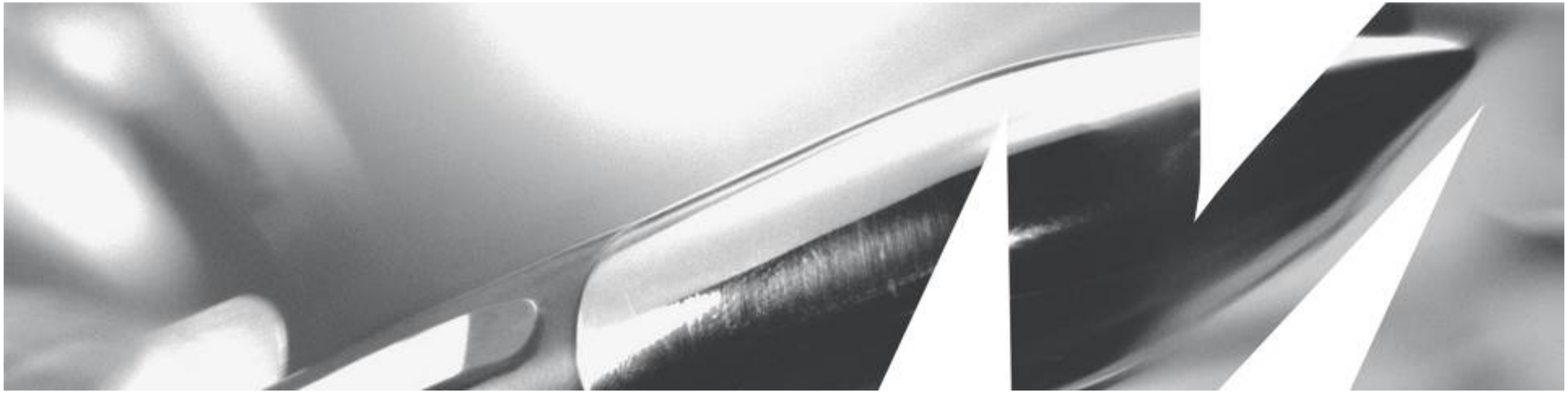


Hardware support for scheduling with Ada

Rod White

This document and the information contained herein is proprietary information of MBDA and shall not be disclosed or reproduced without the prior authorisation of MBDA. © MBDA 2008.

MBDA MISSILE SYSTEMS

Topics

- Context – the application domain
- Background and motivation
- Overall scheme
- What is a “Butler”
  - concept
  - hardware perspective
  - “instruction” set
- Integration with Ada
  - Zero-footprint
  - Ravenscar(-like)
- Cooperative vs. preemptive scheduling
- Results and observations
- Future developments
- Conclusions/questions

This document and the information contained herein is proprietary information of MBDA and shall not be disclosed or reproduced without the prior authorisation of MBDA. © MBDA 2008.

Context – the application domain

- Real-time embedded systems
  - Constrained by
    - Available volume
    - Allowable mass
    - Power availability and dissipation
  - Harsh operational environment (temperature, vibration, shock etc)
    - COTS can not be used – few real technology choices available
- High integrity
  - Defstan 00-55/56 SIL3/4, DO-178B Level A/B
- Long lived products
  - Periodic updates – obsolescence and technology insertion
  - Evolving roles – requirements change and become more demanding over time
  - Moderate production volumes – limited opportunity for economy of scale

This document and the information contained herein is proprietary information of MBDA and shall not be disclosed or reproduced without the prior authorisation of MBDA. © MBDA 2008.

