Free RTOS - IoT Node UART + ADC Design and Results , Joseph Kuss

It looks like my UART + ADC conversion implantation can read the ADC twice and report results, but upon doing this a third time it crashes and ends up at the "HardFault_Handler(void)" routine inside of "stm32l4xx it.c"

```
File Edit Setup Control Window Help

Serial Command Interpreter: v0.03 Copyright March 23, 2019, J.M. Kuss

c[1]

Command requested: C[1]

OK, trigger the ADC !

ADC value read: 4042

One of 16384 samples in 0 Milliseconds.

c[1]

Command requested: C[1]

OK, trigger the ADC !

ADC value read: 4042

One of 16384 samples in 0 Milliseconds.

c[1]

Command requested: C[1]

OK, trigger the ADC !

ADC value read: 4042

One of 16384 samples in 0 Milliseconds.
```

```
■ HAL_ADC_ConvCpltCallback() at adc_jmk.c:44 0x8 ∨
                                             □ a serialCmdParser.c
                                  >
          adc_jmk.c @ main.c @ tasks.c @ serialCmdParser.c
c freertos.c
1019/**
       * @brief This function handles Hard fault interrupt.
102
103
1040 void HardFault_Handler(void)
105 {
106
      /* USER CODE BEGIN HardFault IRQn 0 */
      /* USER CODE END HardFault_IRQn 0 */
108
109
      while (1)
110
         /* USER CODE BEGIN W1_HardFault_IRQn 0 */
111
112
         /* USER CODE END W1_HardFault_IRQn 0 */
 113
114 }
```

I am using breakpoints and the debugger to do this, this does seem to be repeatable.

Note: I am surprised that the problem occurs on the third occurrence and not the second, since at if there was a problem restarting the acquisition after the first set of readings, one would think would occur after the first restart, rather than after the second restart.

Digging deeper I will look at this info:

https://www.freertos.org/Debugging-Hard-Faults-On-Cortex-M-Microcontrollers.html

Maybe also:

https://stackoverflow.com/questions/50243996/what-are-valid-values-of-hal-nvic-setpriority-when-using-stm32-and-freertos

But before I do this, let me illustrate to a greater degree the design of my implementation code.

One important comment consideration:

My command interpreter and UART functionality, seems to work OK, as implemented using FreeRTOS, except when I attempt to use a command "C[1]" to trigger a set of 16K ADC readings.

My command interpreter is generic. It is coded to report not implemented commands so I can easily test and demonstrate functionality. I can send a bunch of commands, and send back to the serial console expected stub responses.

So, If I only wanted to implement UART and command/response functionality in FreeRTOS, then things are working great!.

As shown below:

```
COM10 - Tera Term VT
File Edit Setup Control Window Help
Serial Command Interpreter: v0.03 Copyright March 23, 2019, J.M. Kuss
Unknown command !
nnnnn
Unknown command !
c[0]
Command requested: C[0]
Index Expected.
Unknown command !
 ommand requested: C[2]
[255]
ommand requested: C[255]
[256]
Index range Exceeded.
Unknown command !
```

But it is also desired to exercising the ADC conversion capability of the STM32L475 while using FreeRTOS. The only command in my "do whatever you want" generic command interpreter that is not a stub, is command "c[1]" to start a 16K sample continuous conversion which as shown in the first illustration, works twice and then cause a Hard fault trap.

So my big quest is to debug this sample conversion command.

I will now show you the sum total design for Free RTOS on following pages:

