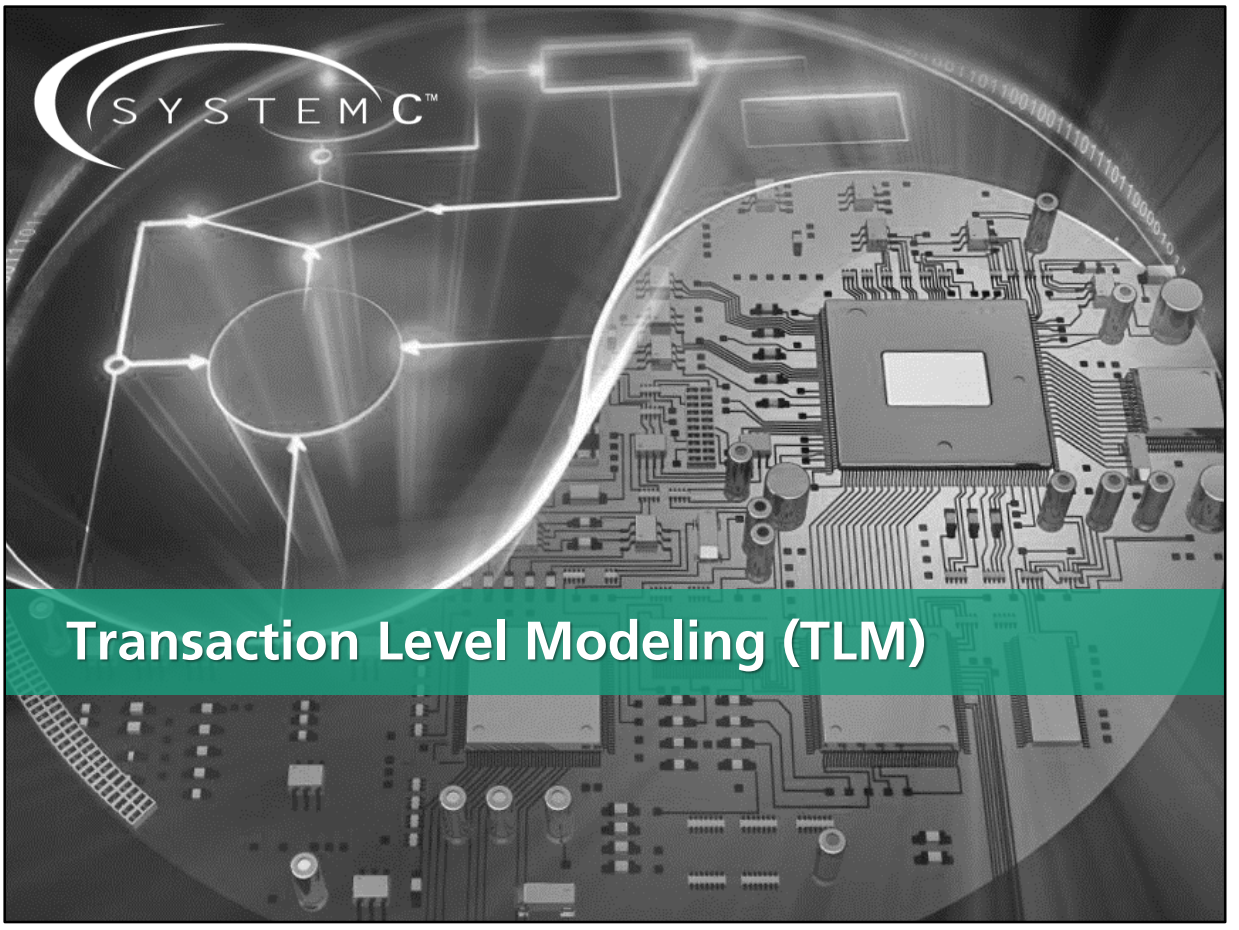
SystemC and Virtual Prototyping

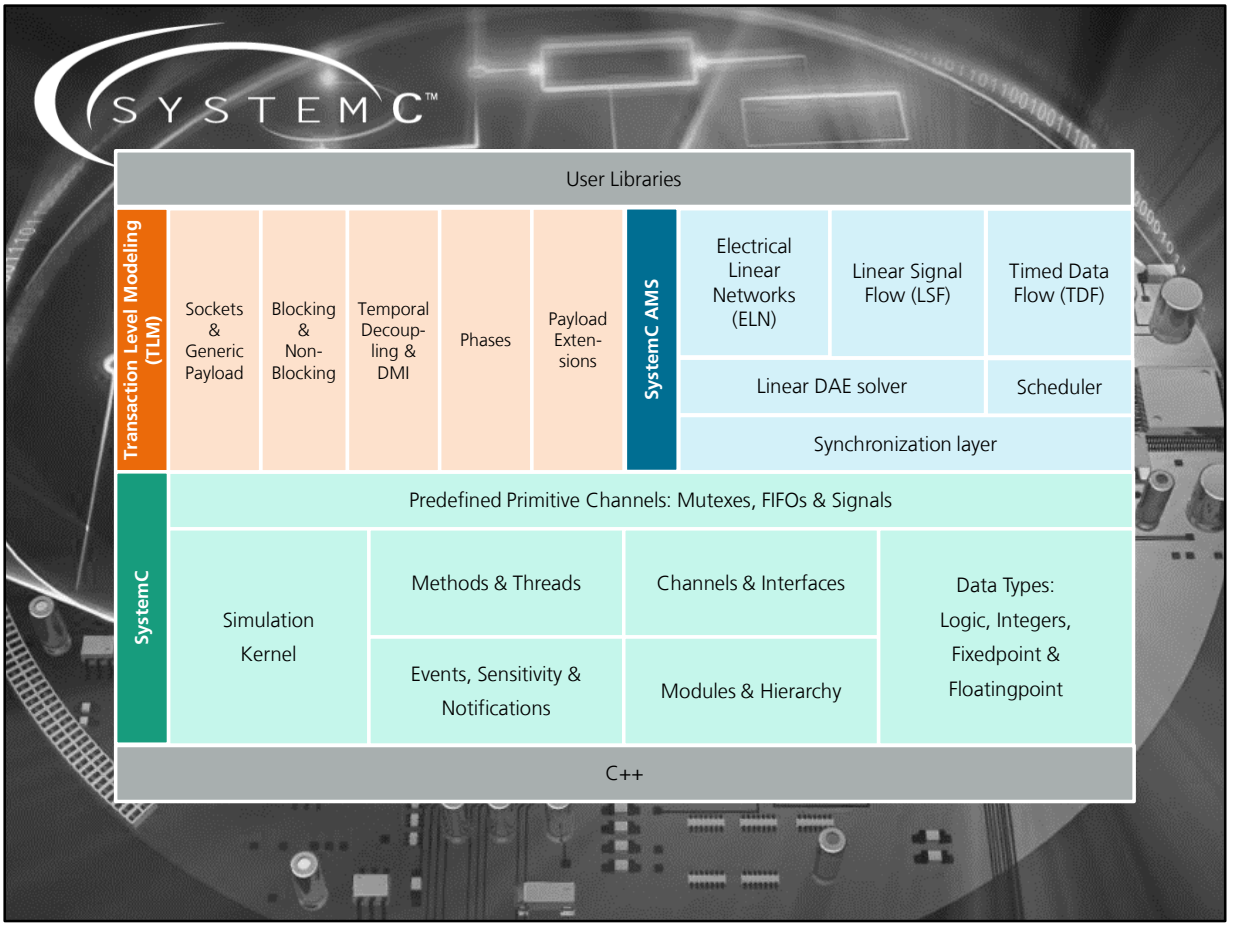
Dr. Matthias Jung, Fraunhofer Institute IESE
matthias.jung@iese.fraunhofer.de

Fraunhofer IESE

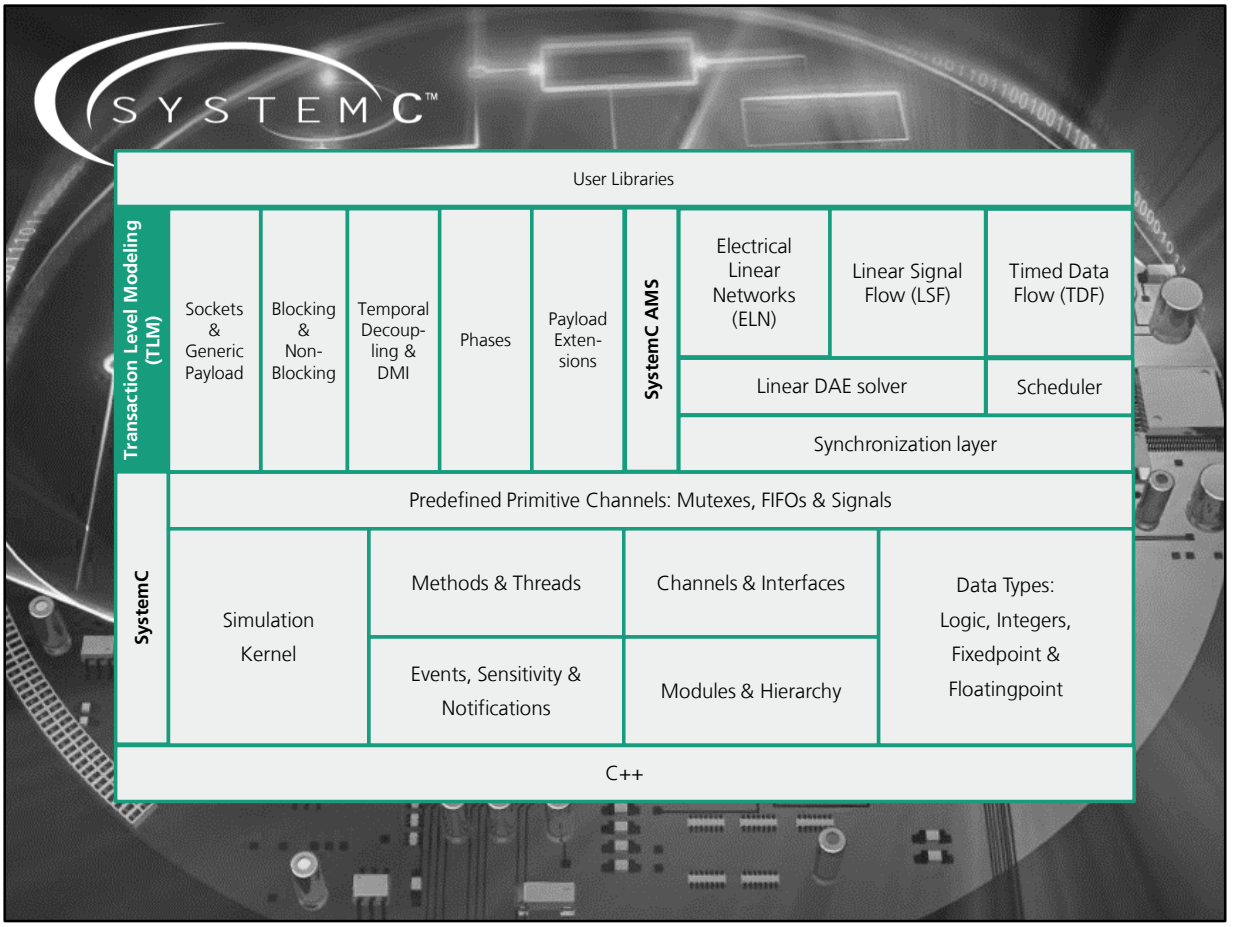
Your notes: TLM has several Topics, we will talk more about the trades of FF between simulation speed and accuracy, in TLM we are raising the Abstraction to higher level which gives us higher speed But accuracy goes down, But we can also find some cases that accuracy doesn't go down and also I will show you some examples that they did where we can get up to 10x speed by having same accuracy by just applying some clever modeling techniques



Your notes:

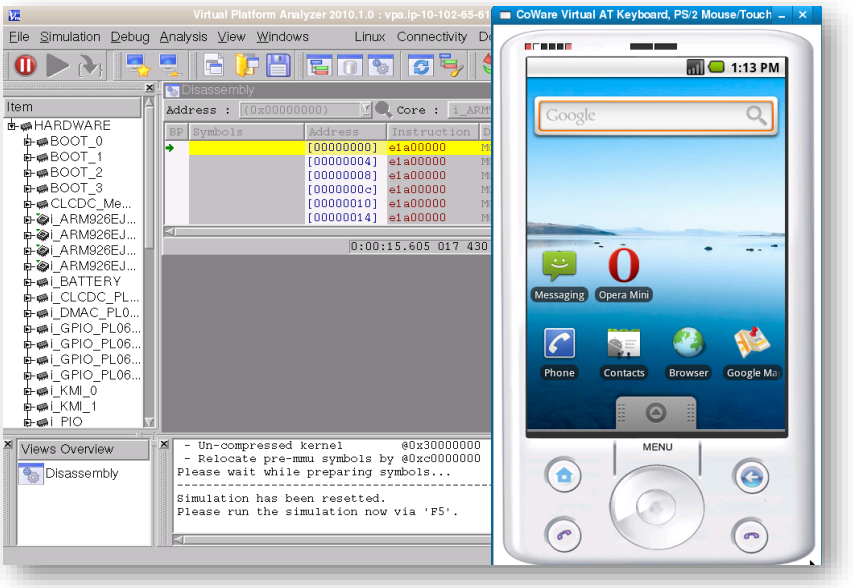


Your notes:



Your notes:

How to build a virtual Smartphone?



Fraunhofer IESE

Your notes:

- If we model the complete SmartPhone in RTL, if you model everything and everything clocked and everything is detailed the simulation will take 100 years to complete Linux Boot.

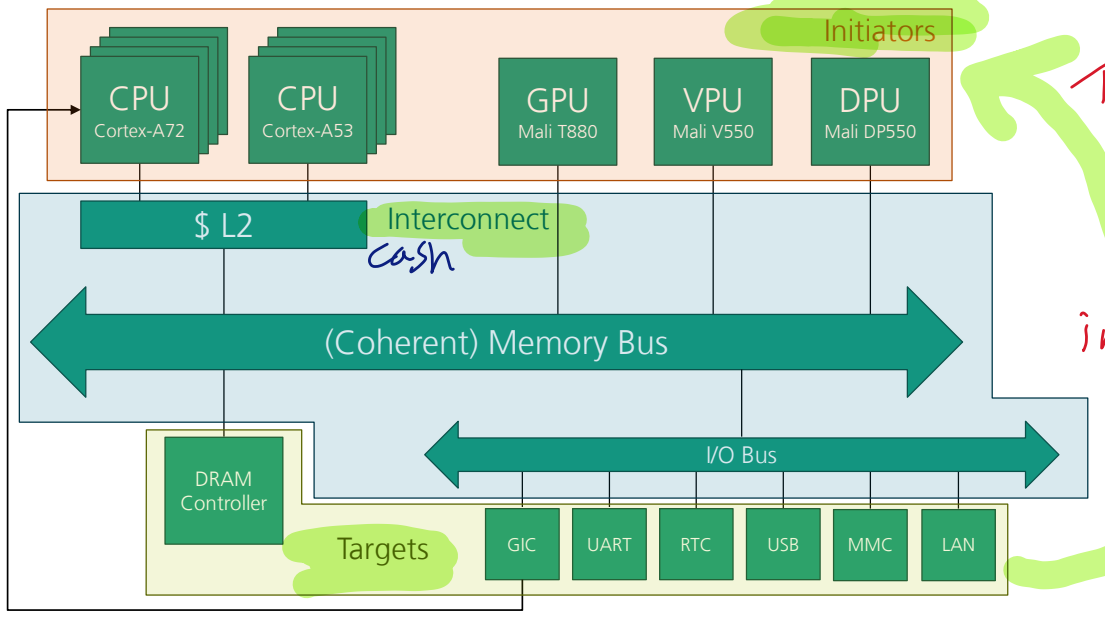
- So for very accurate models it takes so long time because we have so many events that are happening.

- So what we are trying to do in TLM is to reduce the amount of events

we don't want to simulate the clock because every clock is an event it's signal change in our systemc kernel and our kernel needs to do something for that change and this gives overhead.

