HW4 Report

謝翔丞, 109550025

*Abstract*—以FreeRTOS搭配原先已掛載cache版本之Aquila，同時以ILA為輔，嘗試對於RTOS執行過程之context-switch以及synchronization進行行為分析與計算，發現Context-Switch次數隨Time Slice上升而下降，同時超過99.9%的比例在處理asynchronous的case，並且利用atomic instruction嘗試撰寫mutex並進行測試功能正常。. (*Abstract*)

Keywords— Aquila, FreeRTOS, mutex, atomic instruction (key words)

# Introduction

本次作業是建立在HW3的基礎上，也就是在掛載cache之後的aquila上執行FreeRTOS，透過分析RTOS的thread management並且藉由ILA輔助計算出context-switch以及synchronization的overhead， 同時以atomic instruction來實作出mutex。.

# Analysis Of FreeRTOS

## Task

本次旨在分析FreeRTOS對於thread的管理.，因此從Task.c著手我認為是再適合不過的選擇。在此處，task可以被視為一個thread，而這個thread的排程則是藉由TCB來負責掌控，我們可以先從幾個變數與函式看起。

1. xStateListItem, 這個變數記錄了task的三個相關state資訊，包含ready, blocked 或是Suspended。
2. uxPriority,用來記錄每個task的priority，值得注意的是FreeRTOS裏頭將0視為最低的priority逐而遞增，恰好與我們熟知的Nice Value判斷順序顛倒。
3. pxReadyTaskList，這個array給定ConfigMAX\_PRIORITIES的長度，目的是儲存特定priority並且已經ready的task。
4. xDelayList1&2,其一如其名是為了計算Delay task，而另一個的用意則是為了應付前者overflow時的情況而存在。
5. xTickCount,這個變數在此次分析中頗為重要，是一個初始為0，用來計算程式執行過程發生幾次tick interrupt的角色。

## Main

接下來會從Main function逐步trace幾個重要的流程，在Main 中依序進到兩個主要函式，而第二項也是本篇著墨的重點處:

1. xTaskCreate
   1. 這個函式判斷一旦putSTACK\_GROWTH >= 0 則要先使用pvPortmalloc 去allocate TCB 再allocate stack ，反之則是與前述相反的操作。
   2. 接著再依序進入prvInitialiseNewTask以及prvAddTaskList，於此同時pxCurrentTCB也會根據自身使否為Null來選擇要更新成pxTCB或是選擇 “優先度較高或是較為新者”，之所以這邊會補充"較為新者”是因為，在FreeRTOS的分級裡，相同優先度者以較新加入者為優先。
2. vTaskSchedular
   1. 這個函式在一開始會建立一個優先度為0的IdleTask，並且進行一些基本的設置，例如將xSchedularReady設為True、xTickCount設為0，設置xSchedularRunning,用來表示Schedular已經開始運作。
   2. 呼叫xStartSchedular函式，內部會再尋訪兩個xPortStartFirstTask以及vPortSetupTimeInterrupt，後者如其名負責設置riscV csr的mtime與mtimecmp，這裡我們可以一併看到aquila中CLINT.v 122行的 ” assign tmr\_irq\_o = (mtime >= mtimecmp) & (| mtimecmp); “ 一旦mtime>=mtimecmp時會觸發interrupt。
   3. 我們再進一步向下深入，當發生interrupt要執行context-switch時，xPortFirstTask會先跳到freertos\_risc\_v\_trap\_handler進行多次的sw，接著更新currentTCB，隨後將csr mvtvecv寫入; 此處可一併參照Aquila的programcounter.v 和csr\_file.v，programcounter拿到的PC\_handler\_i就是csr\_file.v中透過讀取csr mtvec而提供的Output，如此完成兩邊的r / w。

Figure .Context Switch times to Time Slice

Figure .Context Switch Cycles to Time Slice

# Context-Switch

經過前面的步驟，我們接著開始探討task management的重點，Context-Switch。

Context-Switch產生的流程可以用Figure 5.Context-Switch Workflow來表示，首先經由freertos\_risc\_v\_trap\_handler，不管任何狀態都會先進入這個函示，接著才會到test\_if\_asynchronous進行分流，以下分成兩條路:

## Asynchronous

Asynchronous這條路trace下去會是一條相對較深較長的行程，判斷為Asynchronous後會走到handle\_asynchronous並判斷test\_if\_external\_interrupt，接著會分為yes / no 兩支

1. Yes :
   1. 首先進入as\_yet\_unhandled，直到a0與t1 register value相同時，便繼續load xTSRStackTop並往後走到vExternalISR，經過一個指令便進行processed\_source。
2. No :
   1. 第一步透過Load\_from\_pullmachine取得mtime並且計算出mtimecmp，經過load\_pull\_next\_time、uxTimerIncrementsForOneTick並load xISRStack以及xTaskIncrementTick，在這裡會更新一些關於tick的變數，如xTickCount等等，同時此函式的回傳直式另個韓式，也就是xSwitchRequired，會在此處判斷成式是否需要進行Context-Switch。
   2. 承接上述，若需要進行Context-Switch會先到vTaskSwitchContext，這裡會先將Schedular暫停、透過taskSELECT\_HIGHEST\_PRIORITY\_TASK找到下一個具有較高prioroty的task並TCB存到pxCurrentTCB。
   3. 之後歸入processed\_source，將pxCurrentTCB load 進入register內就完成一次的interrupt了。

## Synchronous

如果是Synchronous會先進入handle\_synchronous， 接著test\_if\_enviroment\_call判斷是否是enviroment\_call ，下一步就直接進行processed\_source。

