行政院原子能委員會

委託研究計畫研究報告

核能儀控系統軟體模組安全功能驗證研究

On Verifying the Safety Functions of Software Modules in Computerized Nuclear Instrumentation and Control Systems

計畫編號：1012001INER013

受委託機關(構)：國立臺灣大學

計畫主持人：蔡益坤

聯絡電話：(02) 3366-1189

E-mail address：tsay@im.ntu.edu.tw

核研所聯絡人員：游原昌

報告日期：中華民國 101 年 11 月 30 日

目　錄

中文摘要

在上一年度的研究計畫中，我們整理了國內外以正規方法發展安全軟體之應用實績與發展工具，以及更基礎的正規方法應用於安全軟體驗證之理論與技術，進而探討了正規方法應用於國內自主核能儀控系統軟體模組驗證之可行性。

本年度，我們更深入探討正規驗證方法的應用，除了持續觀察整理相關正規驗證方法與工具的發展外，針對兩方面的課題做進一步的研究：一、深入掌握模型檢測與演繹式兩大類正規驗證工具互補的必要性，理出更有系統的整合運用方式；二、在排程分析模型與即時多執行緒程式語意之關聯的確立尋求更嚴謹且精準的方法。

有鑑於模型檢測與演繹式驗證工具互補之必要，我們持續觀察探討 SPIN 及 VeriFast 的功能及應用。在第一季期中報告「整合應用模型檢測與演繹式驗證方法與工具」中，我們藉由範例介紹一套 SPIN 的輔助工具 Modex。該輔助工具可將 C 程式碼轉換為 SPIN 模型，更進一步提升驗證自動化程度並避免人工轉換之錯誤。我們同時也藉由範例更深入解說 VeriFast 這套演繹式程式驗證工具的使用方式。

另一方面，多工即時系統排程分析所需的工作負載模型應依據實際即時多執行緒程式的語意，包含執行緒的編碼及優先權的

設定等。即便有了作業系統排程行為模型，確認工作負載模型是否真實反映執行緒的編碼及優先權的設定仍有一定的難度。在第二季期中報告「如何確立排程分析模型與即時多執行緒程式語意關聯」中，我們應用外顯時間的描述法，提出一套排程相關精準語意模型之建構與工作負載模型之萃取的框架。

在這份期末報告暨第四季期中報告「正規方法應用於核能儀控系統軟體模組驗證之實務」中，我們以一個更貼近實務應用情境的系統為例，說明如何從即時多執行緒程式原始碼中，萃取出工作負載模型，以作為更精確之排程分析的依據。

關鍵詞：模型檢測、Modex、程式證明、即時程式、排程分析、SPIN、時間分析、VeriFast、工作負載模型

Abstract

In the previous project, we reviewed the applications of formal methods and related development tools in safety-critical software as well as the underlying theory and technology. The main purpose was to assess the feasibility of applying formal methods in verifying Taiwan's probable own computerized nuclear instrumentation and control systems.

For this year, we strive to gain a deeper understanding of the application of formal methods. Aside from watching the research literature on the development of relevant formal methods and tools, we will investigate two particular issues: one is the necessity for model checking and deductive verification tools to complement each other. We hope to obtain a more systematic way of integrating the two kinds of tools. The other issue concerns the link between the task model in scheduling and the semantic model of a real-time multithreaded program. We hope to find a more rigorous and precise way of establishing their correct correspondence. Considering the time and manpower we have for this project, we shall focus more on the integration of existing methods and tools.

In view of the necessity of combining model checking and

deductive verification tools, we continue to observe and study the functions and applications of SPIN and VeriFast. We reviewed in the first-quarter midterm report a companion tool of SPIN called Modex, which can be used to translate a C program into a SPIN model. This further increases the level of automation and avoids mistakes in manual translation. We also reviewed via examples further details in the use of the program verifier VeriFast.

On the other hand, the workload model needed for the schedulability analysis of a multitasking real-time system relies on precise semantic modeling of the corresponding multithreaded real-time program. In the second-quarter midterm report, we proposed a framework based on the explicit-time approach for such semantic modeling, enabling extraction of an abstract workload model from a multithreaded real-time program. In this fourth-quarter midterm and final report, we use a simple yet realistic example to illustrate how the framework may be applied to extract a workload model as the basis of more precise timing analysis.

