EECS 690

Project 1 Report

Program Profiling on a TI Tiva C TI\_TM4C1294NCPDT using FreeRTOS

Written By

Ben Sokol

Kaiser Mittenburg

The University of Kansas

Electrical Engineering and Computer Science

September 18, 2018

Copyright © 2018 by Ben Sokol and Kaiser Mittenburg.

All rights reserved.

# Abstract

Modern programs are very complex and often run many tasks. For the developer, it can be difficult to determine how many resources will be used to execute any given block of code. This report discusses a simple program tracing technique to determine where the program is spending its time. Knowing which bits of memory are being used the most will outline a good place to start optimizing and will lead to more efficient programs. The example is executed using a TI\_TM4C1294NCPDT board running FreeRTOS. We found that our program with several small tasks spent its time in just a few areas of memory, which was expected.

# Revision History

The following table (*Table 2-1*) lists the revision history for this document.

**Table 2‑1 Revision History**

|  |  |  |
| --- | --- | --- |
| **Date** | **Revision** | **Description** |
| September 18, 2018 | 1.0 | Initial Release |

# Table of Contents

[1 Abstract 1](#_Toc524993336)

[2 Revision History 1](#_Toc524993337)

[3 Table of Contents 2](#_Toc524993338)

[4 List of Figures 2](#_Toc524993339)

[5 List of Tables 2](#_Toc524993340)

[6 Principles of Operation (POP) 3](#_Toc524993341)

[7 Data Structure Descriptions 4](#_Toc524993342)

[8 Function Descriptions 4](#_Toc524993343)

[9 Parameters 7](#_Toc524993344)

[The following table contains a list of parameters used in 7](#_Toc524993345)

[10 Testing 9](#_Toc524993346)

[11 Lessons Learned 9](#_Toc524993347)

[Appendix A – Program Source Code 10](#_Toc524993348)

# List of Figures

[**Figure 10‑1 Results Histogram** 9](#_Toc525002126)

# List of Tables

[**Table 2‑1 Revision History** 1](#_Toc524994793)

[**Table 7‑1 Data Structures** 4](#_Toc524994794)

[**Table 8‑1 Functions** 4](#_Toc524994795)

[**Table 8‑2 Referenced External Functions** 6](#_Toc524994796)

[**Table 9‑1 Parameters** 7](#_Toc524994797)

# 

# Principles of Operation (POP)

The project implements a task that repeatedly samples and records hits on the program counter (PC). Our Task\_ProgramTrace initializes a timer that will run periodically and call an interrupt service routine (ISR). The mainline of our task will then “spin” (do nothing but still loop) while the ISR obtains the current program counter and saves it. After one minute of sampling, a flag is switched and the ISR will spin, ceasing to collect more data. Now the mainline for Task\_ProgramTrace will take the collected data and output the results to a console. Once the results are output, the mainline will flip the flag back so the ISR will collect more data. This will continue until externally terminated.

The key instrument used in synchronizing the task mainline and the ISR is a binary semaphore. Effectively, both will continue to execute forever as the timer that calls the ISR is periodic with no end and the mainline is wrapped in a while ( 1 ) block. The mainline while loop will likely execute much faster than the timer takes to terminate a period, so as mentioned, the semaphore will be used for synchronization. This is done by immediately taking the semaphore in the mainline. Once taken, execute the logic. The while loop will be executed again, only this time, it will block on the call to take the semaphore because it has already been taken. Next, the timer will terminate a period and the ISR will be called. The ISR will call a function to obtain the current PC. It is worth noting that the “current” PC we are interested in is the PC that we have context switched from upon entering the interrupt - that PC is the memory address we *were* in, which effectively traces where the program is spending some time. It would be pointless to obtain the real PC, because we already know we are in an interrupt. It is likely possible that the PC can be obtained using standard C mechanisms, however, it is much easier to write a short assembly function to do the job. We know that at least 10 items of 4 bytes each are pushed onto the stack during a context switch and the PC is the 9th item. The stack pointer points to the bottom of the stack, so an offset of 32 bytes will get us to the PC. It is standard assembly convention to load return values into register 0. We then branch back from where we came from using the Link Register.

