Environment for efficient and reusable SystemC module level verification

Flavia Gonția
Siemens Corporate Technology Romania
CT RTC ELE ELD-RO

Modeling with SystemC®

- IEEE standardized language (a C++ class library)
- Mainly used in large electronic system-level designs
- Used to model abstract hardware models

Important benefits

- System design **optimization**
- Architecture **exploration**
- **Early software development**

SystemC module verification

- Using **UVM**, based on SystemVerilog or *e*
- Using **SCV** (SystemC Verification Standard), based on C++
- Using an **ad-hoc verification environment**, using SystemC library

Motivation

- A minimal verification is needed for any SystemC module
  - Standard methodologies tend to be complex and resource consuming
  - The fastest solution is the **SystemC environment**
- SystemC testbenches usually address particular modules
  - No verification methodology ⇒ **lack of reusability & lack of portability**
- Most SystemC designs are configured/controlled by a CPU (embedded software)
  - Configuration & control is spread throughout testbench logic, code etc.
  - **Extraction of configuration** into an embedded software **is often impossible**

Proposed solution improves SoC development with SystemC

1. SystemC module implementation
2. Basic module verification
3. Module integration in the platform
4. Verification of correct integration of the module in the platform
5. Platform delivery for software development
6. Physical chip validation

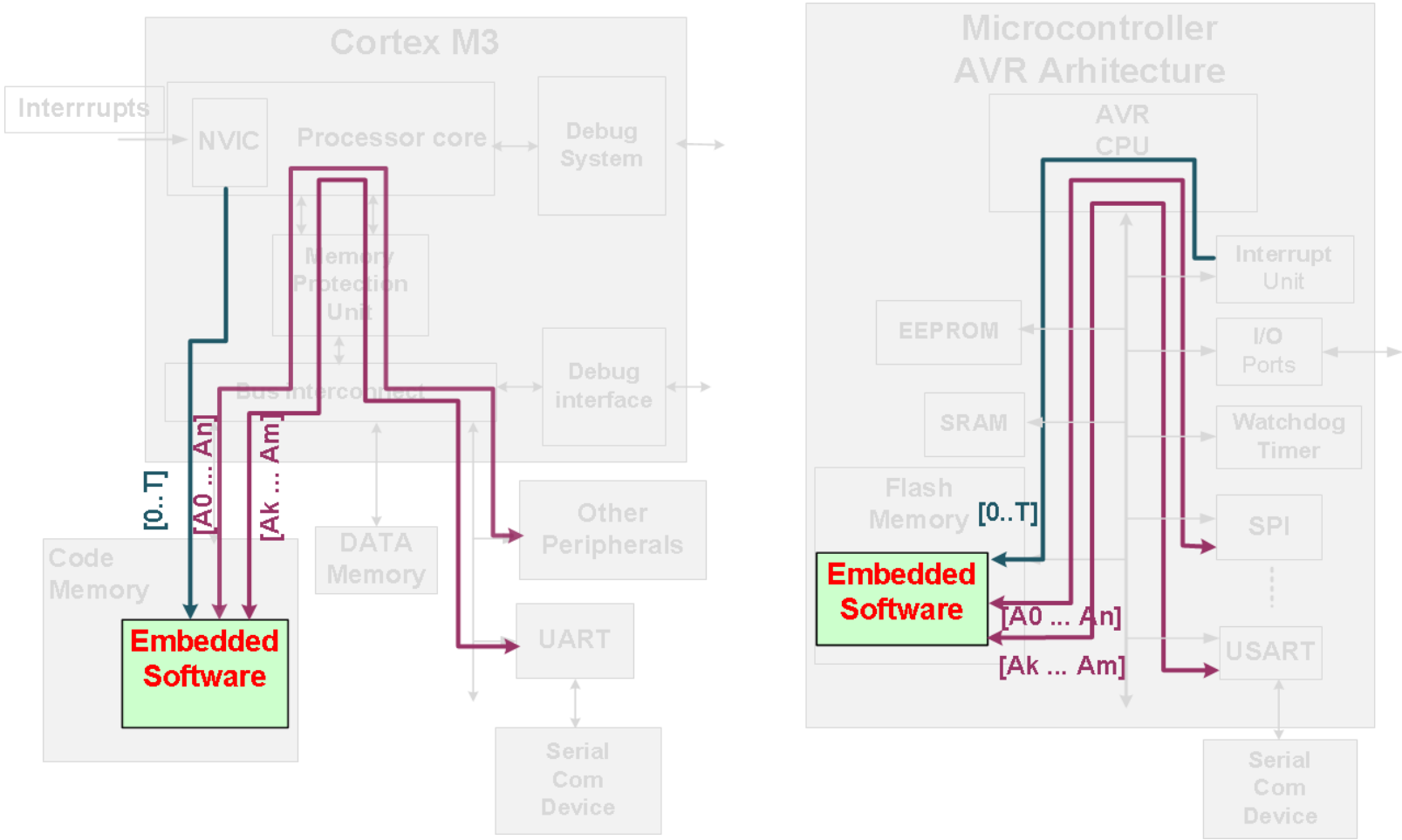
- **A structured verification environment:**
  - **High degree of reusability**
  - **High degree of portability**
- **Write embedded tests**
  - ⇨ ***software-driven verification***
- **Write C++/SystemC tests**
- **Run regressions**

Embedded tests used to validate a correct integration of a module inside of a platform

