□□
412 □□□
413 □□□□□□ccrNum = 0;
414 □□□
415 □□□□□□button = 0;
416 □□□
417 □□□□□□delay ();                                 //1 sec
delay
418 □□□
419 □□□□□□delay ();                                 //1 sec
delay
420 □□□
421 □□□}
422 □□□
423 □□□
```

```
424 □□□
425 □□□/*-------------------------------------------------------
---*/
426 □□□
427 □□□/* interrupt handler EXTI0 - keypad has been pressed */
428 □□□
429 □□□/*-------------------------------------------------------
---*/
430 □□□
431 □□□□void EXTI0_IRQHandler() { //maybe put void in ()
432 □□□
433 □□□
434 □□□
435 □□□
436 □□□
437 □□□□□□debounce();
438 □□□
439 □□□□□□go=~go;
440 □□□
441 □□□□□□keypad();                          //keypad logic
442 □□□
443 □□□□□□PinSetup1();
444 □□□
445 □□□□□□debounce();
446 □□□
447 □□□
448 □□□
449 □□□□□□EXTI->PR1 |= 0x0001;                //clear EXTI0
pending bit*
450 □□□
451 □□□□□□NVIC_ClearPendingIRQ(EXTI0_IRQn);        //clear NVIC
pending bit with EXTI line 1 interrupt
452 □□□
453 □□□□□□__enable_irq();
//enable interupts* Maybe this has to go before above line
454 □□□
455 □□□}
456 □□□
457 □□□
458 □□□
459 □□□/*-----------------------------------------------*/
460 □□□
461 □□□/* main program */
462 □□□
463 □□□/*-----------------------------------------------*/
```

```
464 □□□
465 □□□□int main(void) {
466 □□□
467 □□□
468 □□□
469 □□□□□□Setup();                                    //configure
clocks and GPIOB pins
470 □□□
471 □□□□□□PinSetup1();
472 □□□
473 □□□□□□InterruptSetup();                 //configure
interrupts
474 □□□
475 □□□□□□PulseSetup();
476 □□□
477 □□□□□□go = 1;
//initialize go
478 □□□
479 □□□
480 □□□
481 □□□   /* Endless loop */
482 □□□
483 □□□□□□while(1){                              //endless loop
484 □□□□□□
485 □□□□□□□□□delay();                               //delay for
1 seconds
486 □□□□□□
487 □□□□□□□□□if(go != 0x01){                    //see if
button has been pressed
488 □□□□□□□□□
489 □□□□□□□□□□□□go=~go;
490 □□□□□□□□□
491 □□□□□□□□□}
492 □□□□□□
493 □□□□□□} /* repeat forever */
494 □□□
495 □□□}
496 □□□
```