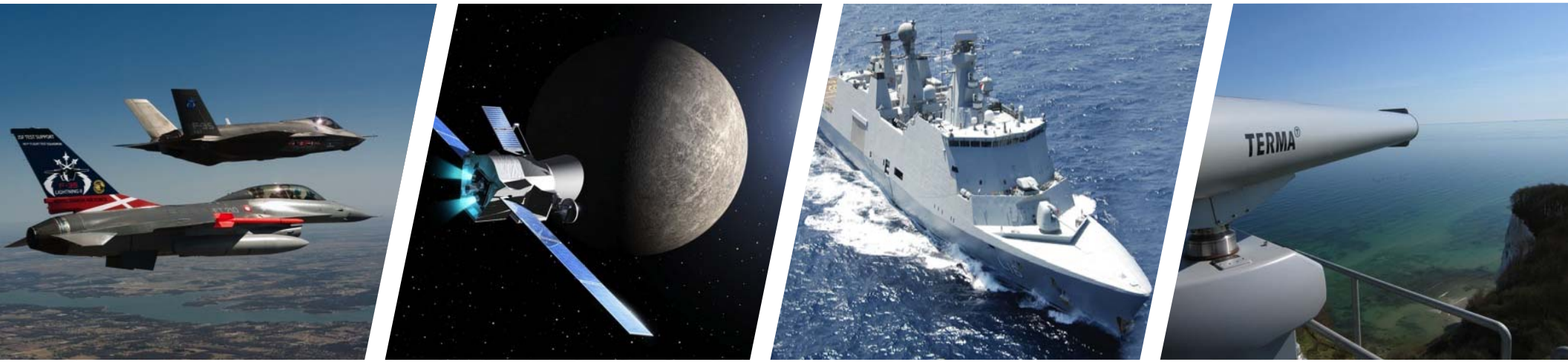
Extension and Validation of Virtual Platform for Complex System-on-Chip and IP-Cores Design for Space

Luca Fossati, Technical Officer
Dr. Mattias Holm, Project Manager
Alberto Ferrazzi, Technical Manager
Mathilde Maury, Software Engineer



Outline

- Introduction
  - Project objectives
  - What is SoCRocket
  - SystemC
  - TLM2.0
- Project
  - Enhancement (models implementation)
    - SpaceWire router
    - GRCAN
    - GR1553B
  - Validation
    - Method
    - Results
  - Analysis
    - Evaluate benefit / effort to merge with upstream version of SoCRocket
    - Design bus-size configurability and split and evaluate the implementation effort
- Conclusion and future work

Introduction: Project objectives

Terma had 3 objectives, they were:

- Analysis
  - Understand the current status of SoCRocket and plan future development
- Enhance: implement new models
  - Allow to simulate more SoC
  - Bus models are particularly important as the SoC are often connected to other systems
- Validation
  - Understand if the platform and Terma enhancements were reliable

Introduction: What is SoCRocket

- Initially developed by TU Braunschweig under ESA contract, development carried on by Terma in the past 3 years under two projects
- It is a framework to assemble custom simulators of SoC (System on Chip) typically used in space
  - A simulator is created by configuring and connecting different models together
  - Each model brings the functionalities of its hardware counterpart
- SoCRocket Purposes
  - Early software development
  - Architecture exploration
- Usage in the hardware design process
  - Early stages, for preliminary verification, before VHDL production/usage
  - VHDL stage to have a reference during modelling
- SoCRocket is composed by
  - Models library (i.e. MSDRAM, L2C, ...)
  - Base classes that provide the principles for interconnecting the models
  - Configuration generator wizard
  - Build/Test execution system

Introduction: SystemC

- Core Library that moves SoCRocket
- Allows to use C++ features to write “executable specifications” (C++ program that exhibits the same behaviour of the emulated system)
- Target digital electronic systems
- Provides
  - Scheduler: runs the simulation
  - Class library that provides the basic blocks to model any kind of system:
    - Models: partitioning of code
    - Processes: implement logic of model
    - Ports: pass data through processes
    - Signals: connect ports

- Supports different level of abstraction from RTL (Register Transfer Level) to Functional

