Verifying the FreeRTOS Real-Time OS

Deepak D’Souza

Department of Computer Science and Automation
Indian Institute of Science, Bangalore.

Joint work with Sumesh Divakaran, Anirudh Kushwah, Prahlad Sampath, Jim Woodcock, and others.

Project funded by UKIERI (2009-12)

10 January 2013
Outline

1. How RTOS works
2. ADT’s and refinement in Z
3. RTOS as an ADT
4. Verification strategy
5. Bugs found
How interrupts are handled on an ARM processor

- Various kinds of interrupts may be generated (hardware eg. timer, software, instruction exceptions).
- Corresponding mode bits are set in CPSR[4:0] (Eg. 10011 for SWI).

<table>
<thead>
<tr>
<th>Exception</th>
<th>Resulting Mode</th>
<th>IVT address</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reset</td>
<td>Supervisor</td>
<td>0x00000000</td>
<td>1</td>
</tr>
<tr>
<td>Undefined Inst.</td>
<td>Undef</td>
<td>0x00000004</td>
<td>6</td>
</tr>
<tr>
<td>Software Interrupt</td>
<td>Supervisor</td>
<td>0x00000008</td>
<td>6</td>
</tr>
<tr>
<td>Abort prefetch</td>
<td>Abort</td>
<td>0x0000000C</td>
<td>2</td>
</tr>
<tr>
<td>Abort data</td>
<td>Abort</td>
<td>0x00000010</td>
<td>2</td>
</tr>
<tr>
<td>IRQ</td>
<td>IRQ</td>
<td>0x00000018</td>
<td>4</td>
</tr>
<tr>
<td>FIQ</td>
<td>FIQ</td>
<td>0x0000001C</td>
<td>3</td>
</tr>
</tbody>
</table>
What the processor does on an interrupt

- Saves current PC in LR’, CPSR in CPSR’.
- Changes to “super” mode.
- Disables lower priority interrupts.
- Branches to appropriate IVT entry.
What RTOS provides the programmer

Ways to:

- Create and manage multiple tasks.
- Schedule tasks based on priority-based pre-emption.
- Let tasks communicate (via message queues, semaphores, mutexes).
- Let tasks delay and timeout on blocking operations.
Example application that uses RTOS

Sample RTOS application

```c
int main(void){
    xTaskCreate(foo, "Task 1", 1000, NULL, 1, NULL);
    xTaskCreate(bar, "Task 2", 1000, NULL, 2, NULL);
    vTaskStartScheduler();
}

void foo(void* params){
    for(;;);
}

void bar(void* params){
    for(;;){
        for(;;){
            vTaskDelay(2);
        }
    }
}
```
Task execution in example application

```c
int main(void){
    xTaskCreate(foo, "Task 1", ...,);
    xTaskCreate(bar, "Task 2", ...,);
    vTaskStartScheduler();
}

void foo(void* params){
    for(;;);
}

void bar(void* params){
    for(;;){
        vTaskDelay(2);
    }
}
```
Example application: execution sequence

main
  vtaskCreate(1)
  vtaskCreate(2)
  vtaskStartScheduler()
    create Idle task
  task2
    vtaskdelay()
    yield()
  task1
    timer interrupt
  task2
    timer interrupt
About RTOS implementation

- Written mostly in C.
- Assembly language for processor-specific code.
- Portable:
  - Processor independent code is in 3 C files.
  - Processor dependent code (called a “port” in RTOS) is organised by Compiler-Processor pairs.
  - (19 compilers, 27 processors supported).
- Small footprint (≈3,000 lines), engineered for efficiency.
- Well-written, and well-documented through comments.
### Key data structures: Task Control Block