MBDA
MISSILE SYSTEMS

Background and motivation

- Nature of the application is hard real-time
  - A large number of external stimuli which can be at a high rate (e.g. > 15 kHz)
  - Complex and wide ranging algorithmic requirements
    - e.g. control, navigation, and image/radar processing
    - Safety related/critical parts
  - Processing deadlines must be met
  - Can't afford to use all of the capacity on first delivery
    - Evolving roles require some usable processing margin
- Products can only be supplied with limited processing resources
  - The computer system is always being squeezed
    - Small volume and mass (fuel and payload are considered to be more "useful")
    - Aim to minimise cost in production

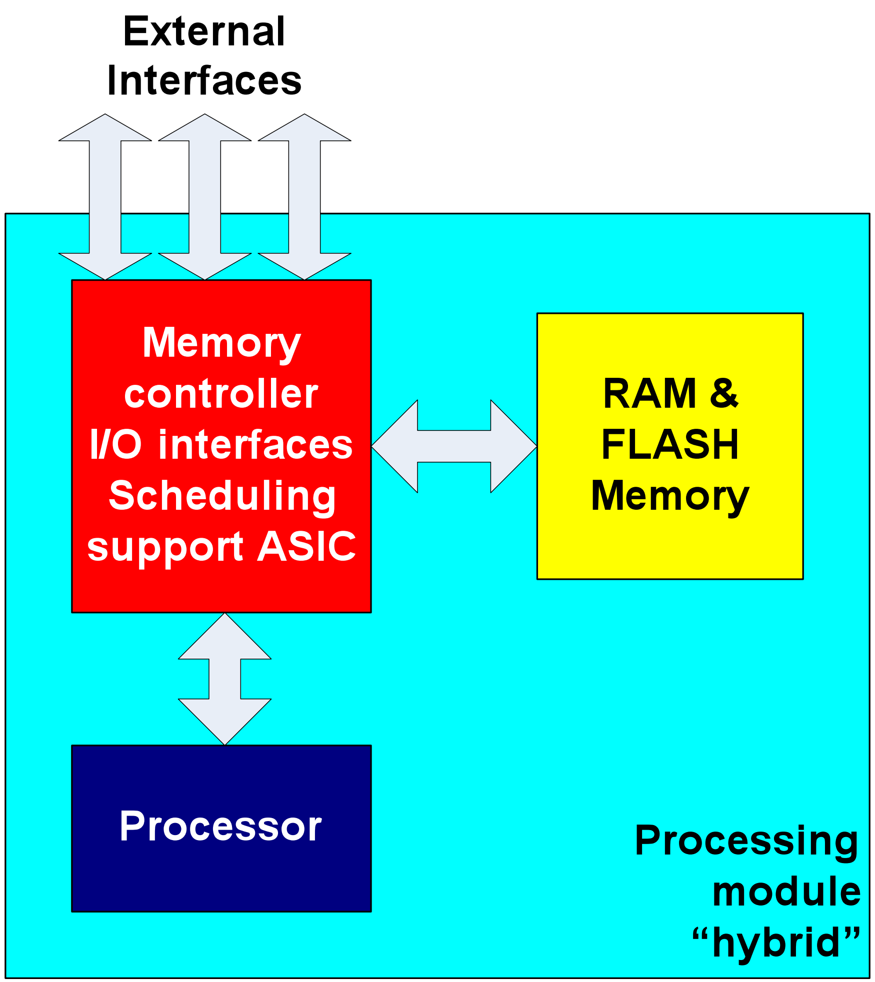
  → limited processing resources
- Need to maximise the potential capacity of processing platform
  - Dedicated hardware functions can be an efficient way of offloading tasks from the general purpose processor
  - Diverse tasks and a high rate of interrupts make scheduling decisions potentially costly
    - Can some of the scheduling "problem" be offloaded to hardware support?

This document and the information contained herein is proprietary information of MBDA and shall not be disclosed or reproduced without the prior authorisation of MBDA. © MBDA 2008.

Overall scheme

- The goal is to offload the collection and organisation of information that is used to handle interrupts and schedule tasks in a unified manner
  - Minimal space and power footprint
  - Efficient coupling to the processor – low latency, simple access
  - High integrity – sufficiently deterministic for safety related applications

- COTS processors and memory parts in a custom package
  - ASIC is the glue that binds it all together



This document and the information contained herein is proprietary information of MBDA and shall not be disclosed or reproduced without the prior authorisation of MBDA. © MBDA 2008.