- By the way here is a hint to speed up our simulation is to clock gate the clock for ex, that's something we did in one project, so for ex if there is a component is not required for simulation we shut down the clock of the component and the simulation get faster, so the simulation gets faster, that's kind of optimization in RTL simulation.

Typical Smartphone SoC Consists of the following



Fraunhofer IESE

Your notes:

When you see now such a system on chip how you classify this components

① initiators because they initiates communication

② interconnects initiators and Targets at same time

③ Targets because memory controller will not suddenly send data to CPU, No CPU requested this data

…s a active component but it's pass ve
From Comication Point of view

Recap: Accuracy vs. Speed Trade-Off

That's how we classify our models in TLM -

8

Fraunhofer IESE

Your notes: You should always ask what you want to have what you want to answer by this simulation.

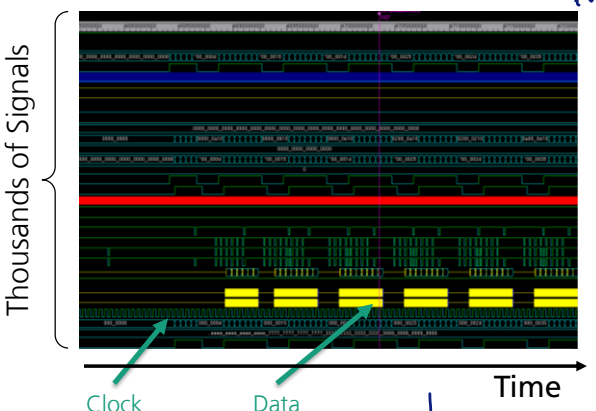
Recap: Accuracy vs. Speed Trade-Off

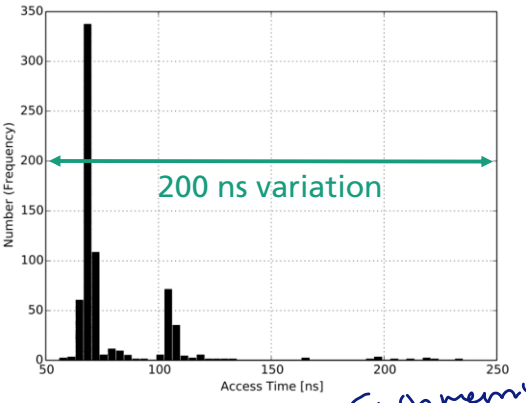
E.g. RTL Simulations:

- VHDL / Verilog / SystemC
- Very accurate
- Very very very … slow
- Inflexible

E.g. System Level Simulations:

- Fast
- Large flexibility
- Inaccurate



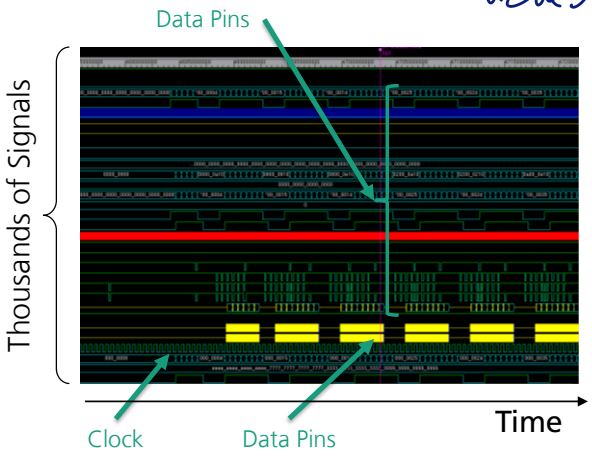


Fraunhofer IESE

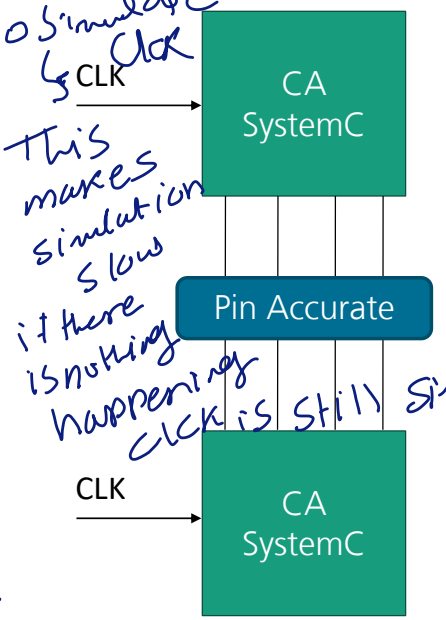
Your notes:

in system C we can Model like RTL
our Models can have thousands of Pins we simulate all the events

Remember: Cycle Accurate Simulation



we also simulate Clock



This makes simulation slow

if there is nothing happening CLK is still Simulating

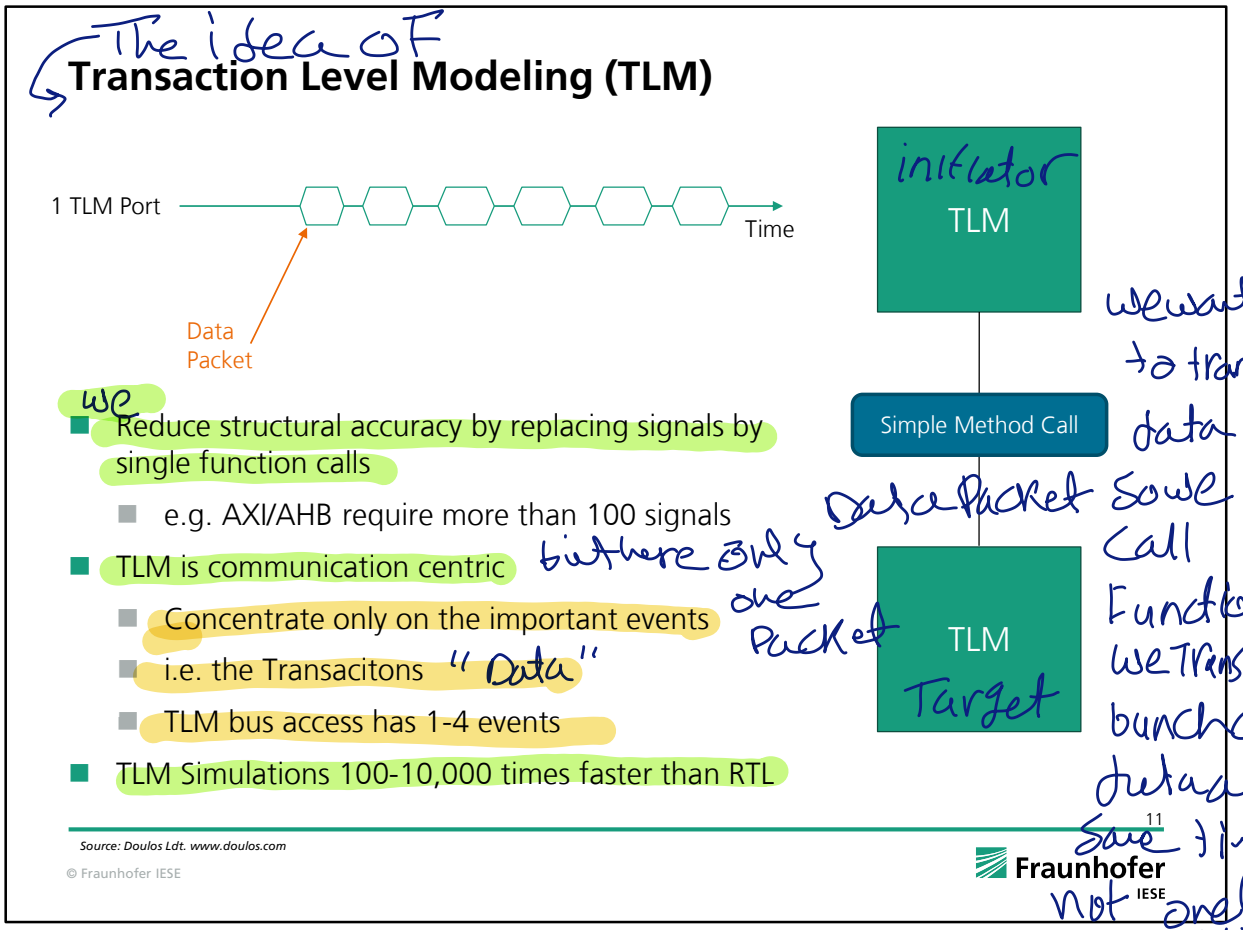
- RTL models can have thousands of pins
- Simulate all events on all pins
  - RTL bus access has ~75 events

10

Source: Doulos Ldt. www.doulos.com

Fraunhofer IESE

Your notes:



Transaction level modeling is speeding up simulation by replacing all pin-level events with a single method call. In an RTL simulation the communication requires multiple events at the level of individual bits. In a transaction-level model, complete bus cycles can be modeled with one method call. Therefore, TLM models can be 100 to 10000 times faster than their RTL counter parts. However, there is no free lunch: the increased speedup of simulation is payed with reduced accuracy of the results.

Your notes: What is actually based on Target will have a function and the initiator will call a function of this Component and in the Parameter we will put a packet and we will not do Call by value here

but we will do this by call by reference (Pointer)

- So we will create in the initiator this data Packet and we won't transfer the whole Packet a sucopy we just transfer a Pointer, just 64 bit number Transfered and this make simulation faster.

Transaction: A custom TLM Implementation

```
#include <iostream>
#include <systemc.h>
#include <queue>

using namespace std;

enum cmd {READ, WRITE};

struct transaction {
    unsigned int data;
    unsigned int addr;
    cmd command;
};

class transactionInterface : public sc_interface {
    public:
    virtual void transport(transaction &trans) = 0;
};
```

Try code on github:
https://github.com/TUK-SCVP/SCVP.artifacts/tree/master/custom_tlm

Consumer Module as Hierarchical Channel

Fraunhofer IESE

Your notes:

- As you remember Hierachal chanel: it's Channel that is not only inherit From SC_InterFace but it also inherit From SC_Module, so it's a channel that can have processes, that's the trick in TLM that we say we have some components but the components have an interFace like channal and we can connect and bind them.

Transaction: A custom TLM Implementation

```
SC_MODULE(PRODUCER)
{
    sc_port< transactionInterface > master;

    SC_CTOR(PRODUCER)
    {
        SC_THREAD(process);
    }

    void process()
    {
        for(unsigned int i=0; i < 4; i++) {
            wait(1,SC_NS);
            transaction trans;
            trans.addr = i;
            trans.data = rand();
            trans.command = cmd::WRITE;
            master->transport(trans);
        }
        for(unsigned int i=0; i < 4; i++) {
            wait(1,SC_NS);
            transaction trans;
            trans.addr = i;
            trans.data = 0;
            trans.command = cmd::READ;
            master->transport(trans);
            cout << trans.data << endl;
        }
    }
};
```

```
class CONSUMER : public sc_module,
                 public transactionInterface
{
    private:
    unsigned int memory[1024];

    public:
    SC_CTOR(CONSUMER) {
        for(unsigned int i=0; i < 1024; i++) {
            memory[i] = 0; // Initialize memory
        }
    }

    void transport(transaction &trans) {
        if(trans.command == cmd::WRITE) {
            memory[trans.addr] = trans.data;
        }
        else /* cmd::READ */ {
            trans.data = memory[trans.addr];
        }
    }
};

int sc_main(…) {
    PRODUCER cpu("cpu");
    CONSUMER mem("memory");
    cpu.master.bind(mem);
    sc_start();
    return 0;
}
```

Write

Read

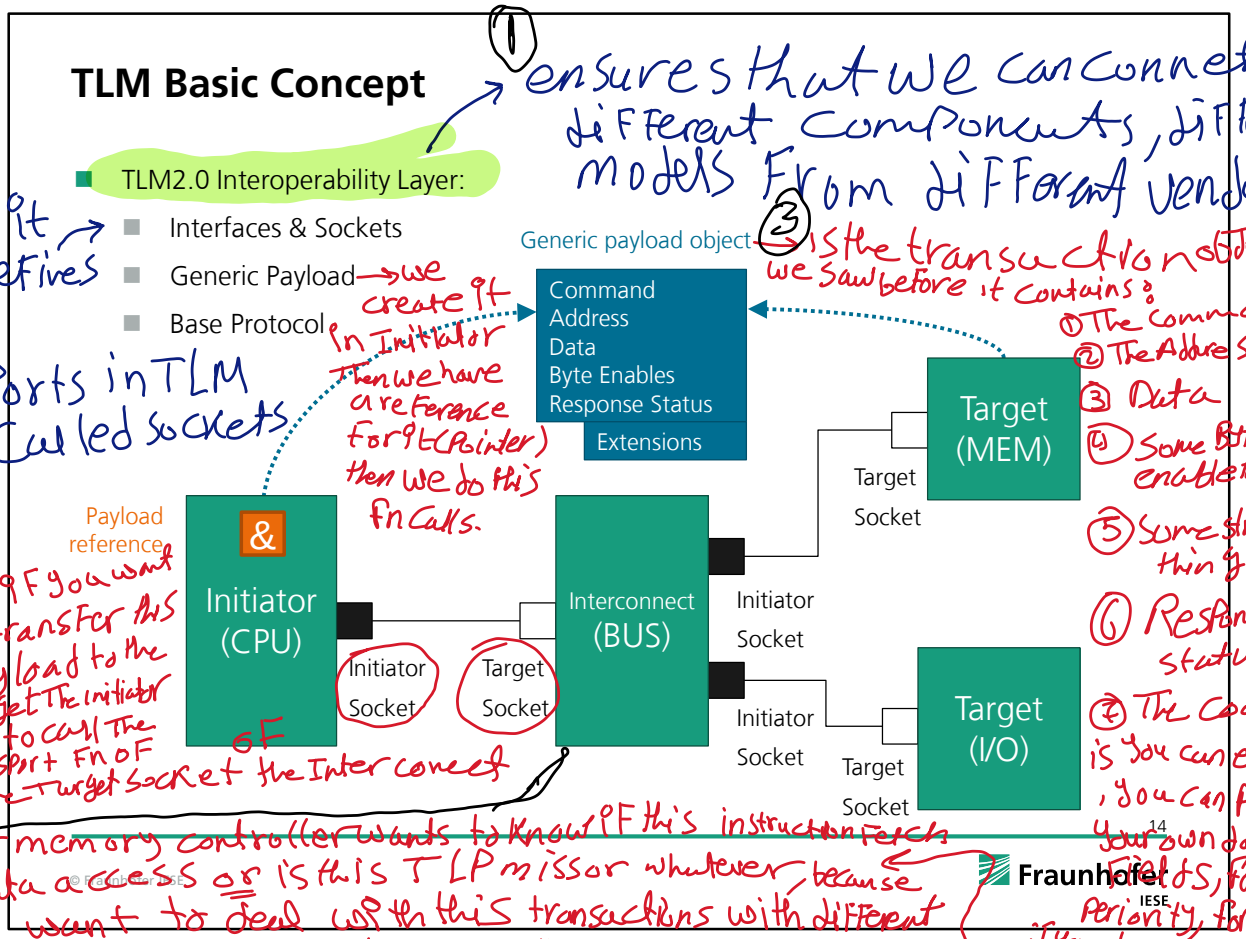
The Produce is active, the consumer is passive

Fraunhofer IESE

Your notes:

You would notice that we don't have delta cycle, This makes simulation faster because we JUST Communicate with function calls and the transport call is blocking call, your Simulation is trapped inside this Fn, thats why we call it Blocking Transport. execution

- later on we will learn that calling wait statements in Targets in this Transport Function within the Target, it's not a good Practice as it make simulation slower we will see that today



ensures that we can connect different components, different models from different vendors

it defines

3 is the transaction object we saw before it contains:
① The command
② The Address
③ Data
④ Some Byte enable things
⑤ Some streaming thing
⑥ Response status
⑦ The cool thing is you can extend it, you can put your own data fields, for ex: priority, for ex if you do memory Access

Ports in TLM Called sockets

we create it in Initiator Then we have a reference for it (Pointer) then we do this fn calls.

5 So if you want to transfer this payload to the target the initiator have to call the transport fn of the target socket of the Interconnect

4 your memory controller wants to know if this instruction fetch or data access or is this TLP missor whatever, because you want to deal with this transactions with different priorities and so on, that's cool thing you can extend it easily.