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pxTopOfStack</td>
<td>points to the top element in stack</td>
</tr>
<tr>
<td>xMPUSettings</td>
<td>MPU setting – part of port layer</td>
</tr>
<tr>
<td>xGenericListItem</td>
<td>to place the Task in READY and BLOCKED lists</td>
</tr>
<tr>
<td>xEventListItem</td>
<td>to place the Task in event lists</td>
</tr>
<tr>
<td>uxPriority</td>
<td>priority of the task</td>
</tr>
<tr>
<td>pxStack</td>
<td>points to the start of the stack</td>
</tr>
<tr>
<td>pcTaskName</td>
<td>descriptive name for task – for debugg</td>
</tr>
<tr>
<td>pxEndOfStack</td>
<td>points to the end of stack – for checking overflows</td>
</tr>
<tr>
<td>uxCriticalNesting</td>
<td>for critical section nesting</td>
</tr>
<tr>
<td>uxTCBNumber</td>
<td>for tracing the scheduler – the task count</td>
</tr>
<tr>
<td>uxBasePriority</td>
<td>for priority inheritance – last assigned priority</td>
</tr>
<tr>
<td>pxTaskTag</td>
<td>task hook function</td>
</tr>
<tr>
<td>ulRunTimeCounter</td>
<td>MPU time used by the task</td>
</tr>
</tbody>
</table>

**Task Control Block (TCB)**
Key data structures: xList

- Operations it provides: initialize, insert at end, insert ordered by itemvalue, remove a node, set itemvalue of a node, etc. (some 13 operations).
- xList is used to implement
  - FIFO queues (ReadyQ),
  - priority queues (Delayed list, Event lists).
RTOS data structures using xList

- currentTask: Task1
- pxReadyTasksList:
  - 0: idle
  - 1: NULL
  - 2: NULL
  - 3: NULL
  - 4: NULL
  - n-1: NULL
- delayedTasks: Task2
- xQueue:
  - waitingToSnd: Task2
  - waitingToRcv: NULL
How RTOS works
ADT's and refinement in Z
RTOS as an ADT
Verification strategy
Bugs found

Extracts from code: macro to add task to ready queue

/* Place the task represented by pxTCB into the appropriate ready queue for the task. It is inserted at the end of the list. */

#define prvAddTaskToReadyQueue(pxTCB){
    if(pxTCB->uxPriority > uxTopReadyPriority){
        uxTopReadyPriority = pxTCB->uxPriority;
    }
    vListInsertEnd((xList*) &(pxReadyTasksLists[pxTCB->uxPriority]),
                    &(pxTCB->xGenericListItem));
}
void vTaskDelay(portTickType xTicksToDelay){
    portTickType xTimeToWake;
    signed portBASE_TYPE xAlreadyYielded = pdFALSE;

    if( xTicksToDelay > (portTickType) 0){
        vTaskSuspendAll();
        /* Calculate the time to wake - this may overflow but this is not a problem. */
        xTimeToWake = xTickCount + xTicksToDelay;
        /* We must remove ourselves from the ready list before adding ourselves to the blocked list as the same list item is used for both lists. */
        vListRemove((xListItem *) &(pxCurrentTCB->xGenericListItem));
        /* The list item will be inserted in wake time order. */
        listSET_LIST_ITEM_VALUE(&(pxCurrentTCB->xGenericListItem), xTimeToWake);
    ....
    portYIELD_WITHIN_API();
}
Extracts from code: vPortYieldProcessor()

```c
/* ISR to handle manual context switches like taskYIELD(). */
void vPortYieldProcessor(void) __attribute__((interrupt("SWI"), naked));

void vPortYieldProcessor(void){
    /* Within an IRQ ISR the link register has an offset from
     * the true return address... */
    __asm volatile ("ADD LR, LR, #4");

    /* Perform the context switch. First save the context of
     * the current task. */
    portSAVE_CONTEXT();

    /* Find the highest priority task that is ready to run. */
    __asm volatile ("BL vTaskSwitchContext");

    /* Restore the context of the new task. */
    portRESTORE_CONTEXT();
}
```
#define portSAVECONTEXT() {
    extern volatile void * volatile pxCurrentTCB;
    /* Push R0 as we are going to use the register. */
    /* Set R0 to point to the task stack pointer. */
    "STMDB SP,{SP}~"
    "SUB SP, SP, #4"
    "LDMIA SP!,{RO}"
    /* Push the return address onto the stack. */
    "STMDB R0!, {LR}"
    /* Now we have saved LR we can use it instead of R0. */
    "MOV LR, R0"
    /* Pop R0 so we can save it onto the system mode stack. */
    "STMDB LR,{RO-LR}"
    /* Store the new top of stack for the task. */
    "LDR RO, =pxCurrentTCB"
    "LDR RO, [RO]"
    "STR LR, [RO]"
}