Keywords: Model Checking, Modex, Program Verification, Real-Time Programs, Schedulability Analysis, SPIN, Timing Analysis, VeriFast, Workload Models.

壹、 計畫緣起與目的

研究文獻與工業標準的演進顯示正規方法（formal methods）在電腦化系統的安全功能認證上逐漸受到重視。雖然測試與模擬目前仍是安全功能認證所仰賴的主要驗證手段，但隨著正規方法理論發展的突破以及相關工具實用性的提高，在航太、汽車、核能與醫療器材等安全度要求較高的工業或電腦軟硬體設計等高價值的工業，有愈來愈多的歐美甚至亞洲廠商積極地研發或引進基於正規方法的系統開發與驗證工具。正規方法的運用讓這些廠商得以縮短產品開發時程、提高產品品質、進而增加獲利或提升公司形象。即使未全程使用正規方法，以嚴謹的數學及邏輯方法與工具驗證軟體程式（即正規驗證）已被視為確保日趨複雜之軟體安全功能正確無誤的最根本且有效之道（參考文獻 10 [Hoare 2003]）。既然正規方法於電腦化系統的安全功能認證有如此根本且策略的重要性，我們認為如欲自主開發高安全性之電腦化系統甚至提升台灣整體的電腦軟硬體技術水準，持續努力投入正規方法相關技術的研發與應用實有其必要與價值。

在上一年度的研究計畫中，我們以「核能儀控系統電腦化發展之安全功能認證」為目標，整理了國內外以正規方法發展安全軟體之應用實績與發展工具，以及更基礎的正規方法應用於安全軟體驗證之理論與技術，進而探討了正規方法應用於國內自主核

能儀控系統軟體模組驗證之可行性。我們以一簡化但與核能儀控系統功能相近之化學反應爐溫度控制器程式的驗證為例，檢視運用現有正規方法與工具的可行性。該控制器程式範例具備即時與多執行緒等挑戰驗證技術的基本要素，具相當之代表性。就我們所考慮的代表範例而言，現有的正規驗證方法與工具已能將包括基本程式分析、功能正確性驗證、以及執行時間與排程分析等驗證工作的大部分過程自動化，可顯著節省通過認證所需的人力與時間。正規驗證方法與工具仍在持續進步當中，未來所能發揮的效益應當更為顯著。我們進行範例程式驗證的步驟以及在各步驟中如何使用適切的方法與工具，將是未來從事類似驗證工作的重要參考。

在驗證代表範例與探索相關工具的過程中，有兩方面的發現我們認為特別值得注意：

模型檢測與演繹式驗證工具之互補：多執行緒程式功能正確性的模型檢測有相當的難度，加上即時性的需求條件後，驗證的難度更高。不論是動態的產生多執行緒，或是固定多執行緒的數量，對使用模型檢測方法來驗證的難度將是一大挑戰。學研界目前仍持續地努力研發更節省記憶空間、更有效率的方法與工具。此外，傳統上模型檢測方法大多針對系統的設計階段，較少考慮複雜的程式語言要素，因此在運用到實際程式碼時，往往需要人

工來做程式與模型之間的轉換。相對的，演繹式的驗證方法有其廣用性，也較能因應複雜的程式語言要素，但可惜並非全自動化。輔以模型檢測自動化的便利性是演繹式方法能否被實務界接受的關鍵，兩者如何最適切地互補是重要課題。

排程分析模型與即時多執行緒程式語意之關聯：排程分析所需的任務模型應依據實際即時多執行緒程式的語意，包含執行緒的編碼及優先權的設定等，而即時多執行緒程式之語意又需依賴即時作業系統排程等行為之語意模型。然而，絕大多數即時作業系統缺乏排程等行為之精準模型。這也是造成目前以靜態分析為主要手段的執行時間估算方法與工具，無法掌握特定即時作業系統排程行為的主因。為即時作業系統排程等行為建立精準語意模型並驗證其正確性，以做為程式執行時間分析之依據，應是根本解決之道。但即便有了作業系統排程行為模型，確認任務模型是否真實反映執行緒的編碼及優先權的設定仍有一定的難度。這部分研究文獻相當缺乏，因此排程相關精準語意模型之建構與驗證如何從事亦是重要課題。

