codesign environment

Schedule

- 1. Introduction
- 2. System Modeling Language: System C *
- 3. HW/SW Cosimulation *
- 4. C-based Design *
- 5. Data-flow Model and SW Synthesis
- 6. HW and Interface Synthesis
  (Midterm)
- 7. Models of Computation
- 8. Model based Design of Embedded SW
- 9. Design Space Exploration
  (Final Exam)
  (Term Project)

codesign environment

References

- **TLM Cosimulation**
  - [1] L. Cai and D. Gajski, “Transaction Level Modeling: An Overview,” CODES+ISSS 2003, pp.19-24, October, 2003.
  - [2] F. Ghenassia, “Transaction-Level Modeling with SystemC,” Springer, 2005.
  - [3] Thorsten Grotker et al. System design with SystemC, Kluwer Academic Publishers, 2002.
- **SystemC**
  - [4] 기안도, “SystemC 시스템모델링 언어,” 대영사, 2004.

codesign environment

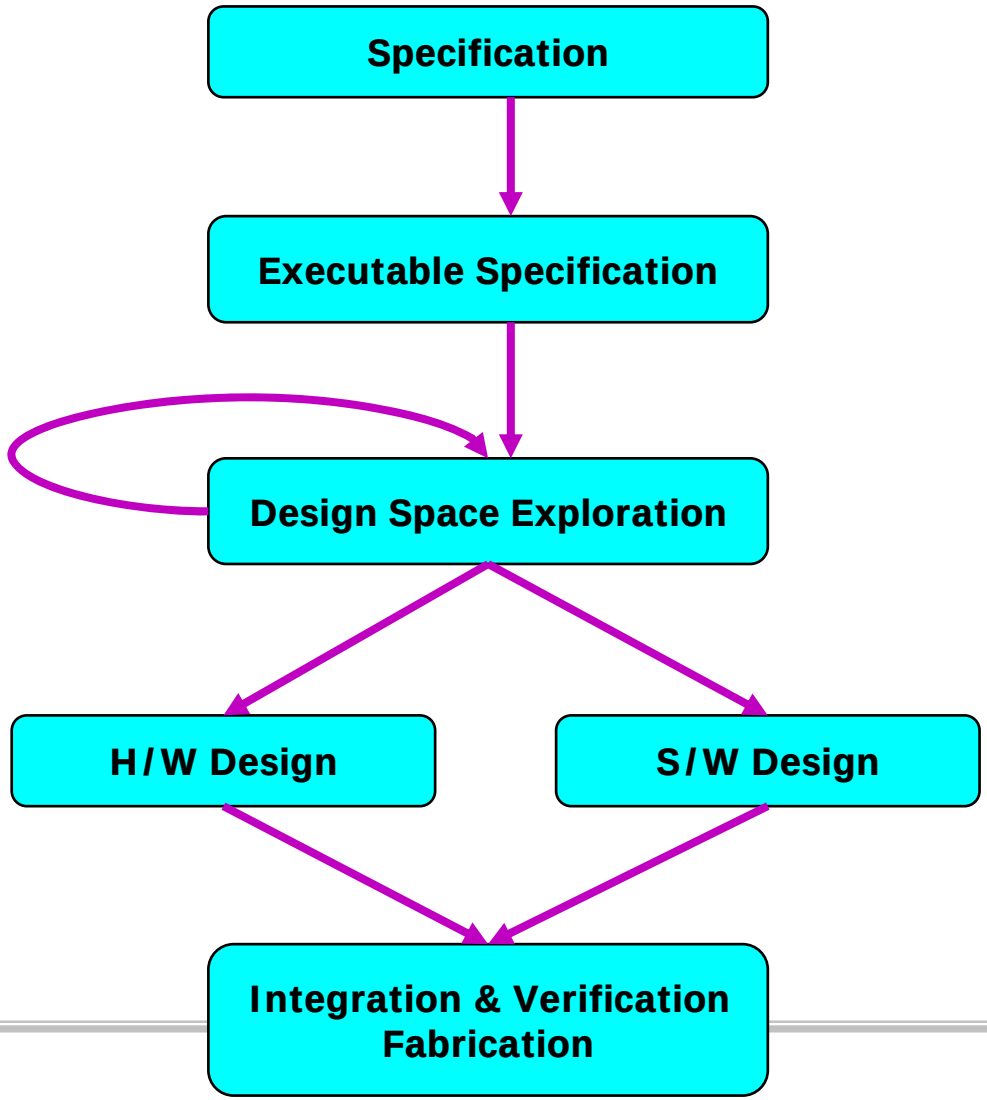
Contents

- **Transaction Level Modeling**
- **System C Language**

Motivation

- **Two different languages are used for SoC Design**
  - Software: C or C++
  - Hardware: HDL (Hardware Description Language)
- **System verification**
  - Real prototype: too expensive
  - Slow co-verification: RTL simulator + Processor simulator
- **Need**
  - System-level design language with higher abstraction
    - *Earlier examples: SpecC, CowareC, Superlog*
    - *OSCI SystemC*
  - Faster co-simulation with higher level modeling
    - *TLM (Transaction Level Modeling)*

System Design Flow



Natural Language

Informal Specification : English
Formal Specification : UML

C/C++ Programming
MATLAB

Transaction Level Modeling

Untimed Model
Cycle Approximate Model
Cycle Accurate Model

H/W Design

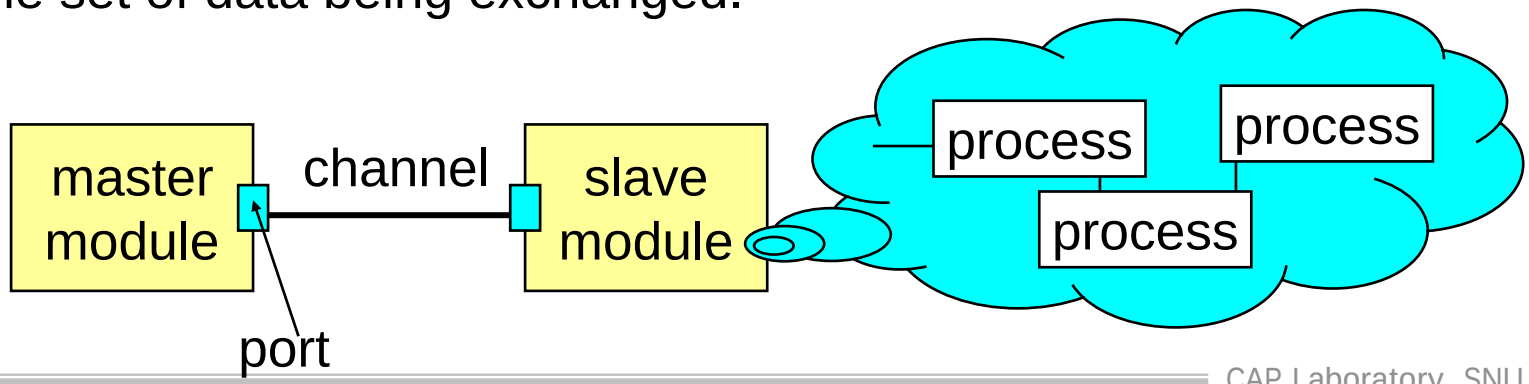
RTL Description (using HDL)