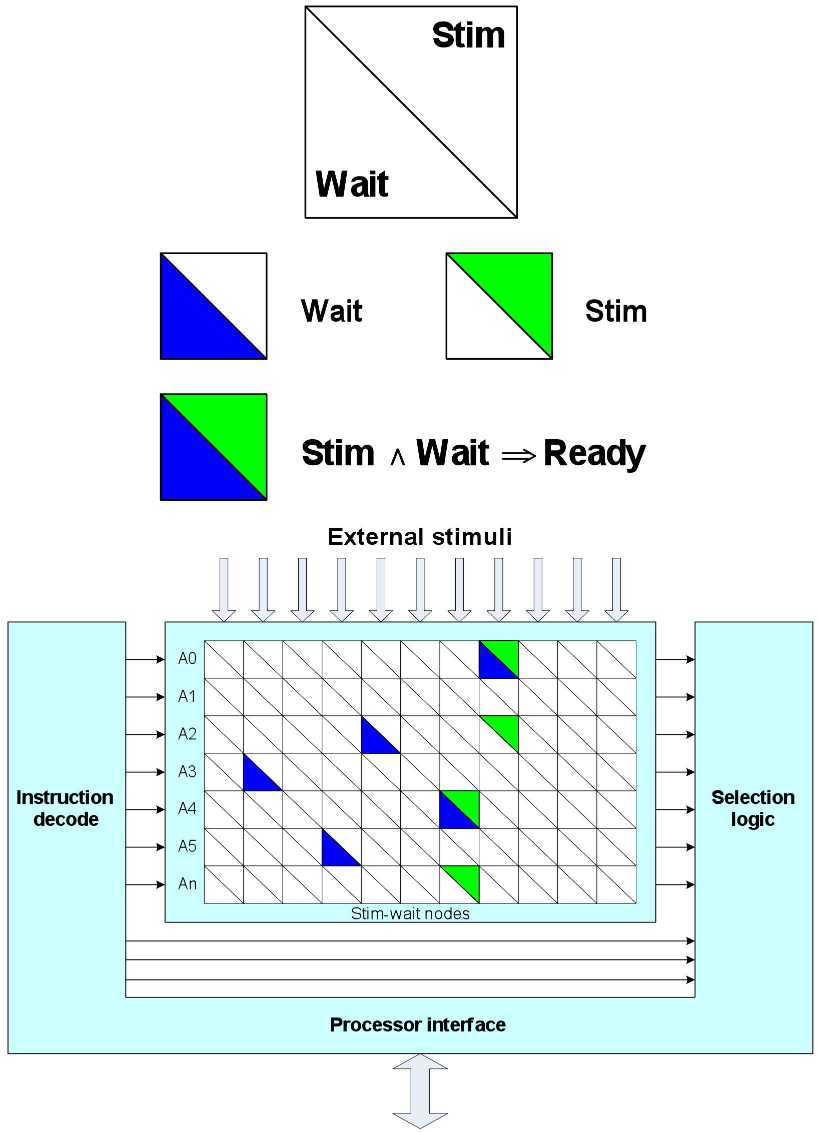
What is a butler – the concept

- Why a Butler? Using the term in its traditional (British) sense:
  - the person who organises the smooth operation of the household whilst deferring to a higher "authority" for the important decisions
- The hardware Butler handles all of the interrupts and is the focus for all of the inter-task interactions, determining what is available to execute
  - Highly parallel hardware is good at handling a diverse range of asynchronous sources
  - The decision to execute what is suggested is taken by a simple software kernel
    - Controls and interacts with the Butler using a small number of simple instructions
- The fundamental "element" is the activity – something that may be scheduled
  - All of the other elements relate to the organisation and "control" of the activities – waiting, stimulating, delaying etc
- One idea but there have been several implementation variants depending on:
    - the platform
    - the technology
    - the production volumes
    - …
  - Range from a bespoke ASIC solution, through FPGA firmware, to software in a co-processor

This document and the information contained herein is proprietary information of MBDA and shall not be disclosed or reproduced without the prior authorisation of MBDA. © MBDA 2008.