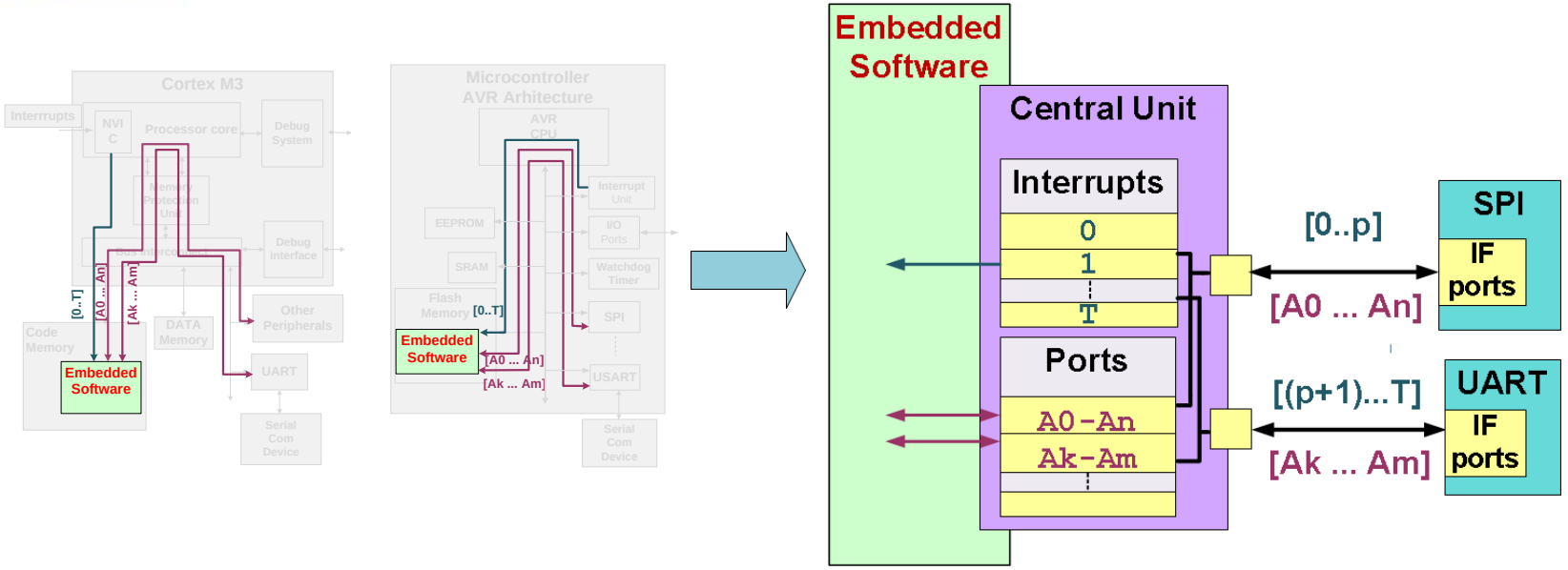
Embedded tests become blueprints for software development

Embedded tests validate correct chip functionality

Communication from the embedded software perspective



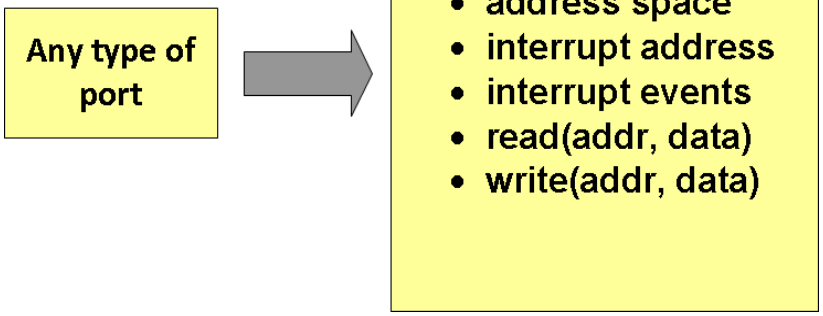
A new idea enters the game…



- Embedded software interacts with the “outside world” by means of:
  - **memory map**
  - **interrupt vector**
- As a result, any port can be identified by:
  - **address space**
  - **interrupt address**

Decoupling tests is also possible

- Tests can be **decoupled** from DUT port interface by converting the communication with each port, of any type, to a ***single, base type***

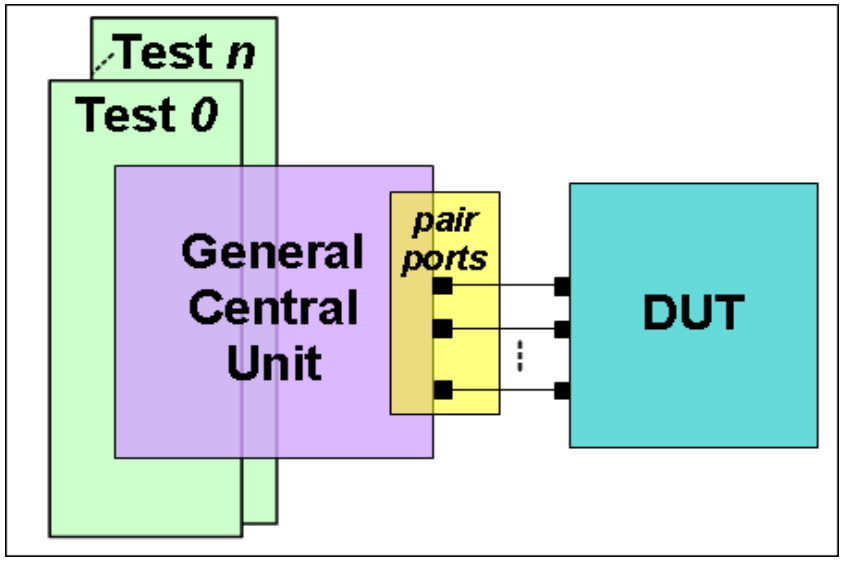


- SystemC provides **a broad range of port and socket** types: **`tlm_target_socket<>`**, **`tlm_initiator_socket`**, **`sc_in<>`**, **`sc_fifo_out<>`**, **`user defined port`**, etc.

Decoupling tests from port types represents a major advantage

SystemC General Verification Environment (SGEV)