S/W Design

Optimized C/C++ Programming
DSP Assembly Coding

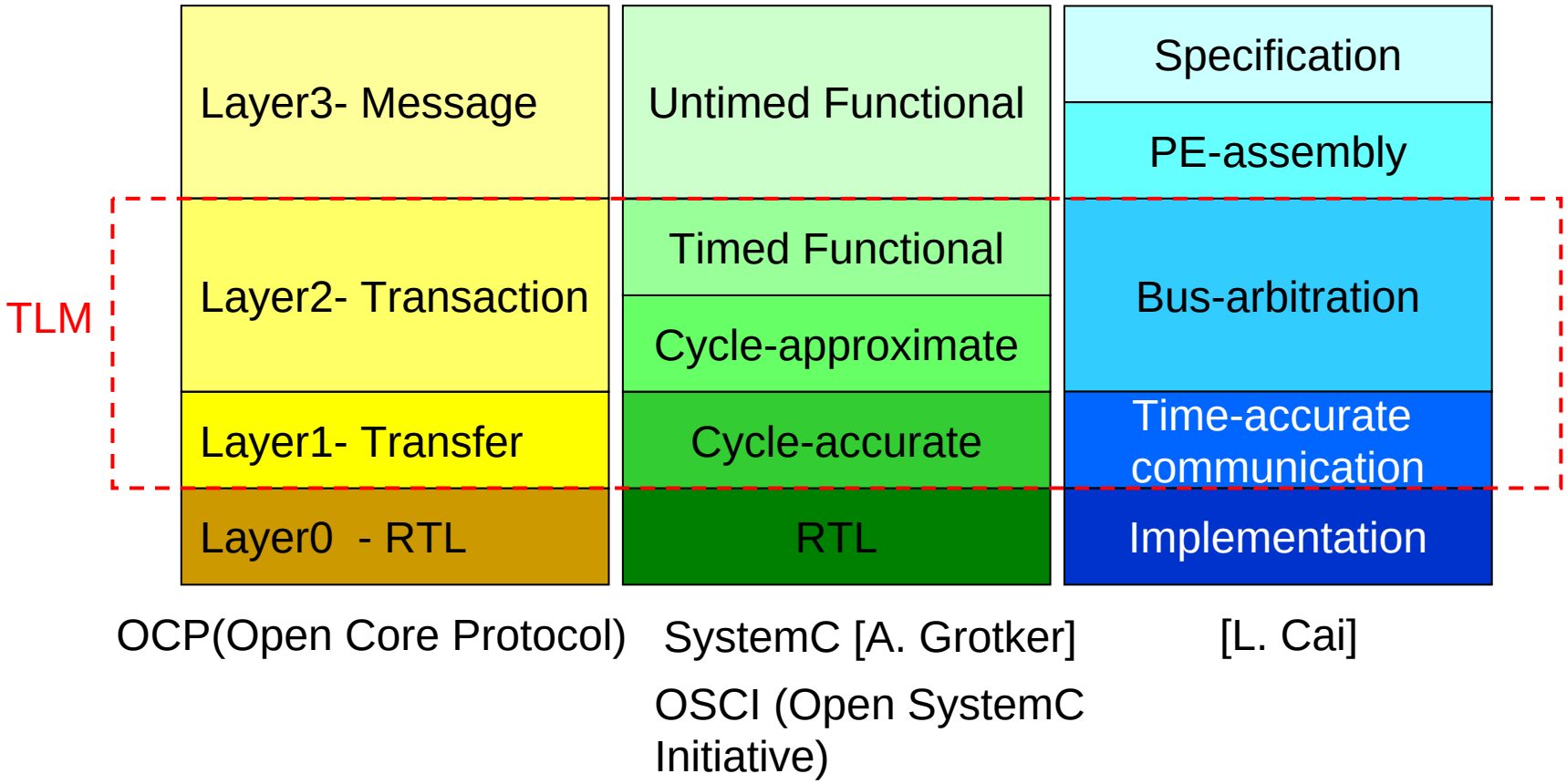
Principles of TLM

- **TLM separates communication from computation**
  - A system consists of components that are modeled as modules with a set of concurrent processes (or threads) that represent their behavior.
  - The modules exchange communication in the form of transaction through an abstract channel.
  - Modules and channels are bound to each other by means of communication ports.
  - A master module initiates a transaction and a slave module receives the set of data being exchanged.



Abstraction Models

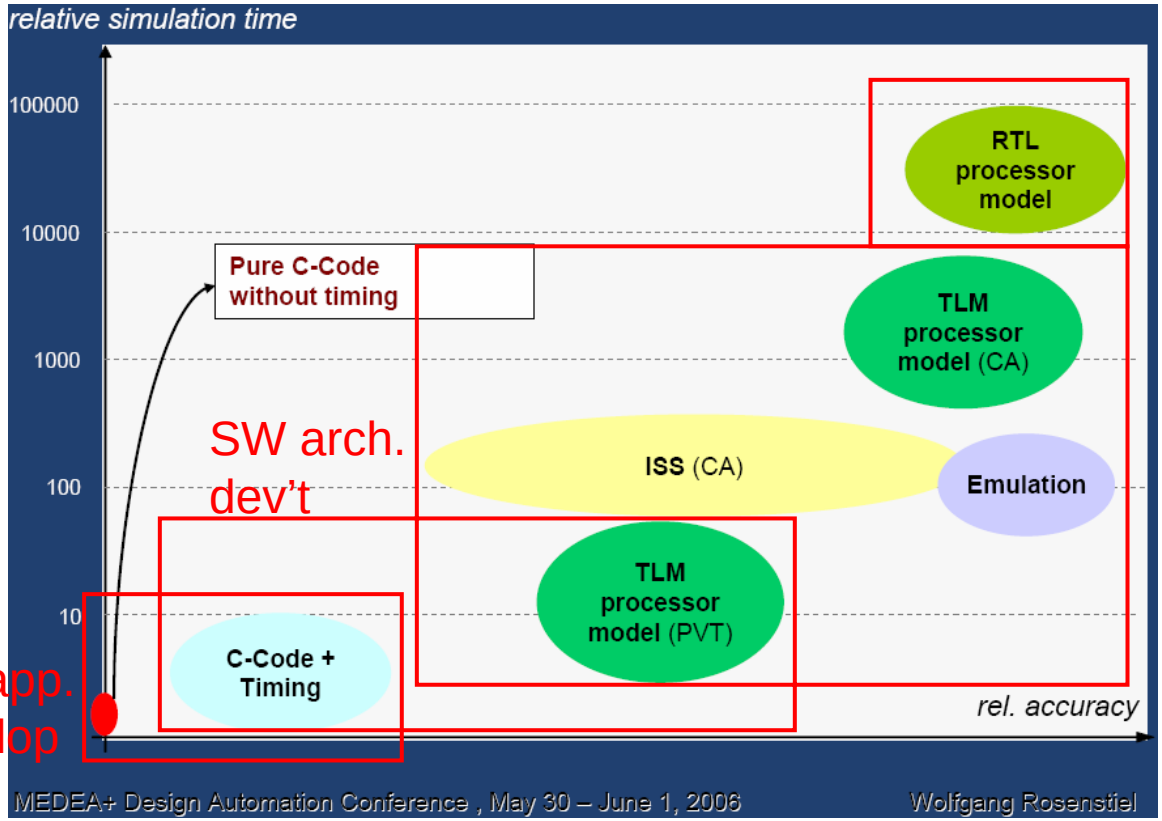
- **Different classification**



Simulation Performance

- **Tradeoff between speed and accuracy**
  - We can select the appropriate level of abstraction

Low speed



SW app. develop ment

SW arch. dev't

implementation

HAL/HdS develpment

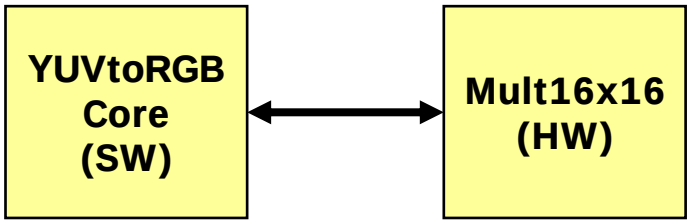
System integration/verfication

*PVT: programmer's view + timing = cycle-approximate

High accuracy

Untimed TLM

- **Purpose**
  - Functional verification before architectural decision is made
  - Early functional SW development
- **Example: YUV-to-RGB conversion**




```
int YUVtoRGB (int Y, int U, int V)
{
    int R,G,B;
    R = Y+ (MULT(90,V-128) >>6);
    G = Y- ((MULT(22,U-128) + MULT(46,V-128))>>6);
    B = Y+ MULT(114,U-12);
    return (R<<16) | (G<<8) | B;
}
```

```
int MULT (int A, int B)
{
    return A*B;
}
```

Timed TLM – function part

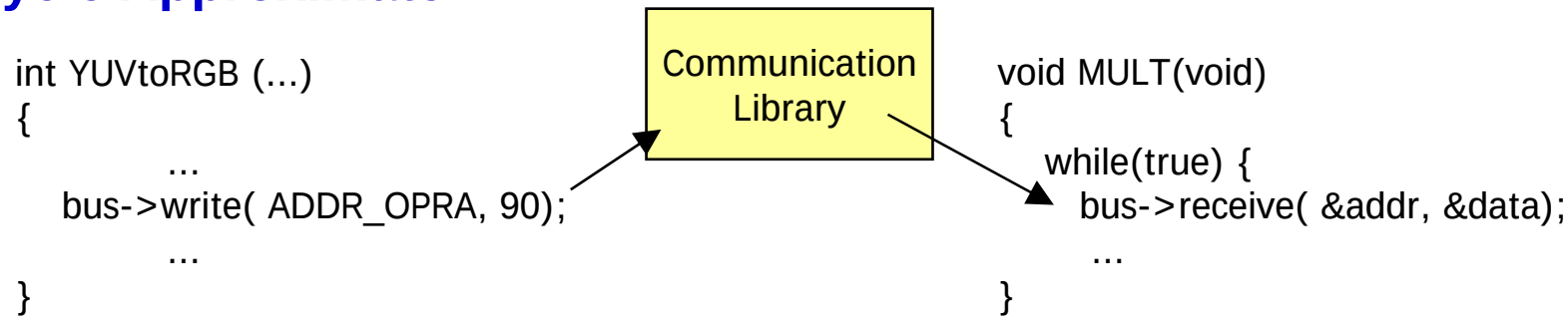
- **Timed Functional**
  - Annotate timing information

```
int MULT(int A, int B)
{
    wait(5);
    return A*B;
}
```

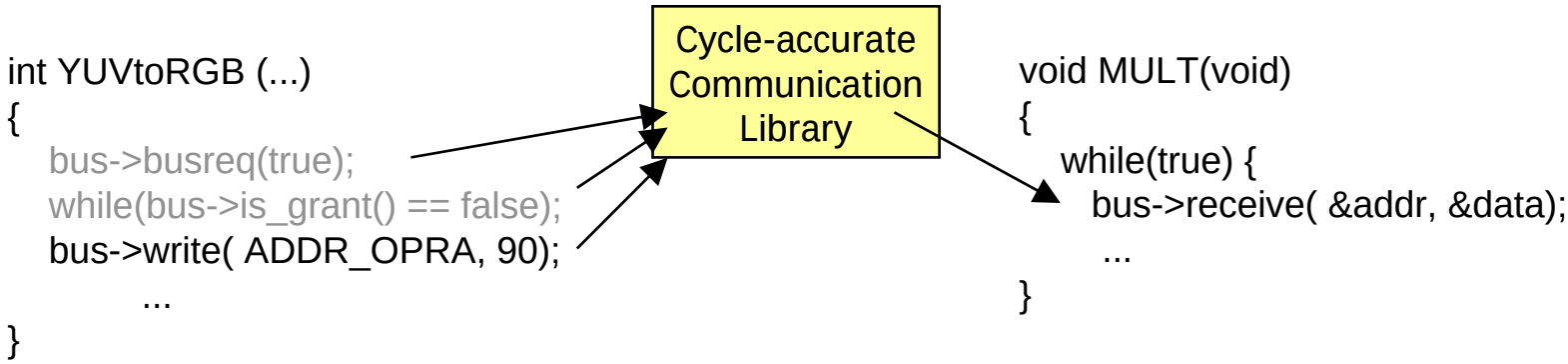
- **Cycle Approximate (Instruction Accurate)**
  - Execute SW code on the instruction-level processor simulator
    - *Performance: a few hundreds MIPS (as of 2007)*
- **Cycle Accurate**
  - Execute SW code on the cycle-accurate processor simulator
    - *Performance: a few MIPS (as of 2007)*
  - HW: functional description at each clock cycle

