بنام خداوند جان و خرد

هم‌طراحی سخت‌افزار نرم‌افزار

جلسه دهم: توصیف سیستم-زبان SystemC -۳

ارائه‌دهنده: آتنا عبدی
a_abdi@kntu.ac.ir

مباحث این بخش

- توصیف یک سیستم (System Specification)
- مدل‌های محاسباتی
- معماری‌ها
- زبان‌های توصیف
  - اشنایی با زبان توصیف سیستم SystemC

انواع Process از لحاظ اجرا

- Method (sc_method)
  - یکبار صدا زده می‌شود و پس از فراخوانی امکان متوقف کردن آن وجود ندارد و تا پایان اجرا می‌شود
  - مناسب برای طراحی بلوک‌های منطق ترکیبی مدار (محدود به این حالت نمی‌باشد)
- Thread (sc_thread)
  - قابلیت متوقف شدن با دستور wait() را دارد و با فعال شدن لیست حساسیت، پروسه دوباره فعال می‌شود
- Clock Thread (sc_cthread)
  - مشابه حالت قبل است و فقط حساس به لبه یک کلاک است و نیاز به تعریف لیست حساسیت نیست
  - sc_cthread (name,clock.pos());
  - متوقف شدن با wait_until (<signal condition>) و پس از پایان شرط بولی، پروسه دوباره فعال می‌شود

انواع تعریف Process در SystemC (Method)

```
#include "systemc.h"
SC_MODULE (first_counter) {
 sc_in_clk clock; // Clock input of the design
 sc_in<bool> reset; // active high, synchronous Reset input
 sc_in<bool> enable; // Active high enable signal for counter
 sc_out<sc_uint <4> > counter_out; // 4 bit vector output of the counter
//------------Local Variables Here---------------------
 sc_uint<4> count;

void incr_count () {
 // At every rising edge of clock check if reset is active
if (reset.read() == 1) {
count = 0;
counter_out.write(count);
// If enable is active, then we increment the counter
} else if (enable.read() == 1) {
count = count + 1;
counter_out.write(count);
} }
```

- مثال:
  - شمارنده رو به بالا چهاربیتی
  - دارای سیگنال ریست و enable

```
// Below functions prints value of count when ever it changes
void print_count () {
cout<<"@" << sc_time_stamp() <<
" :: Counter Value "<<counter_out.read()<<endl; }
// Constructor for the counter
SC_CTOR(first_counter) {
// Edge and level sensitive
SC_METHOD(incr_count);
sensitive << reset;
sensitive << clock.pos();
// Level Sensitive method
SC_METHOD(print_count);
sensitive << counter_out;
} // End of Constructor
 }; // End of Module counter
```

انواع تعریف Process در SystemC (Thread)

- مثال شمارنده:

```
#include "systemc.h"
SC_MODULE (first_counter) {
 sc_in_clk clock; // Clock input of the design
 sc_in<bool> reset; // active high, synchronous Reset input
 sc_in<bool> enable; // Active high enable signal for counter
sc_out<sc_uint<4>> counter_out; // 4 bit vector output of the counter
//------------Local Variables Here---------------------
 sc_uint<4> count;
 //------------Code Starts Here------------------------
 // Below function implements actual counter logic
 void incr_count () {
 // For threads, we need to have while true loop
 while (true) {
 // Wait for the event in sensitivity list to occur (In this example - positive edge of clock)
wait();
 if (reset.read() == 1) {
 count = 0;
 counter_out.write(count);
 // If enable is active, then we increment the counter
} else if (enable.read() == 1) {
 count = count + 1;
 counter_out.write(count);
 }
 }
 } // End of function incr_count
```

```
// Below functions prints value of count when ever it changes
 void print_count () {
 while (true) {
 wait();
 cout<<"@" << sc_time_stamp() <<
 " :: Counter Value "<<counter_out.read()<<endl;
 }
}
// Constructor for the counter
SC_CTOR(first_counter) {
// Edge sensitive to clock
SC_THREAD(incr_count);
sensitive << clock.pos();
// Level Sensitive to change in counter output
SC_THREAD(print_count);
sensitive << counter_out;
} // End of Constructor
}; // End of Module counter
```

