SystemC/TLM

Emad Ebeid
PhD Student
Department of Computer Science
University of Verona
Italy

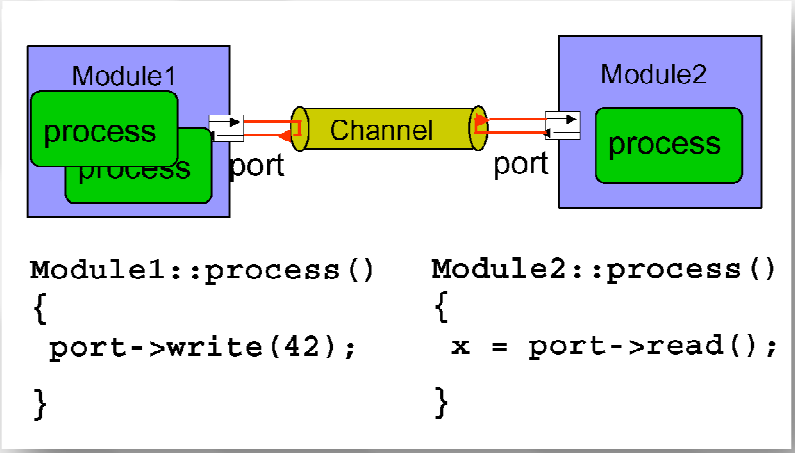
Davide Quaglia
Assistant Professor
Department of Computer Science
University of Verona
Italy

outline

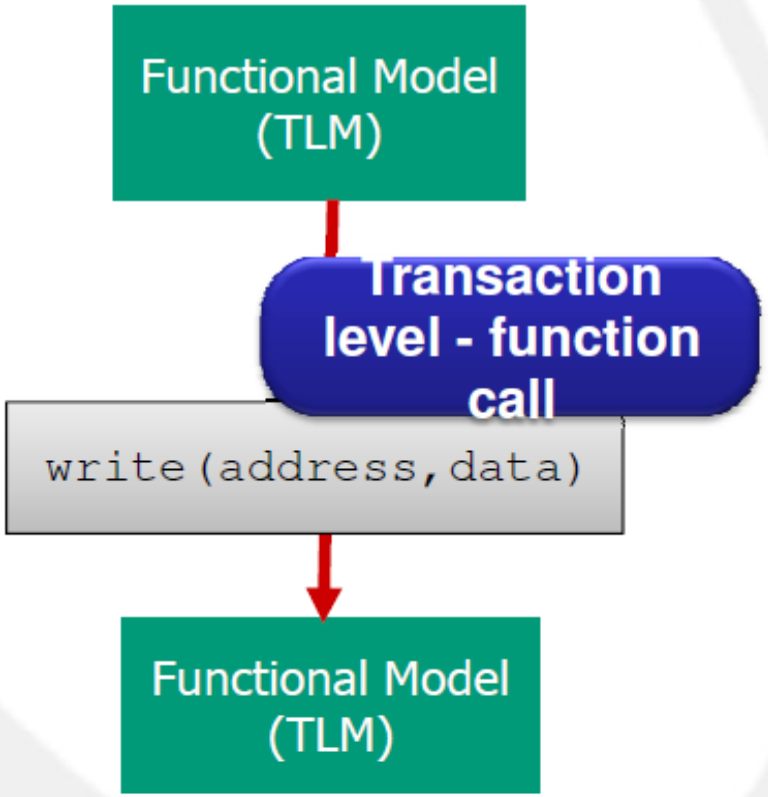
- Introduction
  - Reasons for using TLM
  - TLM-based design flows
- Transaction
- Socket
- Initiator
- Target

Introduction

- Transaction-level modeling (TLM) is a high-level approach to modeling digital systems where details of communication among modules are separated from the details of the implementation of functional units or of the communication architecture.



Introduction



Simulate every event

100-10,000 X faster simulation

TLM: Modeling Comparison

- More emphasis on the data transfer functionality
  - less on their implementation details at the early design stage