To keep things simple, we are not going to record every unique memory address that is obtained from the PC - that would be overkill as the discernable memory mappings have a range of memory, not a single location. We notice that the program will likely use less than 32 KiB, so we create 512 bins of 64 bytes each. After obtaining the PC we determine which bin that memory address falls into and increment that bin to denote a hit. Next, we need to check when to stop sampling. Right after the ISR timer was enabled, the current system tick was polled, and the stop system tick mark was calculated. The ISR will now check if that time has been reached, if it has, the control flag will be flipped – this is the mechanism that controls when the ISR collects and the mainline spins, and when the ISR spins and the mainline reports data. The last thing to be done in the ISR is “give” the semaphore back. We did not take the semaphore in the ISR but giving the semaphore back will allow the mainline that has been blocking on a call to take the semaphore, to successfully take it. After taking the semaphore, the mainline will check if the control flag has been flipped – if it has, then we will use a third-party tool, Report Data, to send our results to a console. After all the results have been reported, the bins are emptied, and the control flag is flipped back to allow data collection once again.

# Data Structure Descriptions

The following table (*Table 7-1*) is a list of all Data Structures used in this project

**Table 7‑1 Data Structures**

|  |  |  |
| --- | --- | --- |
| **Data Structure Name** | **Type** | **Data Structure Description** |
| **histogram\_array** | uint32[512] | This array is used to store data values collected from the Program Counter by the ISR. It is initialized to all 0’s by the functionzero\_histogram\_array() |

# Function Descriptions

The following table (*Table 8-1*) contains each function defined for this project. *Table 8-2* contains the referenced external functions for each function in *Table 8-1*