Timed TLM – communication part

- **Cycle Approximate**



- **Cycle Accurate**
  - After communication architecture is determined, model it accurately at the cycle level

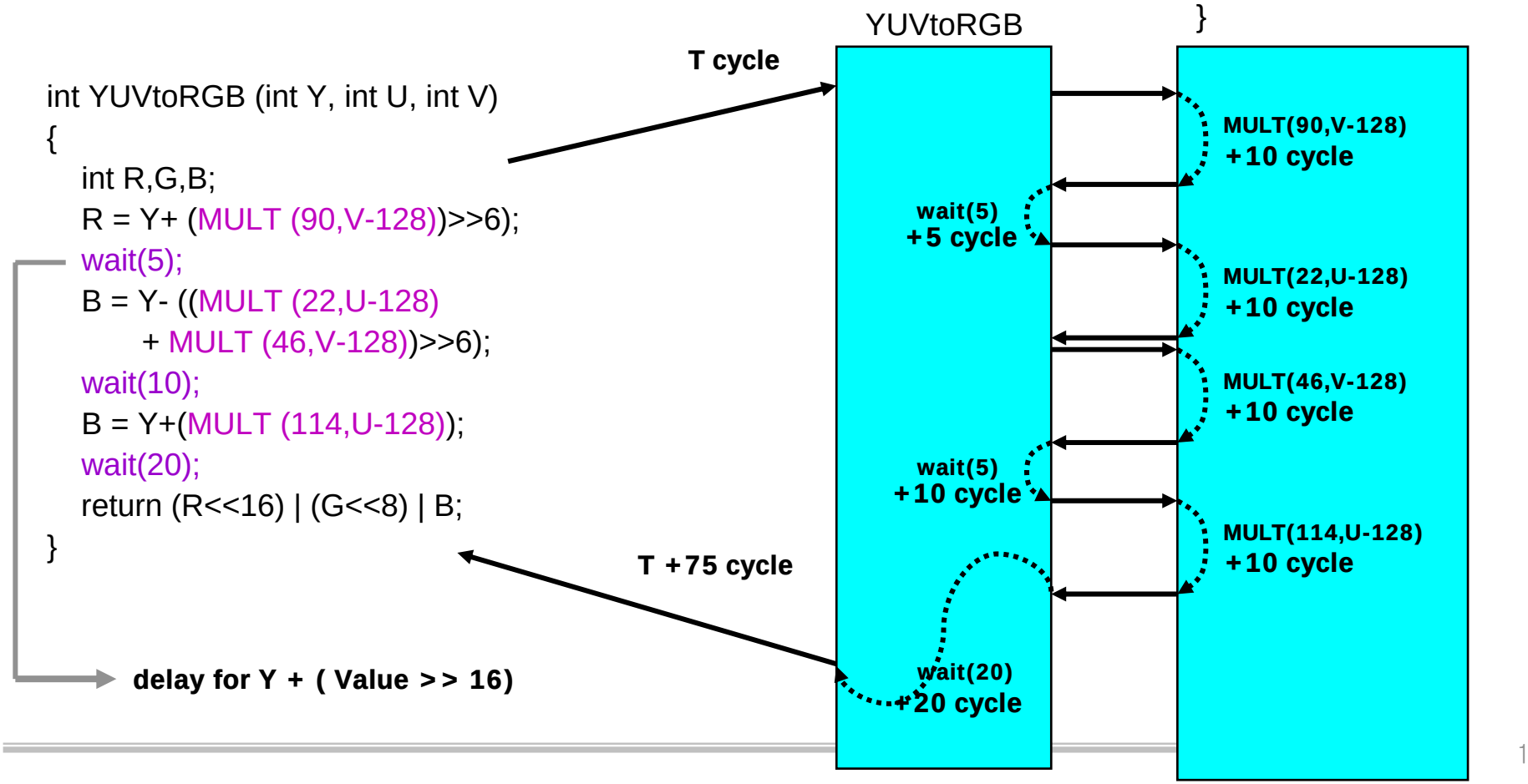


Timed Functional TLM

- **Direct function call for communication**
- **No communication overhead is modeled**

```
int MULT (int A, int B)
{
    wait(10);
    return A*B;
}
```

```
int YUVtoRGB (int Y, int U, int V)
{
    int R,G,B;
    R = Y+ (MULT (90,V-128))>>6);
    wait(5);
    B = Y- ((MULT (22,U-128)
          + MULT (46,V-128))>>6);
    wait(10);
    B = Y+(MULT (114,U-128));
    wait(20);
    return (R<<16) | (G<<8) | B;
}
```



Courtersy by Prof. Kiyoung Choi, SNU

Cycle Approximate TLM

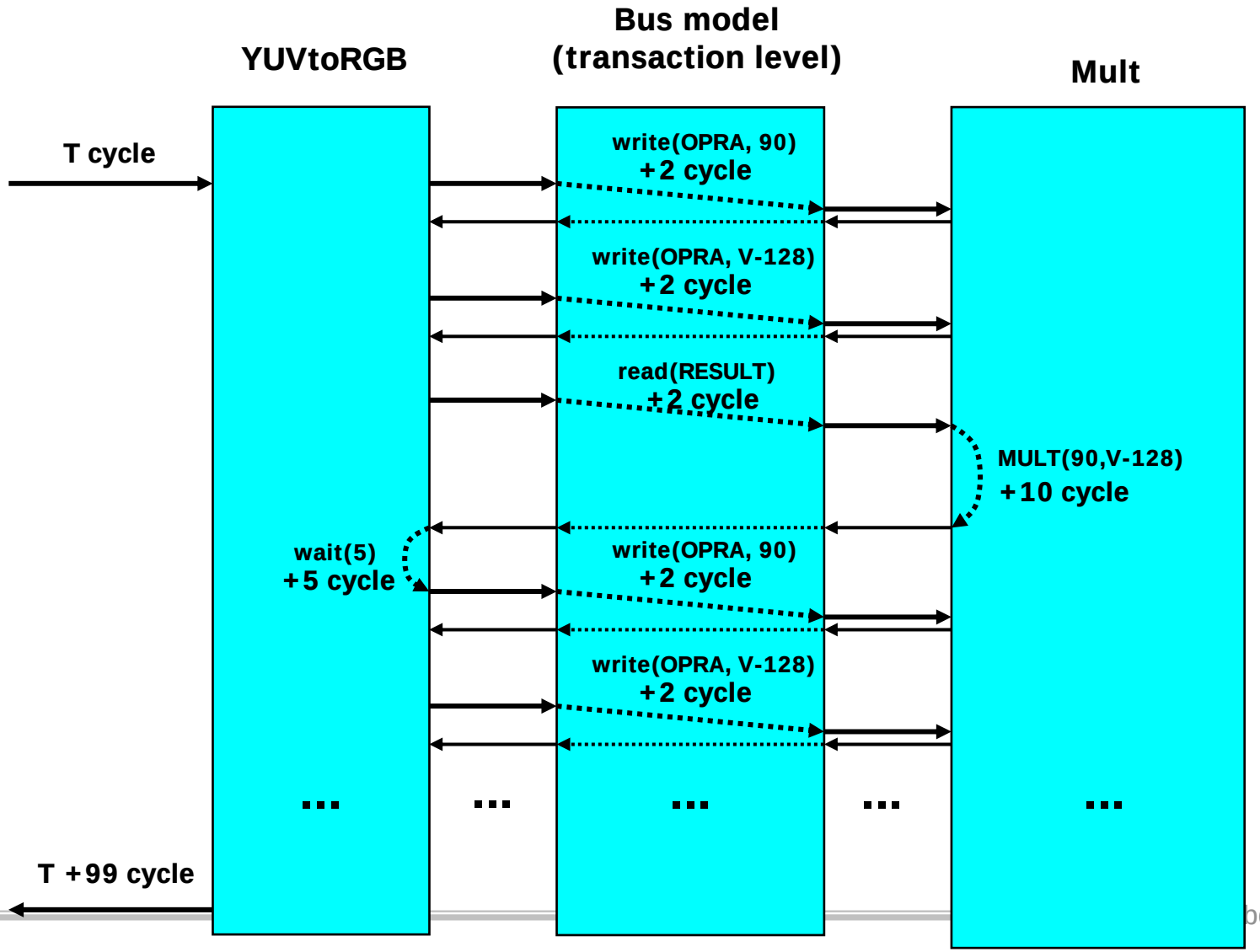
- **After communication method is determined, estimate the communication overhead approximately**
  - (ex) HW: AHB bus, SW: polling

```
int YUVtoRGB(int Y, int U, int V)
{
    int R,G,B;
    bus->write( ADDR_OPRA, 90);
    bus->write( ADDR_OPRB, V-128);
    R = Y+ (bus->read( ADDR_RESULT)>>6);
    wait(5);

    bus->write( ADDR_OPRA, 22);
    bus->write( ADDR_OPRB, V-128);
    R = bus->read( ADDR_RESULT);
    bus->write( ADDR_OPRA, 46);
    bus->write( ADDR_OPRB, U-128);
    R += bus->read( ADDR_RESULT);
    R = Y – (R>>6);
    wait(10);

    bus->write( ADDR_OPRA, 114);
    bus->write( ADDR_OPRB, U-128);
    R = Y +(bus->read( ADDR_RESULT)>>6);
    wait(20);
    return (R<<16) | (G<<8) | B;
}
```

```
void MULT(int A, int B)
{
    int opra, int oprb;
    int addr, data;
    while(1)
    {
        bus->receive( &addr, &data);
        if(addr == ADDR_OPRA)
            opra = data;
        else if(addr == ADDR_OPRB)
            oprb = data;
        else {
            wait(10);
            bus->set_data( opra*oprb);
        }
    }
}
```

