

**TẬP ĐOÀN CÔNG NGHIỆP  
VIỄN THÔNG-QUÂN ĐỘI  
VIETTEL**



**DESIGN SPECIFICATION**

**SOC PLATFORM**



**Author: Huan Nguyen-Duy  
Date: 8/16/2024**

## CATALOG

1. OVERVIEW .....	4
2. SOC PLATFORM ARCHITECTURE .....	5
3. FEATURES .....	6
3.1. Transferring data among internal components .....	7
3.2. Setting registers using dummy master .....	9
3.3. Monitoring internal ports using dummy slave .....	11
3.4. Register interface implementation .....	12
3.5. External memory .....	13
3.6. Wrapper for connection with RTL model .....	14
4. TOP MODULE .....	15

## FIGURE

Figure 1 : The basic SoC platform architecture .....	5
Figure 2 : Transferring data using dummy master .....	7
Figure 3 : Read or Write register operation .....	9
Figure 4 : Monitoring ports using dummy slave .....	11
Figure 5 : Register interface support .....	12
Figure 6 : Wrapper common architecture for connecting RTL model	14

## TABLE

Table 1 : Variable definition of class bus

Table 2 The list of functions

# 1. OVERVIEW

This document illustrates the design of custom SoC platform using systemC. This platform contains all essential models that are connected by bus with memory mapping I/O. In particular, the basic SoC platform includes a dummy master and dummy slave for controlling register read/write operation and monitoring all the status of interconnected ports. Furthermore, external memory is support for DMA operation and other purposes. Specially, the SoC platform supports a register interface that is implemented independently to use as common interface, it helps developer easily create registers of models that are mapped into memory mapping I/O that can access by dummy master model (having a crucial role as the cpu core of system).

On the other hand, to connect with models that are implemented at RTL level, we propose a way to implement wrapper model that will map all registers to TLM socket and using bus memory mapping I/O to control it.

