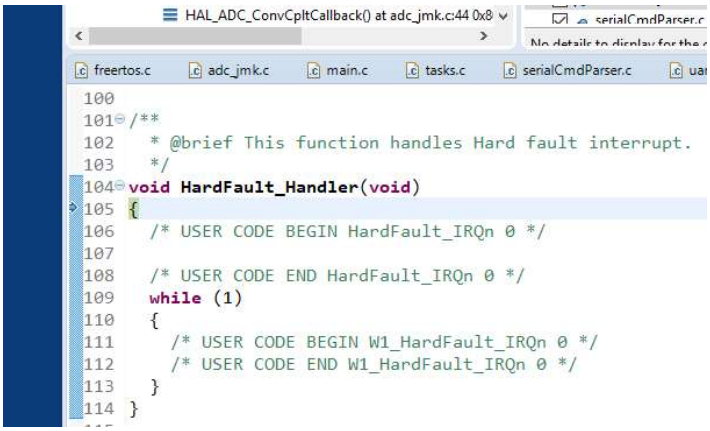
Free RTOS - IoT Node UART + ADC Design and Results , Joseph Kuss

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”



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.

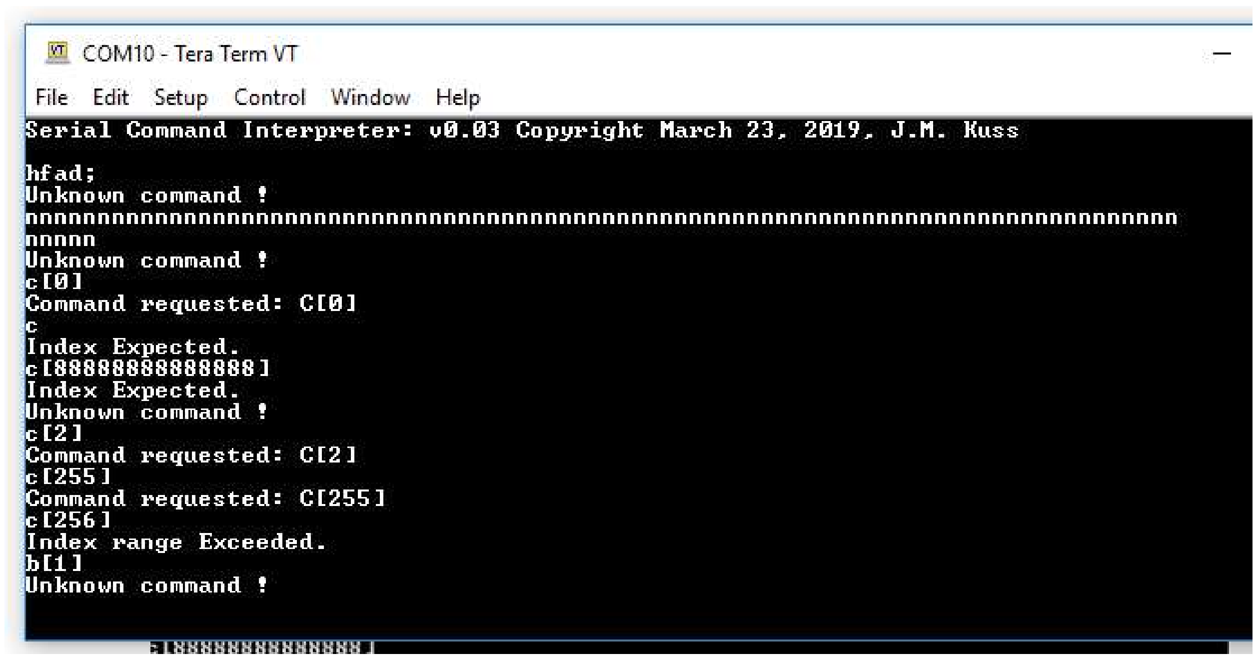
One important comment consideration:

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:

