A Framework for Generic HW/SW Communication using Remote Method Invocation

Philipp A. Hartmann, Kim Grüttnner, F. Oppenheimer
OFFIS – Institute for Information Technology
Oldenburg, Germany

Philipp Ittershagen, Achim Rettberg
Carl v. Ossietzky University
Oldenburg, Germany

The utilisation of processing cores and shared resources in today’s systems is a key challenge for developing efficient, predictable and correct applications. The SystemC-based OSSS design methodology enables modelling, simulation, and refinement of parallel applications on different levels of abstraction. The underlying platform model explicitly considers shared resources such as buses, shared memories (Figure 1).

On the Application Layer, our object-oriented approach distinguishes active tasks and passive so-called Shared Objects providing a set of well-defined modelling and synchronisation primitives. With the simulation model on the Virtual Target Architecture Layer, detailed task interferences, inter-core communication and bandwidth effects are exposed, and different mappings of application components onto processing units can be explored. Shared Objects can be mapped both to dedicated hardware blocks (with support for automatic synthesis) and shared memory regions.

To close the gap towards a physical implementation, the application model, the platform information, and the obtained mapping is then combined to the final system. Since the task/object communication and synchronisation is based on the OSSS Remote Method Invocation (RMI) protocol, the needed hardware-dependent drivers can be provided in terms of a generic RMI framework, which hides the mapping details from the application and enables direct cross-compilation for the different implementation variants seamlessly.

Figure 1: OSSS Design Methodology

Contact Information
Philipp A. Hartmann <philipp.hartmann@offis.de>
http://www.system-synthesis.org
A Framework for Generic HW/SW Communication using Remote Method Invocation

Philipp A. Hartmann¹, Philipp Ittershagen², Kim Grüttnner¹, Frank Oppenheimer¹, Achim Retterberg²

¹ OFFIS Institute for Information Technology, Oldenburg, Germany
² University of Oldenburg, Germany

Motivation
Software is increasingly important in today’s embedded system design. Both hardware and software architectures have to be considered already in early design phases.

Platform exploration
- Hardware/Software partitioning
- Number of processors / cores
- Interconnect (Point-to-Point, Buses, ...) 
- Memory layout / hierarchy

Software Architecture Exploration
- Mapping of tasks onto processing units
- Task priorities, periods, deadlines, ...
- Selection of underlying RTOS, scheduling policy, ...

Task communication/synchronisation based on common RTOS primitives is difficult and error-prone. 
Goal: Abstraction from platform details enables automatic cross-compilation.

System Modelling & Refinement for Multi-Core Platforms

Remote Method Invocation
1. · Serializing the Arguments of the function being called
   · Arguments are sent via communication channel (bus).
2. · Shared Object is notified about the requested call
   · Client sends its unique client ID along with the method ID to the Shared Object
   · SO scheduler can now arbitrate and grant the call
3. · Arguments are received by the Shared Object and deserialized
   · The method call is performed
4. · A possible return value performs these steps in reverse order

Software Shared Objects
- Shared Object state is stored across RTOS and core boundary in a shared memory block
- OS independent access protocol used for synchronisation
- Execution of Software Shared Object methods in software task context

Inter-core communication model using Shared Objects

Application
OSSS (Oldenburg System Synthesis Subset) is a SystemC-based HW/SW system design flow.

Virtual Target Architecture (VTA) Layer
During refinement, components of Application Layer are mapped to Virtual Target Architecture:
- Software Tasks onto processors
- Hardware modules to dedicated hardware blocks
- Abstract communication links between processes & Shared Objects are mapped to specific interconnect infrastructure, like buses, point-to-point channels, etc, enabling platform exploration.

Communication via Shared Objects
Inter-task communication/synchronisation modelled explicitly via user-defined Shared Objects.
- User-defined, method-based interfaces
- Safer communication mechanism, than common RTOS primitives, like events or mutexes.

Mutual exclusive access ensures consistency.

Guard conditions (guarded methods)
- Boolean conditions, depending on object’s state
- Can block a caller, until condition holds

Method-calls to Shared Objects are (automatically) refined to a Message Passing protocol.

Inter-core communication in different flavours:
- dedicated HW Shared Objects
- inter-core, shared memory
- local (per core) based on RTOS primitives

Automatic translation of Shared Objects to VHDL with synthesis tool FOSSY.

Runtime architecture for generic Shared Object access

Runtime driver architecture
Generic cross-compilation framework for SystemC-based application model
- Software tasks are mapped to RTOS processes
- RMI communication provided by generic driver library

Transparent Shared Object access for the application
- Unified library interface for all supported implementation variants (dedicated hardware, global/firmware shared memory, core-local memory)

Separation of platform-specific details and driver logic
- Mapping information (Type of SO, Address information)
- Available interface functions, argument types and sizes
- Platform description in terms of Device Tree
- Device Tree enables static information access and storage outside the runtime binary

Contact:
Philipp A. Hartmann
philipp.hartmann@offis.de

This work is supported by the COMPLEX project (FP7-247999), co-funded by the European Commission within the 7th Framework Programme.

http://complex.offis.de