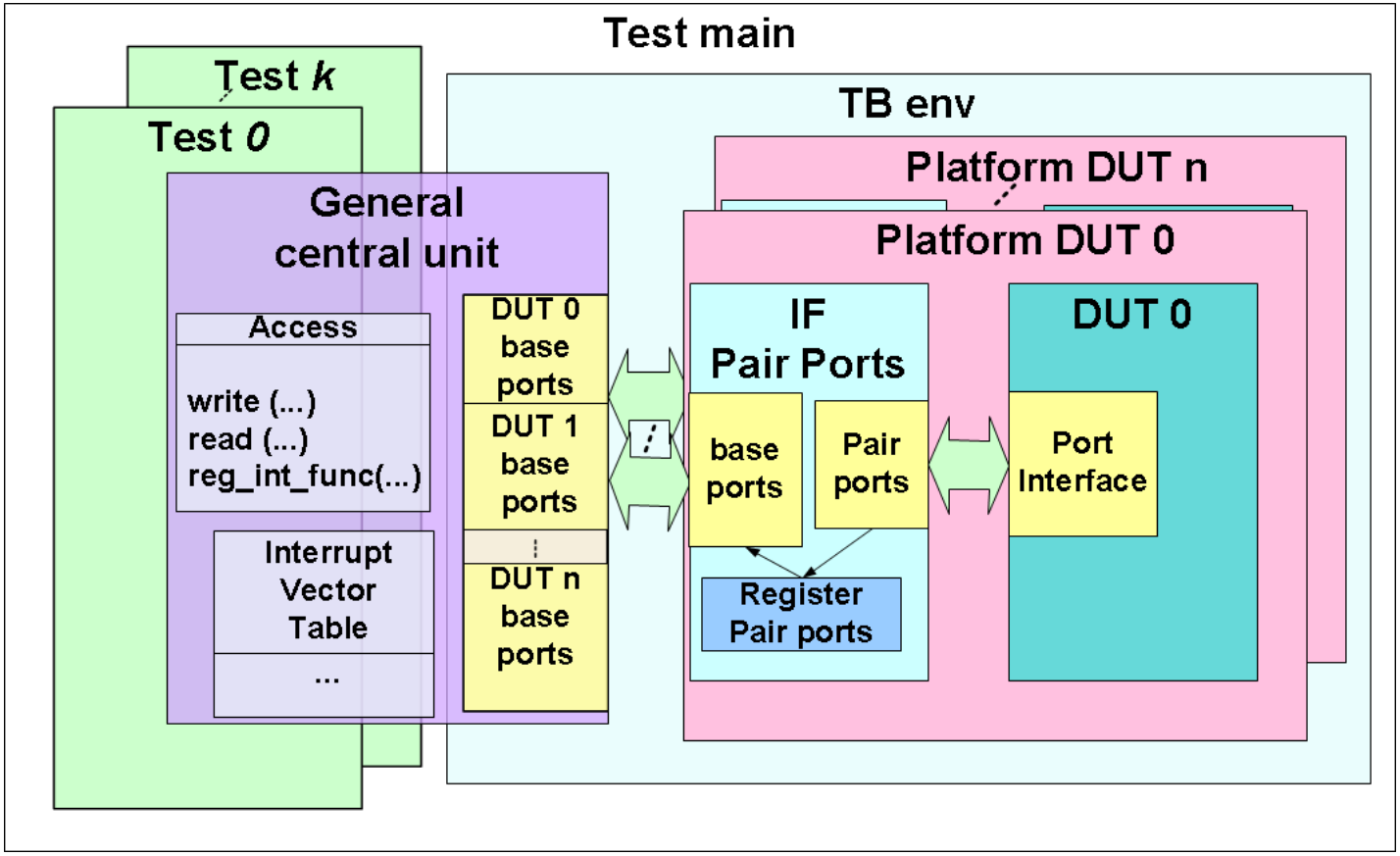
- **Device Under Test** (DUT)
- **Pair ports:**
  - adapt the communication from a DUT *specific port* to a *base port* communication
  - identified by an *address space* and a *interrupt address*



- **General Central Unit (GCU):**
  - groups the *pair ports* that are bound to DUT-ports
  - provides means for communication with DUTs (through out pair ports)
- **Tests:**
  - access the DUT's ports through the GCU
  - implement complex scenarios or simple data generators, monitors, etc.

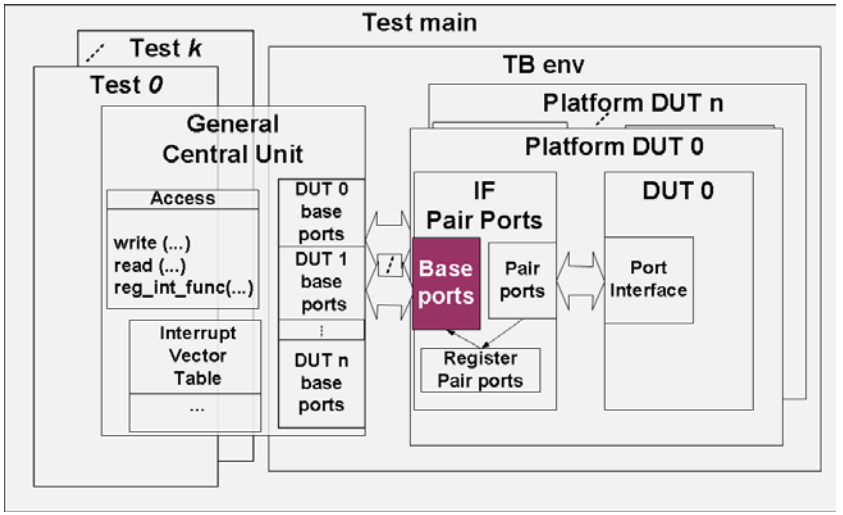
SGEV library

- **A SystemC/C++ library** that implements the presented concept



Base Port class

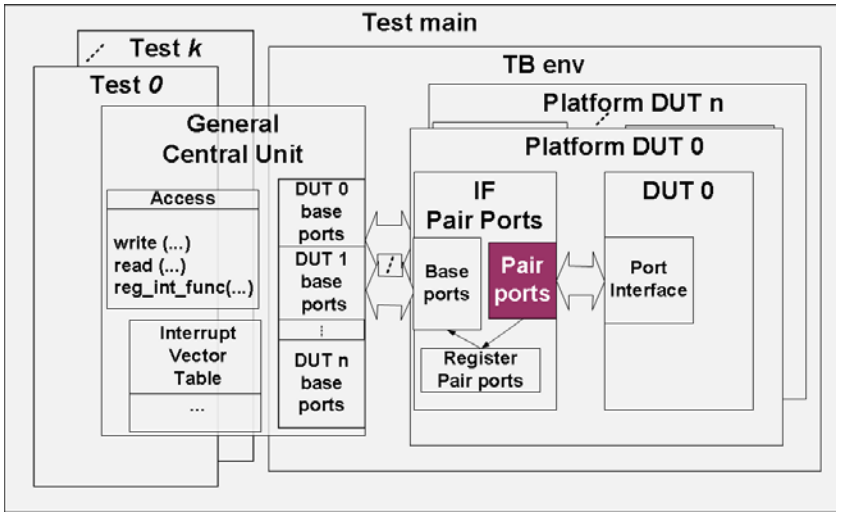
- The *base port class* defines the base communication functions and members:
  - **address space**
  - **interrupt address**
  - **array of events**
  - **read(addr, *data*)**
  - **write(addr, *data*)**