Introduction: TLM2.0

- **TLM = Transaction Level Modelling**
  - Focus is on modelling the transactions on the bus
  - Transactions are modelled as calls to a function
    - Example: ahb.b_transport(dataPayload, delay);
  - Just a set of standard interfaces that have to be implemented by the models
  - Faster then RTL
- **The support to this kind of modelling is provided to SystemC by TLM library**
- **Two coding style that use 2 different transport interfaces**
  - Loosely Timed (LT): Transaction complete with the return of a blocking call. Models are allowed to run ahead of simulation time. Faster but less accurate simulation. Used for software development
  - Approximately Timed (AT): Transaction is modelled with a set of non blocking calls. This allow modelling of phases of the (bus) protocol. Models remains synchronized with simulation time. Better timing accuracy but slower simulation. Used for architecture exploration purpose
- **SoCRocket models must support both these code styles.**
- **SoCRocket provides some base classes that abstract the code styles so the model developer does not have to deal with them in simple cases.**

Introduction: connecting models & transaction

```
24  void MemoryModel::b_transport_implementation(tlm_generic_payload &trans,
25  ··sc_core::sc_time &delay)
26  {
27  ··// Get the transaction address
28  ··uint64_t regAddr = trans.get_address();
29  ··if (registers.fallOnRegisterArea(regAddr))
30  ··{
31  ····registers.write(regAddr, trans.get_data_ptr()); // Perform the action.
32  ····trans.set_response_status(TLM_OK_RESPONSE);
33  ··}
34  ··else
35  ··{
36  ····trans.set_response_status(TLM_ADDRESS_ERROR_RESPONSE);
37  ··}
38
39  ··delay += 1 * clockCycleTime; // Add delay due to this operation.
40  }
```

Implement the transport function.

```
43  SC_MODULE(MemoryModel)
44  {
45  public:
46  ··simple_target_socket<MemoryModel> targetSocket; // Declare a TLM socket.
47
48  ··SC_CTOR(MemoryModel) : clockCycleTime(10, sc_time_unit::SC_NS)
49  ··{
50  ····targetSocket.register_b_transport(this, &MemoryModel::b_transport_implementation);
51  ··}
52
53  ··void b_transport_implementation(tlm_generic_payload &trans, sc_core::sc_time &delay);
54
55  ··sc_time clockCycleTime;
56
57  ··Registers registers;
58  };
```

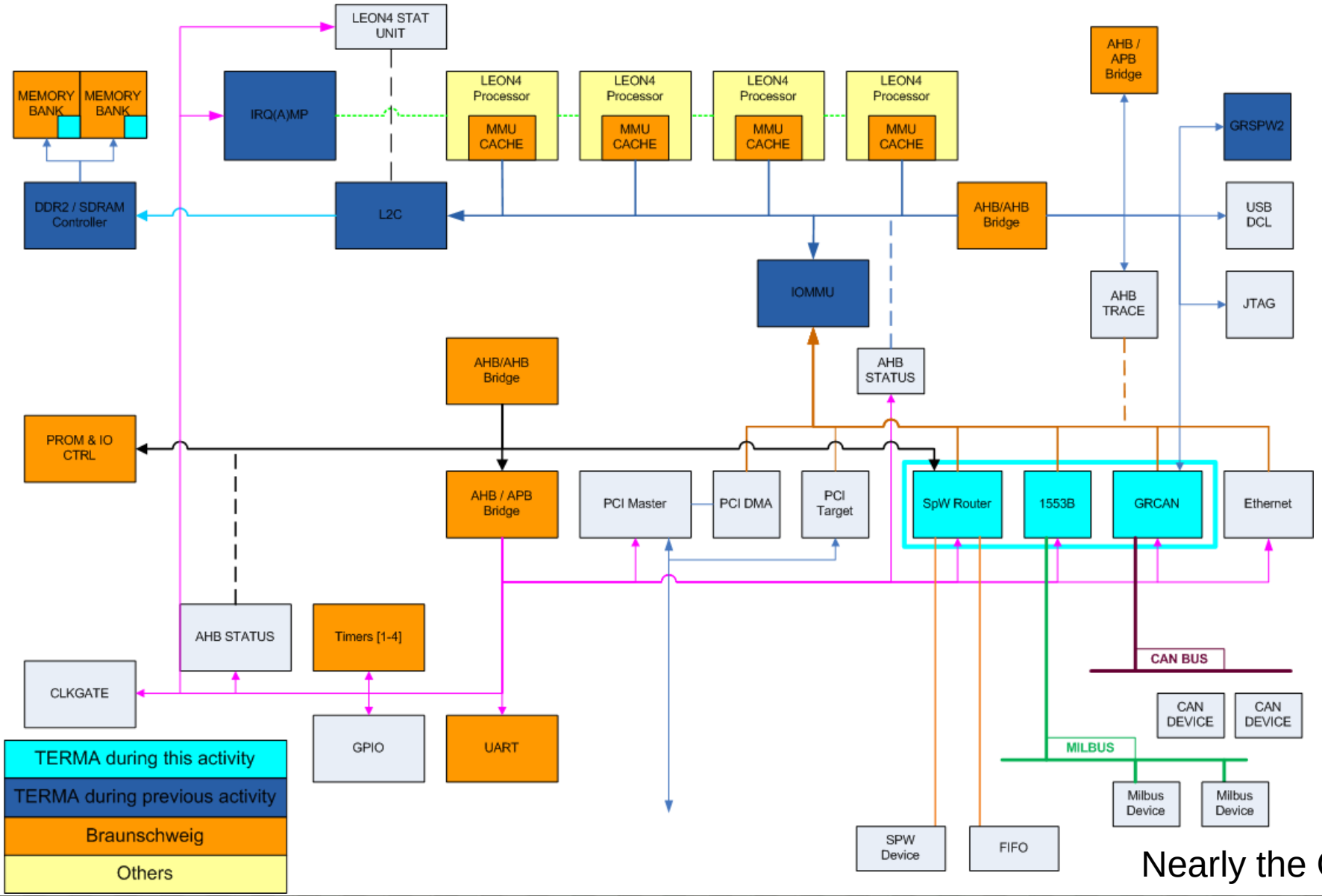
Declare a target socket and register the transport function (b_transport)

```
71  ··initiatorModel.AHB(memoryModel.targetSocket);
```