## 2. SOC PLATFORM ARCHITECTURE

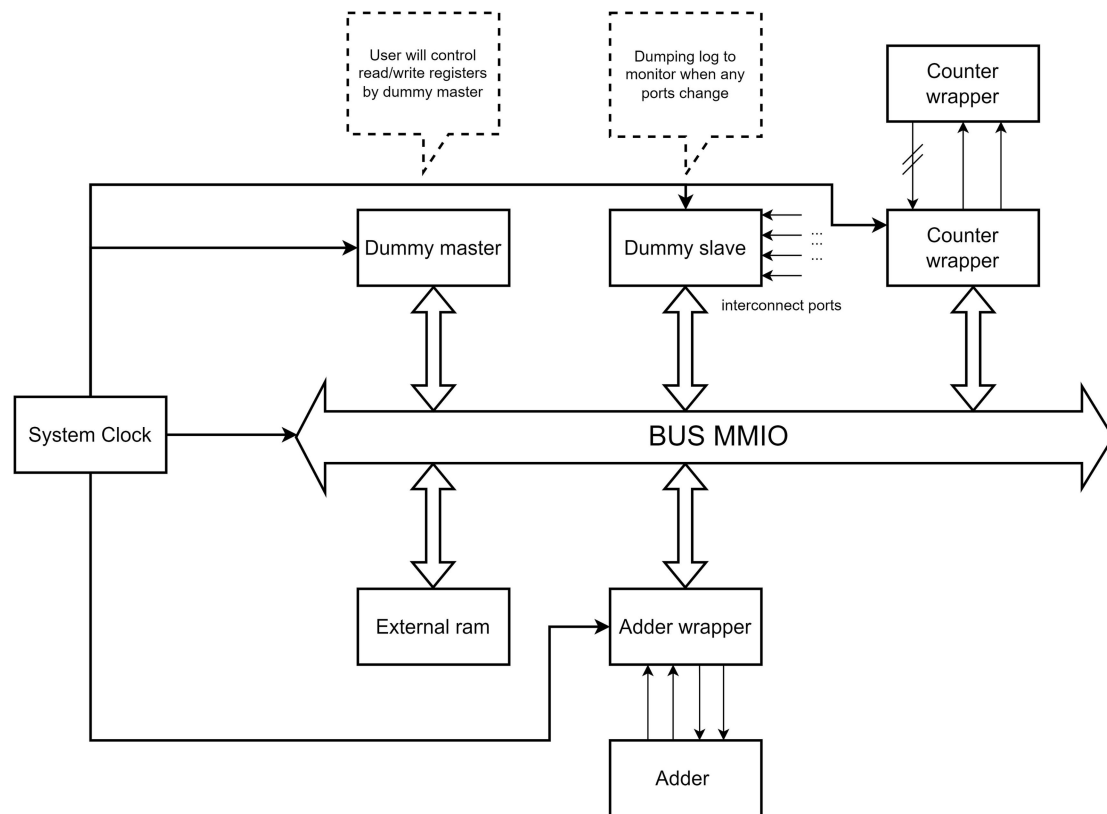


Figure 1: The basic SoC platform architecture

The fig 1 shows the basic of SoC platform architecture, user will control read/write register operation through dummy master that provided several public application programming interface. Wrapper models are implemented dependently on each RLT level model (RTL modules are write by verilog that is converted to systemC model using v2sc tool), it will implement register interface for each reg ports and binding them on bus memory mapping I/O.

### **3. FEATURES**

In this section, the document illustrates all features that are supported by SoC platform. It contains all basic operation of SoC system. In particular, peripherals are mapped into bus MMIO with specific address, dummy master can access to all peripherals by sending specific TLM payload with the corresponding address.

