Nathan Bunnell
Embedded Real-Time Operating Systems
ECE-40290
Student ID: U08895857

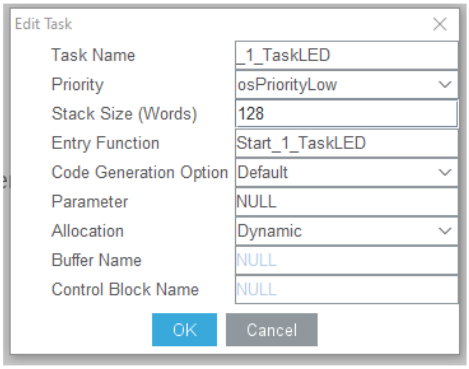
Date: 11/18/2021

Final Assignment 8

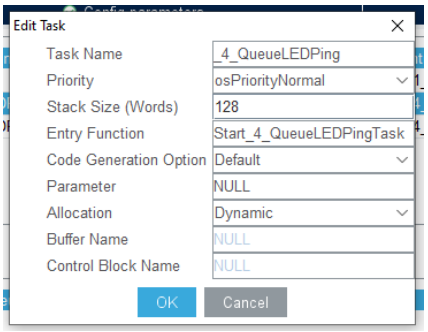
The following will document completion of the final assignment for ECE-40290, with the stated goals of:

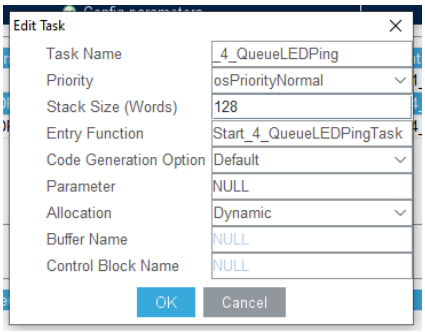
- **TaskLED: This LED will be controlled by a free running "LED Task" that will flash the LED at a given rate.**
- **PeriodicTimerLED: This LED will flash at a rate controlled by a periodic timer.**
- **OneShotTimerLED: This LED will be triggered by the press of the blue button on the STM board. It will cause a one-shot timer to flash the LED on, then off.**
- **QueueLEDs: Here you will create two tasks that "ping pong" back and forth, via a queue, to turn their respective LEDs on/off. Using a queue, the "ping" task will send a command to the "pong" task to flash it's LEDs. In return, the "pong" task will send, via another queue, a command back to the "ping" task to flash it's LED.**

Initial setup will include several steps covered in previous assignments. After the generic FreeRTOS setup steps, we will configure TaskLED:



Next will be the task pair for QueueLEDs, _4_QueueLEDPing & _4_QueueLEDPong:

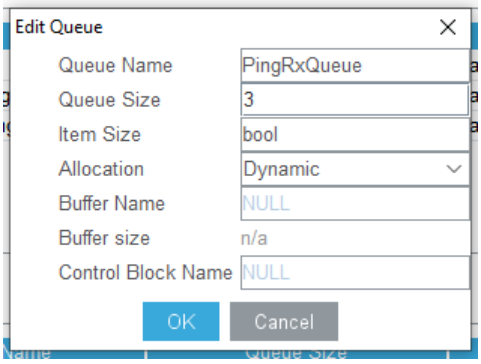


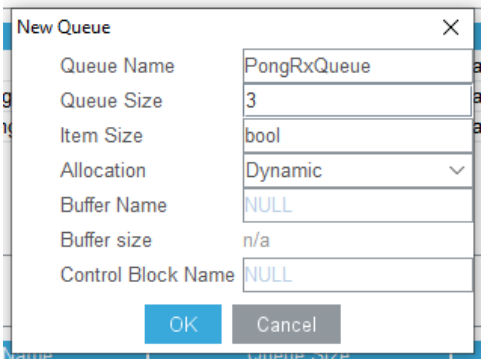


Nathan Bunnell
Embedded Real-Time Operating Systems
ECE-40290
Student ID: U08895857

