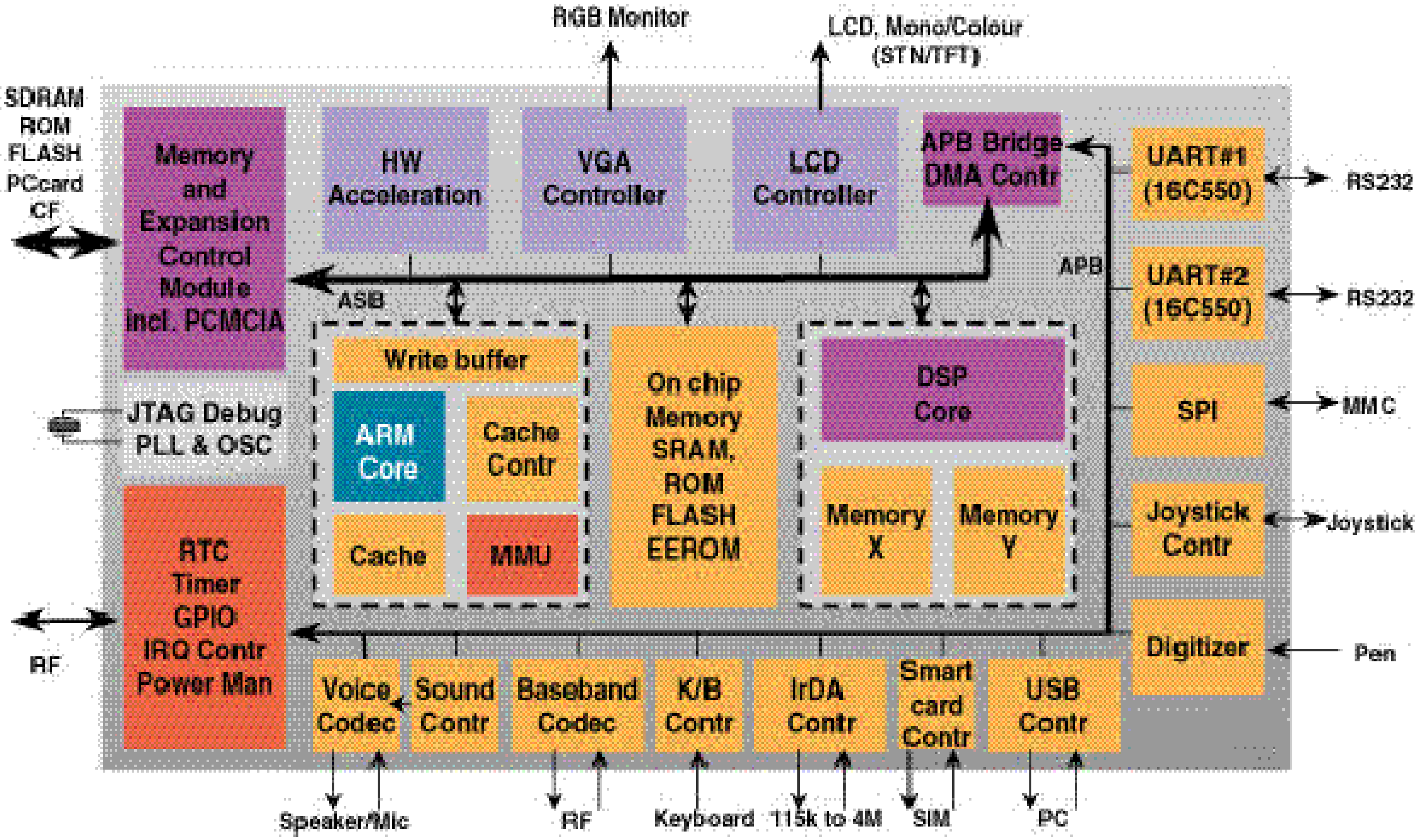
SystemC®

P. Marwedel*
Informatik 12, TU Dortmund

---

* Partially using slides prepared by Tatjana Stankovic from the University of Nis (Serbia and Montenegro), visiting the University of Dortmund under the TEMPUS program.