Figure 3.Cycles between Asynchronous / Synchronous

Figure .Ratio of Different State

# Mutex

這次關於Mutex會分為兩個部份來進行，首先是trace FreeRTOS內部建立mutex的流程，其次是利用Atomic Instructions實作Mutex:

## Mutex in FreeRTOS

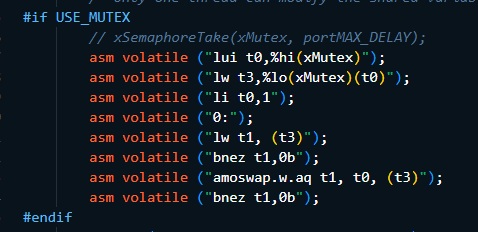
FreeRTOS中建立mutex的流程相當遠長，先調用xSemaphore函示，而此函式背後會呼叫xQueueGenericCreate利用建立Queue的方式先宣告一個基本的mutex樣貌，接著使用prcInitialiseMutex將前步驟的queue調整為針對mutex的基本設置，例如將pcHead設為Null，因為mutex本身不需要儲存資料不同於一般的queue，並且將mutext的task也設為Null，再呼叫xQueueGenericSend，至此就完成一個FreeRTOS內Mutex的建構。

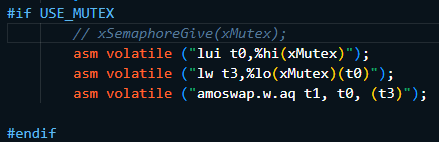
## Mutex Implementing Using Atomic Instructions

本次在利用Atomic Instruction實作Mutex也花費我許多時間，首先是尋找要在何處撰寫程式碼，其次是在撰寫完程式碼以及成功編譯出elf檔案後，放到板子上執行竟然卡死在幾個lw、amoswap.w.aq與bnez指令中陷入無限循環，為此又花了好些時間處理這個問題，很可惜最後並未如願解決。

#### 關於我實作手法，此次我是以老師講義提供的範例assembly code取代原本rtos\_test.c中，task1\_handler與task2\_handler的xSemaphoreTake以及xSemaphoreGive，並將早前透過xSemaphoreCreateMutex函式宣告取得的xMutex作為lock\_addr傳入組語進行訊算。

#### 實作程式碼





#### 前述提及本次陷入deadlock的指令起自 “lw t1,(t3)”，因為t1為0因此可以往後進行，卻在遇到amoswap之後，導致更新過的t1不為0，表示此時的lock已經被占用住因而跳回lw重新等待lock被釋放出來。這部份很可惜的是雖然我已經使用ILA搭配omjdump逐步分析我的程式碼執行過程，卻未能在時間內完成足夠的bug分析進而修正我的atomic instruction mutex。

# 心得

這次的作業對我個人而言算是本學期難度最高的，花上許多時間在trace FreeRTOS整份code並做整理，另外此次作業讓我收穫最多的是在一步步尋訪code的過程，如丙、，真正體會到軟體和硬體是如何整合在一起的，這樣的過程對我而言十分有趣，也很感謝老師規劃這樣的課程內容!

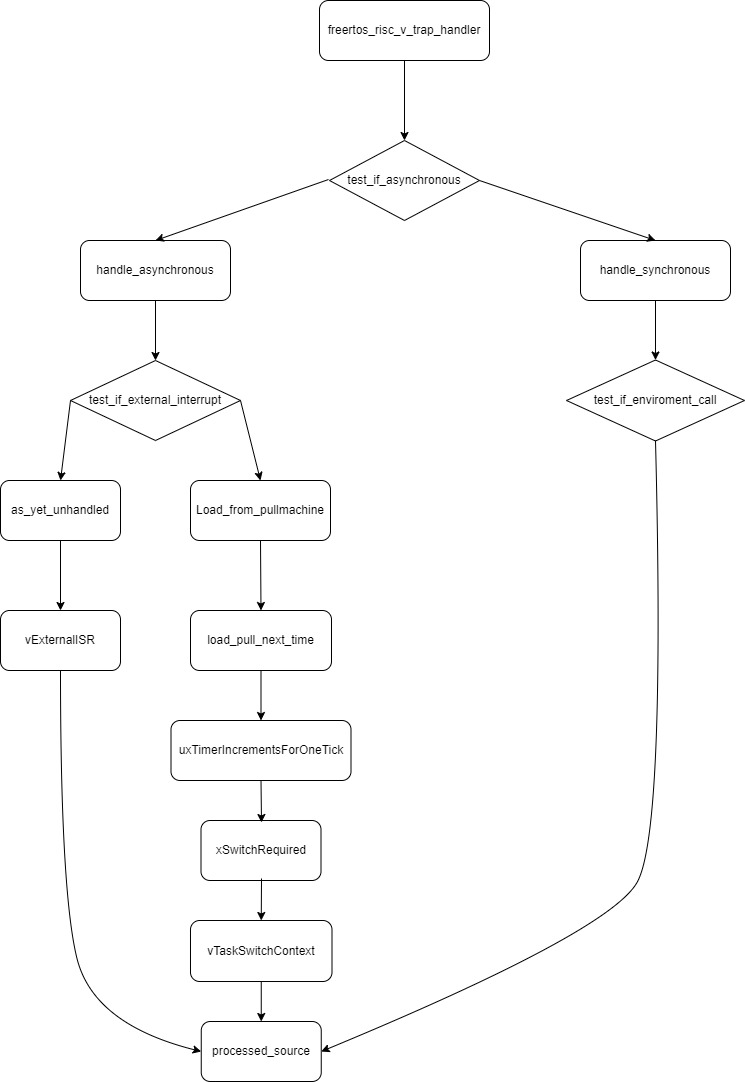


Figure .Context-Switch Workflow