### 3.1. Transferring data among internal components

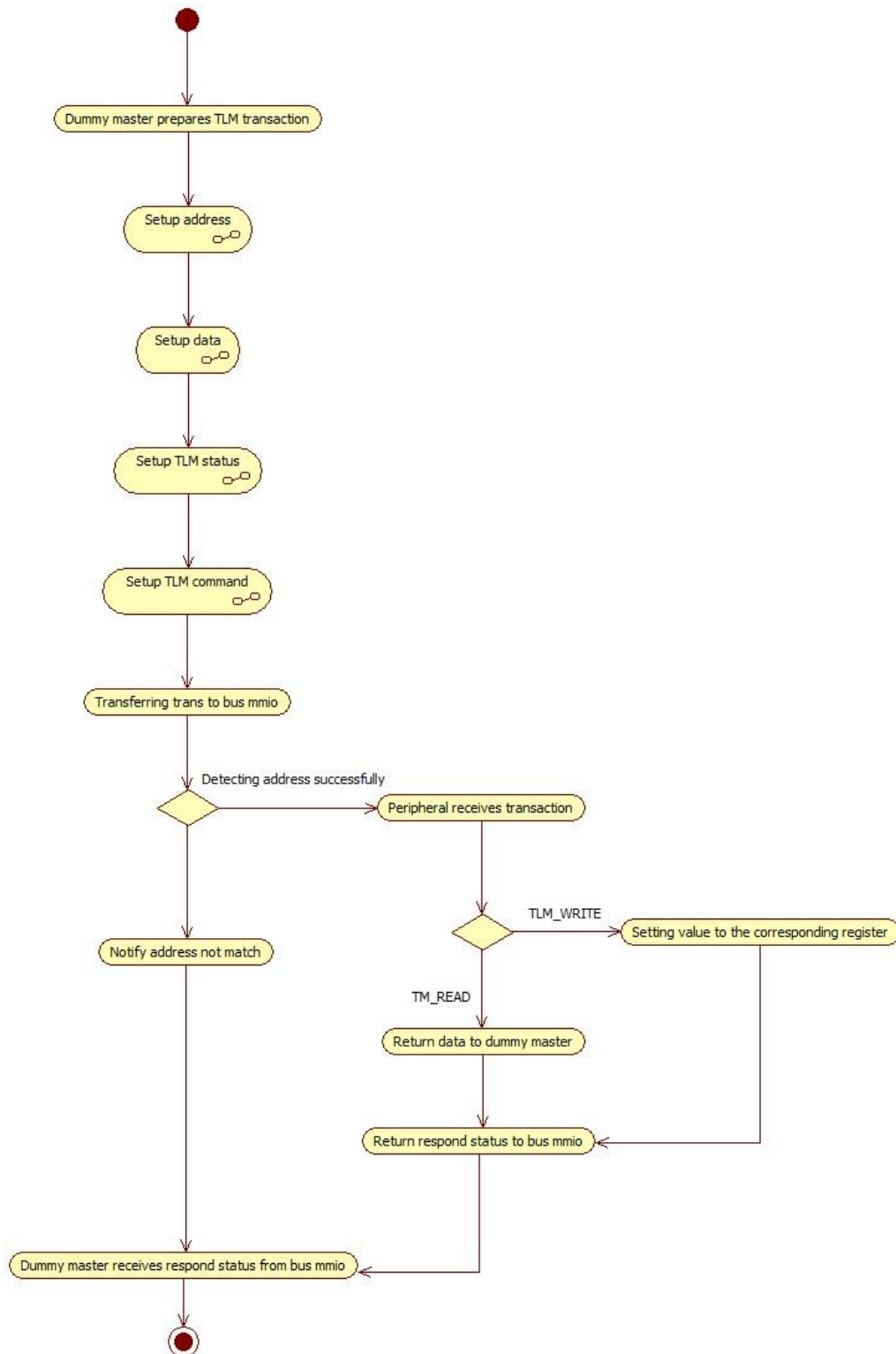


Figure 2: Transferring data using dummy master



The fig 2 shows operation using dummy master access to peripheral model in SoC platform. The dummy master provides application programming interface to setup TLM transaction, user will define the address of register and the value for write operation and 0x0 value for read operation. When TLM transaction is forwarded to peripheral successfully, peripheral will check the TLM command and handling another operation.