Motivation



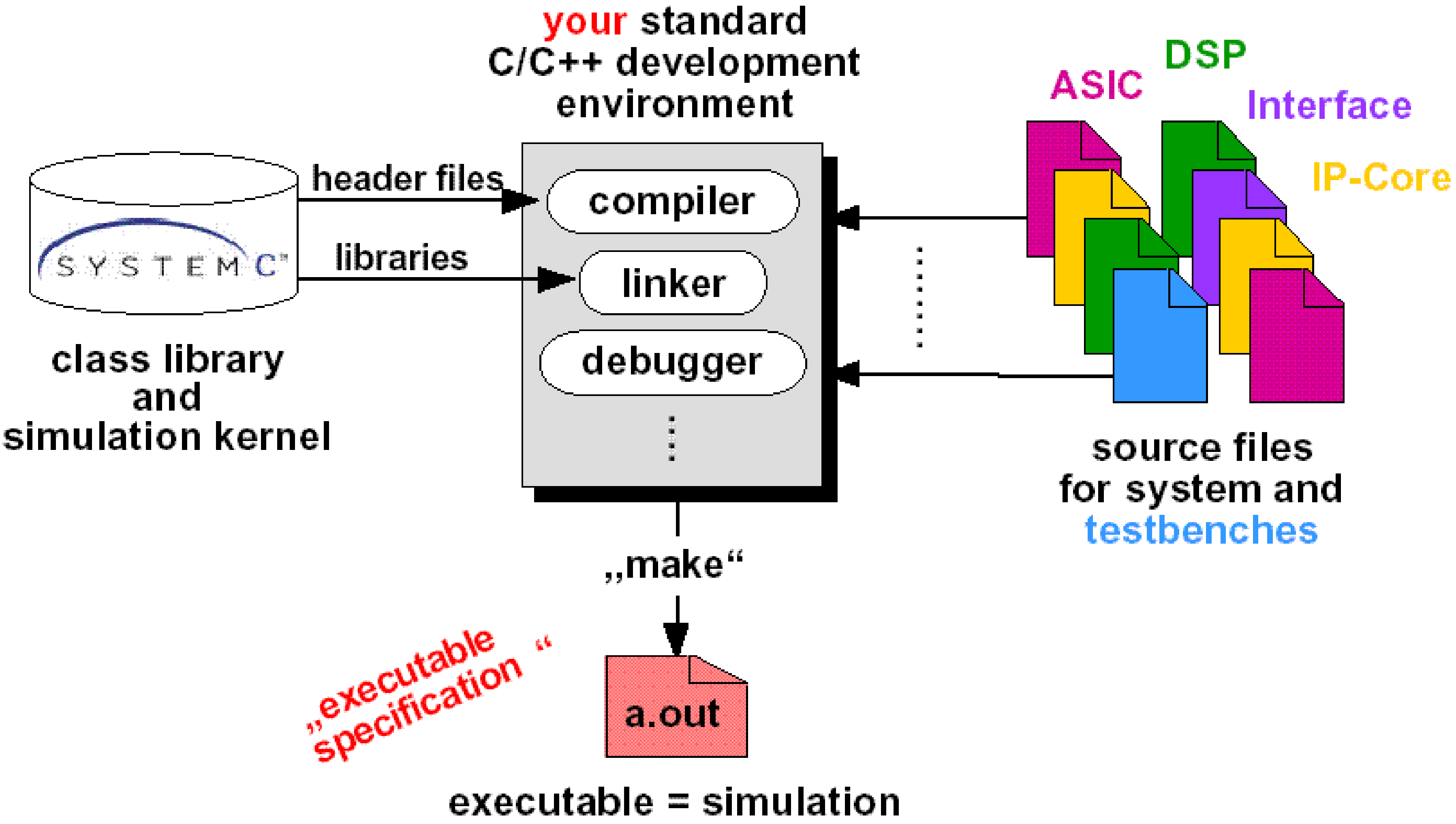
Today’s complex systems consist of HW & SW.

Single model both!

What is SystemC®?

