Advances in T-EMU

Software debugging, bus models and SMP2 improvements

TERMA

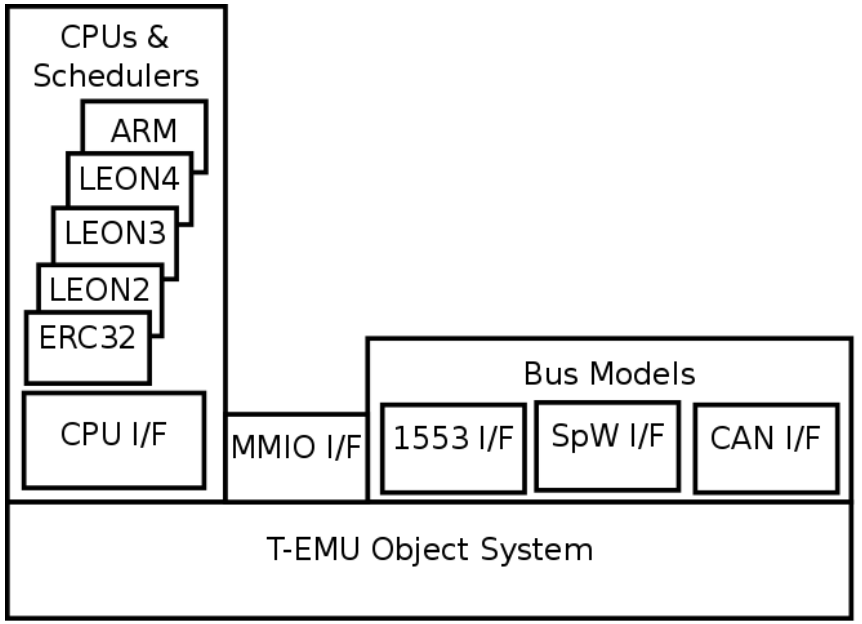
ALLIES IN INNOVATION

Presenters and Authors

- Dr. Mattias Holm maho@terma.com
- Alberto Ferrazi albf@terma.com
- Joshua Whitty jowh@terma.com
- Mathilde Maury mmau@terma.com

T-EMU

- Full system simulation framework and CPU emulator
- Centred around a set of high performance CPU emulation cores
  - SPARCv8: ERC32, LEON2, LEON3 and LEON4 models
- Multi-core support
- Device models
  - MEC, LEON2-on-chip-devices, GRLIB models, …
- Advanced and easy to use object system based C-API
  - Classes, interfaces, properties
  - Checkpointing (simulation breakpoint) support
  - External (non-native) model integration support (SMP2, System-C etc)
- Library based design
  - Can run stand alone or be bundled as part of existing simulator infrastructure
- Command Line Interface is first class citizen for advanced automation
- T-EMU 2.2 will be formally released within a few months
  - Nightly builds are already available



T-EMU 2.2 Improvements

- Performance
  - 22 % performance improvement since T-EMU 2.1 increasing from 90 MIPS to 110 MIPS when running Dhrystone.
- Configurations
  - Additional standard configurations (UT700, NGMP etc)
- Architectures
  - Adding ARMv7-R support
- Features
  - Components
  - Bus models (adding 1553, CAN, SpW, Ethernet)
  - OBSW source level debugging
  - Asynchronous events (socket and wall clock timer events)