- Propagate data of a single type: **BASE_DATA_TYPE**

Pair-Ports classes

- Extend the *base port* class and implement the **read()** and **write()** functions which adapt port specific communication to base communication



- SGEV library **provides pair port classes** for common types of **SystemC ports** (LT modules):

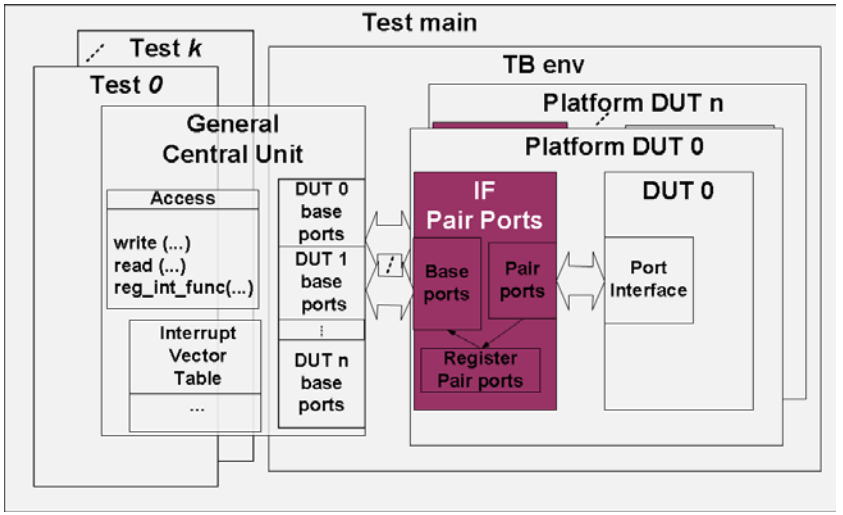
SystemC port/TLM type	SGEV pair port type
`sc_in<bool>`	`sgev_sc_out<bool>`
`tlm_target<BUSWIDTH>`	`sgev_initiator_socket<BUSWIDTH>`

…

- **User defined pair-ports** can be created

Pair-Ports interface class

- For each desired port of the DUT, the corresponding *pair-port* must be instantiated in a *pair-port interface class*



- Here, the *pair-ports* must be registered with the following:
  - **address space** that identifies the port
  - **interrupt address** where a callback functions can be registered
  - **type of interface** associated: accessible by embedded software (IF_CU) or not (IF_NOT_CU)

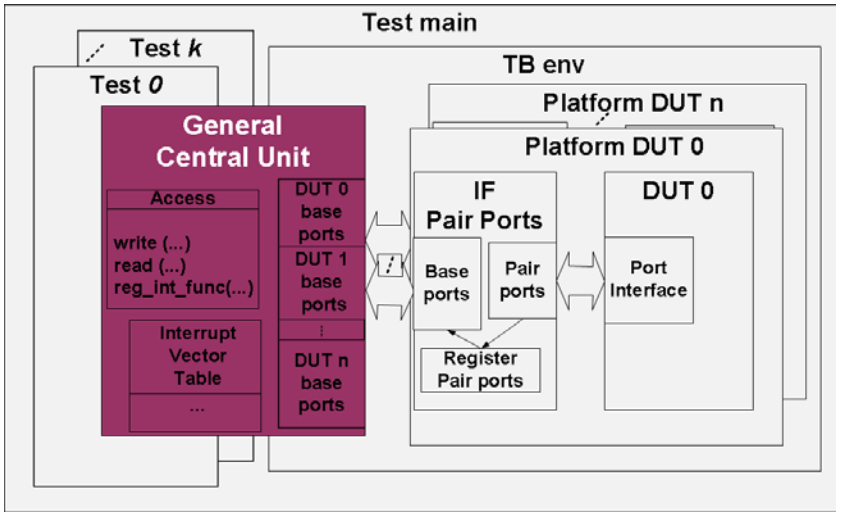
Pair-Ports interface example

```
class uart: public sc_module{
public:
   tlm_target_socket<32>        p_reg_targ;
   sc_fifo_out<unsigned char>   p_serial_out;
          …
};
```

```
template<class B_DT> class if_uart_pair_ports: public SGEV_if_base<B_DT>{
 public:
   SGEV_init_socket<32, B_DT>            reg_init_sockt;
   SGEV_sc_fifo_in<unsigned char, B_DT> in_serial;

   if_uart_pair_ports(sc_module_name &n): SGEV_if_base<B_DT>(n){
                        //    Port    |Port address|   Address    |  Interface
                        //   Object   |Start | End |  Interrupt   |  Type
     SGEV_REGISTER_PORT(reg_init_sockt,  0x0,  0XFD, ADDR_NOT_USED, IF_CPU);
     SGEV_REGISTER_PORT(     in_serial, 0x0FF, 0x0FF,          0x1, IF_NOT_CPU);
   }
};
```

General Central Unit (GCU)

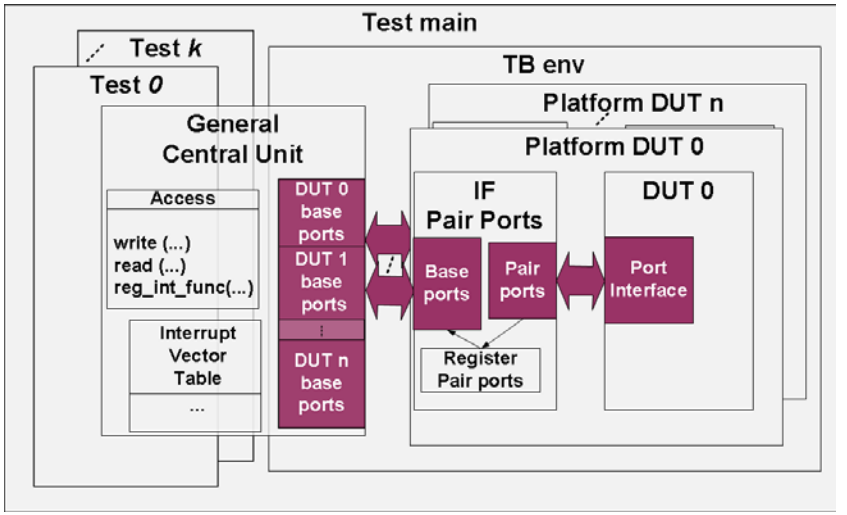
- Provide the **functions for** *pair port interface* or single **pair port registration**
- Implements **read()** and **write()** functions: data is routed to/from the corresponding port, based on the address