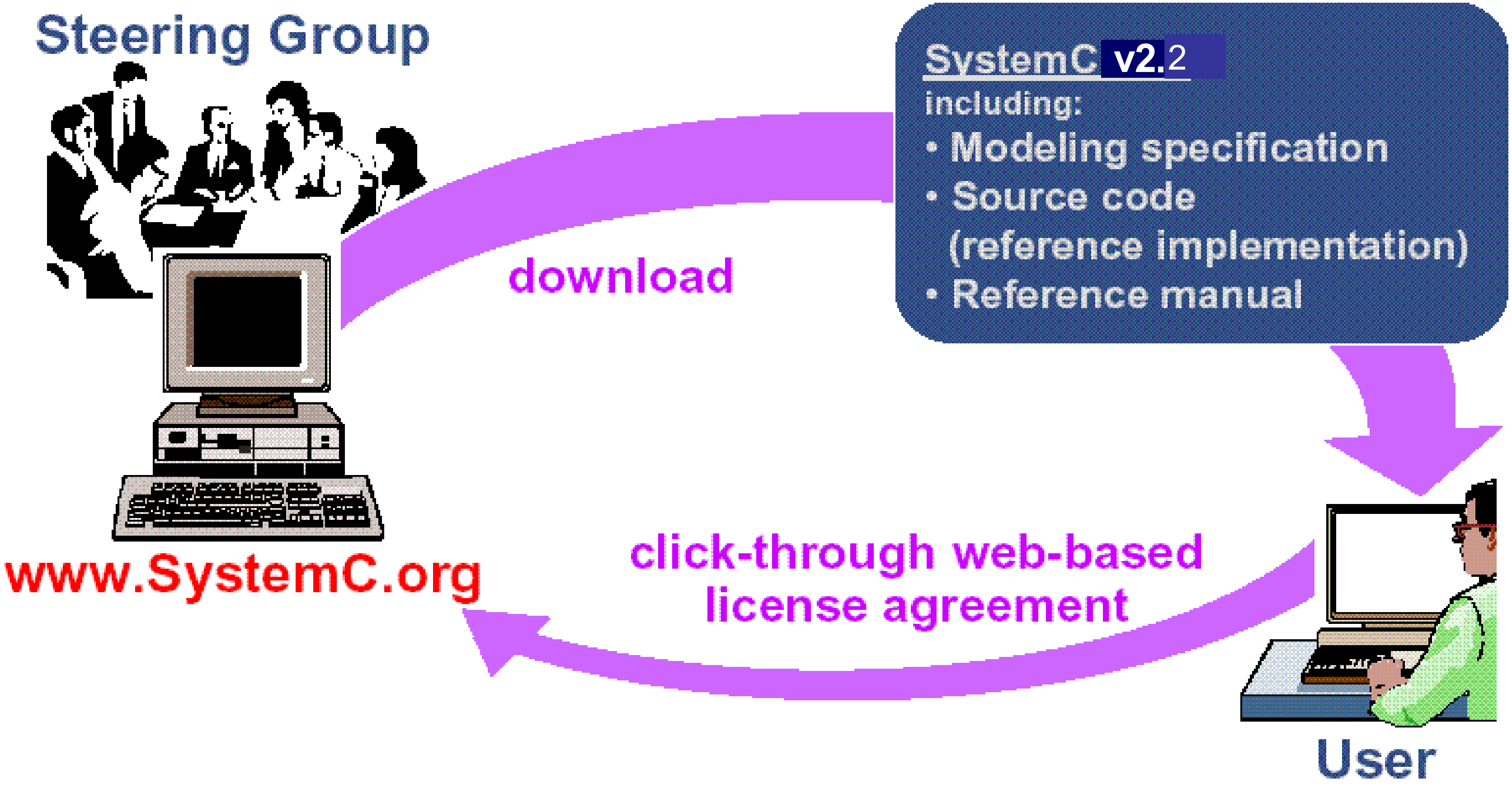
- Library-enhanced C++ intended to represent
  - functionality & communication,
  - software & hardware,
  - at multiple abstraction levels,
  - concurrency,
  - data & control
- Useful for cycle-accurate model of SW algorithms, HW architectures, and their interfaces.

SystemC® Design Methodology



Open Community Licensing

How to get SystemC ?