انواع تعریف Process در SystemC (CThread)

- مثال شمارنده:

```
#include "systemc.h"
SC_MODULE (first_counter) {
 sc_in_clk clock; // Clock input of the design
 sc_in<bool> reset; // active high, synchronous Reset input
 sc_in<bool> enable; // Active high enable signal for counter
 sc_out<sc_uint<4> > counter_out; // 4 bit vector output of the counter
//------------Local Variables Here---------------------
 sc_uint<4> count;
//------------Code Starts Here-------------------------
 // Below function implements actual counter logic
 void incr_count () {
 // For threads, we need to have while true loop
while (true) {
// Wait for the event in sensitivity list to occur
wait();
if (reset.read() == 1) {
count = 0;
counter_out.write(count);
// If enable is active, then we increment the counter
} else if (enable.read() == 1) {
count = count + 1;
counter_out.write(count);
}
}
} // End of function incr_count
```

```
// Below functions prints value of count when ever it changes
void print_count () {
while (true) {
wait();
cout<<"@" << sc_time_stamp() <<
" :: Counter Value "<<counter_out.read()<<endl;
}
}

// Constructor for the counter
SC_CTOR(first_counter) {
// cthreads require to have thread name and triggering
// event to passed as clock object
SC_CTHREAD(incr_count, clock.pos());
// Level Sensitive to change in counter output
SC_THREAD(print_count);
sensitive << counter_out;
} // End of Constructor
}; // End of Module counter
```

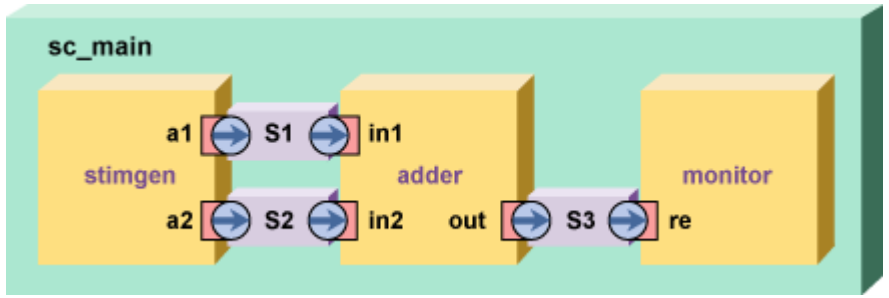
ساختار برنامه‌نویسی پایه در SystemC

- هر برنامه C/C++ نیاز به تابع main() دارد
  - در زبان SystemC: sc_main() که نقطه شروع هر کاربرد است
  - پیش از اجرای هر برنامه لازم است این تابع صدا زده شود تا
    - کرنل شبیه‌سازی و ساختارهای SystemC مقداردهی اولیه شوند
  - فراخوانی همه زیربخش‌های سیستم لازم است
    - شکل‌دهی سیستم در قالب زیرماژول‌ها و اجزا

```
int sc_main (int argc, char *argv [ ] )
    {
// body of function
return 0 ;
}
```

ساختار برنامه‌نویسی پایه در SystemC

```
int sc_main(int argc, char *argv[ ]){
// Create fifos with a depth of 10
     sc_signal<int> s1;
     sc_signal<int> s2;
     sc_signal<int> s3;

     // Module instantiations
     // Stimulus Generator
     stimgen stim("stim");
     stim(s1, s2);
     // Adder
     adder add("add");
     add(s1, s2, s3);
     // Response Monitor
     monitor mon ("mon");
     mon.re(s3);
     sc_start();
     return 0;
}
```

