**Hardware-in-the-loop for Hardware/Software Co-design of Real-time Embedded Systems**

**Connect your real (sub)system to your model!**

### Why would I want to do that?
- To avoid modeling complex systems when physical implementations of such systems are available.
- To obtain a higher modeling accuracy with less modeling effort.
- To test virtual subsystems with a real testbed.

### Why? We have to establish communication between real and virtual systems.

**Create the hybrid channel**
- Get the hardware interface for your communication type (here Ethernet) and its I/O driver.
- Pick a communication class (analog/digital, parallel/serial) fitting your type of communication.
- Inherit your SystemC channel from the appropriate hybrid channel class and encapsulate the integration to the I/O driver inside.

**Manage output timing**
Decide in which phase to transmit the output values to real subsystems:
- Different types of communication need different timing of output operations.

### Incorporate external events (inputs)
- Use an asynchronous OS thread to get the external event which sets a fast-to-check flag for SystemC. (optional)
- Specify a polling rate.
- Poll the external event at the specified rate from a dedicated SystemC thread and set a SystemC event on detection.
- Rest of the model can use that event.

**Why?**
- Discrete event simulators advance the simulation clock according to the next event in the event queue. If an external event occurs between the current time and the next internal event, it will not be processed until the next internal event.

---

**SystemC uses a discrete event simulator**  
- Simulation clock is advanced in discrete intervals.
- Delta cycles are assumed to consume zero time.

**Why?** Virtual subsystems need to behave according to the real time clock as the real subsystems do.

---

**Domain: Industrial Applications**
- Strict hard real-time constraints
- Data exchange rate 10 KHz achievable with current modeling platforms
- System-level modeling is the new design trend

---

**Patch SystemC kernel for real-time execution**

---

**How does it perform?**

**Performance**
- Round-trip time measurement over Ethernet
  - Polling time is the main factor.
  - Frame size has a linear, mild effect.
  - 1 ms communication cycle feasible with low polling period.

**Determinism**
- Measurments coincide with integration to the I/O driver inside.
- CPU load has minimal effect.
  - Average load <15% at 50 microseconds signal change time with a simple model.
  - There is room for higher performance or more complex model, but jitter is the problem.

**Application settings**
- Set application threads to real-time scheduling and set priorities.
- Extend thread stacks at initialization to avoid page faults during execution.

**Latency hunting**
- Eliminate further sources of latency (corrupt driver, hardware causing unnecessary interrupts etc.)
- Use trace facility of Linux for diagnosis.

---

**Example**
- Interrupt handlers moved to thread context
- Spinlocks made preemptible
- Priority inheritance implemented for semaphores and spinlocks
- Linux timers replaced with higher resolution counterparts
- Turn off power management
- Disable swap memory

---

**RTOS vs. GPOS**
- GPOS: More facilities, I/O interfaces
- RTOS: Deterministic behavior
- Proposed solution: GPOS with real-time improvements (Linux with RT PREEMPT)

**Improvements in OS**
- Patch the Linux kernel with RT PREEMPT.
  - Interrupt handlers moved to thread context
  - Spinlocks made preemptible
  - Priority inheritance implemented for semaphores and spinlocks
  - Linux timers replaced with high resolution counterparts
  - Turn off power management
  - Disable swap memory