Steering Group

SystemC v2.2
including:
- Modeling specification
- Source code
  (reference implementation)
- Reference manual

download

www.SystemC.org

click-through web-based
license agreement

User

Drawbacks of a C/C++ Design Flow

- C/C++ is *not* created to design hardware !
- C/C++ does not support
  - Hardware style communication - Signals, protocols
  - Notion of time - Clocks, time sequenced operations
  - Concurrency - Hardware is inherently concurrent, operates in parallel
  - Reactivity - Hardware is inherently reactive, responds to stimuli, interacts with its environment (requires handling of exceptions)
  - Hardware data types - Bit type, bit-vector type, multi-valued logic types, signed and unsigned integer types, fixed-point types
- Missing links to hardware during debugging

References

- D. Black, J. Donovan: SystemC: From the Ground Up, *Springer*, 2004
- http://www.doulos.com/knowhow/systemc
- SystemC 2.2 Language Reference Manual
- SystemC™ Version 2.2 User's Guide, http://www.systemc.org
- Joachim Gerlach: System-on-Chip Design with SystemC, U. Tübingen, Wilhelm-Schickard-Institut, Department of Computer Engineering
- Stuart Swan: An Introduction to System Level Modeling in SystemC 2.0, Cadence Design Systems Inc., 2001
- Thorsten Grötker, Stan Liao, Grant Martin, Stuart Swan, System Design with SystemC™, *Kluwer Academic Publishers*

SystemC language architecture

Channels for MoCs
Kahn process networks, SDF, etc

Methodology-specific Channels
Master/Slave library

Elementary Channels
Signal, Timer, Mutex, Semaphore, FIFO, etc

Core Language
Module
Ports
Processes
Events
Interfaces
Channels
Event-driven simulation kernel

Data types
Bits and bit-vectors
Arbitrary precision integers
Fixed-point numbers
4-valued logic types, logic-vectors
C++ user defined types