What is a “Butler” – hardware perspective

- Set of “resources” that relate to the activities and their grouping
  - The activities
  - Pollsets – groupings of activities considered to be at the same priority
  - Stim-wait nodes allowing activities to wait on, and be released by a particular event
  - A mechanism for the selection of the next activity to be executed
- Number of activities and width of the set of stim/wait nodes depends on the application
- Other (per activity) features include:
  - Watchdog timers to detect over-runs
  - Delay timers



This document and the information contained herein is proprietary information of MBDA and shall not be disclosed or reproduced without the prior authorisation of MBDA. © MBDA 2008.

What is a Butler – instruction set

- Small number of instructions are use to control the activities operationally
  - `Suspend` – causes the current activity to release control and make itself schedulable, allows other ready activities at the same or higher priority to be run
  - `Wait <bit vector>` – causes the current activity to release control and set the wait bits defined in the <bit vector>. Becomes reschedulable once a corresponding stim bit has been set by an event or `stim` instruction
  - `Stim <activity, bit vector>` – sets the stim bits in the bit vector for the defined activity, if any stim matches a corresponding wait then the activity becomes schedulable (used for s/w-s/w interactions)
  - `Next_Activity <activity>` – causes the selection logic to complete the evaluation of the next activity to execute, the highest priority one is returned. [used in conjunction with the `suspend` and `wait` instructions]
  - `Curract <activity>` – return the number of the current activity
  - `AMI <boolean>` – the returned state of the boolean indicates if there is an activity of higher priority than the current one ready to be run (Anything More Important)

- … and a few other are used in their initialisation, organisation into pollsets, and general house-keeping functions

This document and the information contained herein is proprietary information of MBDA and shall not be disclosed or reproduced without the prior authorisation of MBDA. © MBDA 2008.

MBDA MISSILE SYSTEMS

Integration with Ada – general approach

- The general design approach follows that of a real-time network
  - Focus on threads and their interactions
  - Somewhat old-fashioned but well suited to these kinds of applications
- Implementation uses a restricted subset of the Ada language
  - Only requires a simple runtime
  - High integrity – lots of SPARK Ada (keeping the required runtime small)
- Two approaches:
  - A virtually runtime free environment (zero-footprint)
    - Butler activities are related to threads supplied by a simple kernel
    - Kernel supports a wide variety of inter-thread communication protocols
  - A runtime supporting Ravenscar-like tasking constructs
    - Butler activities are related to Ada tasks
    - … which in turn are implemented over the threads supplied by the simple kernel
    - Protected object are used for inter-task communication
- The latter has the advantages of application portability at the cost of some complexity in the kernel and runtime

This document and the information contained herein is proprietary information of MBDA and shall not be disclosed or reproduced without the prior authorisation of MBDA. © MBDA 2008.

Integration with Ada – Zero-footprint runtime

- Main program creates all of the threads
  - Associates these with Butler activities
  - Organises them into pollsets

```
procedure Main is
begin
   Kernel.Initialise;  --  Ensures that the Butler is initialised
                       --  and that Activity'Last is the current activity

   Create_Thread (Activity_Ident   => Act_Id_1,
                  Pollset_Boundary => True,
                  Entry_Point      => Thread_1_Main'Access);
   Create_Thread (Activity_Ident   => Act_Id_2,
                  Pollset_Boundary => False,
                  Entry_Point      => Thread_2_Main'Access);
   Create_Thread (Activity_Ident   => Act_Id_3,
                  Pollset_Boundary => True,
                  Entry_Point      => Thread_3_Main'Access);

   --  All threads are now created but none have yet run
   --
   loop
      Yield_To_Higher; --  Allows the highest priority thread to
                       --  run, it will only return when there is
                       --  nothing else that is ready to run
   end loop;
end Main;
```

- Inside the inter-task communication protocols the Butler `Wait`, `Stim` and `Next_Activity` instructions are used to provide the desired behaviours

- Events are either handled by polling the `AMI` instruction periodically or as interrupts
  - Interrupts cause a full context switch to the thread mapped to the appropriate activity

```
if Butler.AMI then
   Butler.Suspend;
   Butler.Next_Activity (Next);
   Switch_To (Next);
end if;
```


This document and the information contained herein is proprietary information of MBDA and shall not be disclosed or reproduced without the prior authorisation of MBDA. © MBDA 2008.