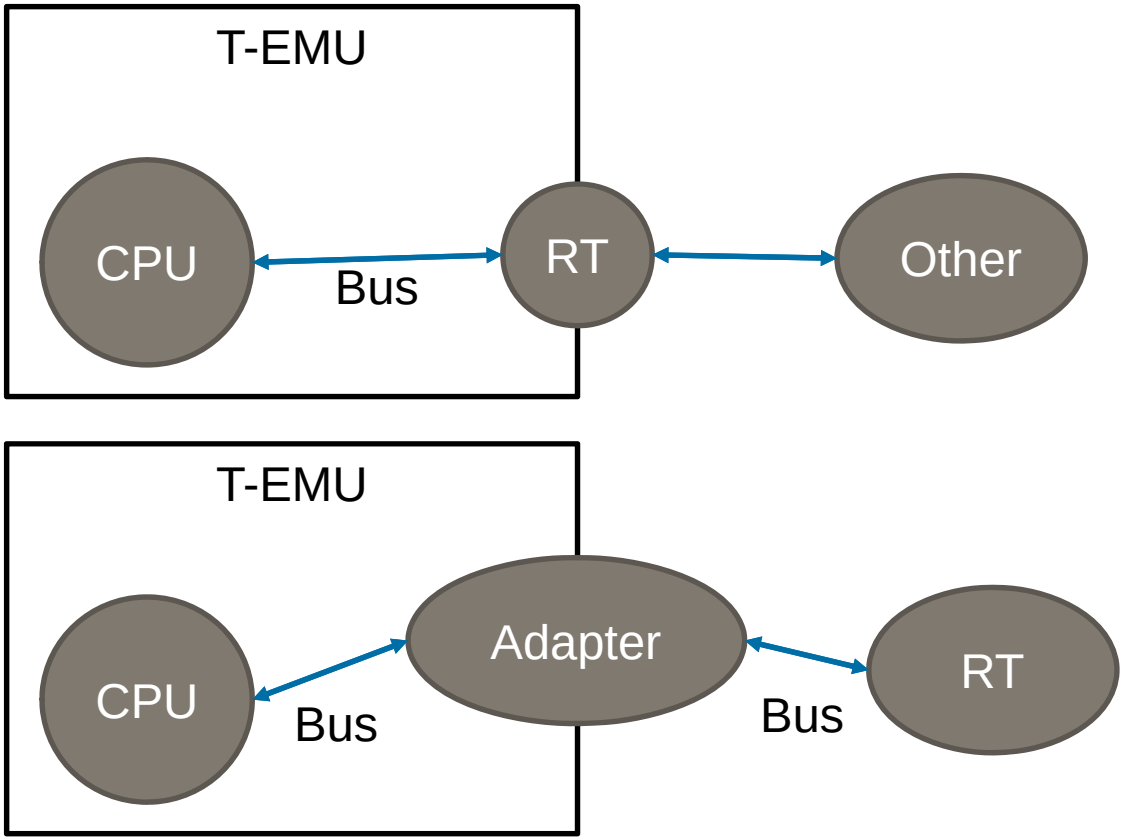
Components

- T-EMU object system extended with component objects
  - Collections of sub components and objects
  - API simplifies time source assignment, creation of objects with unique names etc.
  - Components will completely replace the machine interface in due time.
  - Simplifies creation of complex multi-object devices (e.g. UT700)
  - Allows delegation of interfaces and properties

```
# Used to be 150 line script:
t-emu> object-create    \
  class=UT700-Component \
  name=ut700_0
  args="ramsize:8MiB"
```

Bus Models

- T-EMU provides a set of standard built in bus models
  - Serial
  - CAN
  - 1553
  - Ethernet
  - SpaceWire
- Fully virtual
- Transaction based
- High performance
- Easy to tap
- Note, these models are optional
- Easy to setup (unit test for drivers)