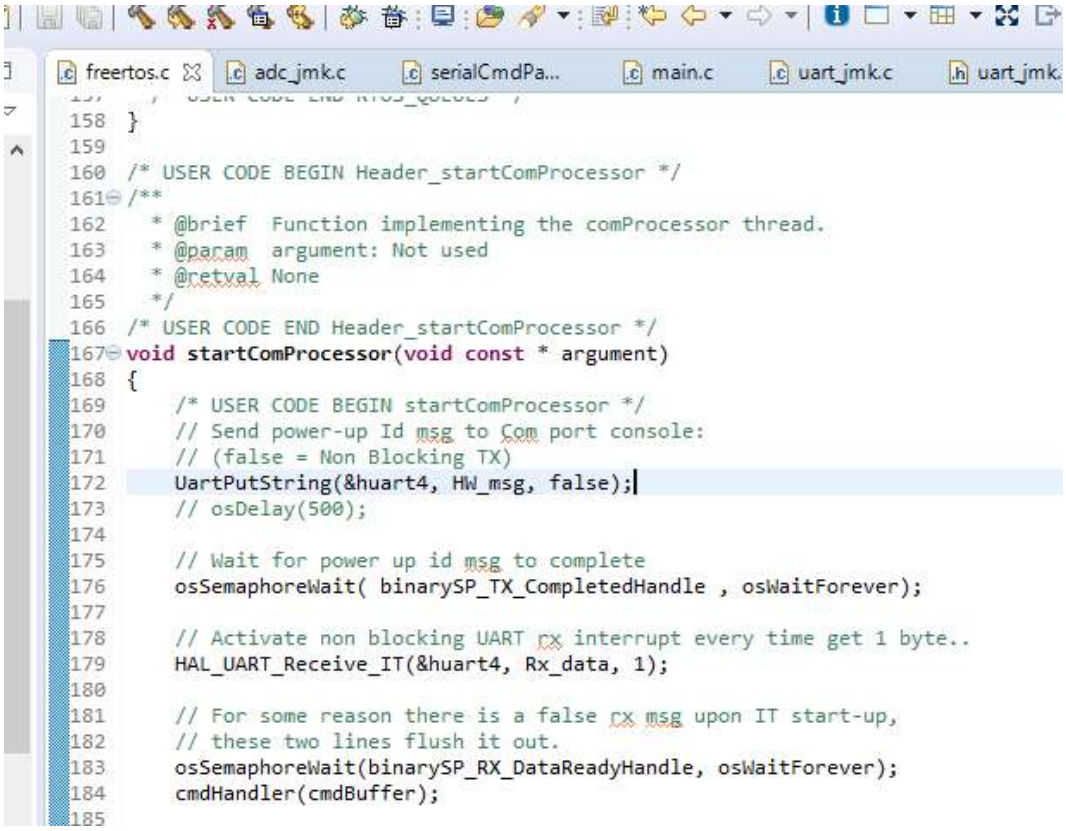


But it is also desired to exercising the ADC conversion capability of the STM32L475 while using FreeRTOS. The only command in my "do whatever you want" generic command interpreter that is not a stub, is command "c[1]" to start a 16K sample continuous conversion which as shown in the first illustration,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

