Multi-core debugger integration in OSCI SystemC

Peter de Jager

Intel Corporation

accellera SYSTEMS INITIATIVE™

intel

SYSTEMC™ EVOLUTION DAY
OCT 29, 2020 | VIRTUAL WORKSHOP

Copyright Permission

- A non-exclusive, irrevocable, royalty-free copyright permission is granted by **Intel Corporation** to use this material in developing all future revisions and editions of the resulting draft and approved Accellera Systems Initiative **SystemC** standard, and in derivative works based on the standard.

Outline

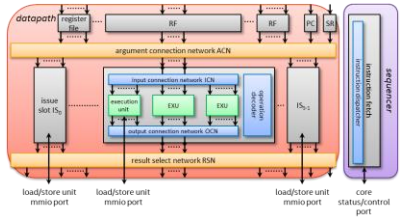
- Background & Motivation
- High-level requirements
- Generic design proposal
- Synchronization
- Synchronization - Implementation 1
- Synchronization - Implementation 2
- Next steps
- Discussion

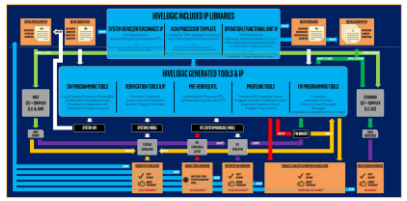
Background & Motivation

- About the author:
  - Located @ Intel Eindhoven, Silicon Hive team
  - Group develops tools (HiveLogic) to create cores and systems
  - Technology has been used in a variety of products for a variety of application domains, including :
    - video coding
    - video post-processing
    - imaging
    - communications

Silicon Hive technology: Four key elements

Design-time configurable processor & system architecture templates supported by elaborate libraries of hand-optimized, fully parameterized processor & peripheral building blocks

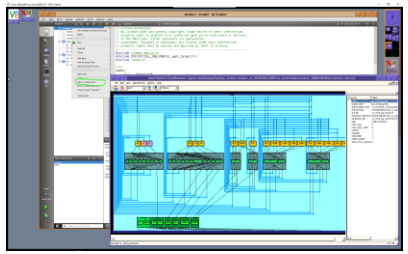


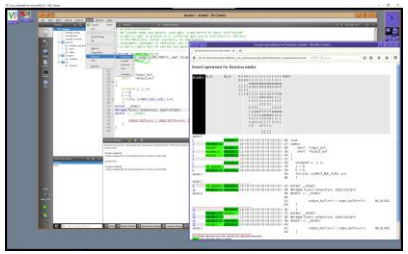


HIVELOGIC

A unique methodology for fast & vast design space exploration at processor and system-level, supported by highly abstract design entry through high-level languages

A fully automated flow and corresponding tools for (multi-) processor & system hardware generation





A fully retargetable programming tool suite based on ANSI-C source entry

Slide courtesy of Jeroen Leijten, Sr. Principal Engineer, Intel Corporation

Background & Motivation

- Products that use our technology are
  - Multi-core
  - Heterogeneous
  - Application-specific
- System-Simulation technology used to be proprietary
  - Supported application-software debug-capabilities
- Moved towards SystemC/TLM
  - Previous debug-solution not applicable anymore
- A generic mechanism to support application-software debugging is not standard available in OSCI SystemC

High-level requirements

Usability (functional)

- Debugger should be able to connect remotely
- Attach/detach to specific core in simulation
  - Attach: core/simulation stops
  - Detach: core/simulation continues

- Support debug-commands (like standard debuggers)
  - step/continue/breakpoints/watchpoints/registers/memories/..
  - Support for user-break to stop current step/continue command

- Debug multiple cores simultaneously
  - via one multi-core debugger and/or multiple independent debuggers

High-level requirements

Reliability (functional)

- Simulation should behave same when debugger is attached

Applicability (functional)

- Solution should not depend on debug-hw being present

Maintainability (scalable)