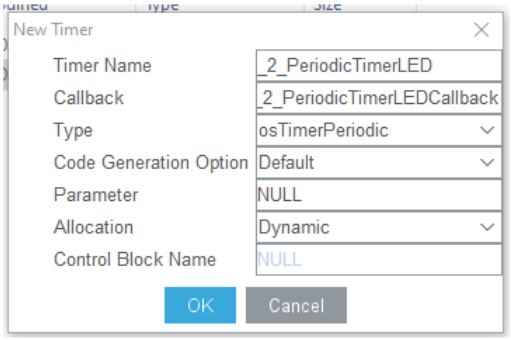
Date: 11/18/2021

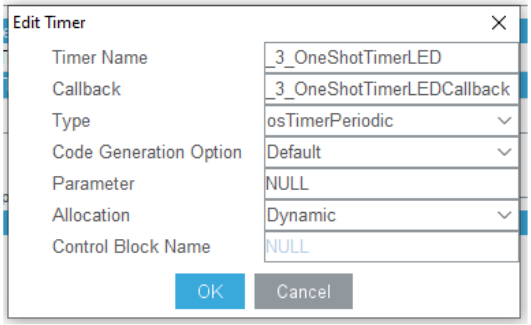
Along with their associated queues, PingRxQueue & PongRxQueue:





Next is the timer configuration for PeriodicTimerLED & OneShotTimerLED, ensuring that configUSE_TIMERS is enabled:





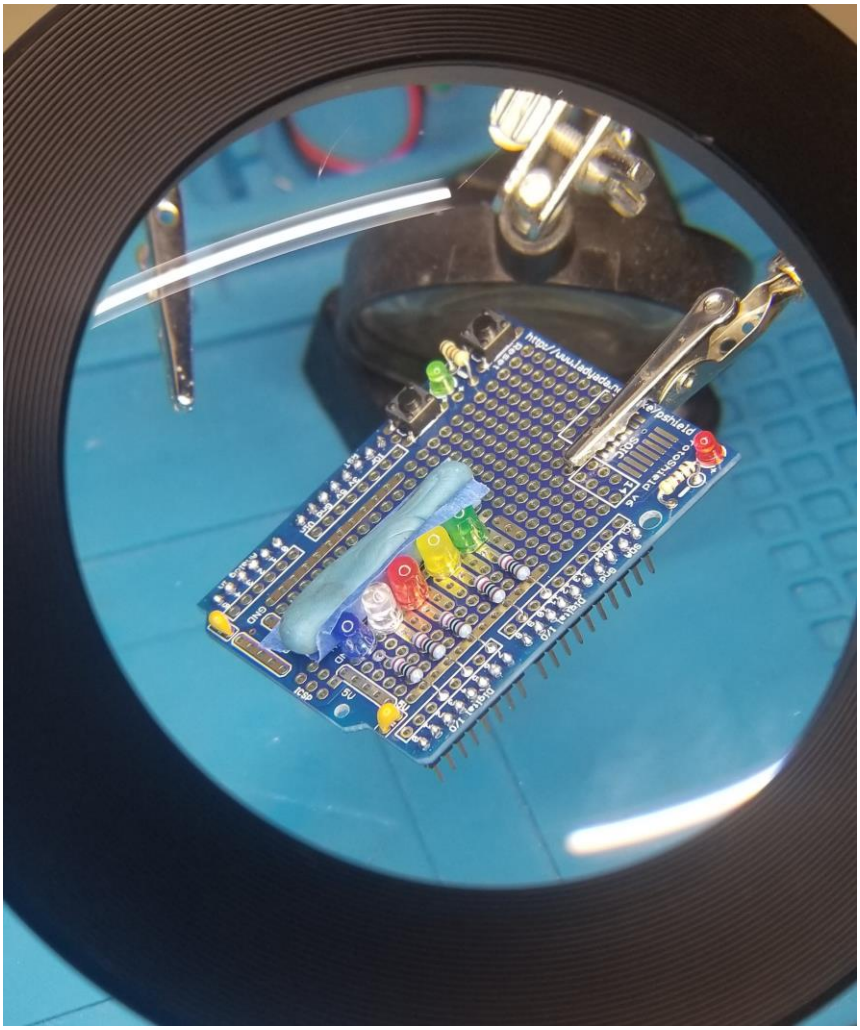
As a final step in the IDE setup, I disabled all non-used peripherals and relabeled the GPIO that would be used for this assignment:

Pin Name	Signal o...	GPI...	GPIO mode	GPIO Pull-up/Pull-down	Maximum o...	Fast Mo...	
PB0	n/a	Low	Output Push Pull	No pull-up and no pull-d...	Low	n/a	ARD_D03
PB4 (NJTRST)	n/a	Low	Output Push Pull	No pull-up and no pull-d...	Low	n/a	ARD_D05
PA4	n/a	Low	Output Push Pull	No pull-up and no pull-d...	Low	n/a	ARD_D07
PA15 (JTDI)	n/a	Low	Output Push Pull	No pull-up and no pull-d...	Low	n/a	ARD_D09
PA7	n/a	Low	Output Push Pull	No pull-up and no pull-d...	Low	n/a	ARD_D11
PB12	n/a	Low	Output Push Pull	No pull-up and no pull-d...	Low	n/a	ISM43362_BOO

