SystemC Tutorial (II)

Original Slide by 蘇培陞 Alan P. Su, asu@itri.org.tw
Ver.2 by C. H. Chao, 2006. 9. 21, chihhao@access.ee.ntu.edu.tw
Ver.3 by Shao-Yi Chien, Feb 27, 2008
Ver.4 by Shao-Yi Chien, Feb 25, 2009
Ver.4.5 by Shao-Yi Chien, March 10, 2017

Contents

Chapter 7
Data Types

sc_bit

- 2 value single bit type
- ‘0’ (false) and ‘1’ (true)

sc_bit operators

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

- a = a & b;
- a = a | b;

a &= b;

a |= b;

sc_logic

- 4 value single bit type
- '0' (false), '1' (true), 'X' (unknown), 'Z' (high-impedance)

sc_logic operators

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

- x = '1';
- x ^= y;

x = 'Z';

x != y;

sc_int & sc_uint

- 1 to 64 bit fixed precision signed/unsigned integer type

Fixed precision integer operators

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

- mybit = myint[7]; myrange = myint.range(7,4);
- intc = (inta, intb);

sc_bigint & sc_biguint

- Arbitrary size signed/unsigned integer type

Fixed precision integer operators

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

```
sc_biguint<128> b1;      sc_biguint<64> b2;      sc_biguint<150> b3;
b3 = b1*b2;      // 42 bits will be truncated before assign into b3
```

sc_bv

- Arbitrary sized 2 value vector type

Fixed precision integer operators

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

```
sc_bv<64> databus;          sc_logic result;
result = databus.or_reduce();
// or_reduce returns the result of ORing all bits in databus
// and_reduce(), or_reduce() and xor_reduce() return bool value
```

sc_lv

- Arbitrary sized 4 value vector type

Fixed precision integer operators

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

```
sc_lv<16> bus1;
Bus1 = “ZZZZZZZZZZZZZZZ”;
// this string like assignment also works for sc_bv
```

Choose the Right Data Type

- Choose a data type that is closest to native C++ as possible for the modeling needs at hand

Fastest	Native C/C++ Data Types (e.g., `int`, `double` and `bool`)
	`sc_int<>, sc_uint<>`
	`sc_bit, sc_bv<>`
	`sc_logic, sc_lv<>`
	`sc_bigint<>, sc_biguint<>`
	`sc_fixed_fast<>, sc_fix_fast,` `sc_ufixed_fast<>, sc_ufix_fast`
Slowest	`sc_fixed<>, sc_fix, sc_ufixed<>, sc_ufix`

Chapter 8
Fixed Point Types

Fixed Point Types Basics

- Total word length (**wl**): the total number of bits. Word length must be greater than 0.
- Integer word length (**iwl**): the integer fraction bit length. iwl can be positive or negative, and can larger than wl.
  - xxx.xx : wl = 5, iwl = 3
  - .ssxxxxx : wl = 5, iwl = -2
  - xxxxx00 : wl = 5, iwl = 7
- For above examples, notice carefully the position of the decimal point

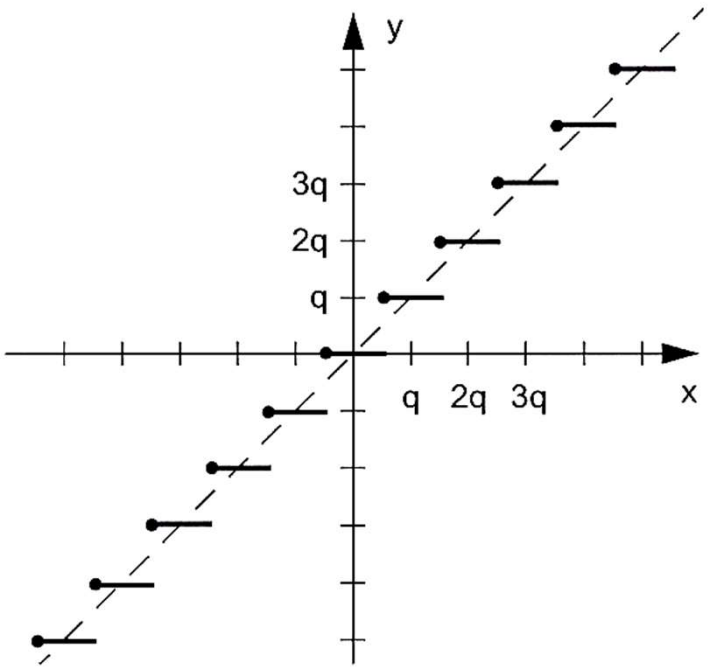
Quantization Mode

- How to handle the situation when the result of an operation generates more precision in the LSB than is available as specified by wl and iwl. For example, a division, 1/3.

Quantization Mode	Name
Round to plus infinity	SC_RND
Round to zero	SC_RND_ZERO
Round to minus infinity	SC_RND_MIN_INF
Round to infinity	SC_RND_INF
Convergent round	SC_RND_CONV
Truncate	SC_TRN (default)
Truncate to zero	SC_TRN_ZERO

SC_RND

- Round the value to the closest representable number by adding the MSB of the removed bits to the remaining bits. (二進位之四捨五入)
  - 01001.010[10100] → 01001.011



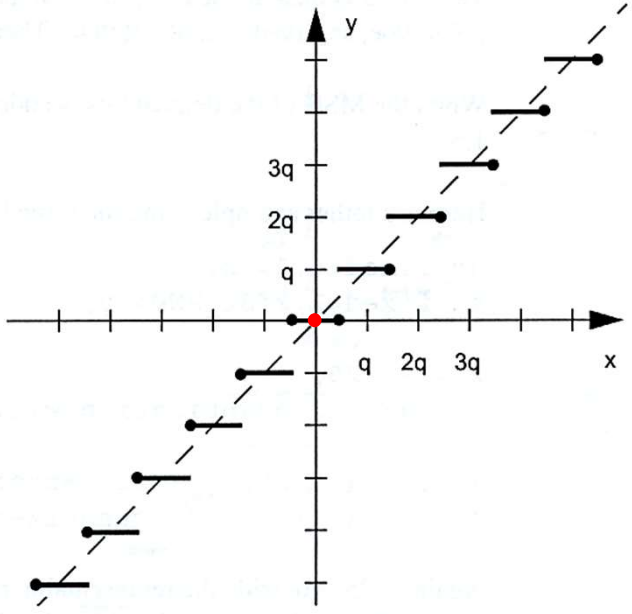
SC_RND Examples

```
sc_fixed<4, 2> x;
sc_fixed<3, 2, SC_RND> y;
x = 1.25; // 1.25 = 01.01
y = x;      // 01.0[1] → 01.1 = 1.5

x = -1.25 // -1.25 = 10.11 (01.01 2's complement)
y = x;      // 10.1[1] → 11.0 = -1
```

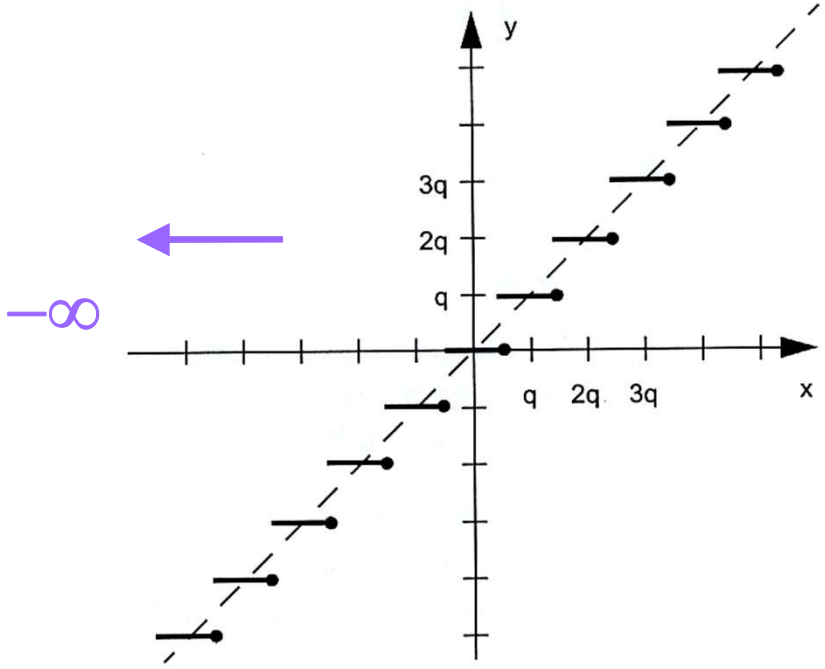
SC_RND_ZERO

- Performs SC_RND if the two nearest representable numbers are not an equal distance apart, otherwise round to zero is performed.
- x.x is not of equal distance. .xx or xx. Are of equal distance
- When round to zero, for a positive number redundant bits are deleted, and for a negative number SC_RND is performed.



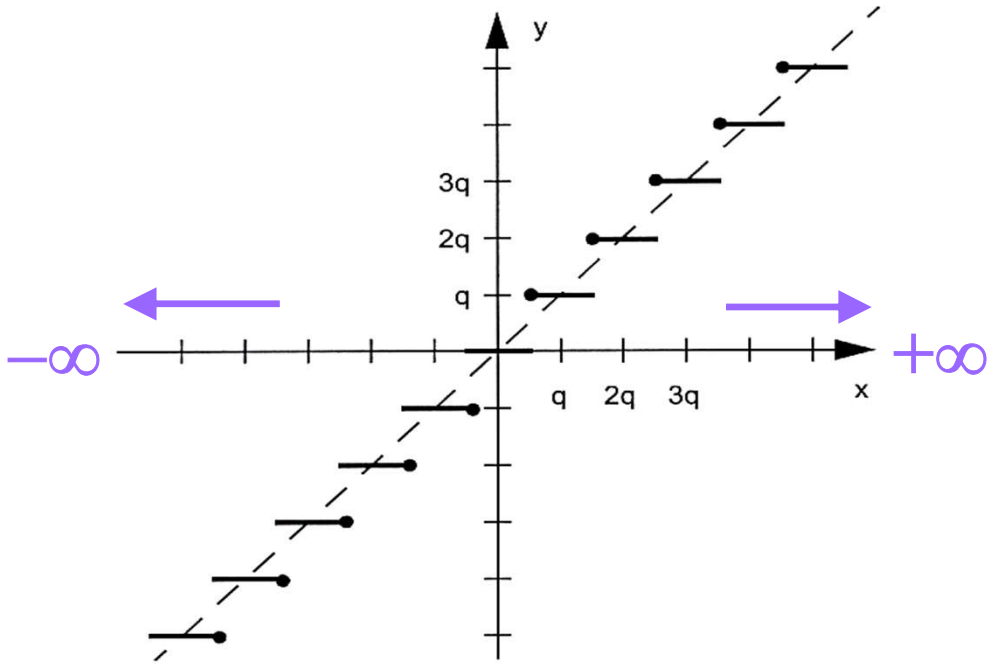
SC_RND_MIN_INF

- Performs SC_RND if the two nearest representable numbers are not an equal distance apart, otherwise round to minus infinity is performed.
- Minus infinity: for negative numbers perform SC_RND. For positive numbers eliminate the redundant bits.



SC_RND_INF

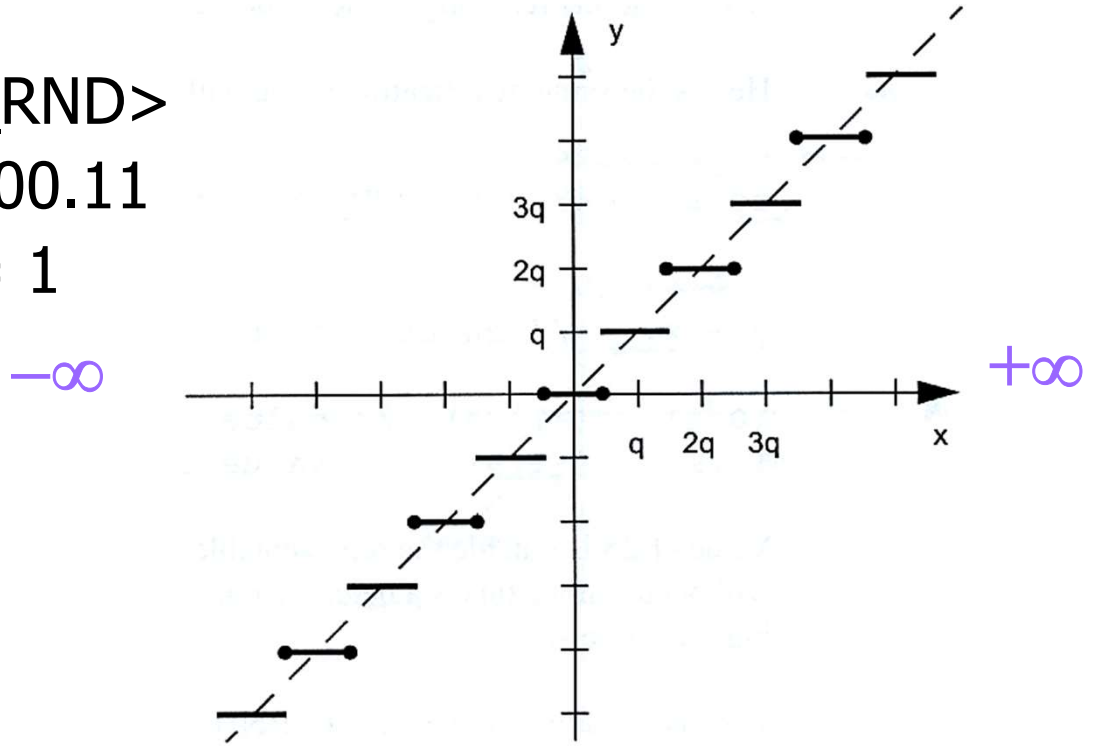
- Performs SC_RND if the two nearest representable numbers are not an equal distance apart, otherwise round to plus infinity is performed.
- Plus infinity: for positive numbers perform SC_RND. For negative numbers eliminate the redundant bits.



SC_RND_CONV

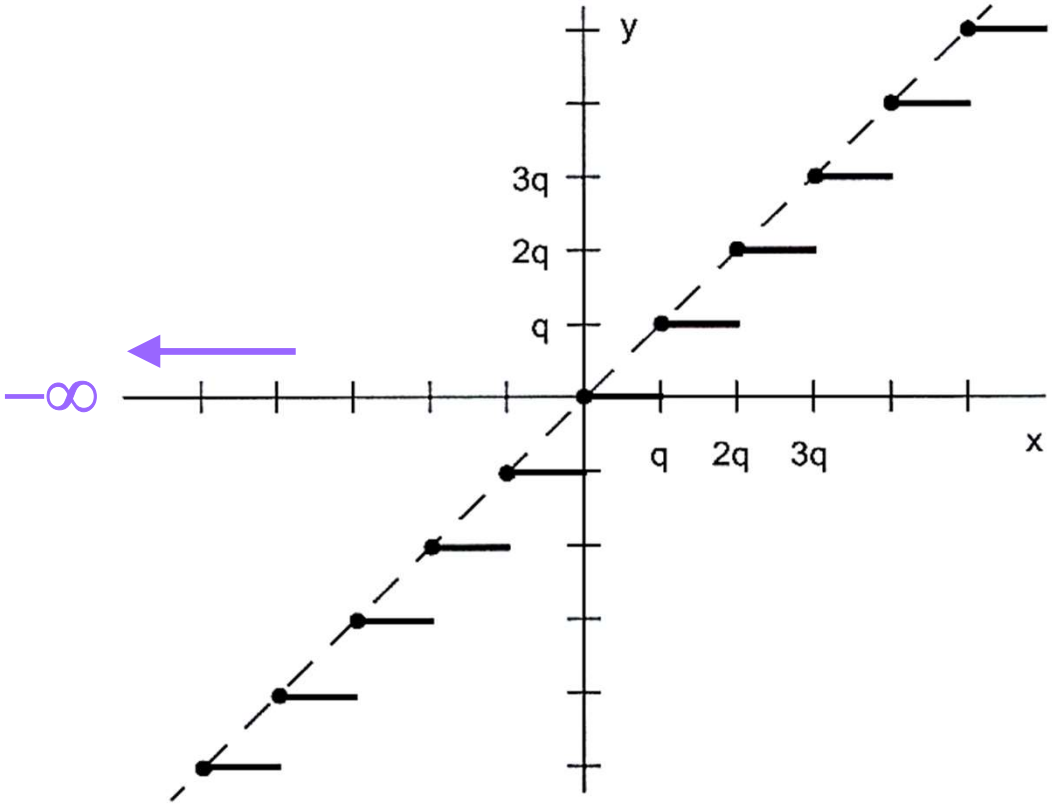
- Performs SC_RND if the two nearest representable numbers are not an equal distance apart, otherwise checks the LSB of the remaining bits. If the LSB is 1 then round towards plus infinity. If the LSB is 0 then round towards minus infinity.

```
sc_fixed<4, 2> x;
sc_fixed<3, 2, SC_RND>
x = .75;   // .75 = 00.11
y = x;     // 01.0 = 1
```