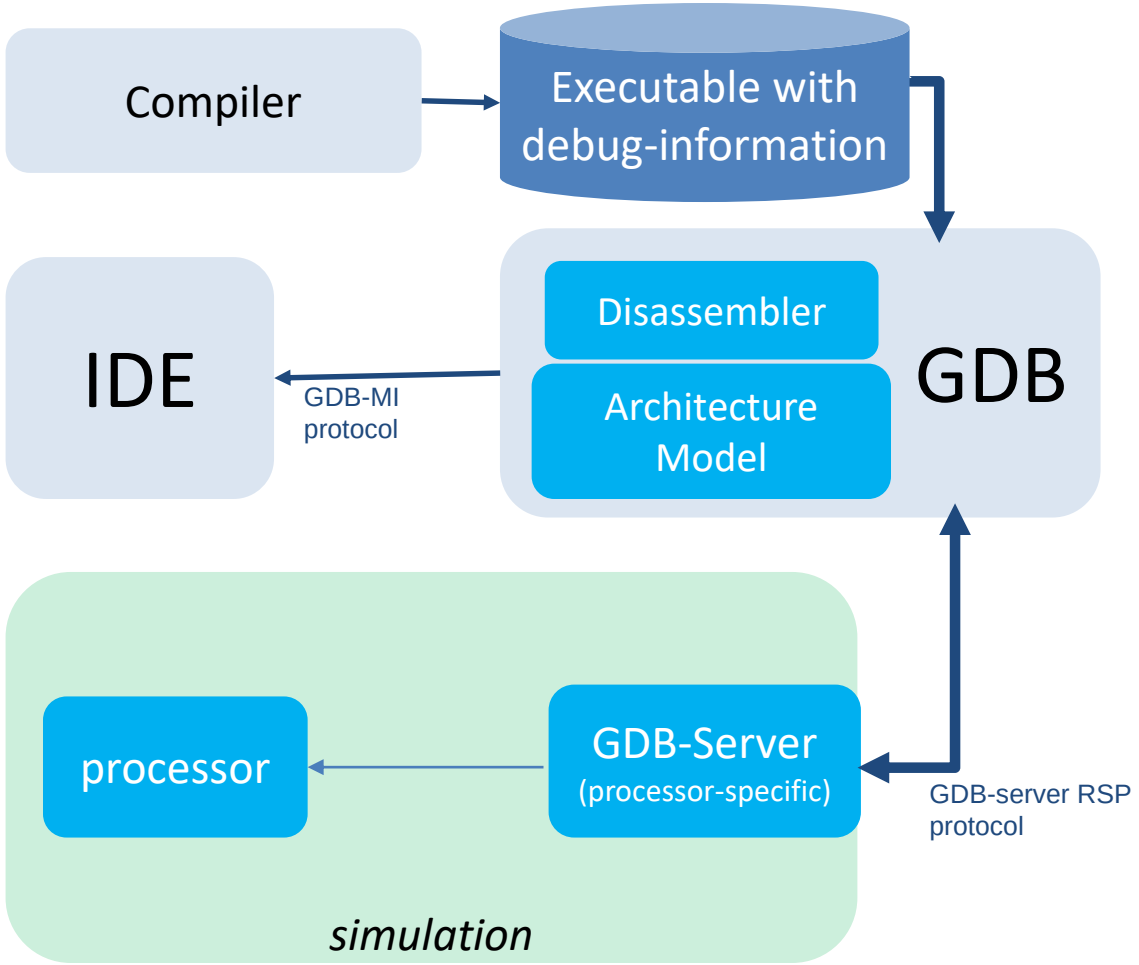
- Debugger & connection to simulation must be core-independent

Generic design proposal

- Create common infrastructure for debug-servers in system-simulation
- Automatically instantiate debug session-server for each core
- Provide a consistent abstraction layer (DebugAPI) to processor models
  - Introspection
  - Run-control

- Propagate architecture information from processor model → debugserver → debugger
  - Use a generic retargetable debugger
  - Use generic DebugServer