本年度，我們更深入探討正規驗證方法的應用，除了持續觀察整理相關正規驗證方法與工具的發展外，將針對上述兩方面的課題做進一步的研究。我們希望能深入掌握模型檢測與演繹式兩大類正規驗證工具互補的必要性，理出更有系統的整合運用方

式，也希望在排程分析模型與即時多執行緒程式語意之關聯的確立尋求更嚴謹且精準的方法。基於時間與人力的考量，本研究將較著重在既有方法與工具之整合，必要時可能提出新理論或方法；至於自主驗證工具之研發則非本研究之範疇，未來如有需求宜另案規劃。

核能儀控系統或類似安全攸關電腦化系統之軟體模組安全功能的正規驗證工作主要包括（1）基本靜態程式分析、（2）功能正確性驗證、以及（3）執行時間與排程分析等。這些驗證工作各自針對問題的不同面向，所需的方法與工具也不相同。以下我們僅就功能正確性驗證方面，說明本研究計畫現階段的進展。

多執行緒程式功能正確性的驗證可使用模型檢測工具或演繹式程式驗證工具。一般來說，模型檢測自動化程度高，但多執行緒程式的模型檢測有相當的難度，學研界目前仍持續地努力研發更節省記憶空間、更有效率的方法與工具。此外，傳統上模型檢測方法大多針對系統的設計階段，較少考慮複雜的程式語言要素，因此在運用到實際程式碼時，往往需要人工來做程式與模型之間的轉換。相對的，演繹式的驗證方法有其廣用性，也較能因應複雜的程式語言要素，但可惜並非全自動化。同時使用兩種工具，各取其長，仍有其必要。

有鑑於此，我們持續觀察探討模型檢測工具 SPIN（參考文獻

22 [SPIN]）及演繹式程式驗證工具 VeriFast（參考文獻 26 [VeriFast]）的功能及應用。在**第一季期中報告「整合應用模型檢測與演繹式驗證方法與工具」**中，我們藉由範例介紹一套 SPIN 的輔助工具 Modex（參考文獻 17 [Modex]）。該輔助工具可將 C 程式碼轉換為 SPIN 模型，更進一步提升自動化程度並避免人工轉換之錯誤。我們同時也藉由範例更深入解說 VeriFast 的使用方式。

另一方面，多工即時系統排程分析所需的工作負載模型應依據實際即時多執行緒程式的語意，包含執行緒的編碼及優先權的設定等，而即時多執行緒程式之語意又需依賴即時作業系統排程等行為之語意模型。然而，絕大多數即時作業系統缺乏排程等行為之精準模型。這也是造成目前以靜態分析為主要手段的執行時間估算方法與工具，無法掌握特定即時作業系統排程行為的主因。為即時作業系統排程等行為建立精準語意模型並驗證其正確性，以做為程式執行時間分析之依據，應是根本解決之道。但即便有了作業系統排程行為模型，確認任務模型是否真實反映執行緒的編碼及優先權的設定仍有一定的難度。這部分研究文獻相當缺乏，因此排程相關精準語意模型之建構與驗證如何從事亦是重要課題。在**第二季期中報告「如何確立排程分析模型與即時多執行緒程式語意關聯」**中，我們應用外顯時間的描述法（參考文獻 15 [Lamport 2005]），提出一套排程相關精準語意模型之建構與驗