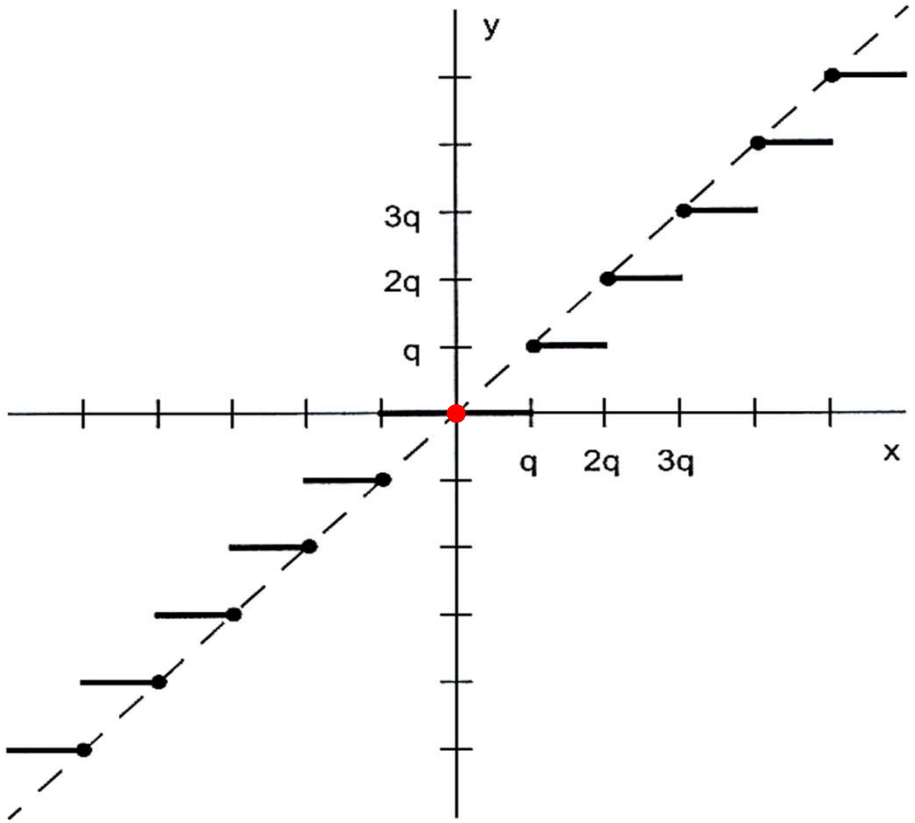
SC_TRN

- Default mode. The redundant bits are always deleted. Other ways of saying is it always rounds towards minus infinity.



SC_TRN_ZERO

- Performs truncate to positive numbers.
- Performs SC_RND to negative numbers.



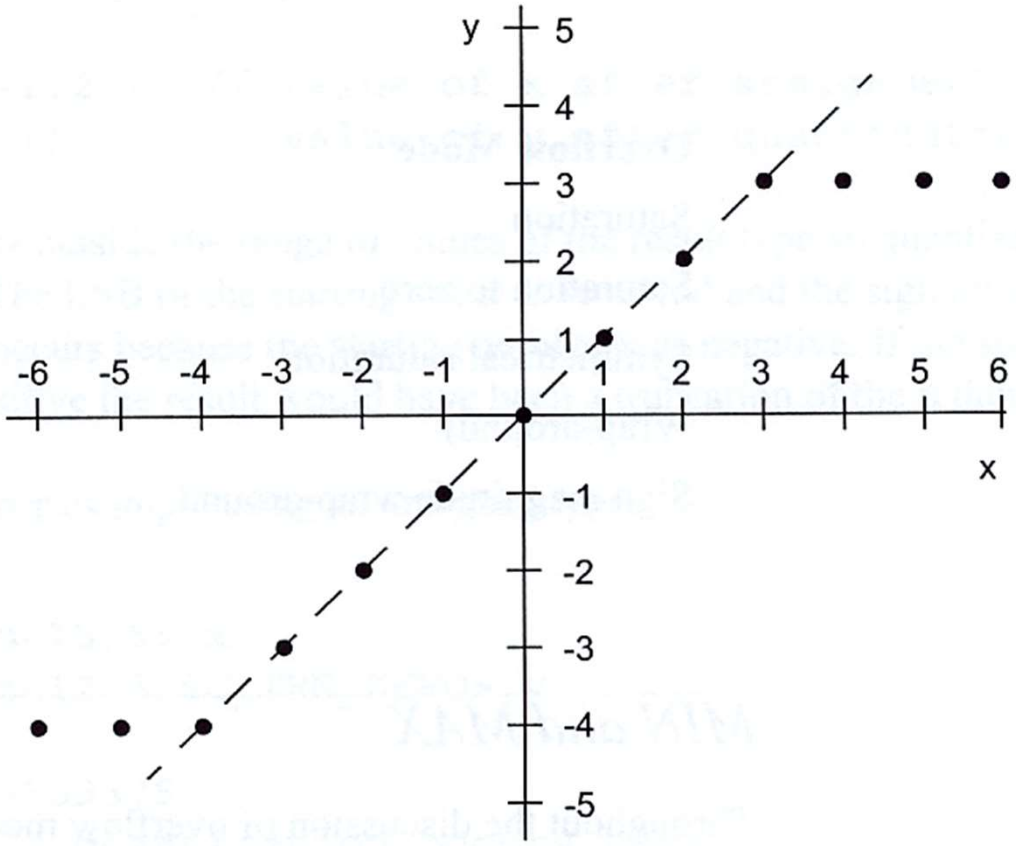
Overflow Mode

- Overflow: when the result of an operation generated more bits on the MSB side than are available for representation.
- Overflow mode is specified by the o_mode and n_bits parameters to a fixed point data type, it has two major modes, saturate and wrap-around.
- MIN is the smallest negative number that can be represented.
- MAX is the largest positive number that can be represented.

Overflow Mode	Name
Saturate	SC_SAT (default)
Saturate to zero	SC_SAT_ZERO
Symmetrical saturate	SC_SAT_SYM
Wrap-around	SC_WRAP
Sign magnitude wrap-around	SC_WRAP_SM

SC_SAT

- Converts the value to MAX for an overflow and MIN for an underflow.



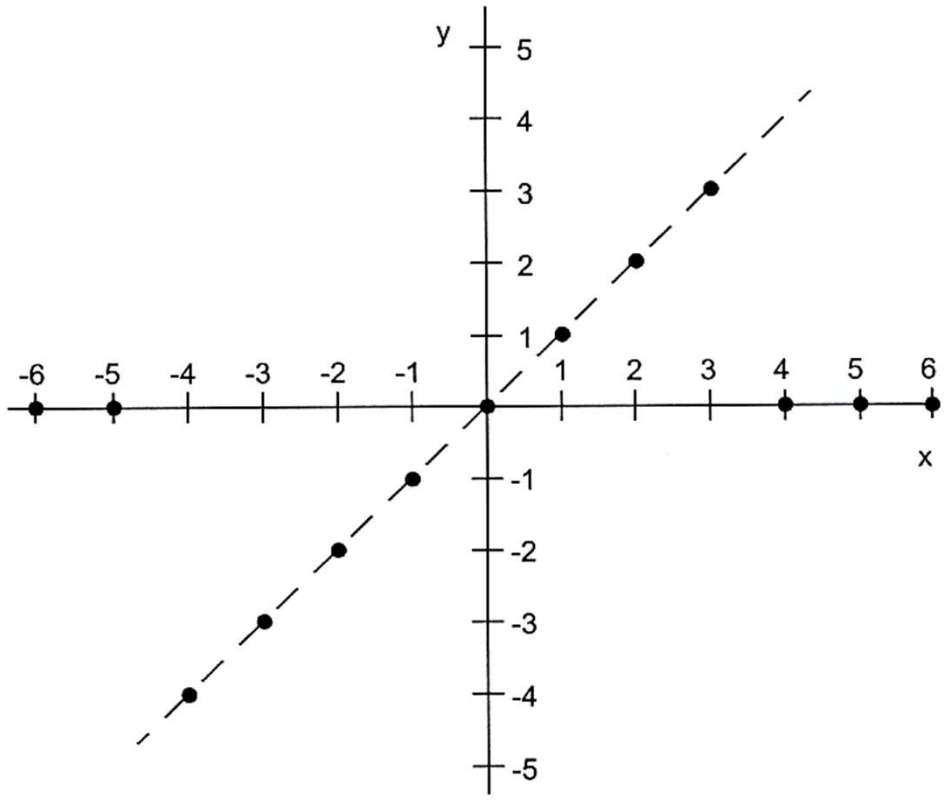
SC_SAT Examples

```
sc_fixed<4,4> x;
sc_fixed<3,3,SC_TRN,SC_SAT> y;
x = 6; // 0110, with a sign bit
y = x; // 011 = 3, remember the sign bit

x= -5; // 1011
y = x; // 100 = -4
```

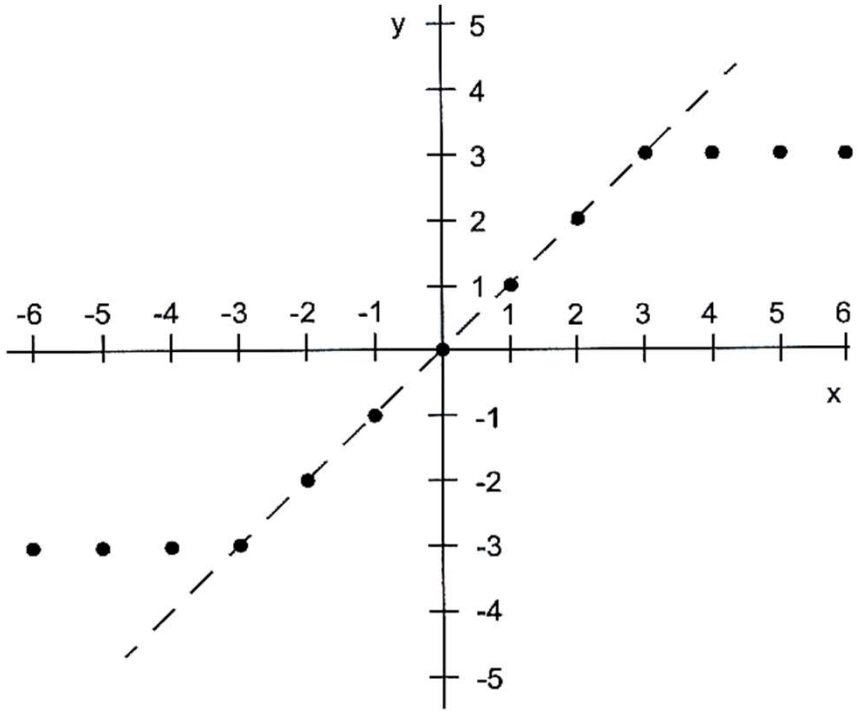
SC_SAT_ZERO

- Sets the result to 0 for any value that is outside the representable range of the fixed point data type.



SC_SAT_SYM

- In 2's complement notation there is one more negative representation than positive. Sometimes it is desirable to have the same positive and negative numbers of representations. SC_SAT_SYM generates MAX and negative overflow generates –MAX for signed numbers.

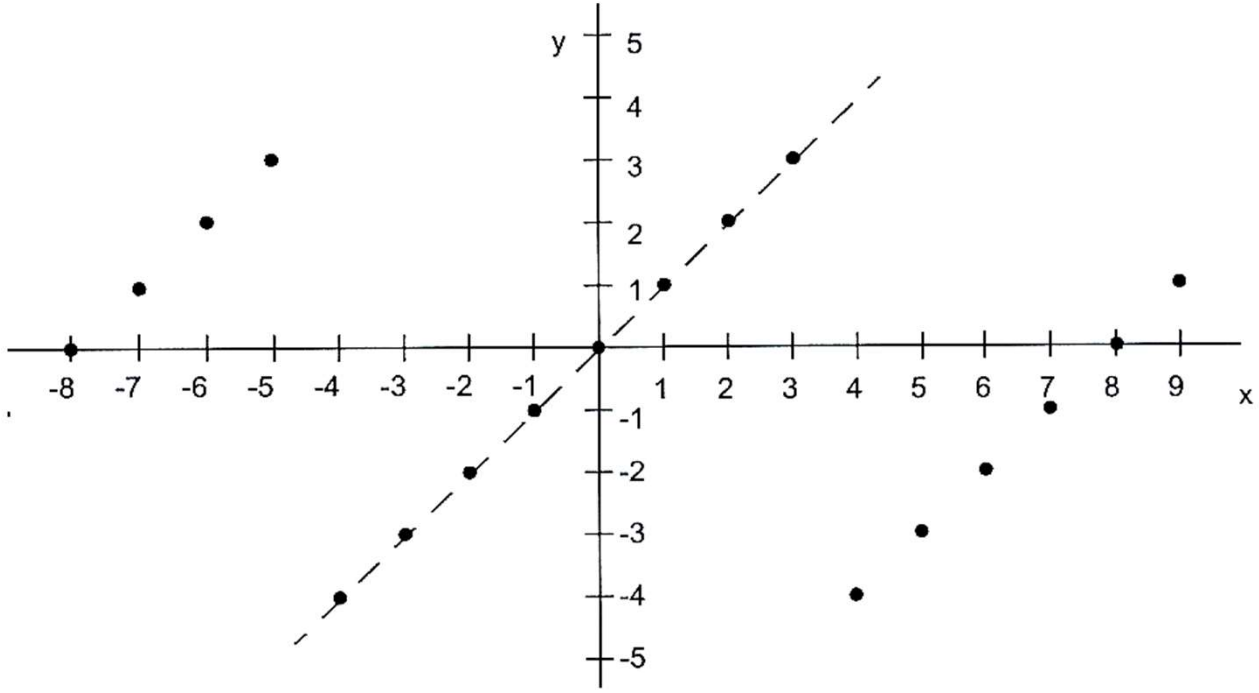


Wrap-around

- When overflow occurs, this operation wraps around the overflowed value from MAX to MIN. The unsigned case is similar to the way a counter would work in hardware.
- There are two different cases within the wrap-around operation: n_bits parameter set to 0 or is greater than 0. The n_bits parameter sets the number of bits to be wrapped around.

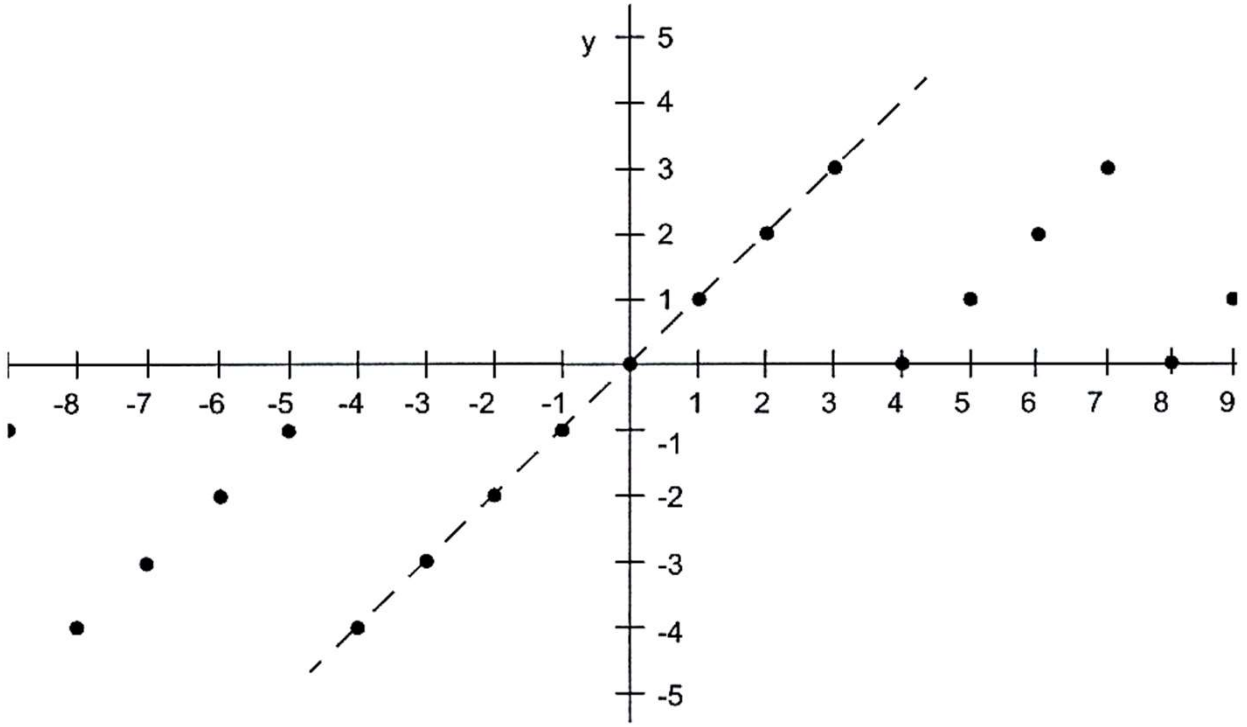
SC_WRAP, n_bits = 0

- The default overflow mode.
- Any MSB bits outside the range of the target type are deleted.