Your notes:

2 For ex: if you get a System-C module from ARM for a cortex M0, you can plug it with a sensor model that you have developed, just Plug and Play because of Interoperability layer, it's defined in standard and if everyone follows the rules of standard, so we are interoperable and we can Plug models.

6 The interconnect has to check the address which target it will go then the initiator socket of the interconnect will call the transport function of the target socket here.

Note: Target Socket Symbol

[redacted] Initiator Socket Symbol -

7

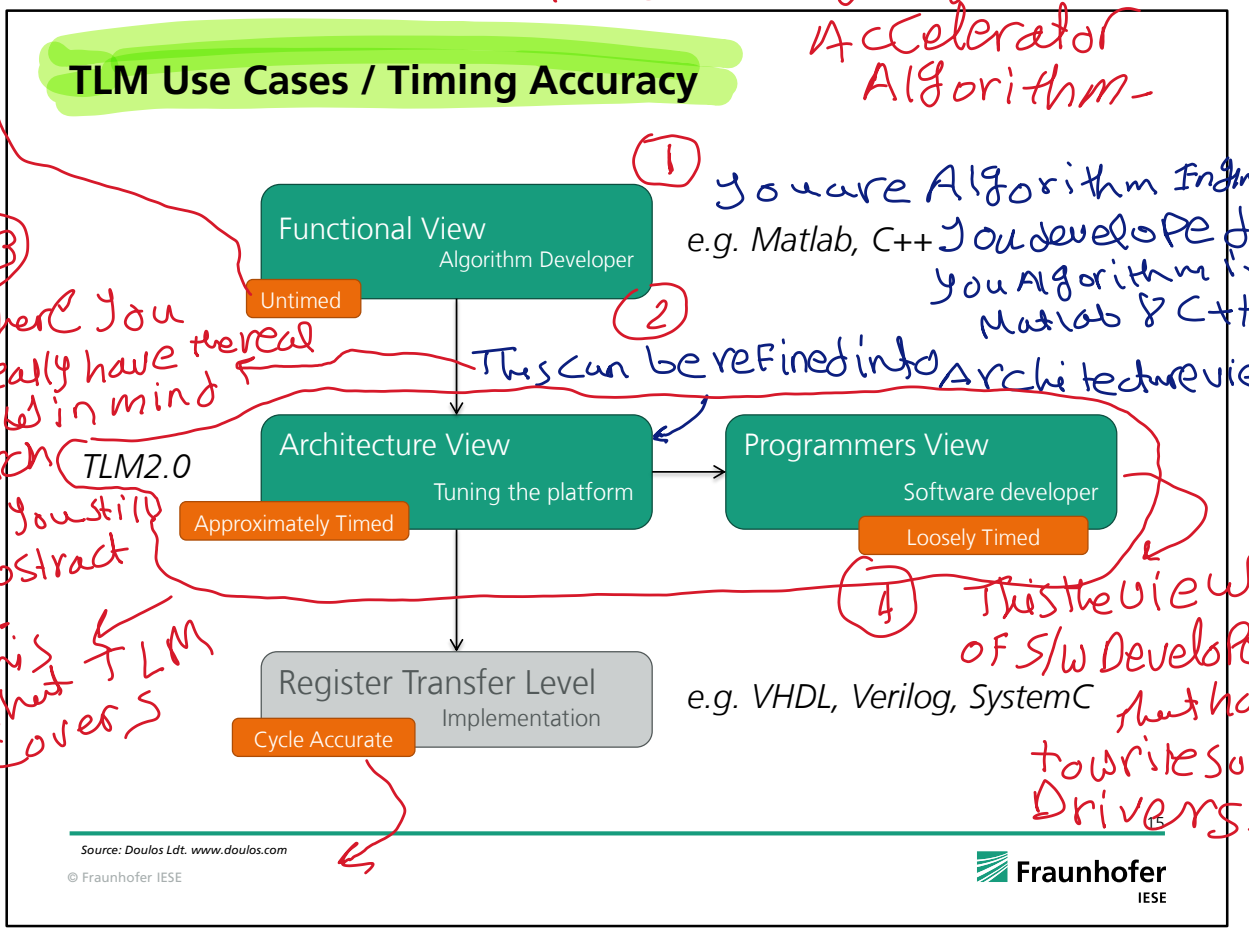
how we transfer things, through function calls we get the reference over there in the memory then we do some access on the data of Generic payload object then it go back to initiator

Base Protocol : it describe how this transport and these function calls should happen

Note

we create the Payload in the initiator and have a pointer to it in the Generic Payload object.

6 we usually call it untimed, we don't simulate time when we build Algorithm like Accelerator Algorithm.



7 Your notes:

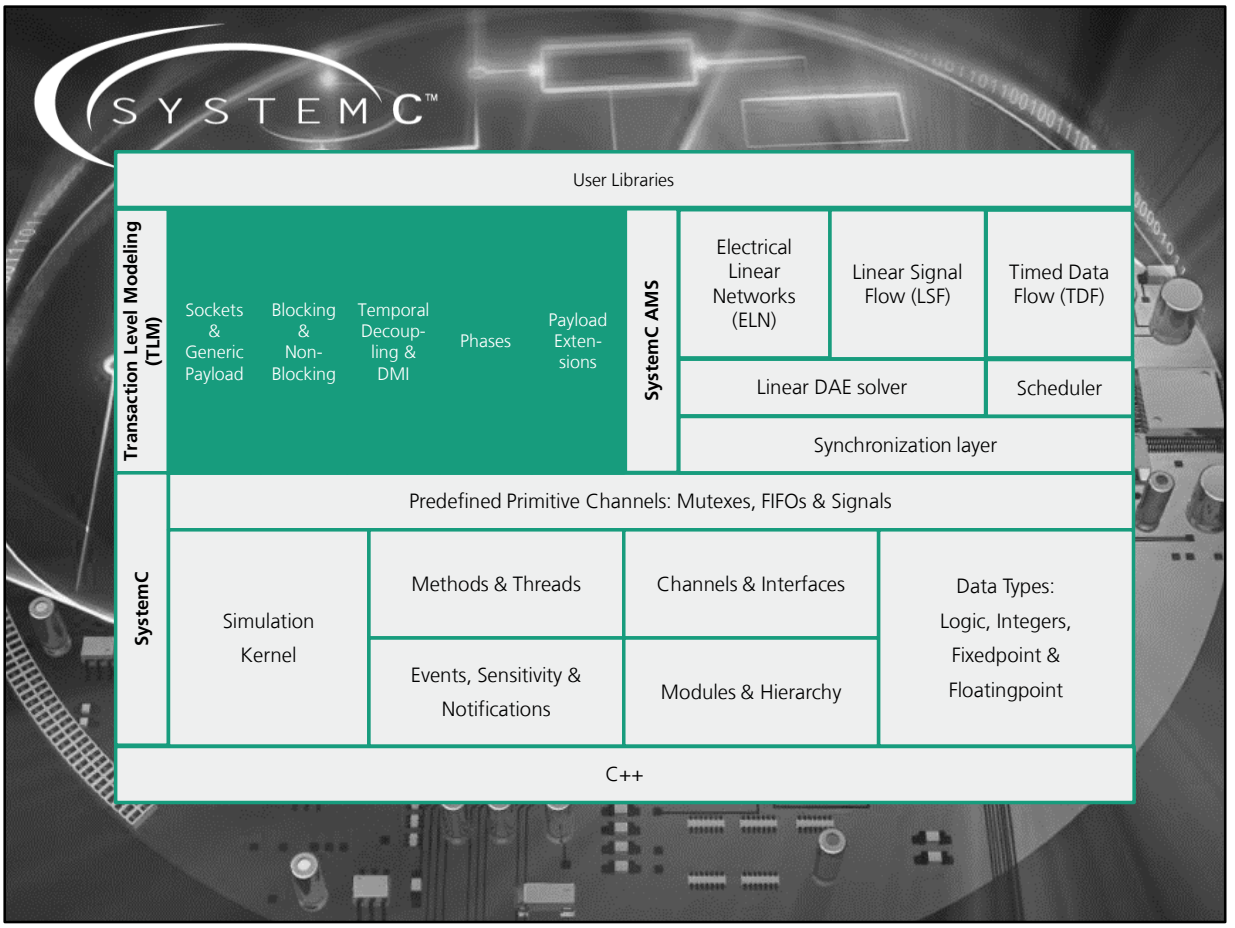
TLM have mainly 2 different views :-

1 ARCH view.

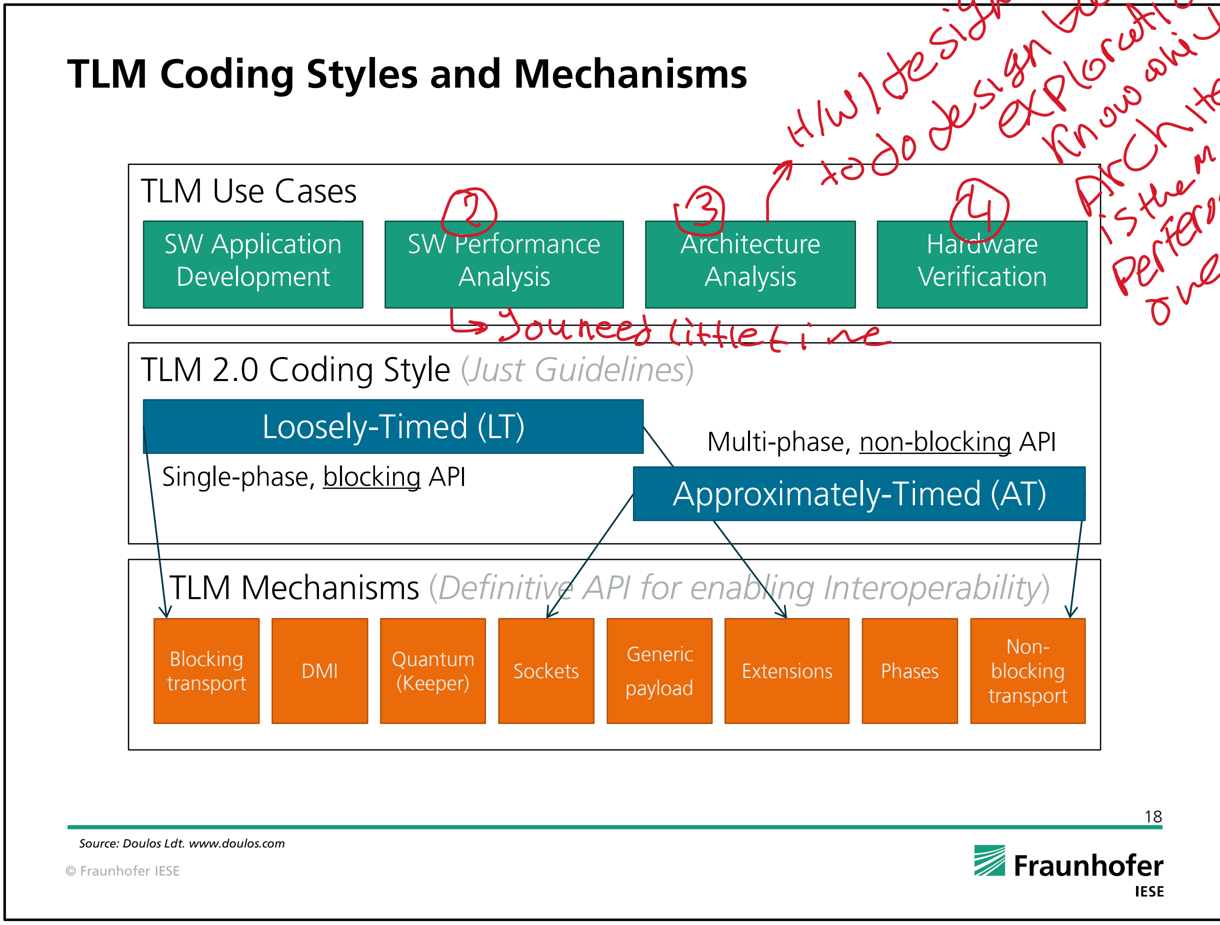
it's called APProximately Timed because it's very very Close to the RTL Timing but it's not accurate 100% but 99%

2 Programmers view

it's called Loosely timed, it has some timing information but it's not accurate as the APProximately timed but it's accurate to write driver.



Your notes:



Your notes:

- we have different TLM use cases:

① we want to do S/W Development on H/W that is not there already so we want to do this on Model

Check the rest of TLM use cases in up.

TLM have "coding style", it's always nice to stick to one :

① Loosely Timed

"single Phase, blocking API"

- it's alway Blocking, and we have just Function call here.

② APProximately Timed

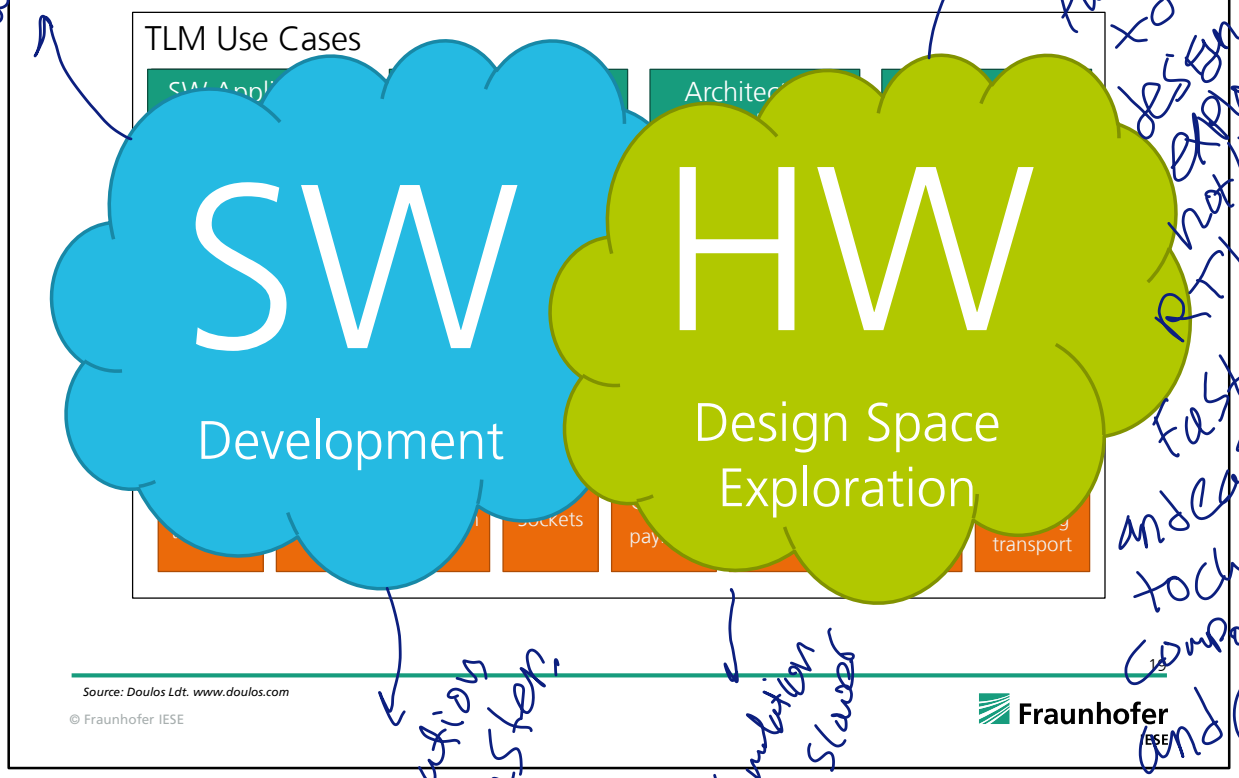
we have more than 1 Function call and more than event, and it's not blocking API.

- The best thing that you can mix between booth styles

- we have Different TLM mechanisms to support this different coding styles :

Check it in diagram LP.