ADDR_OPRA : Operand A Address
ADDR_OPRB : Operand B Address
ADDR_RESULT : Result Address

Cycle Approximate TLM: Timing



Courtersy by Prof. Kiyoung Choi, SNU

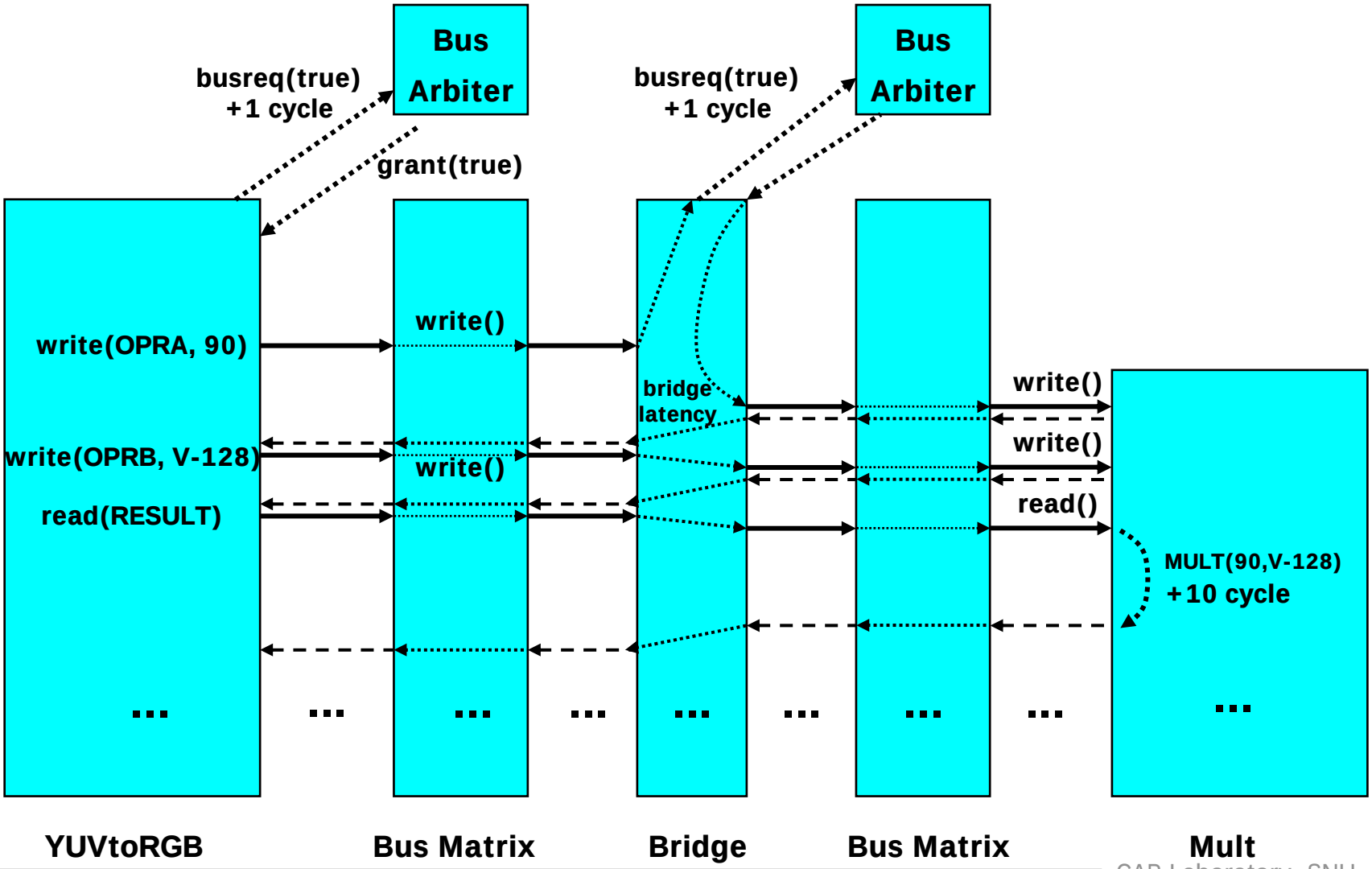
Bus Functional Model

- **Communication is modeled at the cycle level**

```
int YUVtoRGB(int Y, int U, int V)
{
    int R,G,B;
    bus->busreq(true);
    while( bus->grant() == false);
    bus->write( ADDR_OPRA, 90);
    bus->write( ADDR_OPRB, V-128);
    R = Y+ (bus->read( ADDR_RESULT)>>6);
    bus->bus_req(false);
    wait(5);

    bus->busreq(true);
    while(bus->grant() == false);
    bus->write( ADDR_OPRA, 22);
    bus->write( ADDR_OPRB, V-128);


... Omitted ...

}
```

```
void MULT1(int A, int B)
{
    int opra, int oprb, int oprc;
    int addr, data;
    while(1)
    {
        bus->receive(&addr, &data);
        if(addr == ADDR_OPRA)
            opra = data;
        else if(addr == ADDR_OPRB)
            oprb = data;
        else {
            bus->ready(false);
            wait(10);
            bus->set_data( opra*oprb);
        }
    }
   bus->ready(true);
}
```