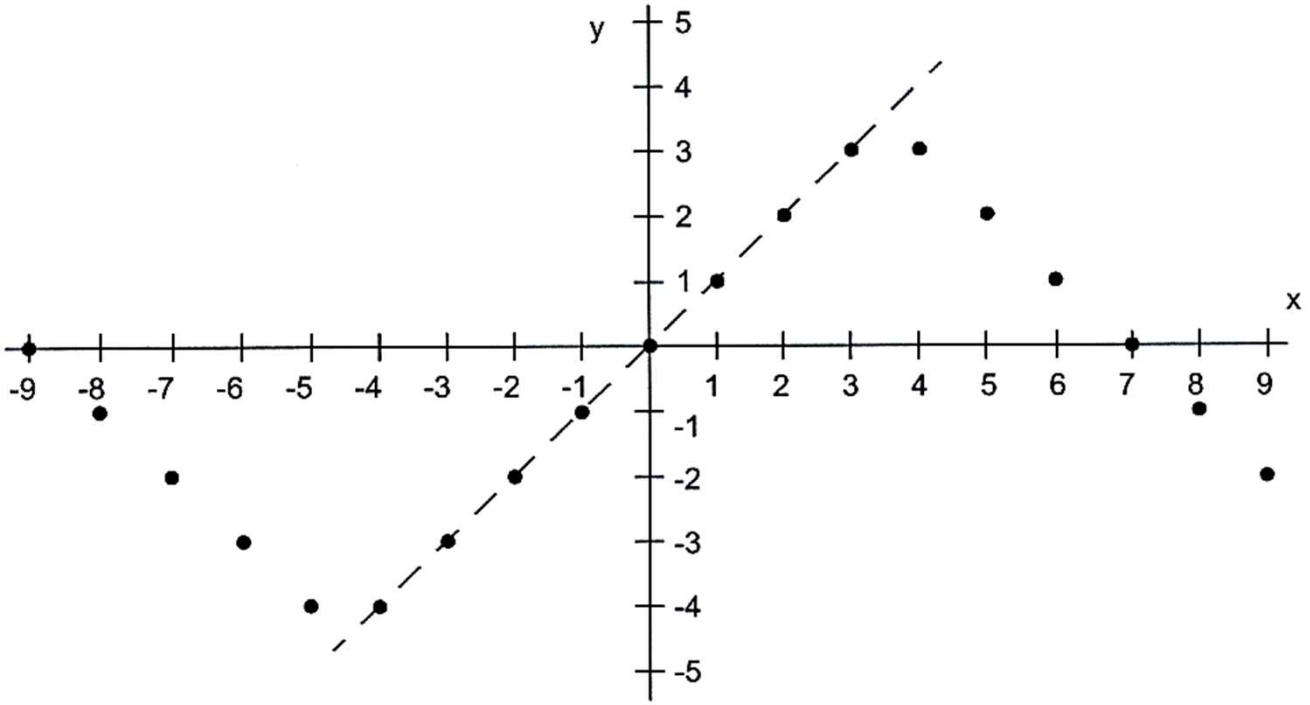
SC_WRAP, n_bits > 0

- n_bits MSB bits are to be saturated. The sign bit is retained. The bits that are not saturated are maintained.



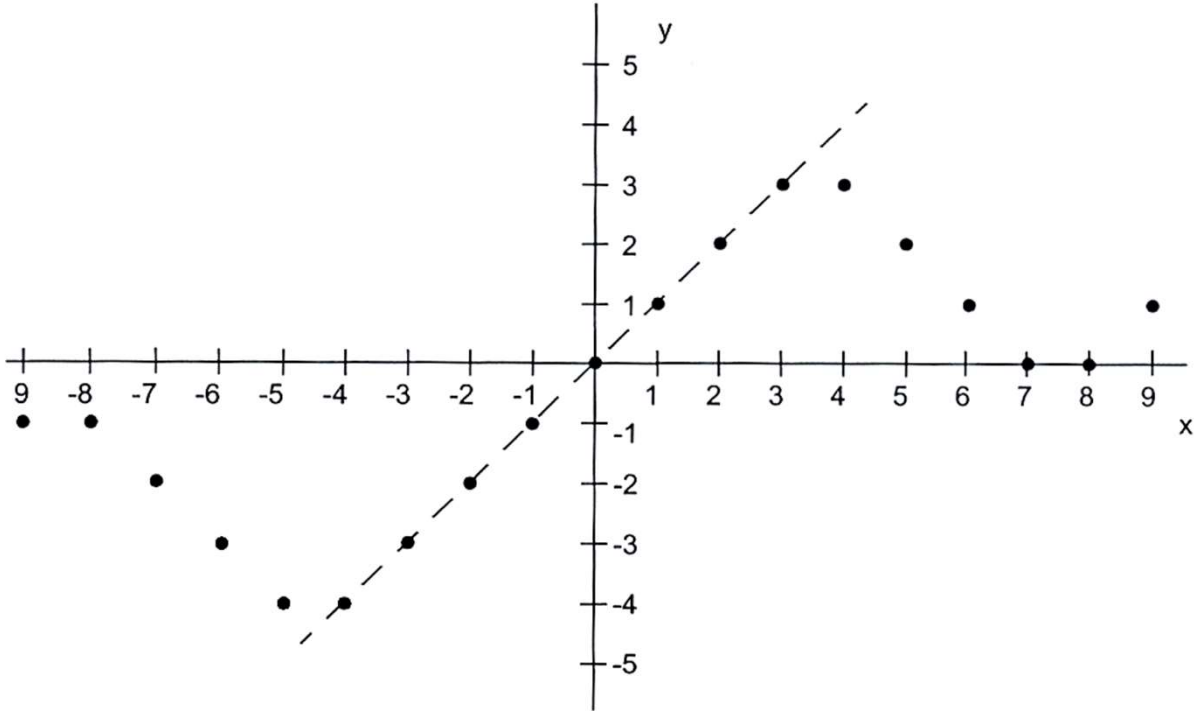
SC_WRAP_SM, n_bits = 0

- Deletes any MSB bits that are outside the result work length.
- Sign bit is set to the value of the LSB of deleted bits.



SC_WRAP_SM, n_bits > 0

- A sign magnitude wrap is performed and n-bits MSB bits will be saturated.
- The first n_bits MSB bits are saturated to MAX for positive numbers and to MIN for negative numbers.



sc_fixed & sc_ufixed

- Signed and unsigned fixed point data type
- Use static way to set parameters
- Declaration Syntax:

  sc_fixed <wl, iwl [,q_mode [, o_mode [, n_bits]]]>
  var_name([init_val][,cast_switch] [,observer]);

- Examples

  sc_ufixed<16, 1, SC_RND_CONV, SC_SAT_SYM, 1> a(1.5);

sc_fix & sc_ufix

- Declaration syntax:

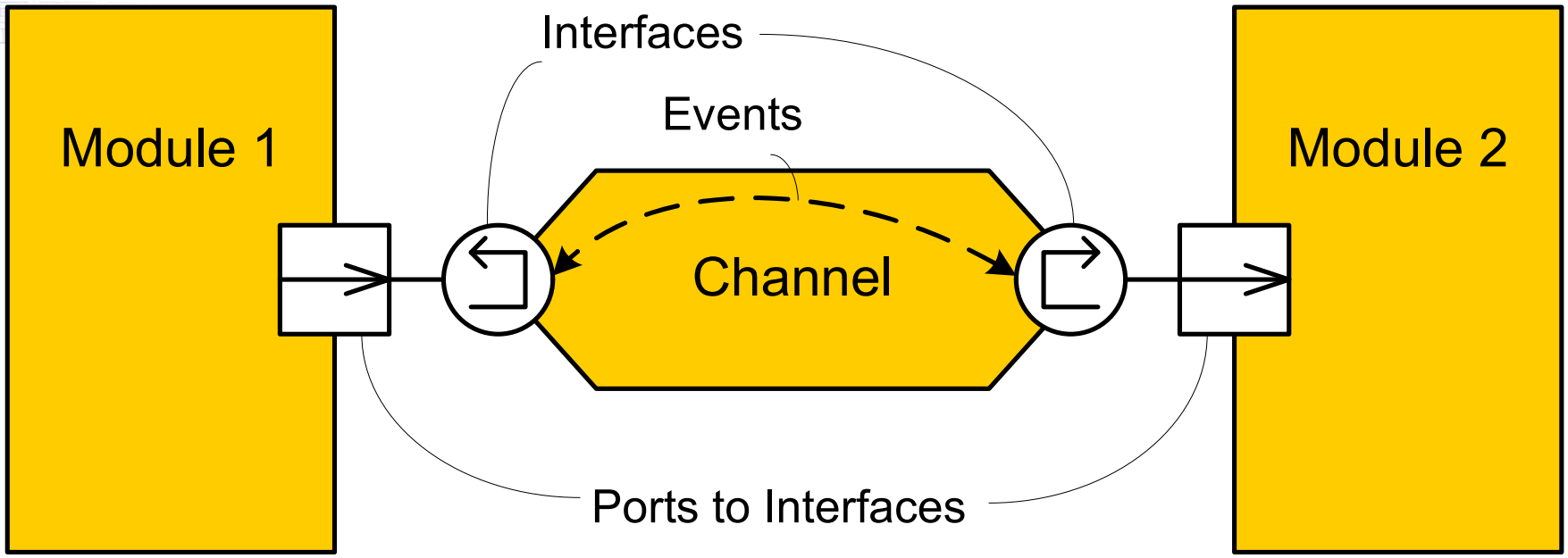
```
sc_fix var_name([init_val]
                [,wl,iwl]
                [,q_mode,o_mode[,n_bits]]
                [,cast_switch]
                [,observer]);
sc_fix var_name([init_val]
                ,type_params
                [,cast_switch]
                [,observer]);
```

- Example

```
sc_ufix a(1.5, 16, 1, SC_RND_CONV,SC_SAT_SYM, 1);
```

Chapter 9
Basic Channels and Interfaces

What is Interface & Channel?



Interface

- An interface defines the set of access functions (methods) of a channel. It specifies the name, parameters and return type of functions but does not have the implementation details.
  - sc_fifo_in_if
  - sc_fifo_out_if
  - sc_mutex_if
  - sc_semaphore_if
  - sc_signal_in_if
  - sc_signal_inout_if

Channel

- A channel implements the interface functions. Channel provides the communication between methods or within a module provides the communication between processes.
- Channel may implement one or more interfaces
- Different channels may implement the same interface in different ways
  - sc_buffer
  - sc_fifo
  - sc_mutex
  - sc_semaphore
  - sc_signal
  - sc_signal_resolved
  - sc_signal_rv

Basic Channels

- sc_mutex
- sc_semaphore
- sc_fifo
- sc_signal
- sc_buffer

sc_mutex

- Mutual exclusion object, let multiple program threads share a common resource without colliding

```
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();
```

Example of sc_mutex

```
class bus {
  sc_mutex bus_access;
  …
  void write(int addr,  int data) {
    bus_access.lock();
    // perform write
    bus_access.unlock();
  }
  …
};//endclass
```

sc_semaphore

- Have more than one copy of resources

```
sc_semaphore NAME(COUNT);

NAME.wait();          // Lock one semaphore
                      // Wait until available if in use
int NAME.trywait()    // Non-blocking, return success

int NAME.get_value() // Returns available semaphores

NAME.post();          // Free one previously locked
                      // semaphore
```

Example of sc_semaphore

```
class multiport_RAM {
  sc_semaphore read_ports(3);
  sc_semaphore write_ports(2);
  …
  void read(int addr, int& data) {
    read_ports.wait();
    // perform read
    read_ports.post();
  }
  void write(int addr, const int& data) {
    write_ports.wait();
    // perform write
    write_ports.post();
  }
  …
};//endclass
```

sc_fifo

- Probably the most popular channel for modeling at the architectural level

```
sc_fifo<ELEMENT_TYPENAME>  NAME(SIZE);

NAME.write  (VALUE);
NAME.read (REFERENCE);
... = NAME.read ()  /* function style */
if (NAME.nb_read (REFERENCE)) { // Non-blocking
                                // true if success
  ...
}
if (NAME.num_available() == 0)
  wait(NAME.data_written_event());
if (NAME.num_free() == 0)
  next_trigger(NAME.data_read_event());
```

If the fifo is empty, suspends until a value has been written to the fifo

If the FIFO is empty, returns immediately.

Example of sc_fifo

```
SC_MODULE(kahn_ex) {
  …
  sc_fifo<double> a, b, y;
  …
};
// Constructor
kahn_ex::kahn_ex() : a(10), b(10), y(20)
{
  …
}
void kahn_ex::addsub_thread() {
  for(;;) {
    y.write(kA*a.read() + kB*b.read());
    y.write(kA*a.read() - kB*b.read());
  }//endforever
}
```

sc_signal

- Like *reg* in Verilog

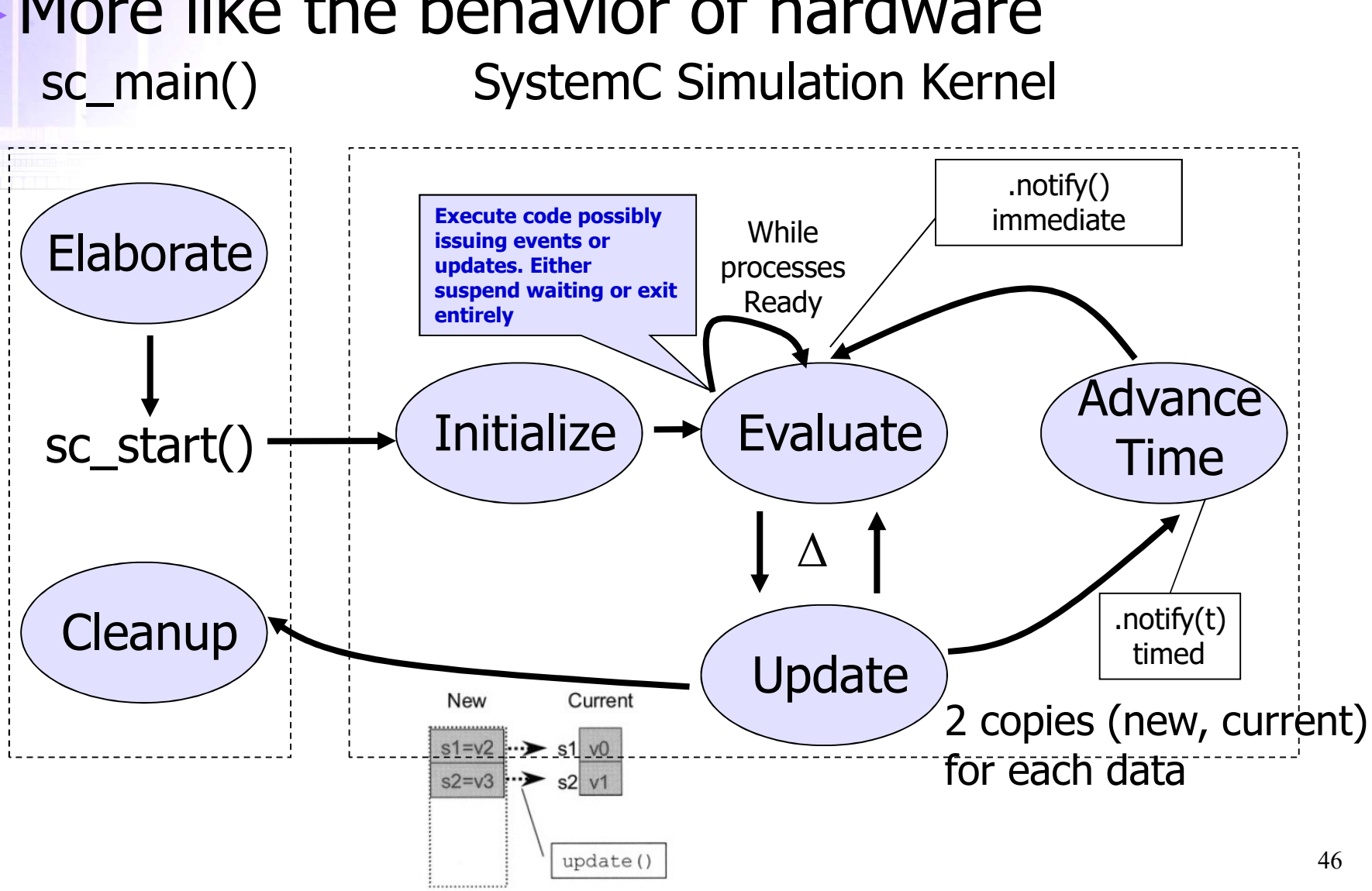
```
sc_signal<datatype> signame[, signame_i]...;
signame.write(newvalue);
signame.read(varname);
sensitive << signame.default_event();
wait(signame.default_event()|...);
if (signame.event() == true) {
  // occurred in previous delta-cycle
```