- Propagate data of a single type: **BASE_DATA_TYPE**
- Implements an **interrupt vector list** (increases dynamically)
- **Synchronization** with SystemC **scheduler**

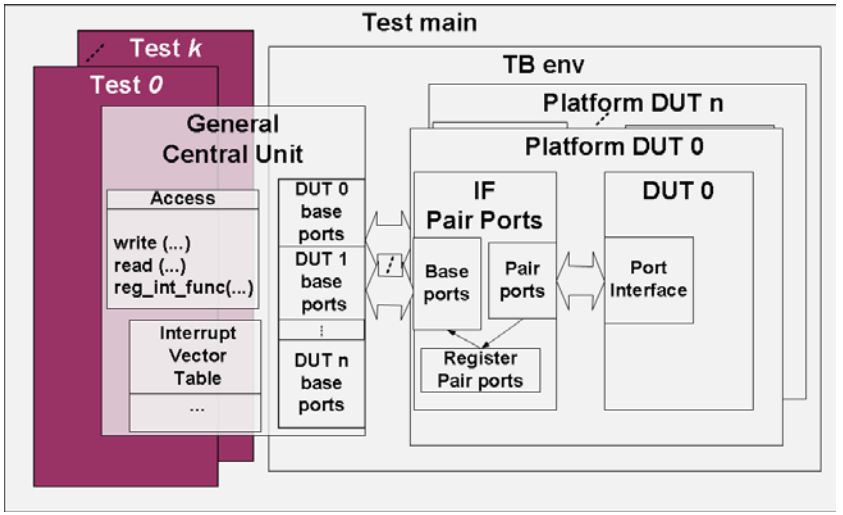
GCU example

```
template<class B_DT>
Class tb_uart_env{
 public:

 SGEV_general_cu<B_DT>     i_gcu;
 if_uart_pair_ports<B_DT> i_pports;
 uart                          i_uart;


 tb_uart_env(const char* name): i_if_pair_ports("if_p_ports"){
    i_uart.p_reg_targ.bind(i_pports.reg_init_sockt);
    i_uart.p_serial_out(i_pports.in_serial);
 }
 void end_of_elaboration(){
    i_uc.register(i_pports);
 }
};
```



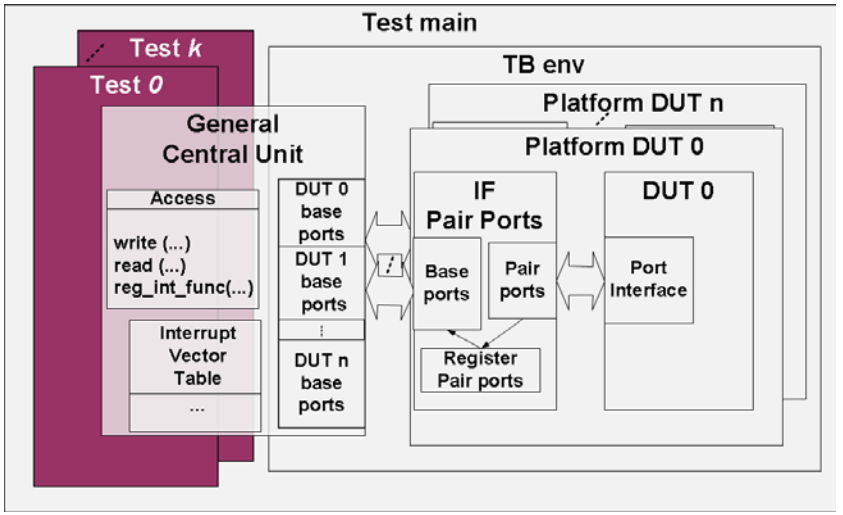
Tests

- Receive the **reference of a GCU**, transmitted at instantiation
- Can **access all *pair ports* throughout GCU** read/write functions
- Two types of tests are defined:
  - **SGEV_TEST_NOT_CPU**: can access any port
  - **SGEV_TEST_CPU (embedded tests)**: can access only *pair ports* previously registered with IF_CPU
- The tests can **contain** both **SystemC and C++ code**
- **Embedded tests** can also be written



Embedded tests

- **Run on** the **host processor**
- **Register accesses** can be performed **using aliases** (UART_RX, I2C_STATUS, etc.) instead of read/write functions
- **Interrupt callback functions** can be registered



- Already written **embedded tests can be adapted to run in a SGEV** environment

***#define* constructs** used for adapting the embedded tests from SGEV to CPU

SGEV_REG_TYPE	- register type	(e.g.: **`unsigned int *`** on CPU)
SGEV_REG_CAST_TYPE	- cast to register type	(e.g.: **`(unsigned int *)`** on CPU)
SGEV_NOP	- no operation	(will be empty on CPU)
SGEV_TYPE_CLASS	- class type	(will be empty on CPU)

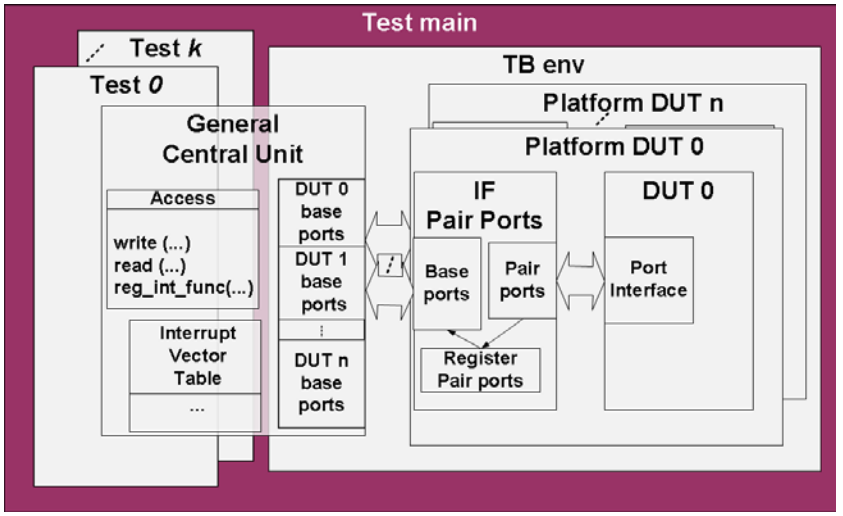
Embedded tests example

```
                    // File of SystemC embedded test
SGEV_TEST_CPU(main_uart, test_uart_sc)
  #include "../UART_TESTS_EMB/main.h"
};
```

```
                    // File ../UART_TESTS_EMB/main.h
#define Uart_BASE 0x100

void wr_to_uart(){
  SGEV_REG_TYPE UARTDR = SGEV_REG_TYPE_CAST(Uart_BASE + 0x000);
  UARTDR = value; // write to UART
}

void main_uart(){
  register_int_func(UARTTX_INT_ADDR, &SGEV_TYPE_CLASS wr_to_uart);
  while(1){
    SGEV_NOP;
  }
}
```