C++ Language Standard

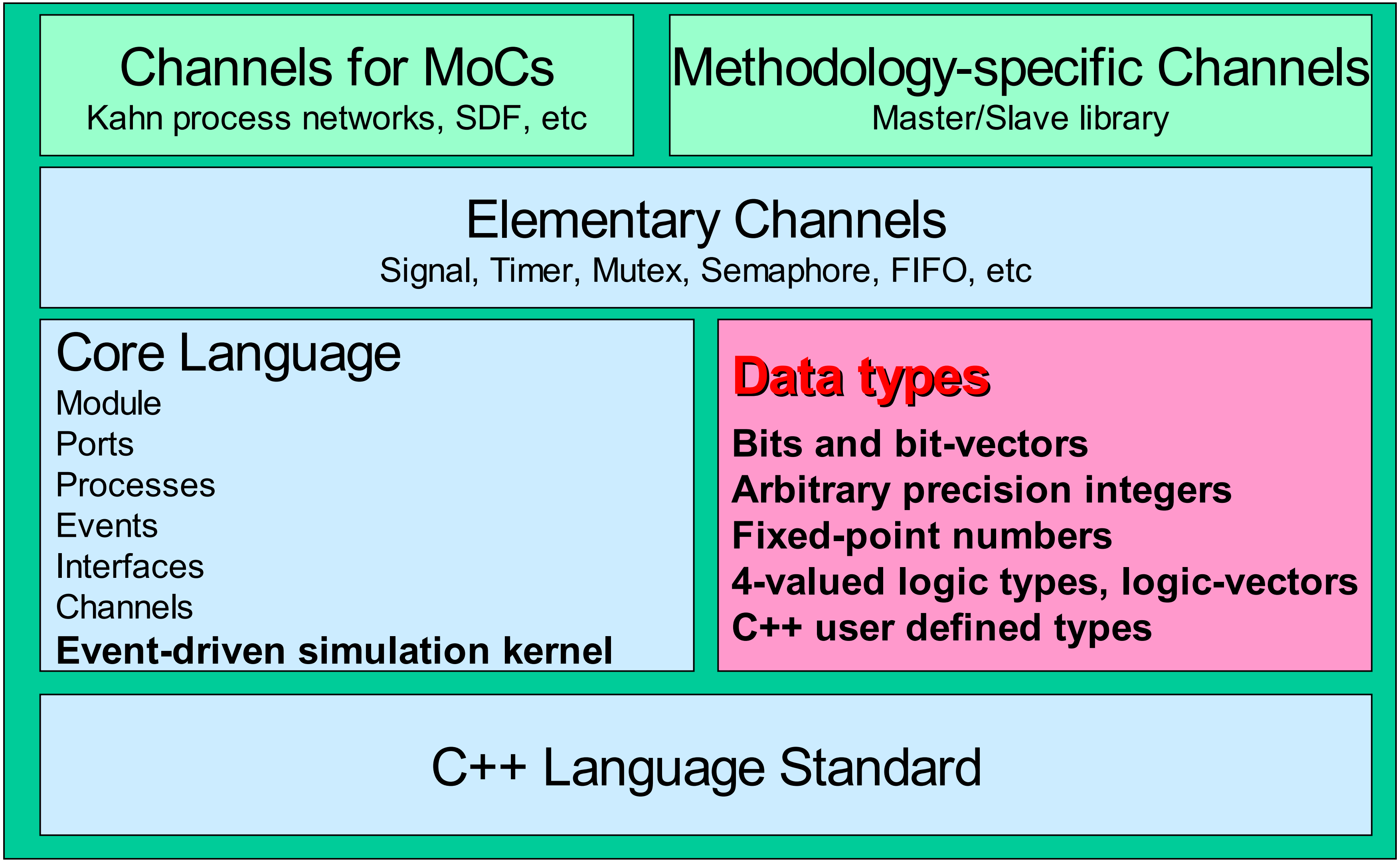
SystemC language architecture

- Upper layers built on top of the lower layers.
- Core language: minimal set of modeling constructs for structural description, concurrency, communication, and synchronization.
- Commonly used communication mechanisms such as signals and FIFOs can be built on top of the core language.
- Data types separate from the core language and user-defined data types are fully supported.
- Commonly used models of computation (MOCs) can also be built on top of the core language.
- If desired, lower layers within the diagram can be used without needing the upper layers.

Contents

- Introduction
- ➡ Data types
- A Notion of Time
- Modules
- Concurrency
- Structure
- Communication, Channels
- Ports & Interfaces
- Advanced Topics

SystemC language architecture



Channels for MoCs
Kahn process networks, SDF, etc

Methodology-specific Channels
Master/Slave library

Elementary Channels
Signal, Timer, Mutex, Semaphore, FIFO, etc

Core Language

- Module
- Ports
- Processes
- Events
- Interfaces
- Channels
- **Event-driven simulation kernel**

Data types

- **Bits and bit-vectors**
- **Arbitrary precision integers**
- **Fixed-point numbers**
- **4-valued logic types, logic-vectors**
- **C++ user defined types**

C++ Language Standard

Data Types

SystemC supports

- Native C/C++ types
- SystemC types
- User-defined types

SystemC types

- 2-valued (‘0’, ‘1’) logic / logic vector
- 4-valued (‘0’, ‘1’, ‘Z’, ‘X’) logic / logic vector
- Arbitrary sized integer (signed/unsigned)
- Fixed point types
  (signed/unsigned, templated/ untemplated)

Native C/C++ Data Types

Integer types:

- char
- unsigned char
- short
- unsigned short
- int
- unsigned int
- long
- unsigned long

Floating point types:

- float
- double
- long double

SystemC Data Types

- **sc_bit** – 2 valued single bit type
- **sc_logic** – 4 valued single bit type
- **sc_int** – 1 to 64 bit signed integer type
- **sc_uint** – 1 to 64 bit unsigned integer type
- **sc_bigint** – arbitrary sized signed integer type
- **sc_biguint** – arbitrary sized unsigned integer type
- **sc_bv** – arbitrary sized 2 valued vector type
- **sc_lv** – arbitrary sized 4 valued vector type
- **sc_fixed** – templated signed fixed point type
- **sc_ufixed** – templated unsigned fixed point type
- **sc_fix** – untemplated signed fixed point type
- **sc_ufix** – untemplated unsigned fixed point type

Type sc_bit

sc_bit is a 2-valued data type representing a single bit.
A variable of type **sc_bit** can have values
’0’(false) or ’1’(true) only.

sc_bit Operators

Bitwise	&(and)	|(or)	^(xor)	~(not)
Assignment	=	&=	|=	^=
Equality	==	!=		

Declaration of object of type **sc_bit:**

sc_bit s;

Type sc_logic

4 values, ’0’(false), ’1’(true), ’X’ (unknown), and ’Z’ (hi impedance or floating). Models designs with multi driver busses, X propagation, startup values, and floating busses.

sc_logic Operators : operators used for type **sc_bit** also available for **sc_logic**.