- sc_signal<bool> and sc_signal<sc_logic> have the following extensions:

```
sensitive << signame.posedge_event()
          << signame.negedge_event();
wait(signame.posedge_event()
     |signame.negedge_event());
if (signame.posedge_event()
    |signame.negedge_event()) {
```

Evaluation-Update Channels

- More like the behavior of hardware



Evaluate Phase

1. From the set of processes that are ready to run, select a process and resume its execution. The order in which processes are selected for execution from the set of processes that are ready to run is unspecified.
2. The execution of a process may include calls to the request_update() function which schedules pending calls to update() function in the update phase. The request_update() function may only be called inside member functions of a primitive channel.
3. Repeat evaluation for any other processes that are ready to run - synchronization

Update Phase

1. Execute any pending calls to update() from calls to the request_update() function executed in the evaluate phase.
2. If there are pending delta-delay notifications, determine which processes are ready to run and go to evaluation phase.
3. If there are no more timed event notifications, the simulation is finished.
4. Else, advance the current simulation time to the time of the earliest (next) pending timed event notification.
5. Determine which processes become ready to run due to the events that have pending notifications at the current time. Go to evaluation phase.

Chapter 10
Structure

Structure

- Hierarchical design is necessary to conquer complex systems
- There are many ways to build structure/hierarchy in SystemC
  - Direct top-level
  - Indirect top-level
  - Direct sub-module header-only
  - Indirect sub-module header-only
  - Direct sub-module
  - Indirect sub-module

Direct Top-Level

```
//FILE: main.cpp
  #include "Wheel.h"
  int sc_main(int argc, char* argv[]) {
    Wheel wheel_FL("wheel_FL");
    Wheel wheel_FR("wheel_FR");
    sc_start();
  }
```

Indirect Top-Level

```
//FILE: main.cpp
  #include "Wheel.h"
  int sc_main(int argc, char* argv[]) {
    Wheel* wheel_FL; // pointer to FL wheel
    Wheel* wheel_FR; // pointer to FR wheel
    wheel_FL = new Wheel ("wheel_FL"); // create FL
    wheel_FR = new Wheel ("wheel_FR"); // create FR
    sc_start();
    delete wheel_FL;
    delete wheel_FR;
  }
```

Direct Sub-Module Header-Only

```
//FILE:Body.h
  #include "Wheel.h"
  SC_MODULE(Body) {
    Wheel wheel_FL;
    Wheel wheel_FR;
    SC_CTOR(Body)
     : wheel_FL("wheel_FL"), //initialization
       wheel_FR("wheel_FR")  //initialization
    {
      // other initialization
    }
  };
```

Indirect Sub-Module Header-Only

```
//FILE:Body.h
  #include "Wheel.h"
  SC_MODULE(Body) {
    Wheel* wheel_FL;
    Wheel* wheel_FR;
    SC_CTOR(Body) {
      wheel_FL = new Wheel("wheel_FL");
      wheel_FR = new Wheel("wheel_FR");
      // other initialization
    }
  };
```

Direct Sub-Module

```
//FILE:Body.h
  #include "Wheel.h"
  SC_MODULE(Body) {
    Wheel wheel_FL;
    Wheel wheel_FR;
    SC_HAS_PROCESS(Body);
    Body(sc_module_name nm);
  };
```

```
//FILE: Body.cpp
  #include "Body.h"
  // Constructor
  Body::Body (sc_module_name nm)
   : wheel_FL("wheel_FL"),
     wheel_FR("wheel_FR"),
     sc_module(nm)
  {
    // other initialization
  }
```

Indirect Sub-Module

```
//FILE:Body.h
  struct Wheel;
  SC_MODULE(Body) {
    Wheel* wheel_FL;
    Wheel* wheel_FR;
    SC_HAS_PROCESS(Body);
    Body(sc_module_name nm); // Constructor
  };
```

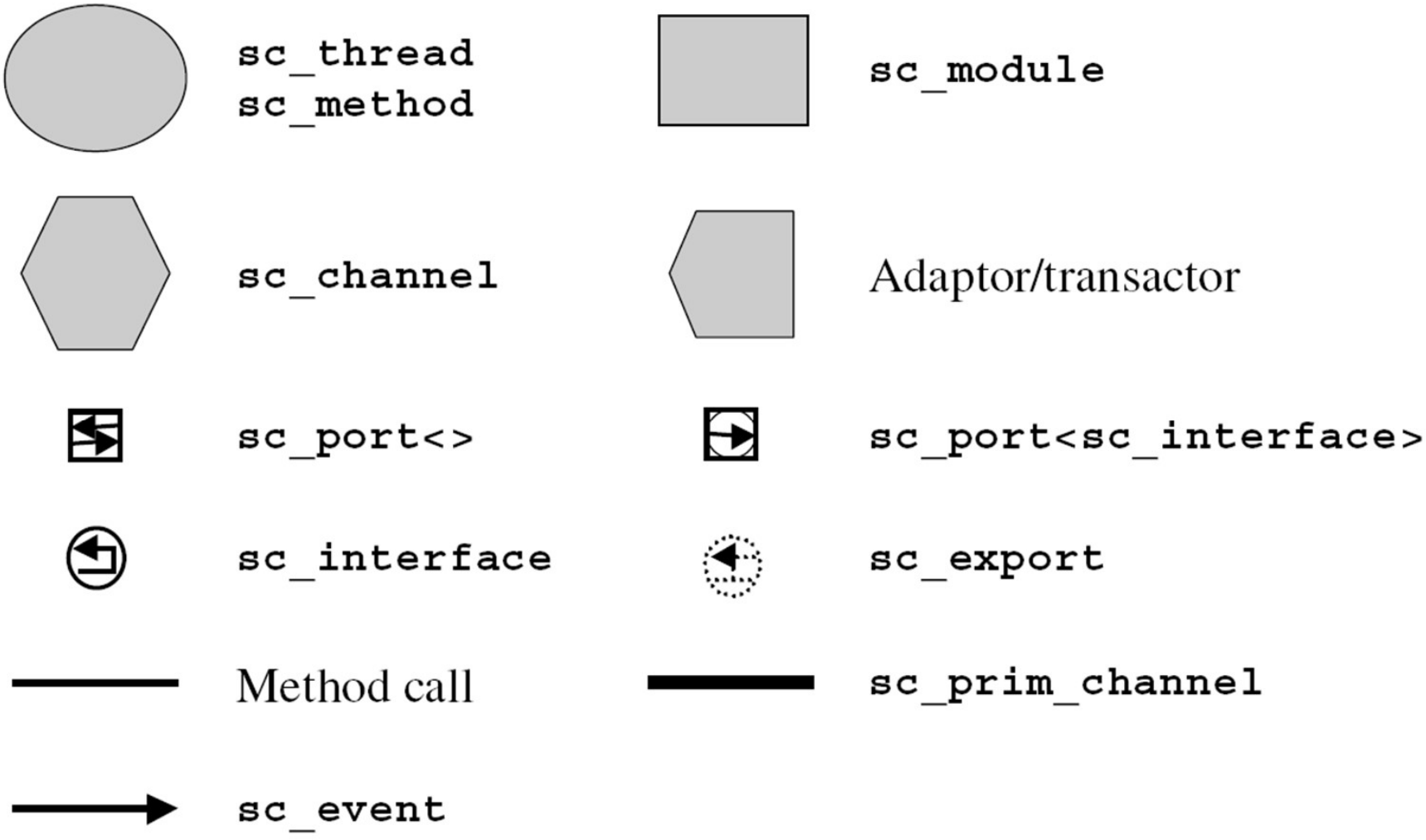
```
//FILE: Body.cpp
  #include "Wheel.h"
  // Constructor
  Body::Body(sc_module_name nm)
   : sc_module(nm)
  {
    wheel_FL = new Wheel("wheel_FL");
    wheel_FR = new Wheel("wheel_FR");
    // other initialization
  }
```

Comparison

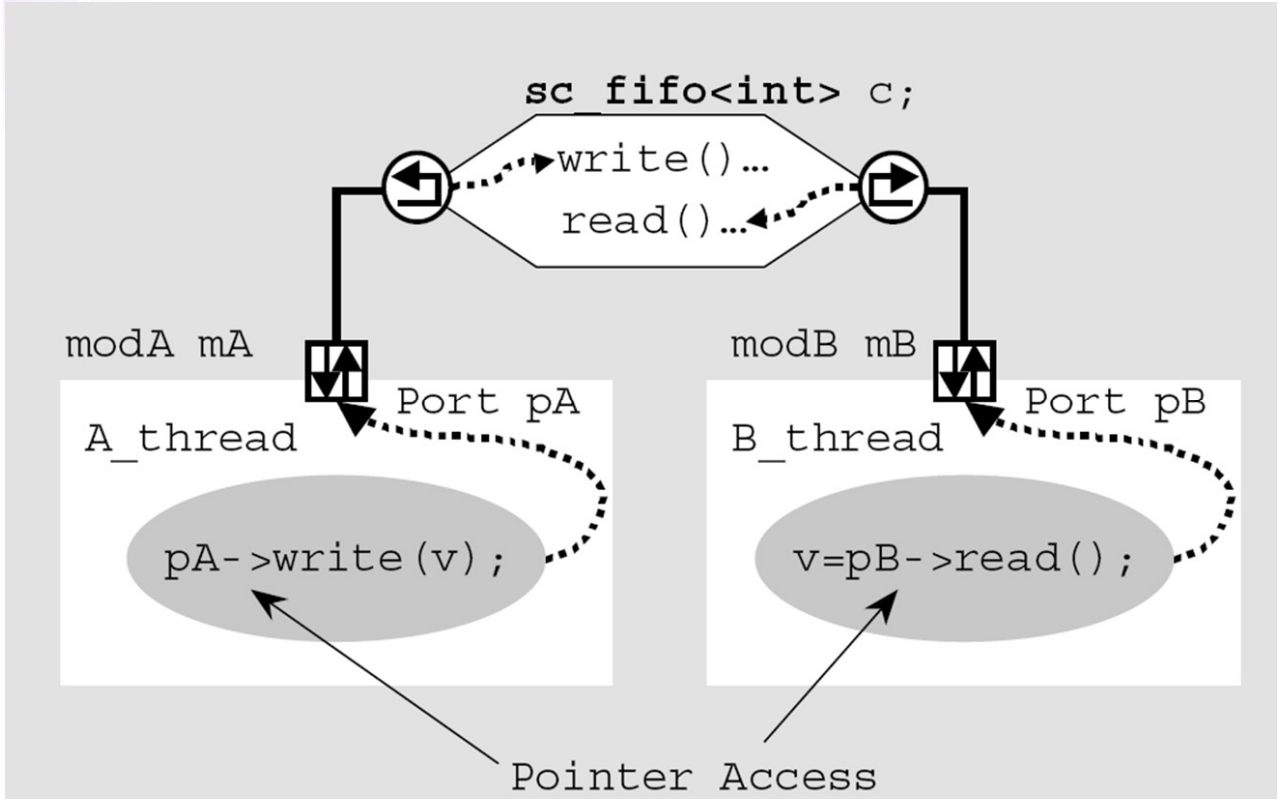
Level	Allocation	Pros	Cons
Main	Direct	Least code	Inconsistent with other levels
Main	Indirect	Dynamically configurable	Involves pointers
Module	Direct header only	All in one file Easier to understand	Requires sub-module headers
Module	Indirect header only	All in one file Dynamically configurable	Involves pointers Requires sub-module headers
Module	Direct with separate compilation	Hides implementation	Requires sub-module headers
Module	Indirect with separate compilation	Hides sub-module headers and implementation Dynamically configurable	Involves pointers

Chapter 11
Communication

Standard Graphical Notations

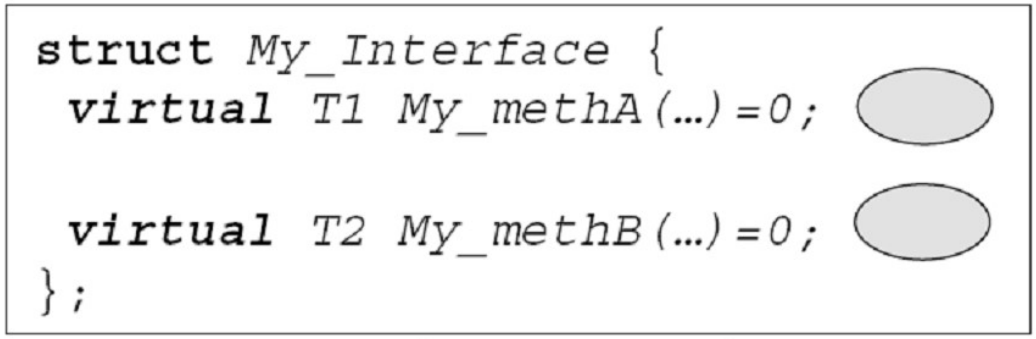


Channel, Port, Interface

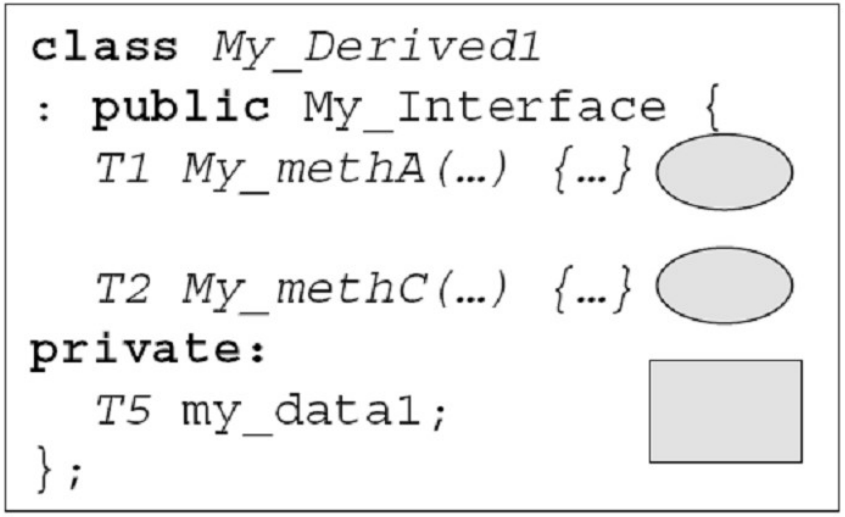


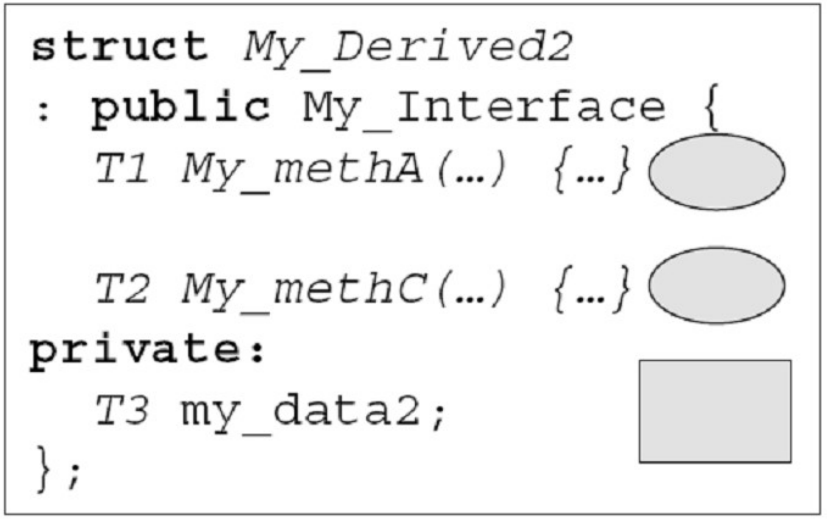
- For safety and ease of use
- A port is a pointer to a channel outside the module

