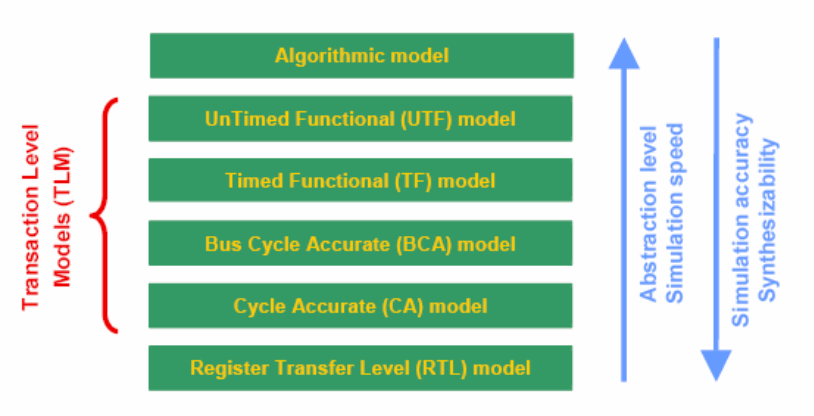
Systemy wbudowane

Wykład 9: SystemC – modelowanie na różnych poziomach abstrakcji

Poziomy abstrakcji projektowania systemów HW/SW



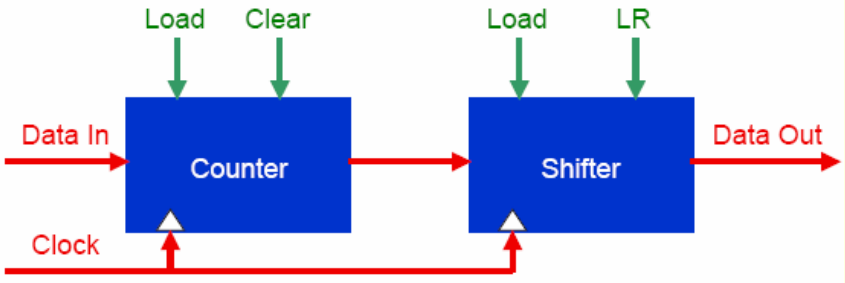
Model czasu

- Algorithmic model
- UnTimed Functional (UTF) model
- Timed Functional (TF) model
- Bus Cycle Accurate (BCA) model
- Cycle Accurate (CA) model
- Register Transfer Level (RTL) model

- No notion of time (processes and data transfers)
- Notion of time (processes and data transfers)
- Cycle accuracy, signal accuracy

Poziom RTL (1)

- RTL level: signal accurate, cycle accurate, resource accurate
- Can not use abstractions (functional units, communication infrastructures, …)



Poziom RTL (2)

- Licznik:
  - 8-bitowy
  - clear=1: licznik ← 0
  - load=1: licznik ← Data_in
  - clk= ↑ : licznik ← (licznik +1) mod 256
- Rejestr przesuwający:
  - 8-bitowy
  - load=1: licznik ← din
  - clk= ↑ :
    - LR=1: licznik ← licznik <<1
    - LR=0: licznik ← licznik >>1

Poziom RTL (3)

```
// 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;
  sc_uint<8> countval;
  void counting();
  SC_CTOR(counter) {
    SC_METHOD(counting);
    sensitive << clk.pos();
  }
};
```

Only SC_METHODs for synthesis. Tools don't like the "wait()" concept ☹

Poziom RTL (4)

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

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

Which control priorities did we choose?

Poziom RTL (5)

```
// 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();
  }
};
```

Poziom RTL (6)

```
// 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);
}
```


Bus Cycle Accurate Model

- Specyfikacja na poziomie cykli kanałów komunikacyjnych
- Nie odwzorowuje obliczeń w komponenty sprzętowe

Przykład: GCD (1)

```
#include "systemc.h"
SC_MODULE(euclid) {
  sc_in_clk            clock;
  sc_in<bool>          reset;
  sc_in<unsigned>      a, b;
  sc_out<unsigned>     c;
  sc_out<bool>         ready;
  void compute();
  SC_CTOR(euclid){
        SC_THREAD(compute);
        sensitive << clock.pos;
        reset_signal_is(reset,true);
  }
};
```