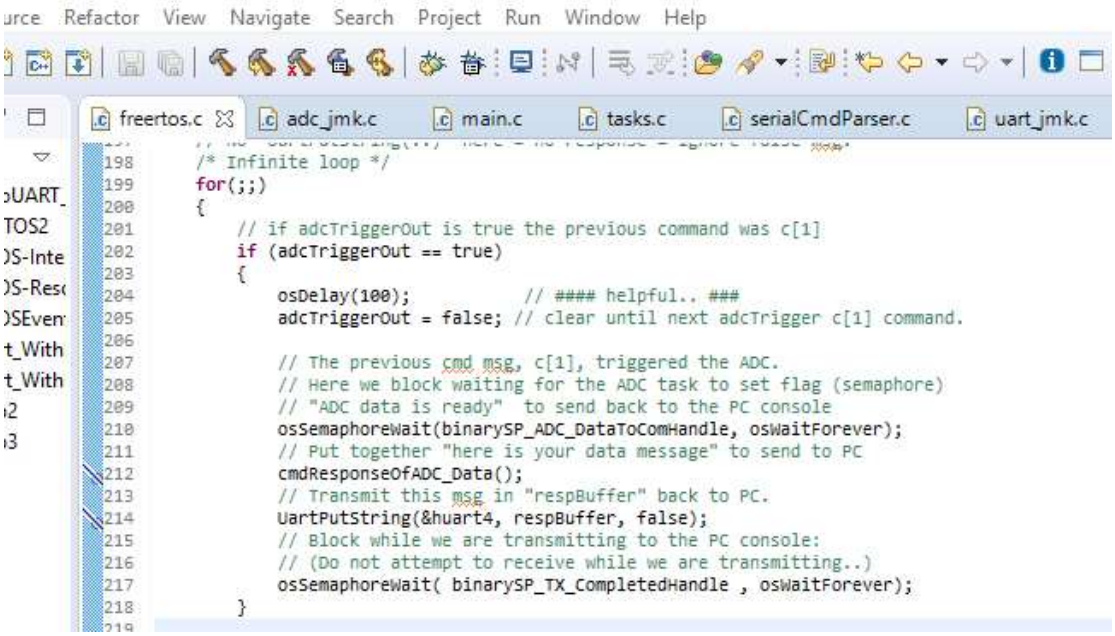
The most important task is “comProcessor” explained and illustrated in 3 screen shots below:



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.



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.

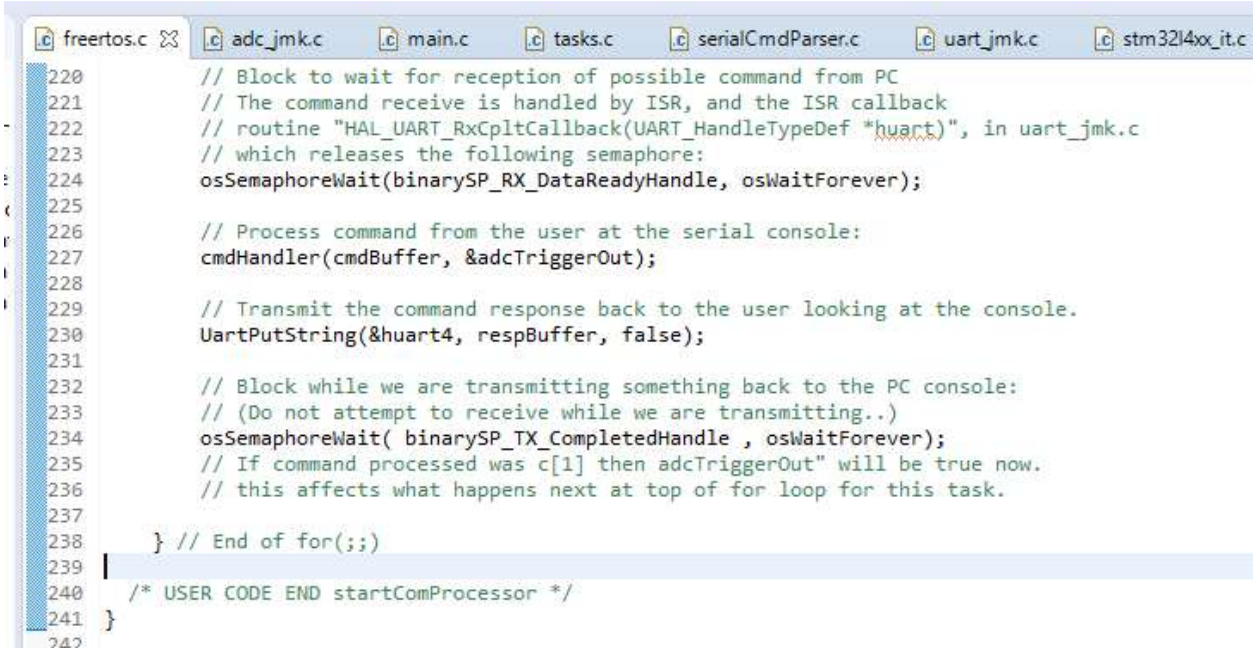
Notes
"CheckADC" task (higher priority than "ComProcessor") is a middle man task, and could be eliminated.
To keep things simple even though the ADC has taken 16K samples only the first sample is sent back to the PC