Abstract Class



Abstract Class
- Pure virtual methods
- No data





Definitions

- A SystemC interface is an abstract class that inherits from **sc_interface** and provides only pure virtual declarations of methods referenced by SystemC channels and ports. No implementations or data are provided in a SystemC interface
- A SystemC channel is a class that implements one or more SystemC interface classes inherits from either **sc_channel** or **sc_prim_channel**. A channel implements all the mothods of the inherited interface class

Examples for Model Refining

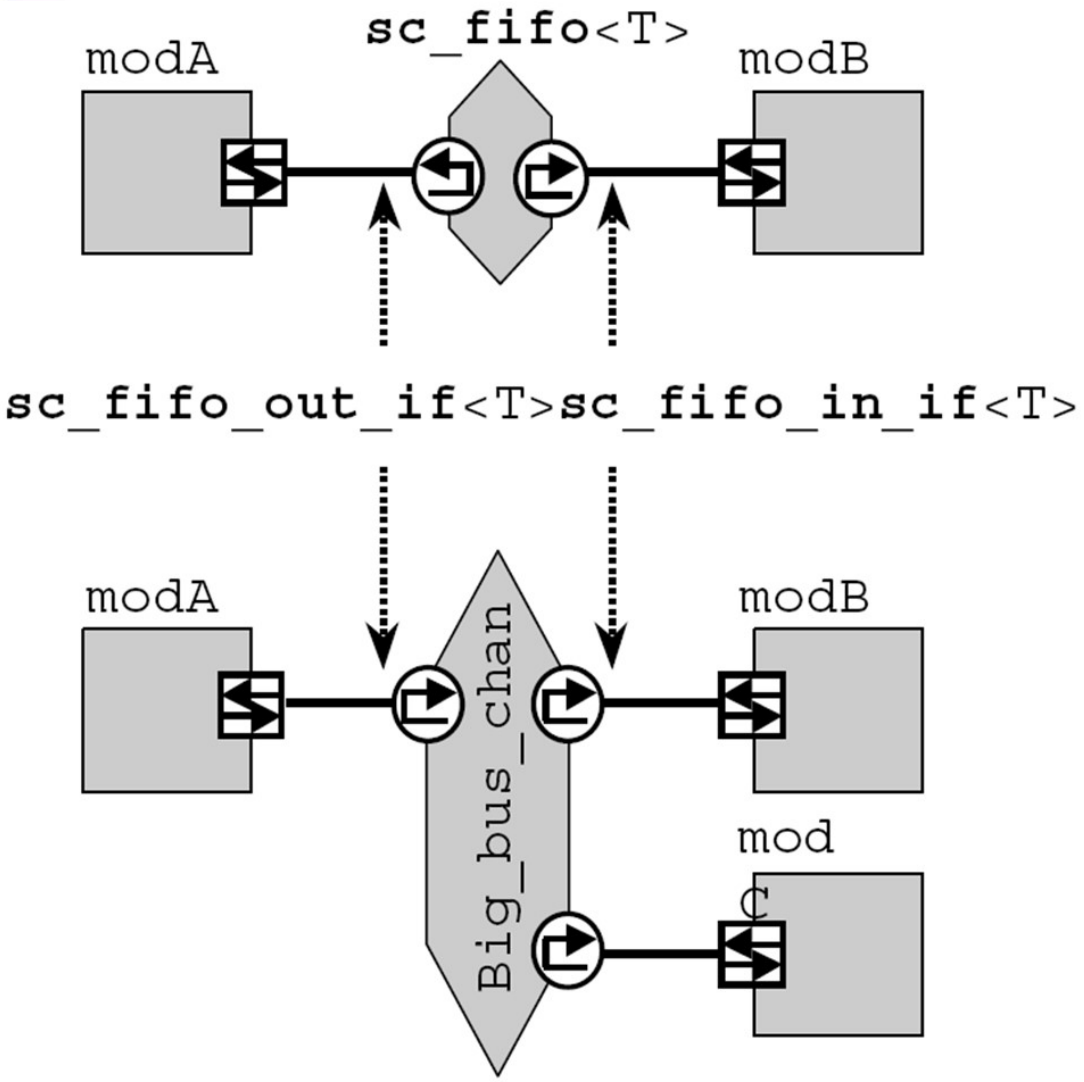
```
struct multiport_memory_arch: public my_interface {
  virtual void write(unsigned addr, int data) {
    mem[addr] = data;
  }// end write
  virtual int read(unsigned addr) ) {
    return mem[addr];
  }//end read
private:
  int mem[1024];
};
```

```
void memtest(my_interface mem) {
  // complex memory test
}

multiport_memory_arch fast;
multiport_memory_RTL  slow;
memtest(fast);
memtest(slow);
```

```
struct multiport_memory_RTL: public my_interface {
  virtual void write(unsigned addr, int data) {
    // complex details of RTL memory write
  }// end write
  virtual int read(unsigned addr) ) {
    // complex details of RTL memory read
  }// end read
private:
  // complex details of RTL memory storage
};
```

Examples for Model Refining



SystemC Port Declarations

- **sc_port** *<interface> portname*;

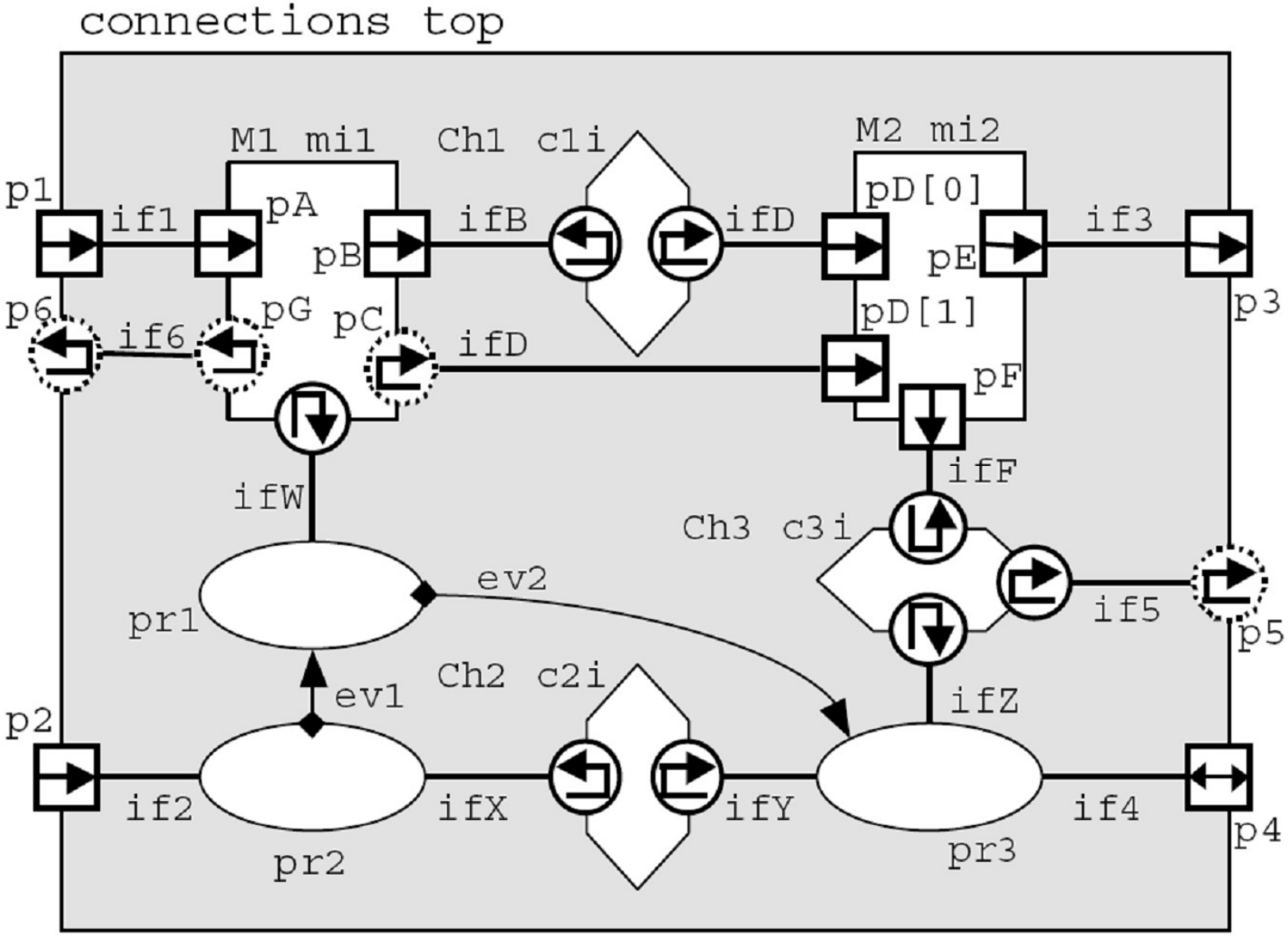
- Example:

```
SC_MODULE(stereo_amp) {
    sc_port<sc_fifo_in_if<int> > soundin_p;
    sc_port<sc_fifo_out_if<int> > soundout_p;
    …
};
```

Ways to Interconnect

From	To	Method
Port	Sub-module	Direct connect via `sc_port`
Process	Port	Direct access by process
Sub-module	Sub-module	Local channel connection
Process	Sub-module	Local channel connection –or- via `sc_export` –or- interface implemented by sub-module[31]
Process	Process	Events or local channel
Port	Local channel	Direct connect via `sc_export`

Illustration of Interconnection



Port Connection

```
//FILE:  Rgb2YCrCb.h
SC_MODULE(Rgb2YCrCb) {
  sc_port<sc_fifo_in_if<RGB_frame> >     rgb_pi;
  sc_port<sc_fifo_out_if<YCRCB_frame> > ycrcb_po;
};
```

```
//FILE: YCRCB_Mixer.h
SC_MODULE(YCRCB_Mixer) {
  sc_port<sc_fifo_in_if<float> >          K_pi;
  sc_port<sc_fifo_in_if<YCRCB_frame> > a_pi, b_pi;
  sc_port<sc_fifo_out_if<YCRCB_frame> > y_po;
);
```

Port Connection

```
//FILE: VIDEO_Mixer.h
SC_MODULE(VIDEO_Mixer) {
  // ports
  sc_port<sc_fifo_in_if<YCRCB_frame> >    dvd_pi;
  sc_port<sc_fifo_out_if<YCRCB_frame> > video_po;
  sc_port<sc_fifo_in_if<MIXER_ctrl> >    control;
  sc_port<sc_fifo_out_if<MIXER_state> > status;
  // local channels
  sc_fifo<float>          K;
  sc_fifo<RGB_frame>      rgb_graphics;
  sc_fifo<YCRCB_frame>    ycrcb_graphics;
  // constructor
  SC_HAS_PROCESS(VIDEO_Mixer);
  VIDEO_Mixer(sc_module_name nm);
  void Mixer_thread();
};
```

Port Connection

By Name

```
VIDEO_Mixer::VIDEO_Mixer(sc_module_name nm)
 :  sc_module(nm)
{
   // Instantiate
   Rgb2YCrCb  Rgb2YCrCb_i("Rgb2YCrCb_i");
   YCRCB_Mixer YCRCB_Mixer_i("YCRCB_Mixer_i");
   // Connect
   Rgb2YCrCb_i.rgb_pi(rgb_graphics);
   Rgb2YCrCb_i.ycrcb_po(ycrcb_graphics);
   YCRCB_Mixer_i.K_pi(K);
   YCRCB_Mixer_I.a_pi(dvd_pi);
   YCRCB_Mixer_i.b_pi(ycrcb_graphics);
   YCRCB_Mixer_i.y_po(video_po);
};
```

SC_THREAD(Mixer_thread);
sensitive << control;

Port Connection

By Position

```
VIDEO_Mixer::VIDEO_Mixer(sc_module_name nm)
 : sc_module(nm)
{
  // Instantiate
  Rgb2YCrCb  Rgb2YCrCb_i("Rgb2YCrCb_i");
  YCRCB_Mixer  YCRCB_Mixer_i("YCRCB_Mixer_i");
  // Connect
  Rgb2YCrCb_i(rgb_graphics,ycrcb_graphics);
  YCRCB_Mixer_i(K,dvd_pi,ycrcb_graphics,video_po);
};
```

```
SC_THREAD(Mixer_thread);
sensitive << control;
```

Port Access

- *portname->method(optional_args);*
- Example

```
void VIDEO_Mixer::Mixer_thread() {
  …
  switch (control->read()) {
    case MOVIE: K.write(0.0f); break;
    case MENU:  K.write(1.0f); break;
    case FADE:  K.write(0.5f); break;
    default:    status->write(ERROR); break;
  }
  …
}
```

- K is a local channel instance

Chapter 12
More on Ports

sc_fifo

- sc_fifo_out_if
  - write()
  - nb_write()
  - num_free()
  - data_read_event()
- sc_fifo_in_if
  - read()
  - nb_read()
  - num_available()
  - data_written_event()

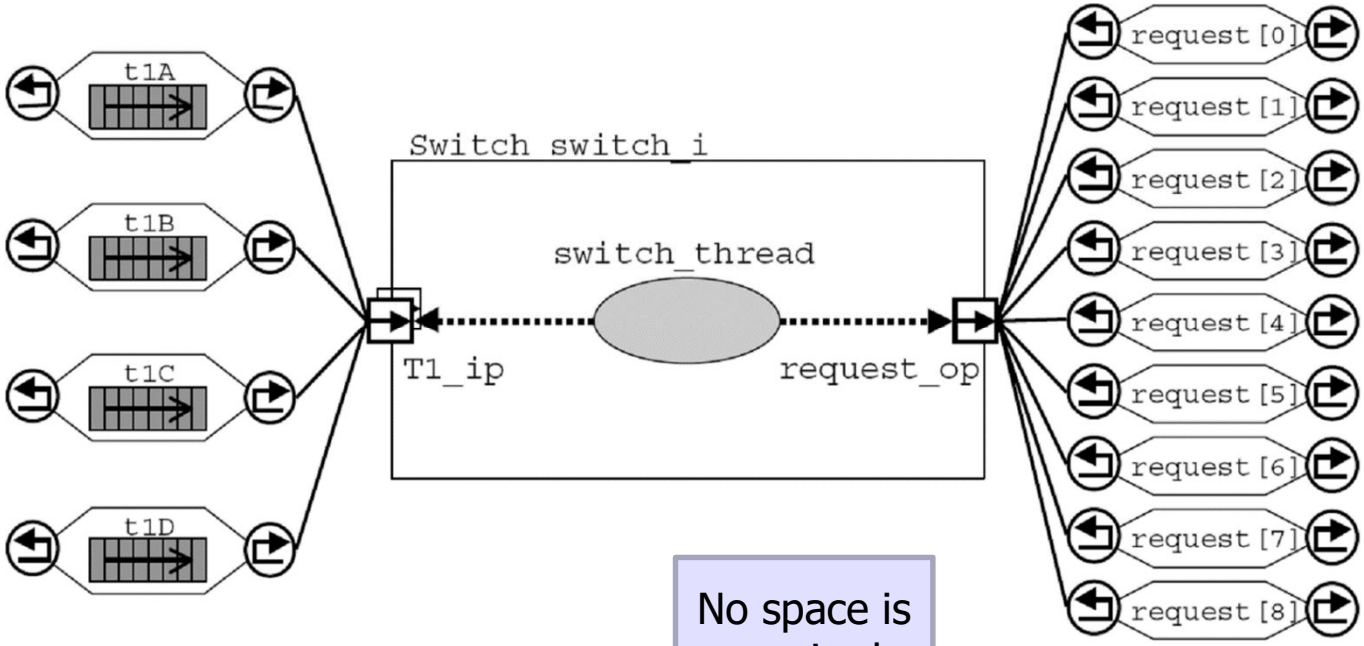
Note

- Tell the difference between
  - sc_fifo
  - sc_fifo_in_if
  - sc_fifo_in

- Answer:
  - sc_fifo
    - A primitive channel
  - sc_fifo_in_if
    - An interface proper and is implemented by the predefined channel sc_fifo. (sc_interface)
    - sc_port<sc_fifo_in_if<float> > in;
  - sc_fifo_in
    - A specialized port class for use when reading from a fifo. (sc_port)

sc_port<> Array

- **sc_port** *<interface [,N]> portname;*
  - N=0..MAX Default N=1



No space is required

```
//FILE: Switch.h
SC_MODULE(Switch) {
  sc_port<sc_fifo_in_if<int>, 4> T1_ip;
  sc_port<sc_signal_out_if<bool>,0> request_op;
  ...
};
```