Generic design proposal



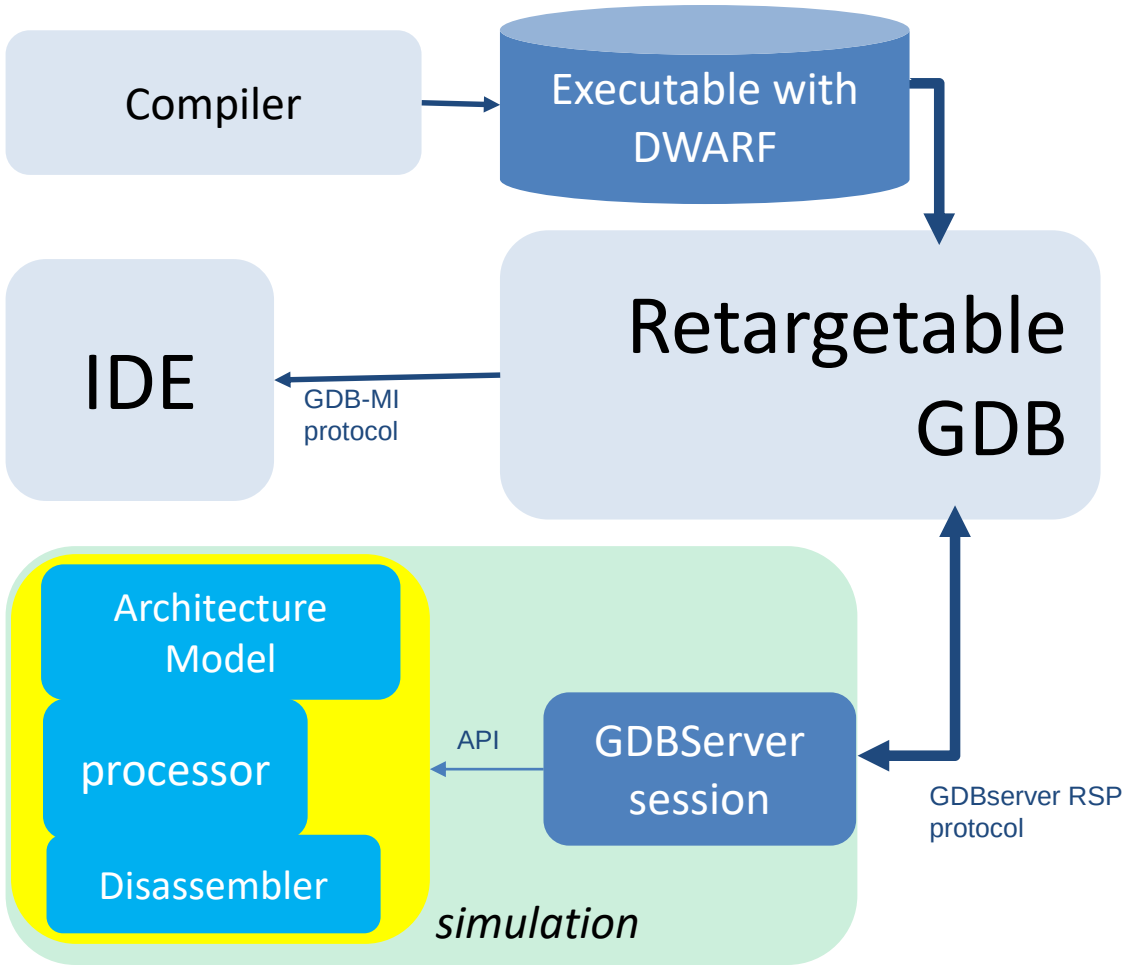
GDB specific

- To interactively debug firmware on a (custom) processor (new for GDB) in a system-simulation **required** to retarget GDB
  - Custom GDB-Server required
- Normally GDB needs to be build including the architecture information

Generic design proposal

- GDB by default supports lots of different cores
  - Architecture information part of source, compiled
- Modifications to GDB required
  - Add runtime retargetability
- GDB ‘remote target’ command used to connect to simulation
  - On connection, GDB receives details of the processor connected to
    - Register-files
    - Memories
    - sizes of c-types