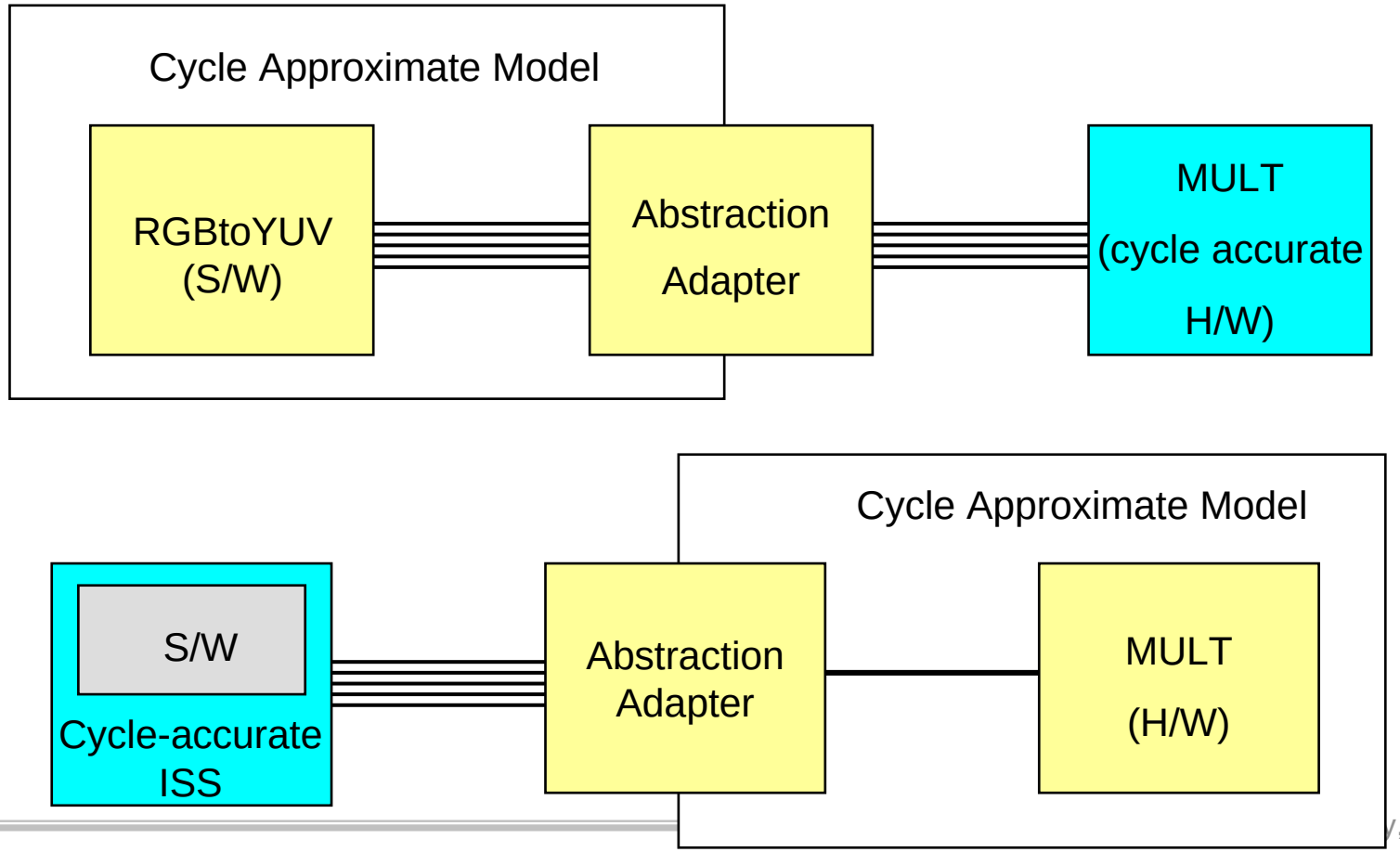
Bus Functional Model – Timing



Courtersy by Prof. Kiyoung Choi, SNU

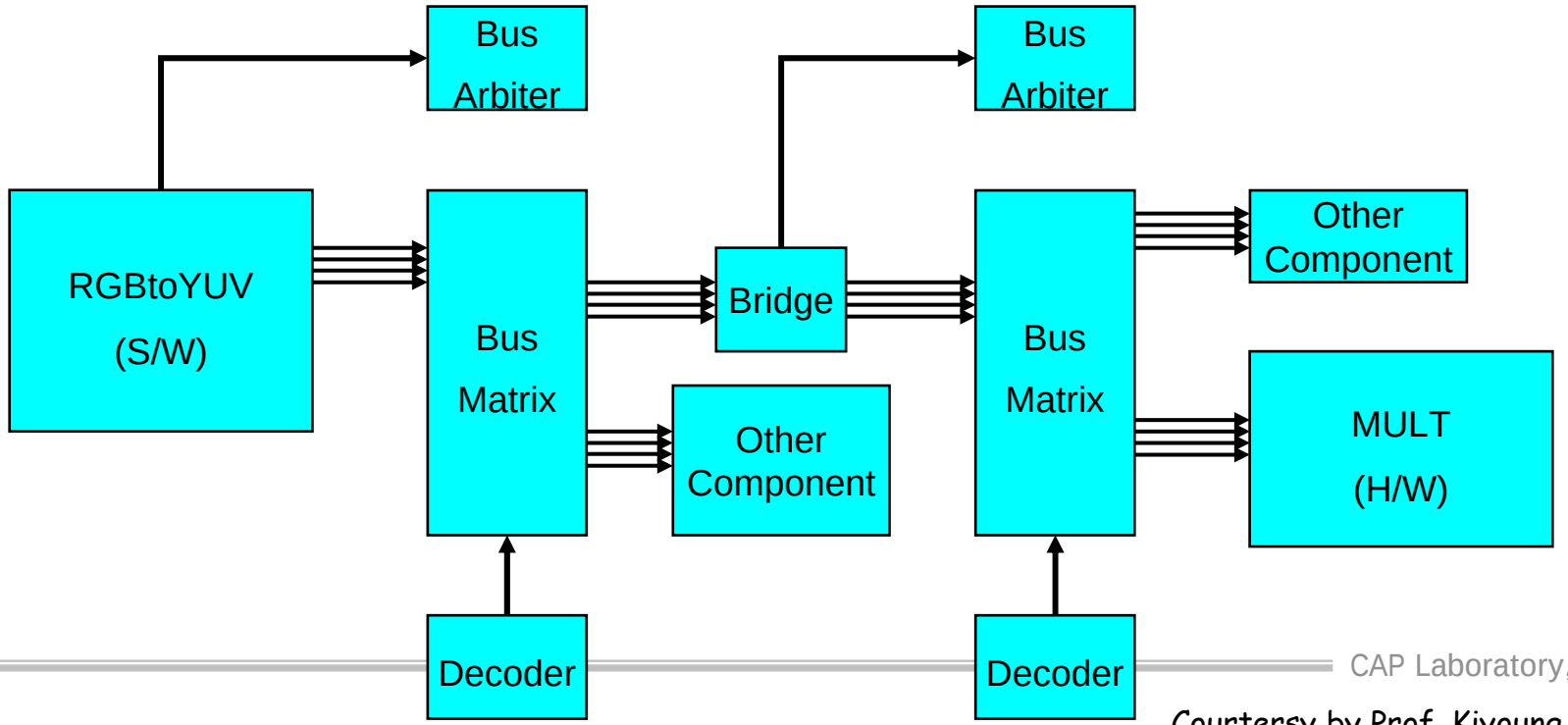
Cycle Accurate TLM

- **Cycle accurate computation model is mainly used for verification of individual component**



Cycle Accurate Model

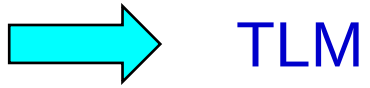
- **Cycle Accurate TLM**
  - Cycle-accurate ISS + cycle accurate HW model + bus functional model of communication
- **RTL (Pin Accurate Level)**
  - Cycle-accurate ISS + pin-level HW model



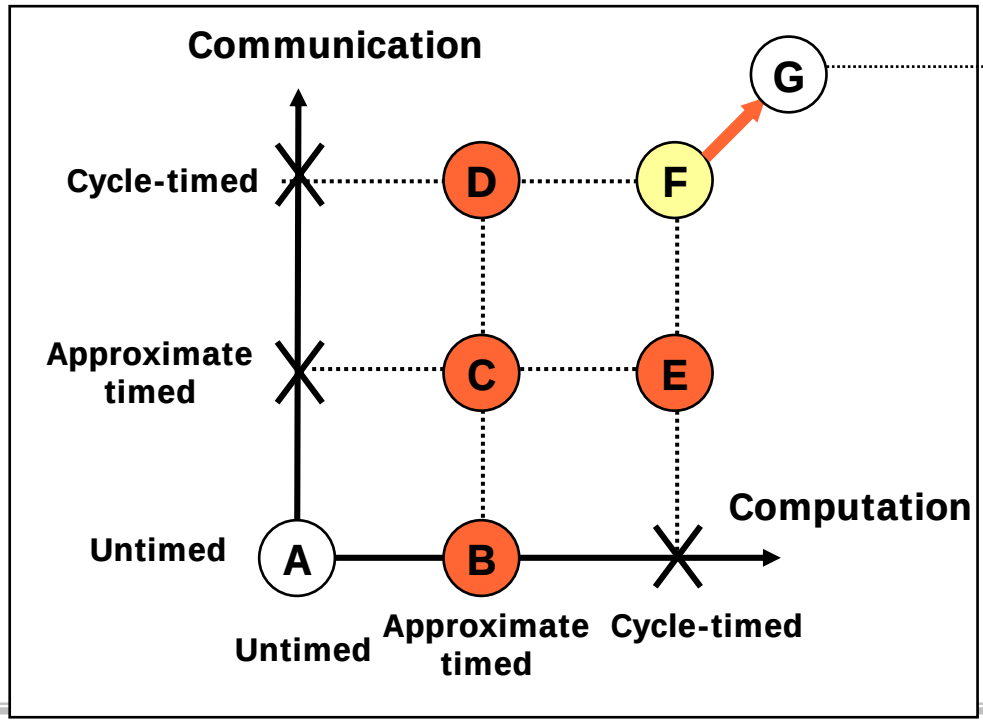
Courtersy by Prof. Kiyoung Choi, SNU

Transaction Level Model: Summary

B : Timed Functional Model
C : Cycle Approximate Model
D : Bus Functional Model
E : Cycle Accurate Computation Model



TLM



codesign environment

Contents

- **Transaction Level Modeling**
- **System C Language**

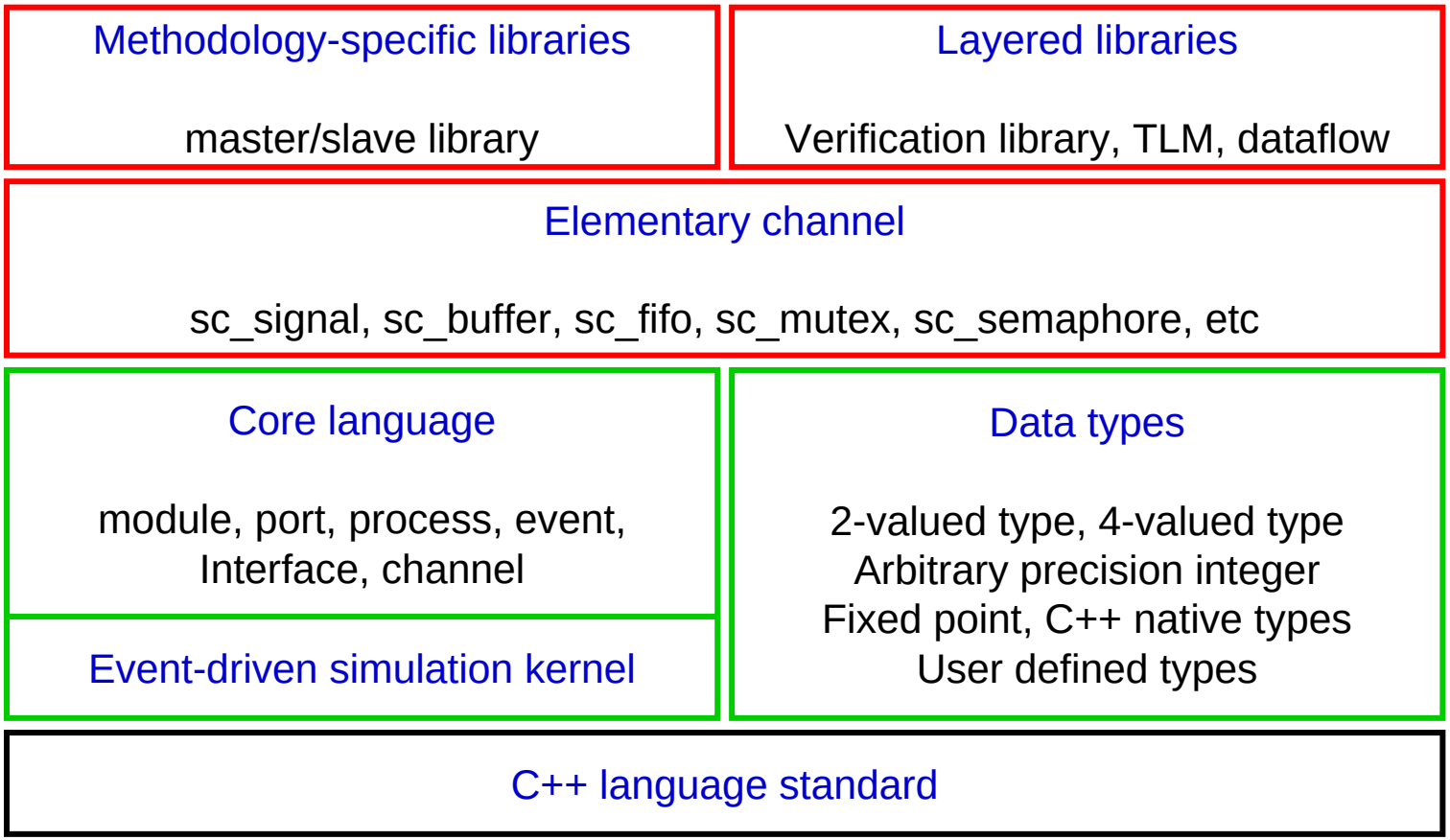
Introduction

- SystemC is an ANSI standard C++ class library for system and hardware design for use by designers and architects who need to address complex systems that are a hybrid between hardware and software (IEEE Std 1666-2005)
  - **SystemC 1.x: HW modeling + simulation kernel**
  - **SystemC 2.x: System modeling (algorithm/functional level modeling)**
  - **SystemC 3.x: (yet to be released) software APIs**
- Any C++ compiler can compile SystemC
- Language definition is publicly available
  - **OSCI: Open SystemC Initiative (www.systemc.org)**