The most important task is "comProcessor" explained and illustrated in 3 screen shots below:

```
] 🔚 🐚 🦠 🦠 🐧 😘 🍅 🏺 🗎 😕 🧳 🕶 💬 🐤 🗘 🕶 🖯 🕶 🗆 🗆 🗆 🗆 🕶 🛣 🗗
    d freertos.c ⋈ d adc_jmk.c
                                 .c serialCmdPa...
                                                   .c main.c
                                                                .c uart_imk.c
                                                                               .h uart_imk.
7
     158 }
     159
     160 /* USER CODE BEGIN Header startComProcessor */
     1619 /**
          * @brief Function implementing the comProcessor thread.
     162
     163 * @param argument: Not used
     164 * @retval None
     165
     166 /* USER CODE END Header startComProcessor */
     167@ void startComProcessor(void const * argument)
     168 {
              /* USER CODE BEGIN startComProcessor */
     159
             // Send power-up Id msg to Com port console:
     170
     171
              // (false = Non Blocking TX)
     172     UartPutString(&huart4, HW_msg, false);
     173
             // osDelay(500);
     174
     175
             // Wait for power up id msg to complete
             osSemaphoreWait( binarySP_TX_CompletedHandle , osWaitForever);
     176
     177
     178
             // Activate non blocking UART xx interrupt every time get 1 byte..
     179
             HAL UART Receive IT(&huart4, Rx data, 1);
     180
             // For some reason there is a false \underline{r}\underline{x} \underline{m}\underline{s}\underline{g} upon IT start-up,
     181
     182
             // these two lines flush it out.
     183
             osSemaphoreWait(binarySP_RX_DataReadyHandle, osWaitForever);
     184
              cmdHandler(cmdBuffer);
     185
```

This is the "before the standard infinite for loop expected in a task" code. This code tests as reliable as seen in the previous illustration (..) It sends the opening message (Name and copyright info) to the console, and then enable receive interrupts in the HAL, at line 179, to get commands for command interpreter)

Lines 183 and 184 handle an unexpected interrupt I seem to get, at startup, indicating an incoming command, even though the end user has not typed anything at the console. Adding these two lines causes this "noise" command to be ignored.

The top part of the infinite for loop is an if block that only is executed if the previous command was c[1] which is the request to start a 16K sample continuous ADC reading.

```
urce Refactor View Navigate Search Project Run Window Help
.c freertos.c ⋈ .c adc_jmk.c
                                      .c main.c
                                                   c tasks.c
                                                                c serialCmdParser.c
                                                                                      c uart_imk.c
                 /* Infinite loop */
         198
         199
                 for(;;)
UART
         200
TOS2
         201
                     // if adcTriggerOut is true the previous command was c[1]
                     if (adcTriggerOut == true)
)S-Inte
         283
)S-Resc
                                              // #### helpful.. ###
         204
                        osDelay(100);
                        adcTriggerOut = false; // clear until next adcTrigger c[1] command.
)SEven
         205
         286
t_With
         207
                        // The previous cmd msg, c[1], triggered the ADC.
t_With
                        // Here we block waiting for the ADC task to set flag (semaphore)
         208
                        // "ADC data is ready" to send back to the PC console
         209
12
                       osSemaphoreWait(binarySP_ADC_DataToComHandle, osWaitForever);
         210
13
                        // Put together "here is your data message" to send to PC
         211
                       cmdResponseOfADC_Data();
// Transmit this msg in "respBuffer" back to PC.
        212
         213
                       UartPutString(&huart4, respBuffer, false);
                        // Block while we are transmitting to the PC console:
         215
         216
                        // (Do not attempt to receive while we are transmitting..)
                        osSemaphoreWait( binarySP_TX_CompletedHandle , osWaitForever);
         217
         218
```

After a c[1] command and its initial response of "OK, trigger the ADC" that is setup at line 463 of serialCmdParser.c, then we are expecting ADC results, and adcTriggerOut has been set to true. So in this case we are in this "if" statement and block at line 210 until the ADC has completed it's work which occurs when the following chain of events happened.

- 1) The ADC ISR has executed the single sample received call back "HAL_ADC_ConvCpltCallback(..)" for 16 K samples as seen starting at line 45 in the file adc_jmk.c, where it Releases the semaphore "ADC_DataReady" at line 58.
- 2) The second task in this program known as "CheckADC" unblocks upon getting the semaphore flag "ADC_DataReady" form the ADC ISR callback, and then releases a different semaphore to "ADC_DataToCom" to signal the "ComProcessor" task as that data is ready to return to the PC.

Notes

"CheckADC" task (higher priority than "ComProcessor") is a middle man task, and could be eliminated.