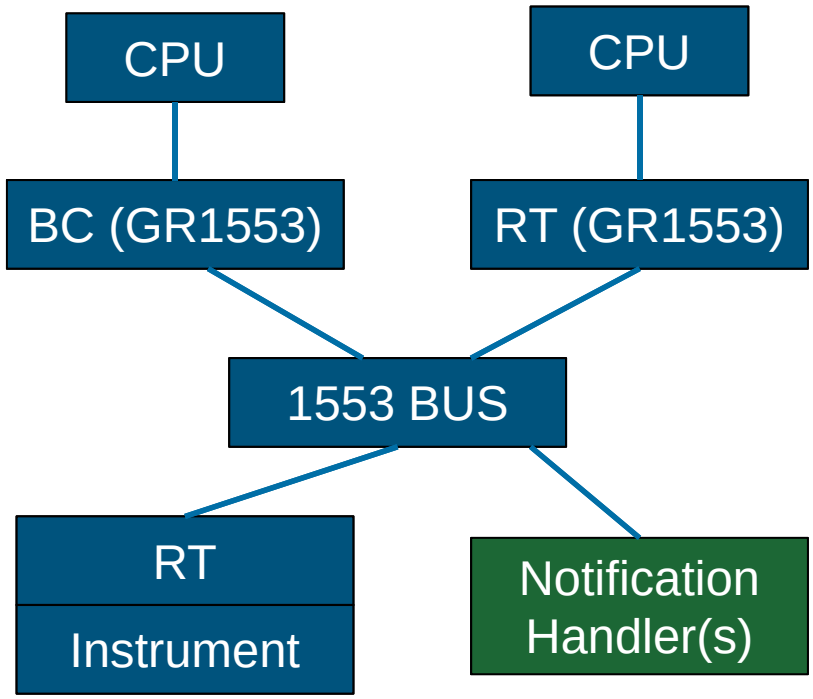
CAN

- CAN bus and device interface

- CAN bus model
  - Multi node buses need models.
  - Simple CAN model provided.
  - User can implement their own CAN bus model using the bus interface.
  - Optional notifications on every sent frame.
  - Can emit bit rate notifications.
- CAN controllers
  - CAN_OC
  - Others added if needed

MIL-STD-1553

- Similar to CAN (multi-node bus with bus and device interfaces)
- Bus interface can be implemented by custom bus models
- Device interface implemented by remote terminals and bus controllers
- Notifications can be emitted on every sent frame (can modify message in flight).
- Separate command and data phases) to support both BC-RT and RT-RT transactions
- Bundled bus model provides error detection and recovery logic
- Bit rate notifications (user can request when to emit these and tie emission to e.g. OBSW minor cycles or PPS device).



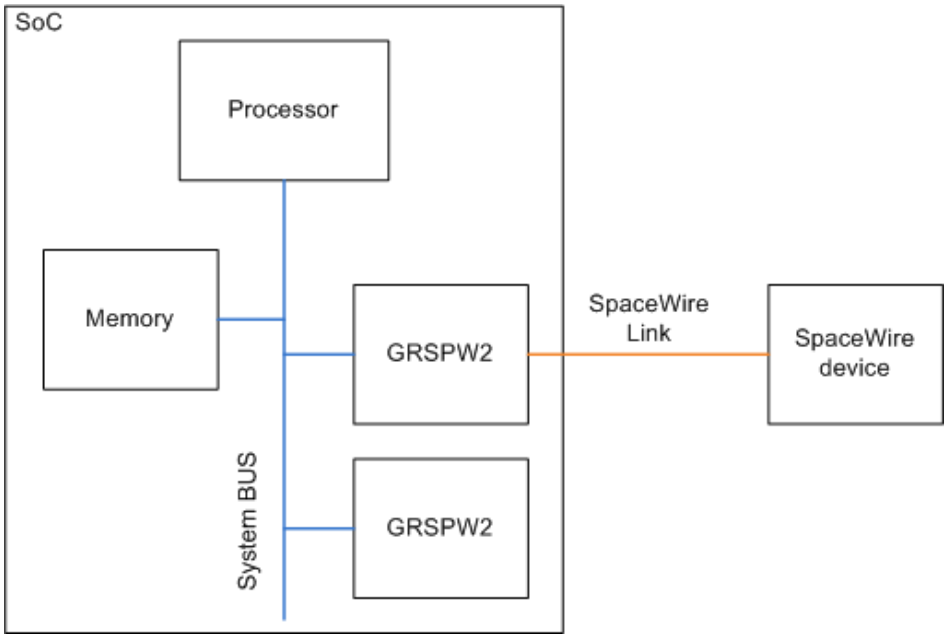
SpaceWire for T-EMU

- SpaceWire is a standard for links and network often used to connect devices on a spacecraft
  - Serial
  - Full duplex
  - Application data transferred as discrete packets
  - 2MB/s - 200MB/s

- T-EMU 2.2 adds SpaceWire support through:
  - Models
    - Grspw2
    - SpaceWire Router
  - SpaceWire API

