Metodologie di Progettazione Hardware/Software: SystemC II parte

Livelli di astrazione

Algorithmic Model	Nessuna nozione del tempo
UnTimed Functional (UTF) model	
Timed Functional (UTF) model	Nozione del tempo
Bus Cycle Accurate (BCA) model	
Cycle Accurate (BCA) model	Accuratezza a livello di ciclo
Register Transfer Level (RTL) model	

Astrazioni high-level delle risorse HW

- Per il bassi livelli si può usare il channel sc_signal
- Per i livelli più alti si possono usare:
  - ✓ sc_fifo
  - ✓ sc_mutex
  - ✓ sc_semaphore
- Questi channel possono essere **collegati** alle porte come gli sc_signal, ma rispetto ad essi:
  - ✓ Sono più adatti per funzionalità ad alto livello
  - ✓ Non sono cycle-accurate

3

Register Transfer Level (RTL) modeling

- RTL si riferisce al livello di astrazione in cui un circuito è descritto come trasferimenti sincroni tra unità funzionali
- I trasferimenti sono coordinati da un controller e sono sincronizzati da un clock
- Il comportamento del controller è determinato in parte dalle condizioni provenienti dal datapath e in parte da segnali di ingresso esterni
- La descrizione è cycle accurate, ovvero è possibile conoscere il ciclo di clock in cui avviene ogni operazione.
- Computazione e trasferimenti si assume che vengano realizzati in un tempo nullo, regolati dal clock.

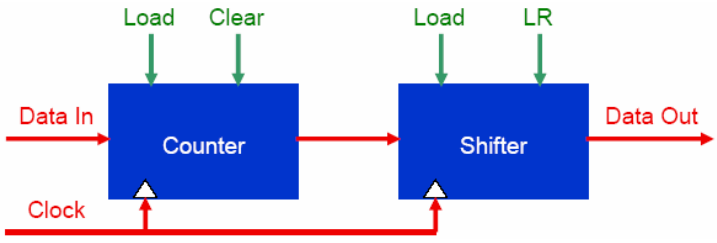
4

Register Transfer Level (RTL) modeling: Signal e Delta-cycle

- Un signal supporta il metodo di accesso request-update
- Un *read* simultaneo a un *write* legge il valore corrente e non quello che verrà scritto dalla *write*.

5

Un esempio di modello RTL



6

Un esempio di modello RTL

- Un contatore a 8 bit.
  - ✓ Questo contatore può essere aggiornato sul fronte di salta del **clk** ponendo l'ingresso **load** a 1 e ponendo un valore sull'ingresso **din**. Il contatore può essere azzerato ponendo **clear** alto.
- Un elementare shifter a 8 bit.
  - ✓ Lo shifter può essere aggiornato su fronte di salita del **clk** piazzando un valore su **din** e ponendo **load** a 1. Lo shifter sposta i dati a sinistra o a destra sulla base del valore di **LR**. Se **LR** is basso lo shifter sposterà a destra di un bit, altrimenti di uno a sinistra.
- Valori locali temporanei sono necessari poiché il valore della porte di uscita non possono essere lette.

7

Un esempio di modello RTL (1)

```
// counter.h
SC_MODULE(counter) {
 sc_in<bool> clk;
 sc_in<bool> load;
 sc_in<bool> clear;
 sc_in<sc_uint<8> > din;
 sc_out<sc_uint<8> > dout;
 unsigned int countval;

 void counting();

 SC_CTOR(counter) {
  SC_METHOD(counting);
   sensitive << clk.pos(); }
};
```

8

Un esempio di modello RTL (2)

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

void counter::counting()
{
 if (clear)
   countval = 0;
 else if (load.read())
      countval = (unsigned int)din.read();
   else
     countval++;
 dout.write((sc_uint<8>)countval);
}
```

9

Un esempio di modello RTL (3)

```
// shifter.h
SC_MODULE(shifter) {
 sc_in<sc_uint<8> > din;
 sc_in<bool> clk;
 sc_in<bool> load;
 sc_in<bool> LR; // shift left if true
 sc_out<sc_uint<8> > dout;
 sc_uint<8> shiftval;

 void shifting();

 SC_CTOR(shifter) {
    SC_METHOD(shifting);
      sensitive << clk.pos(); }
};
```

10

Un esempio di modello RTL (4)