### 3.2. Setting registers using dummy master

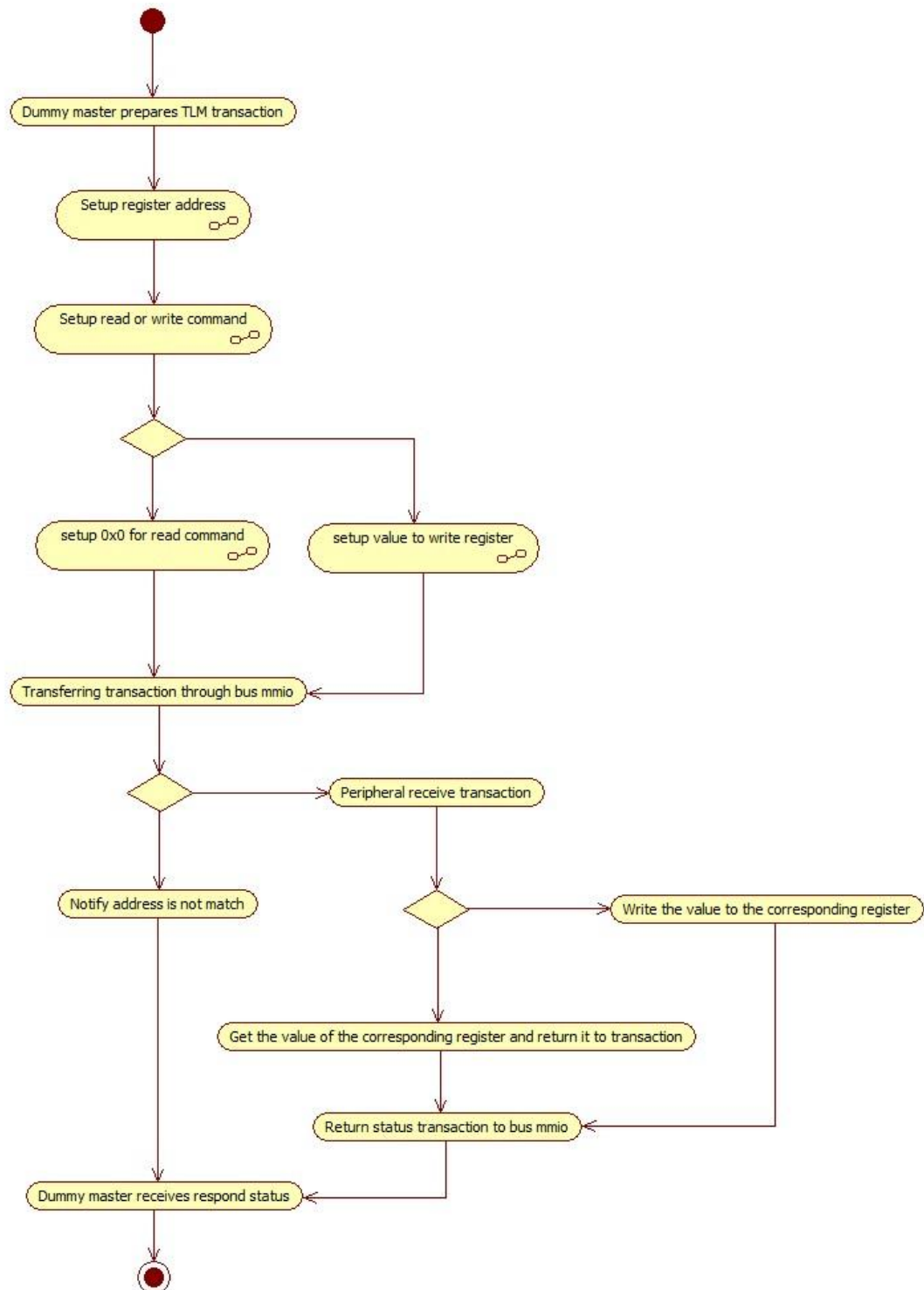


Figure 3: Read or Write register operation

The fig 3 shows the read / write register operation of SoC platform. User define the address space of each peripherals in the top module that is used to mapping with all registers, to read or write register of each peripheral, user writes or reads to the corresponding register address.