SpaceWire for T-EMU: Grspw2

- Allows to connect the system bus to a SpaceWire network.
- Implements all major features of Grspw2 except dual port (to be added soon)
  - Link state
  - 4 DMA channels that receive and transmit basing on send lists
  - RMAP

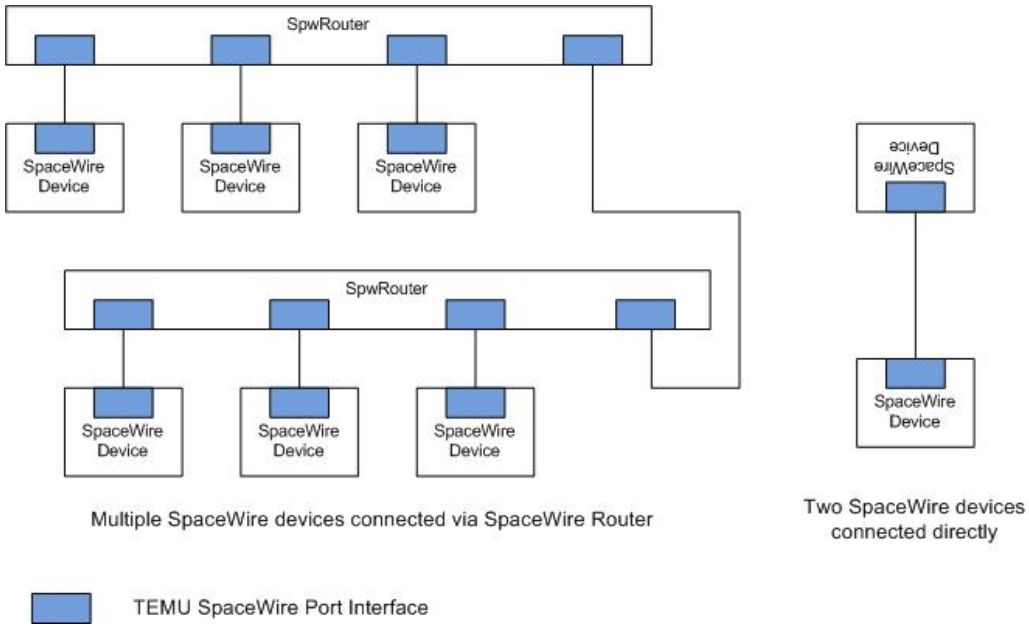


SpaceWire for T-EMU: Router

- SpaceWire is a point to point protocol using wormhole routing: to connect multiple device together a router is required

- T-EMU bus package includes a simple SpaceWire router model that allows creating complex networks
  - Simple: forward the packet basing on the destination address
  - The destination address to port forwarding mapping can be configured via T-EMU API (not via RMAP)

- Two SpaceWire devices can be hot plugged using two T-EMU commands, connect and disconnect



SpaceWire for T-EMU: SpaceWire API

- Users can easily create their custom SpaceWire-enabled models using the T-EMU SpaceWire API
  - Set of components and interfaces tuned for high performance
  - RMAP packets library

- T-EMU defines an interface that both the devices at the two ends of a SpeceWire link have to implement (SpaceWire is full duplex)
  - Send data to the implementing device
  - Allow simulation of the link state machine

- SpaceWire devices usually have multiple ports
  - Make the link between two devices fault tolerant (Grspw2)
  - Form a network (Router)
- The T-EMU concept of interface is used to model ports

SpaceWire for T-EMU: Performance

- T-EMU focuses on performance
  - Message level interface: full messages are passed through interface, not character
  - Control characters are abstracted away

- State machine can be simulated with two methods of the SpaceWire interface:
  - A method that is called by the other device to notify a change in state
  - A method that provides the current state to the other device

- The two methods implementing the state machine has to be provided by the device as the behavior is device dependent.