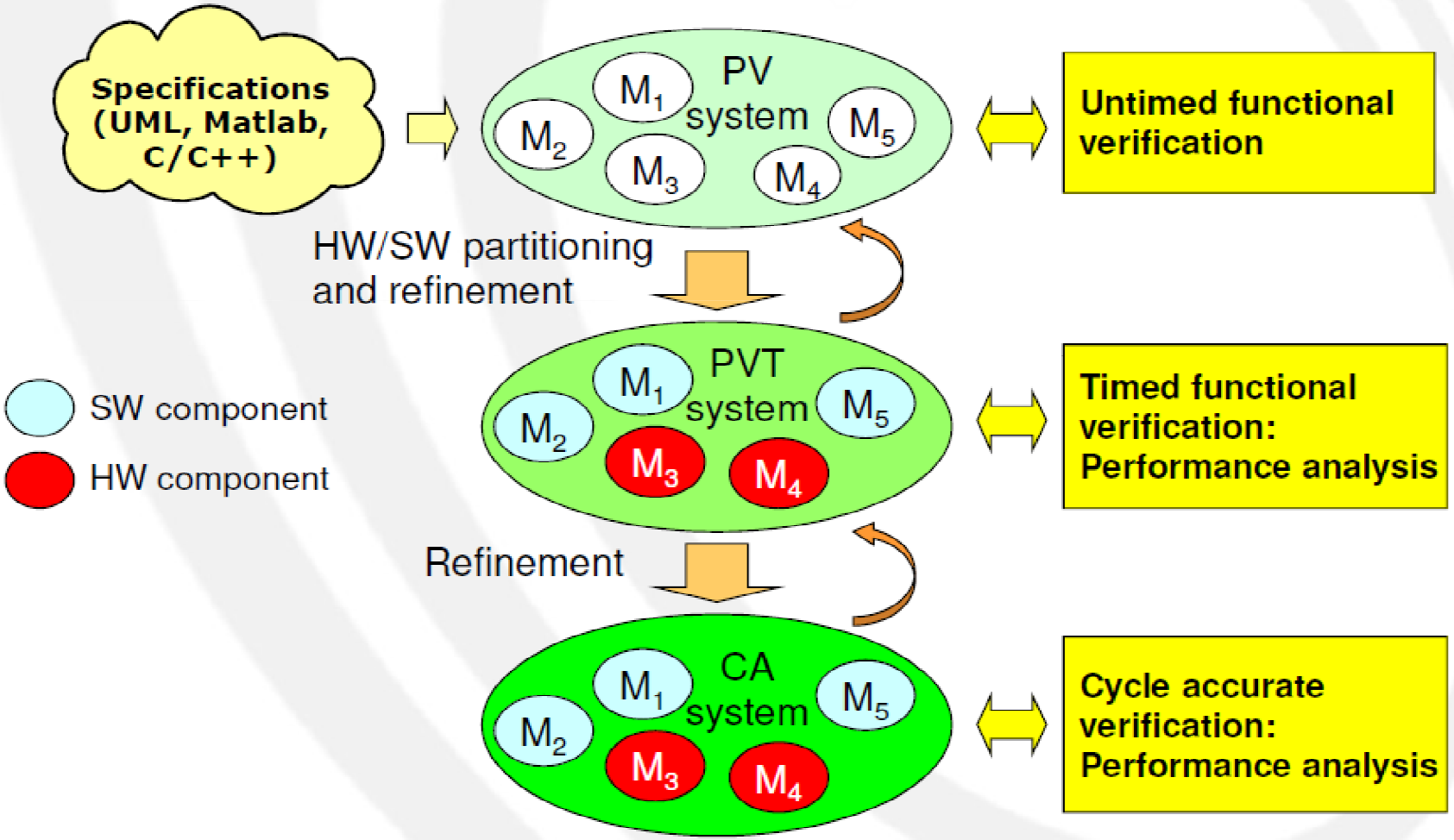
RTL

```
process(clock)
IF (clock'event and clock = '1')
THEN
CASE fsm_state IS:
        WHEN s0 =>
          request_port <= '1';
          fsm_state := s1;
        WHEN s1 =>
          IF (grant_port = '1')
            THEN
            fsm_state := s2;
        WHEN s2 =>
          data_port <= data;
...       addr_port <= addr;
```