To keep things simple even though the ADC has taken 16K samples only the first sample is sent back to the PC

So, now then, after we got the ADC sampling done then our task above unblocks (resumes) at line 210 of freertos.c as seen above, and then calls "cmdResponseOfADC_Data" which formats it to look like

```
ADC value read: 4042
One of 16384 samples in 0 Milliseconds.
```

And puts it in respBuffer, which is transmitted at line 214 and afterwards we block (wait) at line 217 for UART ISR callback "HAL_UART_TxCpltCallback(..)" to call osSemaphoreRelease(binarySP_TX_CompletedHandle)", in uart jmk.c line 146, to flag the transmission as complete.

The rest of the "ComProcessor" task is also important, for all other routine command processing as shown below:

```
€ freertos.c 🛭 🖟 adc_jmk.c
                                         c tasks.c
                                                     c serialCmdParser.c
                                                                          .c uart_imk.c
                                                                                          .c stm32l4xx_it.c
                             .c main.c
220
              // Block to wait for reception of possible command from PC
              // The command receive is handled by ISR, and the ISR callback
 221
             // routine "HAL_UART_RxCpltCallback(UART_HandleTypeDef *hwart)", in uart_jmk.c
222
223
             // which releases the following semaphore:
224
             osSemaphoreWait(binarySP_RX_DataReadyHandle, osWaitForever);
225
226
             // Process command from the user at the serial console:
 227
              cmdHandler(cmdBuffer, &adcTriggerOut);
 228
229
              // Transmit the command response back to the user looking at the console.
 230
             UartPutString(&huart4, respBuffer, false);
 231
 232
             // Block while we are transmitting something back to the PC console:
 233
              // (Do not attempt to receive while we are transmitting...)
 234
             osSemaphoreWait( binarySP_TX_CompletedHandle , osWaitForever);
 235
             // If command processed was c[1] then adcTriggerOut" will be true now.
 236
              // this affects what happens next at top of for loop for this task.
 237
          } // End of for(;;)
 238
 239
240
        /* USER CODE END startComProcessor */
 241 }
 242
```

This code is a bit more straight forward and occurs for incoming command messages from the USB virtual comport on the PC.

One line 224 we wait for the flag (semaphore) "SP_RX_DataReady" this flag is set (released) at line 109 of uart_jmk.c inside of the UART RX ISR callback routine "HAL_UART_RxCpltCallback(..)" when the the message has arrived.

On line 227 we call "cmdHandler(..)", written by JMK, earlier, to interpret and respond to serial commands. This routine does not actually transmit back a response but instead just sets up the appropriate response in "respBuffer" which is then transmitted back to the PC via call to "UartPutString(..)" at line 230.

Finally we wait (block) for the response to the command transmission to complete at line 234 waiting for UART ISR callback "HAL_UART_TxCpltCallback(...)" to call osSemaphoreRelease(binarySP_TX_CompletedHandle)", in uart_jmk.c line 146, to flag the transmission as complete.

Also provided below is "CheckADC" task (which possibly be eliminated) as well as the HAL ISR call back functions for the ADC, and RX byte and TX msg complete UART interrupts, as well as the execution of command c[1] inside of serialCmdParser.c.

"CheckADC" Task:

```
c serialCmdParser.c
     ☐ freertos.c 🏻 🖸 adc_jmk.c
                                           c tasks.c
                                                                        c uart_imk.c
                               .c main.c
                                                                                       .c stm.
     247 * @retval None
      248 */
JART_
      249 /* USER CODE END Header_startCheckADC */
DS2
     2500 void startCheckADC(void const * argument)
      251 {
-Inte
     252
            /* USER CODE BEGIN startCheckADC */
-Resc
      253
Even
           /* Infinite loop */
      254
With
      255
           for(;;)
With
      256
           {
              // Wait for data read from ADC (1 or more samples acquired.)
      257
      259
      260
              // The semaphore release/give here will trigger,
              // certain actions in the "ComProcessor" task to let it know
      261
      262
              // in has a/d data it can send back to user console on PC.
      263
      264
              osSemaphoreRelease(binarySP_ADC_DataToComHandle);
      266
              // Under certain conditions we may want to automatically
      267
              // resume the next acquisition once we get the previous results here:
              // ####### halADC_Status = HAL_ADC_Start_IT(&hadc1);
      268
      269
              // Presently this is called rather upon the next command to get samples
      270
              // "c[1]" at line 466 of serialCmdParser.c.
      271
      272
            /* USER CODE END startCheckADC */
      273
      274 }
     275
```