An example assignment:

```
sc_logic x;   // object declaration
x = ’1’;      // assign a 1 value
x = ’Z’;      // assign a Z value
```

Fixed Precision Unsigned and Signed Integers

SystemC integer type provides integers from 1 to 64 bits in signed and unsigned forms.

- **sc_int<n>** is a Fixed Precision Signed Integer type
- **sc_uint<n>** is a Fixed Precision Unsigned Integer type

Signed type represented in 2's complement.

Examples

sc_int<64> x; // declares a 64 bit signed integer

sc_uint<48> y; // declares a 48 bit unsigned integer

Fixed Precision Unsigned and Signed Integers

Fixed Precision Integer Operators

<table>
<tr><td>Bitwise</td><td>~ & | ^ >> <<</td></tr>
<tr><td>Arithmetic</td><td>+ - * / %</td></tr>
<tr><td>Assignment</td><td>= += -= *= /= %= &= |= ^=</td></tr>
<tr><td>Equality</td><td>== !=</td></tr>
<tr><td>Relational</td><td>< <= > >=</td></tr>
<tr><td>Autoincrement</td><td>++</td></tr>
<tr><td>Autodecrement</td><td>--</td></tr>
<tr><td>Bit Select</td><td>[x]</td></tr>
<tr><td>Part Select</td><td>range()</td></tr>
<tr><td>Concatenation</td><td>(,)</td></tr>
</table>

Arbitrary Precision Signed and Unsigned Integer Types

For operands ≥ 64 bits:

- **sc_biguint<n>** (arbitrary size unsigned integer) or
- **sc_bigint<n>** (arbitrary sized signed integer).

Works on integers of any size, limited only by underlying system limitations.

Operators for Fixed Precision Integers also available for Arbitrary Precision Integers.

Types **sc_biguint, sc_bigint, sc_int, sc_uint**, and C++ integer types can be mixed together in expressions. Operator = can be used for conversion between types.

Arbitrary Length Bit Vector

A 2-valued arbitrary length vector, i.e. **sc_bv** type used for large bit vector manipulation.

Arbitrary Length Bit Vector Operators

Bitwise	~	&	|	^	<< >>
Assignment	=	&=	|=	^=	
Equality	==	!=			
Bit Selection	[x]				
Part Selection	range()				
Concatenation	(,)				
Reduction	and_reduce()	or_reduce()	xor_reduce()		

Arbitrary Length Logic Vector

Type **sc_lv<n>** represents arbitrary length vector value, each bit can have one of four values.

Type **sc_lv<n>** a variable sized array of sc_logic objects.

Declaration Example:

```
sc_signal<sc_lv<64> > databus;   //a 64 bit wide signal
                                     //called databus
```

The same operations can be performed on **sc_lv** and **sc_bv**
sc_bv will simulate much faster than **sc_lv**.

sc_bv / sc_lv

Features:

- Assignment between **sc_bv** and **sc_lv**
- Use of string literals for vector constant assignments
- Conversions between **sc_bv/sc_lv** and SystemC integer types
- No arithmetic operation available

Fixed Point Types

DSP applications frequently require fixed point data types. SystemC contains signed & unsigned fixed point data types, used to accurately model hardware.

4 basic fixed point types:

- **sc_fixed:** static arguments (known at compile time)
- **sc_ufixed:** dto.
- **sc_fix:** variable arguments (configurable at runtime)
- **sc_ufix:** dto.

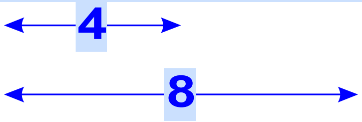
Additional "fast" versions **sc_fix_fast**, .. limited to 53 bits.