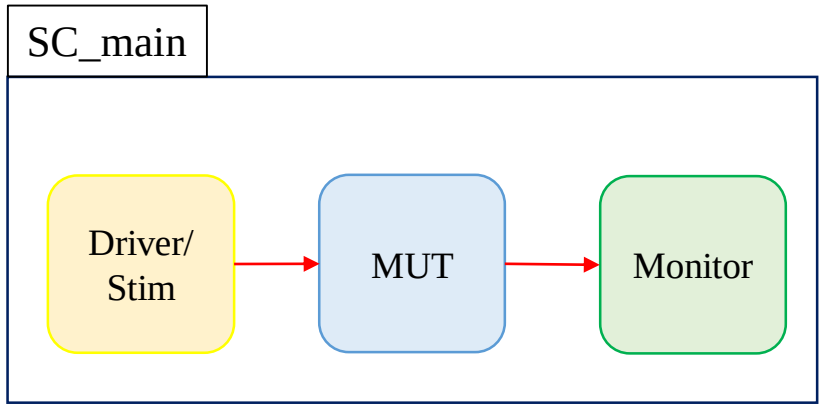


اعتبارسنجی کارکرد مدل در SystemC

- پس از ایجاد مدل لازم است اعتبارسنجی و تست روی آن داشته باشیم
- این عملیات در تابع main به دو صورت تولید کلاک و شکل‌موج انجام می‌گیرد
  - Testbench
- عملیات تست را برای ماژول تمام جمع‌کننده انجام می‌دهیم
  - تولید همه ترکیب‌های ورودی
  - اعمال ورودی‌های هر ۵ نانوثانیه

اعتبارسنجی کارکرد مدل در SystemC

- برای تست و شبیه‌سازی هر ماژول در SysyemC لازم است دو ماژول نوشته شود:
  - Driver/Stim: تولید الگوهای ورودی که در این مثال هر ۵ نانوثانیه است
    - نوع پروسه؟
  - Monitor: نمایش مقادیر پورت‌های ماژول تحت تست که در اینجا تمام جمع‌کننده است
    - با چه نرخی؟



اعتبارسنجی کارکرد مدل در SystemC

- تست ماژول تمام جمع‌کننده: (Driver/Stim)

```
// File: driver.h
#include "systemc.h"

SC_MODULE (driver) {
  sc_out<bool> d_a, d_b, d_cin;

  void prc_driver ();

  SC_CTOR (driver) {
    SC_THREAD (prc_driver);
  }
};
```

```
// File: driver.cpp
#include "driver.h"

void driver::prc_driver () {
  sc_uint<3> pattern;
  pattern = 0;

  while (1) {
    d_a = pattern[0];
    d_b = pattern[1];
    d_cin = pattern[2];
    wait (5, SC_NS);
    pattern++;
  }
}
```

اعتبارسنجی کارکرد مدل در SystemC

- تست ماژول تمام جمع‌کننده: (Monitor)

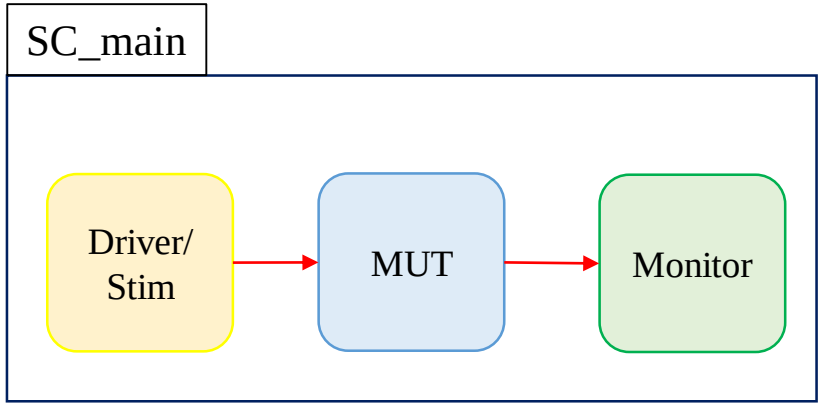
```
// File: monitor.h
#include "systemc.h"

SC_MODULE (monitor) {
  sc_in<bool> m_a, m_b, m_cin, m_sum, m_cout;

  void prc_monitor ();

  SC_CTOR (monitor) {
    SC_METHOD (prc_monitor);
    sensitive << m_a << m_b << m_cin << m_sum << m_cout;
  }
};
```

```
// File: monitor.cpp
#include "monitor.h"

void monitor::prc_monitor () {
  cout << "At time " << sc_time_stamp() << "::";
  cout << "(a, b, carry_in): ";
  cout << m_a << m_b << m_cin;
  cout << "  (sum, carry_out): " << m_sum
       << m_cout << endl;
}
```