Main Test

- Contains **tests** and **TBs** instances
- Each test will **receive the reference of the GCU** instance from the desired TB

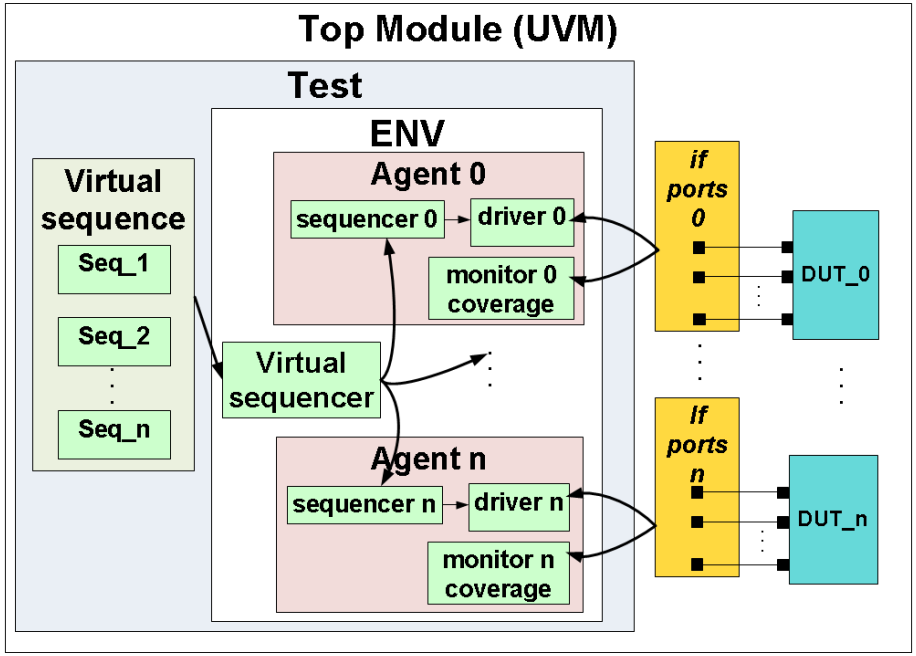


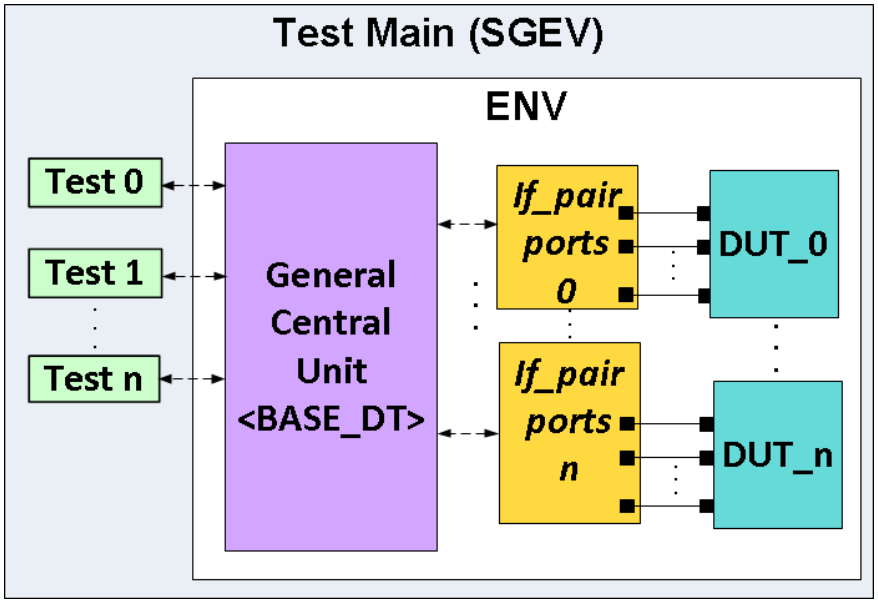
- **Any number of TB and TEST** can be instanced and run in parallel
- **A test** can be launched to **run in parallel** or **sequentially**
- The **main test** can be used as **environment for regressions**

Main Test example

```
template<class B_DT>
class test_uart_sc_main: public SGEV_test_main_base{
 public:
   tb_uart_env <B_DT>       i_uart_tb_env;
   test_uart_sc_emb<B_DT> i_uart_test_emb;
   test_out_uart<B_DT>    i_uart_out_test;

  test_uart_sc_main(sc_module_name &n): SGEV_test_main_base(n)
    ,i_uart_tb_env("i_uart_tb_env")
    ,i_uart_test_emb("i_uart_test",     &i_uart_tb_env.i_gcu)
    ,i_uart_out_test("i_uart_out_test", &i_uart_tb_env.i_gcu)
  {
  }
  void main(){
     SGEV_RUN_TEST_PARALLEL(&i_uart_out_test);
     SGEV_RUN_TEST_PARALLEL(&i_uart_test_emb);
  }
};
```