Bind target and initiator sockets

ENHANCEMENT

Overview



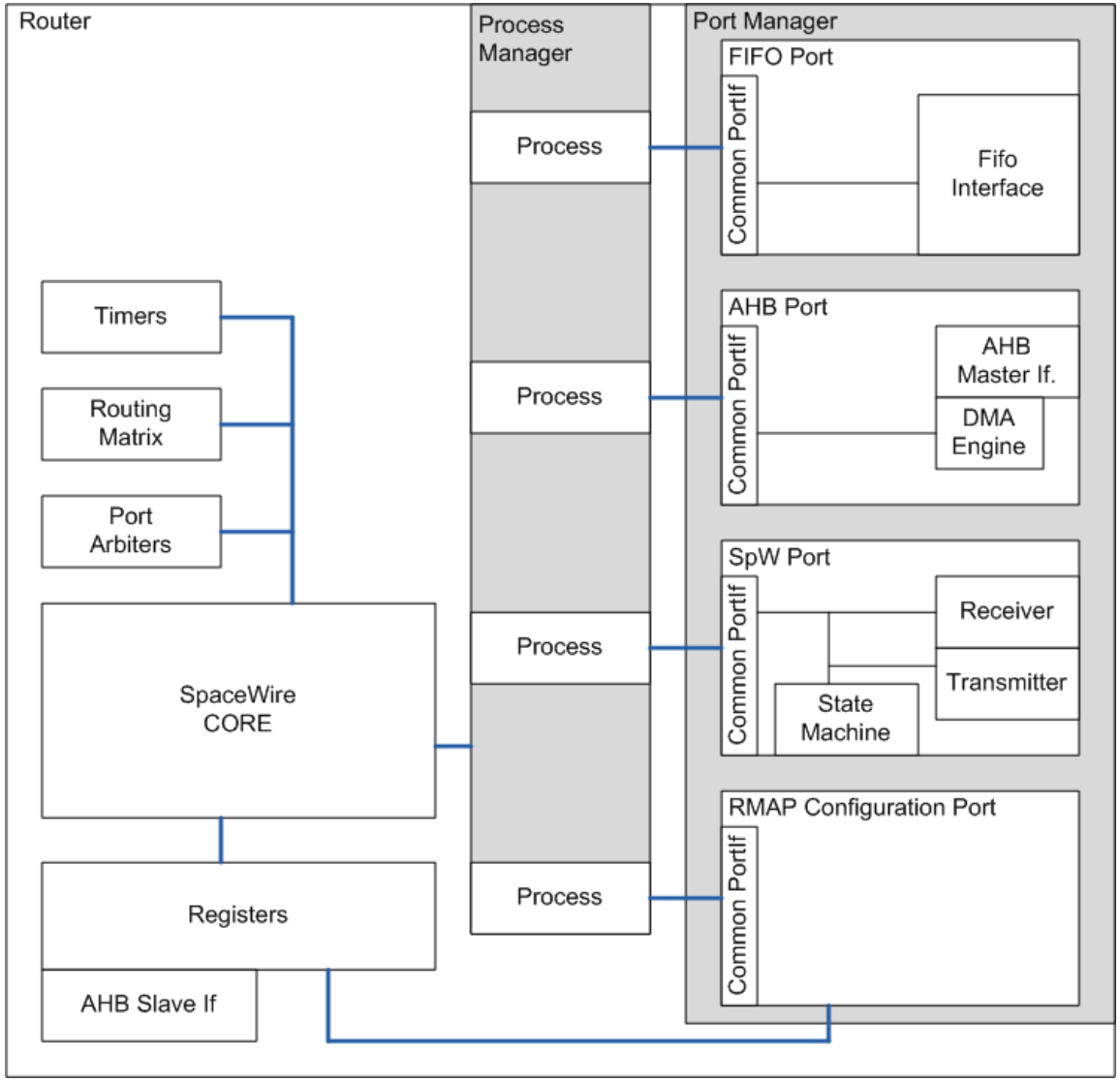
Nearly the GR740 SoC

SpaceWire Router

Features

- The SpaceWire Router models the Gaisler SpaceWire Router
- It allows transfer of packets between any kind of the following ports:
  - SpaceWire Port
  - AHB Port
  - Fifo Port
- The number of ports is configurable, up to 31 in total
- The configuration registers are accessible through:
  - A special RMAP target port at address 0
  - An optional AHB slave interface
- Two routing modes
  - Packet distribution
  - Group adaptive routing
- The model implements the correct prioritization and delays to access ports
- The model implements timers to interrupt blocked transfer that may block a port

SpaceWire Router - Architecture



- There are four types of port modeled in four different classes.
- Port class must implement a common port interface
- Each port has a corresponding process. The process uses the common port interface to transfer data from its own port to the target port
- To each port correspond a port arbiters the manage the queues for accessing the port. A process that desire to transfer to a port will wait for the port available signal
- Core manage interaction between the transfer processes, registers and timers

SpaceWire Router - Architecture

- Internally the SpaceWire router transfer characters which allows for accurate simulation but results in much more complicated code
- To keep the transferring code easy to read and maintainable, it has been organized in “phases”. Examples are:
  - Discard Packet