Require #**define SC_INCLUDE_FX** prior to #**include** <systemc.h>

Fixed Point Types

Example:

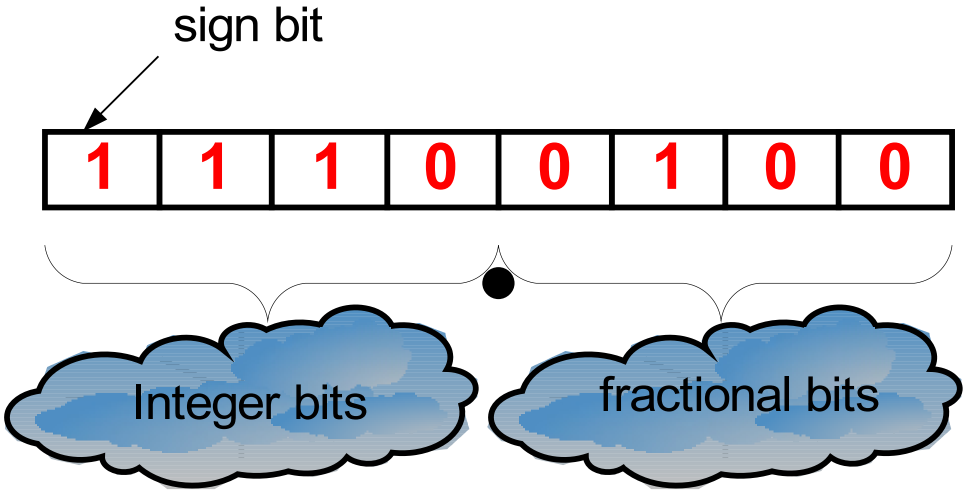
(1.75) = $(0001.1100)_2$



1's complement of $(0001.1100)_2 = (1110.0011)_2$

2's complement of $(0001.1100)_2 = (1110.0100)_2$

my_var:



Fixed Point Types

Syntax for declaration of fixed point object:

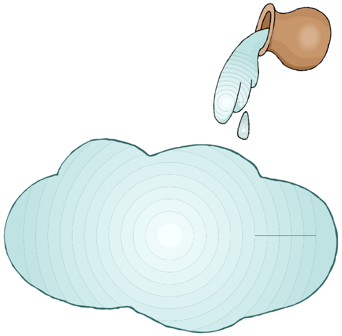
- sc_fixed<wl, iwl, q_mode, o_mode, n_bits> x;
- sc_ufixed<wl, iwl, q_mode, o_mode, n_bits> y;
- sc_fix x(list of options);
- sc_ufix y(list of options);

With

wl	Total word length
iwl	# of bits left of the binary point
q_mode	quantization mode
o_mode	overflow mode
n_bits	number of saturated bits
x,y	object name, name of the fixed point object

Meaning of options

Name	***Overflow meaning***
SC_SAT	Saturate
SC_WRAP	Wrap around
…	(other values)



Name	***Quantization Mode***
SC_RND	Round
SC_RND_ZERO	Round towards zero
SC_RND_MIN_INF	Round towards minus infinity
SC_RND_INF	Round towards infinity
SC_TRN	Truncate
…	(other values)

High Levels of Abstraction and the STL

STL container classes are available

- string
- vector
- map
- list
- deque

Also available

- for_each
- count
- min_element
- reverse
- sort
- search
- ….

Contents

- Introduction
- Data types
- ➡ A Notion of Time
- Modules
- Concurrency
- Structure
- Communication, Channels
- Ports & Interfaces
- Advanced Topics

Time

- SystemC uses an integer-valued absolute time model.
- Internal representation: unsigned integer, ≥ 64 bits.
- Starts at 0, & moves forward only.
- Type **sc_time** represents time or a time interval.
- Time objects are pairs (numeric value, time unit).

Time units

Unit	**Meaning**
SC_SEC	seconds
SC_MS	milliseconds
SC_US	microseconds
SC_NS	nanoseconds
SC_PS	picoseconds
SC_FS	femtoseconds

Declarations

Syntax:

sc_time *name* ..; // no initialization

sc_time *name*(*magnitude*, *timeunits*)…;

Examples:

sc_time t_PERIOD (5, **SC_NS**);

sc_time t_TIMEOUT(100, **SC_MS**);

Usage

Examples:

- t_MEASURE = (t_CURRENT – t_LAST_CLOCK);
- **if** (t_MEASURE > t_HOLD) { error ("setup violated") };
- **wait**(t_HOLD) // allowed within **SC_THREAD**
  // processes (see below)

sc_start()

sc_start() starts the simulation phase.