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:



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

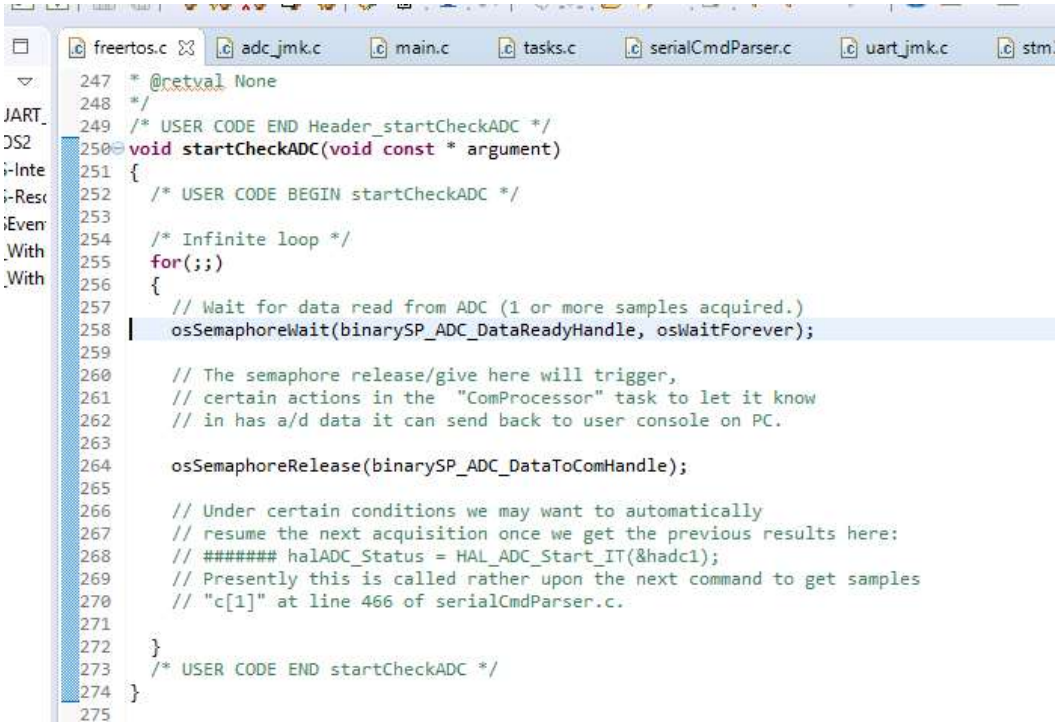
One line 224 we wait for the flag (semaphore) “SP_RX_DataReady” this flag is set (released) at line 109 of uart_jmk.c inside of the UART RX ISR callback routine “HAL_UART_RxCpltCallback(..)” when the the message has arrived.

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:



“ADC ISR Callback”:

freertos.c | adc_jmk.c | main.c | tasks.c | serialCmdParser.c | *uart_jmk

```
29  void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc)
30  {
31      HAL_StatusTypeDef  halADC_Status;
32      if (hadc->Instance == ADC1)
33      {
34          tsPair.sample[adcSampleCount % 2] = HAL_ADC_GetValue(&hadc1);
35          if ((adcSampleCount % 2) == 1) // We just sampled the second of the pair.
36          {
37              // Time to save both in two sample element of packed sample Array
38              adcSample[adcElementCount] = tsPair.element;
39              adcElementCount++;
40
41          }
42          adcSampleCount++;
43          if (adcElementCount == MAX_ADC_ELEMENT_COUNT )
44          {
45              // We filled up the buffer, next we will signal the
46              // runADCtoUART task to switch to UART mode to stop
47              // acquiring samples for a while, and instead dump all (or one)
48              // of the 16 K samples, 32 K bytes of data to the USB com port.
49
50              // Check the completion time,
51              // reported by serialCmdParser.c after c[1] cmd.
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(...);
58              // The semaphore release/give here will trigger,
59              // certain actions in the for(;;) loop of task "checkADC"
60              osSemaphoreRelease(binarySP_ADC_DataReadyHandle);
61          }
62      }
63  }
```

