

Formal Methods Expert to IMA-SP Kernel Qualification – Preparation

Andrew Butterfield



Formal Methods Expert to IMA-SP Kernel Qualification – Preparation
ESTEC Contract No 4000112208

9th June 2016

Formal Methods Expert for IMA-KQP

➤ Formal Methods expertise

- provided by Lero@TCD
- How might formal methods and techniques assist in a kernel qualification activity?
 - complementing traditional techniques?
 - replacing traditional techniques?
 - Supporting traditional techniques?

➤ Builds on Lero@TCD experience with earlier MTOBSE activity

workflow

- **FMEIMAKQP matched the IMAKQP Phases**
- **Phase 1 Requirements**
 - Assisting with baseline development
- **Phase 2 Planning**
 - Identify how/what to formalise
 - Support Airbus in their FM planning activities
- **Phase 3 Case-Study**
 - Carry out some formalisation explorations
 - Support Airbus in their FM case-study activities
- **Phase 4 Report**

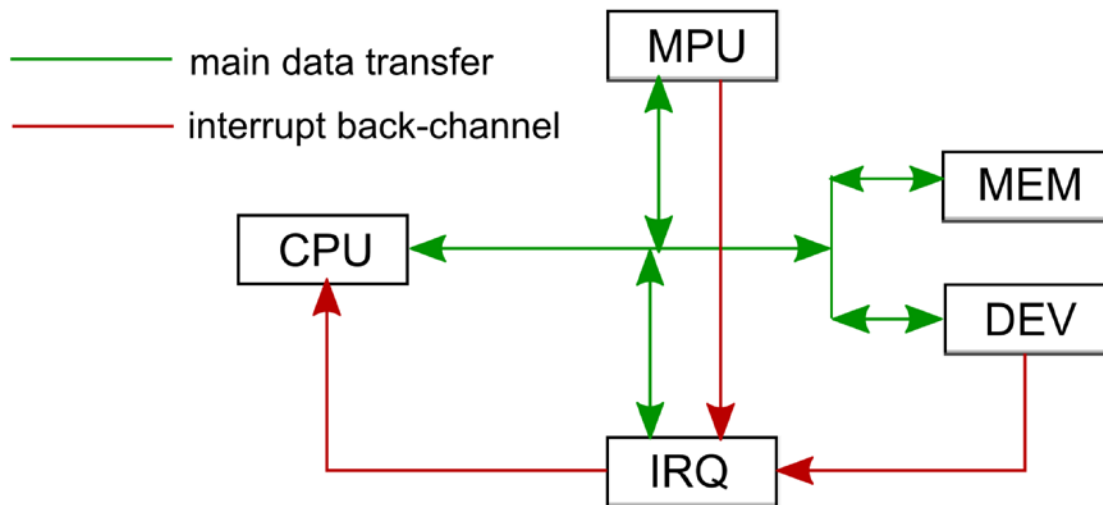
Focus

- **Limited the scope to requirements involving interrupt handling.**
 - the behaviour, timing and unpredictability of interrupts meant testing could be difficult.
- **Two case studies**
 - Lero@TCD: using process algebra to model essential concurrency in the system (high-level)
 - Airbus DS: using the Frama-C tool to verify selected XtratuM hypercalls (low level)
- **Also some exploratory high-level modelling of Requirements with an emphasis on key Data Invariants**

Unavoidable Concurrency

- even with Single-Core!

- In a single-core system, the CPU is time-shared between the hypervisor and partition code with no parallelism.
- Flow control change is managed via traps (exceptions, interrupts, ...).
- However, we have an essentially concurrent system



- CPU executes instructions in a sequential manner on behalf of either the kernel or partition
- Memory (MEM) responds to CPU memory requests
- The MMU/MPU observes the bus traffic, raises memory fault interrupts when appropriate
- IO Devices (DEV) signal via interrupt when done
- Interrupt request hardware (IRQ) takes in interrupt requests from MMU/MPU/DEV/CPU and forwards the highest in priority to the CPU

Concurrency Tool Support

(with Kevin Hennessy, TCD)

- **Notation: Communicating Sequential Processes (CSP)**
- **Tool: FDR3 (Failures/Divergences Refinement).**
- **Models:**
 - Concurrent Hardware behaviour
 - Kernel and Partition Software Behaviour
 - Requirements of interest.
- **Analysis:**
 - FDR3 is a Model (Refinement) Checker
 - Deadlock can be represented as a refinement property.
 - Model Requirements as behaviour that deadlocks if violated.