Nathan Bunnell
Embedded Real-Time Operating Systems
ECE-40290
Student ID: U08895857

Date: 11/18/2021

Moving over to the hardware side, I used the Arduino GPIO D3, 5, 7, 9, & 11 as opposed to those strictly specified in the requirements. This choice was made because I also employed an Adafruit Proto Shield instead of a breadboard and these allowed for a very clean layout in the final build.



I was pretty pleased with how clean this came out until I plugged the shield in and ran a simple test program to toggle each GPIO when I realized I'd made that rookie mistake: inverting the polarity on **ALL FIVE** of LEDs… what should have been a 10-minute job turned into an hour of work removing the diodes, cleaning remaining solder out of each via with a braid and vacuum, then starting over. On a second test, I also discovered a short between the 5v and GND rails but couldn't trace it down and ended up cutting the 5v trace to that section of the board. Finally, hardware is assembled and ready to go and we can move to programming.

Nathan Bunnell
Embedded Real-Time Operating Systems
ECE-40290
Student ID: U08895857

Date: 11/18/2021

1. **TaskLED: This LED will be controlled by a free running "LED Task" that will flash the LED at a given rate.**

This task is extremely straightforward: simply toggle the LED tied to Arduino Digital 3 on a 2.5 second cadence.

```
519  void Start_1_TaskLED(void const * argument)
520  {
521    /* USER CODE BEGIN 5 */
522    /* Infinite loop */
523    for(;;)
524    {
525          // Description: This LED will be controlled by a free running
526          // "LED Task" that will flash the LED at a given rate.
527          HAL_GPIO_TogglePin(ARD_D03_GPIO_Port, ARD_D03_Pin);
528          osDelay(2500);
529
530    }
531    /* USER CODE END 5 */
532  }
```

2. **PeriodicTimerLED: This LED will flash at a rate controlled by a periodic timer.**

Again, pretty simple: configure the timer's period in *main()* and then define a callback function to toggle the LED on Arduino Digital 5:

```
137     /* USER CODE BEGIN RTOS_TIMERS */
138     /* start timers, add new ones, ... */
139     osTimerStart(_2_PeriodicTimerLEDHandle, 5000);
140
```

```
618  /* _2_PeriodicTimerLEDCallback function */
619  void _2_PeriodicTimerLEDCallback(void const * argument)
620  {
621    /* USER CODE BEGIN _2_PeriodicTimerLEDCallback */
622
623          // Description: This LED will flash at a rate controlled by a
624          // periodic timer.
625          HAL_GPIO_TogglePin(ARD_D05_GPIO_Port, ARD_D05_Pin);
626
627    /* USER CODE END _2_PeriodicTimerLEDCallback */
628  }
```

Nathan Bunnell
Embedded Real-Time Operating Systems
ECE-40290
Student ID: U08895857

Date: 11/18/2021

3. **OneShotTimerLED: This LED will be triggered by the press of the blue button on the STM board. It will cause a one-shot timer to flash the LED on, then off.**

Here we first define an interrupt callback to define how the button press is handled, which will server to start the timer, then a callback for the timer, which will toggle the LED on Arduino Digital 7:

```
76  // Define callback function for user button
77  void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
78  {
79          if (GPIO_Pin == GPIO_PIN_13)
80          {
81                  osTimerStart(_3_OneShotTimerLEDHandle, 3000);
82          }
83  }
84
```

```
630 /* _3_OneShotTimerLEDCallback function */
631 void _3_OneShotTimerLEDCallback(void const * argument)
632 {
633   /* USER CODE BEGIN _3_OneShotTimerLEDCallback */
634
635         // Description: This LED will be triggered by the press of the
636         // blue button on the STM board. It will cause a one-shot timer
637         // to flash the LED on, then off.
638         HAL_GPIO_TogglePin(ARD_D07_GPIO_Port, ARD_D07_Pin);
639
640   /* USER CODE END _3_OneShotTimerLEDCallback */
641 }
```