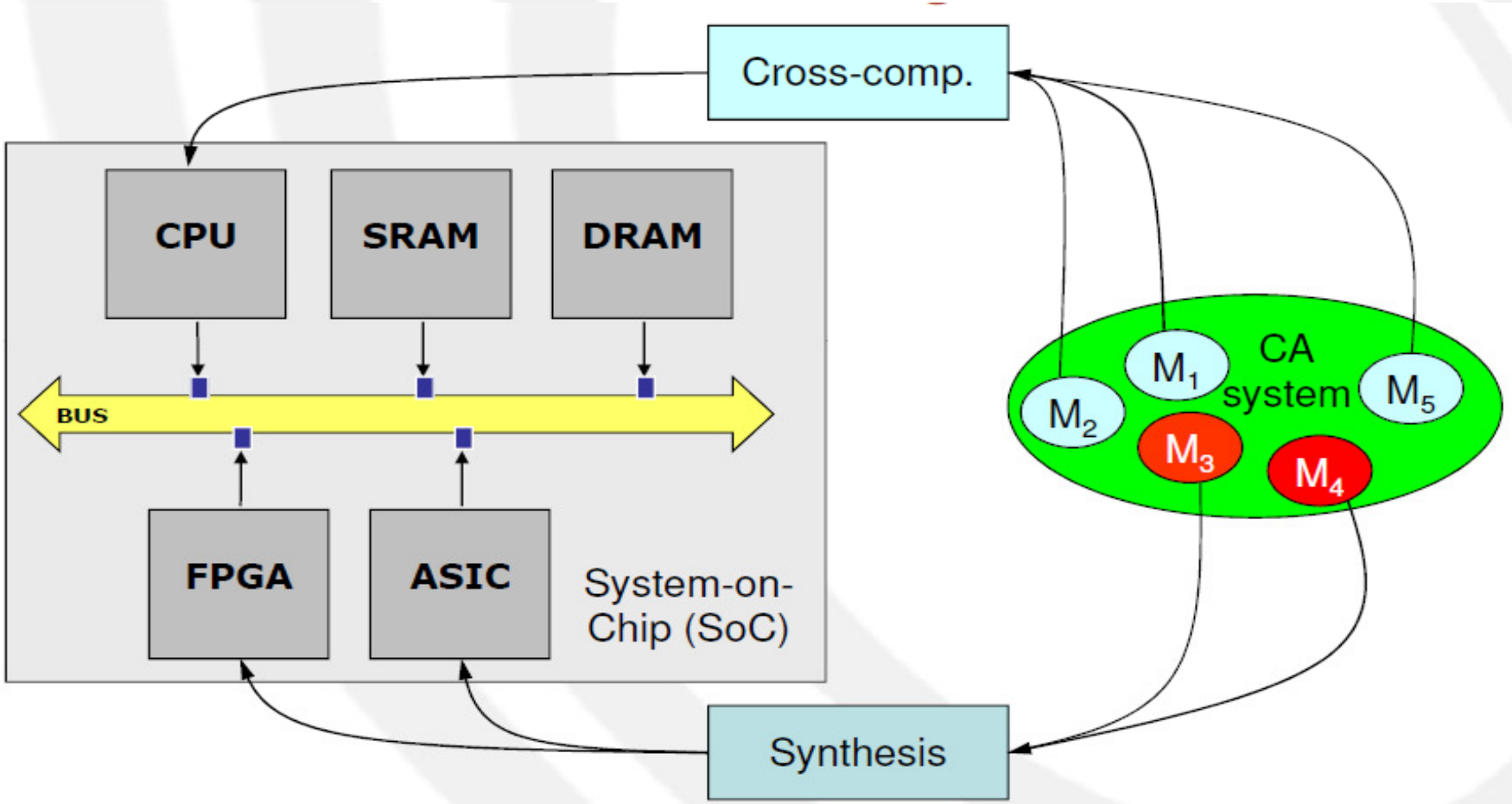
TLM

⇨ **write (data, addr);**

TLM-based design flow



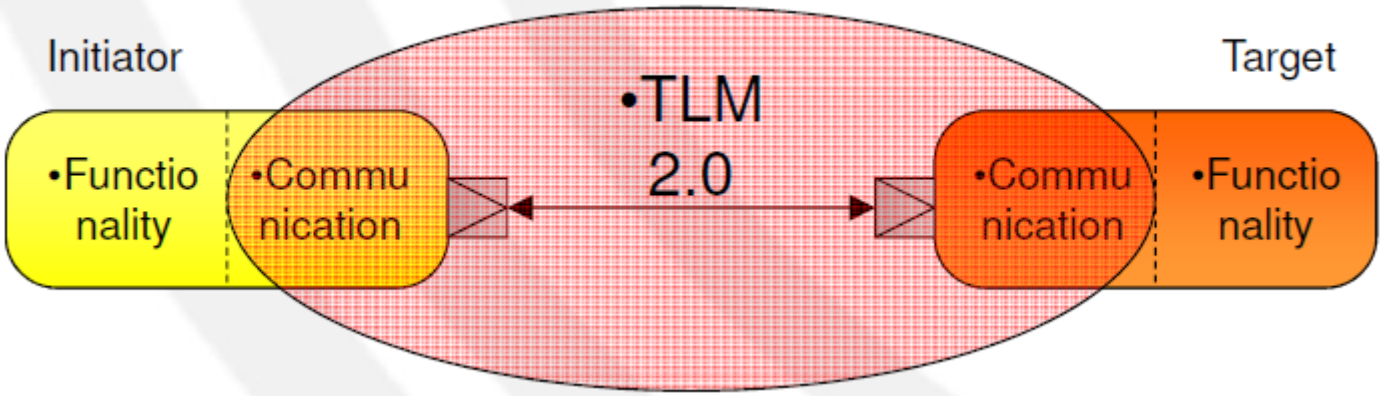
TLM-based design flow



Functionality vs communication

• TLM manages to keep distinct functionality and communication in each module

• TLM 2.0 allows to model the communication part (i.e. how each module interacts with the others)



Transactions

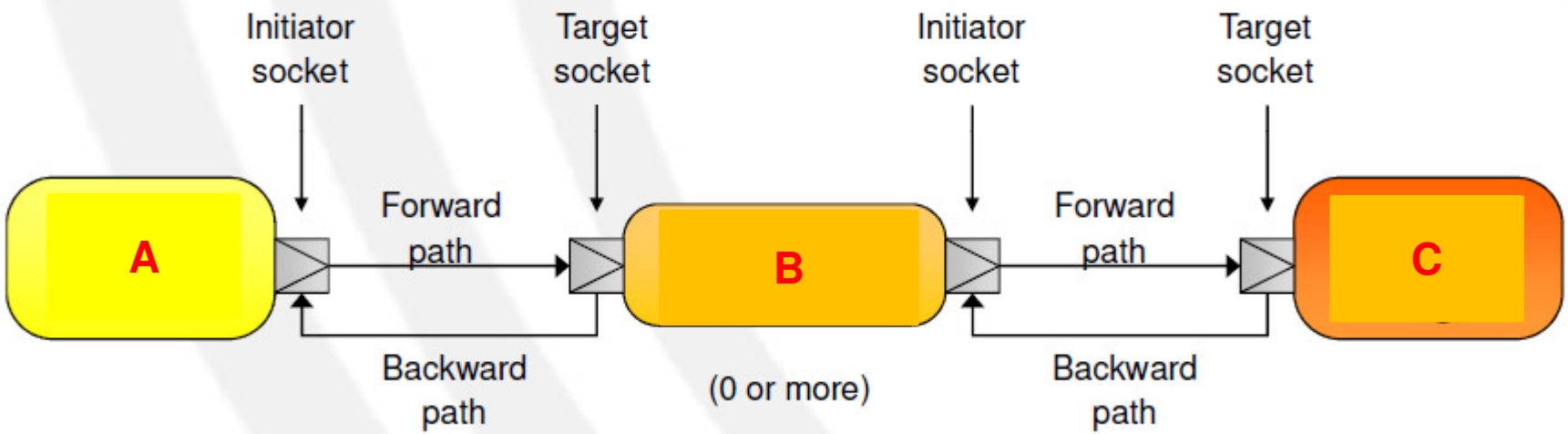
- TLM relies on the notion of *transaction*
- A *transaction* consists of a data transfer from a design module to another one
  - Write and read operations are examples of transactions
  - It is usually represented by a *generic payload* object in the code
    - This object contains both data and control information (e.g. address and type of command)
  - It is exchanged between modules through primitive calls

Socket, initiator and target

Communication is achieved by exchanging packets between an *initiator* module and a *target* module, through a *socket*

- The *initiator* starts a transaction
- The *target* is the end point of a transaction
- An *interconnect component* is an intermediate point in the path from the *initiator* to the *target*
- A *socket* connects two modules, and allows them to communicate by means of the available interfaces
- The *forward path* runs from the *initiator* to the *target*
- The *backward path* runs from the *target* to the *initiator*

Socket, initiator and target



Blocking interface

- Appropriate where an initiator wishes to complete a transaction with a target in a **single function call**
- Two timing points
  - Call to and return from the blocking transport function
- It only uses the forward path from initiator to target

Transaction type

```
template < typename TRANS = tlm_generic_payload >
class tlm_blocking_transport_if : public virtual
sc_core::sc_interface {
public:
    •virtual void b_transport ( TRANS& trans ,sc_core::sc_time& t
    ) = 0;
```

Timing annotation