  - Wait for valid character
  - Wait for port
  - Perform transfer

- Each phase is represented by a function that contains the code to be executed. The code can wait for an event to append or return the next phase to be executed
- The process execute the current phase from which gets the next phase to execute and continue in a cycle

```
79  /// Phase: WaitValidInitialChar.
80  /// Wait that a valid char arrive. This will also remove initial EOP.
81  RoutingProcess::Phase RoutingProcess::PhaseWaitValidInitialChar() {
82    OUTPUTCHAN << name() << "Phase: PhaseWaitValidInitialChar" << v::endl;
83
84    do {
85      if (!m_SourcePort->HasDataTx()) {
86        wait(m_SourcePort->GetTxCharEvent());
87      }
88      uint32_t removedEP = RemoveInitialEP();
89      if (removedEP) {
90        v::warn << name() << "Removed " << removedEP
91                          << " invalid (initial) end of packet char(s)."
92                          << v::endl;
93      }
94    } while (!m_SourcePort->HasDataTx());
95    m_Terminate = false;
96    return Phase(&RoutingProcess::PhasePreRouting);
97  }
```

```
72  void RoutingProcess::Run() {
73    m_CurrentPhase = GetInitialPhase();
74    while (m_CurrentPhase.IsValid()) {
75      m_CurrentPhase = (*this.*m_CurrentPhase())();
76    }
77  }
```

SpaceWire Router - Architecture

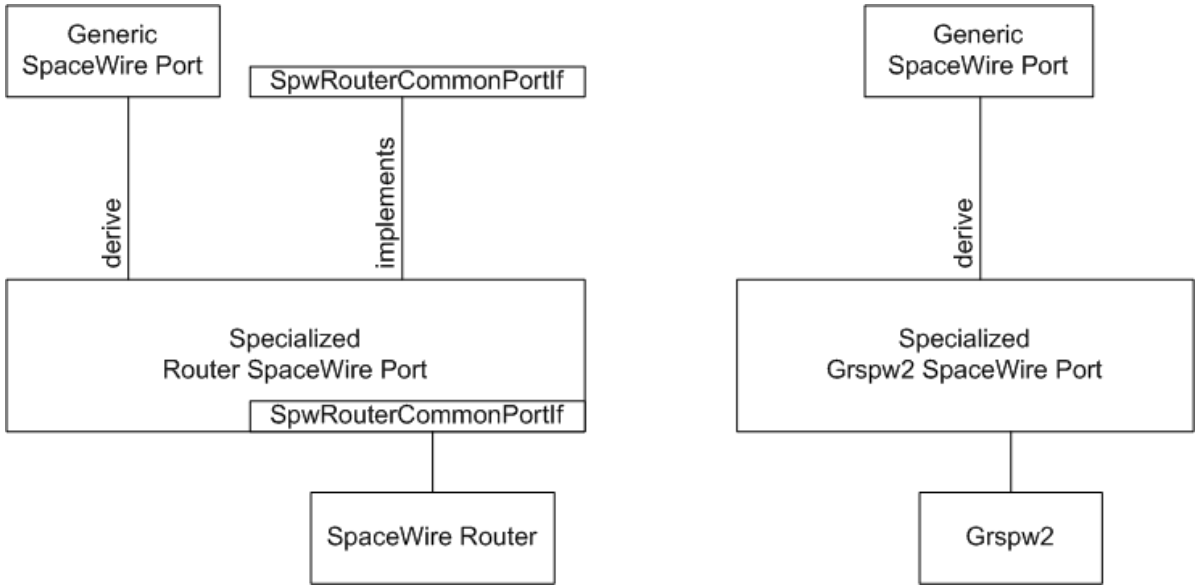
- GRSPW2 core share lot of functionalities with the router:
  - The GRSPW2 SpaceWire link interface has the same functionalities of a Router SpaceWire port
  - The GRSPW2 DMA interface has the same functionalities of a Router AHB port