CSP Model of MMU

```
MMU(blocked)
= tick -> ( TRANSFER(blocked)
            [] BLOCK_MEM_REGION(blocked)
            [] UNBLOCK_MEM_REGION(blocked)
            [] MMU(blocked) )

TRANSFER(blocked)
= bus?dir?addr -> ( if member(addr, blocked)
                   then ( badaccess -> raise!memfault -> MMU(blocked) )
                   else ( mmuOK -> MMU(blocked) )
                   )

BLOCK_MEM_REGION(blocked)
= mmuBlock?a -> MMU(union(blocked, {a}))

UNBLOCK_MEM_REGION(blocked)
= mmuUnBlock?a -> MMU(diff(blocked, {a}))
```

- Synchronise with global clock (`tick`)
- Accepts bus read or write (`bus?dir?addr`)
- Permits operation if address not blocked (`blocked, mmuOK`)
- Objects if address is blocked (`blocked, badaccess, raise!memfault`)

Platform as parallel components

```
PLATFORM
= (((CPU(<(kernel, supBitOne)>, 10, true)
  [| {| tick, read, bus, write |} |]
  MEM_BUS (mem_init))
  [| {| tick, bus, badaccess, mmuOK, mmuBlock, mmuUnBlock |} |]
  MMU(mmu_init))
  [| {| tick, devStart, devEnd |} |]
  DEVICES)
  [| {| noPendingIRQs, raise, irqForward, devStart, devRaise, devEnd |} |]
  IRQ(interruptVector))
  [| {| tick |} |]
  CLOCK

assert PLATFORM :[deadlock free [F]]
```

- Components run in parallel with common events on which they must agree
- The platform model should be deadlock-free
- Kernel and requirement models in parallel with the platform should also be deadlock-free

Expected Benefits:

- **Help understand the relevant interactions between the relevant fragments of kernel code.**
 - Want to determine the appropriate pre- and post-conditions.
- **The validated CSP models will give a high-level “shape” to specifications required at a lower level for code verification.**
- **The FDR3 tool can generate graphs showing the extent of non-deterministic interleaving of events**
 - Can assist in ensuring full coverage by both formal and test-based verification techniques.

Challenges:

- **CSP models very quickly get too large for the tools to handle.**
 - FDR3 supports model-checking in the cloud to help mitigate this problem
- **Very careful abstraction is required to minimise state size, while retaining modelling accuracy.**
- **Hardware model needs to be configurable w.r.t. requirements so only the state specific to a requirement is used.**

Verifying Kernel Code

(supporting Alexandre Cortier, ADS)

- **Notation: ANSI C Specification Language (ACSL)**
- **Tool: Frama-C**
- **Approach:**
 - Property annotations (ACSL) using special comments
 - Pre/Post Conditions, Invariants (Data and Loop)
- **Analysis:**
 - Weakest Precondition Analysis
 - Semi-automatic entailment checking.
 - Full theorem prover available if required.

Hypercalls

- used by partitions to call for kernel services.
 - invoked using a TRAP instruction
 - kernel-installed handler is executed,
 - Handler checks for the appropriate permission before delivering the requested service.
- Focus on two hypercalls in the XtratuM sources:
 - `SuspendPartition`
 - `SwitchSchedulePlan` ... we shall look at this one

Defining the hypercall table entry

```
#define HYPERCALLR_TAB(_hc, _args) \  
    __asm__ (".section .hypercallstab, \"a\"\\n\\t\" \  
            \".align 4\\n\\t\" \  
            \".long \"#_hc\"\\n\\t\" \  
            \".previous\\n\\t\" \  
            \".section .hypercallflagstab, \"a\"\\n\\t\" \  
            \".long (0x80000000|\"#_args\")\\n\\t\" \  
            \".previous\\n\\t\"")  
  
// Hypercall table  
HYPERCALL_TAB(MulticallSys, 0); // 0  
...  
HYPERCALL_TAB(SuspendPartitionSys, 1); // 2  
...  
HYPERCALLR_TAB(SwitchSchedPlanSys, 2); // 27  
...
```

Argument Checking

```
static inline xm_s32_t CheckGParam( void *param, xmSize_t size
                                   , xm_u32_t alignment
                                   , xm_s32_t flags)
{
    if (!(flags&PFLAG_NOT_NULL)&&!param)
        return 0;
    if ( ( ((xmAddress_t)param >= CONFIG_XM_OFFSET)
         &&
         ((xmAddress_t)param<(CONFIG_XM_OFFSET+1*16*1024*1024))
         )
        ||
        ( ((xmAddress_t)param+size >= CONFIG_XM_OFFSET)
         &&
         (((xmAddress_t)param+size)<(CONFIG_XM_OFFSET+1*16*1024*1024))
         )
        )
        return -1;
    return 1;
}
```

Formalisation challenges

➤ We need to identify:

1. What these hypercalls actually do
2. What data they need to perform their task
3. Where and how they actually get invoked.