```
// shifter.cpp
#include "shifter.h"

void shifter::shifting()
{
  if (load.read())
    shiftval = din.read();
  else if (!LR.read())      { // shift right
     shiftval.range(6,0) = shiftval.range(7,1);
     shiftval[7] = '0'; }
  else if (LR.read()) { // shift left
     shiftval.range(7,1)=shiftval.range(6,0);
     shiftval[0] = '0'; }
 dout.write(shiftval);
}
```


Behavioral-Level modeling

- È il livello in cui è definito l'ordine degli eventi degli ingressi e delle uscite, piuttosto che l'esatto ciclo di clock in cui ciò avviene
- Il livello di astrazione è tale che permettete di pensare il progetto come program flow
- Il mapping delle operazioni interne sui cicli di clock (scheduling) e sulle unità funzionali (resource binding) è irrilevante a questo livello
- Pin-accurate come RTL, ma non cycle-accurate

Behavioral-Level modeling

- Si usa il clock solo come meccanismo di ordinamento e di sincronizzazione.
- Ad un clock a quello livello possono corrispondere diversi cicli di clock a più basso livello
- Per modellare un behavioral process si può usare un CTRHEAD e i costrutti reset_signal_is e wait()

13

Un esempio di un modello Behavioral-Level

- Algoritmo di Euclide per trovare il Massimo Comune Divisore
- (MCD) di due numeri:
  - ✓ Dati a, b, con $a \geq 0$, $b > 0$,
  - ✓ If b divide a, then MCD(a, b) = b;
  - ✓ Else, MCD(a, b) = MCD(b, a mod b).

14

Un esempio di un modello Behavioral-Level (1)

```
// euclid.h
SC_MODULE (euclid) {
 sc_in_clk clock;
 sc_in<bool> reset;
 sc_in<unsigned int> a, b;
 sc_out<unsigned int> c;
 sc_out<bool> ready;

 void compute();

 SC_CTOR(euclid) {
  SC_CTHREAD(compute, clock.pos());
    reset_signal_is(reset, true); }
};
```

15

Un esempio di un modello Behavioral-Level (2)

```
// euclid.cpp
void euclid::compute()
{ unsigned int tmp_a = 0, tmp_b; // reset section
 while (true) {
  c.write(tmp_a);    // signaling output
  ready.write(true);
  wait();          // moving to next cycle
  tmp_a = a.read();  // sampling input
  tmp_b = b.read();
  ready.write(false);
  wait();          // moving to next cycle
  while (tmp_b != 0) { // computing
  unsigned int r = tmp_a;
  tmp_a = tmp_b;
  r = r % tmp_b;
  tmp_b = r;       }
 } }
```

16

UnTimed Functional (UTF) model

- Descrive la funzionalità, ma non le temporizzazioni
- Esse spesso sono implementate come “Dataflow model”: i moduli comunicano attraverso blocking FIFOs

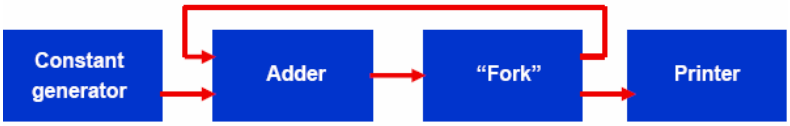
17

UnTimed Functional (UTF) model

- Nel costruire questo modello bisogna:
  - ✓ Usare moduli con SC_THREAD process
  - ✓ Fare comunicare i moduli mediante sc_fifo channel
  - ✓ Modellare i valori iniziali o scrivendo nella FIFO prima di invocare sc_start() o inserendo un modulo che genera dati prima del loro consumo

18

Un esempio di un modello UnTimed Functional (UTF)



19

Un esempio di un modello UnTimed Functional (UTF) (1)

```
// constgen.h
SC_MODULE(constgen) {
{
 sc_fifo_out<float> output;

 SC_CTOR(constgen) {
  SC_THREAD(generating());
 }
 void generating() {
   while (true) {
     output.write(0.7);
     }
  }
};
```

20

Un esempio di un modello UnTimed Functional (UTF) (2)

```
// adder.h
SC_MODULE(adder) {
{
 sc_fifo_in<float> input1, input2;
 sc_fifo_out<float> output;

  SC_CTOR(adder) {
   SC_THREAD(adding());
   }
 void adding() {
  while (true) {
   output.write(input1.read() + input2.read());
     }
 }
};
```

21

Un esempio di un modello UnTimed Functional (UTF) (3)

```
// forker.h
SC_MODULE(forker) {
{sc_fifo_in<float> input;
 sc_fifo_out<float> output1, output2;