### 3.3. Monitoring internal ports using dummy slave

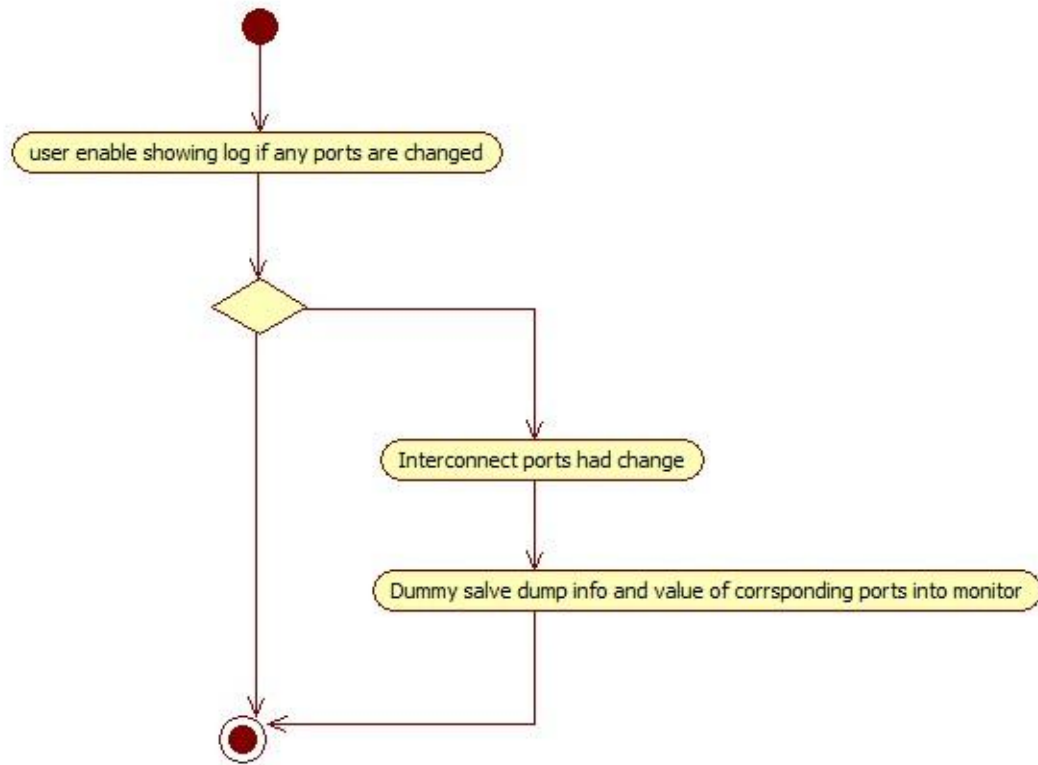


Figure 4: Monitoring ports using dummy slave

For debug demand, the SoC platform provides a method to monitor all interconnected signals in its. If any ports change its status, the dummy slave will show log in the monitor and user can use it for debug demand.

### 3.4. Register interface implementation

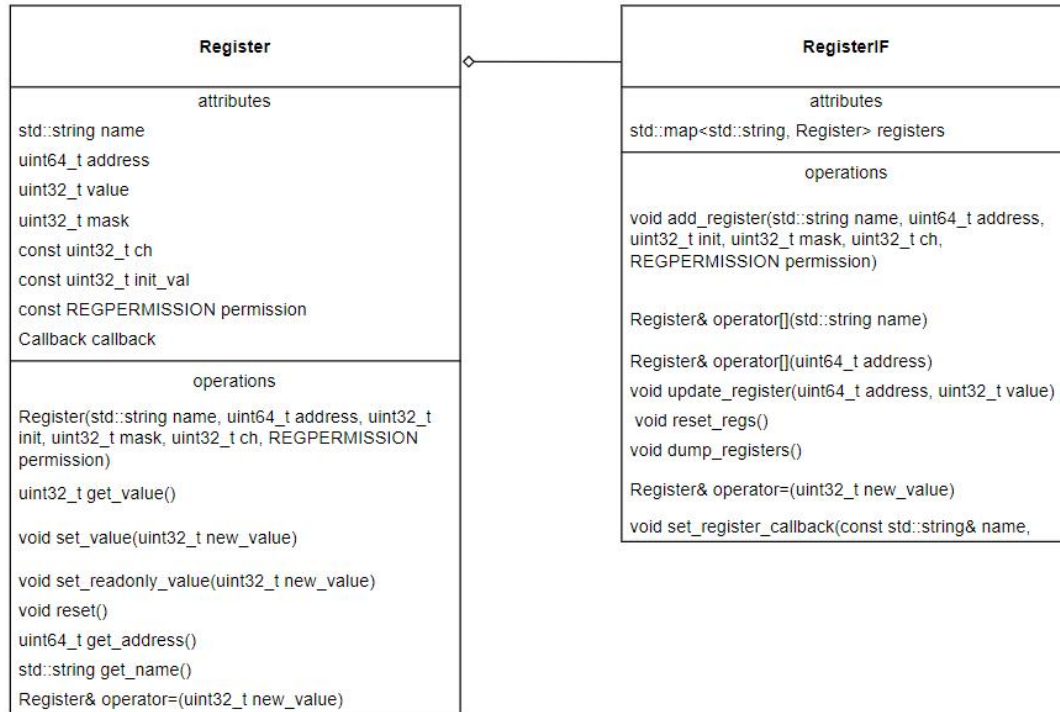


Figure 5: Register interface support

The SoC platform offers a register interface that support to instance all registers for each model. The address of registers will be mapped into bus MMIO and register the specific addresses. User can access to registers through the address of TLM transaction. Specially, this interface support for callback registration, if user register callback function for specific register, when it change the value, the callback function is called. It is helpful for implement HW simulation.

### **3.5. External memory**

The SoC platform support ram model that is connected directly to bus MMIO, it is used for further DMA operation.