Extracts from code: portSAVECONTEXT()
void vTaskEnterCritical(void){ /* in task.c */
    ...
    portDISABLE_INTERRUPTS();
}

#define portDISABLE_INTERRUPTS()
    __asm volatile ( 
    "STMDB SP!, {R0}" /* Push R0.*/
    "MRS  R0, CPSR " /* Get CPSR.*/
    "ORR  R0, R0, #0xC0" /* Disable IRQ, FIQ.*/
    "MSR  CPSR, R0" /* Write back modified value. */
    "LDMIA SP!, {R0}" /* Pop R0.*/
    )
Main functionality of RTOS

- Implement its stated scheduling policy (fixed priority pre-emptive scheduling).
- Trap SWI interrupts
  - Find highest priority ready task to run.
  - Save context of yielding task.
  - Restore context of new task.
- Trap timer IRQ interrupt
  - Update tickcount,
  - Check delayed tasks, and move to ready if required,
  - Switch context if required.
- Provide API’s for:
  - Task creation, deletion, set priority, etc.
  - Inter-task communication through queues, semaphores, and mutexes.
Separate requirements: Scheduling-related and port-specific

- **Scheduling-related**
  - Implement its stated scheduling policy (fixed priority pre-emptive scheduling).
  - Provide API’s for:
    - Task creation, deletion, set priority, etc.
    - Inter-task communication through queues, semaphores, and mutexes.
  - Handle timer event correctly
    - Update tickcount,
    - Check delayed tasks, and move to ready if required,

- **Port-specific**
  - Trap SWI and timer interrupts correctly.
  - Perform context-switching (save and restore) correctly.
  - Provide correct “enterCritical” and “exitCritical” implementation.
Example Z model: Resource allocator

```
ResourceManager
free : IF \in \mathbb{N}

Allocate
\Delta ResourceManager
r! : \mathbb{N}

r! \in free \land free' = free \setminus \{r!\}
```
Notion of refinement

Program using a Stack ADT:

Consider Stack’ which satisfies property:

Every sequence of operations on Stack’ can be matched by a sequence on Stack.

Then Stack’ is said to refine Stack.

If a client program is happy with an ADT, it will also be happy with a refinement of it.
Refinement example: Resource allocator

\[
\text{ResourceManager} \\
\text{free : \mathbb{N}}
\]

\[
\text{Allocate} \\
\Delta \text{ResourceManager} \\
\text{r! : \mathbb{N}}
\]

\[
r! \in \text{free} \land \text{free'} = \text{free} \setminus \{r!\}
\]
Refinement example: Resource allocator

Allocate

\[ \Delta ResourceManager \]

\[ r! : \mathbb{N} \]

\[ r! \in free \land free' = free \setminus \{r!\} \]

A refinement of allocate:

Allocate_1

\[ \Delta ResourceManager \]

\[ r! : \mathbb{N} \]

\[ r! = \min free \land free' = free \setminus \{r!\} \]
Sufficient condition for refinement

Init strengthening: every concrete init state is related to an abstract init state.