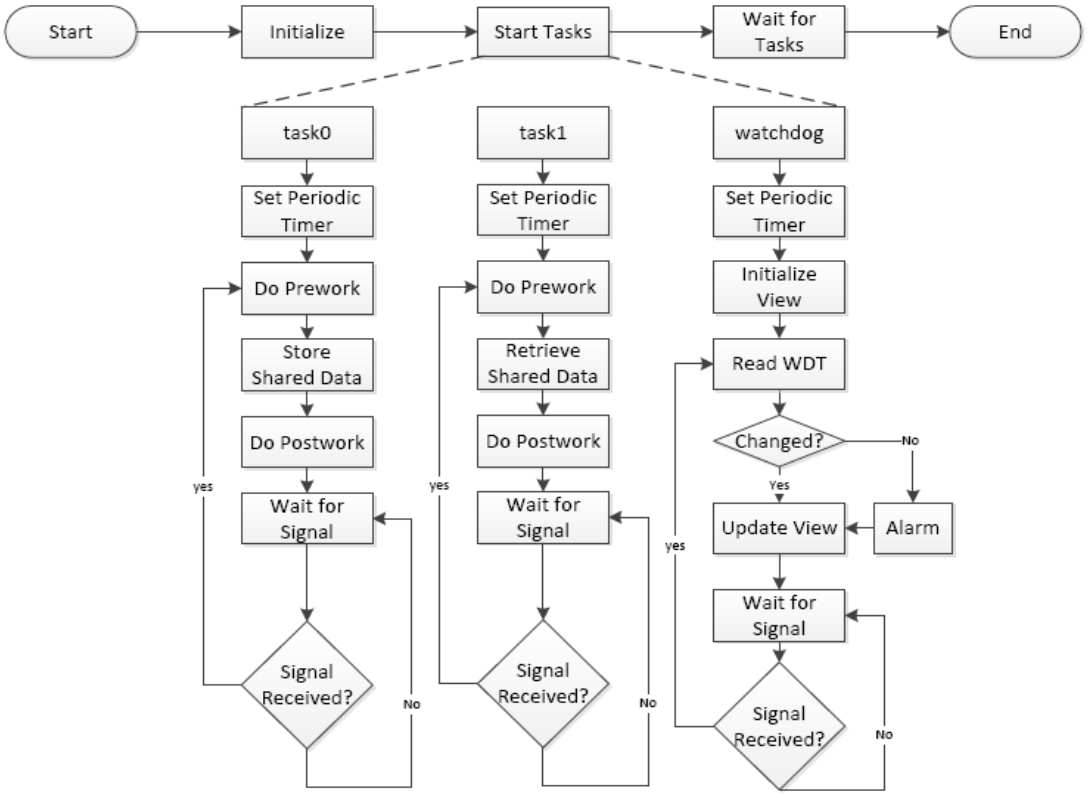
證的框架。

在這份**期末報告暨第四季期中報告「正規方法應用於核能儀控系統軟體模組驗證之實務」**中，我們以一個更貼近實務應用情境的系統為例，說明如何從即時多執行緒程式原始碼中，萃取出工作負載模型，以作為更精確之排程分析的依據。

貳、 研究方法與過程

一、 實務系統模型建立

我們以一個簡單但貼近實務應用情境的系統為例，探討如何從即時多執行緒程式原始碼中，萃取出工作負載模型。這個實例系統包含兩個週期性任務執行緒及一個監督執行緒，其工作流程如下圖所示（請參考完整報告中之程式碼）。監督執行緒觀察任務執行緒是否週期性地有所動作，確保無因軟體之錯誤而造成程式提前終止或停滯不前。



兩個週期性任務執行緒的內容各設計為三個部分，分別是前置作業、讀或寫關鍵性資料與後製作業。而取得關鍵性資料，也就是進入 critical section，採用的是 Peterson 演算法來實作 mutual exclusion，其中讀或寫的內容，會更改多維陣列的值與讀取該多維陣列的值，更改的方式會不斷的使陣列內容累加，如 111 → 222 → 333…，而讀取的另一執行緒則是印出陣列的內容。如此，若違反了 mutual exclusion 便能夠由印出的陣列內容有異如 544，而輕易的察覺。另外的前置與後置作業則分別呼叫耗時運算的 unit 函式來模擬一般程式執行的運算。

在週期性控制的方面，我們採用 periodic 的計時器來使執

行緒能夠週期性的執行。主要的方式是在執行緒一週期執行完畢後，等待下一週期開始的信號。

二、 實務系統模擬程式

(一)主程式

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <signal.h>
#include <pthread.h>
#include <sched.h>