- Lot of the code that was written for the GRSPW2 model could have been shared with the router, but the GRSPW2 was not written with the reuse in mind

- When designing the SpaceWire ports, attention has been paid to produce code that can be reused

SpaceWire - Architecture

- Reusable code has been achieved through:
  - Use of template method pattern
  - Use of delegates
- The generic, reusable code for a port has been put into a generic class that uses the template method pattern. The port cannot be used as is but must be derived in order to “specialize” it for a specific purpose (i.e a SpaceWire router port) by implementing the required method (i.e accessor to registers)
- The generic port uses delegates to notify relevant events (i.e. link state change) so that the specialized port can perform custom handling

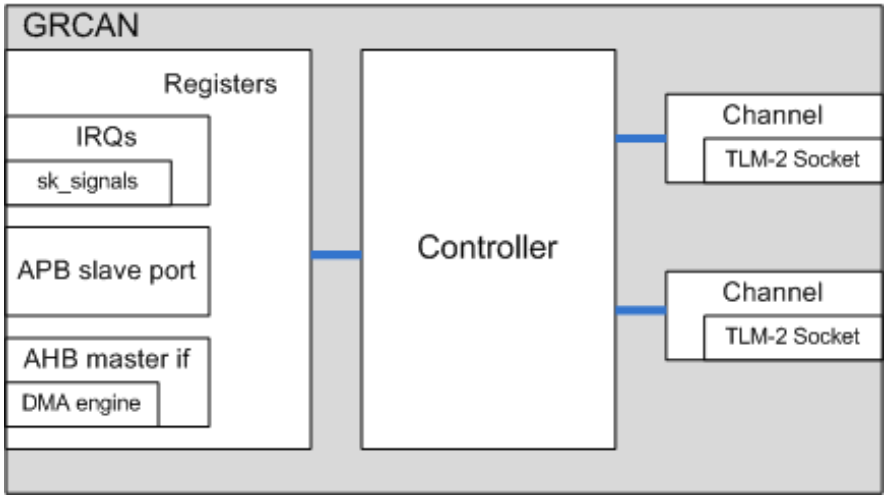


GRCAN

Features

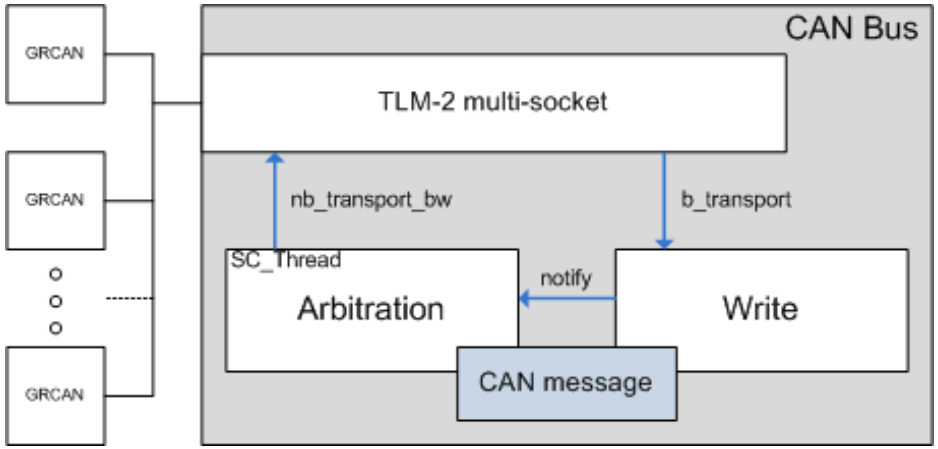
- The GRCAN models the Gaisler CAN IP Core
- It provides an interface between an AHB Bus and two redundant CAN bus
- The controller is configurable by registers, accessible through an APB slave interface
- The controller model implements the following features:
  - Basic and extended CAN messages support
  - Reading/Writing automatically CAN messages on a circular buffer using the AHB bus
  - Interrupts
  - Reset
- The bus model implements the following features:
  - Arbitration
  - Bus speed
  - Error injection
  - Message priority
  - Message acknowledgement