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

freertos.c | adc_jmk.c | main.c | tasks.c | serialCmdParser.c

```
459
460                 // 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”

freertos.c | adc_jmk.c | main.c | tasks.c | serialCmdParser.c | *uart_jmk.c | stm32l4xx_

```
64  void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
65  {
66      uint32_t i;
67      if ((huart->Instance == USART1) || (huart->Instance == UART4)) // Current UART1 or 4
68      {
69          if (Rx_index == 0)
70          {
71              for (i=0; i<256; i++)
72              {
73                  // Clear buffer before getting new data, to assures is 0 terminated.
74                  Rx_Buffer[i]=0;
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.
79
80          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              // New
86              HAL_UART_Receive_IT(huart, Rx_data, 1); // Activate UART rx interrupt every time get 1 byte..
87          }
88          else
89          {
90              // If we receive Rx_data[0] == 0x0D, this byte will be ignored it is only a flag.
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.
93              Rx_index = 0; // Rx_indx set to zero for next msg.
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(...);
102             // The semaphore release/give here will trigger,
103             // certain actions in the for(;;) loop of task "ComProcessor"
104             osSemaphoreRelease(binarySP_RX_DataReadyHandle);
105         }
106     }
107 }       // end HAL_UART_RxCpltCallback
```

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”

```
124
125 void HAL_UART_TxCpltCallback(UART_HandleTypeDef *huart)
126 {
127     // For starters
128     /* Prevent unused argument(s) compilation warning */
129     UNUSED(huart);
130
131     // This call back is actually from a HAL  ISR
132     // CMSIS wrapper here actually calls xSemaphoreGiveFromISR(...);
133     // The semaphore release/give here will trigger,
134     // certain actions in the for(;;) loop of task "ComProcessor"
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.

2) 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.

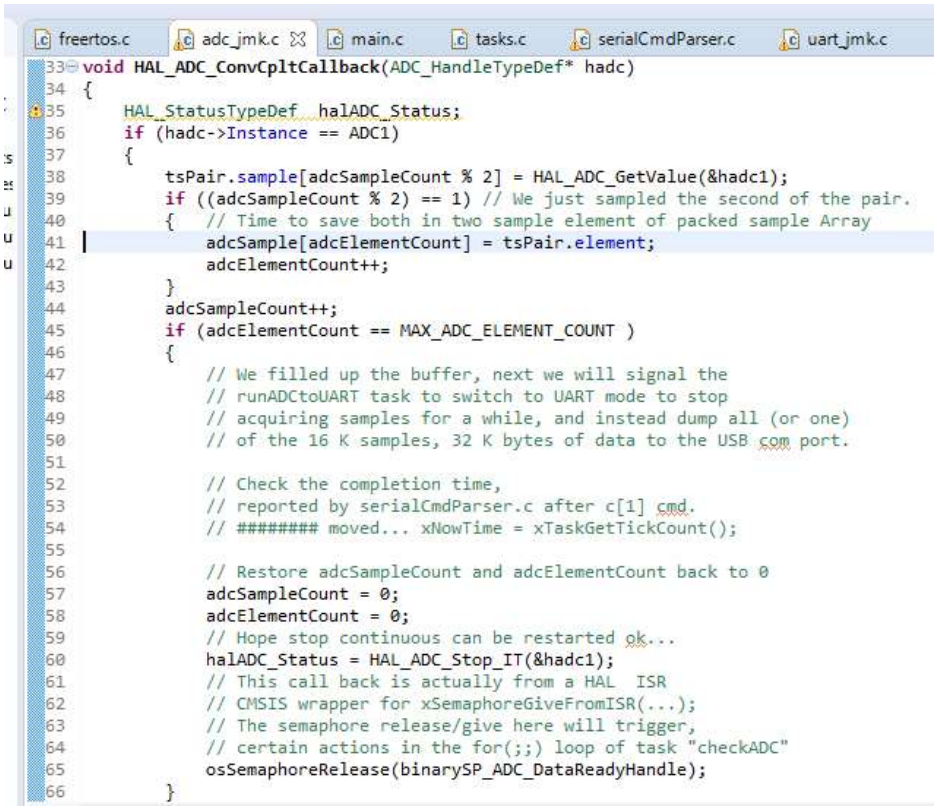
I will post you on future updates and solutions; see last two pages below:

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.

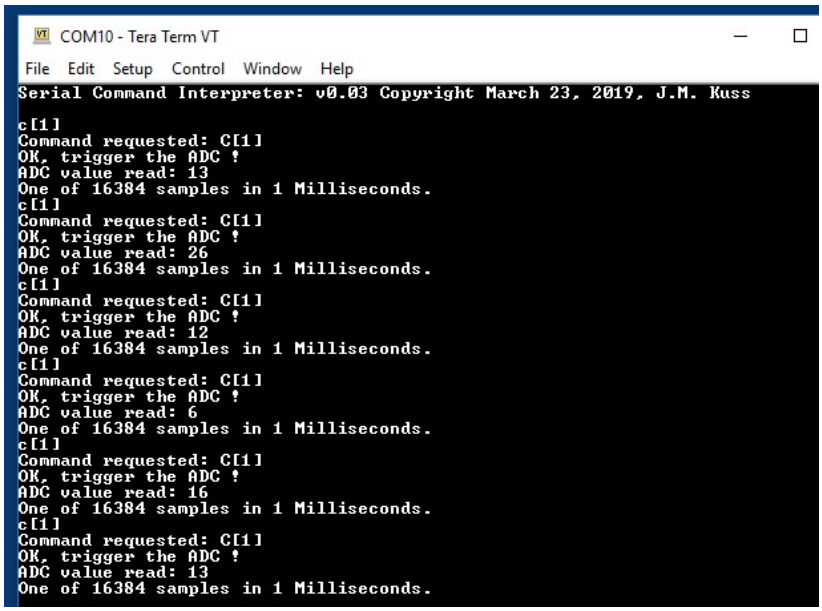



```
33 void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc)
34 {
35     HAL_StatusTypeDef  halADC_Status;
36     if (hadc->Instance == ADC1)
37     {
38         tsPair.sample[adcSampleCount % 2] = HAL_ADC_GetValue(&hadc1);
39         if ((adcSampleCount % 2) == 1) // We just sampled the second of the pair.
40         {   // Time to save both in two sample element of packed sample Array
41             adcSample[adcElementCount] = tsPair.element;
42             adcElementCount++;
43         }
44         adcSampleCount++;
45         if (adcElementCount == MAX_ADC_ELEMENT_COUNT )
46         {
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
52             // Check the completion time,
53             // reported by serialCmdParser.c after c[1] cmd.
54             // ######## moved... xNowTime = xTaskGetTickCount();
55
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  ISR
62             // CMSIS wrapper for xSemaphoreGiveFromISR(...);
63             // The semaphore release/give here will trigger,
64             // certain actions in the for(;;) loop of task "checkADC"
65             osSemaphoreRelease(binarySP_ADC_DataReadyHandle);
66         }
```

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)
251 {
252   /* USER CODE BEGIN startCheckADC */
253
254   /* Infinite loop */
255   for(;;)
256   {
257     // Wait for data read from ADC (1 or more samples acquired.)
258     osSemaphoreWait(binarySP_ADC_DataReadyHandle, osWaitForever);
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:



Best regards,

Joe Kuss
jmkuss@arrl.net