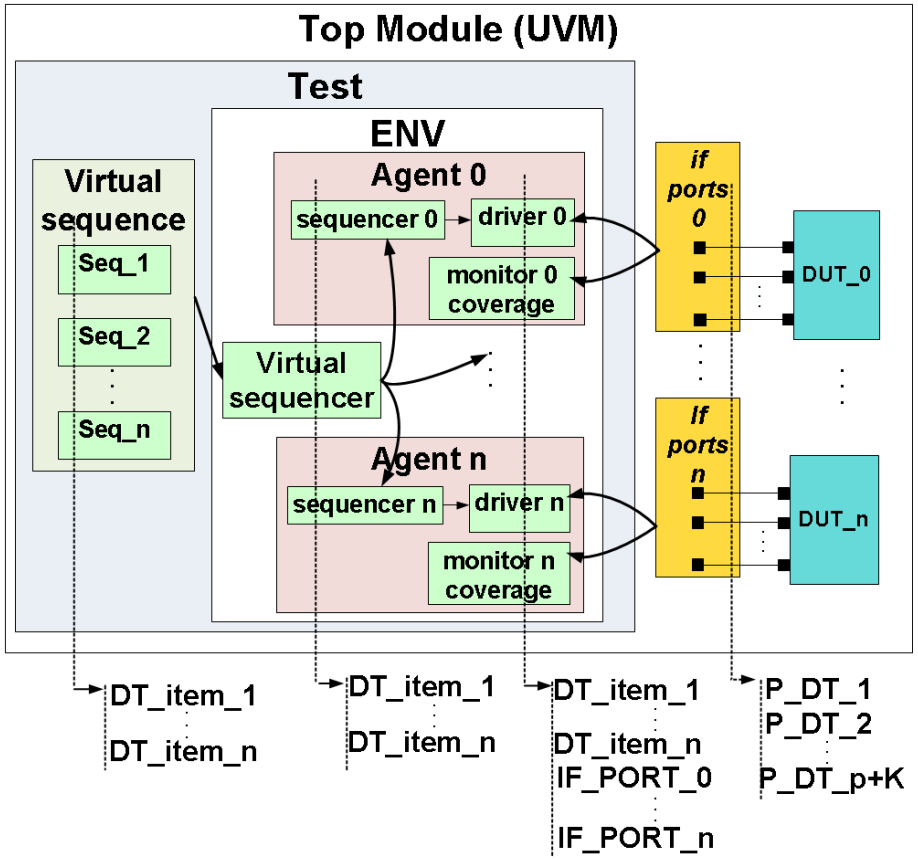
UVM vs. SGEV: *A User's Perspective*

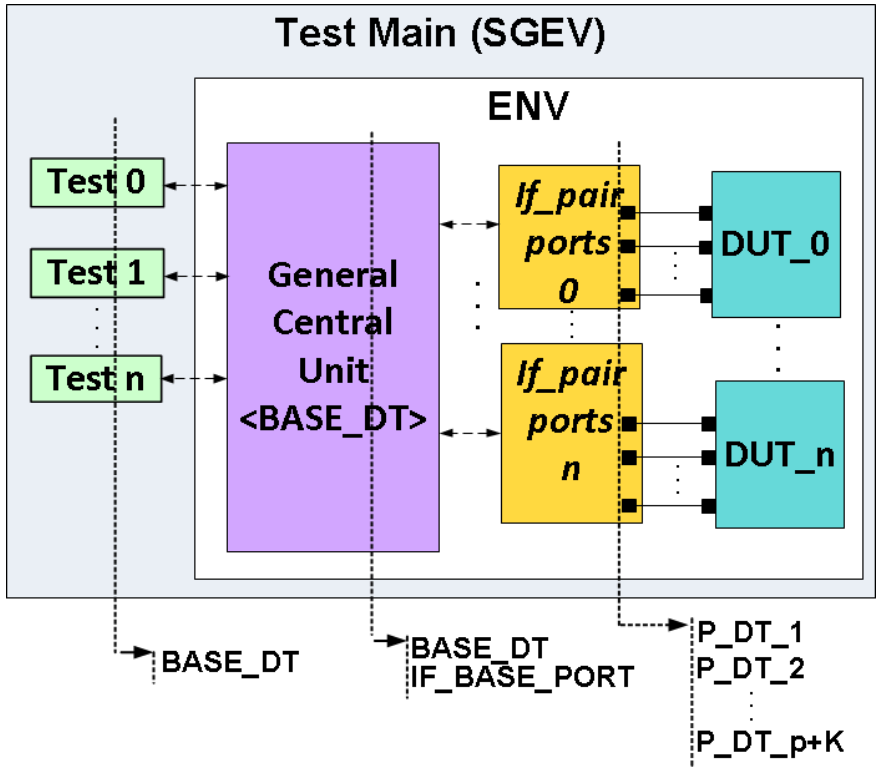




Using UVM System Verilog library			Using SGEV library
DUT virtual interface			Pair Port Interface
Agent	Driver		-
	Monitor	Checks	
		Coverage	
	Sequencer		
Environment (ENV)			TB environment (ENV)
Sequences			Tests
Test			
Top			Main Test