TLM Coding Styles and Mechanisms



Your notes:

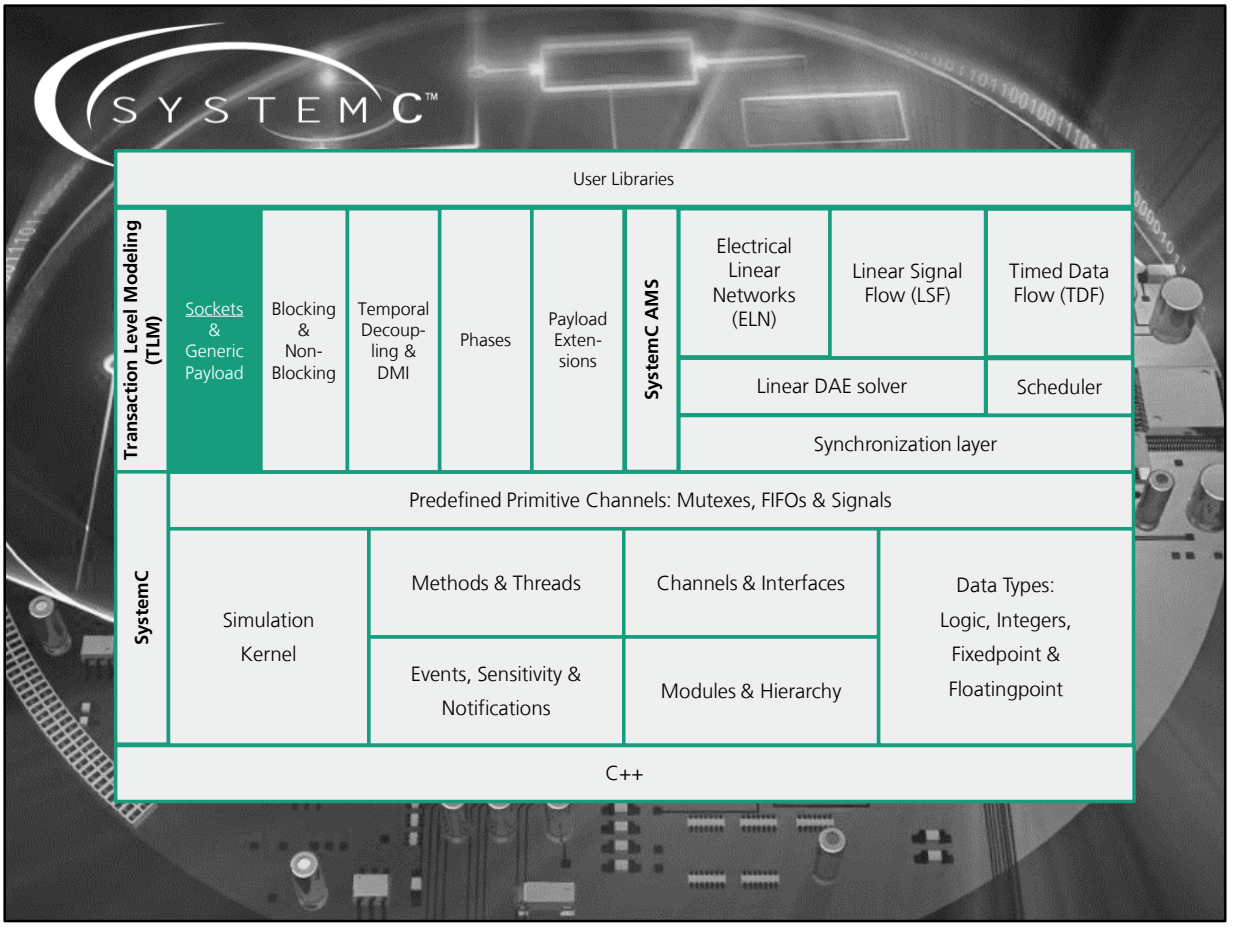
Coding Styles in TLM

- **Loosely-Timed (LT):**
  - As fast as possible
  - Sufficient timing detail to boot OS and run multicore systems and to develop SW or drivers
  - Processes can run ahead of simulation time (temporal decupling)
  - Each transaction completes in one blocking function call
  - Usage of Direct Memory Interface (DMI) e.g. for boot process

- **Approximately-Timed (AT):**
  - Accurate enough for performance modelling
  - Sufficient for architectural HW design space exploration
  - Processes run in lockstep with simulation time
  - Each transaction has usually 4 timing points i.e. 4 function calls (extensible if required, also less possible); non-blocking behavior
  - More detailed than LT and therefore also slower than LT

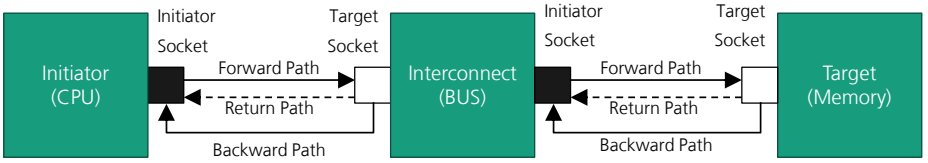
Fraunhofer IESE

Your notes:



Your notes:

Initiators, Targets and Interconnect



- TLM components are divided into Initiators, Targets and Interconnect components:
  - A initiator initiates (construct and send) new transactions
  - A target is a module that acts as the end point for a transaction. It executes requests from an initiator and send responses
  - An interconnect component forwards and routes transaction objects between initiators and targets
- Transactions are sent through initiator sockets (■) and received through target sockets (□)
- References to the object are passed along the forward and backward paths:
  - LT uses Forward and Return Path
  - AT uses Forward, Backward and Return Path

Fraunhofer IESE

Your notes: we have different Paths of the Information Flow.

ex:

when i talk about the ForwardPath here, The initiator calls the Fn of the Target socket and the returns Path if this fn returns.

- That means we can Transfer Information on the return Path, we Transfer forex The req is successful or not successful

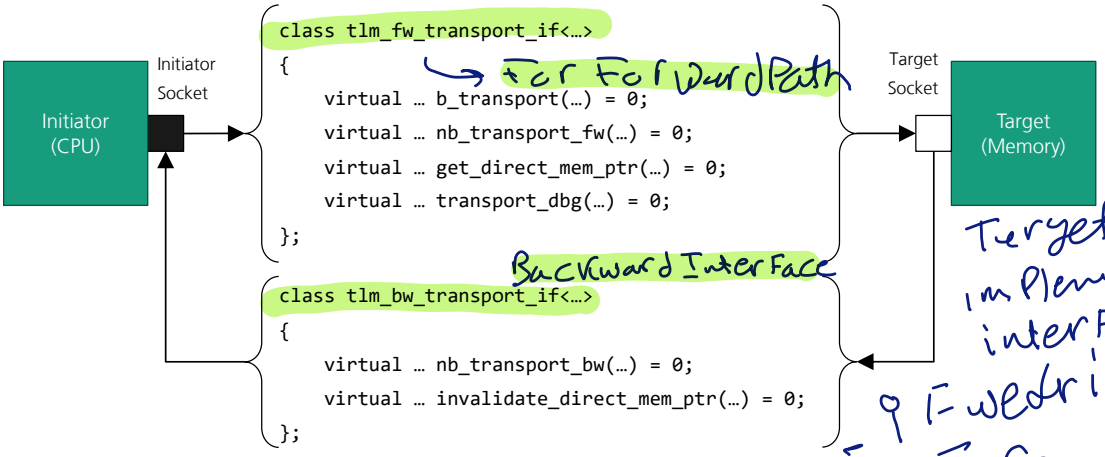
- we have Backward Path but we will discuss it in Approximately timed style because

* initiator : always initiate that means it always construct and send new Transaction.

* Target : it's a module that acts as an end point of Transaction and then it executes the request from the initiator and then it sends a response either by returning or backward Path.

SystemC provides Interfaces for TLM. The Interface defines which Functions for Forward Path, return path, Backward Path

Initiator and Target Sockets



for Forward Path

Backward Interface

TargetSocket implements this interface

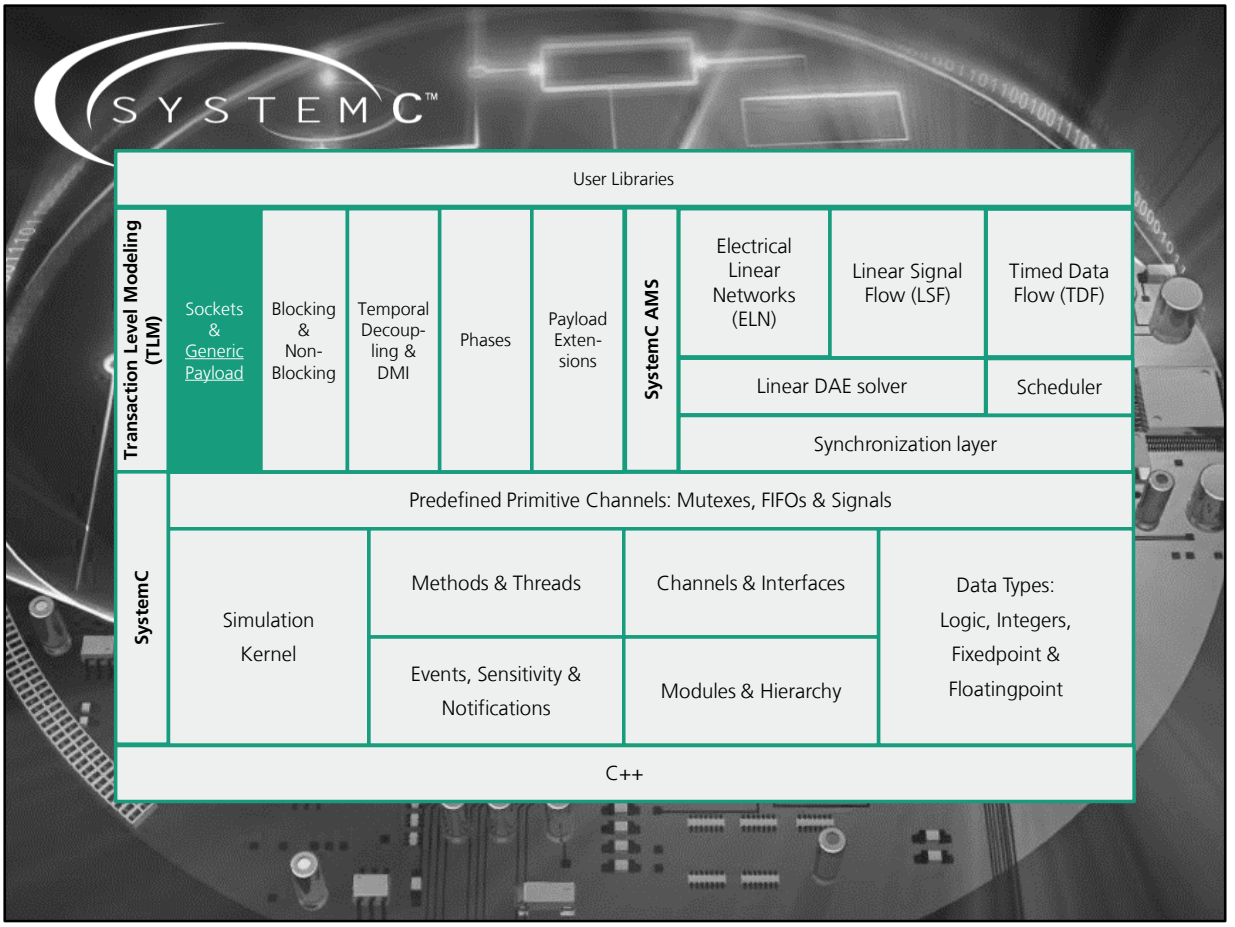
- 9 Fwd/Bw

From this Interfaces we should define all the fns of the Interface without Implementation

- Target port must implement `tlm_fw_transport_if` methods
- Initiator must implement `tlm_bw_transport_if` methods
- `b_transport` and `mem_ptr` functions are used for LT modeling
- `nb_transport` functions are used for AT modeling

Fraunhofer IESE

Your notes:

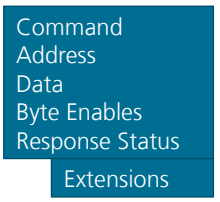


Your notes:

Generic Payload

- The generic payload is designed to include typical attributes of memory mapped busses (e.g. AXI, AHB, etc.)
- It can supports
  - READ and WRITE transactions
  - Byte enables
  - Single word transfer
  - Burst transfer
  - Streaming transfer
- Extensions can be used to carry further metadata or to model more complex bus and NoC protocols and maintain 100% compatibility because they are ignorable
- Very efficient implementation for simulation speed

Generic payload object



Fraunhofer IESE

Your notes:

Generic Payload Attributes

Attribute	Type	Modifiable?
Command	`tlm_command`	No
Address	`uint64_t`	Interconnect Only
Data Pointer	`unsinged char *`	No (array yes)
Data Length	`unsigned int`	No
Byte Enable Pointer	`unsigned char *`	No
Byte Enable Length	`unsinged int`	No
Streaming Width	`unsigned int`	No
DMI Hint	`bool`	Yes
Response Status	`tlm_response_status`	Target Only
Extensions	`(tlm_extension_base*)[]`	Yes

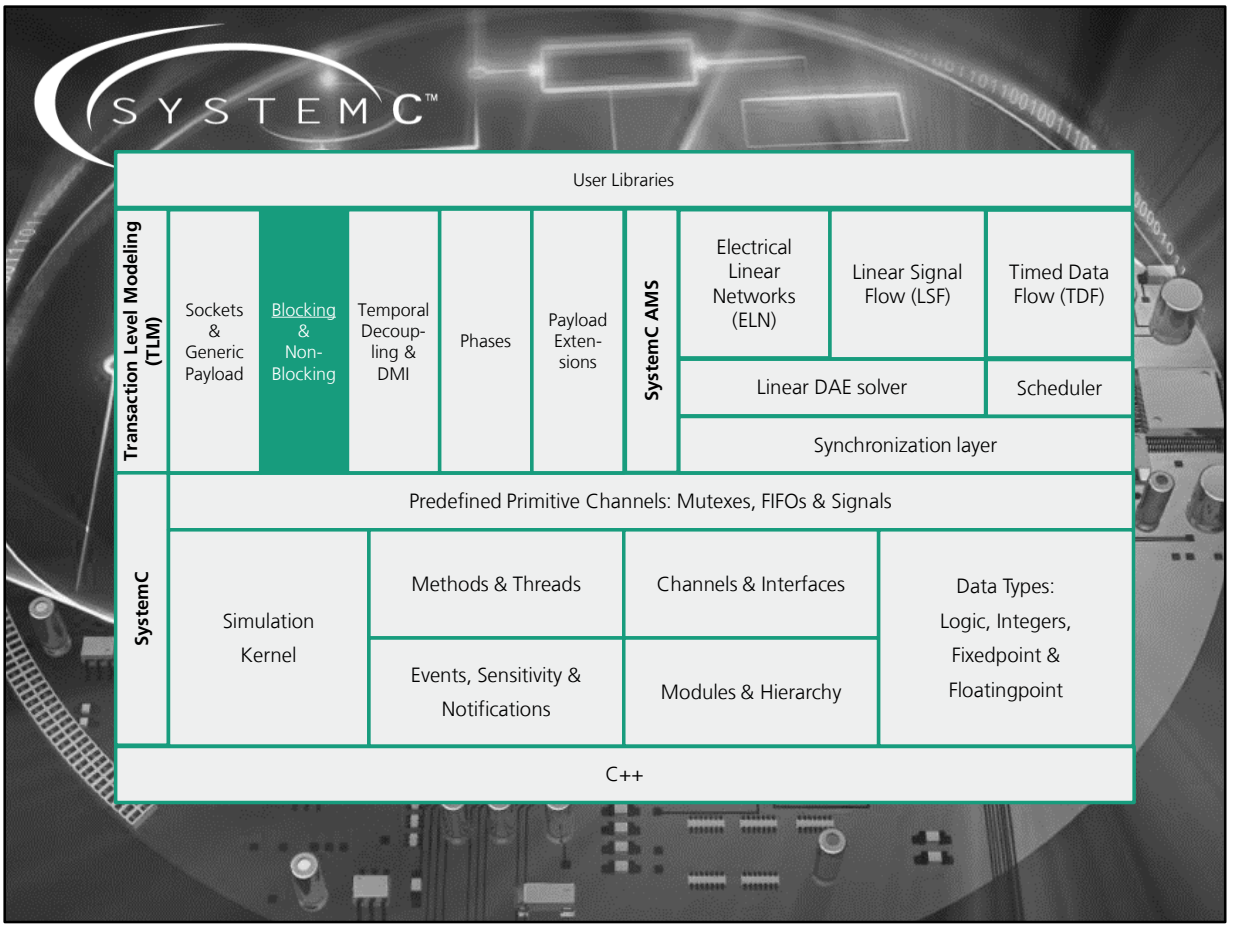
- Initiator should initialize attributes before sending the transaction object
- Set-Methods like `set_address()` etc. are provided
- The majority of the attributes must not be changed by Interconnects & Targets

Fraunhofer IESE

Your notes: in TLM they decided to make all communication byte based.