"ADC ISR Callback":

```
ic adc_jmk.c ⋈ ic main.c
   c freertos.c
                                                     c tasks.c
                                                                    c serialCmdParser.c
                                                                                             .c *uart_iml
    29@ void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc)
     30 {
T
             HAL_StatusTypeDef halADC_Status;
     31
             if (hadc->Instance == ADC1)
     32
     33
e
                 tsPair.sample[adcSampleCount % 2] = HAL_ADC_GetValue(&hadc1);
     34
     35
                 if ((adcSampleCount % 2) == 1) // We just sampled the second of the pair.
S(
n
                      // Time to save both in two sample element of packed sample Array
     37
h
     38
                      adcSample[adcElementCount] = tsPair.element;
     39
                      adcElementCount++;
h
     40
     41
                 adcSampleCount++;
     43
                 if (adcElementCount == MAX_ADC_ELEMENT_COUNT )
     44
                 {
     45
                     // We filled up the buffer, next we will signal the
                     // runADCtoUART task to switch to UART mode to stop
     46
     47
                     // acquiring samples for a while, and instead dump all (or one)
                     // of the 16 K samples, 32 K bytes of data to the USB com port.
     48
     49
     50
                     // Check the completion time,
                      // reported by serialCmdParser.c after c[1] cmd.
     51
     52
                     xNowTime = xTaskGetTickCount();
     53
     54
                      // Hope stop continuous can be restarted ok...
     55
                     halADC_Status = HAL_ADC_Stop_IT(&hadc1);
     56
                     // This call back is actually from a HAL ISR
     57
                     // CMSIS wrapper for xSemaphoreGiveFromISR(...);
                     // The semaphore release/give here will trigger,
// certain actions in the for(;;) loop of task "checkADC"
     58
     59
     68
                     osSemaphoreRelease(binarySP_ADC_DataReadyHandle);
     61
                 }
     62
            }
     63 }
```

Code that executes command C[1] inside of file "serialCmdParser.c" to start ADC conversion:

```
c freertos.c
               .c adc_jmk.c
                               .c main.c
                                             c tasks.c
                                                           459
468
                         // Make C[1] as a functional command to trigger the ADC"
461
                         if (index == 1)
462
 463
                             strcat(respBuffer, "OK, trigger the ADC ! \r\n");
 464
                             // Start to measure time to convert (16K samples)
465
                             xLastTime = xTaskGetTickCount();
 466
                             halADC_Status = HAL_ADC_Start_IT(&hadc1);
 467
                             // Let calling function know we just triggered the ADC.
 468
                             *adcTriggerOut = true;
                         }
 469
 470
```