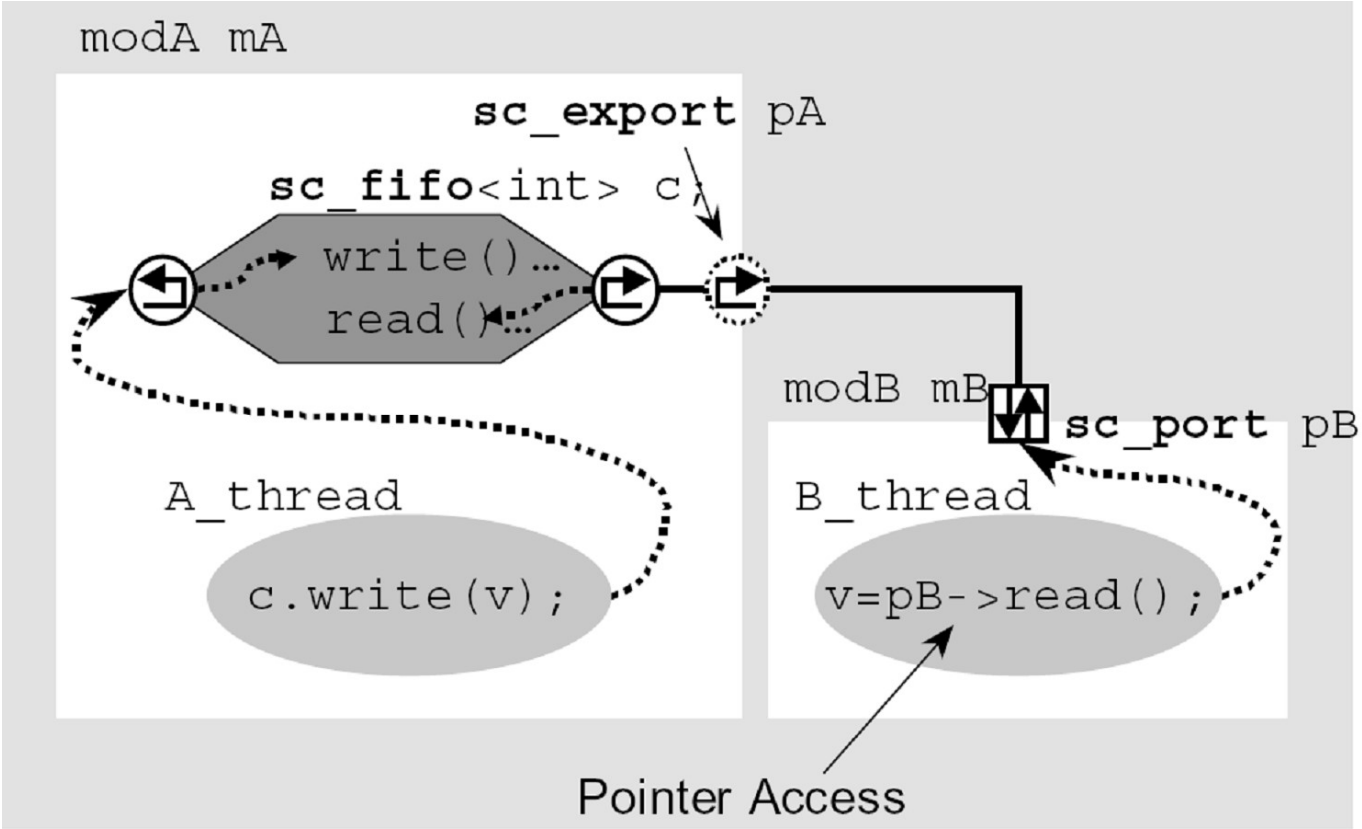
sc_port<> Array

```
//FILE: Board.h
#include "Switch.h"
SC_MODULE(Board) {
  const unsigned REQS;
  Switch switch_i;
  sc_fifo<int> t1A, t1B, t1C, t1D;
  sc_signal<bool> request[9];
  SC_CTOR(Board): switch_i("switch_i")
  {
    // Connect 4 T1 channels to the switch
    // input T1 ports
    switch_i.T1_ip(t1A); switch_i.T1_ip(t1B);
    switch_i.T1_ip(t1C); switch_i.T1_ip(t1D);
    // Connect 9 request channels to the
    // switch request output ports
    for (unsigned i=0;i!=9;i++) {
      switch_i.request_op(request[i]);
    }//endfor
    ...
  }//end constructor
  ...
};
```

```
//FILE: Switch.cpp
void Switch::switch_thread() {
  // Initialize requests
  for (unsigned i=0;i!=request_op.size();i++) {
    request_op[i] ->write(true);
  }//endfor
  // Startup after first port is activated
  wait(T1_ip[0]->data_written_event()
       |T1_ip[1]->data_written_event()
       |T1_ip[2]->data_written_event()
       |T1_ip[3]->data_written_event()
  );
  for(;;) {
    for (unsigned i=0;i!=T1_ip.size();i++) {
      // Process each port…
      int value = T1_ip[i]->read();
    }//endfor
  }//endforever
}//end Switch::switch_thread
```

sc_export

- A new type in SystemC 2.1
- Move the channel inside the defining module, and use the port externally as though it was a channel



Example

```
SC_MODULE(clock_gen) {
  sc_export<sc_ signal<bool> > clock_xp;
  sc_signal<bool> oscillator;
  SC_CTOR(clock_gen) {
    SC_METHOD(clock method);
    clock_xp(oscillator); // connect sc_signal
                          // channel
                          // to export clock_xp
    oscillator.write(false);
  }
  void clock_method() {
    oscillator.write(!oscillator.read());
    next_trigger(10,SC_NS);
  }
};
```

Example

```
#include "clock_gen.h"
…
clock_gen clock_gen_i;
collision_detector collision_detector_i;
// Connect clock
collision_detector_i.clock(clock_gen_i.clock_xp);
…
```

Chapter 13
Custom Channels and Data

Custom Channels and Data

- Primitive channels
- **Custom signals**
- Custom hierarchical channels
- **Custom adaptors**

Channels

- Inherit either from sc_prim_channel or sc_channel
- Primitive channel
  - Very simple and fast communication
  - No hierarchy, no port, and no SC_METHODs or SC_THREADs
- Hierarchical channel
  - May access ports and can have processes and contain hierarchy
  - Another modules that implement one or more interfaces
  - Intended to model complex communication buses, such as PCI, HyperTransport, or AMBA

The Interrupt, a Custom Primitive Channel *(project: interrupt)*

```
#include "ea_interrupt_evt_if.h"
#include "ea_interrupt_gen_if.h"

struct ea_interrupt
: public sc_prim_channel
, public ea_interrupt_evt_if
, public ea_interrupt_gen_if
{
  explicit ea_interrupt()
  : sc_prim_channel(sc_gen_unique_name("ea_interrupt")) {}
  explicit ea_interrupt(sc_module_name nm)
  : sc_prim_channel(nm) {}
  void notify() { m_interrupt.notify(); }
  void notify(sc_time t) { m_interrupt.notify(t); }
  const sc_event& default_event() const { return m_interrupt; }
private:
  sc_event m_interrupt;
  // Copy constructor so compiler won't create one
  ea_interrupt( const ea_interrupt& );
};
```

```
struct ea_interrupt_evt_if: public sc_interface {
  virtual const sc_event& default_event() const = 0;
};
```

```
struct ea_interrupt_gen_if: public sc_interface {
  virtual void notify() = 0;
  virtual void notify(sc_time t) = 0;
};
```

The Interrupt, a Custom Primitive Channel

- In interrupt.cpp

```
interrupt::interrupt(sc_module_name nm) {
  irq = new ea_interrupt();
  stim_i = new stim("stim_i");
  resp_i = new resp("resp_i");
  stim_i->irq_op(*irq);
  resp_i->irq_ip(*irq);
}
```

- In stim.cpp

```
wait(2,SC_NS);  // Get off zero
  for (unsigned i=0; i!=5; i++) {
    cout << "INFO: " << name() << " Sending event at " <<
sc_time_stamp() << endl;
    irq_op->notify(SC_ZERO_TIME);
    wait(5,SC_NS);
  }
```

Can we use sc_signal to model interrupt?

The Packet, a Custom Data Type for SystemC *(project: pkt_switch)*

- For custom data types, SystemC requires you to define several methods for your data type
  - Ex. Operator=()
  - Operator<<()
  - sc_trace

The Packet, a Custom Data Type for SystemC

```
struct pkt {
      sc_int<8> data;
      sc_int<4> id;
      bool dest0;
      bool dest1;
      bool dest2;
      bool dest3;

      inline bool operator == (const pkt& rhs) const
  {
    return (rhs.data == data && rhs.id == id &&
rhs.dest0 == dest0 && rhs.dest1 == dest1 &&
rhs.dest2 == dest2 && rhs.dest3 == dest3);
  }

};
```

```
inline
ostream&
operator << ( ostream& os, const pkt& a )
{
   os << "streaming of struct pkt not implemented";
   return os;
}

inline
void
#if defined(SC_API_VERSION_STRING)
   sc_trace( sc_trace_file* tf, const pkt& a, const std::string&
name )
#else
   sc_trace( sc_trace_file* tf, const pkt& a, const sc_string&
name )
#endif
{
  sc_trace( tf, a.data, name + ".data" );
  sc_trace( tf, a.id, name + ".id" );
  sc_trace( tf, a.dest0, name + ".dest0" );
  sc_trace( tf, a.dest1, name + ".dest1" );
  sc_trace( tf, a.dest2, name + ".dest2" );
  sc_trace( tf, a.dest3, name + ".dest3" );
}
```

The Heartbeat, a Custom Hierarchical Channel *(project: heartbeat)*

- Hybrid module
- Inherit from sc_channel

```
struct ea_heartbeat_if: public sc_interface {
  virtual const sc_event& default_event() const = 0;
  virtual const sc_event& posedge_event() const = 0;
};
```

The Heartbeat, a Custom Hierarchical Channel

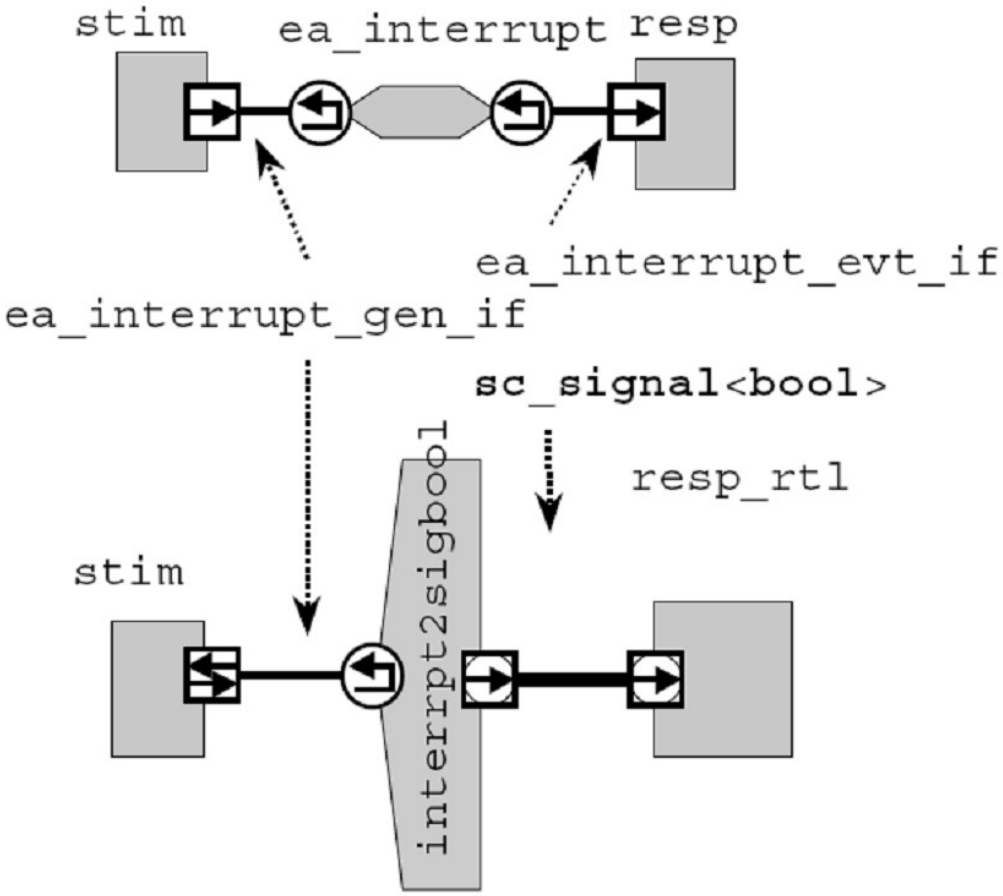
```
struct ea_heartbeat
: public sc_channel
, public ea_heartbeat_if
{
  SC_HAS_PROCESS(ea_heartbeat);
  // Constructors

  explicit ea_heartbeat(sc_module_name nm, sc_time _period)
  : sc_channel(nm)
  , m_period(_period) {
    SC_METHOD(heartbeat_method);
    sensitive << m_heartbeat;
  }
  // User methods
  const sc_event& default_event() const { return m_heartbeat; }
  const sc_event& posedge_event() const { return m_heartbeat; }
  // Processes
  void heartbeat_method();
private:
  sc_event m_heartbeat;
  sc_time  m_period;
  // Copy constructor so compiler won't create one
  ea_heartbeat( const ea_heartbeat& );
};
```

```
void ea_heartbeat::heartbeat_method(void)
{
  m_heartbeat.notify(m_period);
}
```

The Adaptor, a Custom Primitive Channel

- Adaptors are used when moving between different abstractions



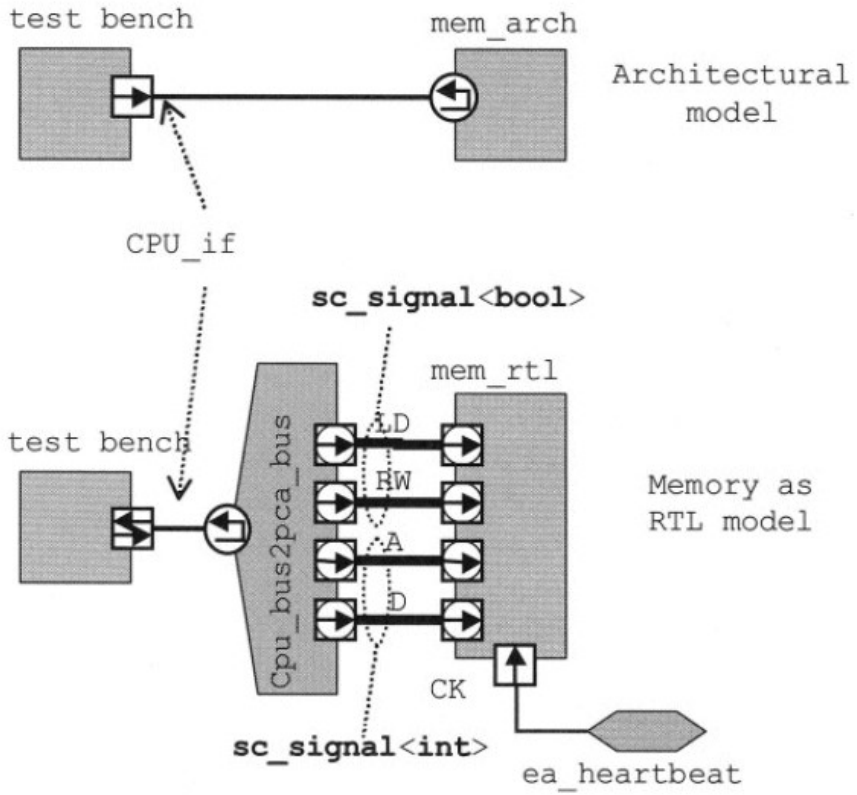
The Adaptor, a Custom Primitive Channel

```
struct interrupt2sigbool
: public sc_prim_channel
, public ea_interrupt_gen_if
, public sc_signal_in_if<bool>
{
  explicit interrupt2sigbool()
  : sc_prim_channel(sc_gen_unique_name("interrupt2sigbool")) {}
  void notify() { m_delay = SC_ZERO_TIME; request_update(); }
  void notify(sc_time t) { m_delay = t; request_update(); }
  // get the value changed event
  const sc_event& value_changed_event() const { return m_interrupt; }
  const sc_event& posedge_event() const { return value_changed_event(); }
  const sc_event& negedge_event() const { return value_changed_event(); }
  const sc_event& default_event() const { return value_changed_event(); }
  // current value is true if last delta cycle was active
  const bool& read() const { m_val = event(); return m_val; }
  // get a reference to the current value (for tracing)
  const bool& get_data_ref() const { return read(); }
  // was there a value changed event?
  bool event() const { return ( simcontext()->delta_count() == m_delta + 1 ); }
  bool posedge() const { return event(); }
  bool negedge() const { return event(); }
  const sc_signal_bool_deval& delayed() const {
     const sc_signal_in_if<bool>* iface = this;
     return RCAST<const sc_signal_bool_deval&>( *iface );
  }
protected:
    // every update is a change
    virtual void update() {
      m_interrupt.notify(m_delay);
      m_delta = simcontext()->delta_count();
    }

private:
  sc_event m_interrupt;
  mutable bool m_val;
  sc_time  m_delay;
  uint64   m_delta;    // delta of last event
  // Copy constructor so compiler won't create one
  interrupt2sigbool( const interrupt2sigbool& );
};
```