(lines deleted)

int main(int argc, char *argv[])
{
   pthread_t t0, t1, wd;
   pthread_attr_t attr;
   struct sched_param param;
   int ret;
   pthread_attr_init(&attr);

   sigset_t alarm_sig;
   int i;

   /* Block all real time signals so they can be used for the
timers.
      Note: this has to be done in main() before any threads are
created
      so they all inherit the same mask. Doing it later is subject
to
```

```
        race conditions */
    sigemptyset (&alarm_sig);
    for (i = SIGRTMIN; i <= SIGRTMAX; i++)
       sigaddset (&alarm_sig, i);
    sigprocmask (SIG_BLOCK, &alarm_sig, NULL);

    param.sched_priority = T0_PRIORITY;
    pthread_attr_setschedparam(&attr, &param);
    ret = pthread_create(&t0, &attr, watchdog, NULL);
    param.sched_priority = T1_PRIORITY;
    pthread_attr_setschedparam(&attr, &param);
    ret = pthread_create(&t1, &attr, task1, NULL);
    param.sched_priority = WD_PRIORITY;
    pthread_attr_setschedparam(&attr, &param);
    ret = pthread_create(&wd, &attr, watchdog, NULL);

    pthread_join(t0, NULL);
    pthread_join(t1, NULL);
    pthread_join(wd, NULL);
    return 0;
}
```

(二)工作執行緒

```
#define T0_PRIORITY 7
#define T1_PRIORITY 7
#define WD_PRIORITY 7
#define T0_PERIOD 10 // seconds
#define T1_PERIOD 10 // seconds
#define T0_PREWORK 6 // units
#define T0_CS 4 // units
#define T0_POSTWORK 5 // units
#define T1_PREWORK 3 // units
#define T1_CS 5 // units
#define T1_POSTWORK 7 //units

int flag[2] = {0, 0};
int turn;
int critical_array[3] = {0,0,0};
int wdt[2] = {0, 0};

struct periodic_info
{
   int sig;
   timer_t timer_id;
   sigset_t alarm_sig;
   int wakeups_missed;
};
```

```
void unit(int times)
{
   int i, j;
   for (j = 0; j < times; j++)
      for (i = 0; i < 100000000; i++)
         ;
}

void* watchdog()
{
   struct periodic_info info;
   int pre1[2] = {2,2}, pre2[2] = {3,3}, now[2];

   make_periodic(1000000 * T0_PERIOD, &info);

   while (1) {
      now[0] = wdt[0];
      now[1] = wdt[1];
      if(pre1[0] == now[0])
         printf("t0 error\n");
      else printf("t0 is running\n");
      if(pre1[1] == now[1])
         printf("t1 error\n");
      else printf("t1 is running\n");
      pre1[0] = pre2[0];
      pre1[1] = pre2[1];
      pre2[0] = now[0];
      pre2[1] = now[1];
```

```
        wait_period(&info);
    }
}
void* task0()
{
    struct periodic_info info;
    int c = 0, i, j, x;

    make_periodic(1000000 * T0_PERIOD, &info);

    while (1) {
        c++;
        time_t clock = time(NULL);
        printf("#%d: task0 started ,%s", c, ctime(&clock));

        // prework
        clock = time(NULL);
        printf("#%d: task0 starts prework ,%s", c, ctime(&clock));
        unit(T0_PREWORK);

        clock = time(NULL);
        printf("#%d: task0 ends prework ,%s", c, ctime(&clock));



        flag[0] = 1;
        turn = 1;
        int first = 1;
        while (flag[1] == 1 && turn == 1){
```

```
    // busy wait
    if (first == 1){
        clock = time(NULL);
        printf("#%d: task0 is waiting ,%s" ,c , ctime(&clock));
    first = 0;
    }
}

// critical section
clock = time(NULL);
printf("#%d: task0 enters CS ,%s", c, ctime(&clock));
unit(T0_CS);
for (x=0; x<3; x=x+1) {
    critical_array[x] = (critical_array[x]+1)%10;
}
// end of critical section
flag[0] = 0;
clock = time(NULL);
printf("#%d: task0 leaves CS ,%s", c, ctime(&clock));