"UART ISR RX Byte callback"

```
c tasks.c
                                                                                         © *uart_jmk.c \ □ stm32l4xx_
   c freertos.c
                   c adc_jmk.c
                                    .c main.c
                                                                 c serialCmdParser.c
     64@ void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
     65 {
     66
             uint32 t i:
             if ((huart->Instance == USART1) || (huart->Instance == UART4)) // Current UART1 or 4
     67
     68
e
     69
                 if (Rx_index == 0)
     70
50
     71
                     for (i=0; i<256; i++)
n.
     72
1
     73
                          // Clear buffer before getting new data, to assures is 0 terminated.
     74
                         Rx_Buffer[i]=0;
h
     75
                     }
     76
     77
                  // We presently assume that end of line from "Enter" key, / produces only a <CR>
     78
                 // and not a <LF>. This received byte <CR> is presently not put into the Rx_data buffer.
     88
                 if ((Rx_data[0] != 0x0D) && (Rx_index<255))// If received data different than 13 <CR>
     81
                                                              / and we are not about to overflow buffer,
     82
                                                             // presently limit msg to 255 chars.
     83
                                                             // index 0..254 so Rx_Buffer[255] remains 0
     84
                     Rx_Buffer[Rx_index++] = Rx_data[0];
                                                             // to terminate string.
     85
                     HAL_UART_Receive_IT(huart, Rx_data, 1); // Activate UART xx interrupt every time get 1 byte..
     86
     87
     88
                 else
     89
                     // If we receive Rx data[0] == 0x0D, this byte will be ignored it is only a flag.
     98
     91
                     // Also if Rx index == 255 we will also end up here to attempt to parse
     92
                      // what we have so far since we will loose data if we go any further.
                     Rx_index = 0; // Rx_indx set to zero for next msg.
     93
     94
     95
                     strcpy(cmdBuffer, (char *) Rx_Buffer); // Save the msg buffer into cmd buffer.
     96
     97
                      // New (Want to enable before osSemaphore..) (TRY IT BOTH WAYS ???)
     98
                     HAL_UART_Receive_IT(huart, Rx_data, 1); // Activate UART rx interrupt every time get 1 byte..
     99
     100
                      // This call back is actually from a HAL ISR
     101
                      // CMSIS wrapper here actually calls xSemaphoreGiveFromISR(...);
                       // The semaphore release/give here will trigger,
     102
                       // certain actions in the for(;;) loop of task "ComProcessor"
     103
     104
                      osSemaphoreRelease(binarySP_RX_DataReadyHandle);
     105
                  }
              }
                  // end HAL_UART_RxCpltCallback
     107
          }
     100
```

Note: As part of the development process for this project, I was at first very concerned about (what I believe to be the fact that these callback functions are actually calls made inside of the ISR so any native FreeRTOS semaphore calls must actually call the versions ending in "FromISR(..)". To my pleasant surprise I found this is taken care of by the CMSIS wrapper functions so that the proper semaphore call is actually made inside of the CMSIS function "osSemaphreRelease", eg. at line 104 above.

"UART ISR TX MSG callback"

```
125@ void HAL_UART_TxCpltCallback(UART_HandleTypeDef *huart)
126 {
127
         // For starters
         /* Prevent unused argument(s) compilation warning */
128
129
         UNUSED(huart);
130
         // This call back is actually from a HAL ISR
131
         // CMSIS wrapper here actually calls xSemaphoreGiveFromISR(...);
132
         // The semaphore release/give here will trigger,
133
         // certain actions in the for(;;) loop of task "ComProcessor"
134
135
         osSemaphoreRelease(binarySP_TX_CompletedHandle);
136 }
```

Conclusion:

I still have to work on the program goes to "Hard Fault" upon the third 16 K ADC sample set conversion command. Some things I will try

- 1) Revert to single conversion mode since (the 16K sample) mode is more intended to measure how fast the sampling rate was. (I need to look into why use of the Free RTOS TaskGetTickCount(), calls did not work ... I wonder if xTaskGetTickCount() only works if called from inside a task? Note the call to xTaskGetTickCount() to get completion time for 16 K samples is made inside of the ADC ISR callback function rather than inside a task.
- Look into what exactly may have caused the "Hard fault" as mentioned on page 1
 https://www.freertos.org/Debugging-Hard-Faults-On-Cortex-M-Microcontrollers.html
 One temporary work around is to see single conversion mode allows more than 2 calls to c[0].

Finally, please consider the information provided in this document, does provide a viable starting point for future 100% completion of my objectives and as such believe it is a valuable component to aid in understanding. I hope this document is also useful to others.

I will post you on future updates and solutions; see last two pages below:

Next Day - Success!

So what was causing the "hard fault" anyways?