The Transactor, a Custom Hierarchical Channel *(project: hier_chan)*

- The adaptor is often referred to as a transactor because it allows the test bench to convert transactions into pin-level stimulus



The Transactor, a Custom Hierarchical Channel

- Simple CPU I/F

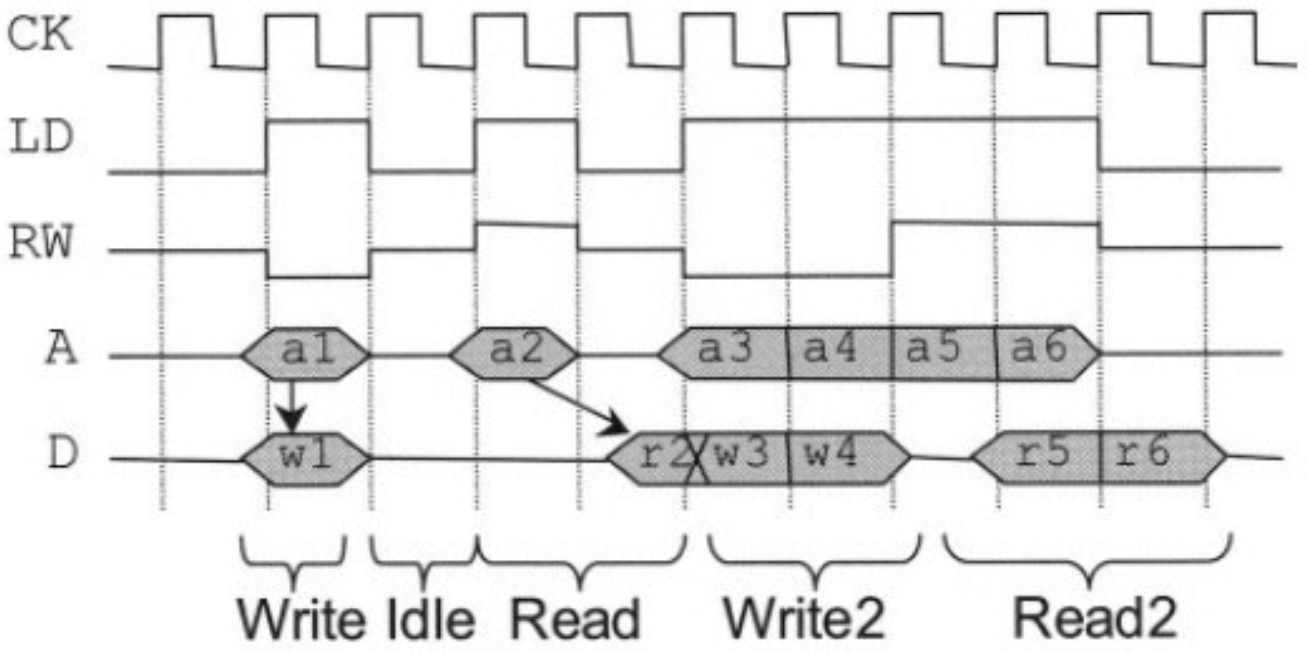
```
struct CPU_if: public sc_interface {
  virtual void write(unsigned long addr, long data)=0;
  virtual long read(unsigned long  addr)=0;
};
```

The Transactor, a Custom Hierarchical Channel

```
#include "CPU_if.h"
struct mem_arch: public sc_channel, CPU_if {
  // Constructors & destructor
  explicit mem_arch(sc_module_name nm,
                    unsigned long ba, unsigned sz)
  : sc_channel(nm), m_base(ba), m_size(sz)
  { m_mem = new long[m_size]; }
  ~mem_arch() { delete [] m_mem; }
  // Interface implementations
  void write(unsigned long addr, long data) {
    if (m_base <= addr && addr < m_base+m_size) {
      m_mem[addr-m_base] = data;
    }
  }
  long read(unsigned long  addr) {
    if (m_base <= addr && addr < m_base+m_size) {
      return m_mem[addr-m_base];
    } else {
      cout << "ERROR:"<<name()<<"@"<<sc_time_stamp()
           << ": Illegal address: " << addr << endl;
      return 0;
    }
  }
private:
  unsigned long m_base;
  unsigned      m_size;
  long*         m_mem;
  mem_arch(const mem_arch&); // Disable
};
```

The Transactor, a Custom Hierarchical Channel

- Pin-cycle accurate model



The Transactor, a Custom Hierarchical Channel

The tansactor

```
#include "CPU_if.h"
#include "ea_heartbeat_if.h"
SC_MODULE(cpu2pca), CPU_if {
  // Ports
  sc_port<ea_heartbeat_if> ck; // clock
  sc_out<bool>             ld; // load/execute command
  sc_out<bool>             rw; // read high/write low
  sc_out<unsigned long>    a;  // address
  sc_inout_rv<32>          d;  // data
  // Constructor
  SC_CTOR(cpu2pca) : FLOAT(sc_string("0xZZZZZZZZ")) {
    //SC_THREAD(cpu2pca_thread);
    //sensitive << ck;
  }
  // Processes
  //void cpu2pca_thread(); // used for non-blocking implementations
  // Interface implementations
  void write(unsigned long addr, long data);
  long read(unsigned long addr);
  …
private:
  const sc_signal_rv<32>   FLOAT;
  cpu2pca(const cpu2pca&); // Disable
};
```

The Transactor, a Custom Hierarchical Channel

```
void cpu2pca::write(unsigned long addr,
long data) {
    //cout << "INFO: " << name() <<
"::write starting @ " << sc_time_stamp()
<< endl;
    wait(ck->posedge_event());
    ld->write(true);
    rw->write(WRITE);
    a->write(addr);
    d->write(data);
    wait(ck->posedge_event());
    ld->write(false);
  }
```

```
long cpu2pca::read(unsigned long addr) {
    //cout << "INFO: " << name() << "::read
starting @ " << sc_time_stamp() << endl;
    wait(ck->posedge_event());
    ld->write(true);
    rw->write(READ);
    a->write(addr);
    d->write(FLOAT);
    wait(ck->posedge_event());
    ld->write(false);
    return d->read().to_long();
  }
```

Chapter 14
Other Topics

Debug with Waveforms

```
sc_trace_file* tracefile;
tracefile = sc_create_vcd_trace_file(tracefile_name);
if (!tracefile) cout <<"There was an error."<<endl;
…
sc_trace(tracefile,signal_name,"signal_name");
…
sc_start(); // data is collected
…
sc_close_vcd_trace_file(tracefile);
```

```
//FILE: wave.h
SC_MODULE(wave) {
  sc_signal<bool> brake;
  sc_trace_file*  tracefile;
  …
  double temperature;
};
```

```
//FILE: wave.cpp
wave::wave(sc_module_name nm) //Constructor
: sc_module(nm) {
  …
  tracefile = sc_create_vcd_trace_file("wave");
  sc_trace(tracefile,brake,"brake") ;
  sc_trace(tracefile,temperature,"temperature");
}//endconstructor
wave::~wave() {
    sc_close_vcd_trace_file(tracefile);
    cout << "Created wave.vcd" << endl;
}
```

VCD Waveform Viewers

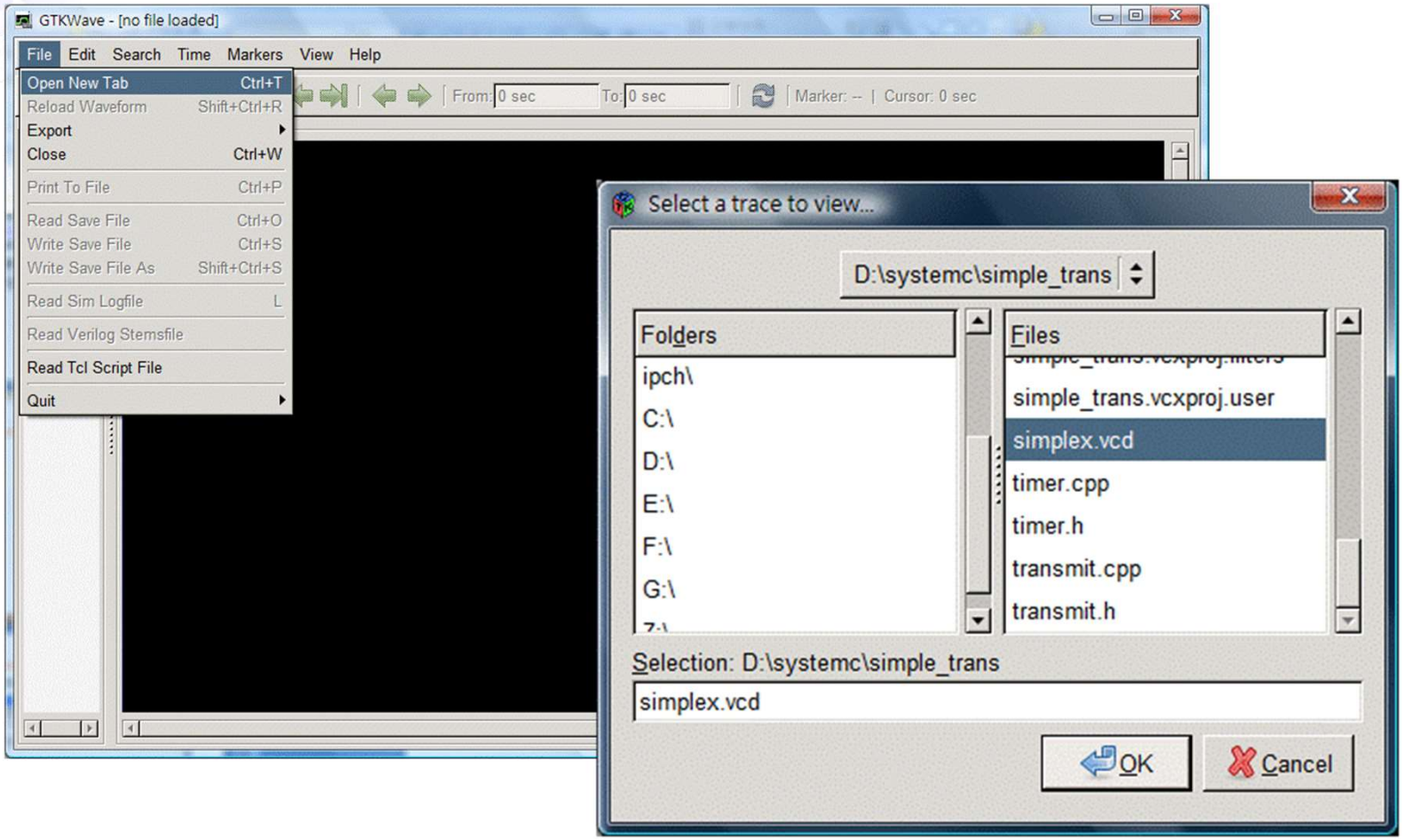
- A lot of VCD waveform viewers
- Freewares
  - GTKWave
    - http://www.dspia.com/gtkwave.html
  - WaveViewer (large but complete)
    - http://www.syncad.com/freeviewer.htm
  - Wave VCD viewer (need to change the date of VCD file)
    - http://www.iss-us.com/wavevcd/index.htm

GTKWave

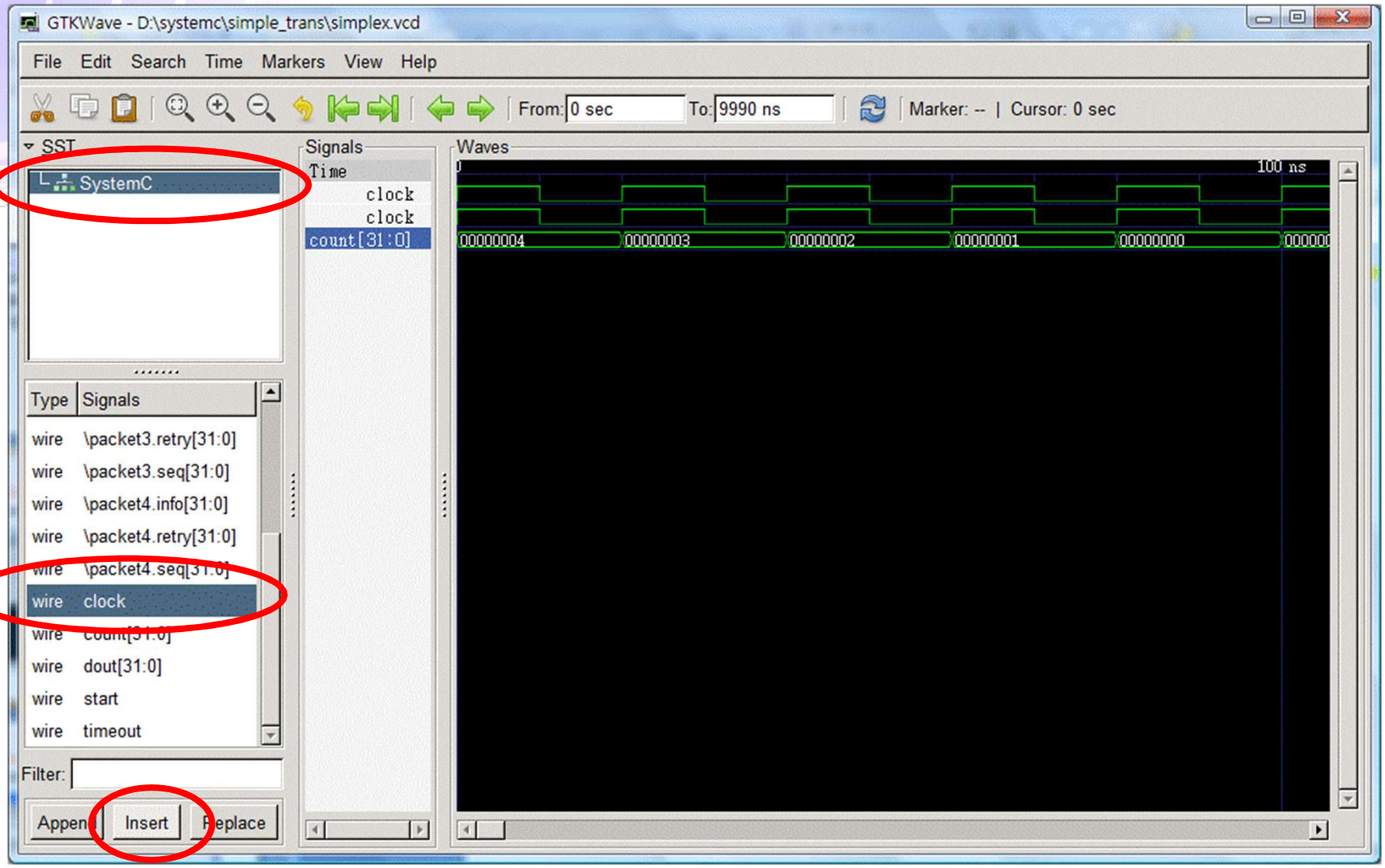
- Download gtkwave.exe.gz and all_libs.tar.gz, decompress them
- Move gtkwave.exe inside bin directory
- Execute gtkwave