Integration with Ada – Ravenscar(-like) runtime

- All tasks are declared at library level in a single package
  - Simplifies activation management
  - Each task is associated with a Butler activity
  - Pollsets are are organised to group the tasks of equal priority
- Interrupts are mapped to protected procedures
  - Each “interrupt” is also associated with a particular Butler activity
- Delay until statements use dedicated timers within the Butler to stimulate the activity as necessary
- Protected entries are underpinned by the use of the `Wait` and `Stim` instructions

```
package body tasks is

    task A is
    begin
       Register;
       loop
          Perform_Application_A;
       end loop;
    end A;
…
    task C is
    begin
       Register;
       loop
          Perform_Application_C;
       end loop;
    end A;

end Tasks;
```

```
Suspend;
loop
   Next_Activity (Next);
   exit when Type_Of (Next) = Thread;
   Invoke (Next);  --  Protected operation executed in context
                   --  of the current task
end loop;
Switch_To (Next);  --  New task
```

Where the procedure `Invoke` is:

```
procedure Invoke (Act : Activity_Id) is
begin
   Entry_Points (Act).all;     -- Execute the protected procedur
   Wait (Release_Mask (Act));  -- Make the protected procedure's
                               -- activity ``sleep'' again
end Invoke;
```


This document and the information contained herein is proprietary information of MBDA and shall not be disclosed or reproduced without the prior authorisation of MBDA. © MBDA 2008.

MBDA MISSILE SYSTEMS

Cooperative vs. preemptive scheduling

- Cooperative – no interrupts, the possible task switching points are defined *a priori*
  - The assumed advantages of cooperative scheduling are better schedulability and lower processing overheads
  - Disadvantage in hard-real time system is ensuring an adequate response to external events
    - Application needs to be "seeded" with cooperation points
    - Not possible to automate (so far) – too many points swamp the benefits
- The support provided by the Butler makes the implementation of such a scheme efficient and simple
  - Especially the `AMI` instruction
- Two primitives used to seed application
  - `Yield` (same or higher priority)
  - `Yield_to_Higher` (strictly higher priority)

```
procedure Yield is
   Current, Next : Butler.Activity_Id;
begin
   Butler.Suspend;
   Butler.Curract (Current);
   Butler.Next_Activity (Next);
   if Next /= Current then
      Switch_To (Next);  --  Pointless switching to self..
   end if;
end Yield;
```

```
procedure Yield_To_Higher is
   Next : Butler.Activity_Id;
begin
   if Butler.AMI then
      Butler.Suspend;
      Butler.Next_Activity (Next);
      Switch_To (Next);  --  If the priority is higher
                         --  it cannot be self
   end if;
end Yield_To_Higher;
```


This document and the information contained herein is proprietary information of MBDA and shall not be disclosed or reproduced without the prior authorisation of MBDA. © MBDA 2008.