- They consider dat in Array (character) so we define the Data length

-



Your notes:

Note
*This is always the Pointer that Points to the object of this class

itself So, it's own Pointer.

loosely timed

Building a Blocking (LT) Initiator

SW

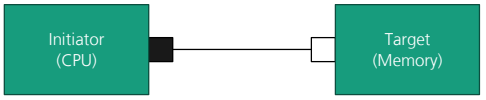
```
class Initiator: sc_module, tlm::tlm_bw_transport_if<> {
    public:
    tlm::tlm_initiator_socket<> iSocket;
    SC_CTOR(Initiator) : iSocket("iSocket") {
        iSocket.bind(*this);
        SC_THREAD(process);
    }

    void process() {
        for (int i = 0; i < 4; i++) {
            tlm::tlm_generic_payload trans;
            unsigned char data = rand();
            trans.set_address(i);
            trans.set_data_length(1);
            trans.set_command(tlm::TLM_WRITE_COMMAND);
            trans.set_data_ptr(&data);
            sc_time delay = sc_time(0, SC_NS);
            iSocket->b_transport(trans, delay);
            wait(delay);
        }
        for (int i = 0; i < 4; i++) {
            tlm::tlm_generic_payload trans;
            unsigned char data;
            trans.set_address(i);
            trans.set_data_length(1);
            trans.set_command(tlm::TLM_READ_COMMAND);
            trans.set_data_ptr(&data);
            sc_time delay = sc_time(0, SC_NS);
            iSocket->b_transport(trans, delay);
            wait(delay);
        }
    }
```

Write to memory

Read from memory

```
    // Dummy method:
    void invalidate_direct_mem_ptr(
        sc_dt::uint64 start_range,
        sc_dt::uint64 end_range)
    {}

    // Dummy method:
    tlm::tlm_sync_enum nb_transport_bw(
            tlm::tlm_generic_payload& trans,
            tlm::tlm_phase& phase,
            sc_time& delay)
    {
        return tlm::TLM_ACCEPTED;
    }
};
```

Must be implemented



Try code on github:
https://github.com/TUK-SCVP/SCVP.artifacts/tree/master/tlm_lt_initiator_target

Fraunhofer IESE

Your notes:

when we use(this) it gets the connection between methods that we have to implement here and later on the connection of this socket

Building a Blocking (LT) Target


```
class Target:sc_module, tlm::tlm_fw_transport_if<> {
    private:
    unsigned char mem[1024];

    public:
    tlm::tlm_target_socket<> tSocket;

    SC_CTOR(Target) : tSocket("tSocket") {
        tSocket.bind(*this);
    }

    void b_transport(tlm::tlm_generic_payload &trans,
                     sc_time &delay)
    {
       if(trans.get_address() >= 1024){
         SC_REPORT_FATAL(this->name(),"Out of Range");
       }

       if(trans.get_command() == tlm::TLM_WRITE_COMMAND)
       {
         memcpy(mem+trans.get_address(),  // destination
                trans.get_data_ptr(),     // source
                trans.get_data_length()); // size
       } else {
         memcpy(trans.get_data_ptr(),     // destination
                mem+trans.get_address(),  // source
                trans.get_data_length()); // size
       }
       delay = delay + sc_time(40, SC_NS);
    }
    ...

    // Dummy method
    virtual tlm::tlm_sync_enum nb_transport_fw(
            tlm::tlm_generic_payload& trans,
            tlm::tlm_phase& phase,
            sc_time& delay )
    {
       return tlm::TLM_ACCEPTED;
    }

    // Dummy method
    bool get_direct_mem_ptr(
         tlm::tlm_generic_payload& trans,
         tlm::tlm_dmi& dmi_data)
    {
        return false;
    }

    // Dummy method
    unsigned int transport_dbg(
        tlm::tlm_generic_payload& trans)
    {
        return 0;
    }
};
```

Must be implemented

Fraunhofer IESE

Your notes:

Binding Target and Initiator

SW

```
int sc_main (…)
{
    Initiator * cpu = new Initiator("cpu");
    Target * memory = new Target("memory");

    cpu->iSocket.bind(memory->tSocket);

    sc_start();
    return 0;
}
```

Try code on github:
https://github.com/TUK-SCVP/SCVP.artifacts/tree/master/tlm_lt_initiator_target

Summary:

- Initiator and target ports are derived from `sc_port` and `sc_export`
- `b_transport` uses call by reference to transfer the transaction object (from `tlm_generic_payload` class)
- The key idea of timing annotation is that the recipient is obliged to behave as if it had received the transaction at time `sc_time_stamp() + delay`
- All virtual methods must be implemented
- Transaction objects should be reused to avoid always new allocations

Fraunhofer IESE

Your notes:

Error Handling for b_transport

enum tlm_response_status	Meaning
TLM_OK_RESPONSE	Successful transmission
TLM_INCOMPLETE_RESPONSE	Transaction not delivered to the target (default)
TLM_ADDRESS_ERROR_RESPONSE	Unable to work with given address
TLM_COMMAND_ERROR_RESPONSE	Unable to execute command (e.g write to ROM)
TLM_BURST_ERROR_RESPONSE	Unable to work with given datalength
TLM_BYTE_ENABLE_ERROR_RESPONSE	Unabel to work with byte enable
TLM_GENERIC_ERROR_RESPONSE	Any other error

- Initiator should set response status to `TLM_INCLOMPLETE_RESPONSE` (default)
- Targets modify the response status
- Initiator checks status of transaction when it is completed (e.g. after `b_transport`)

Fraunhofer IESE

Your notes:

Error Handling for b_transport

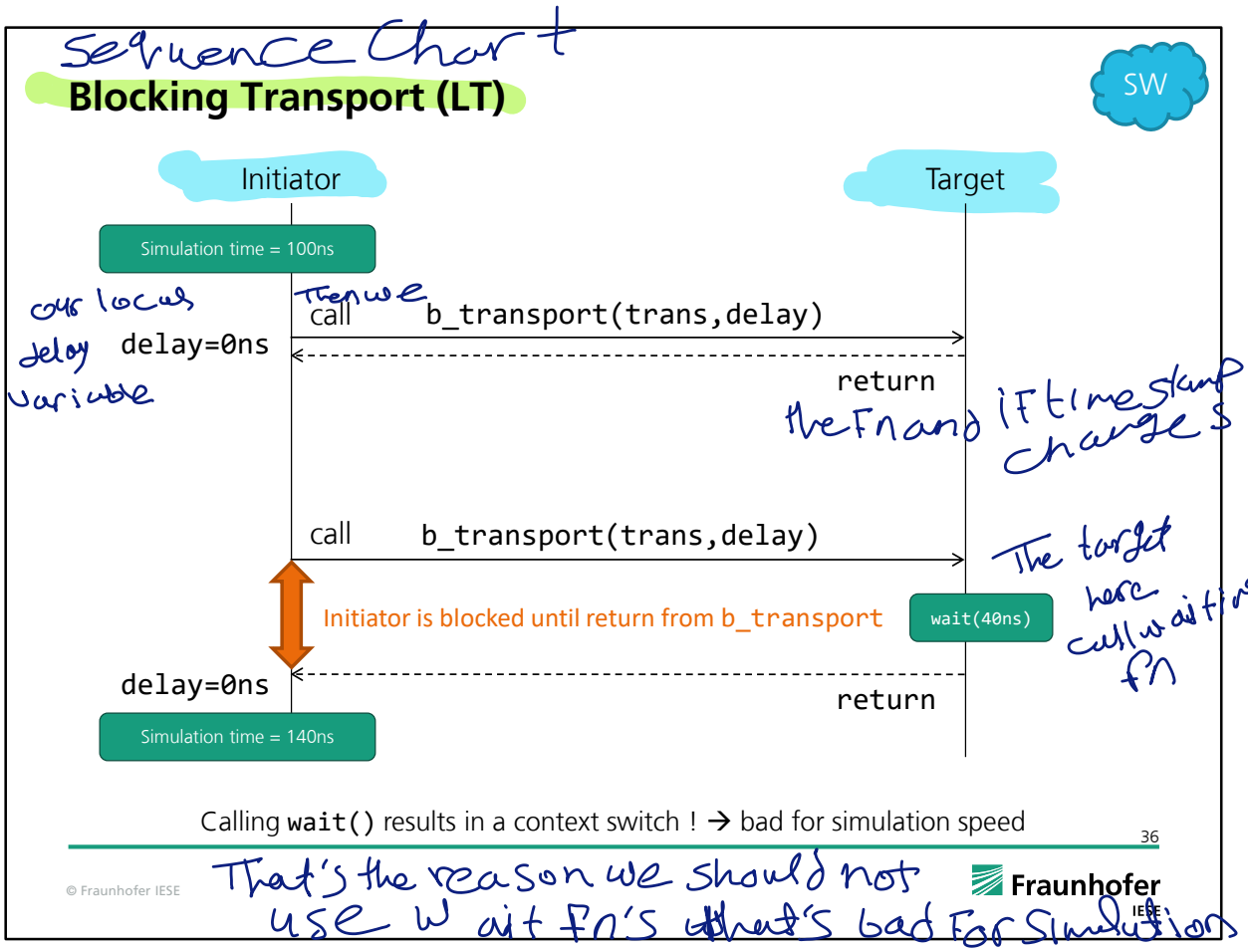

```
class exampleInitiator: sc_module,
tlm::tlm_bw_transport_if<>
{
   ...
   private:
   void process()
   {
      ...
      iSocket->b_transport(trans, delay);
      if(trans.is_response_error())
      {
         SC_REPORT_FATAL(name(), ”Error”)
      }
   }
   ...
};

class exampleTarget : sc_module,
tlm::tlm_fw_transport_if<>
{
   ...
   unsigned char mem[512];

   public:
   tlm::tlm_target_socket<> tSocket;

   SC_CTOR(exampleTarget) : tSocket("tSocket")
   {
      tSocket.bind(*this);
   }
```

Detailed check can be done with `trans.get_response_status()`

```
   void b_transport(... &trans, ... &delay)
   {
      if (trans.get_address() >= 512) {
          trans.set_response_status(
              tlm::TLM_ADDRESS_ERROR_RESPONSE );
          return;
      }
      if (trans.get_data_length() != 4) {
          trans.set_response_status(
              tlm::TLM_BURST_ERROR_RESPONSE );
           return;
      }
      if (byt) {
          trans.set_response_status(
              tlm::TLM_BYTE_ENABLE_ERROR_RESPONSE );
          return;
      }

      if(trans.get_command() == tlm::TLM_WRITE_COMMAND){
         memcpy(...);
      }
      else {
         memcpy(...);
      }

      delay = delay + sc_time(40, SC_NS);

      trans.set_response_status( tlm::TLM_OK_RESPONSE );
   }
};
```


Fraunhofer IESE

Your notes:



An initiator, and only an initiator, may call `b_transport`. The target can return immediately, or can suspend (by calling `wait`) and return later. However, calling `wait` statements in targets should be avoided because a wait statement will result in a context switch of the simulator, which decreases the simulation speed.

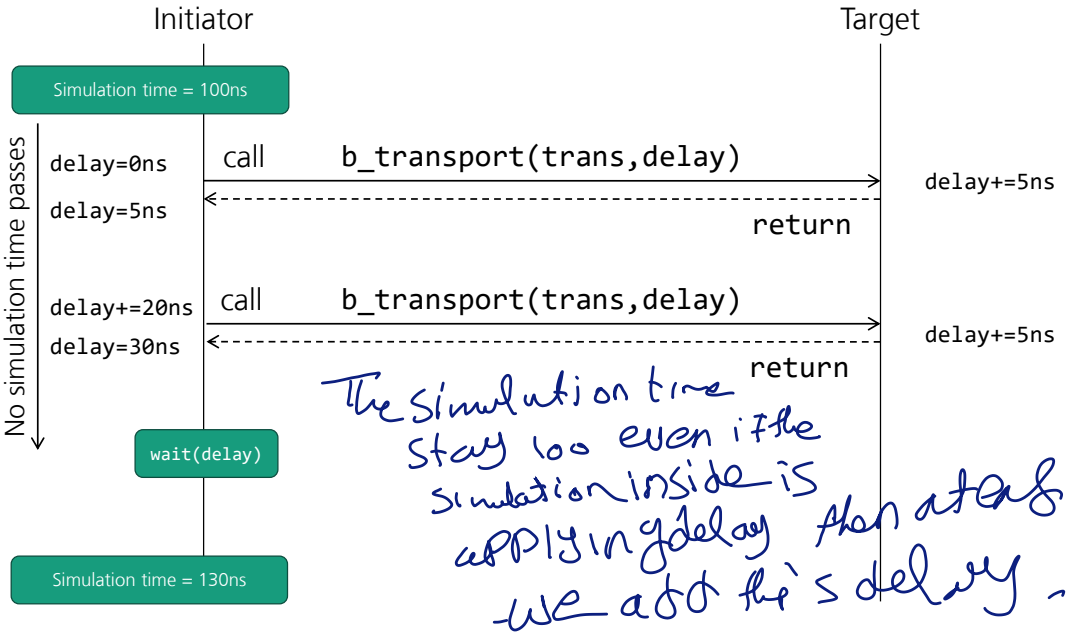
The same transaction object could be reused from one call to the next, but the two calls would count as separate transactions.

Your notes:

& We should do something differenas shown below's

Blocking Transport (LT)

SW



The simulation time stay too even if the simulation inside is applying delay than atas we add this delay.

Initiator should use a local time variable and should call `wait()`! → Less context switches

Fraunhofer IESE

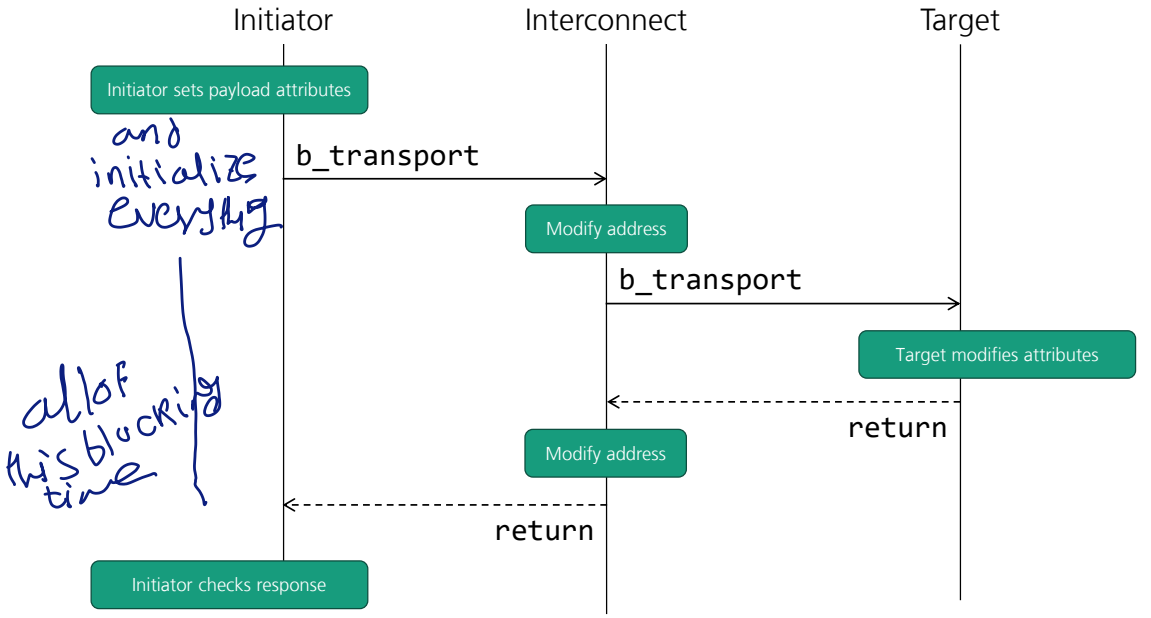
Your notes:

We can chain the b_Transport calls.