Optional argument of type **sc_time** limits simulation time.

Examples:

sc_start(); // no timeout

sc_start(t_TIMEOUT) // simulation timeout of t_TIMEOUT

Time display

- **sc_time_stamp**() returns current simulation time
- **sc_simulation_time**() returns current time as **double**
- Time can be displayed with the stream operator <<
  Examples:
  - **cout** << **sc_time_stamp**() << **endl;**
  - **std::cout** << " current time is " << t_TIMEOUT << **std::endl**

Time Resolution

- **Time resolution:** smallest amount of time that can be represented by all **sc_time** objects.
  Default resolution: 1 ps.
  - Default may be changed prior to other uses of **sc_time**: **sc_set_time_resolution**(*value*,*unit*)
  - User may ascertain current time resolution by calling function **sc_get_time_resolution()**
- **Default time unit:** unit used when only the numeric value is specified.
  - It may be changed prior to other uses of **sc_time**: **sc_set_default_time_unit**(*value*,*unit*)

Application

```
int sc_main …
  sc_set_time_resolution(1,SC_MS);
  sc_set_default_time_unit (1,SC_SEC);
  simple_process instance("instance");
  sc_start(7200, SC_SEC); // max 2 hours
  double t = sc_simulation_time();
  unsigned hours = int (t/3600.0);
  t-=3600.0*hours;
  unsigned minutes = int (t/60.0);
…
```

Source & ©: D. Black, J. Donovan: SystemC from the ground up, Springer, 2004

sc_main() Function

The **sc_main()** function is the entry point from the SystemC library to the user‘s code.

- Its prototype is:
  int sc_main(int argc, **char*** argv[] **);**
- The arguments argc and argv[] are the standard command-line arguments. They are passed to **sc_main()** from **main()** in the library.
- Body of **sc_main()** configures simulation variables (default time unit, time resolution, etc.), instantiates module hierarchy and channels, simulation start, clean-up and returning a status code.

sc_main() Function

Instantiation syntax:

module_type module_instance_name(“string_name“);

where:

- module_type is the module type (a class derived from **sc_module**)
- module_instance_name is the module instance name (object name)
- string_name is the string the module instance is initialized with

sc_main() Function

After a module is instantiated in **sc_main()**, binding of its ports to channels may occur.

Named port binding syntax:

`module_instance_name.port_name(channel_name);`

where:

- port_name is the instance name of the port being bound
- channel_name is the instance name of the channel to which the port is bound

Hello World in SystemC

```
//FILE:main.cpp
#include <Hello_SystemC.h>
int sc_main(int argc, char* argv[]) {
  const sc_time t_PERIOD(8,SC_NS);
  sc_clock clk("clk",t_PERIOD);
  Hello_SystemC iHello_SystemC
   ("iHello_SystemC");
  iHello_SystemC.clk_pi(clk);
  sc_start(10);
  return 0; }
```

```
//FILE:Hello_SystemC.cpp
#include <Hello_SystemC.h>
void Hello_SystemC::main_method(void)
 {std::cout << sc_time_stamp() <<
  " Hello world!" << std::endl;
}
```

```
#ifndef HELLO_SYSTEMC_H
#define HELLO_SYSTEMC_H
//FILE:Hello_SystemC.h
#include <systemc.h>
#include <iostream>
SC_MODULE(Hello_SystemC) {
  sc_in_clk clk_pi;
  void
Hello_SystemC::main_method(void);
  SC_CTOR(Hello_SystemC) {
    SC_METHOD(main_method);
    sensitive << clk_pi.neg();
    dont_initialize();
  } };
#endif
```

Source & ©: D. Black, J. Donovan, http://eklectically.com/Book/; All usage restrictions imposed by the authors apply.

Summary

- Need to describe SW & HW
- SystemC can model both.
- SystemC library makes C++ adequate for modeling of HW in a SW language.
- SystemC aims at higher levels of abstractions.
- No detailed modeling of gates.
- Data types supporting HW modeling available.
- Time values comprise a number and a unit.