GTKWave

- Open .vcd file



GTKWave


GTKWave - D:\systemc\simple_trans\simplex.vcd
File Edit Search Time Markers View Help
From: 0 sec
To: 9990 ns
Marker: -- | Cursor: 0 sec
SST
SystemC
Signals
Time
clock
clock
count[31:0]
Waves
100 ns
00000004
00000003
00000002
00000001
00000000
Type Signals
wire \packet3.retry[31:0]
wire \packet3.seq[31:0]
wire \packet4.info[31:0]
wire \packet4.retry[31:0]
wire clock
wire dout[31:0]
wire start
wire timeout
Filter:
Insert

Chapter 15
SystemC Verification Library

SystemC Verification Library

- SystemC Verification Library (SCV)
- Provide much of the features required to implement a robust reusable testbench without having to develop these on your own
- Add-on features to SystemC
  - Data introspection
  - Extended data types
  - Random data types
  - Transaction monitoring
  - Transaction recording

Data Introspection

- SCV uses data introspection to enable the manipulation of arbitrary data types
  - C/C++ built-in, SystemC, and user-specified data types
- Map data types to an abstract interface *scv_extensions_if,* which include the following member operations
  - Extraction of type information
  - Value access and value assignment
  - Randomization
  - Callback registration
- **scv_smart_prt**: like a C++ pointer to the **scv_extensions** object

scv_extensions_if

- Include method for static extensions and dynamic extensions
- Static
  - Type information and value access or assignment

```
int data;
int bitwidth =
scv_get_extensions(data).get_bitwidth();
scv_get_extensions(data).print();
```

```
template<typename datatype> void process(datatype& data) {
scv_extensions<datatype> data_ext = scv_get_extensions(data);
process_core(&data_ext);
}
void process_core(scv_extensions_if *);
```

scv_extensions_if

- Another example

```
void print_data(scv_extensions_if* data_ptr) {
  switch(data_ptr->get_type()) {
    case scv_extensions_if::BOOLEAN:
        cout<<data_ptr->get_type_name()
            <<"_value is: "
            <<data_ptr ->get_bool();
…
  }//end switch
}// end print_data
```

scv_extensions_if

- Define the extensions for user-specified types

```
SCV_EXTENSIONS(rw_task_if::write_t) {
public:
    scv_extensions< rw_task_if::addr_t > addr;
    scv_extensions< rw_task_if::data_t > data;
    SCV_EXTENSIONS_CTOR(rw_task_if::write_t) {
        SCV_FIELD(addr);
        SCV_FIELD(data);
    }
};
```

scv_extensions_if

- Dynamic
  - Random and callback function pointer
- Randomization
  - Direction testing with certain created test scenearios➔for unit test
  - Randomization with weighted constraints ➔ for unit test and system-level verification to increase converage

scv_extensions_if

- Randomization
  - Global configuration
    - set_deault_algorithm
      - RAND, RAND32, RAND48 (default), or CUSTOM
    - set_mode
      - RANDOM, SCAN, RANDOM_AVOID_DUPLICATE, DISTRIBUTION
    - set_global_seed
    - set_current_seed
    - enable_randomization()
    - disable_randomization()

scv_extensions_if

- Randomization
  - Basic randomization
    - **next()**: create random value

```
scv_smart_ptr<Packet> pPkt;
pPkt->next(); //creat random values for address & data
```

```
scv_smart_ptr<Packet> pPkt;
pPkt->address.disable_randomization();
pPkt->next(); //creat random values for data
```

```
scv_smart_ptr<Packet> pPkt;
pPkt->address.next(); //creat random values for address
```

scv_extensions_if

- Randomization
  - Constrained randomization

```
class Pkt_constraint
        : virtual public scv_constraint_base{
public:
scv_smart_ptr<Packet> pPkt;
SCV_CONSTRAINT_CTOR(Pkt_constraint){
 //define constraints
SCV_CONSTRAINT(
          (pPkt->address() != 0x00000000) &&
          (pPkt->address() < 0x00000800)
      );
SCV_CONSTRAINT (pPkt->data () >= 0x00001000);
)
);
```

scv_extensions_if

- Randomization
  - Constrained randomization

```
Pkt_constraint cPkt;
cPkt.next();
```

```
Pkt_constraint cPkt;
scv_smart_ptr<Packet> pPkt;
pPkt->use_constraint(cPkt);
pPkt->next();
```

scv_extensions_if

- Weighted randomization

```
scv_smart_ptr<Packet> pPkt;
pPkt->address.keep_only(1,9999);
pPkt->data.keep_out(0);
pPkt->data.keep_out(10000U,(1U<<30));
```

- scv_bag

```
// define a bag
scv_bag<int> intBag;

intBag.add(0, 25); //add 25 objects of value 0 to bag

intBag.add(1, 25); //add 25 objects of value 1 to bag

intBag.add(2, 50); //add 50 objects of value 2 to bag


scv_smart_ptr<int> smart_int;
smart_int->set_mode(intBag);   //set smart_int
                               //distribution
```

scv_bag

```
class Pkt_constraint
: virtual public scv_constraint_base
{
public:
 scv_smart_ptr<sc_uint<16>> address;
 scv_smart_ptr<sc_uint<32>> data;
 SCV_CONSTRAINT_CTOR(Pkt_constraint) {
    // define constraints
SCV_CONSTRAINT(
      (address() != 0x00000000) &&
      (address() < 0x00001000));
SCV_CONSTRAINT(data() >= 0x1000);
  }
};

void test() {

  typedef pair <sc_uint<32> sc_u int<32>> data_range;
  scv_bag<data_range> data_dist;
  //set range distribution for data
  //data range (0x1000, 0xffff) occurs 30%
  //data range (0x10000, 0x20000) occurs 70%)
  data_dist.add(data_range(0x1000, 0xffff),30);
  data_dist.add(data_range(0x10000, 0x20000), 70);

  Pkt_constraint cPkt;
  cPkt.next();          //generate addr and data using
                        //constraints
  cPkt.data->set_mode(data_dist);
  cPkt.next();          //generate addr using
                        //constraints and generate
                        //data using 'data_dist'
                        //distribution
}
```

scv_extensions_if

- Callbacks
  - Once a function has been registered as a callback, it will be called anytime the referenced object changes
  - **register_cb** and **remove_cb**

```
#include "Packet.h"

static unsigned changes;
// A function to monitor changes on a Packet
void Packet_cbA(
scv_extensions_if& obj,
scv_extensions_if::callback_reason reason
) {
if (reason == scv_extensions_if::VALUE_CHANGE)  {
cout<< "Packet " << obj.get_name()
<< " value change to " << obj.get_unsigned()
<<endl;
  } else {
cout<< "Packet " << obj.get_name()
<< " deleted." <<endl;
  }
}
void Packet_cbB(
scv_extensions_if& obj,
scv_extensions_if::callback_reason reason
) {
  if (reason == scv_extensions_if::VALUE_CHANGE)  {
   changes++;
  } else {
cout<< changes << " distinct values"
<<endl;
  }
}
scv_smart_ptr<Packet> pPkt1("pPkt1"),pPkt2("pPkt2");
scv_extensions_if::callback_h
  h1A(pPkt1->register_cb(Packet_cbA)),
h1B(pPkt1->register_cb(Packet_cbB)),
  h2A(pPkt2->register_cb(Packet_cbA));

for (int i=0; i!=10; ++i) {
  pPkt1->next(); pPkt2->next();
}
pPkt1->remove_cb(h1A);
```

Sparse Arrays

- Use for memory modeling
  - A sparse array in normal cases
  - Can simulate large memory while running simulation on a computer with less memory

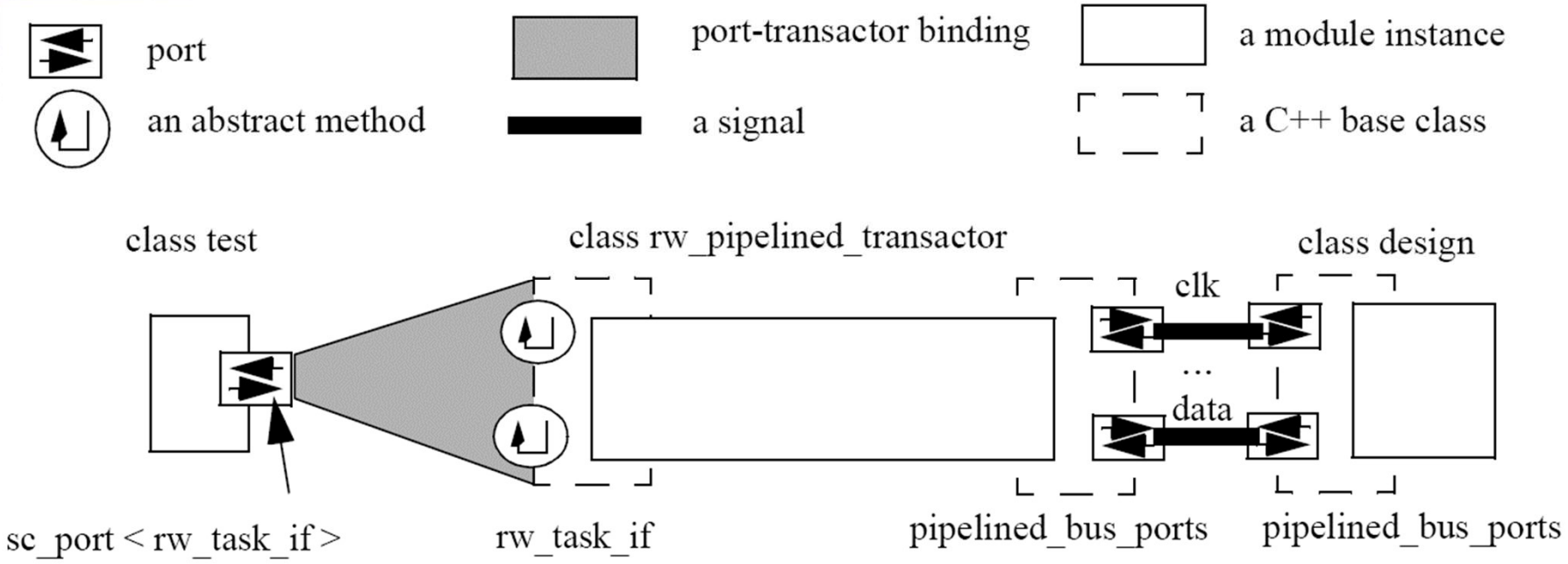
```
scv_sparse_array<T1,T2> NAME(
const char * name,
const T2& default_value,
const T1& indexLB = 0,
const T1& indexUB = INT_MAX
);
```

```
scv_sparse_array<unsigned,short> mem("mem",0,0,1e6);
scv_smart_ptr<unsigned> a_ptr;
scv_smart_ptr<short> d_ptr;
a_ptr->keep_only(0,1e6);
d_ptr->keep_out(0);
for (unsigned count=0; count!=30; ++count) {
  a_ptr->next(); d_ptr->next();
  mem[*a_ptr] = *d_ptr;
}
for (unsigned count=0; count!=30; ++count) {
  a_ptr->next();
  *d_ptr = mem[*a_ptr];
cout<< *d_ptr <<endl;
}
```

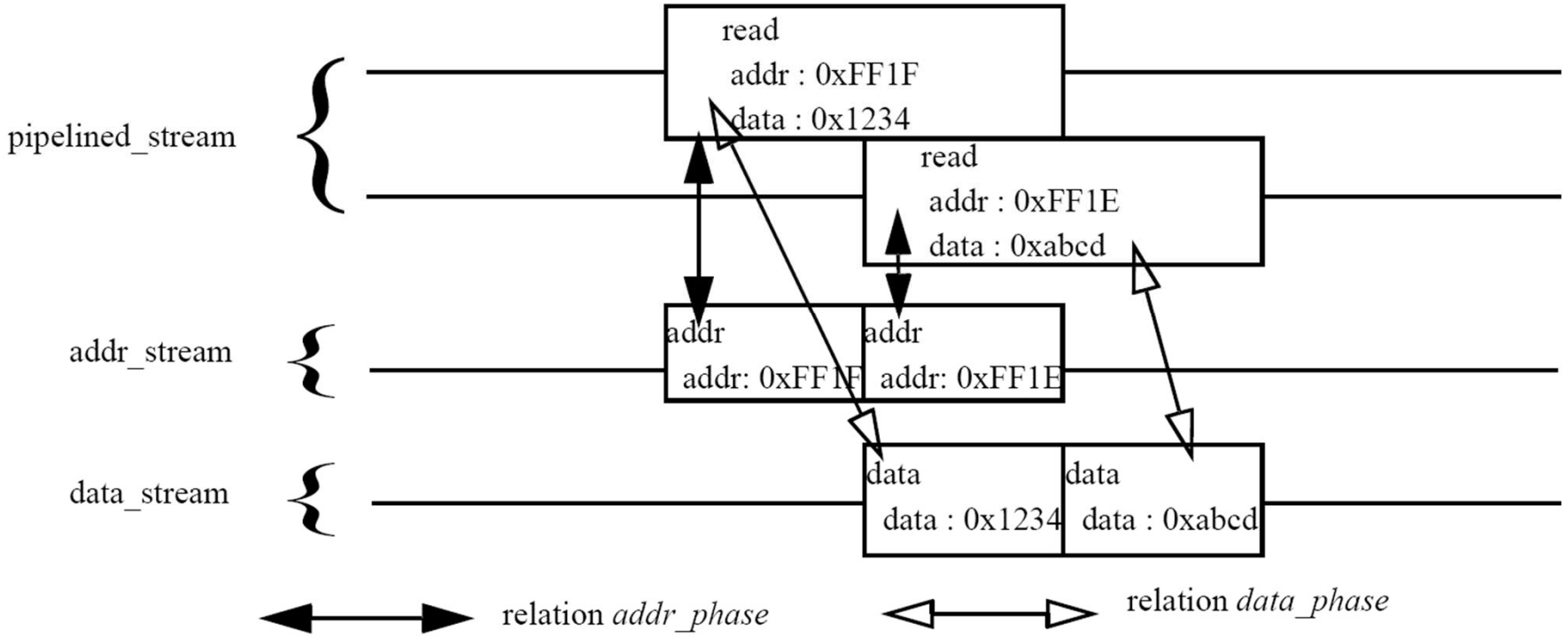
Transaction Recording

- Not an official part
- But many companies are using this interface
- **scv_tr_db**: transaction database containing a collection of transaction streams
- **scv_tr_stream**: transaction stream containing a collection of transactions
- **scv_tr_generator**: transaction generator for a specific tranction type

Example


port
an abstract method
port-transactor binding
a signal
a module instance
a C++ base class
class test
class rw_pipelined_transactor
class design
clk
...
data
sc_port < rw_task_if >
rw_task_if
pipelined_bus_ports
pipelined_bus_ports

Transaction Recording



Transaction Recording

```
data_t rw_pipelined_transactor::read( const addr_t * addr) {
  address_phase.lock();
  scv_tr_handle h = read_gen.begin_transaction(*addr);
  { // address phase
    scv_tr_handle h1
      = addr_gen.begin_transaction(*addr, "addr_phase", h);
    ...// address phase
    addr_gen.end_transaction(h1);
  }
  addr_phase.unlock();

  data_phase.lock();
  { // data phase
    scv_tr_handle h2
      = data_gen.begin_transaction("data_phase",h);
    ...// data phase
    data_gen.end(h2,data);
  }
  read_gen.end_transaction(h, data);
  data_phase.unlock();
  return data;
}
```

Transaction Recording

```
class rw_pipelined_transactor
: public pipelined_bus_ports, public rw_task_if {
fifo_mutex address_phase;
fifo_mutex data_phase;
scv_tr_stream pipelined_stream;
scv_tr_stream addr_stream;
scv_tr_stream data_stream;
scv_tr_generator< addr_t, data_t > read_gen;
scv_tr_generator< addr_t, data_t > write_gen;
scv_tr_generator< addr_t > addr_gen;
scv_tr_generator< data_t > data_gen;
```

```
public:
SC_CTOR(rw_pipelined_transactor)
: pipelined_stream("pipelined_stream"),
addr_stream("addr_stream"),
data_stream("data_stream"),
read_gen("read",pipelined_stream, "addr","data"),
write_gen("write",pipelined_stream,"addr","data"),
addr_gen("addr",addr_stream,"addr"),
data_gen("data",data_stream, "data") { ... }
virtual data_t read( const addr_t * addr);
virtual void write( const write_t * write);
};
```