اعتبارسنجی کارکرد مدل در SystemC

- تست ماژول تمام جمع‌کننده: (Main)

```
// File: full_adder_main.cpp
#include "driver.h"
#include "monitor.h"
#include "full_adder.h"

int sc_main(int argc, char* argv[]) {
  sc_signal<bool> t_a, t_b, t_cin, t_sum, t_cout;

  full_adder f1 ("FullAdderWithHalfAdder");
  // Connect using positional association:
  f1 << t_a << t_b << t_cin << t_sum << t_cout;

  driver d1 ("GenerateWaveforms");
  // Connect using named association:
  d1.d_a(t_a);
  d1.d_b(t_b);
  d1.d_cin(t_cin);

  monitor mo1 ("MonitorWaveforms");
  mo1 << t_a << t_b << t_cin << t_sum << t_cout;

  sc_start(100, SC_NS);

  return(0);
}
```



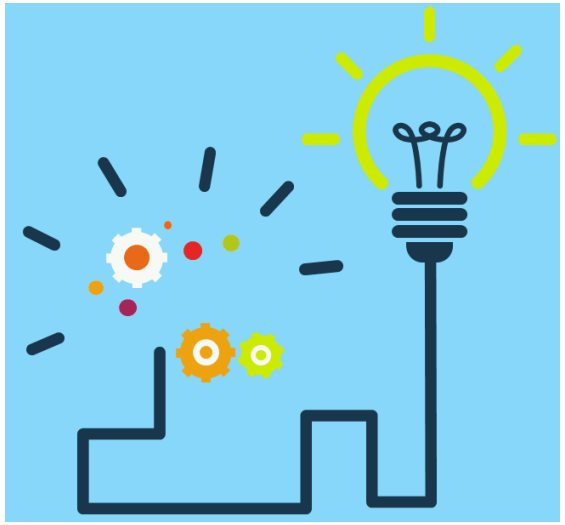
اعتبارسنجی کارکرد مدل در SystemC

- خروجی شبیه‌سازی:

```
At time 0 s::(a, b, carry_in): 000  (sum, carry_out): 00
At time 5 ns::(a, b, carry_in): 100  (sum, carry_out): 00
At time 5 ns::(a, b, carry_in): 100  (sum, carry_out): 10
At time 10 ns::(a, b, carry_in): 010  (sum, carry_out): 10
At time 15 ns::(a, b, carry_in): 110  (sum, carry_out): 10
At time 15 ns::(a, b, carry_in): 110  (sum, carry_out): 01
At time 20 ns::(a, b, carry_in): 001  (sum, carry_out): 01
At time 20 ns::(a, b, carry_in): 001  (sum, carry_out): 11
At time 20 ns::(a, b, carry_in): 001  (sum, carry_out): 10
At time 25 ns::(a, b, carry_in): 101  (sum, carry_out): 10
At time 25 ns::(a, b, carry_in): 101  (sum, carry_out): 00
At time 25 ns::(a, b, carry_in): 101  (sum, carry_out): 01
At time 30 ns::(a, b, carry_in): 011  (sum, carry_out): 01
At time 35 ns::(a, b, carry_in): 111  (sum, carry_out): 01
At time 35 ns::(a, b, carry_in): 111  (sum, carry_out): 11
```

مباحثی که این جلسه آموختیم

- توصیف سیستم و زبان
- آشنایی با زبان SystemC
- انواع پروسه
- اعتبارسنجی مدل
- خروجی متنی



مباحث جلسه آینده

- توصیف سیستم (زبان)
- آشنایی با زبان SystemC
- ارزیابی مدل و خروجی شکل موج
- مثال‌های بیشتر