➤ Things we didn't have














- Design documentation (architecture, data type design and functional call graphs)
- Access to the kernel developer expertise regarding the code and its interdependencies.

➤ We did make some progress

Results

Icons for properties:

-  No proof attempted.
-  The property has not been validated.
-  The property is *valid* but has dependencies.
-  The property and *all* its dependencies are *valid*.

Module	Goal	Model	Qed	Alt-Ergo	Coq	Why
CheckGParam_0	Complete behaviors 'param_ok', 'param_addr_error', 'param_error'	Typed	—			
CheckGParam_0	Disjoint behaviors 'param_ok', 'param_addr_error', 'param_error'	Typed	—			
CheckGParam_0 for param_addr_error:	Post-condition for 'param_addr_error'	Typed	—			
CheckGParam_0 for param_error:	Post-condition for 'param_error'	Typed				
CheckGParam_0 for param_ok:	Post-condition for 'param_ok'	Typed	—			
SwitchSchedPlan	Post-condition	Typed	—			
SwitchSchedPlan	Post-condition	Typed				
SwitchSchedPlanSys	Complete behaviors 'switch_ok', 'no_action', 'error_invalid_parama', 'error_perm'	Typed	—			
SwitchSchedPlanSys	Disjoint behaviors 'switch_ok', 'no_action', 'error_invalid_parama', 'error_perm'	Typed	—			
SwitchSchedPlanSys for error_invalid_parama:	Post-condition for 'error_invalid_parama'	Typed	—			
SwitchSchedPlanSys for error_perm:	Post-condition for 'error_perm'	Typed	—			
SwitchSchedPlanSys for no_action:	Post-condition for 'no_action'	Typed	—			
SwitchSchedPlanSys for switch_ok:	Post-condition for 'switch_ok'	Typed	—			

Unexpected challenge

- A number of baseline requirements were selected in advance as possible candidates for formal verification
 - Criterion: they looked like they might be hard to test.
 - These led to the choice of hypercalls that were looked at in detail.
- The parallel testing activity uncovered a requirement not in the above list that proved hard to test
 - *PK-9: When a partition is resumed by the partitioning kernel at the beginning of its timeslots, the said partition shall restart in the same memory context (memory allocated to the partition), CPU core registers context and FPU context (if FPU is used by the considered partition).*
- Targeting the formal verification of this requirement would be an interesting next step.

Results

- **Small consultative and exploratory activity**
- **High-level modelling helps to “frame” the verification task**
 - Exposing essential concurrency, while avoiding that which is irrelevant
- **Low level annotations can work**
 - But connection to top-level is non-trivial
 - Need a semantic bridge between C data and kernel concepts
 - This requires the modelling of the kernel Data-Invariants at both the Requirements and Code levels.
 - Really need “domain experts” involved
 - the software developers !

Possible Future Investigations

➤ Kernel-related

- Formal Model of Requirements Baseline
 - Formal Model of Data Invariant
- Investigate requirements-to-code verification of PK-9 Context Switching
 - requires top-to-toe infrastructure, which can be re-used for other requirements

➤ General On-Board Software

- High-level Formal Models to verify consistency and other desired semantic properties
 - This is the “sweet-spot” for current formal method techniques
- Reference Architectures/Specifications/Data Models/Interface Standardisation/Protocols

Any Questions?

- **Andrew Butterfield**
Lero @ Trinity College Dublin
Andrew.Butterfield@lero.ie
- **ESA TO: Maria Hernek**
Maria.Hernek@esa.int
- **Mark Hann**
SCISYS UK Ltd
mark.hann@scisys.co.uk
- **Fabrice Cros, Alexandre Cortier**
Airbus Defence and Space
Fabrice.CROS@airbus.com
Alexandre.CORTIER@airbus.com