Generic design proposal



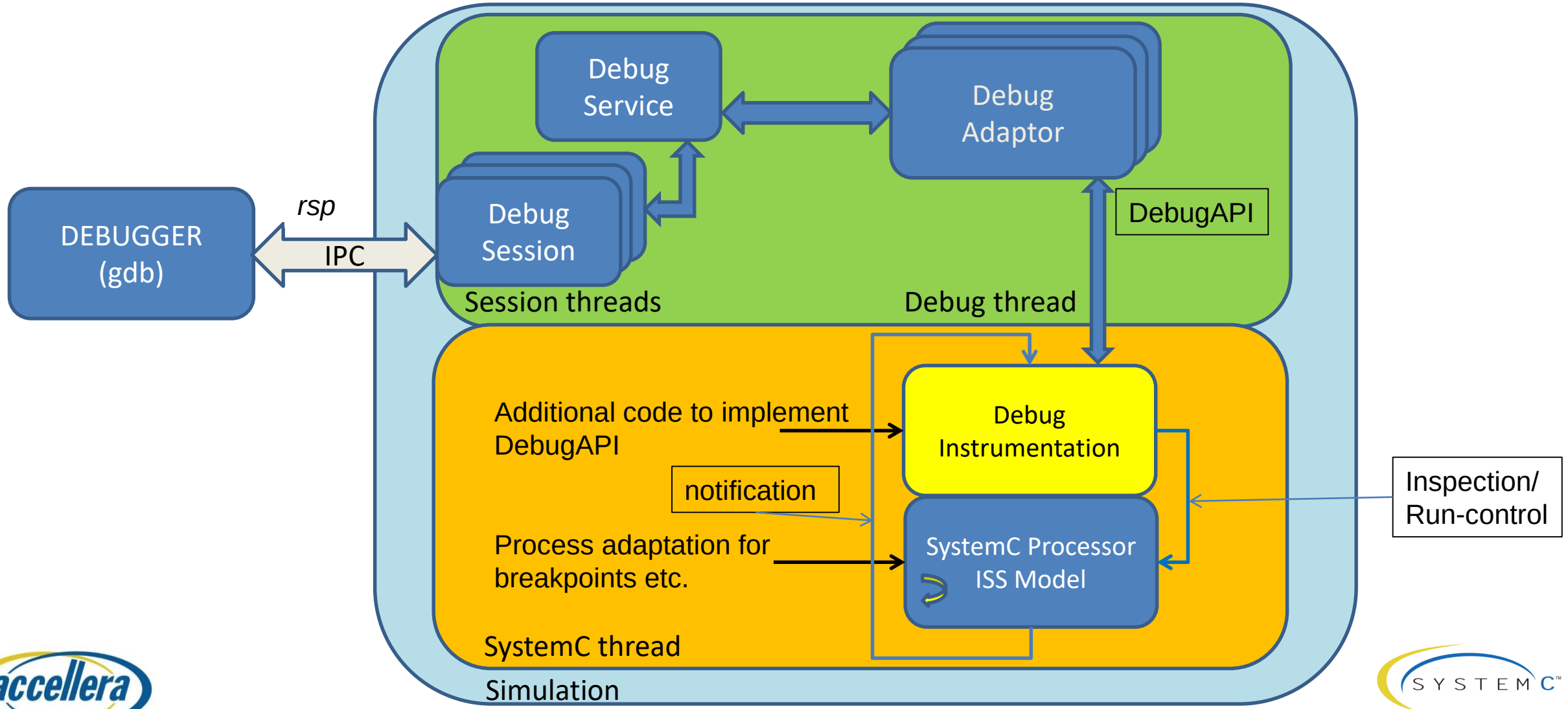
- Retargetable GDB
  - Receives Architecture model upon connecting
- Generic GDBServer using standard DebugAPI to processor

Synchronization

- SystemC simulation and Debug support run in separate threads
- Debugger issues synchronous & asynchronous commands
- Synchronous (simulation is stopped)
  - Inspection: read/write memories/registers
  - Run-control: set breakpoints/watchpoints, step, run_until
- Asynchronous (simulation is running)
  - Attach to simulation
  - User-interrupt

  ➔ thread-safe event (using async_request_update()) required