More than C++

- **Support Concurrency**
  - A SystemC program consists of modules: (*SC_MODULE*)
  - A module can be a concurrent process: three types of processes:
    - *SC_METHOD, SC_THREAD, SC_CTHREAD*
  - Communication between modules is done by signal exchange through ports
- **Timing**
  - Clock module: sc_clock
    - *Clocks are special signals that run periodically and trigger clocked processes*
- **Rich set of data types for HW**

SystemC Language Structure



Modules: Basic Unit of Design

- **Hierarchical Entity**
- **Syntax**

```
SC_MODULE(module_name) {

/* port declaration */
/* local channel(signal) declaration */
/* variable declaration */
/* process declaration */
/* other method declaration */
/* submodule declaration */

/* constructor */
   SC_CTOR(module_name) {
         /* module instantiation and
            channel binding */
         /* process registration */
         /* sensitivity list */
         /* module variable initialization */
  }
}
```

```
// counter.h
#include "systemc.h"

SC_MODULE(counter) {
sc_in<bool> clk, reset, go;
sc_out<char> value;

unsigned char local_value;
void do_count();


SC_CTOR(counter) {
   SC_METHOD(do_count);
   sensitive << reset << go << clk.pos();
   local_value = 0;
   value.initialize(0);
}
};
```

Ports

- **Define the interface to each module**
- **Three types of ports depending on the direction**
  - Input: sc_in
  - Output: sc_out
  - Bidirectional: sc_inout
- **Member functions**
  - read(): read the current value
  - write(value): write a new value
  - event(), pos()/neg()
- **Syntax**

```
sc_in{out,inout}<data_type> port_name;
```

Signals

- **Convey information between sub-modules within a module**
- **Port of sub-module defines the direction of data transfer**
- **Member functions**
  - read(): non-blocking read
  - write(value): non-blocking write
  - event(), pos()/neg(), delay()

module instantiation with named connection

module instantiation with positional connection

```
// main.cpp
#include "counter.h"
#include "test.h"

int sc_main(int argc, char* argv[]) {
sc_signal<char> value;
sc_signal<bool> reset, go;

sc_clock CLK("clock", 10, SC_NS);

counter CNT("CNT");
CNT.reset(reset); CNT.go(go);
CNT.value(value); CNT.clk(CLK);

test TST("TST");
TST(CLK,reset,go,value);
....
}
```

codesign environment

Processes

- A member function without arguments and return value
- No nested call is allowed

	SC_METHOD	SC_THREAD	SC_CTHREAD
Infinite loop	No	Yes	Yes

construct	SC_METHOD(pro)
	sensitive(sig);
	sensitive_pos(sig);
	semsitive_neg(sig);
		SC_THREAD(pro);
		sensitive(sig);
		sensitive_pos(sig);
		semsitive_neg(sig);
			SC_CTHREAD(pro, clock.pos());
			SC_CTHREAD(pro. clock,neg());

suspend	NA	wait()	wait(); wait(n*);
local variable	lost	saved	saved
Static sensitivity	signal events	signal events	clock edge
Dynamic sensitivity	next_trigger(ev);	wait(ev)	wait_until(exp); watching(exp);
Usage	Comb. logic	Behavior, test bench	FSM

n*: positive integer

Processes: Example

```
SC_MODULE(add1){
sc_in<int> A,B;
sc_out<int> C;

void do_add() {C=A+B;}

SC_CTOR(add1) {
   SC_METHOD(do_add);
   sensitive<<A<<B;
   }
}
```

```
SC_MODULE(add2){
sc_in<int> A,B;
sc_out<int> C;

void do_add() {
   while(1) {
         C=A+B;
         wait();
   }
}

SC_CTOR(add2) {
   SC_THREAD(do_add);
   sensitive<<A<<B;
   }
}
```

```
SC_MODULE(add3){
sc_in<int> A,B;
sc_out<int> C;
Sc_in<bool> clk;

void do_add() {
   while(1) {
         C=A+B;
         wait();
   }
}

SC_CTOR(add3) {
   SC_CTHREAD(do_add,clk.pos());
   }
}
```

wait() relinquishes control until the next change of a signal on the sensitivity list

Sensitivity List

- **Determine when to trigger the process**
- **Two kinds**
  - Static sensitivity: defined in SC_CTOR

```
sensitive << event [<< event]…;      //streaming style
sensitive(event [, event]…);         //functional style
```

  - Dynamic sensitivity: defined inside the process definition
    - *SC_METHOD: next_trigger(event)*
    - *SC_THREAD: wait(event)*
    - *SC_CTHREAD: wait_until(expression)*
    - *SC_CTHREAD: watching(expression)*
      - Defined on process registration to restart the process when the expression is met.

Timing in SystemC

- **Time in SystemC**
  - units: SC_SEC, SC_MS, SC_US, SC_NS, SC_PS, SC_FS
  - "sc_time" class defines a time object (ex) *sc_time t(15, SC_15)*
  - Define the time unit
    - *sc_set_default_time_unit(1, SC_NS): set the default time unit*
    - *sc_set_time_resolution(1,SC_NS): set the default time resolution*
      - The default is SC_PS if omitted
  - Keep track of the current simulation time: *cout << sc_time_stamp()<<endl;*
- **Clock**
  - The only thing in SystemC that has a notion of real time
  - *sc_clock name("name", period, duty_cycle, first, rising) or*
    sc_clock name("name", period, time_unit, duty_cycle, first, first_time_unit, rising)
    - *first: time elapsed until encounters the first edge*
    - *rising: set "TRUE" if the first edge is rising or "FALSE" if falling*

Test Bench Module and Main File

```
// test.h
#include "systemc.h"

SC_MODULE(test) {
sc_in<bool> clock;
sc_out<bool> reset, go;
sc_in<char> value;

void do_test();

SC_CTOR(test) {
   SC_CTHREAD(do_test, clock.neg());
   reset.initialize(1);
   go.initialize();
   }
}
```

```
void test::do_test() {
  wait();
  reset.write(0);
  wait();
  while (true) {
    go.write(1);
    wait(5);
    go.write(0);
    wait(2);
  }
}
```

```
// main.cpp
#include "counter.h"
#include "test.h"

int sc_main(int argc, char* argv[]) {
sc_signal<char> value;
sc_signal<bool> reset, go;
sc_clock CLK("clock", 10, SC_NS);

counter CNT("CNT");
CNT.reset(reset); CNT.go(go);
CNT.value(value); CNT.clk(CLK);

test TST("TST");
TST(CLK,reset,go,value);

// trace file generation: optional
sc_trace_file* tf = sc_create_vcd_trace_file("wave");
sc_trace(tf,CNT.local_value, "CNT.local_value");

sc_start(200, SC_NS);  /* simulation length = 20 ns */
sc_close_vcd_trace_file(tf);
}
```

codesign environment

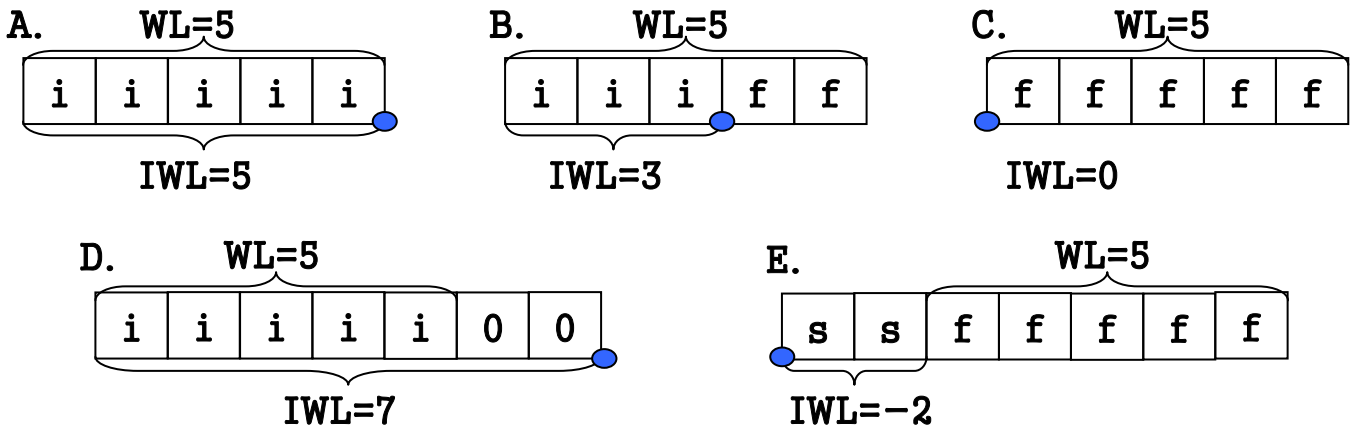
SystemC Data Types

	type	explanation	
Boolean	sc_bit	Single bit (same as C++ bool type)	“0”, “1”
	sc_bv<N>	N-bit vector of sc_bit	
logic	sc_logic	4 value logic	“0”,”1”,”X”,”Z”
	sc_lv<N>	N-bit vector of sc_logic	
integer	sc_int<N>	N-bit signed integer	0 < N <= 64
	sc_uint<N>	N-bit unsigned integer	
	sc_bigint<N>	Arbitrary sized signed int.	
	sc_bigunit<N>	Arbitrary sized unsigned int.	
Fixed-point	sc_fixed<…>	Templated signed fixed point	
	sc_ufixed<…>	Templated unsigned fixed point	
	sc_fix<>	Untemplated signed fixed point	
	sc_fix<>	Untemplated unsigned fixed point	