GRCAN - Architecture



- The device is structured in units, each handling a specific functionality of the model
- The registers provide an abstraction of the various communication ports, and automatically trigger the controller with the appropriate data defined in the registers and on the AHB bus
- The controller implements the logic regarding the CAN protocol : messages send and receive, interrupts generation, and errors checking
- The channels handle reading and writing at the CAN bus level by providing a TLM-2 socket and a backward call

GRCAN - Architecture



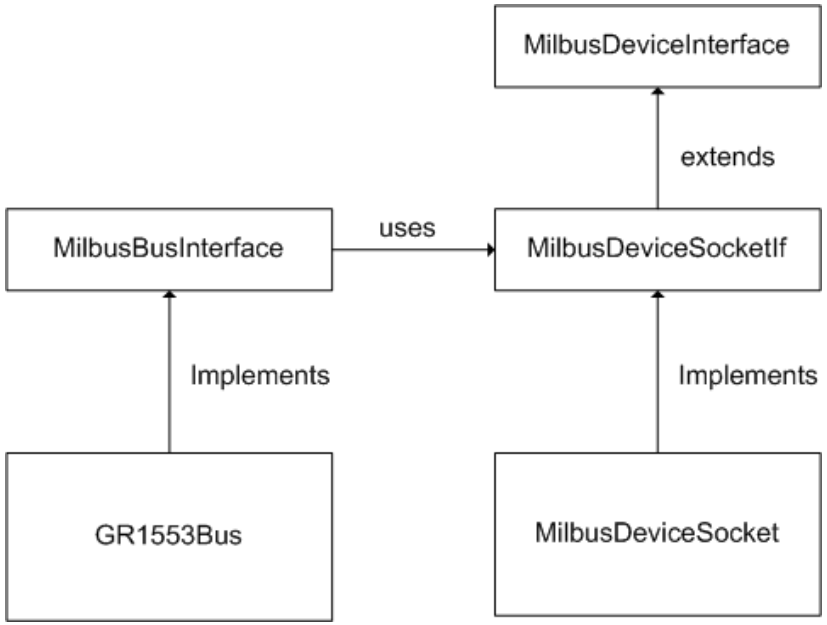
- Genericity of the CAN Bus is handled by being able to accept any device that can connect to a TLM-2 Socket
- When a device write on the bus through the b_transport call, the bus update the CAN message to send if the new message has higher priority, and notify the arbitration thread of a new available data
- The arbitration thread simulates the bus speed by waiting the appropriate amount of time, inserts artificial errors in the frame if configured to do so, and sends back the CAN message to all connected devices and to the original sender last for acknowledgment

GR1553B

Features

- The GR1553B models the Gaisler 1553B IP Core
- The device is configurable by registers, accessible through an APB slave interface
- The device model implements the following modes:
  - Bus Controller, process a transfer list and act as master on the Milbus
  - Remote Terminal, slave device which answer to the Controller using a subaddress table
  - Bus Monitor, passive device which can only receive and log messages passing through the bus
- The bus model implements the following features:
  - Can connect up to 32 devices with unique id for fast transfer
  - Bus failure