**Table 8‑1 Functions**

|  |  |
| --- | --- |
| **Function Name (Purpose)** | **Function Pseudo Code** |
| **Task\_ProgramTrace**  This function’s purpose work with Timer\_0\_A\_ISR to collect and report data about current program profiling via UART. This task watches for a flag to be flipped by the ISR signaling that data collection is complete. The flag being flipped causes this task to begin sending data to the queue to be reported. After sending data is complete, the flag is flipped again and the ISR resumes collecting data. | // INITIALIZE Timer\_0\_A\_Semaphore and SETUP Timer\_A  CALL vSemaphoreCreateBinary WITH Timer\_0\_A\_Semaphore  CALL SysCtlPeripheralEnable WITH SYSCTL\_PERIPH\_TIMER0  CALL IntRegister WITH INT\_TIMER0A AND Timer\_0\_A\_ISR  CALL TimerConfigure WITH TIMER0\_BASE AND the  bitwise-or combination of TIMER\_CFG\_SPLIT\_PAIR AND  TIMER\_CFG\_A\_PERIODIC  CALL TimerPrescaleSet WITH TIMER0\_BASE, TIMER\_A, AND  the PRE\_SCALE\_VALUE  CALL TimerLoadSet WITH TIMER0\_BASE, TIMER\_A, AND  LOAD\_VALUE  CALL TimerIntEnable WITH TIMER0\_BASE AND  TIMER\_TIMA\_TIMEOUT  CALL IntEnable WITH INT\_TIMER0A to enable Timer\_0\_A  interrupt  CALL TimerEnable WITH TIMER0\_BASE AND TIMER\_A to enable start timer  CALL ReportData\_SetOutputFormat WITH Excel\_CSV to use  Excel output format for ReportData  CALL zero\_histogram\_array to initialize  histogram\_array  SET start\_Sys\_Tick TO xPortSysTickCount  SET stop\_Sys\_Tick TO THE CALCULATION start\_Sys\_Tick  PLUS (REPORT\_FREQUENCY\_IN\_SECONDS MULTIPLIEDBY  ONE\_SECOND\_DELTA\_SYS\_TICK  // BEGIN taking data  WHILE TRUE  CALL xSemaphoreTake WITH Timer\_0\_A\_Semaphore AND  portMAX\_DELAY  IF current\_ISR\_Status EQUALS DONE\_COLLECTING  INCREMENT current\_Histogram\_Report  CALL UARTprintf WITH “Done With Report #” message  CALL report\_histogram\_data to send data to queue  CALL zero\_histogram\_array to zero array  SET start\_Sys\_Tick TO xPortSysTickCount  SET stop\_Sys\_Tick TO THE CALCULATION  start\_Sys\_Tick PLUS (REPORT\_FREQUENCY\_IN\_SECONDS  MULTIPLIEDBY ONE\_SECOND\_DELTA\_SYS\_TICK  SET current\_ISR\_Status TO COLLECTING  ENDIF  ENDWHILE |
| **Get\_Value\_From\_Stack**  (Assembly)  This function’s purpose is to obtain the current program counter from the stack | DECLARE Get\_Value\_From\_Stack as global  IN Get\_Value\_From\_Stack  LOAD VALUE FROM Stack Pointer, given an offset,  into Register  Branch back  END Get\_Value\_From\_Stack |
| **Timer\_0\_A\_ISR**  Interrupt Service Routine used to collect data from the program counter. | CALL TimerIntClear WITH TIMER0\_BASE and  TIMER\_TIMA\_TIMEOUT  IF current\_ISR\_Status EQUALTO COLLECTING  SET current\_PC TO RETURN VALUE FROM CALLING  Get\_Value\_From\_Stack WITH PC\_OFFSET  SET current\_PC TO THE FLOOR OF current\_PC DIVIDEDBY  64.0  IF current\_PC LESSTHAN SIZE\_OF\_HISTOGRAM\_ARRAY  INCREMENT the index current\_PC in histogram\_array  ELSE  IF current\_PC GREATERTHANOREQUALTO  SIZE\_OF\_HISTOGRAM\_ARRAY  CALL UARTprinf with error “to large” message  ELSEIF  CALL UARTprinf with error “to small” message  ENDIF    CALL UARTprinf to output current\_PC  ENDIF  IF xPortSysTickCount GREATERTHAN stop\_Sys\_Tick  SET current\_ISR\_Status TO DONE\_COLLECTING  ENDIF  ENDIF  CALL xSemaphoreGiveFromISR WITH Timer\_0\_A\_Semaphore  AND &xHigherPriorityTaskWoken |
| **report\_histogram\_data**  This function will add each index of the histogram\_array to a ReportData\_Item, then add each of those items to the ReportData\_Queue | FOR each i between 0 and 512  INIT a ReportData\_Item named item;  SET item TimeStamp as xPortSysTickCount  SET item ReportName as 42  SET item ReportValueType\_Flg as 0x0  SET item ReportValue\_0 as i  SET item ReportValue\_1 as index i in histogram\_array  SET item ReportValue\_2 as 0  SET item ReportValue\_3 as 0    CALL xQueueSend WITH ReportData\_Queue item\_ref AND 0  ENDFOR |
| **zero\_histogram\_array**  This function will set all indices in the histogram\_array to 0 | FOR each index in histogram\_array  SET histogram\_array at index to 0  ENDFOR |

**Table 8‑2 Referenced External Functions**

|  |  |
| --- | --- |
| **Function Name** | **Referenced External Functions** |
| **Task\_ProgramTrace** | IntEnable  IntRegister  ReportData\_SetOutputFormat(Excel\_CSV);  SysCtlPeripheralEnable  TimerConfigure  TimerEnable  TimerIntEnable  TimerLoadSet  TimerPrescaleSet  UARTprintf  vSemaphoreCreateBinary  xSemaphoreTake |
| **Get\_Value\_From\_Stack** | None |
| **Timer\_0\_A\_ISR** | TimerIntClear  UARTprintf  xSemaphoreGiveFromISR |
| **report\_histogram\_data** | xQueueSend() |
| **zero\_histogram\_array** | None |