// postwork
clock = time(NULL);
printf("#%d: task0 starts postwork ,%s", c, ctime(&clock));
unit(T0_POSTWORK);

clock = time(NULL);
printf("#%d: task0 ends postwork ,%s", c, ctime(&clock));
printf("#%d: task0 finished    ,%s", c, ctime(&clock));
```

```
        wdt[0] = (wdt[0] + 1) % 4;
        // periodic control
        wait_period(&info);
    }
}

void* task1()
{
    struct periodic_info info;
    int c = 0, i, j, x;

    make_periodic(1000000 * T1_PERIOD, &info);

    while (1) {
        c++;
        time_t clock = time(NULL);
        printf("#%d: task1 started    ,%s", c, ctime(&clock));

        // prework
        clock = time(NULL);
        printf("#%d: task1 starts prework    ,%s", c, ctime(&clock));
        unit(T1_PREWORK);
        clock = time(NULL);
        printf("#%d: task1 ends prework ,%s", c, ctime(&clock));

        flag[1] = 1;
        turn = 0;
```

```
int first = 1;
while (flag[0] == 1 && turn == 0){
   // busy wait
   if (first == 1){
      clock = time(NULL);
      printf("#%d: task1 is waiting ,%s",c , ctime(&clock));
      first = 0;
   }
   //printf(".");
}
// critical section
clock = time(NULL);
printf("#%d: task1 enters CS ,%s", c, ctime(&clock));
unit(T1_CS);
for (x =0; x<3; x=x+1) {
         printf("%d", critical_array[x]);
}
printf("\n");
// end of critical section
flag[1] = 0;
clock = time(NULL);
printf("#%d: task1 leaves CS ,%s", c, ctime(&clock));

// postwork
clock = time(NULL);
printf("#%d: task1 starts postwork ,%s", c, ctime(&clock));
unit(T1_POSTWORK);
clock = time(NULL);
```

```
        printf("#%d: task1 ends postwork ,%s", c, ctime(&clock));
        printf("#%d: task1 finished ,%s", c, ctime(&clock));

        wdt[1] = (wdt[1] + 1) % 4;
        // periodic controll
        wait_period(&info);
    }
}
```

(三)週期設定

```
 static int make_periodic(
    unsigned int period,
    struct periodic_info *info)
 {
    static int next_sig;
    int ret;
    unsigned int ns;
    unsigned int sec;
    struct sigevent sigev;
    struct itimerspec itval;

    /* Initialise next_sig first time through. We can't use static
    initialisation because SIGRTMIN is a function call, not a
constant */
    if (next_sig == 0)
        next_sig = SIGRTMIN;
    /* Check that we have not run out of signals */
    if (next_sig > SIGRTMAX)
       return -1;
    info->sig = next_sig;
    next_sig++;

    info->wakeups_missed = 0;

    /* Create the signal mask that will be used in wait_period */
```

```
        sigemptyset (&(info->alarm_sig));
        sigaddset (&(info->alarm_sig), info->sig);

        /* Create a timer that will generate the signal we have chosen
*/
        sigev.sigev_notify = SIGEV_SIGNAL;
        sigev.sigev_signo = info->sig;
        sigev.sigev_value.sival_ptr = (void *) &info->timer_id;
        ret = timer_create (CLOCK_REALTIME, &sigev,
&info->timer_id);
        if (ret == -1)
           return ret;

        /* Make the timer periodic */
        sec = period/1000000;
        ns = (period - (sec * 1000000)) * 1000;
        itval.it_interval.tv_sec = sec;
        itval.it_interval.tv_nsec = ns;
        itval.it_value.tv_sec = sec;
        itval.it_value.tv_nsec = ns;
        ret = timer_settime (info->timer_id, 0, &itval, NULL);
        return ret;
      }

      static void wait_period(struct periodic_info *info)
      {
        int sig;
        printf("\n");
```