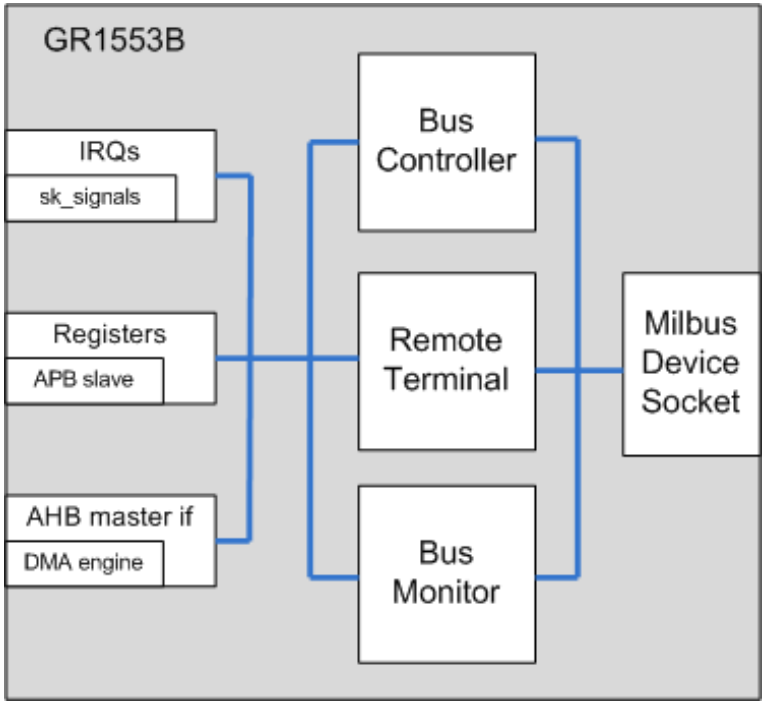
GR1553B - Architecture



- The milbus model has been made extendable through:
  - Use of interfaces
  - Use of delegates
- To be able to connect to the bus, a milbus device has to:
  - Implement the MilbusDeviceInterface
  - Define a socket that implements the MilbusDeviceSocketIf
- The bus needs to implement the MilbusBusInterface to be able to connect to devices
- This architecture allows to be able to reuse the devices while implementing a new bus, or to reuse the bus while implementing new devices

GR1553B - Architecture

- The GR1553B model implements the three types of supported device in parallel, and activates them according to the registers configuration
- Each subdevice is implemented as a separate class, and given access to the registers, the AHB bus, and the signals
- Processing is done from the Bus Controller, who triggers activity on the bus from a transfer list. It spawns its own dedicated thread to start the scheduler and will receive the data from the bus through return call



VALIDATION

Validation

- Two objectives:
  - Behavioural
  - Timing

- Previously co-validation method was used to validate models:
  - Model unit tests are executed both on the SoCRocket model and the VHDL model
  - Can be very accurate

- Co-validation method could not be used for the newest models developed in a previous activity due to availability of VHDL models
  - Gaisler VHDL models for basic components are free
  - Other models are to costly for the project budget

- Change validation approach
  - Use a developer board as golden reference
  - Execute the same set of tests on the chosen board and a virtual platform created with SoCRocket as close as possible to the board
    - GR-CPCI-LEON4-N2X board containing a prototype of the NGMP processor architecture
  - The programs generate a log in memory
  - The log is then compared

Validation

- Randomized Testing
  - Test program are randomly auto generated
  - Help to cover corner cases
  - Helps to cover unforeseen test cases (e.g. when the user does not follow the manual for setting up certain components)

- Implementation
  - Logging functions that write the state of the model (i.e. registers values) in memory are implemented
  - A set of functions that issue simple basic commands (e.g. enable SPW port) is implemented for each core
  - A sequence of functions, taken from the set, is generated randomly (probabilities can be adjusted in the test generator configuration file)
  - The test program
    - Register the logging functions to react to interrupts
    - Execute the random function sequence

Validation Summary

- During the attempts to get the tests executed several bugs were found and corrected:
  - AHBCtrl incorrectly used IO BAR PNP to decode addresses (same algorithm of MEMORY was used)
  - MSDRAM / L2C PNP table had to be corrected after AHBCtrl decoding correction
  - Incorrect tag format of the L2C diagnostic interface
  - L2C lines incorrectly marked dirty on diagnostic read access
  - SpaceWire Port in SpaceWire Router not adding EEP on disconnection

- Log analysis
  - The core models behaves correctly
    - L2C content dump matches
    - MSDRAM tests execute correctly
    - AHB/APB controllers works correctly
  - SpaceWire Router behaves correctly with the exception of some corner cases

Validation Summary

- Log analysis
  - Timing issues
    - The execution time measured on the virtual platform is often shorter than the one on the board
    - No time for investigation but the following hypothesis:
      - A component along the memory chain returns an incorrect delay, too small to be noticed in unit-tests, but that stack up during execution
      - An incorrect configuration of the component

ANALYSIS

Analysis

- The upstream version of SoCRocket has been developed in the last two years and diverged from the version owned by ESA
- In a previous project that involved extending the model library, two possible enhancement were highlighted in order to improve the simulation accuracy:
  - Bus size configurability
  - SPLIT

- Objective of the analysis:
  - Identify the changes in the upstream version of SoCRocket
  - Evaluate if it is worth to merge the two versions and the effort required
  - Evaluate the effort required to implement the SPLIT and bus size functionalities