SpaceWire for T-EMU: Performance

- Ideally, in a packet transmission, the device that originates the packet should allocate it on the stack (or on the heap if too big), send it by passing it to the transfer function and deallocate it when such function returns

- SpaceWire network simulations may involve multiple copies of a transferred packet
  - Packet changes: SpaceWire uses wormhole routing that involves stripping the routing address each time a packet is routed. Packets shrinks while they move through a SpaceWire network
  - Packet buffering: Devices may need to process the packet after the end of the function that delivers it
- There is also a problem of ownership of the packet
  - Routers may deliver a packet to multiple devices (packet distribution) that may buffer it

- To solve all this issues T-EMU uses a copy on write (COW) buffer designed explicitly for the purpose of passing SpaceWire packets through device interfaces
  - Allows address stripping without copying
  - Reference counting is used to free the packet when no device use it anymore
  - Packet is copied only when writing on the packet is needed

Source Level Debugging

- T-EMU already supports GDB RSP for connecting with GDB
- T-EMU 2.2 will support the DWARF debugging format for full source level debugging capability, GDB is then not needed.
- DWARF: Pun on ELF (Executable and Linkable Format) and bacronym of Debugging With Arbitrary Records Format.
- T-EMU uses DWARF to provide CLI based source level debugging of ELF files (Note that SRECs do not contain DWARF info).
- Low level API for programmatic querying of DWARF info will be rolled out.
  - Can be used in programs embedding T-EMU as a library to provide source level debugging capabilities.

```
t-emu> list func=foo
  void foo ( ) {
>* int a = 42;
  }
t emu> step-line
t emu> show var=a
42
```