Guard strengthening:

```
\begin{align*}
\text{If } \text{op} \text{ is enabled in an abstract state then the concrete op is also enabled in any related concrete state.}
\end{align*}
```

Simulation:

```
\begin{align*}
\text{If } a_0 \text{ and } c_0 \text{ are related, if } \text{op} \text{ is enabled in } a_0, \text{ and if a concrete op } \text{op'} \text{ takes us from } c_0 \text{ to } c_1, \text{ then there exists } a_1 \text{ related to } c_1, \text{ such that } \text{op} \text{ takes us from } a_0 \text{ to } a_1.
\end{align*}
```
The concrete is **simulated-by** the abstract.

```
Abstract       a_0   op_1   a_1   op_2   a_2   op_1   a_3
      \     /     \     /     \     /     \     /     \      \
     / \   / \   / \   / \   / \   / \   / \   / \    
 c_0  op_1 c_1  op_2 c_2  op_1 c_3
```
Viewing FreeRTOS as an ADT

application
+
FreeRTOS

FreeRTOS
ADT

<tick, t'>
<startscheduler, t'>
<taskcreate(t,3), t'>

Application
FreeRTOS as an ADT

- The OS essentially services API calls from the running task.
- View as a state machine with operations corresponding to API calls.
RTOS as ADT in Z

\[
\text{CreateTask} \\
\hspace{1cm} \Delta \text{Task} \\
\hspace{1cm} \text{taskIn}? : \text{TASK} \\
\hspace{1cm} \text{prio}? : \mathbb{N} \\
\]

\[
\text{taskIn}? \not\in \text{tasks} \\
\text{tasks}' = \text{tasks} \cup \{\text{taskIn}??\} \\
\text{ready}' = \text{ready} \sim \langle \text{taskIn}? \rangle \\
\text{priority}' = \text{priority} \oplus \{(\text{taskIn}? \rightarrow \text{prio})\}
\]
Specification of FreeRTOS

FreeRTOS
ADT in Z

Application

<tick, t'>
<startscheduler, t'>
<taskcreate(t,3), t'>

C Implementation of FreeRTOS

Refinement
Verification Strategy

For $Z_2 \leftrightarrow C_1$:

- Import guards, invariants, and BAP from $Z_2$.
- Check validity of model $C_1$ in VCC.

Top–level Z model

$Z_1$

Refinement

$Z_2$

Refinement

$C_1$

Refinement

$C_2$

Refinement

$C_3$

RTOS C implementation

Complemented priority, overflow delayed, pending ready

Lists: arrays of (ptrs to) TCBs

Arrays of listitem nodes

Xlists of listitem nodes
Verification Strategy

For $C_1 \leftrightarrow C_2$, $C_2 \leftrightarrow C_3$:

- **Phrase and check validity of models $C_2$ and $C_3$ in VCC.**
- **Check refinement conditions in VCC.**

![Diagram]

- **Top-level Z model $Z_1$**
- **Refinement**
  - **$Z_2$**
    - **Complemented priority, overflow delayed, pending ready**
    - **Refinement**
      - **$C_1$**
        - **Lists: arrays of (ptrs to) TCBs**
      - **Refinement**
        - **$C_2$**
          - **Arrays of listitem nodes**
        - **Refinement**
          - **$C_3$**
            - **RTOS C implementation**
            - **Xlists of listitem nodes**
Verification in VCC: xList data structure