Synchronization



Synchronization

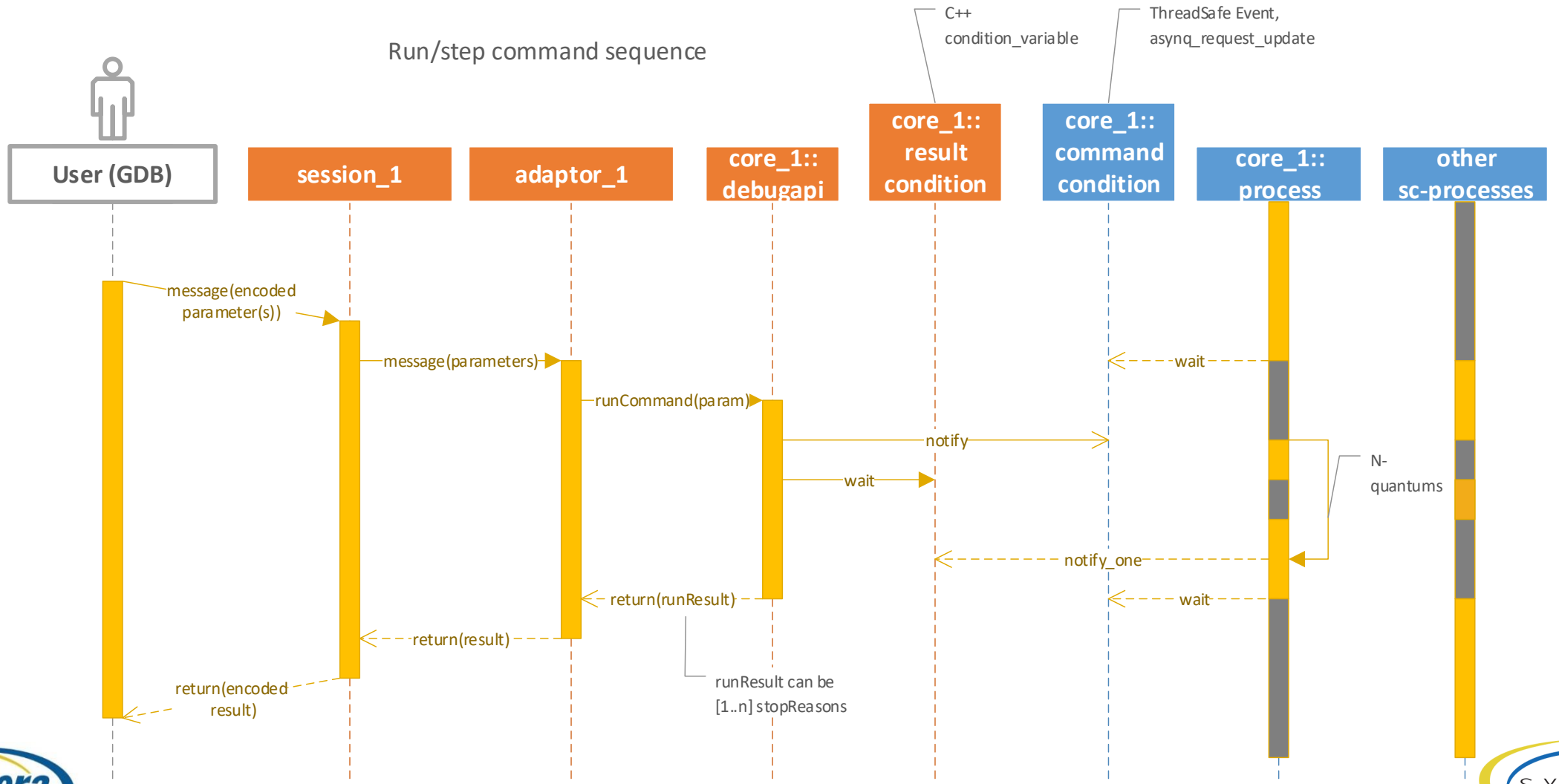
Code for sc_main (replacement for sc_start())

```
if (allowDebug) {
    DebugService::getInstance().createMonitors(dbg_port); // create the sessions
    std::thread debugService(debug_task, &DebugService::getInstance().io_service);
    debugService.detach(); // Do not block execution.
}
std::thread systemSimulation(simulation_task, global_quantum_value); // calls sc_start()
systemSimulation.join(); // wait until simulation finishes
if (allowDebug) {
    debugService::getInstance().io_service.stop(); // cleanup resources
}
```

