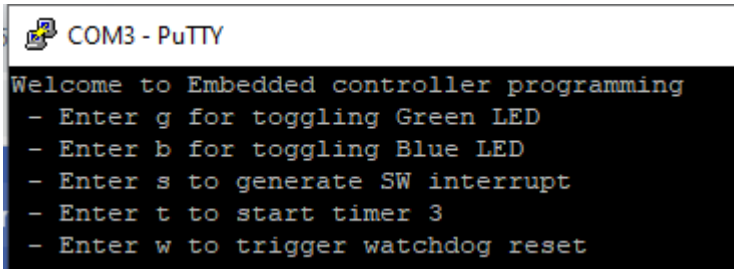
Nathan Bunnell
Embedded Controller Programming for Real-Time Systems
ECE-40097
Student ID: U08895857

Date: 8/30/2021

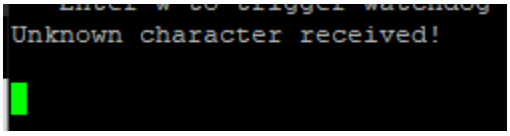
Final Assignment

See below for the terminal output of the methods defined for this assignment. Please see attached files for full comments within function prototypes.

Full prompt developed with the HAL_UART_Transmit_IT() call.



Unknown character input:



Create a software interrupt and use one of the non-used IRQ.

Using the FMC IRQ, the below handler was defined in stm32l4xx_it.c.

```
283 /**
284   * @brief This function handles a user-defined SW interrupt.
285   */
286 void FMC_IRQHandler(void)
287 {
288     FMC_IRQ_CpltCallback();
289 }
290
```

Nathan Bunnell
Embedded Controller Programming for Real-Time Systems
ECE-40097
Student ID: U08895857

Date: 8/30/2021

From there, the below init and callback functions were also define in main.c, along with the accompanying logic in the main() input-handler switch. FMC_IRQn_Init() was deployed external to MX_GPIO_Init() to prevent being overwritten any time changes were made in the Device Configuration tool.

```
147
148 // Added a dedicated init function here as MX_GPIO_Init() would
149 //  overwrite any added code after device cfg changes
150 void FMC_IRQn_Init(void)
151 {
152     // Enable FMC interrupt
153     uint32_t IRQn = FMC_IRQn;
154     uint32_t wordOffset = (IRQn >> 5);          // IRQn / 32
155     uint32_t bitOffset = (IRQn & 0x1f);         // IRQn mod 32
156     NVIC->ISER[wordOffset] = (1 << bitOffset);  // Enable INT
157 }
158
159 // Callback function for the FMC interrupt
160 //  Toggle an LED to give indication of int and
161 //  set flag to be used in main routine
162 void FMC_IRQ_CpltCallback(void)
163 {
164     HAL_GPIO_TogglePin(LED2_GPIO_Port, LED2_Pin);
165     swInterruptComplete = 1;
166 }
167
```

```
339
340         case ('s'):
341         {
342             logMsg(&huart1, "s");
343
344             // Enable SW interrupt at target IRQ, FMC or 48
345             NVIC->STIR = FMC_IRQn;
346
347             // Implement callback under FMC_IRQ_CpltCallback()
348             while (!swInterruptComplete)
349             {
350                 ;   // Loiter until flag is set
351             }
352
353             swInterruptComplete = 0;    // Reset flag
354
355             char buffer[50] = {0};      // Output buffer
356             snprintf(buffer, sizeof(buffer), "SW Interrupt detected");
357             logMsg(&huart1, buffer);
358
359             break;
360         }
```

Entering option ‘s’ presented the below output on the console:

Nathan Bunnell
Embedded Controller Programming for Real-Time Systems
ECE-40097
Student ID: U08895857

Date: 8/30/2021

```
 - Enter w to trigger watchdog reset
s
SW Interrupt detected
```

Create a method myDelay1() using timer2, which will take input in millisecond.