# Parameters

# 

# The following table (*Table 9-1*) contains a list of parameters used in this project

**Table 9‑1 Parameters**

|  |  |  |  |
| --- | --- | --- | --- |
| **Parameter Name** | **Parameter Type** | **Parameter**  **Default Value** | **Parameter Description** |
| **PC\_OFFSET** | const uint32\_t | 32 | Program Counter Offset, (512 << 6) |
| **LOAD\_VALUE** | const uint32\_t | 50000 | Load value, chosen arbitrarily |
| **PRE\_SCALE\_VALUE** | const uint32\_t | 23 | Pre-scale value (must be < 256) |
| **SIZE\_OF\_HISTOGRAM\_ARRAY** | const uint32\_t | 512 | Contains size of the histogram array.  This is the number of possible bins data is collected in |
| **ONE\_SECOND\_DELTA\_SYS\_TICK** | const uint32\_t | 10000 | 1 second in systicks |
| **REPORT\_FREQUENCY\_IN\_SECONDS** | const uint32\_t | 60 | How frequently should the program report data. |
| **Timer\_0\_A\_Semaphore;** | xSemaphoreHandle | N/A | Semaphor for Timer\_0\_A |
| **xHigherPriorityTaskWoken** | portBASE\_TYPE | pdFALSE | Extracted from the ISR to keep stack items to a minimum |
| **Current\_PC** | uint32\_t | 0 | Contains the current Program Counter.  Initialized to 0 because it doesn’t matter initial value, as it is only used as a storage container to hold the value from the program counter. |
| **current\_ISR\_Status** | ISR\_STATUS\_t | COLLECTING | Status of our PC value. Controls when data is collected/reported.  COLLECTING is chosen by default to ISR starts collecting immediately. |
| **start\_Sys\_Tick** | uint32\_t | 0 | Variable to hold initial system tick in for each round of gathering data.  Initialized to 0 because it will be set in Task\_ProgramTrace before it is used. This ensures that as little time is not accounted for by program startup and overhead. |
| **stop\_Sys\_Tick** | uint32\_t | 0 | Variable to hold final system tick in for each round of gathering data.  Same as start\_Sys\_Tick |
| **current\_Histogram\_Report** | uint32\_t | 0 | Tracks how many histogram reports have been output  Initialized to 0 because at program start no reports have been made |

# Testing

After one minute of sampling we discovered that very few bins were even hit once. This is not a cause for concern, given that the program only has four tasks being scheduled, and a relatively low number of libraries and technologies being utilized.

**Figure 10‑1 Results Histogram**

As shown in *Figure 10-1*, bin 323 is by far the highest used block of memory. The data was zero indexed, so bin 323 refers to memory range from 20,672 bytes to 20,736 bytes. According to our map file, that memory range belongs to portasm.obj (.text). portasm.obj contains the assembly context switching routines. It makes sense that context switching is being hit a lot, we are context switching every time the ISR is called and every time the CPU executes a different task. Bin 321 and 322 are also in the range of the portasm.obj. Bins 42, 43, 45, 46, and 47 are the tasks.obj file which also makes sense being so large since everything that is running is propagated through a task. Bins 289 and 290 are driverLib.lib file which are not used very much.

# Lessons Learned

Our results may not be line by line specific, but it provides a good indication of where our time is spent. According to our results, we spend a lot of time context switching - this is not an uncommon resource gobbler. Now that we know this, it may be worth experimenting with parameters that affect context switching. This may also indicate a good reason to use a custom operating system - attributes such as the quantum (CPU execution time) given to each task before switching are changeable. There are many tradeoffs when dealing with context switching, and it may be difficult to pick a solution, however, knowing where to optimize is the first step.