Actually it was a violation of Free RTOS recommended policy. "if the function does not end in "FromISR" then don't call it from an ISR." Ref:

https://www.freertos.org/FreeRTOS Support Forum Archive/February 2010/freertos Call to xTaskGetTickCount in an ISR 3566739.html

One must keep in mind that any HAL ISR callback function is really still part of the HAL ISR. My problem was at line 54 below.

```
c tasks.c
                                                        c serialCmdParser.c
                                                                             uart_jmk.c
   33@ void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc)
   34 {
  35
            HAL_StatusTypeDef halADC_Status;
   36
           if (hadc->Instance == ADC1)
5
   38
               tsPair.sample[adcSampleCount % 2] = HAL ADC GetValue(&hadc1);
25
   39
               if ((adcSampleCount % 2) == 1) // We just sampled the second of the pair.
                    // Time to save both in two sample element of packed sample Array
   40
u
   41
                    adcSample[adcElementCount] = tsPair.element;
   42
                    adcElementCount++:
   44
                adcSampleCount++;
               if (adcElementCount == MAX_ADC_ELEMENT_COUNT )
   47
                    // We filled up the buffer, next we will signal the
    48
                    // runADCtoUART task to switch to UART mode to stop
    49
                    // acquiring samples for a while, and instead dump all (or one)
   50
                   // of the 16 K samples, 32 K bytes of data to the USB com port.
   51
                   // Check the completion time,
                    // reported by serialCmdParser.c after c[1] cmd.
                   // ####### moved... xNowTime = xTaskGetTickCount();
   56
                   // Restore adcSampleCount and adcElementCount back to 0
   57
                   adcSampleCount = 0:
   58
                   adcElementCount = 0;
   59
                    // Hope stop continuous can be restarted ok...
   60
                   halADC_Status = HAL_ADC_Stop_IT(&hadc1);
   61
                    // This call back is actually from a HAL
                    // CMSIS wrapper for xSemaphoreGiveFromISR(...);
                   // The semaphore release/give here will trigger,
// certain actions in the for(;;) loop of task "checkADC"
   63
                    osSemaphoreRelease(binarySP_ADC_DataReadyHandle);
```

I relocated this function back to the "CheckADC" task (outside of the ISR) and now it works, Also my time measured not zero any more (more accurate).

```
249 /* USER CODE END Header startCheckADC
     250⊖ void startCheckADC(void const * argument)
pts
     251 {
rces
     252
            /* USER CODE BEGIN startCheckADC */
rou
erupts 254
            /* Infinite loop */
eru
     255
            for(;;)
     256
     257
              // Wait for data read from ADC (1 or more samples acquired.)
              osSemaphoreWait(binarySP ADC DataReadyHandle, osWaitForever);
     258
     259
     260
              // Check the completion time,
     261
              // reported by serialCmdParser.c after c[1] cmd.
     262
              xNowTime = xTaskGetTickCount();
     263
     264
              // The semaphore release/give here will trigger,
     265
              // certain actions in the "ComProcessor" task to let it know
     266
              // in has a/d data it can send back to user console on PC.
     267
              osSemaphoreRelease(binarySP_ADC_DataToComHandle);
```

So now we can send c[1] as for as many times as we want:

```
File Edit Setup Control Window Help

Serial Command Interpreter: v0.03 Copyright March 23, 2019, J.M. Kuss

c[1]
Command requested: C[1]
OK. trigger the ADC !
ADC value read: 13
One of 16384 samples in 1 Milliseconds.
c[1]
Command requested: C[1]
OK, trigger the ADC !
ADC value read: 26
One of 16384 samples in 1 Milliseconds.
c[1]
Command requested: C[1]
OK, trigger the ADC !
ADC value read: 26
One of 16384 samples in 1 Milliseconds.
c[1]
Command requested: C[1]
OK, trigger the ADC !
ADC value read: 12
One of 16384 samples in 1 Milliseconds.
c[1]
Command requested: C[1]
OK, trigger the ADC !
ADC value read: 6
One of 16384 samples in 1 Milliseconds.
c[1]
Command requested: C[1]
OK, trigger the ADC !
ADC value read: 16
One of 16384 samples in 1 Milliseconds.
c[1]
Command requested: C[1]
OK, trigger the ADC !
ADC value read: 16
One of 16384 samples in 1 Milliseconds.
```

Best regards, Joe Kuss jmkuss@arrl.net