MBDA MISSILE SYSTEMS

Results and observations

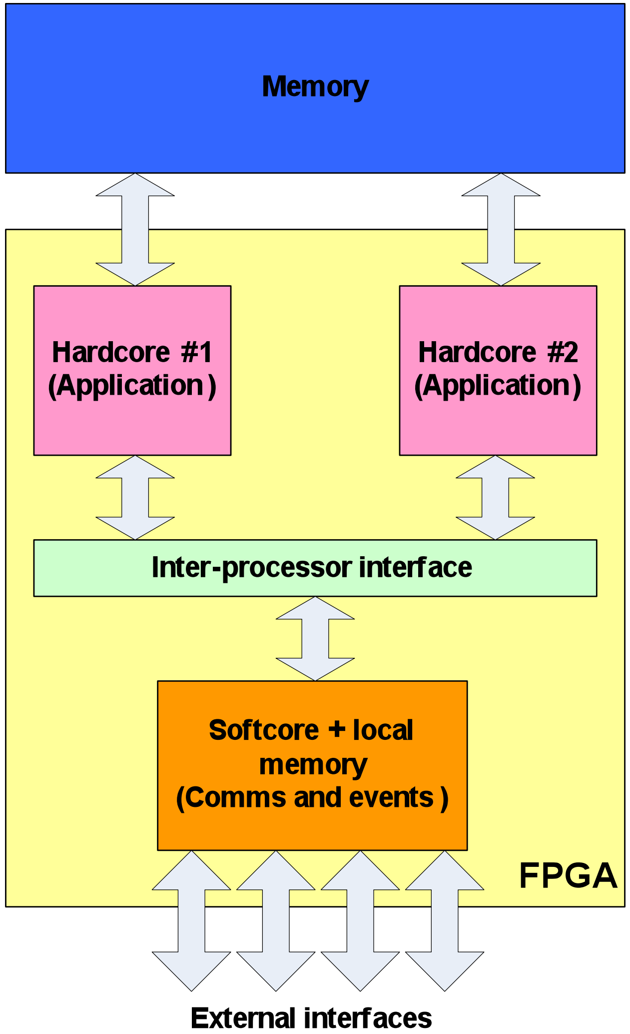
- Using the Butler in a preemptive environment delivers only small gains
  - 1-2% (of processor capacity)
  - However not heavily loaded
- Cooperative savings were better
  - 5% (of processor capacity)
  - Again, not heavily loaded
- More recently, on a more heavily loaded system savings of ~10% have been observed, also the loading profiles are less variable with the cooperative approach
  - 10% is considered to be a very worthwhile saving

- The effort required to seed the application is not insignificant
  - Especially in the early phases of development where the rate of change is high
- Suggests that starting with a preemptive approach and migrating to a cooperative one once the solution is relatively stable is probably a good way to proceed
  - Of course tool support for seeding would make this transition unnecessary

This document and the information contained herein is proprietary information of MBDA and shall not be disclosed or reproduced without the prior authorisation of MBDA. © MBDA 2008.

Future developments

- More powerful platforms based on FPGAs with embedded processing cores
  - Hardcores used for the application
  - Softcore use to manage interfaces and undertake the related Butler functions
  - …challenges of scheduling the application distributed across multiple cores with a single Butler device
- Integration with Ada 2005 features
  - Use of the Butler watchdog to support `Ada.Execution_Time.Timers` for overrun detection
  - Low overhead approach based on mapping the protected procedures in `Ada.Real_Time.Timing_Events` to Butler activities



This document and the information contained herein is proprietary information of MBDA and shall not be disclosed or reproduced without the prior authorisation of MBDA. © MBDA 2008.

MBDA
MISSILE SYSTEMS

Conclusions

- Using hardware to support Ada scheduling has proven to be effective
  - Used across a number of fielded products
  - Has allowed “easy” migration as the processing platform has evolved
- Integration with the limited Ada runtime has not proven to be too problematic
  - Given the high-integrity domain there seems little point in extending support beyond Ravenscar to allow a more “complete” tasking model
  - … still some limited possibilities to explore (Ada 2005 timer related)
- Cooperative scheduling enhances the benefits
  - Larger margins and more stable execution profiles
  - Ada 2012(?) should have support for the concepts of `Yield_to_Higher` and `Yield` for non-preemptive dispatching

- However modern processors present difficult challenges
  - Very high speed
  - Multiple cores

  → … hardware scheduling coordination seems like a good idea – but how? (efficiently)

This document and the information contained herein is proprietary information of MBDA and shall not be disclosed or reproduced without the prior authorisation of MBDA. © MBDA 2008.

Questions…

This document and the information contained herein is proprietary information of MBDA and shall not be disclosed or reproduced without the prior authorisation of MBDA. © MBDA 2008.