- Verified that xList is a refinement of a simple array (ghost map in VCC) based implementation.
- original code 500 lines, VCC annotations 600 lines.

```c
_(ghost xListItem * xArray[portTickType])
/* map is used as an ARRAY. */
    /* ADT that is maintained as a shadow copy. */

_(ghost portTickType xArrayIndex[xListItem * ])
/* map that stores the index of element in the array. */
....

void vListInsertEnd( xList *pxList, xListItem *pxNewListItem ){
    xListItem *pxIndex;
    pxIndex = pxList->pxIndex;
    _(assert pxIndex->pxNext \in pxList->\owns)
    _(assert pxIndex->pxPrevious \in pxList->\owns)
    _(unwrap pxNewListItem)
    pxNewListItem->pxNext = pxIndex->pxNext;
    pxNewListItem->pxPrevious = pxList->pxIndex;
    _(wrap pxNewListItem)
```
void vTaskDelay( portTickType xTicksToDelay )
{
    portTickType xTimeToWake;
    signed portBASE_TYPE xAlreadyYielded = pdFALSE;
    if( xTicksToDelay > ( portTickType ) 0 )
    {
        _(unwrap (&SCR))
        _(assert \forall int i ; ( (i >= 0) && (i < configMAX_PRIORITIES)) ==> \wrapped(&pxReadyTasksLists[i]))
        uxSchedulerSuspended++;
        xList *list = (pxReadyTasksLists + uxTopReadyPriority);
        _(unchecked)xTimeToWake = xTickCount + xTicksToDelay;
        _(unwrapping pxCurrentTCB){
            _(assert (&( pxCurrentTCB->xGenericListItem )) \in (&SCR)->\owns)
            vListRemove( &( pxCurrentTCB->xGenericListItem ) _(ghost (pxReadyTasksLists + uxTopReadyPriority)));
            _(assert \wrapped(pxReadyTasksLists + uxTopReadyPriority))
            vListInsert( pxOverflowDelayedTaskList, &( pxCurrentTCB->xGenericListItem )
            _(ghost SCR.taskGenList[pxCurrentTCB] = pxOverflowDelayedTaskList)
            _(ghost pxCurrentTCB->state = SLEEPING_OVF)
        }
    }else{
        _(unwrapping pxCurrentTCB){
            vListInsert( pxDelayedTaskList, &( pxCurrentTCB->xGenericListItem )
            _(ghost SCR.taskGenList[pxCurrentTCB] = pxDelayedTaskList)
            _(ghost pxCurrentTCB->state = SLEEPING)
        }
    }
    //_(assume itemsAreGenericListItem( pxReadyTasksLists + uxTopReadyPriority))
    _(ghost unsigned portBASE_TYPE sPriority = uxTopReadyPriority)
    while( listLIST_IS_EMPTY( pxReadyTasksLists + uxTopReadyPriority ) )
        _(writes &uxTopReadyPriority)
        _(invariant ( pxReadyTasksLists + 0 )->tSize > 0)
        _(invariant \wrapped(pxReadyTasksLists + uxTopReadyPriority))
        _(invariant (pxReadyTasksLists + uxTopReadyPriority) \in (&SCR)->\owns)
        _(invariant !( pxReadyTasksLists + uxTopReadyPriority)->tasksInArray[pxCurrentTCB])
        _(invariant (pxReadyTasksLists +uxTopReadyPriority)->uxNumberOfItems == (pxReadyTasksLists +uxTopReadyPriority)->tSize)
        _(invariant (!listLIST_IS_EMPTY(pxReadyTasksLists + uxTopReadyPriority)))
        => (uxTopReadyPriority == (uxTopReadyPriority))
        _(invariant (uxTopReadyPriority >= 0) && (uxTopReadyPriority < configMAX_PRIORITIES))
        _(invariant \forall unsigned portBASE_TYPE i ; ( (i < configMAX_PRIORITIES ) &&
        (i > uxTopReadyPriority) ) ==> (((pxReadyTasksLists +i) \in (&SCR)->\owns) && ((pxReadyTasksLists +i)->tSize
        == 0 )))
        _(assert uxTopReadyPriority > 0)
        --uxTopReadyPriority;
    }
    _(assert ! (pxReadyTasksLists[ uxTopReadyPriority ].tasksInArray[pxCurrentTCB]))
    pxCurrentTCB = listGET_OWNER_OF_NEXT_ENTRY( &( pxReadyTasksLists[uxTopReadyPriority] ));
    _(assert (pxReadyTasksLists[ uxTopReadyPriority ].tasksInArray[pxCurrentTCB]))
    /* Assert required */
    _(assert pxCurrentTCB \in (&SCR)->\owns)
    _(unwrapping pxCurrentTCB){
        _(ghost pxCurrentTCB->state = RUNNING)
    }
}
**Problem with taskPrioritySet() function in RTOS**

Problem found by Sumesh Divakaran while trying to understand code in detail.