Function definition for myDelay1() is below. Configuration numbers are noted in line comments.

```
168 // NOTE: APB1 Timer clocks prescaled down to 10MHz in Clock Config tool
169 void myDelay1(uint32_t mSec)
170 {
171     if (mSec == 0)  // Dummy check that we have a real value
172         return;
173 
174     TIM2->CR1 &= ~TIM_CR1_CEN;  // Disable for config
175     TIM2->SR = 0;               // Clear status reg
176     TIM2->CNT = 0;              // Clear accrual
177     TIM2->PSC = 9999;           // PSC = (10MHz / 1KHz) - 1
178     TIM2->ARR = mSec - 1;       // Delay - 1
179     TIM2->CR1 |= TIM_CR1_CEN;   // Re-enable timer
180 
181     while ((TIM2->SR & TIM_SR_UIF) == 0);   // Loop until the UIF flag is set in the SR
182 }
```

Accompanying logic in the main() input-handler switch:

```
322         case ('b'):
323         {
324             // Print received char, print message indicating delay, toggle blue LED on and off with myDelay1()
325             logMsg(&huart1, "b");
326 
327             char buffer[50] = {0};      // Output buffer
328             snprintf(buffer, sizeof(buffer), "Toggling blue LED 3x on every 1000ms");
329             logMsg(&huart1, buffer);
330 
331             HAL_GPIO_TogglePin(LED3_WIFI__LED4_BLE_GPIO_Port, LED3_WIFI__LED4_BLE_Pin);
332             myDelay1(1000);
333             HAL_GPIO_TogglePin(LED3_WIFI__LED4_BLE_GPIO_Port, LED3_WIFI__LED4_BLE_Pin);
334             myDelay1(1000);
335             HAL_GPIO_TogglePin(LED3_WIFI__LED4_BLE_GPIO_Port, LED3_WIFI__LED4_BLE_Pin);
336 
337             break;
338         }
```

Console output from option ‘b’:

```
b
Toggling blue LED 3x on every 1000ms
```

Nathan Bunnell
Embedded Controller Programming for Real-Time Systems
ECE-40097
Student ID: U08895857

Date: 8/30/2021

Create a method myDelay2() using SysTick, which will take input in millisecond.

```
184 // Simple loop implementation to delay for a give number of mSec
185 void myDelay2(uint32_t mSec)
186 {
187     SysTick->LOAD = 80000;  // LOAD * (1/80MHz) = target of 1 mSec
188     SysTick->VAL = 0;       // Clear value
189     SysTick->CTRL |= SysTick_CTRL_TICKINT_Msk;  // Set clock source to internal
190     SysTick->CTRL |= SysTick_CTRL_ENABLE_Msk; // Enable SysTick
191
192
193     while (mSec > 0)
194     {
195         while((SysTick->CTRL & SysTick_CTRL_COUNTFLAG_Msk) == 0)
196             {
197                 ;   // Loiter while the CountFlag in bit 16 is not set
198             }
199         mSec--;
200     }
201 }
```

Accompanying logic in the main() input-handler switch:

```
304         case ('g'):
305         {
306             // Print received char, print message indicating delay, toggle green LED on and off with myDelay2()
307             logMsg(&huart1, "g");
308
309             char buffer[50] = {0};              // Output buffer
310             snprintf(buffer, sizeof(buffer), "Toggling green LED 3x on every 1000ms");
311             logMsg(&huart1, buffer);
312
313             HAL_GPIO_TogglePin(LED2_GPIO_Port, LED2_Pin);
314             myDelay2(1000);
315             HAL_GPIO_TogglePin(LED2_GPIO_Port, LED2_Pin);
316             myDelay2(1000);
317             HAL_GPIO_TogglePin(LED2_GPIO_Port, LED2_Pin);
318
319             break;
320         }
321
```

Console output from option ‘g’:

```
g
Toggling green LED 3x on every 1000ms
```

Nathan Bunnell
Embedded Controller Programming for Real-Time Systems
ECE-40097
Student ID: U08895857

Date: 8/30/2021

Use Timer 3 to count events.

Timer 3 config below, followed by callback definition:

```
633 static void MX_TIM3_Init(void)
634 {
635
636   /* USER CODE BEGIN TIM3_Init 0 */
637
638   /* USER CODE END TIM3_Init 0 */
639
640   TIM_ClockConfigTypeDef sClockSourceConfig = {0};
641   TIM_MasterConfigTypeDef sMasterConfig = {0};
642
643   /* USER CODE BEGIN TIM3_Init 1 */
644
645   /* USER CODE END TIM3_Init 1 */
646   htim3.Instance = TIM3;
647   htim3.Init.Prescaler = 9999;
648   htim3.Init.CounterMode = TIM_COUNTERMODE_UP;
649   htim3.Init.Period = 999;
650   htim3.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
651   htim3.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
652   if (HAL_TIM_Base_Init(&htim3) != HAL_OK)
653   {
654     Error_Handler();
655   }
656   sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
657   if (HAL_TIM_ConfigClockSource(&htim3, &sClockSourceConfig) != HAL_OK)
658   {
659     Error_Handler();
660   }
661   sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
662   sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
663   if (HAL_TIMEx_MasterConfigSynchronization(&htim3, &sMasterConfig) != HAL_OK)
664   {
665     Error_Handler();
666   }
667   /* USER CODE BEGIN TIM3_Init 2 */
668
669   /* USER CODE END TIM3_Init 2 */
670
671 }
672
```

```
203 void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
204 {
205     if (htim->Instance == TIM3)
206     {
207         tim3Accrual++;
208         tim3InterruptComplete = 1;
209     }
210 }
211
```