Chaining b_transport Calls

That's the

SW



Fraunhofer IESE

Your notes:

Let's Build Interconnect Component

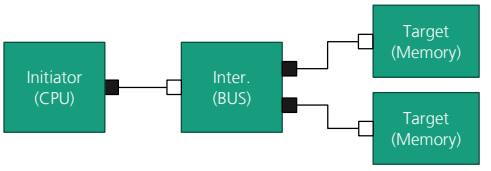
Building an Interconnect Component

```
class Interconnect : sc_module,
                     tlm::tlm_bw_transport_if<>,
                     tlm::tlm_fw_transport_if<>
{
    public:
    tlm::tlm_initiator_socket<> iSocket[2];
    tlm::tlm_target_socket<> tSocket;

    SC_CTOR(exampleInterconnect)
    {
      tSocket.bind(*this);
      iSocket[0].bind(*this);
      iSocket[1].bind(*this);
    }

    void b_transport(
        tlm::tlm_generic_payload &trans,
        sc_time &delay)
    {
      delay = delay + sc_time(40, SC_NS);

      if(trans.get_address() < 512) {
        iSocket[0]->b_transport(trans, delay);
      }
      else {
        trans.set_address(trans.get_address() - 512);
        iSocket[1]->b_transport(trans, delay);
      }
    }
    … // Dummy Methods
};
```

Annotate Bus Time

Modify address

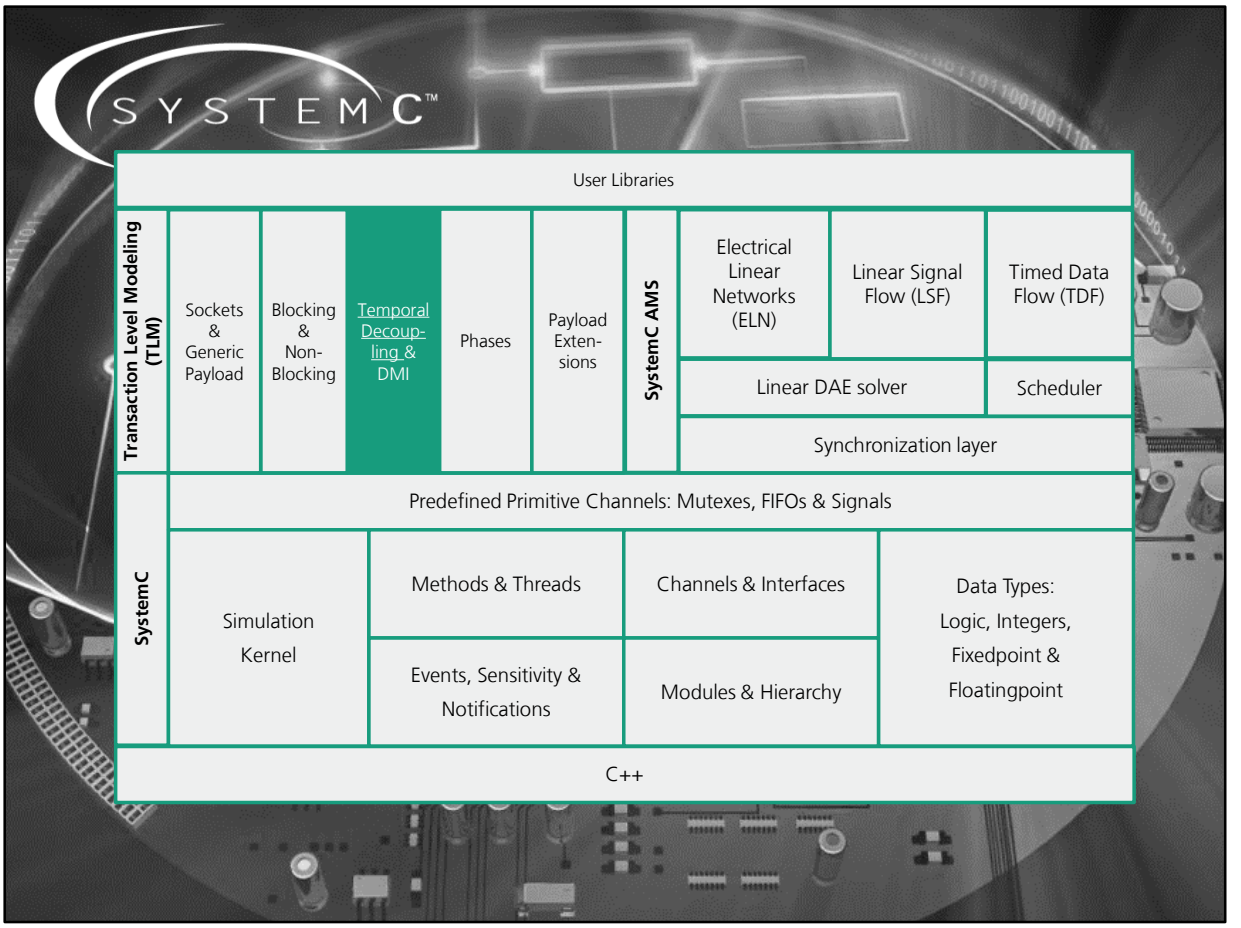
```
int sc_main (int __attribute__((unused)) sc_argc,
             char __attribute__((unused))
*sc_argv[])
{

    Initiator * cpu     = new Initiator("cpu");
    Target * memory1    = new Target("memory1");
    Target * memory2    = new Target("memory2");
    Interconnect * bus = new Interconnect("bus");

    cpu->iSocket.bind(bus->tSocket);
    bus->iSocket[0].bind(memory1->tSocket);
    bus->iSocket[1].bind(memory2->tSocket);

    sc_start();

    return 0;
}
```



Try code on github:
https://github.com/TUK-SCVP/SCVP.artifacts/tree/master/tlm_lt_initiator_interconnect_target

Fraunhofer IESE

Your notes: Before we just had 1 CPU with one Target, now we want to have a bus Model in between.

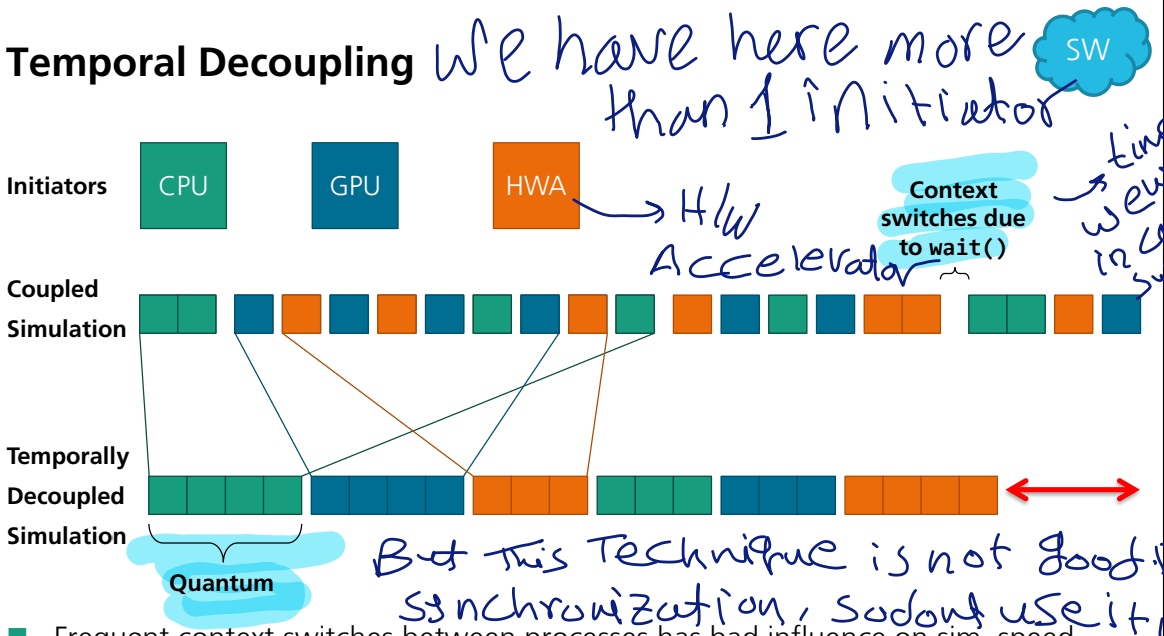


Your notes:

let's talk about how we can make simulation faster.

Temporal Decoupling

we have here more than 1 initiator



But this technique is not good for synchronization, so don't use it if your components communicate a lot together.

- Frequent context switches between processes has bad influence on sim.-speed
- In temporal decoupled simulation mode a process keeps control until a quantum is reached, then its switched to another process
- Processes can run ahead of time! Out-of-order execution! Synchronization!

ideal example you have 3 CPU's and every core is calculating something but they don't communicate with each other.

Fraunhofer IESE

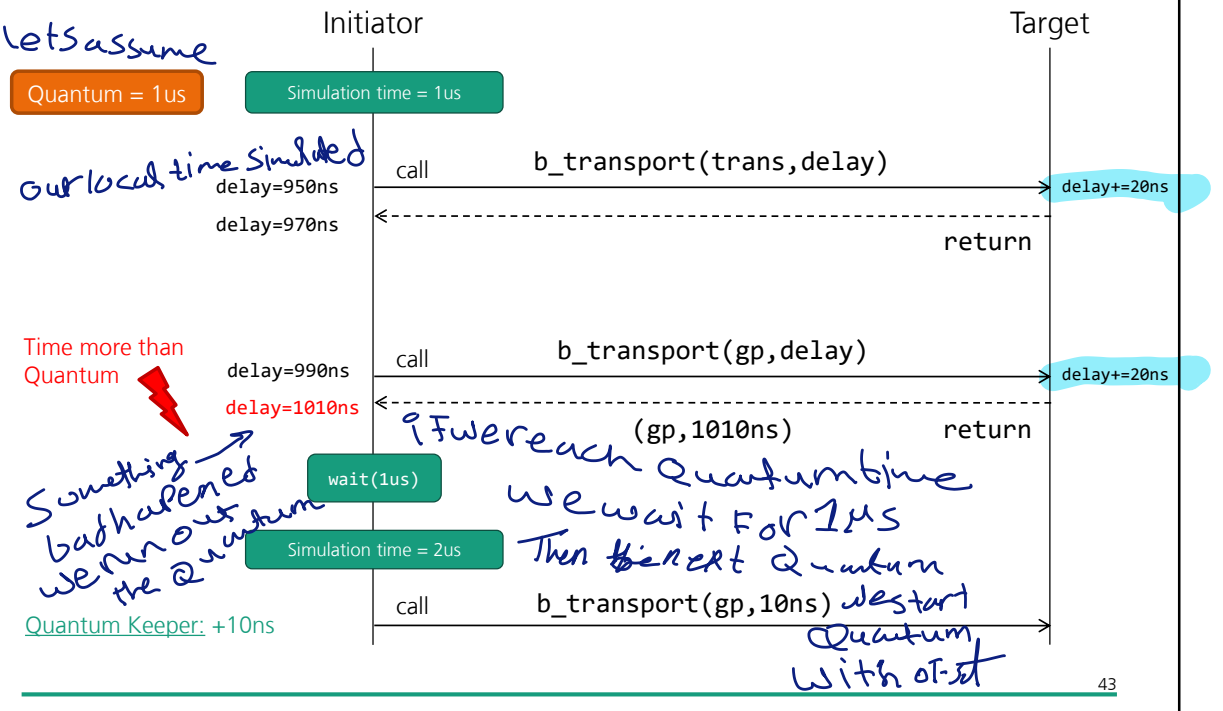
Your notes:

we have now CPU, GPU, H/W A, one initiator is doing memory Access then it did another memory ACCESS, then it did await statement. Then another initiator wants to do something so we go first to the Kernel then we switch to the other initiator, but every time we switch we have an overhead.

- The idea of avoiding this context switching is the Temporally Decoupled Simulation, where we give every initiator to run all transactions during this simulation time and this

time is called Quantum, once Quantum reach we switch to other initiator.

Blocking Interface (LT) with Quantum



Your notes:

So System-C will Perform this B_Transport and the next Quantum will start with an offset, that's how System-C is Solving this Problem when You run out of your Quantum time.

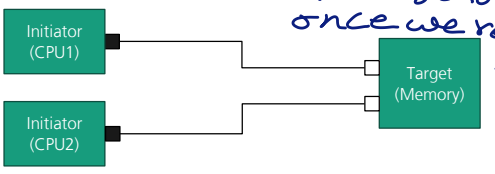
For that reason There exist this Quantum Keeper that stores the remander of time of previous Quantum.

Quantum Keeper

```
class Initiator: sc_module,
                 tlm::tlm_bw_transport_if<>
{
  private:
  tlm_utils::tlm_quantumkeeper quantumKeeper;

  public:
  tlm::tlm_initiator_socket<> iSocket;
  SC_CTOR(exampleInitiator) : iSocket("iSocket")
  {
    iSocket.bind(*this);
    SC_THREAD(process);
    quantumKeeper.set_global_quantum(
      sc_time(1,SC_US)
    );
    quantumKeeper.reset();
  }

  void process()
  {
    // Write to memory:
    for (int i = 0; i < 1024; i++) {
      tlm::tlm_generic_payload trans;
      unsigned char data = rand();
      trans.set_address(i);
      trans.set_data_length(1);
      trans.set_command(tlm::TLM_WRITE_COMMAND);
      trans.set_data_ptr(&data);
      sc_time delay = quantumKeeper.get_local_time();
      iSocket->b_transport(trans, delay);
      // Anotate the time of the target
      quantumKeeper.set(delay);
      // Consume internal computation time
      quantumKeeper.inc(sc_time(10,SC_NS));
      if(quantumKeeper.need_sync())
      {
        quantumKeeper.sync();
      }
    }
  }
  …
  // Dummy methods …
};
```

Static Method

Calls `wait()` internally



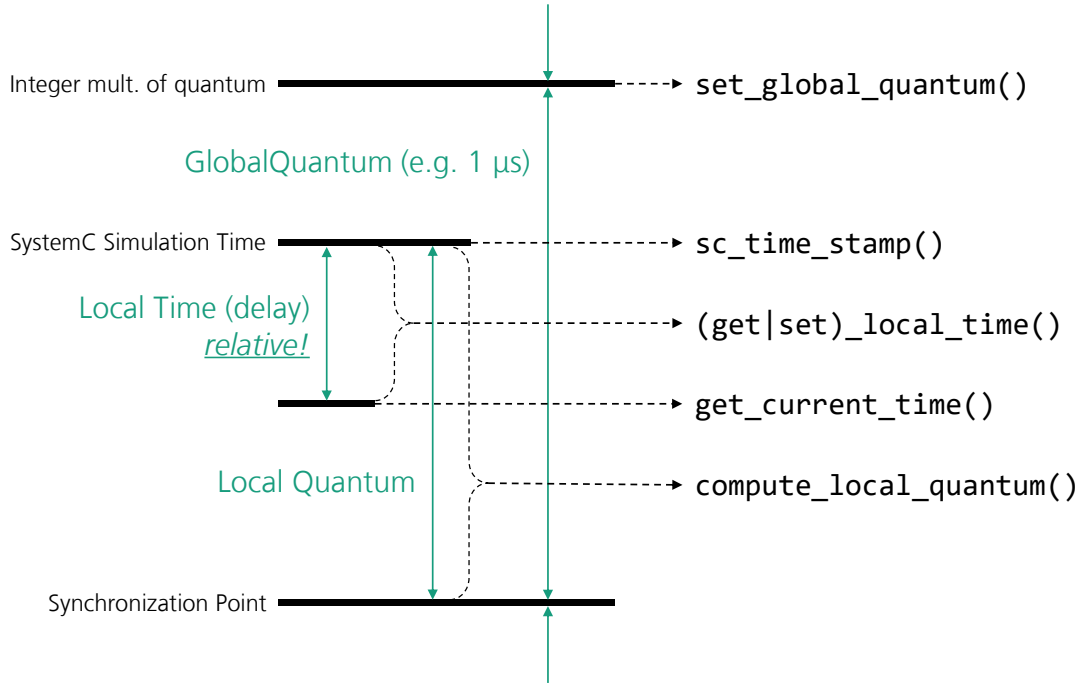
Try code on github:
https://github.com/TUK-SCVP/SCVP.artifacts/tree/master/tlm_quantum_keeper

Fraunhofer IESE

Your notes:

Summarization of Steps

Quantum Keeper Variables and Methods



Fraunhofer IESE

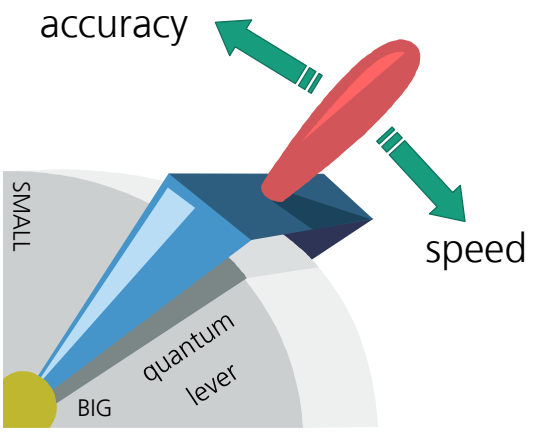
- There are three absolute time variables, namely the current simulation time as returned by sc_time_stamp(), the current time decoupled from simulation time, and the time of the next synchronization point.
- The global quantum is the time between two sync points.
- The local time is the time offset of the current time (as used by a temporally decoupled initiator) from the SystemC simulation time.
- The current time is the absolute time value as used by a temporally decoupled initiator.

