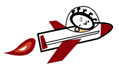
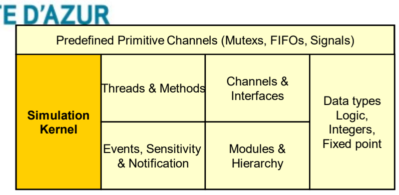
Plan



Notion of Time


Predefined Primitive Channels (Mutexs, FIFOs, Signals)
Simulation Kernel
Threads & Methods
Channels & Interfaces
Data types Logic, Integers, Fixed point
Events, Sensitivity & Notification
Modules & Hierarchy

- sc_time Data Type
- Elaboration and Simulation
- wait() Method

sc_time Type

- Goals
  - to measure time
  - to specify a time (waiting, …)
- VHDL : “time” type

Units

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

default : t1(0, SC_SEC)

```
1  sc_time measure, current, last_clk;

2  sc_time period (8.2, SC_NS);  // period = 8.2 ns

3  sc_time clk(period);  // clk = 8.2 ns

   last_clk = sc_time(2, SC_US);  // last_clk = 2 us

   measure = current - last_clk;
   if (measure > hold)
         cerr << “error: setup violation !” << endl;
```

3

sc_time Class Definition

```
enum sc_time_unit {SC_FS = 0, SC_PS,
                   SC_NS, SC_US, SC_MS, SC_SEC};
class sc_time
{
public:
    sc_time();
    sc_time( double , sc_time_unit );     } constructors
    sc_time( const sc_time& );

    sc_time& operator= ( const sc_time& );

    sc_dt::uint64 value() const;
    double to_double() const;             } converting functions
    double to_seconds() const;

    const std::string to_string() const;

    bool operator== ( const sc_time& ) const;
    bool operator!= ( const sc_time& ) const;
    bool operator< ( const sc_time& ) const;
    bool operator<= ( const sc_time& ) const;
    bool operator> ( const sc_time& ) const;      operator overloading
    bool operator>= ( const sc_time& ) const;     (Member Functions)
    sc_time& operator+= ( const sc_time& );
    sc_time& operator-= ( const sc_time& );
    sc_time& operator*= ( double );
    sc_time& operator/= ( double );

    void print( std::ostream& = std::cout ) const;
};
```

```
const sc_time operator+ ( const sc_time&, const sc_time& );
const sc_time operator- ( const sc_time&, const sc_time& );
const sc_time operator* ( const sc_time&, double );           operator overloading
const sc_time operator* ( double, const sc_time& );           (Non Member Functions)
const sc_time operator/ ( const sc_time&, double );
double operator/ ( const sc_time&, const sc_time& );
std::ostream& operator<< ( std::ostream&, const sc_time& );

const sc_time SC_ZERO_TIME;  ← equal to sc_time(0, SC_SEC) (delta delay)

void sc_set_time_resolution( double, sc_time_unit );
sc_time sc_get_time_resolution();
```

Example

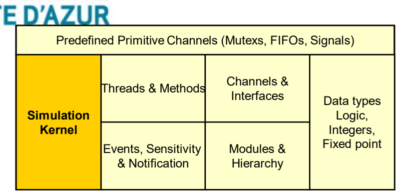
sc_time_class_definition

```
#include <systemc.h>

int sc_main(int argc, char* argv[])
{
      sc_time a = sc_time(2.5, SC_US);
      cout << "to_string() : " << a.to_string() << endl;
      cout << "to_double() : " << a.to_double() << endl;
      cout << "to_seconds() : " << a.to_seconds() << endl;
      return 0;
}
```

```
to_string() : 2500 ns
to_double() : 2.5e+006
to_seconds() : 2.5e-006
```

4


Predefined Primitive Channels (Mutexs, FIFOs, Signals)
Simulation Kernel
Threads & Methods
Channels & Interfaces
Data types Logic, Integers, Fixed point
Events, Sensitivity & Notification
Modules & Hierarchy

Notion of Time

- sc_time Data Type
- Elaboration and Simulation
- wait() Method

Methods

- **sc_start() method** : performs simulation

```
void sc_start();
void sc_start( const sc_time& );
void sc_start( double, sc_time_unit );
```

```
sc_start();                  // Run forever
sc_start(SC_ZERO_TIME);  // Run 1 delta delay
sc_start(8, SC_MS);          // Run 8 ms
```

- **sc_stop() method** : stop simulation
- **sc_time_stamp() method** : current time

```
sc_time t = sc_time_stamp() ;
```