* Use C++ types if exist since they are faster in simulation

Fixed Point Data Types

- **Definition**
  - sc_{fixed,ufixed}<wl, iwl, q_mode, o_mode, n_bits> name;
    or sc_{fix,ufix} name(wl, iwl, q_mode, o_mode, n_bits)
    - *wl: total word length (total number of bits)*
    - *iwl: integer word length*
    - *q_mode: quantization mode (rounding behavior)*
    - *o_mode: overflow mode (saturation behavior)*
    - *n_bits: number of saturated bits*



Modes in Fixed Point Type

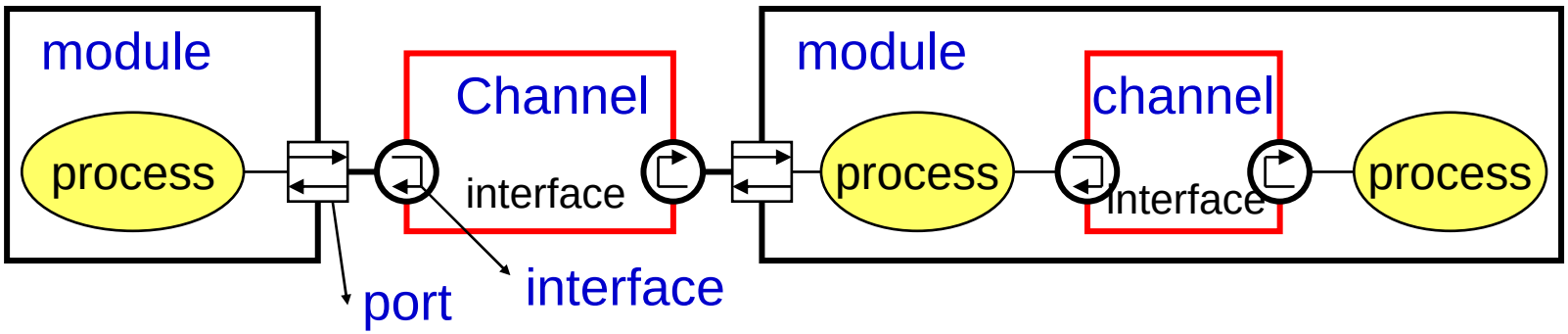
- **Quantization mode**
  - SC_RND: rounding to infinity
  - SC_RND_ZERO: rounding to zero
  - SC_TRN: truncation (default)
  - SC_TRN_ZERO: truncation to zero
  - SC_RND_MIN_INF, SC_RND_CONV
- **Overflow mode**
  - SC_SAT: saturation
  - SC_SAT_ZERO: saturation to zero
  - SC_WRAP, n_bits=0: wrap-around (default)
  - SC_SAT_SYM, SC_WRAP_SM, and others

Operators

- **Comparison** == != < <= > >=
- **Arithmetic** ++ -- * / % + -
- **Bitwise** ~ & | ^
- **Assignment** = &= |= ^= *= /= %= += -+ <<= >>=
- **Bit Selection** **bit**(*idx*), **[***idx***]**
- **Range Selection** **range**(*high, low*), **(***high, low***)**
- **Conversion** **to_double**(), **to_int**(), **to_int64**(), **to_long**(), **to_uint**(), **to_uint64**(), **to_ulong**(), **to_string**(*type*)
- **Testing** **is_zero**(), **is_neg(), length**()

SystemC Channel

- **SystemC has two types of channel : primitive and hierarchical.**
- **Primitive channels**
  - Contains no hierarchy, no processes.
  - Designed to be very fast due to their simplicity.
  - sc_signal, sc_buffer, sc_mutex, sc_fifo, sc_semaphore
    - *Blocking channels (mutex, fifo, semaphore) may not be used in SC_METHOD.*



sc_signal and sc_buffer

- **Both supports 1 writer & multiple readers**
- **Basic methods**
  - read(), write(), delayed(), event(), etc
- **Evaluate-update protocol**
  - New value is visible after the current process is suspended
    - *(ex) sc_signal<int> channel; …; value = 0;*
      while (true) {
      value++; channel.write(value); cout << channel.read();wait();}
- **Difference**
  - sc_buffer generates an event at every write() while sc_signal generates an event only when the value is changed.
    - *(ex) sc_signal(int) c1; sc_buffer(int) c2;*
      c1.write(3); c2.write(3); c1.write(3); c2.write(3)

does not generate an event

sc_fifo

- **Point-to-point channel between 1 reader and 1 writer supporting FIFO principle**
- **Class Definition**
  - sc_fifo_in<T>: sc_fifo<T> *(ex) sc_fifo_out<int> fout; sc_fifo foo(10);*
    sc_fifo_out<T>: sc_fifo<T>
  - The defualt size is 16 if size information is missing.

- **Methods**
  - Blocking read & write: read(), write()
  - Non-blocking read & write: nb_read(), nb_write() returns “false” if failed
  - Querry to check the size of empty space: num_free()

sc_mutex and sc_semaphore

- **These channels are shared by multiple readers and writers through exclusive access privilege.**
- **A mutex is a semaphore with a count of 1**
- **Usage example**

```
sc_mutex NAME;

// To lock the mutex NAME (wait until unlocked if in use)
NAME.lock();

// Non-blocking, returns true if success, else false
NAME.trylock ();

// To free a previously locked mutex
NAME.unlock()
```

```
sc_semaphore NAME(COUNT);

// To lock NAME (wait until unlocked if in use)
NAME.wait();

// Non-blocking, returns true if success, else false
NAME.trywait ();

// Returns number of available semaphore
NAME.get_value()

// To free a previously locked mutex
NAME.post()
```

Summary

- **Transaction Level Modeling**
  - Separates computation from communication
  - Faster cosimulation by higher level of modeling
  - Tradeoff between timing-accuracy and simulation-speed
    - *Timed Functional Model*
    - *Cycle Approximate Model*
    - *Bus Functional Model*
    - *Cycle Accurate Computation Model*

- **SystemC Language**
  - System design languages to support HW and algorithm modeling
  - Features
    - *Concurrency support: modules with three types of processes communicating with each other via channels*
    - *Timing management*
    - *Rich set of data types for HW support*

SystemC Examples-1

- **Simple adder program**
  - Adder_method : SC_METHOD
  - Adder_thread : SC_THREAD, sensitivity – clk
  - Adder_thread2 : SC_THREAD, sensitivity – a, b
  - Adder_thread3 : SC_THREAD, sensitivity – a, b, clk
  - Adder_cthread : SC_THREAD
- **Compare SC_METHOD, SC_THREAD, SC_CTHREAD**

SystemC Examples-1

```
// main.cpp
#include "adder_method.h"
#include "adder_thread1.h"
#include "adder_thread2.h"
#include "adder_thread3.h"
#include "adder_cthread.h"
#include "stimulus.h"
int sc_main(int argc, char *argv[]) {
  sc_signal<int> A, B, R_M, R_T1, R_T2, R_T3, R_C;
  sc_set_time_resolution(1, SC_NS); // V2.0
  sc_set_default_time_unit(1, SC_NS);
  sc_clock clock("clock", 20, SC_NS, 0.5, 15, SC_NS);

  adder_method AD_M("adder_method"); // instantiation
  AD_M(A, B, R_M);                   // port connection
  adder_thread AD_T1("adder_thread");
  AD_T1(A, B, R_T1, clock);
  adder_thread2 AD_T2("adder_thread2");
  AD_T2(A, B, R_T2);
  adder_thread3 AD_T3("adder_thread3");
  AD_T3(A, B, R_T3, clock.signal());
  adder_cthread AD_C("adder_cthread");
  AD_C(A, B, R_C, clock.signal());

  stimulus STIM("stimulus");  // instantiation
    STIM(A, B, clock.signal()); // port connection

    // trace file creation
    sc_trace_file *tf = sc_create_vcd_trace_file("wave");
    sc_trace(tf, clock, "clock");
    sc_trace(tf, A,     "A");
    sc_trace(tf, B,     "B");
    sc_trace(tf, R_M,   "R_M");
    sc_trace(tf, R_T1,  "R_T1");
    sc_trace(tf, R_T2,  "R_T2");
    sc_trace(tf, R_T3,  "R_T3");
    sc_trace(tf, R_C,   "R_C");
    sc_start(80, SC_NS);
    return(0);
}
```

SystemC Examples-1

```
// stimulus.h
#include "systemc.h"

SC_MODULE(stimulus) {
  sc_out<int> A, B;
  sc_in<bool> clk;

  void do_stim() {
    while (1) {
      A = 10; B = 20;
      wait();
      A = 13; B = 11;
      wait(); wait();
      A = 1; B = 3;
      wait();
      A = 11;
      wait();
      A = 100; B = 1;
      wait(); wait(); wait();
    }
  }
```

```
  SC_CTOR(stimulus) {
    SC_THREAD(do_stim);
    sensitive << clk;
  }
};
```