Your notes:

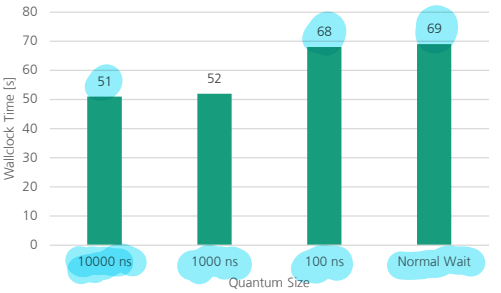
Quantum vs. Accuracy

SW

- Quantum is user configurable
- Trade-off between simulation speed and accuracy
- The smaller the quantum, the more accurate the simulation
- If target uses `wait()` internally, it should set the `delay = SC_ZERO_TIME`



Our Artifacts Example (i.7, MacOS):



Fraunhofer IESE

as we increase Quantum size we get high speed but until we reach a certain Point the speed of Simulation get Saturation Point - no more Benefits From increasing Quantum.

Your notes:

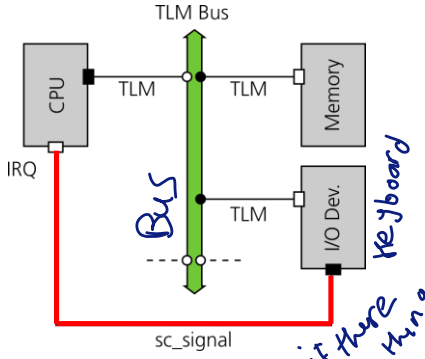
in general

The smaller the Quantum
The more Accurate the simulation

The greater the Quantum
The less accurate the simulation

usually model it looks

A Closer Look on Functional Simulation Errors

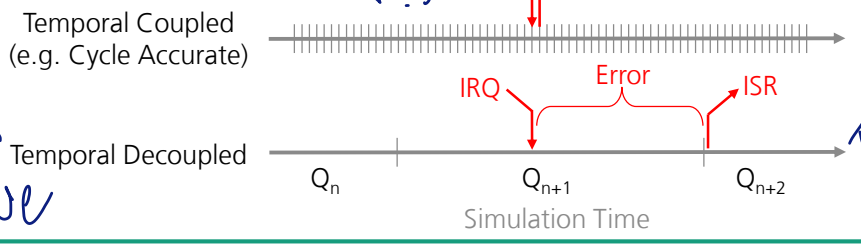


Temporal Coupled (e.g. Cycle Accurate)

- I/O Device makes an *Interrupt Request* (IRQ)
- The *Interrupt Service Routine* (ISR) will be called a few cycles later

Temporal Decoupled Simulation

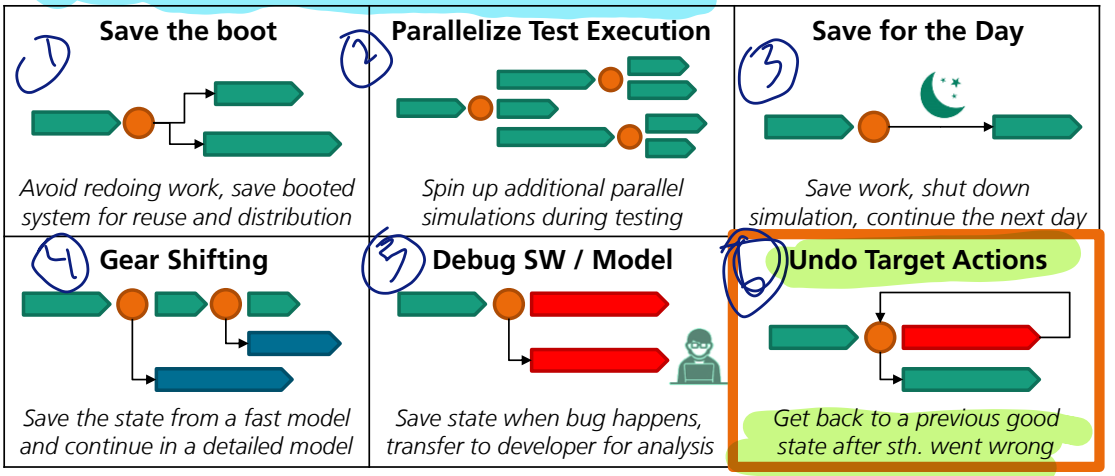
- I/O Device makes an IRQ
- The ISR will be called in the next Quantum



Fraunhofer IESE

Checkpointing

- The ability to save the state of a simulation and later pick up at the exact same point in time.



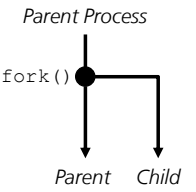
- Gläser et al. [4] presented in 2015 the idea of using checkpointing in order to rollback in simulation time and force an earlier synchronization to correct the occurred errors.

Source: Jacob Engblom, Intel, SystemC Evolution Day, 2017

[4] Temporal decoupling with error-bounded predictive quantum control. Georg Glaeser, Gregor Nitsche, Eckhard Hennig. Published in Forum on Specification and Design Languages (FDL) 2015

Fraunhofer IESE

The Good Old `fork()`



- `fork()` is a system call that allows a process in the OS to create a one-to-one copy of itself, called *child*.
- Supported by all OS: Linux, FreeBSD, macOS, …
- Modern OS do not duplicate the complete memory space of a process
- Instead they use the Copy-on-Write semantic:
  - The copy operation is deferred to the first write to a memory page
  - In other words: a memory page is only copied in the moment of the change

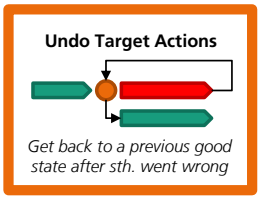
Can fork() be used as an efficient way for checkpointing in order to get an error free temporal decoupled simulation?

Fraunhofer IESE

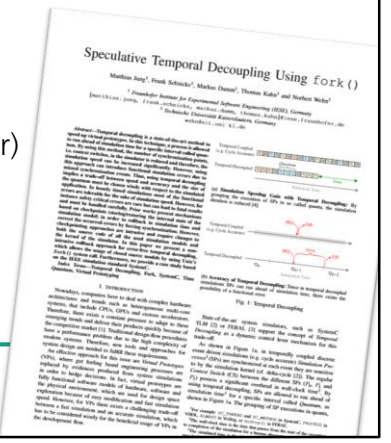
it's possible but it depends.

Speculative Temporal Decoupling using `fork()`

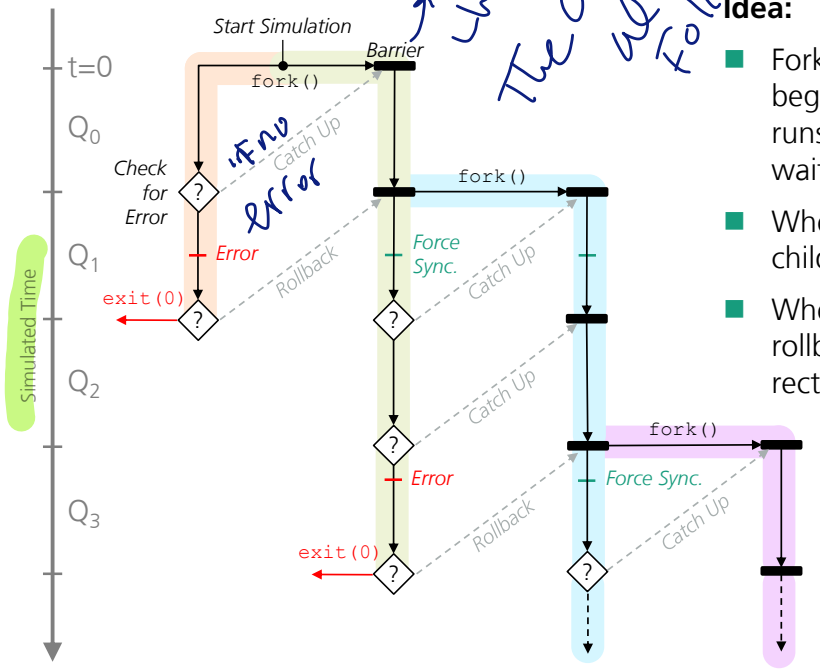
- Idea:
  - Use `fork()` to backup the simulation state
  - Execute the next quantum **speculatively**
  - In case of an error rollback in simulation time
  - Correct the timing error e.g. by temporary decreasing the quantum size



- Two approaches investigated:
  1. **Naïve Approach** (Forking at each quantum)
  2. **Lockstep Approach** (Forking only in case of an error)
- Synchronization of *parent* and *child* is done with `pipe` (acts like a barrier)
- Details of the implementation are in the paper

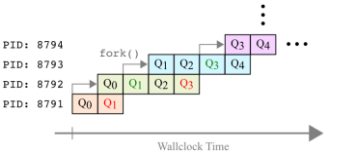

Speculative Temporal Decoupling Using fork()

Approach



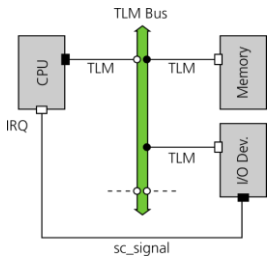
Idea:

- Fork the simulation in the beginning. The parent runs while the child is waiting
- When no error occurs the child can catch up
- When an error happens rollback to the child and rectify the error

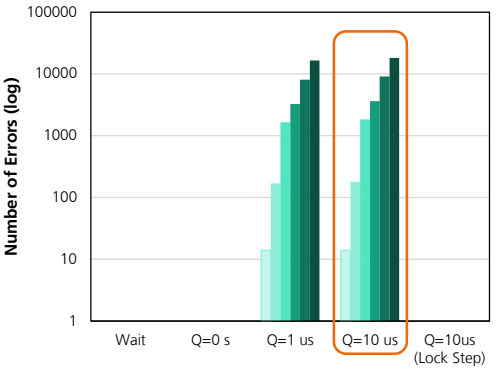


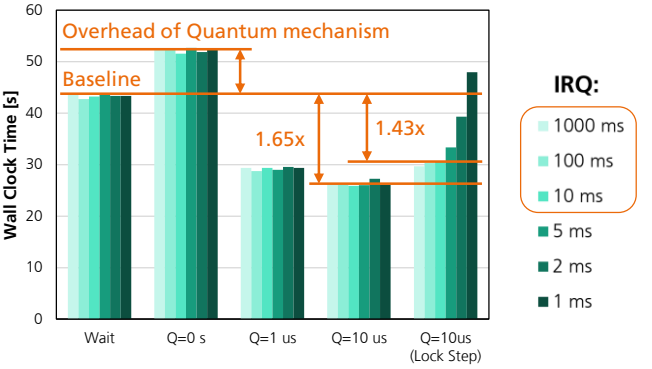
Fraunhofer IESE

Results for the Approach

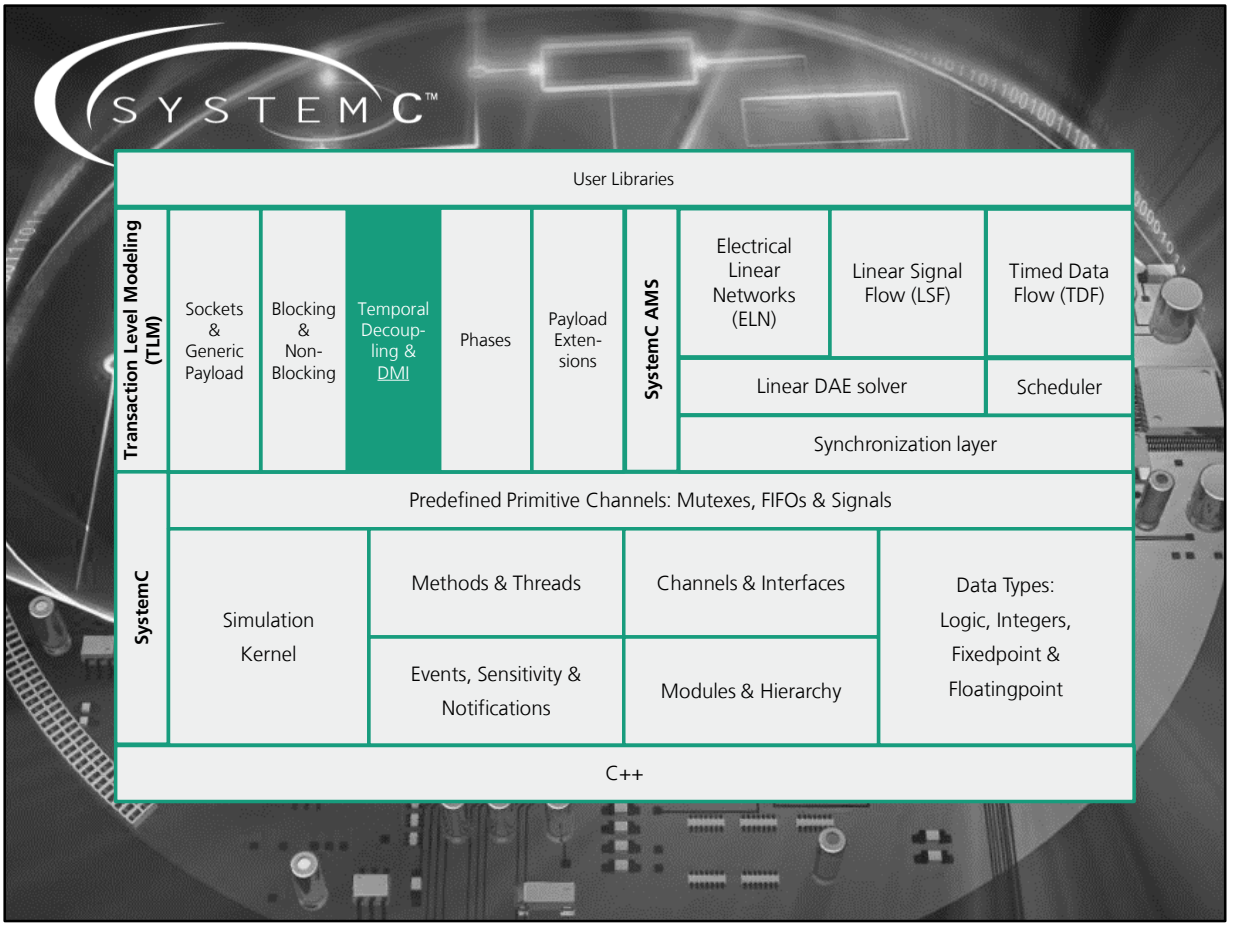


- Example System with one CPU and I/O Device
- Interrupt rates between 1s - 1ms
- Synchronization with `wait()`
- Synchronization with different quanta (0s, 1us, 10us)
- Errors = Number of missed IRQ events