```
    sigwait (&(info->alarm_sig), &sig);
    info->wakeups_missed += timer_getoverrun (info->timer_id);
}
```

三、 工作負載模型萃取框架

我們的語意模型建構與工作負載模型萃取框架包括以下三個部分：

(一)代表工作負載模型的性質

工作負載模型中的各項參數值，包括各任務的週期長度、每個週期內的負載、可能的抖動值，以不變量表示為程式應滿足之性質。週期性則透過一執行緒中訊號的設定及訊號的等待確實成對出現來規範。

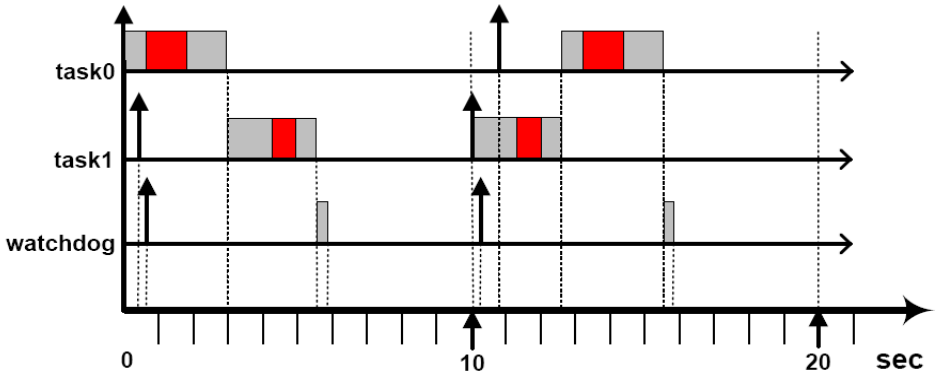
(二)有時間界限之狀態改變系統

這部分將程式之行為以抽象但不失準確的語意模型表現，特別是各執行緒之交替執行與程式執行時間之規範。

(三)有關單執行緒之假設

單一執行緒程式區塊執行時間的上界可由獨立之靜態分析工具獲得，我們將這些資訊以公設之方式呈現。

應用這套工作負載模型萃取框架方法，可從實例系統程式碼獲致如下圖中的工作負載模型之各項參數，以作為更精確之排程分析的依據。



參、　主要發現與結論

核能儀控系統中較複雜的多功能控制軟體一般以即時多執行緒程式實作，必須依賴特定即時作業系統核心方能運作。因此，欲對這類控制軟體進行正規驗證，必須先取得即時作業系統排程等行為之精確語意模型；然而，就現況而言，即時作業系統缺乏排程等行為之精確模型。這也是造成目前以靜態分析為主要手段的執行時間估算方法與工具，無法掌握特定即時作業系統排程行為的主因。我們所提的方法實質上將作業系統行為公設化，也就是自行規範所需的語意模型，其真實性仍有待嚴謹測試方式檢驗。

另一方面，核能儀控系統中也有較簡單的、無需作業系統便能運作的軟體模組，如感應器監控軟體。這類軟體之正規驗證無需擔憂複雜之作業系統行為，應是正規方法更能完整發揮的地方。當然，執行程式的處理器硬體仍須以適當語意模型表述其行為，但這比為作業系統核心塑模要單純許多。

肆、　參考文獻

1. [MISRA-C: 2004 --- Guidelines for the Use of the C Language in Critical Systems] MISRA-C: 2004 --- Guidelines for the Use of the C Language in Critical Systems. MIRA Limited (2004)
2. [aiT] aiT. Available at:　HYPERLINK "http://www.absint.com/ait/" http://www.absint.com/ait/
3. [Bound-T] Bound-T. Available at:　HYPERLINK

"http://www.bound-t.com/" http://www.bound-t.com/

4. [Clarke, Emerson, and Sistla 1986] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. In ACM Transactions on Programming Languages and Systems, vol 8, no. 2, pp. 244-263, 1986.
5. [Clarke, Grumberg, and Peled 1999] E. M. Clarke, O. Grumberg, and D. A. Peled. Model checking, The MIT Press, 1999.
6. [Coq proof assistant] Coq proof assistant. Available at: HYPERLINK "http://coq.inria.fr/" http://coq.inria.fr/
7. [Frama-C software analyzers] Frama-C software analyzers. Available at: HYPERLINK "http://frama-c.com/" http://frama-c.com/
8. [Heckmann and Ferdinand 2004] R. Heckmann and C. Ferdinand. Worst-case execution time prediction by static program analysis. White paper, AbsInt Angewandte Informatik GmbH, 2004. http://www.absint.com/wcet.htm.
9. [Henia et al. 2005] R. Henia, A. Hamann, M. Jersak, R. Racu, K. Richter, and R. Ernst. System level performance analysis-the SymTA/S approach. In IEE Computers and Digital Techniques, vol 152, no. 2, pp. 148-166, 2005.
10. [Hoare 1969] C. A. R. Hoare. An axiomatic basis for computer programming. In Communications of the ACM, vol 12, no. 10, pp. 576-580, 1969.
11. [Holzmann 2003] G. J. Holzmann. The SPIN Model Checker: Primer and Reference Manual, Addison-Wesley, 2003.
12. [Jacobs et al. 2011] B. Jacobs, J. Smans, P. Philippaerts, F. Vogels, W. Penninckx, and F. Piessens. VeriFast: A Powerful, Sound, Predictable, Fast Verifier for C and Java. In NASA Formal Methods, vol LNCS 6617, pp. 41-55, 2011.