Analysis

- In order to identify the changes in the upstream version of SoCRocket with respect to the ESA version, the two code bases have been compared, resulting in:
  - Lot of changes due to commenting and cleanup => diff was not helpful, careful review of all the core classes was necessary
- Some of the differences were:
  - New directory structure that separate the framework code from the model library
  - Code clean up
    - Uniform coding style
    - Added comments & DOXYGEN documentation
  - Introduced a package system into the build system
  - Adjusted the class hierarchy and introduced a new base-class for models
- The advantages in merging the two version are summarized as follow:
  - Code quality: code cleanup has been performed over the entire codebase, removing unnecessary code and reorganizing it in order to be easier to manage
  - No manual patching required during the installation
  - Modularity supported by the build system (packages managing) and new directory structure
  - Standardization of code in models (common initialization methods, common parameters containers)
  - New version of SystemC supported

Analysis

- The effort for merging the two versions results in the following activities:
  - Reapply template parameters on new base classes (coming from a previous activity)
  - Modify ESA models and relative test benches to use the new base classes and reorganize their code to make effective usage of them (i.e. move register initialization code in proper overridden method)
  - Create a package for the models
  - Estimated effort for merging: up to ~80h (half month)

- Conclusion
  - The changes do not bring new features but improved code quality and standardize the way models are implemented
  - If it is foreseen further development on the platform a merge is suggested as the changes will improve the maintainability
  - Future improvements in upstream version will be imported easier

Analysis – bus-size configurability

- A code base analysis was necessary to determine the effort required to implement the bus configurability functionality

- Bus Size
  - The TU-Braunschweig version of SoCRocket is designed to be used with 32 bit buses. This assumption has been found hardcoded through the whole code base

- Configurability implementation
  - The models classes requires templatization. SoCRocket uses template on ports to have compile time checks on connected device ports. A template parameter for port is the BUS size but it is hard-coded to 32
  - The code analysis highlighted that several changes in the whole code base was required, usually where a delay had to be returned
  - Test implementation and adaptation

```
43      // The delay returned by the function model relates to the time for delivering
44      // the data + wait states.
45      address_cycle_base = (((trans.get_data_length()-1) >> 2) +1);
46
47      // In case unvalid access (TLM_ERROR_RESPONSE)
48      // some models might return an unvalid delay.
49      // Set delay to sane default for safe reponse processing
50      if (delay==SC_ZERO_TIME) {
51
52        delay = address_cycle_base * get_clock() + sc_core::sc_time(1, SC_PS);
53
54      }
55
56      //v::debug << this->name() << "Total delay: " << delay << v::endl;
57
58      // Calculating delay for sending END_REQ
59      request_delay = delay - sc_core::sc_time(1, SC_PS);
```

Analysis – SPLIT

- Design
  - Should be implemented in AT mode only
  - Introduction of three new phases:
    - DATA_SPLIT Slave sends this phase to signal the AHBCtrl a SPLIT
    - DATA_CONTINUE Slave sets this phase to signal that the AHBCtrl can grant the bus back
    - BUS_GRANT The AHBCtrl use this phase to grant the bus to the SPLITed transaction

- Implementation
  - AhbCtrl
    - React to the DATA_SPLIT and DATA_CONTINUE phases:
      - Re-arbitrate on DATA_SPLIT, starting another transaction from the queue as normal.
      - Introduce a queue for the devices ready to continue a transaction
      - When arbitrating first pool the queue of devices that wish to continue a transaction
  - Masters
    - No modification required
    - Unless the master is connected to a slave directly (without AhbCtrl). In this case it has be modified to respond to the DATA_CONTINUE phase with BUS_GRANT
  - Slaves
    - Models that whish to use split functionality has to be modified to issue the DATA_SPLIT and DATA_CONTINUE phase, and to react to the BUS_GRANT phase

Conclusion and future work

In **conclusion** Terma achieved:

- Provided a solution to implement SPLIT and BUS_SIZE configurability
- Common spacecraft models (CANbus, Milbus and SpaceWire) implemented
- SoCRocket behavioural characteristics validated

Terma has identified the **possible future work**:

- Merge with the upstream version
- Correct the timing issues
- Implement
  - Bus size configurability
  - Split functionality
- Extend SoCRocket with more models
- Add support for other emulator cores
  - Adapt models from the model library to run also under other emulators (detach from SystemC)

Meet us at…

www.terma.com

www.terma.dk/press/newsletter

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

www.twitter.com/terma_global

www.youtube.com/user/TermaTV

TERMA

ALLIES IN INNOVATION