Przykład (2)

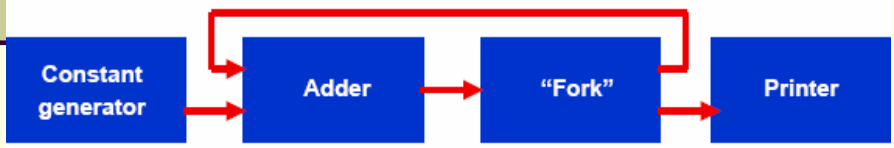
```
// euclid.cpp
void euclid::compute()
{
   unsigned int tmp_a = 0, tmp_b;                  // reset section
   while (true) {
        c.write(tmp_a);                            // signaling output
        ready.write(true);
        wait();                                    // moving to next cycle
        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;
        }
   }
}
```

% operator: how to do in HW?

Recursive: how many cycles will it take?

UnTimed Functional Model

- Very widespread modeling level
- Describes functionality, but not timing
- Most general form of UTF model: “Kahn Process Networks” (KPN)
- But often implemented as “Dataflow model”: modules communicating with each other via blocking FIFOs



Przykład (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);
        }
   }
}
```

Przykład (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());
        }
   }
}
```


Przykład (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);
        }
   }
}
```

Przykład (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
                                // (no data will be flowing any more)
   }
}
```


Przykład (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);
   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);                    // simulate "forever". Will exit
   return 0;                        // when no events are queued
}                                   // (printer exits, fifos saturate)
```

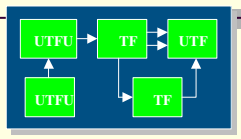
Timed Functional Model

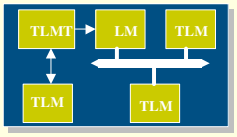
```
// constgen.h
SC_MODULE(constgen) {
{
   sc_fifo_out<float> output;

   SC_CTOR(constgen) {
        SC_THREAD(generating());
   }

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

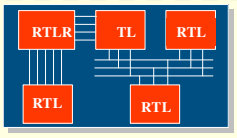
Of course a bit simplistic, but…

Modelowanie na poziomie transakcji (TLM)

- (untimed) Functional level
  - executable specification
- Transaction level
  - analyze SoC architecture, early SW development
  - estimated timing
- Pin level
  - RTL/behavioral HW design and verification







At TLM level, concerns only focus on mapping out data flow details the type of data that flows and where it is stored

Podstawowe komponenty TLM

Transaction : exchange of a data or an event between two components of a modeled and simulated system

Module : structural entity, which contain processes, ports, channels, and other modules

Channel : implements one or more interfaces, and serves as a container for communication functionality

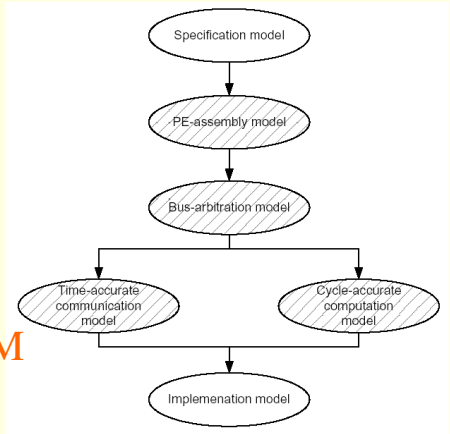
Port object through which a module can access a channel's interface.

Typy modeli

1. Specification model
2. PE*-assembly model
3. Bus-arbitration model
4. Time-accurate communication model
5. Cycle-accurate computation model