- sc_simulation_time() method : current time as a double

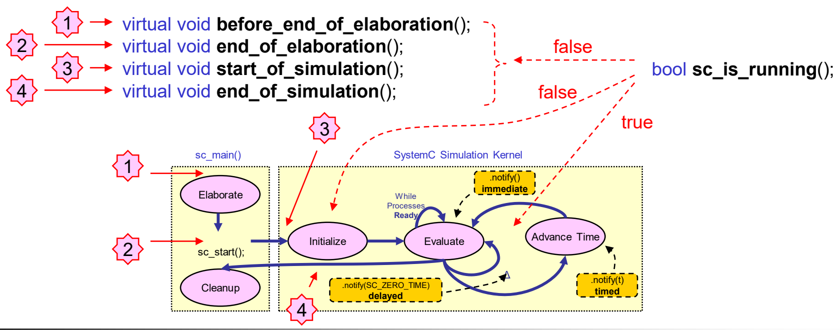
```
double t = sc_simulation_time() ;
```

- sc_set_time_resolution() method : resolution (positive power of ten)
- sc_get_time_resolution() method : get the time resolution
- sc_set_default_time_unit() method : default time unit (power of ten)
- sc_get_default_time_unit() method : get default time unit

- sc_delta_count() method : counts the number of delta cycles (return a value of uint64 type)

Elaboration and Simulation Call Back

- called by the kernel at various stages
  - elaboration
  - simulation
- member functions of class
  - sc_module
  - sc_port, sc_export (Ch10 – Communication)
  - sc_prim_channel (Ch7 – Basic Channels)



```
1 → virtual void before_end_of_elaboration();
2 → virtual void end_of_elaboration();
3 → virtual void start_of_simulation();
4 → virtual void end_of_simulation();

bool sc_is_running();
```

7

Elaboration and Simulation Call Back Example

eloborate_and_sim

```
SC_MODULE(simple_process)
{
    SC_CTOR(simple_process)
    {
        cout << "   Constructor : " << name() << endl;
        SC_THREAD(my_thread_process);
    }
process { void my_thread_process(void) {
        cout << "   my_thread_process executed within ";
        cout << name() << endl; }

    void before_end_of_elaboration()
    {
        cout << "   before_end_of_elaboration : " << name() << endl;
    }

    void end_of_elaboration() {
        cout << "   end_of_elaboration : " << name() << endl;
    }

    void start_of_simulation() {
        cout << "   start_of_simulation : " << name() << endl;
    }

    void end_of_simulation() {
        cout << "   end_of_simulation : " << name() << endl;
    }
};
```

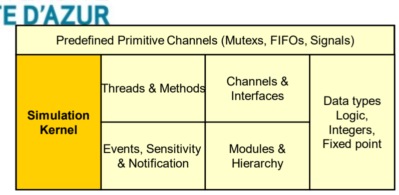
```
int sc_main(int argc, char* argv[])
{
    cout << "Start main()" << endl;
    simple_process my_instance1("my_inst1");

    cout << "Before start()" << endl;
    sc_start(100, SC_MS);  // Run simulation (100 ms)
    cout << "After start()" << endl;

    sc_stop();
    cout << "After stop()" << endl;
    return 0;
}
```

```
Start main()
   Constructor : my_inst1
Before start()
   before_end_of_elaboration : my_inst1
   end_of_elaboration : my_inst1
   start_of_simulation : my_inst1
   my_thread_process executed within my_inst1
After start()
SystemC: simulation stopped by user.
   end_of_simulation : my_inst1
After stop()
Press any key to continue
```

8


Predefined Primitive Channels (Mutexs, FIFOs, Signals)
Simulation Kernel
Threads & Methods
Channels & Interfaces
Data types Logic, Integers, Fixed point
Events, Sensitivity & Notification
Modules & Hierarchy

Notion of Time

- sc_time Data Type
- Elaboration and Simulation
- wait() Method

wait() Method

- delayed a process for specified periods of time
- used this delay to simulate delays of real activities
  - mechanical actions
  - chemical reaction times
  - signal propagation
- More on wait() (Ch5 – Concurrency)

wait(sc_time t);

wait specified amount of time

wait_method

```
void simple_process::my_thread_process(void)
{
    cout << "Now at " << sc_time_stamp() << endl;
    wait(10, SC_NS);
    cout << "Now at " << sc_time_stamp() << endl;

    sc_time t (5, SC_NS);
    t = t * 3;  // Computes delay
    cout << "delaying " << t << endl;
    wait(t);

    cout << "Now at " << sc_time_stamp() << endl;
}
```

```
Now at 0 ns
Now at 10 ns
delaying 15 ns
Now at 25 ns
```