# 

# Appendix A – Program Source Code

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149  150  151  152  153  154  155  156  157  158  159  160  161  162  163  164  165  166  167  168  169  170  171  172  173  174  175  176  177  178  179  180  181  182  183  184  185  186  187  188  189  190  191  192  193  194  195  196  197  198  199  200  201  202  203  204  205  206  207  208  209  210  211  212  213  214  215  216  217  218  219  220  221  222  223  224  225  226  227  228  229  230  231  232  233  234  235  236  237  238  239  240  241  242  243  244  245  246  247  248  249  250  251  252  253  254  255  256  257  258  259  260  261  262  263  264  265  266  267  268  269  270  271 | */\*\**  *\* @Filename: Task\_ProgramTrace.c*  *\* @Author: Kaiser Mittenburg and Ben Sokol*  *\* @Email: ben@bensokol.com*  *\* @Email: kaisermittenburg@gmail.com*  *\* @Created: August 30th, 2018 [1:35pm]*  *\* @Modified: September 17th, 2018 [6:45pm]*  *\* @Version: 1.0.0*  *\**  *\* @Description: Periodically traces current program memory location*  *\**  *\* Copyright (C) 2018 by Kaiser Mittenburg and Ben Sokol. All Rights Reserved.*  *\*/*  *#include "inc/hw\_ints.h"*  *#include "inc/hw\_memmap.h"*  *#include "inc/hw\_sysctl.h"*  *#include "inc/hw\_types.h"*  *#include "inc/hw\_uart.h"*  *#include <stdarg.h>*  *#include <stdbool.h>*  *#include <stddef.h>*  *#include <stdint.h>*  *#include <stdlib.h>*  *#include <math.h>*  *#include "Drivers/UARTStdio\_Initialization.h"*  *#include "Drivers/uartstdio.h"*  *#include "driverlib/gpio.h"*  *#include "driverlib/interrupt.h"*  *#include "driverlib/pin\_map.h"*  *#include "driverlib/sysctl.h"*  *#include "driverlib/timer.h"*  *#include "Tasks/Task\_ReportData.h"*  *#include "FreeRTOS.h"*  *#include "semphr.h"*  *#include "task.h"*  */\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**  *\* External variables*  *\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*  *// Access to current Sys Tick*  **extern** **volatile** **long** **int** xPortSysTickCount**;**  */\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**  *\* External functions declarations*  *\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*  *// Assembly function to get PC from the stack*  **extern** **uint32\_t** Get\_Value\_From\_Stack**(uint32\_t);**  */\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**  *\* Local task constant types*  *\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*  **typedef** **enum** ISR\_STATUS\_t **{**  COLLECTING**,** *// Should ISR collect data*  DONE\_COLLECTING *// ISR is done collecting data, reporting*  **}** ISR\_STATUS\_t**;**  */\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**  *\* Local task constant variables*  *\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*  *// Program Constants*  *// We operate at 120 MHz, which gives a period of 8.33 nS*  *// The equation 8.33nS \* K \* M = 10mS*  *// The LOAD\_VALUE (M) must be < 64k, 50,000 chosen arbitrarily*  *// The PRE\_SCALE\_VALUE (K) must be < 256. Solving, K = 24*  *// Since K is zero indexed, K = 23*  *// We are only interested in memory <= 32KiB which is 2^15*  **const** **uint32\_t** PC\_OFFSET **=** **32;**  **const** **uint32\_t** LOAD\_VALUE **=** **50000;**  **const** **uint32\_t** PRE\_SCALE\_VALUE **=** **23;**  **const** **uint32\_t** HISTOGRAM\_ARRAY\_SIZE **=** **512;** *// (512 << 6) == 32KiB*  **const** **uint32\_t** ONE\_SECOND\_DELTA\_SYS\_TICK **=** **10000;**  **const** **uint32\_t** REPORT\_FREQUENCY\_IN\_SECONDS **=** **60;**  */\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**  *\* Local task variables*  *\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*  *// The semaphore*  xSemaphoreHandle Timer\_0\_A\_Semaphore**;**  *// Data array*  **uint32\_t** histogram\_array**[512];**  *// Extracted from the ISR to keep stack items to a minimum*  portBASE\_TYPE xHigherPriorityTaskWoken **=** pdFALSE**;**  *// The current memory address obtained from the PC*  **uint32\_t** current\_PC **=** **0;**  *// Status of our PC value. Controls when data is collected/reported*  ISR\_STATUS\_t current\_ISR\_Status **=** COLLECTING**;**  *// Sys\_Tick when ISR starts collecting*  **uint32\_t** start\_Sys\_Tick **=** **0;**  *// Sys\_Tick when ISR needs to stop collecting*  **uint32\_t** stop\_Sys\_Tick **=** **0;**  *// How many reports have been output*  **uint32\_t** current\_Histogram\_Report **=** **0;**  */\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**  *\* Local task function declarations*  *\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*  **extern** **void** Timer\_0\_A\_ISR**();**  **extern** **void** Task\_ProgramTrace**(void\*** pvParameters**);**  **extern** **void** report\_histogram\_data**();**  **extern** **void** zero\_histogram\_array**();**  */\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**  *\* Local task function definitions*  *\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*  */\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**  *\* Function Name: Timer\_0\_A\_ISR*  *\* Description: Interrupt Service Routine used to profile tasks*  *\* Parameters: N/A*  *\* Return: void*  *\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*  **extern** **void** Timer\_0\_A\_ISR**()** **{**  TimerIntClear**(**TIMER0\_BASE**,** TIMER\_TIMA\_TIMEOUT**);**  **if** **(**current\_ISR\_Status **==** COLLECTING**)** **{**  *// Get the value from the PC*  current\_PC **=** Get\_Value\_From\_Stack**(**PC\_OFFSET**);**  current\_PC **=** floor**(** current\_PC **/** **64.0** **);**  *// Validate Current\_PC value is within size of array and store*  **if** **(**current\_PC **<** HISTOGRAM\_ARRAY\_SIZE**)** **{**  *// Increment Bin for Current\_PC*  histogram\_array**[**current\_PC**]++;**  **}**  **else** **{**  *// Current\_PC is out of range.*  *// In theory should never enter this else statement*  **if** **(**current\_PC **>=** HISTOGRAM\_ARRAY\_SIZE**)** **{**  UARTprintf**(**"ERROR: (Current\_PC / 64) >= %i"**,** HISTOGRAM\_ARRAY\_SIZE**);**  **}**  **else** **{**  UARTprintf**(**"ERROR: (Current\_PC / 64) < 0"**);**  **}**  UARTprintf**(**" (Current\_PC = %u)\n"**);**  **}**  **if** **(**xPortSysTickCount **>** stop\_Sys\_Tick**)** **{**  current\_ISR\_Status **=** DONE\_COLLECTING**;**  **}**  **}**  *// "Give" the Timer\_0\_A\_Semaphore*  xSemaphoreGiveFromISR**(**Timer\_0\_A\_Semaphore**,** **&**xHigherPriorityTaskWoken**);**  **}**  */\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**  *\* Function Name: Task\_ProgramTrace*  *\* Description: Task used to initialize Timer\_0\_A\_ISR*  *\* Parameters: void\* pvParameters;*  *\* Return: void*  *\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*  **extern** **void** Task\_ProgramTrace**(void\*** pvParameters**)** **{**  *//Initialize Semaphore and setup Timer*  vSemaphoreCreateBinary**(**Timer\_0\_A\_Semaphore**);**  SysCtlPeripheralEnable**(**SYSCTL\_PERIPH\_TIMER0**);**  IntRegister**(**INT\_TIMER0A**,** Timer\_0\_A\_ISR**);**  TimerConfigure**(**TIMER0\_BASE**,** TIMER\_CFG\_SPLIT\_PAIR **|** TIMER\_CFG\_A\_PERIODIC**);**  TimerPrescaleSet**(**TIMER0\_BASE**,** TIMER\_A**,** PRE\_SCALE\_VALUE**);**  TimerLoadSet**(**TIMER0\_BASE**,** TIMER\_A**,** LOAD\_VALUE**);**  TimerIntEnable**(**TIMER0\_BASE**,** TIMER\_TIMA\_TIMEOUT**);**  *// Enable Timer\_0\_A interrupt in NVIC*  IntEnable**(**INT\_TIMER0A**);**  *// Enable (Start) Timer*  TimerEnable**(**TIMER0\_BASE**,** TIMER\_A**);**  *// Set data report to Excel format*  ReportData\_SetOutputFormat**(**Excel\_CSV**);**  *// Init the data array*  zero\_histogram\_array**();**  *// Set start time based on current systick, stop time = 1 minute later.*  start\_Sys\_Tick **=** xPortSysTickCount**;**  stop\_Sys\_Tick **=** start\_Sys\_Tick  **+ (**REPORT\_FREQUENCY\_IN\_SECONDS **\*** ONE\_SECOND\_DELTA\_SYS\_TICK**);**  *// Add values to the histogram when appropriate*  **while** **(1)** **{**  xSemaphoreTake**(**Timer\_0\_A\_Semaphore**,** portMAX\_DELAY**);**  **if** **(**current\_ISR\_Status **==** DONE\_COLLECTING**)** **{**  current\_Histogram\_Report**++;**  UARTprintf**(**"DONE COLLECTING (%u)\n"**,** current\_Histogram\_Report**);**  report\_histogram\_data**();**  *// Zero array to make sure overflow doesnt happen*  zero\_histogram\_array**();**  *// Reset time to start collecting again for 1 minute*  start\_Sys\_Tick **=** xPortSysTickCount**;**  stop\_Sys\_Tick **=** start\_Sys\_Tick  **+** **(**REPORT\_FREQUENCY\_IN\_SECONDS **\*** ONE\_SECOND\_DELTA\_SYS\_TICK**);**  *// Set flag current\_ISR\_Status to start collecting again*  current\_ISR\_Status **=** COLLECTING**;**  **}**  **}**  **}**  */\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**  *\* Function Name: report\_histogram\_data*  *\* Description: Function used to send data to ReportData\_Queue*  *\* Parameters: N/A*  *\* Return: void*  *\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*  **extern** **void** report\_histogram\_data**()** **{**  **uint32\_t** i **=** **0;**  **for** **(**i **=** **0;** i **<** **512;** **++**i**)** **{**  ReportData\_Item item**;**  item**.**TimeStamp **=** xPortSysTickCount**;**  item**.**ReportName **=** **42;**  item**.**ReportValueType\_Flg **=** **0x0;**  item**.**ReportValue\_0 **=** i**;**  item**.**ReportValue\_1 **=** histogram\_array**[**i**];**  item**.**ReportValue\_2 **=** **0;**  item**.**ReportValue\_3 **=** **0;**  *// This sends copy of data*  xQueueSend**(**ReportData\_Queue**,** **&**item**,** **0);**  **}**  **}**  */\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**  *\* Function Name: zero\_histogram\_array*  *\* Description: Function used to initialize histogram array to all zeros*  *\* Parameters: N/A*  *\* Return: void*  *\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*  **extern** **void** zero\_histogram\_array**()** **{**  **uint32\_t** i **=** **0;**  **for** **(**i **=** **0;** i **<** **512;** **++**i**)** **{**  histogram\_array**[**i**]** **=** **0;**  **}**  **}** |