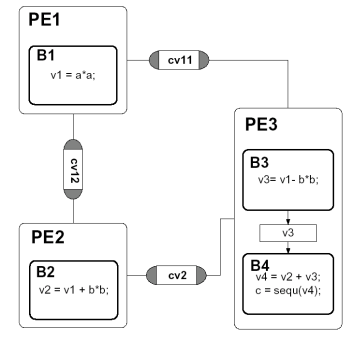
TLM

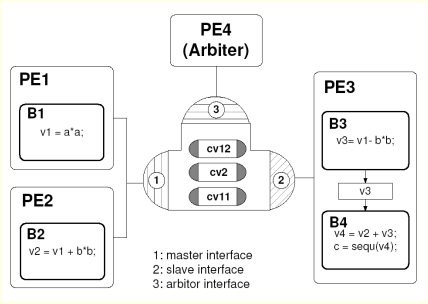
6. Implementation model



* Processing elements

PE-assembly & Bus-arbitration Models

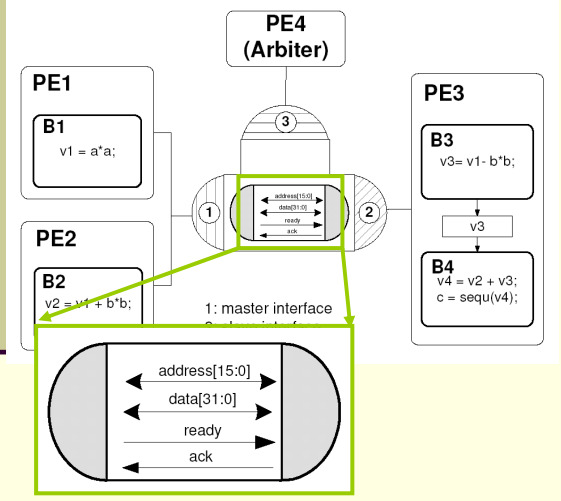




- Processing elements (PEs)
- Message-passing channels

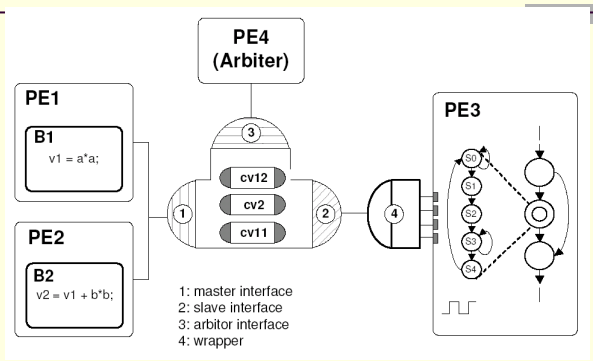
- Abstract bus channels
- Bus arbiter arbitrates bus conflict

Time-accurate Communication model



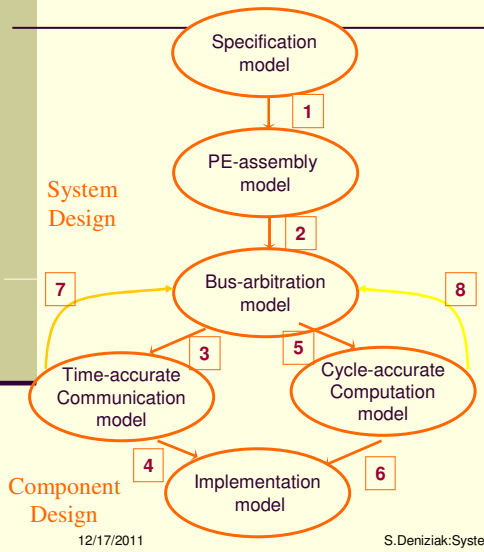
- Time/cycle accurate communication (time constraint)
- Approximate timed computation
- Protocol channel provides functions for all abstraction bus transaction

Cycle-accurate computation model



- Modeled at register-transfer level
- PE are pin accurate and execute cycle-accurately
- Wrappers convert data transfer from higher level of abstraction to lower level abstraction

Design Tasks



1. PE assembly and model generation
2. Communication exploration and bus-arbitration model generation
3. Protocol refinement and time-accurate communication model generation
4. RTL/ISS* synthesis
5. IP replacement
6. Communication synthesis and interconnect network generation
7. Accurate communication feedback
8. Accurate computation feedback

* ISS : Instruction set simulator

Źródła informacji o SystemC

- www.systemc.org
  - SystemC User Guide
- SystemC Language Reference Manual
  - http://standards.ieee.org/getieee/1666/index.html

Koniec