 SC_CTOR(forker) {
   SC_THREAD(forking());
  }
  void forking() {
    while (true) {
      float value = input.read();
       output1.write(value);
       output2.write(value);
         }   }
}
```

22

Un esempio di un modello UnTimed Functional (UTF) (4)

```
// printer.h
SC_MODULE(printer) {
{ sc_fifo_in<float> input;

 SC_CTOR(printer) {
  SC_THREAD(printing());
  }
 void printing() {
  for (unsigned int i = 0; i < 100; i++) {
    float value = input.read();
    printf("%f\n", value);
  }
  return; // this indirectly stops the simulation
   }
}
```

23

Un esempio di un modello UnTimed Functional (UTF) (5)

```
// main.cpp
int sc_main(int, char**) {
constgen my_constgen("my_constgen_name");  // module
adder my_adder("my_adder_name");        // instantiation
forker my_forker("my_forker_name");
printer my_printer("my_printer_name");
sc_fifo<float> constgen_adder("constgen_adder", 5); // FIFO
sc_fifo<float> adder_fork("adder_fork", 1); //instantiation
sc_fifo<float> fork_adder("fork_adder", 1);
sc_fifo<float> fork_printer("fork_printer", 1);
```

24

Un esempio di un modello UnTimed Functional (UTF) (6)

```
fork_adder.write(2.3); // initial setup
my_constgen.output(constgen_adder);
  my_adder.input1(constgen_adder);
my_adder.input2(fork_adder);
  my_adder.output(adder_fork);
my_fork.input(adder_fork);
  my_fork.output1(fork_adder); // binding
my_fork.output2(fork_printer);
  my_printer.input(fork_printer);
sc_start(-1);
return 0;
}
```

25

Refining verso il modello Timed Functional (TF)

```
// constgen.h
SC_MODULE(constgen) {
{
 sc_fifo_out<float> output;

 SC_CTOR(constgen) {
  SC_THREAD(generating());
}
void generating() {
    while (true) {
     wait(200, SC_NS);
     output.write(0.7);
    }
 }
};
```

26

Il channel sc_mutex

- È utilizzato per regolare l'accesso di variabili condivise da diversi process
- È caratterizzata dei seguenti metodi:
  - ✓ lock() per bloccare l'accesso
  - ✓ unlock() per sbloccare l'accesso
  - ✓ trylock() per verificare se il lock è attivo

27

Esempio di uso di sc_mutex (1/3)

```
SC_MODULE(mutex_es) {
 sc_in_clk clk;
 sc_in<bool> lockA, lockB;
 sc_out<sc_uint<3> > stato;
 sc_out<bool> ackA,ackB;

sc_mutex mt;
sc_uint<16>  SV;

void proc_A();
void proc_B();

SC_HAS_PROCESS(mutex_es);
mutex_es(sc_module_name name, int i):
sc_module(name), SV(i)
 {  SC_CTHREAD(proc_A,clk.pos());
    SC_CTHREAD(proc_B,clk.pos());
    stato.initialize(0);
  }
};
```

28

Esempio di uso di sc_mutex (2/3)

```
#include "mutex_es.h"

void mutex_es::proc_A() {
 while(1){
  do wait();
  while (lockA.read()==false);
       mt.lock(); ackA = true;
       cout << "A locked" << endl;
       stato.write(1);
       SV=SV+5;
       cout << "SV: " << SV << endl;
  do wait();
  while (lockA.read()==true);
mt.unlock(); ackA = false;
     cout << "A unlocked" << endl;
     stato.write(2);
  }
 }
```

29

Esempio di uso di sc_mutex (3/3)

```
void mutex_es::proc_B() {

 while(1){
  do wait();
  while (lockB.read()==false);
      mt.lock(); ackB = true;
   cout << "B locked" << endl;
   stato.write(3);
   SV=SV+2;
       cout << "SV: " << SV << endl;
  do wait();
  while (lockB.read()==true);
     mt.unlock(); ackB = false;
     cout << "B unlocked" << endl;
        stato.write(4);
   }
 }
```

30

Il channel sc_semaphore

- E' caratterizzato da :
  - ✓ wait(), decrementa il valore del semaforo se positivo, sospende fino a quando il valore viene incrementato
  - ✓ trywait(), decrementa il valore del semaforo se positivo
  - ✓ post(), incrementa il valore del semaforo
  - ✓ get_value(), restituisce il valore del semaforo