Synchronization: implementation 1

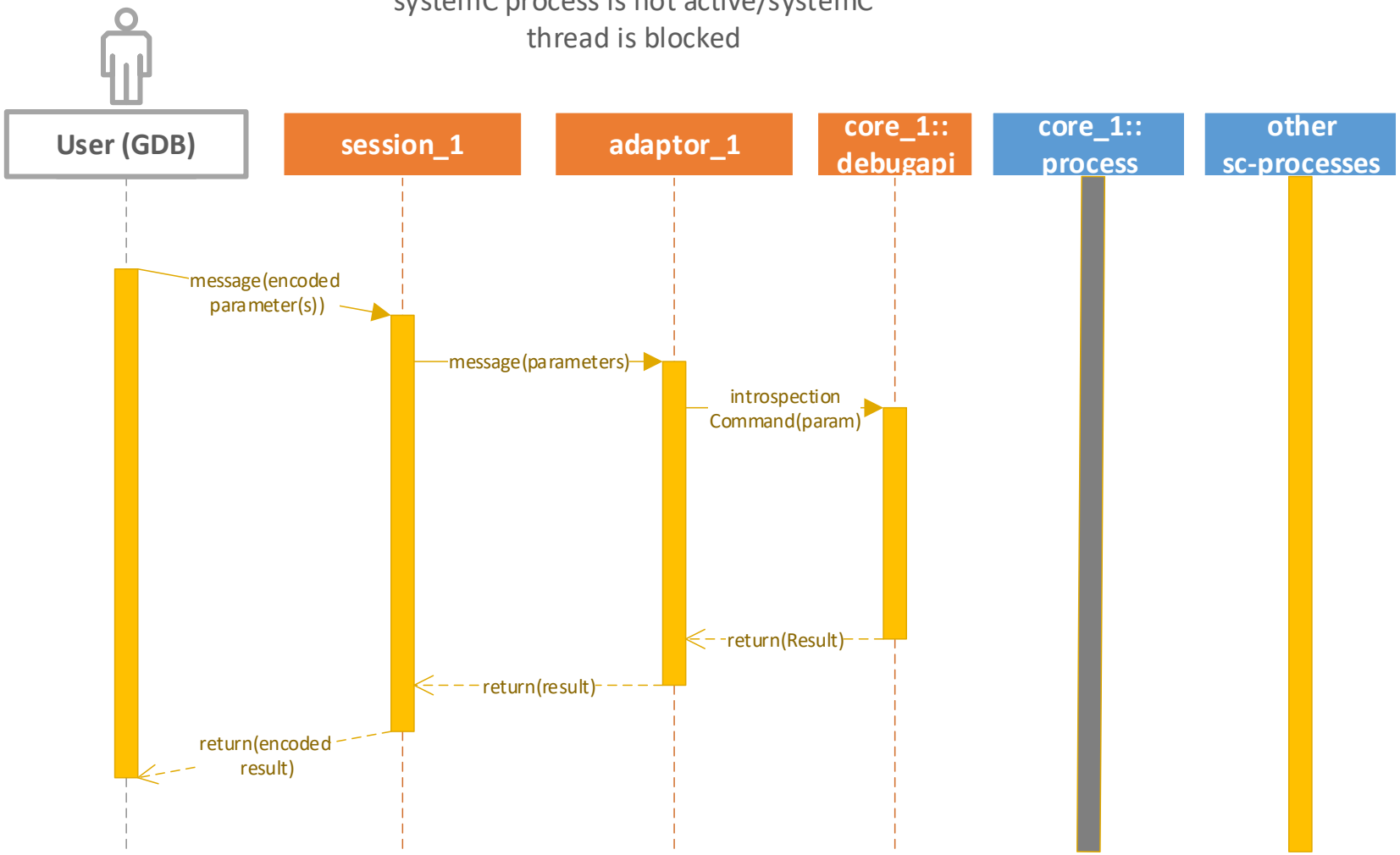
- First implementation
  - Control each core separately
  - Only specific core under debug is stopped (e.g. when breakpoint hit), remaining parts of system continue simulation
  - Behaves like hw without cross-trigger/cross-resume functionality