SystemC Examples-1

```
// adder_method.h
#include "systemc.h"

SC_MODULE(adder_method) {
  sc_in<int> A, B;
  sc_out<int> C;

  void do_adder() { C = A + B; }

  SC_CTOR(adder_method) {
    SC_METHOD(do_adder);
    sensitive << A << B;
  }
};
```

```
// adder_thread.h
#include "systemc.h"

SC_MODULE(adder_thread) {
  sc_in<int> A, B;
  sc_out<int> C;
  sc_in<bool> clk;

  void do_adder() {
    while (1) {
      C = A + B;
      wait();
    }
  }

  SC_CTOR(adder_thread) {
    SC_THREAD(do_adder);
    sensitive_pos(clk); // sensitive_pos
<< clk;
  }
};
```

SystemC Examples-1

```
// adder_thread.h
#include "systemc.h"

SC_MODULE(adder_thread2) {
  sc_in<int> A, B;
  sc_out<int> C;

  void do_adder() {
    while (1) {
      C = A + B;
      wait();
    }
  }

  SC_CTOR(adder_thread2) {
    SC_THREAD(do_adder);
    sensitive(A); sensitive(B);
  }
};
```

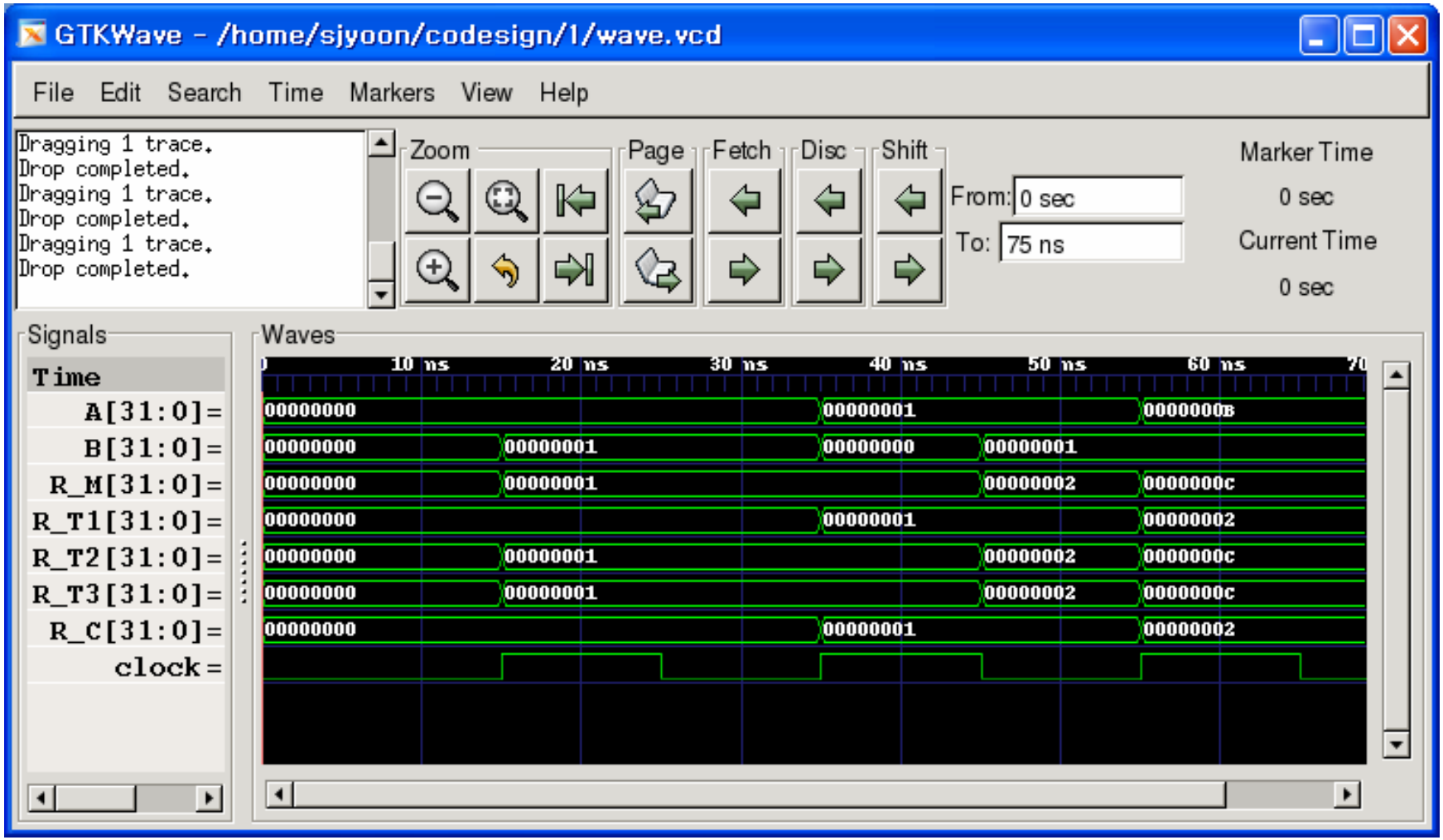
```
// adder_thread.h
#include "systemc.h"

SC_MODULE(adder_thread3) {
  sc_in<int> A, B;
  sc_out<int> C;
  sc_in<bool> clk;

  void do_adder() {
    while (1) {
      C = A + B;
      wait();
    }
  }

  SC_CTOR(adder_thread3) {
    SC_THREAD(do_adder);
    sensitive << A << B << clk.pos();
  }
};
```

codesign environment

SystemC Examples-1

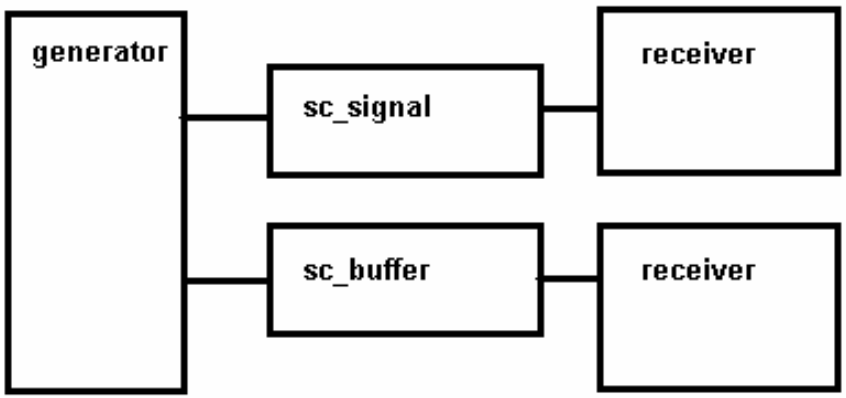
```
// adder_cthread.h
#include "systemc.h"

SC_MODULE(adder_cthread) {
  sc_in<int> A, B;
  sc_out<int> C;
  sc_in<bool> clk;

  void do_adder() {
    while (1) {
      C = A + B;
      wait();
    }
  }

  SC_CTOR(adder_cthread) {
    SC_CTHREAD(do_adder, clk.pos());
  }
};
```

SystemC Examples-1



SystemC Examples-2

- **Simple signal generator and receiver**



- **Compare sc_signal, sc_buffer**
  - Generator generates events: 5 (at 5), 6 (at 15), 6 (at 25), 7 (at 35), 8 (at 45)

SystemC Examples-2

```
// main.cpp
#include "systemc.h"

SC_MODULE (generator) {
  sc_out<short>  sig;
  sc_out<short>  buf;

  void do_it(void) {
    wait(5, SC_NS);
    sig.write(5); buf.write(5); wait(10, SC_NS);
    sig.write(6); buf.write(6); wait(10, SC_NS);
    sig.write(6); buf.write(6); wait(10, SC_NS);
    sig.write(7); buf.write(7); wait(10, SC_NS);
    sig.write(7); buf.write(7); wait(10, SC_NS);
  }

  SC_CTOR(generator) {
    SC_THREAD(do_it);
    sig.initialize(0);
    buf.initialize(0);
  }
};
```

```
SC_MODULE (receiver) {
  sc_in<short>   iport;

  void do_it(void) {
    cout << sc_time_stamp() << ":" << name()
<< " got " << iport.read() << endl;
  }

  SC_CTOR(receiver) {
    SC_METHOD(do_it);
    sensitive << iport;
    dont_initialize();
  }
};
```

SystemC Examples-2

```
int sc_main(int argc, char *argv[]) {
  sc_signal<short> sig;
  sc_buffer<short> buf;

  generator GEN("GEN");
  GEN.sig(sig); GEN.buf(buf);

  receiver  REV_SIG("REV_SIG");
  REV_SIG.iport(sig);
  receiver  REV_BUF_A("REV_BUF");
  REV_BUF.iport(buf);

  // trace file creation
  sc_trace_file *tf = sc_create_vcd_trace_file("wave");
  sc_trace(tf, sig, "sig"); sc_trace(tf, buf, "buf");

  sc_start();
  sc_close_vcd_trace_file(tf);
  return(0);
}
```

SystemC Examples-2

- **Result**

```
sjyoon@oddeye:~/codesign/2> ./run.x
           SystemC 2.1.v1 --- Jul  4 2007 10:23:41
       Copyright (c) 1996-2005 by all Contributors
                 ALL RIGHTS RESERVED
0 s:REV_BUF got 0
WARNING: Default time step is used for VCD tracing.
5 ns:REV_SIG got 5
5 ns:REV_BUF got 5
15 ns:REV_SIG got 6
15 ns:REV_BUF got 6
25 ns:REV_BUF got 6
35 ns:REV_SIG got 7
35 ns:REV_BUF got 7
45 ns:REV_BUF got 7
```