- According to RTOS User Guide: When an unblocking event occurs,
  
  *The task that is unblocked will always be the highest priority task that is waiting for the event. If the blocked tasks have equal priority, then the task that has been waiting for the longest period will be unblocked.*

- However, if taskPrioritySet is called on a blocked task, its new priority is **not** considered while selecting the task to be unblocked.
Example to illustrate setpriority problem

RTOS application

```c
int main(void){
    xTaskCreate(vTask1,"Task1",configMINIMAL_STACK_SIZE,NULL,1,&xTask1Handle);
    xTaskCreate(vTask2,"Task2",configMINIMAL_STACK_SIZE,NULL,2,&xTask2Handle);
    xTaskCreate(vTask3,"Task3",configMINIMAL_STACK_SIZE,NULL,3,&xTask2Handle);
    vTaskStartScheduler();
}

void vTask1(void *pvParameters){
    long lData = 10;
    xQueueSendToBack(xQueue,&lData,0);
    for(;;);
}

void vTask2(void *pvParameters){
    long lData = 20;
    xQueueSendToBack(xQueue,&lData,0);
    for(;;);
}

void vTask3(void *pvParameters){
    long lData = 30;
    xQueue = xQueueCreate(1,sizeof (long));
    xQueueSendToBack(xQueue,&lData,0);
    vTaskDelay(2);
    vTaskPrioritySet(xTask1Handle,4);
    xQueueReceive(xQueue,&lData,0);
    vTaskPrioritySet(NULL,0);
    for(;;);
}
```
Sequence of events produced by application

Event sequence produced by test application

```c
main, xTaskCreate(Task1,1)
main, xTaskCreate(Task2,2)
main, xTaskCreate(Task3,3)
Task3, xQueueCreate(xQueue,1)
Task3, xQueueSendToBack(xQueue,30)
Task3, vTaskDelay(2);

Task2, xQueueSendToBack(xQueue,20)_b;
Task1, xQueueSendToBack(xQueue,10)_b;

Task3, vTaskPrioritySet(Task1,4);
Task3, xQueueReceive(xQueue,30);
Task3, vTaskPrioritySet(Task3,0);

Task2, xQueueSendToBack(xQueue,20)_e;
```
Other bugs found

- Problem with priority inheritance mechanism.
- Problem with vTaskSuspend and vTaskResume API’s.
- Problem with scheduling newly created tasks.
Benefits of verification technique

- **Finding bugs:**
  - Problem with `vTaskSuspend` and `vTaskResume` API’s.
  - Problem with scheduling newly created tasks.
  - Problem with `vTaskPrioritySet` API in RTOS.

- **Gives us conditions for correct API usage:**
  - `vTaskDelay` should not be called on sole ready task (FreeRTOS crashes!).
Other things to check: Absence of data-races

- Tasks are essentially threads that can be interleaved in execution.
- API code called by tasks should not lead to a data-race when interrupted and interleaved with an API call from another task.
- For example:
  - Task1 calls QueueSend API
  - Pointers start getting adjusted to insert new message in the Queue
  - Tick interrupt occurs
  - Task2 runs and makes a call to QueueReceive.
- Need to ensure that API’s use critical sections when they have to.
- Could use a data-race detection tool on suitably modified RTOS code.
Port-specific aspects of correctness

- Trap SWI and timer interrupts correctly.
- Perform context-switching (save and restore) correctly.
  - SAVE_CONTEXT saves the “necessary” information about the swapped-out task on its stack.
  - RESTORE_CONTEXT correctly restores this information from the task’s stack.
- Provide correct “enterCritical” and “exitCritical” implementation.
  - Interrupts should be correctly disabled and re-enabled.
The End

Thank you.