4. **QueueLEDs: Here you will create two tasks that "ping pong" back and forth, via a queue, to turn their respective LEDs on/off. Using a queue, the "ping" task will send a command to the "pong" task to flash it's LEDs. In return, the "pong" task will send, via another queue, a command back to the "ping" task to flash it's LED.**

This portion was easily the most complicated to implement. The premise is that the Ping and Pong tasks will each read from their respective receive-queues. Each queue is configured to contain Boolean flags equal to either *pdTRUE* or *pdFALSE*. The main body of each task is essentially the same: if there is something to be read in the queue, store it as the associated received flag and test it for true or false. If true, toggle the associated LED, either Arduino Digital 9 or 11, then return a *pdTRUE* as a transmit flag to the other's receive-queue. If false, return the same. Of note, one of the tasks, Ping in this case, needs to provide the initial serve to start the back and forth interaction. Reviewing this code the day after writing it, I think the *else{}* section is probably unnecessary, as the receive-queue will either have a value or it won't, and sending a false could possibly start a chain where neither would ever send a true again. A simpler version is probably just to receive, toggle, & send. Either way, this worked without issue in my testing.

Nathan Bunnell
Embedded Real-Time Operating Systems
ECE-40290
Student ID: U08895857

Date: 11/18/2021

```
541  void Start_4_QueueLEDPingTask(void const * argument)
542  {
543    /* USER CODE BEGIN Start_4_QueueLEDPingTask */
544
545          bool pingRxFlag = pdFALSE;
546          bool pongTxFlag = pdTRUE;
547
548          // Initial "serve"
549          xQueueSend(PongRxQueueHandle, &pongTxFlag, portMAX_DELAY);
550
551    /* Infinite loop */
552    for(;;)
553    {
554            if (uxQueueMessagesWaiting(PingRxQueueHandle) != 0)
555            {
556                    xQueueReceive(PingRxQueueHandle, &pingRxFlag, portMAX_DELAY);
557
558                    if (pingRxFlag == pdTRUE)
559                    {
560                            HAL_GPIO_TogglePin(ARD_D09_GPIO_Port, ARD_D09_Pin);
561                            pongTxFlag = pdTRUE;
562                    }
563                    else
564                    {
565                            pongTxFlag = pdFALSE;
566                    }
567
568                    xQueueSend(PongRxQueueHandle, &pongTxFlag, portMAX_DELAY);
569
570            }
571
572      osDelay(1000);
573    }
574    /* USER CODE END Start_4_QueueLEDPingTask */
```

Nathan Bunnell
Embedded Real-Time Operating Systems
ECE-40290
Student ID: U08895857

Date: 11/18/2021

```
584  void Start_4_QueueLEDPongTask(void const * argument)
585  {
586    /* USER CODE BEGIN Start_4_QueueLEDPongTask */
587
588          bool pongRxFlag = pdFALSE;
589          bool pingTxFlag = pdFALSE;
590
591    /* Infinite loop */
592    for(;;)
593    {
594            if (uxQueueMessagesWaiting(PongRxQueueHandle) != 0)
595            {
596                    xQueueReceive(PongRxQueueHandle, &pongRxFlag, portMAX_DELAY)
597
598                    if (pongRxFlag == pdTRUE)
599                    {
600                            HAL_GPIO_TogglePin(ARD_D11_GPIO_Port, ARD_D11_Pin);
601                            pingTxFlag = pdTRUE;
602                    }
603                    else
604                    {
605                            pingTxFlag = pdFALSE;
606                    }
607
608                    xQueueSend(PingRxQueueHandle, &pingTxFlag, portMAX_DELAY);
609
610            }
611
612      osDelay(1000);
613
614    }
615    /* USER CODE END Start_4_QueueLEDPongTask */
```

Nathan Bunnell
Embedded Real-Time Operating Systems
ECE-40290
Student ID: U08895857

Date: 11/18/2021

Closing Thoughts

This was a great wrap up to the course. The first three sections were simple to implement, and I think more importantly, easy to do from memory, which reinforces how easy this course made the learning to be. The fourth component was a bit more difficult but fun to put together, especially considering the table tennis analogy. I didn't want to risk running out of time to implement it but I actually spent lunch on Tuesday at work sketching out how I'd create a one-dimensional ping-pong game between two players on that proto board with a report on scoring and a play clock reported over serial or a small display. Maybe a project for the future…

I can see a lot of potential to applying these skills in other areas; one I'm particularly interested in experimenting with is the FreeRTOS port to the RP2040 and implementing one core to handle the real-time interactions and another to handle the various task management.