### 3.6. Wrapper for connection with RTL model

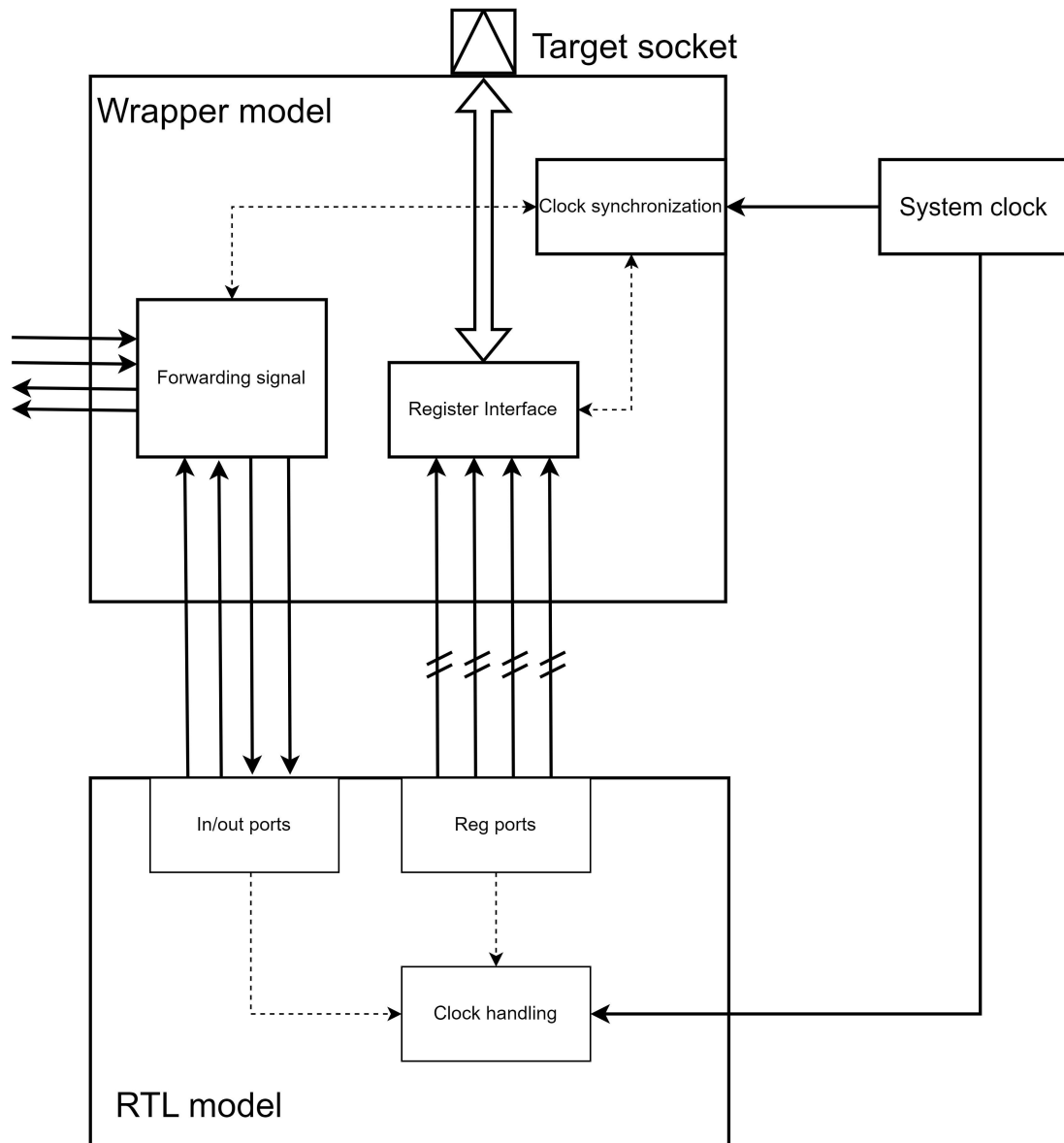


Figure 6: Wrapper common architecture for connecting RTL model

The fig 6 illustrates the idea for implementation wrapper that converts bus MMIO interface to reg ports and forwards in/out signals.

## 4. TOP MODULE

In this section, the document will show the illustration of top module for SoC platform.