13. [Jacobs, Smans, and Piessens 2010] B. Jacobs, J. Smans, and F.

Piessens 2010, 'A Quick Tour of the VeriFast Program Verifier', APLAS 2010, 2010, pp. 304-311.

14. [Künzli et al. 2007] S. Künzli, A. Hamann, R. Ernst, and L. Thiele. Combined approach to system level performance analysis of embedded systems. In Proceedings of the 5th IEEE/ACM International Conference on Hardware/Software Codesign and System Synthesis, pp.63-68, Salzburg, Austria, 2007.
15. [Lamport 2005] Leslie Lamport 2005, 'Real Time is Really Simple', Technical Report MSR-TR-2005-30, 2005.
16. [Martin and Leslie] Abadi Martin and Lamport Leslie, 'An Old-Fashioned Recipe for Real Time', Research Report 91, Digital Equipment Corp., Systems Research Center, Palo.
17. [Modex] Modex. Available at: HYPERLINK "http://cm.bell-labs.com/cm/cs/what/modex/" http://cm.bell-labs.com/cm/cs/what/modex/
18. [Muhammad and Nahida 2011] I. H. Muhammad and S. C. Nahida. A Practical approach on Model checking with Modex and Spin. In International Journal of Electrical & Computer Sciences IJECS-IJENS, no. 5, pp. 1-7, 2011.
19. [Nipkow, Paulson, and Wenzel 2002] T. Nipkow, L. C. Paulson, and M. Wenzel. Isabelle/HOL: a proof assistant for higher-order logic, Springer, 2002.
20. [RapiTime] RapiTime. Available at: HYPERLINK "http://www.rapitasystems.com/rapitime" http://www.rapitasystems.com/rapitime
21. [RT-Druid] RT-Druid. Available at: HYPERLINK "http://www.evidence.eu.com/content/view/28/51/" http://www.evidence.eu.com/content/view/28/51/
22. [SPIN] SPIN. Available at: HYPERLINK "http://spinroot.com/spin/whatispin.html"

http://spinroot.com/spin/whatispin.html

23. [SWEET] SWEET. Available at: HYPERLINK "http://www.mrtc.mdh.se/projects/wcet/sweet.html" http://www.mrtc.mdh.se/projects/wcet/sweet.html
24. [SymTA/S] SymTA/S. Available at: HYPERLINK "http://www.symtavision.com/symtas.html" http://www.symtavision.com/symtas.html
25. [VeriFast Tutorial] VeriFast Tutorial. Available at: HYPERLINK "http://people.cs.kuleuven.be/~bart.jacobs/verifast/tutorial.pdf" http://people.cs.kuleuven.be/~bart.jacobs/verifast/tutorial.pdf
26. [VeriFast] VeriFast. Available at: HYPERLINK "http://people.cs.kuleuven.be/~bart.jacobs/verifast/" http://people.cs.kuleuven.be/~bart.jacobs/verifast/
27. [Wilhelm et al. 2008] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Staschulat, and P. Stenström. The worst-case execution-time problem - overview of methods and survey of tools. In ACM Transactions in Embedded Computing Systems, vol 7, no. 3, pp. 36-1, May 2008.