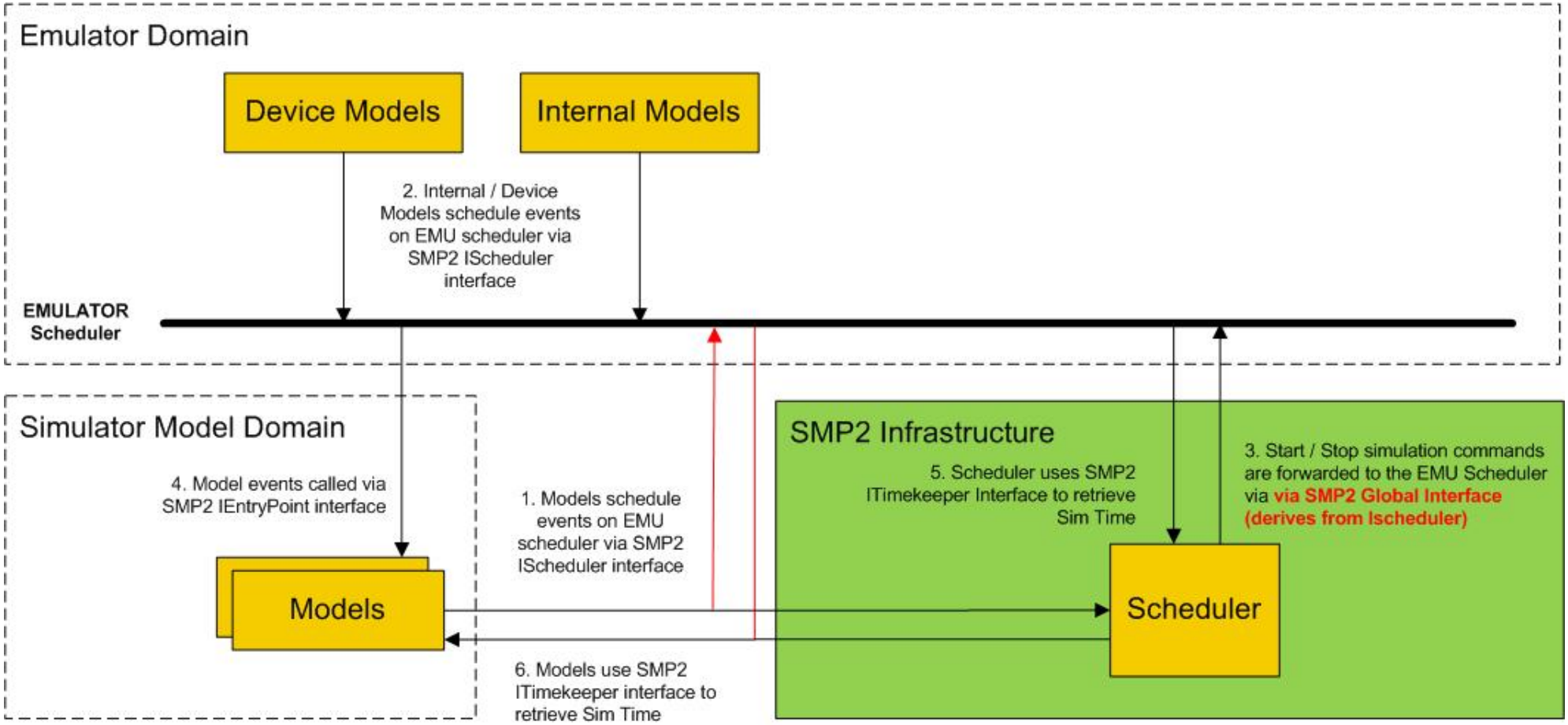
Scheduling

- Topic currently discussed with the SMP2 Working Group
- TERMA's solution to its SVF Simulators and possibly Operational Simulators
- Problems when scheduling models and Emulator sequentially
  - Events need to execute when emulator is running -> Wait or Stop?
  - Time synchronization when 2 schedulers exist -> Hybrid System
  - OBSW detecting interrupts when raised by models
- Different solutions have fixed individual problems but not all, maybe not optimal
  - "Pre" and "Post" notifications before each SMP2 scheduled event to allow emulator to run
  - Device models scheduled on emulator scheduler and others on SMP2 scheduler -> better performance but time synchronisation issues
  - Custom implementation to Stop() the emulator to execute other events

  => "custom" solutions and simulation infrastructure dependent, not portable!
- "One Scheduler" solution, rather than 2
  - Our cases (Ops Sims and SVFs) this scheduler is best placed in the Emulator -> Performance
  - All events in the simulation are executed from the emulator scheduler
  - Supports scheduling with upcoming multicore processors -> events scheduled per CPU / clock domain

  => External Scheduler support using SMP2 and prototype in SIMSAT4

“One Scheduler” SMP2 Supporting an External Scheduler



- The Emulator derives from SMP2 IExternalScheduler / IScheduler interface (retrieved)
- Changes to SMP2 infrastructure Scheduler -> retrieves IExternalScheduler interface
- Red lines -> bypass SMP2 Scheduler in infrastructure (not fully compliant)

“One Scheduler” Supporting an External Scheduler

- Existing SMP2 IScheduler almost supports all functionality needed
  - GetSimulationTime
  - AddSimulationTimeEvent / ModifySimulationTimeEvent
  - Start / Stop scheduler is not supported -> IExternalScheduler published to SMP2 Global Service so simulation infrastructure and Emulator can access
- Solves issues
  - Only one scheduler and one in control of simulation time
  - Eliminates multiple calls to a scheduler to synchronise or change simulation time
  - No need for “pre” or “post” events to catch interrupts raised by models

But it is still a “custom” solution as the Start() and Stop() functions are not part of the SMP2 standard (IScheduler) and therefore this solution is not portable across different simulation infrastructures

Welcome any criticism on the design logic -> prototyped the solution and profiled it with SIMSAT4 on a dummy simulator, will **propose to prototype in ESOC’s SIMULUS Next Generation Study and will be used on TERMA’s future SVFs**

- Multicore processor emulation
  - Existing SMP2 standard solution = proxy has container of IScheduler interfaces, one per processor core so device models can schedule events on their core
  - OR IScheduler interface modified to take CPU ID
- Will be discussed in SMP2 Working Group

Future Work in T-EMU

- Additional architectures
  - RISC V
  - PowerPC
- Simulation standards:
  - SMP2 simulation interface
  - FMI?
  - System-C
  - Co-simulation with all the above

Points of Contact

http://t-emu.terma.com/

Technical: maho@terma.com

Sales: rmp@terma.com

Meet us at

www.terma.com

www.terma.com/press/newsletter

www.linkedin.com/company/terma-a-s

www.twitter.com/terma_global

www.youtube.com/user/TermaTV