Nathan Bunnell
Embedded Controller Programming for Real-Time Systems
ECE-40097
Student ID: U08895857

Date: 8/30/2021

Nathan Bunnell
Embedded Controller Programming for Real-Time Systems
ECE-40097
Student ID: U08895857

Date: 8/30/2021

Pairing the above callback with the below logic in the main() input-handler switch resulted in the results in the console:

```
362         case ('t'):
363         {
364             logMsg(&huart1, "t");
365 
366             // Implement event counter using timer 3
367             tim3Accrual = 0;    // Reset accrual
368 
369             while (tim3Accrual < 10)
370             {
371                 tim3InterruptComplete = 0;
372 
373                 while (!tim3InterruptComplete)
374                 {
375                     ;   // Loiter until target count is reached
376                 }
377             }
378 
379             char buffer[50] = {0};      // Output buffer
380             snprintf(buffer, sizeof(buffer), "Total counted timer3 events: %d", tim3Accrual);
381             logMsg(&huart1, buffer);
382             break;
383         }
```

```
t
Total counted timer3 events: 10
```

Nathan Bunnell
Embedded Controller Programming for Real-Time Systems
ECE-40097
Student ID: U08895857

Date: 8/30/2021

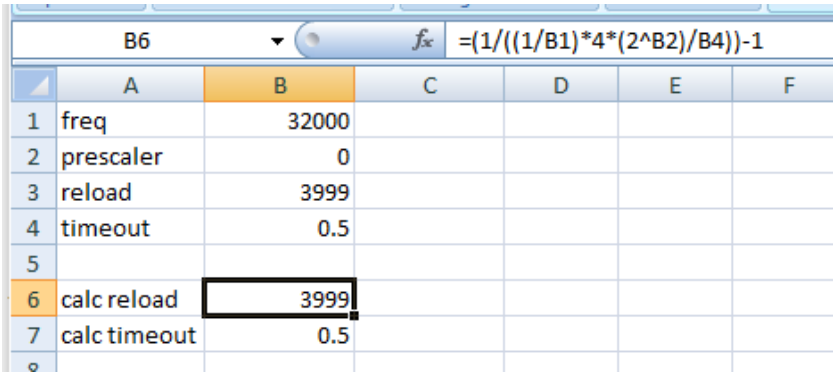
Uncomment MX_IWDG_Init() code to test the watchdog.

Calculating the reload value using the formula defined by ST seen below, we end up with an IWDG_RLR value of 3999.

- Setting the IWDG timeout by using the following formula:
- t_IWDG (ms) = t_LSI (ms) x 4 x 2^(IWDG_PR[2:0]) x (IWDG_RLR[11:0] +1)
- where t_LSI (ms) = 1/32000 = 0.03125
- Min. and max. timeout values from 125 µs to 32.8 s

Using the below variables:

- t_IWDG = 500
- t_LSI = 0.03125
- Prescaler of 4 or IWDG_PR = 0



Nathan Bunnell
Embedded Controller Programming for Real-Time Systems
ECE-40097
Student ID: U08895857

Date: 8/30/2021

```
476  * @brief IWDG Initialization Function
480 static void MX_IWDG_Init(void)
481 {
482
483   /* USER CODE BEGIN IWDG_Init 0 */
484
485   /* USER CODE END IWDG_Init 0 */
486
487   /* USER CODE BEGIN IWDG_Init 1 */
488
489   /* USER CODE END IWDG_Init 1 */
490   hiwdg.Instance = IWDG;
491   hiwdg.Init.Prescaler = IWDG_PRESCALER_8;
492   hiwdg.Init.Window = 3999;
493   hiwdg.Init.Reload = 3999;
494   if (HAL_IWDG_Init(&hiwdg) != HAL_OK)
495   {
496     Error_Handler();
497   }
498   /* USER CODE BEGIN IWDG_Init 2 */
499
500   /* USER CODE END IWDG_Init 2 */
501
502 }
503
```