Synchronization: implementation 1

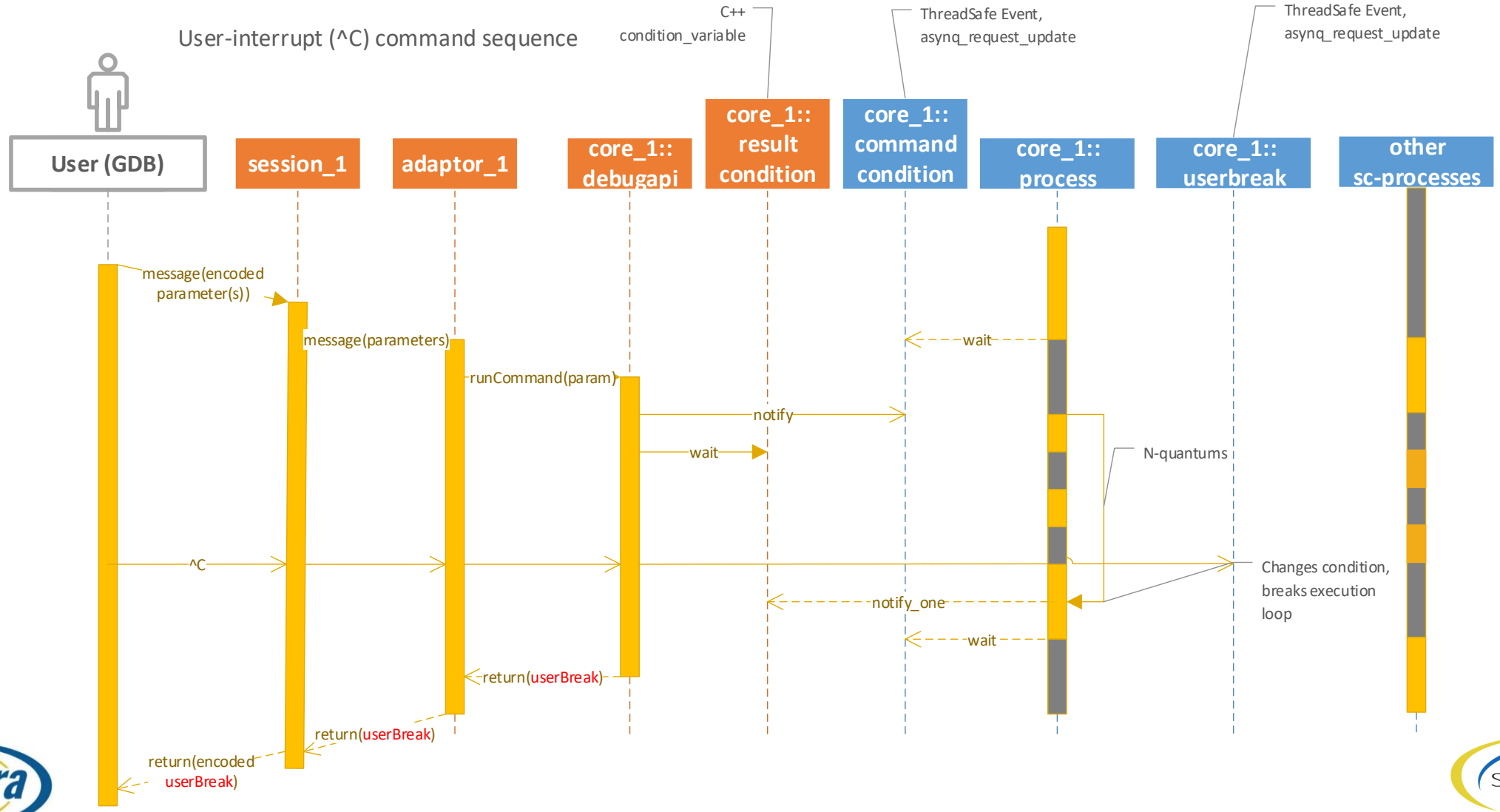


Synchronization: implementation 1

Introspection command sequence:
systemC process is not active/systemC thread is blocked



Synchronization: implementation 1



Synchronization: implementation 1

Results:

- Relatively easy to implement

- User feedback: not happy ☹
  - When multiple cores/initiators are active, state of system is not preserved
  - Issues with reproducibility
  - Restart after break difficult/nearly impossible

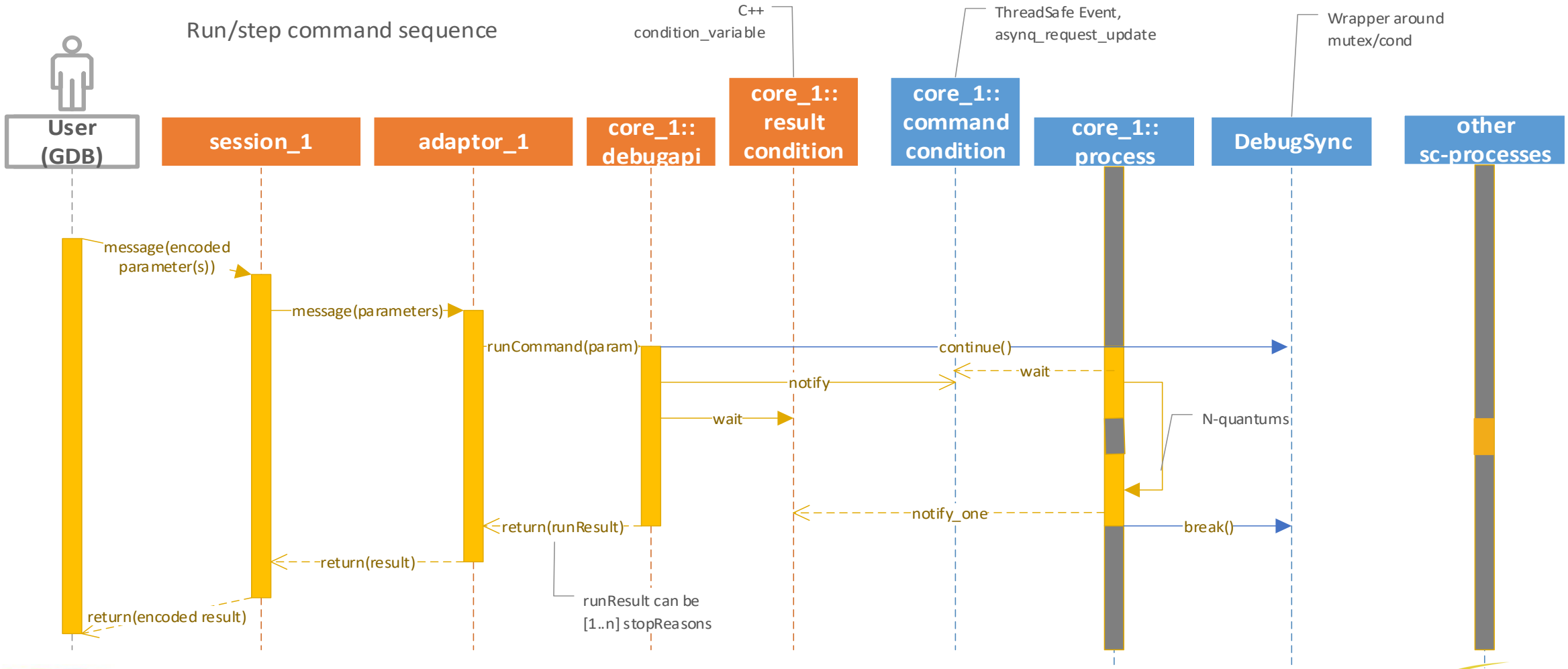
Synchronization: implementation 2

Added requirement of state-preservation

Initial ‘naive’ solution

- Stop complete SystemC thread on stop of a core (bp-hit, step etc.)
  - Lock mutex on stop
  - Unlock on continue