UVM vs. SGEV: *A User's Perspective*





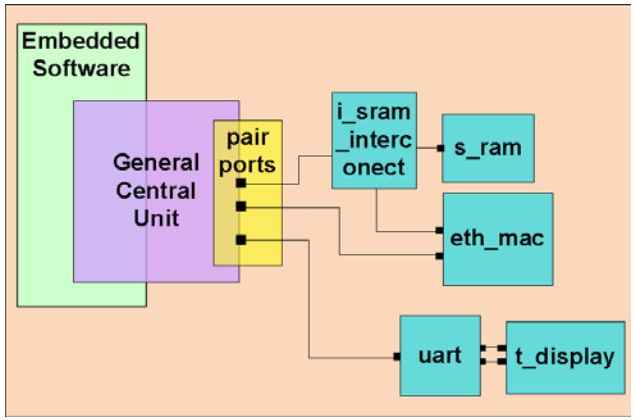
- **Data Type (DT) dependency**

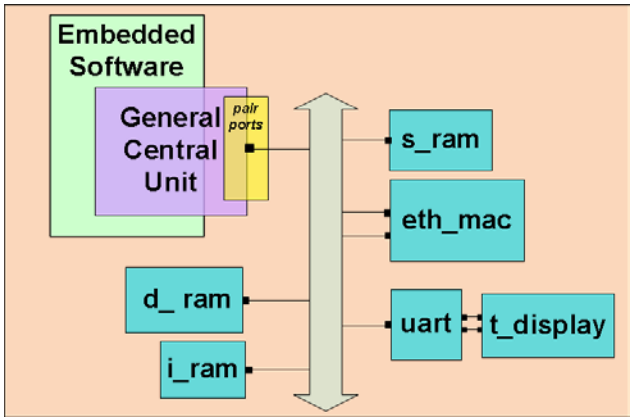
SGEV
Performance analysis

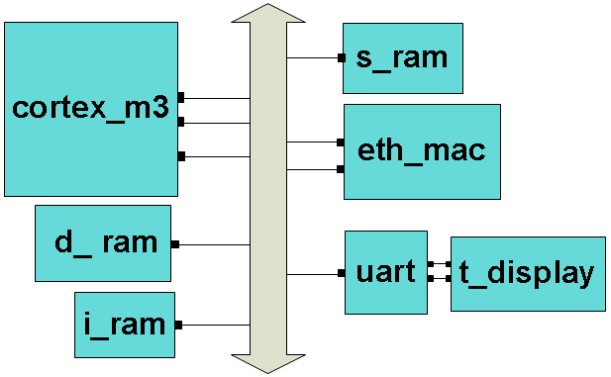
- **One single use case**: embedded software that implements the communication protocol for receiving and sending Ethernet frames to an Ethernet Controller
- **Running environments**
  - Synopsys Virtual Prototype Analyzer (VPA)
  - Eclipse
- **3 Platforms**
  - **VPA *+* CortexM3** model + Embedded software compiled image
  - **VPA + SGEV** + Embedded software C files
  - **ECLIPSE + SGEV** + Embedded software C files

SGEV

Performance analysis results





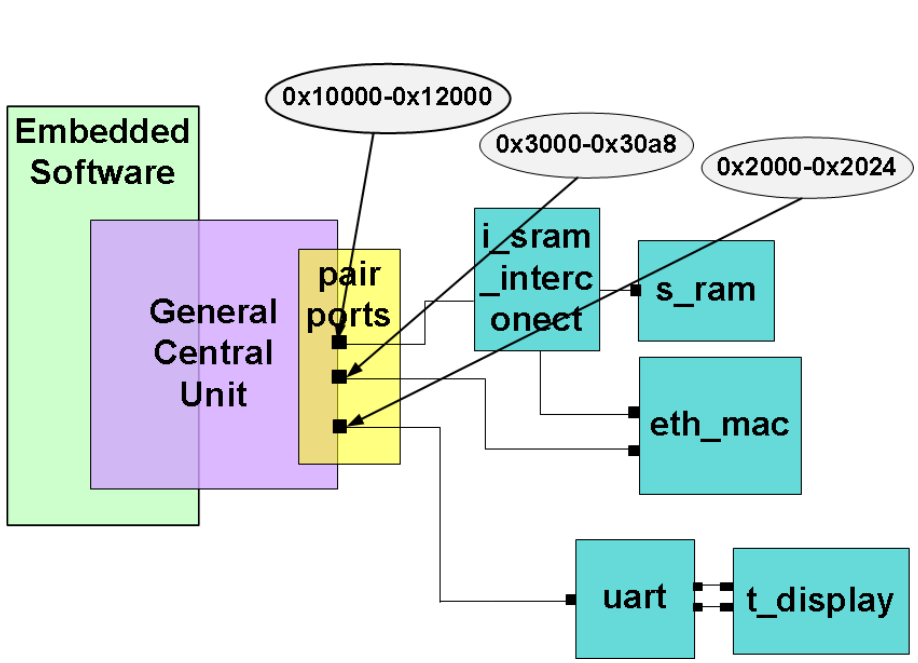


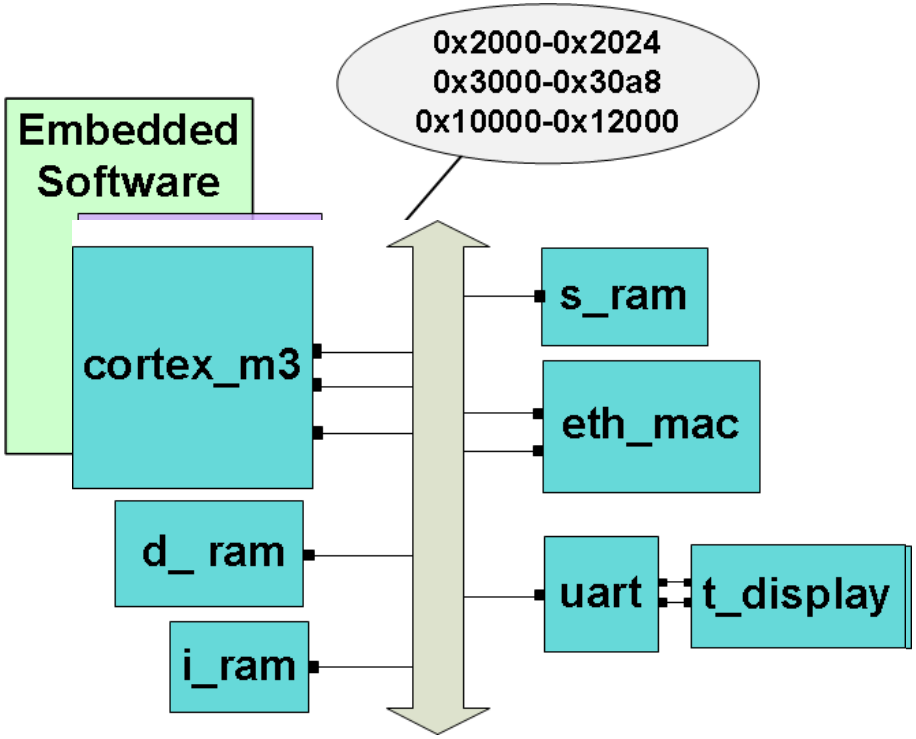
SGEV platforms >20X faster

Simulation time	Execution time		
	Eclipse + SGEV	VPA + SGEV	VPA + CortexM3
5 s	14.05 s	15.48 s	46.64 s
7 s	16.81 s	17.97 s	97.56 s
10 s	18.07 s	19.36 s	171.19 s
20 s	24.55 s	26.17 s	429.73 s
30 s	**29.05 s**	**31.2 s**	**641.8 s**

SGEV
Performance analysis results

- **Module port number independence**





Conclusions

SGEV benefits:

- Perform software-driven verification with the help of embedded software (that runs on host computer)
- SystemC/C++ tests can be used together with embedded tests
- Tests + GCU can replace: a basic SystemC CPU and a SystemC module
- Faster ramping-up of the verification environment
- Offers a high degree of tests reusability and portability
- Regression environment can be implemented with reduced effort
- It is not tool dependent

Thank you for your attention!

Any questions?