Implementation of this in the main switch could have been better handled, the blocking nature of the logGetMsg() call meant that if an input was not given in that reset window, the board would reset on its own, as expected but difficult for the purpose of generating an example here.

```
389
390         case ('w'):
391         {
392             logMsg(&huart1, "w");
393
394             // Implement code to delay 1 second and miss the watchdog pet. Should reset board
395             myDelay2(1000);
396
397             break;
398         }
```

Holding down “w” on reset would produce the expected results though.

```
Welcome to Embedded controller programming
 - Enter g for toggling Green LED
 - Enter b for toggling Blue LED
 - Enter s to generate SW interrupt
 - Enter t to start timer 3
 - Enter w to trigger watchdog reset
w
Welcome to Embedded controller programming
```

Nathan Bunnell
Embedded Controller Programming for Real-Time Systems
ECE-40097
Student ID: U08895857

Date: 8/30/2021

Uncomment MX_RTC_Init() to test RTC Alarm

Unfortunately, I was unable to get the alarm portion of the RTC to work. I successfully programmed it to the current time and was able to implement a simple loop to show it was ticking but never saw the alarm callback triggered, despite being able to capture the RTC TR and ALRMAR registers matching while single stepping in the debugger. In the below example, I was testing for an alarm on the 10 second mark and did note that the HAL-provided alarm masks left a value of 0x173010, thus the custom mask set below the commented out section.

```
531  /** Initialize RTC and set the Time and Date
532  */
533  sTime.Hours = 0x07;
534  sTime.Minutes = 0x30;
535  sTime.Seconds = 0x0;
536  sTime.DayLightSaving = RTC_DAYLIGHTSAVING_NONE;
537  sTime.StoreOperation = RTC_STOREOPERATION_RESET;
538  if (HAL_RTC_SetTime(&hrtc, &sTime, RTC_FORMAT_BCD) != HAL_OK)
539  {
540    Error_Handler();
541  }
542  sDate.WeekDay = RTC_WEEKDAY_SATURDAY;
543  sDate.Month = RTC_MONTH_AUGUST;
544  sDate.Date = 0x28;
545  sDate.Year = 0x21;
546
547  if (HAL_RTC_SetDate(&hrtc, &sDate, RTC_FORMAT_BCD) != HAL_OK)
548  {
549    Error_Handler();
550  }
551  /** Enable the Alarm A
552  */
553  sAlarm.AlarmTime.Hours = 0x07;
554  sAlarm.AlarmTime.Minutes = 0x30;
555  sAlarm.AlarmTime.Seconds = 0x10;
556  sAlarm.AlarmTime.SubSeconds = 0x0;
557  sAlarm.AlarmTime.DayLightSaving = RTC_DAYLIGHTSAVING_NONE;
558  sAlarm.AlarmTime.StoreOperation = RTC_STOREOPERATION_RESET;
559  //sAlarm.AlarmMask = RTC_ALARMMASK_SECONDS;
560  //sAlarm.AlarmMask |= (RTC_ALARMMASK_DATEWEEKDAY | RTC_ALARMMASK_HOURS | RTC_ALARMMASK_MINUTE
561  //sAlarm.AlarmMask &= ~RTC_ALARMMASK_SECONDS;
562  sAlarm.AlarmMask &= ~0x01ffffff;
563  sAlarm.AlarmSubSecondMask = RTC_ALARMSUBSECONDMASK_ALL;
564  sAlarm.AlarmDateWeekDaySel = RTC_ALARMDATEWEEKDAYSEL_DATE;
565  sAlarm.AlarmDateWeekDay = 0x0;
566  sAlarm.Alarm = RTC_ALARM_A;
567  if (HAL_RTC_SetAlarm(&hrtc, &sAlarm, RTC_FORMAT_BCD) != HAL_OK)
568  {
569    Error_Handler();
570  }
571  /* USER CODE BEGIN RTC_Init 2 */
```

Nathan Bunnell
Embedded Controller Programming for Real-Time Systems
ECE-40097
Student ID: U08895857

Date: 8/30/2021

```
Current time is: 07:30:09
Unknown character received!

Current time is: 07:30:10
Unknown character received!

Current time is: 07:30:11
Unknown character received!

Current time is: 07:30:12
Unknown character received!
```

Expression	Type	Value
rtcTime	RTC_TimeType...	{...}
hrtc	RTC_HandleTy...	{...}
hrtc.Instance.TR	volatile uint32_t	0x73010
hrtc.Instance.ALRMAR	volatile uint32_t	0x73010
Add new expression		

```
212 void HAL_RTC_AlarmAEventCallback(RTC_HandleTypeDef *hrtc)
213 {
214     if (hrtc->Instance == RTC)
215         logMsg(&huart1, "RTC alarm A detected");
216 }
217
```