Synchronization: implementation 2

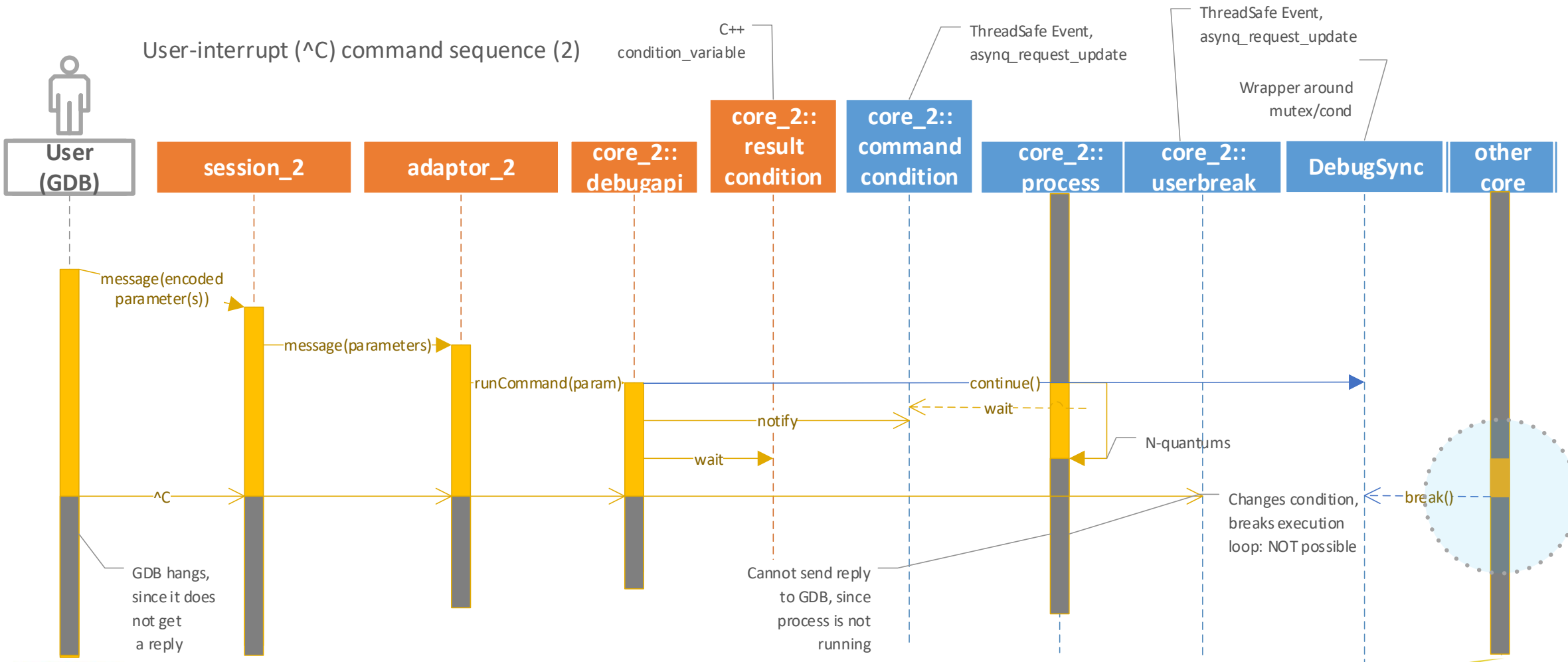


Synchronization: implementation 2

Results:

- Also easy to implement
- User feedback: happier 😐 but not completely
  - State of system is preserved
  - Scenarios are reproducible
  - Breaks async-requests on other cores:
    - Userbreak (^C), attach not possible anymore when 1 core in broken state

Synchronization: implementation 2



Next steps

- Move control on SystemC thread stop/continue into global DebugService

- Keep administration on corestates & debuggers
  - Intercept userbreak when SystemC-thread is already stopped
  - Continue only when all cores in ‘broken’-state have received continue-command

Discussion

- Is this approach worthwhile to standardize, e.g. as part of CCI?
  - https://www.accellera.org/activities/working-groups/systemc-cci
  - The SystemC Configuration, Control and Inspection WG is responsible for developing standards that allow tools to interact with models in order to perform activities such as setup, debug and analysis

- Further questions?