### Instance internal SoC model

```
DummyMaster<32> m_dummymaster;  
DummySlave<32> m_dummyslave;  
BUS<APB> m_bus;  
Target<32> m_target1;  
Target<32> m_target2;  
Target<32> m_target3;  
Target<32> m_target4;  
RAM<32> m_ram1;  
wrapper_four_bit_adder<32> m_wrapper_four_bit_adder;  
wrapper_counter<32> m_wrapper_counter;  
wrapper_uart<32> m_wrapper_uart;
```

### Mapping address of peripheral into bus MMIO

```
void do_bus_binding()  
{  
    // Connecting Initiator socket to bus MMIO  
    m_dummymaster.initiator_socket.bind(m_bus.target_sockets);  
    m_dmac.initiator_socket.bind(m_bus.target_sockets);  
  
    // Binding target sockets into bus MMIO  
    m_bus.mapping_target_sockets(BASE_RAM1,SIZE_RAM1).bind(m_ram1.socket);  
  
    m_bus.mapping_target_sockets(BASE_DUMMYSLAVE,SIZE_DUMMYSLAVE).bind(m_dummyslave.target_s  
ocket);  
  
    m_bus.mapping_target_sockets(BASE_TARGET1,SIZE_TARGET1).bind(m_target1.target_socket);  
    m_bus.mapping_target_sockets(BASE_TARGET2,SIZE_TARGET2).bind(m_target2.target_socket);  
    m_bus.mapping_target_sockets(BASE_TARGET3,SIZE_TARGET3).bind(m_target3.target_socket);  
    m_bus.mapping_target_sockets(BASE_TARGET4,SIZE_TARGET4).bind(m_target4.target_socket);  
  
    m_bus.mapping_target_sockets(BASE_FOUR_BIT_ADDER,SIZE_FOUR_BIT_ADDER).bind(m_wrapper_fo  
ur_bit_adder.target_socket);  
    m_bus.mapping_target_sockets(BASE_COUNTER,SIZE_COUNTER).bind(m_wrapper_counter.target_s  
ocket);  
    m_bus.mapping_target_sockets(BASE_UART,SIZE_UART).bind(m_wrapper_uart.target_socket);  
}
```



## Binding signal

```
void do_signals_binding()
{
    m_dummyslave.add_input_port("system_reset")->bind(m_sysrst);
    // binding system clock
    m_bus.m_clk(m_sysclk);
    m_dmac.clk.bind(m_sysclk);
    m_dummymaster.clk.bind(m_sysclk);
    m_dummyslave.clk.bind(m_sysclk);
    m_dummyslave.rst(m_sysrst);
    m_wrapper_counter.m_clk(m_sysclk);
    m_wrapper_uart.m_clk(m_sysclk);

    // binding reset
    m_dummymaster.rst.bind(m_sysrst);
    m_bus.m_rst.bind(m_sysrst);
    m_dmac.rst.bind(m_sysrst);
    m_wrapper_uart.m_rst(m_sysrst);

    /* Counter */
    m_wrapper_counter.m_clear(counter_clear);
    m_wrapper_counter.m_load(counter_load);
    m_wrapper_counter.m_start(counter_start);
    m_dummyslave.add_output_port("counter_clear")->bind(counter_clear);
    m_dummyslave.add_output_port("counter_load")->bind(counter_load);
    m_dummyslave.add_output_port("counter_start")->bind(counter_start);

    /* UART */
    m_wrapper_uart.m_nrw(uart_nrw);
    m_wrapper_uart.m_cs(uart_cs);
    m_wrapper_uart.m_sin(uart_sin);
    m_wrapper_uart.m_int2(uart_int2);
    m_wrapper_uart.m_sout(uart_sout);
    m_dummyslave.add_output_port("uart_nrw")->bind(uart_nrw);
    m_dummyslave.add_output_port("uart_cs")->bind(uart_cs);
    m_dummyslave.add_output_port("uart_sin")->bind(uart_sin);
    m_dummyslave.add_input_port("uart_int2")->bind(uart_int2);
    m_dummyslave.add_input_port("uart_sout")->bind(uart_sout);
}
```