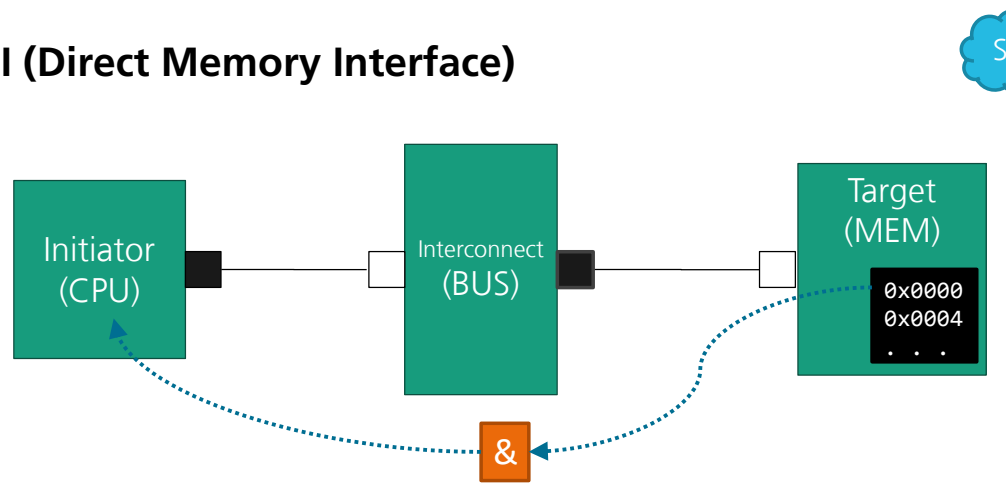
Fraunhofer IESE



Your notes:

There are another Technique For making Simulation Fast

DMI (Direct Memory Interface)



- Bypasses the Interconnects (e.g. Bus or Cache) and all socket & transport calls!
- Gives an initiator a pointer to memory region in the target
- Target can give a hint to initiator that DMI is available → it tells initiator you can connect with me through DMI.
- Uses also generic payload, can use also extensions
- Target can invalidate DMI regions
- Gives higher simulation speed, e.g. for booting an OS …

By that you reach high simulation speed

Fraunhofer IESE

The main reason for DMI is simulation speed. DMI allows the transport interface to be bypassed by giving an initiator a direct pointer to an area of memory in a target. Instead of calling transport for each read and write operation, the initiator can simply access the memory directly. For operations other than read or write, it is possible to add extensions to the DMI transaction using the standard generic payload extension mechanism. It is the responsibility of the initiator to honor any invalidation calls received from the target

Your notes: The idea is as the following when you have a CPU and you have Target memory and you have some data in memory and you have the interConnect lets assume that this a very detailed interConnect model and it's very slow because you may have an RTL inside and now you want to boot linux, it would take a week to boot Linux because

these interconnect is so damn slow.

What we can do for this as follows

The idea is with DMA we bypass this interconnect.

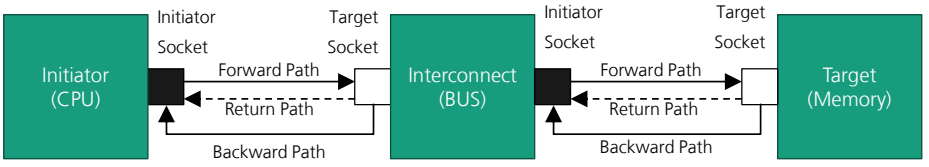
-

You remember we had some functions in our interface

DMI (Direct Memory Interface)

SW

```
bool get_direct_mem_ptr(tlm_generic_payload trans, tlm_dmi dmiData);
```




```
void invalidate_direct_mem_ptr(uint64 start, uint64 end);
```

- Same routing as e.g. `b_transport`
- Class `tlm_dmi`:
  - `unsigned char* dmi_ptr`
  - `uint64 start_address`
  - `uint64 end_address`
  - `dmi_access_e granted_access`
  - `sc_time read_latency`
  - `sc_time write_latency`

Fraunhofer IESE

Your notes:

DMI Initiator

SW

```
class Initiator: sc_module, tlm::tlm_bw_transport_if<> {
  bool dmi // set to false in the constructor;
  tlm::tlm_dmi dmiData;
  ...
  void process() {
    for (int i = 0; i < 16; i++)
    {
      tlm::tlm_generic_payload trans;
      unsigned char data = rand();
      trans.set_address(i);
      trans.set_data_length(1);
      trans.set_command(tlm::TLM_WRITE_COMMAND);
      trans.set_data_ptr(&data);
      sc_time delay = sc_time(0, SC_NS);

      if ( dmi == true
           && i >= dmiData.get_start_address()
           && i <= dmiData.get_end_address())
      {
        if( trans.get_command() == tlm::TLM_READ_COMMAND
            && dmiData.is_read_allowed())
        {
          memcpy(&data,
          dmiData.get_dmi_ptr() + i
              - dmiData.get_start_address(),
          trans.get_data_length());
          delay += dmiData.get_read_latency();
        }
        else if( trans.get_command()==tlm::TLM_WRITE_COMMAND
          && dmiData.is_write_allowed())
        {
          memcpy(dmiData.get_dmi_ptr() + i
              - dmiData.get_start_address(),
          &data,
          trans.get_data_length());
          delay += dmiData.get_write_latency();
        }
```

DMI start here

Normal b_transport

Get DMI Hint!

```
      } else {
        iSocket->b_transport(trans, delay);
      }
      if(trans.is_dmi_allowed() == true) {
        dmiData.init(); // Reset DMI descriptor
        dmi = iSocket->get_direct_mem_ptr(trans, dmiData);
      }
    }
    wait(delay);
  } // end for
  sc_stop();
} // end process

void invalidate_direct_mem_ptr(sc_dt::uint64 start_range,
                               sc_dt::uint64 end_range)
{
  dmi = false;
}

// Dummy methods
...
};
```

Try code on github:
https://github.com/TUK-SCVP/SCVP.artifacts/tree/master/tlm_lt_dmi

Fraunhofer IESE

Your notes:

we Pay Pass all the interconnect

DMI Interconnect


```
class exampleInterconnect : sc_module, tlm::tlm_bw_transport_if<>, tlm::tlm_fw_transport_if<>
{
    public:
    tlm::tlm_initiator_socket<> iSocket;
    tlm::tlm_target_socket<> tSocket;

    SC_CTOR(exampleInterconnect) {
        tSocket.bind(*this);
        iSocket.bind(*this);
    }

    void b_transport(tlm::tlm_generic_payload &trans, sc_time &delay) {
        delay = delay + sc_time(40, SC_NS);
        iSocket->b_transport(trans, delay);
    }

    bool get_direct_mem_ptr(tlm::tlm_generic_payload& trans, tlm::tlm_dmi& dmi_data) {
        bool dmi = iSocket->get_direct_mem_ptr(trans, dmi_data);

        dmi_data.set_read_latency( dmi_data.get_read_latency() + sc_time(40, SC_NS));
        dmi_data.set_write_latency( dmi_data.get_write_latency() + sc_time(40, SC_NS));

        return dmi;
    }

    void invalidate_direct_mem_ptr(sc_dt::uint64 start_range, sc_dt::uint64 end_range)
    {
        tSocket->invalidate_direct_mem_ptr(start_range, end_range);
    }

    // Dummy methods …
};
```

Forwarding DMI request on forward and backward path

Fraunhofer IESE

Your notes:

DMI Target

SW

```
class exampleTarget : sc_module, tlm::tlm_fw_transport_if<> {
    unsigned char mem[512];
    public:
    tlm::tlm_target_socket<> tSocket;

    SC_CTOR(exampleTarget) : tSocket("tSocket") {
        tSocket.bind(*this);
        SC_THREAD(invalidateProcess);
    }
    void invalidateProcess() {
        while(true) {
            wait(500, SC_NS);
            tSocket->invalidate_direct_mem_ptr(0,511);
        }
    }
    void b_transport(tlm::tlm_generic_payload &trans, sc_time &delay) {
        ...
        trans.set_dmi_allowed( true );
    }
    bool get_direct_mem_ptr(tlm::tlm_generic_payload& trans, tlm::tlm_dmi& dmi_data) {
        std::cout << "get_direct_mem_ptr called" << std::endl;
        dmi_data.set_dmi_ptr(mem);
        dmi_data.set_start_address(0);
        dmi_data.set_end_address(511);
        dmi_data.set_read_latency(sc_time(40, SC_NS));
        dmi_data.set_write_latency(sc_time(40, SC_NS));
        dmi_data.allow_read_write();
        return true;
    }
    // Dummy methods ...
};
```

In this example the DMI access is invalidated every 500 ns, which is just an artificial example

when you get B_transport you say hey DMA is enabled

Give Initiator a hint that DMI is possible

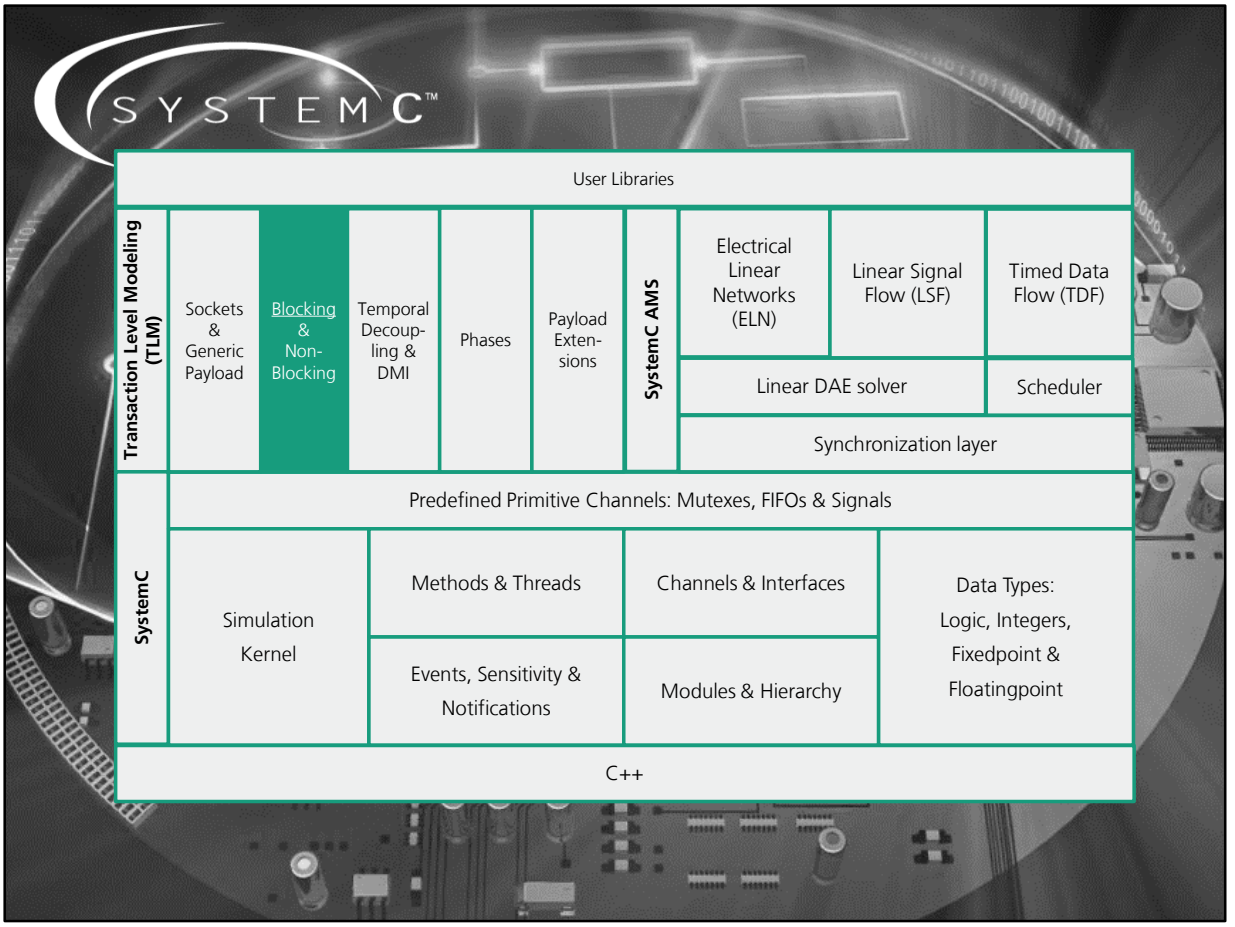
You Put here a pointer to memory structure and Put addresses ranges and you can also note latency here For this processes then when you go back to initiator so you can add up your local delay variable with this latencies

Configure DMI object with all relevant information

Fraunhofer IESE

Your notes:

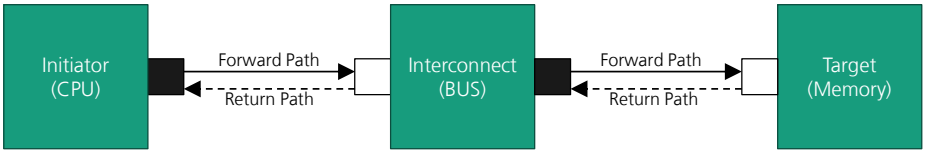
Just have an overview it's not important.



Your notes: There is another thing that helps you alot to set up, i built alot of virtual Platforms already and the booting is really annoying, we talked about booting of Linux that we can Fix it using DMI, but there is another problem which is the bootloader itself and logically you dont want to simulate bootloader but we have something really cool which called Transport Debug, which is nothing else than ablocking transport without any time.

Debug Transport transport_dbg

```
unsigned int transport_dbg(tlm_generic_payload trans);
```



- Gives an initiator debug access to memory in a target
- Similar to b_transport
  - Different: delay free, no waits, no event notifications
  - Uses generic payload
  - Same routing as b_transport
- Used for initialization, e.g. for bootloading

Fraunhofer IESE

Your notes: it has 2 Application;

① bootloader

you want to put bootloader in your memory and this should not simulated. you want to have everything in your memory and at once you start simulation on everything in memory. This can be done either by giving the memory the possibility of an interface to load the stuff in local memory or if the CPU can do this job using the transport_dbg fn.

② IF You want to check specific variable in your target for debuging purposes, maybe your target is close source thing so you can't change it anymore because it's combiled library and you want to access this register to check something, you can use transport-dbg

① Bootloading
② Debuging

Debug Transport transport_dbg

```
class Initiator : sc_module, tlm::tlm_bw_transport_if<> {
    ...
    void process() {
        ... // End of simulation:
        dumpMemory();
    }

    void dumpMemory()
    {
        unsigned char buffer[64];

        tlm::tlm_generic_payload trans;
        trans.set_address(0);
        trans.set_read();
        trans.set_data_length(64);
        trans.set_data_ptr(buffer);

        unsigned int n = iSocket->transport_dbg(trans);

        for(unsigned int i = 0; i < n; i++) {
            std::cout << std::hex
                      << std::setfill('0')
                      << std::setw(2)
                      << (unsigned int)buffer[i];

            if((i+1)%8 == 0) {
                std::cout << std::endl;
            }
        }
    }
};
```

```
class Target : sc_module, tlm::tlm_fw_transport_if<>
{
  ...
  void b_transport(... &trans, ... &delay)
  {
    ...
  }

  unsigned int transport_dbg(&trans)
  {
    if (trans.get_address() >= 1024) {
      return 0;
    }

    if(trans.get_command() == tlm::TLM_WRITE_COMMAND)
    {
       memcpy(&mem[trans.get_address()],
              trans.get_data_ptr(),
              trans.get_data_length());
    } else /* tlm::TLM_READ_COMMAND */ {
      memcpy(trans.get_data_ptr(),
             &mem[trans.get_address(),
             trans.get_data_length());
    }
    return trans.get_data_length();
  }
};
```

Try code on github:
https://github.com/TUK-SCVP/SCVP.artifacts/tree/master/tlm_lt_debug_transport

Fraunhofer IESE

Your notes: