Systemy wbudowane

Wykład 6: Modelowanie systemów wbudowanych w SystemC, cz.I – struktura specyfikacji.

SystemC

- Standard IEEE 1666-2005
- „Open source”
  - www.systemc.org
- „nakładka” na C++
  - biblioteka dla standardowego środowiska C++
  - Sprawdzona dla Visual C++ 6.0/7.1, GNU C++ 2.95.3

Przykład specyfikacji

1. Plik nagłówkowy, np. euclid.h

```
#include "systemc.h"
SC_MODULE(euclid_gcd) {
  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_gcd){
        SC_CTHREAD(compute,CLOCK.pos());
        reset_signal_is(RESET,true);
  }
};
```


2. Plik z imlementacją, np. euclid.cpp, euclid.sc

```
#include "euclid.h"
void euclid_gcd::compute(){
 unsigned tmp_a = 0, tmp_b;
 if(RESET){
  tmp_a=0;
  READY.write(false);
 }
 while (true) {
  wait();
  C.write(tmp_a);
  READY.write(true);
  wait();
  tmp_a = A.read();
  tmp_b = B.read();
  READY.write(false);
  wait();
  while (tmp_b != 0) {
   unsigned r = tmp_a;
   tmp_a = tmp_b;
   while (r >= tmp_b)
      r = r - tmp_b;
   tmp_b = r;
  }
 }
}
```

Struktura deklaracji modułu

- deklaracje portów:
  - ***sc_in, sc_out, sc_inout, sc_clock***
- deklaracje zmiennych i sygnałów lokalnych:
  - ***sc_buffer*** - zdarzenia są związane z przypisaniem wartości
  - ***sc_signal*** - zdarzenia są związane ze zmianą wartości
- deklaracje metod
- konstruktor:
  - definicje procesów,
  - definicje zdarzeń aktywujących procesy: ***sensitive***
  - definiowanie zerowania procesów: ***reset_signal_is()***
  - inicjalizacja modułu

Typy danych

- typy języka C++,
- sc_bit → bool
- sc_logic - '0'=0=false, '1'=1=true, 'X', 'Z'
- sc_bv<n> - tablica bitów bool lub sc_logic ('0' i '1')
- sc_lv<n> - tablica bitów sc_logic
- sc_int<n> - liczby int ze znakiem do 64 bitów
- sc_uint<n> - liczby int bez znaku do 64 bitów
- sc_bigint<n> - liczby int ze znakiem o dowolnej długości
- sc_biguint<n> - liczby int bez znaku o dowolnej długości

Typy danych

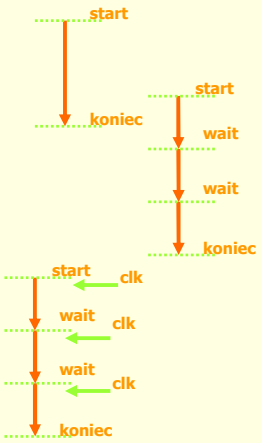
- **sc_fixed<wl,iwl,q_mode,o_mode,n_bits>, sc_fix(wl,iwl,q_mode,o_mode,n_bits)**
- **sc_ufixed<wl,iwl,q_mode,o_mode,n_bits>, sc_ufix(wl,iwl,q_mode,o_mode,n_bits)**
  - **wl: liczba bitów (wl>0)**
  - **iwl: liczba bitów części całkowitej (również iwl>wl lub iwl<0)**
  - **q_mode: tryb zaokraglania (SC_RND, SC_RND_ZERO, SC_RND_MIN_INF, SC_RND_INF, SC_RND_CONV, SC_TRN, SC_TRN_ZERO)**
  - **o_mode: tryb nadmiaru (SC_SAT, SC_SAT_ZERO, SC_SAT_SYM, SC_WRAP, SC_WRAP_SM),**
  - **n_bits: parametr dla trybu SC_WRAP (domyślne 0)**
- **sc_fixed_fast<wl,iwl,q_mode,o_mode,n_bits>, sc_fix_fast(wl,iwl,q_mode,o_mode,n_bits)**
- **sc_ufixed_fast<wl,iwl,q_mode,o_mode,n_bits>, sc_ufix_fast(wl,iwl,q_mode,o_mode,n_bits)**
  - **typy o ograniczonej precyzji (max 53 bity mantysy)**
- **sc_fxval, sc_fxval_fast - liczby o nieustalonej precyzji.**

Zmienne wielobitowe

- *sc_int<8> a,b;* a, a[0], a.range(7,4)
- *sc_int<16> c;* c=(a,b);

Procesy

- SC_METHOD(m1);
  - Wykonywany zawsze do końca
- SC_THREAD(m2);
  - Może być zawieszany instrukcją wait()
- SC_CTHREAD(m3,clk.pos())
  - Zawieszany instrukcją wait() oczekującą na sygnał zegarowy



Tworzenie procesów

- Statyczne:
  - W konstruktorze
  - funkcją SC_HAS_PROCESS
- Dynamiczne:
  - Funkcja ***sc_spawn(&m)***
    - tylko procesy SC_METHOD i SC_THREAD

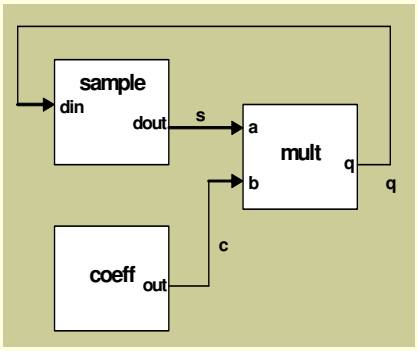
Aktywacja procesów

- SC_CTHREAD: tylko clk
- SC_METHOD, SC_THREAD:
  - ***sensitive*** << sygnał << sygnał ;
- ***next_trigger****(parametry jak w wait)*
  - czas kolejnej aktywacji aktualnego procesu
  - Tylko dla SC_METHOD

Zawieszanie procesów

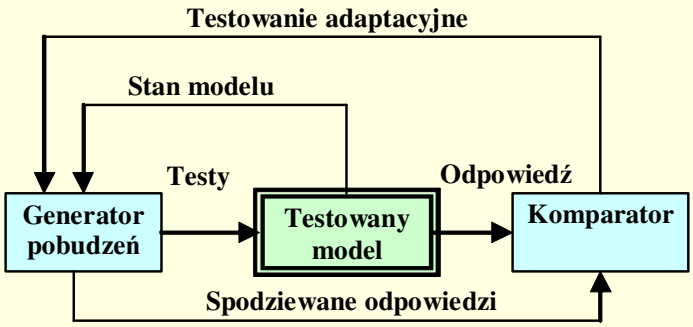
- SC_METHOD: zawsze wykonuje się do końca,
- SC_CTHREAD: ***wait(n)*** – oczekiwanie n cykli zegara (domyślnie n=1),
- SC_THREAD:
  - ***wait()***
  - ***wait(n)*** - wait() n razy
  - ***wait(event)***
  - ***wait(event1 | event2 | … | eventn)***
  - ***wait(event1 & event2 & … & eventn)***
  - ***wait(time)***
  - ***wait(time, event)***
  - ***wait(time, event1 | event2 | … | eventn)***
  - ***wait(time, event1 & event2 &…& eventn)***

Opis hierarchiczny




```
#include "systemc.h"
#include "mult.h"
#include "coeff.h"
#include "sample.h"
SC_MODULE(filter) {
  sample *s1;
  coeff *c1;
  mult *m1;
  sc_signal<sc_uint<32> > q,s,c;
  SC_CTOR(filter) {
    s1 = new sample ("s1");
    (*s1)(q,s);
    c1 = new coeff ("c1");
    c1->out(c);
    m1 = new mult ("m1");
    (*m1)(s,c,q);
  }
}
```


Testowanie modelu



Specyfikacja zadania testowego

int sc_main(argc,argv){
- utworzenie instancji modułów
- utworzenie sygnałów zegarowych:

sc_clock ck(nazwa,okres,wypełnienie,przesunięcie,wart_początkowa)
- zdefiniowanie śledzonych sygnałów:

sc_trace_file *plik=**sc_create_vcd_file(**nazwa**)**
sc_trace(plik, sygnał, nazwa**)**
- start symulacji:

sc_start() lub **sc_start(**czas**)**
- sterowanie symulacja:

sc_stop(), sc_write_comment(), report(), stop_after(), set_actions(), sc_time_stamp(),
- zakończenie symulacji:

sc_close_vcd_trace_file(plik**)**

}

Specyfikacja czasu

- Zmienne czasowe:
  - **sc_time** t(wartość, jednostka);
- Jednostki czasu:
  - **SC_FS, SC_PS, SC_NS, SC_US, SC_MS, SC_SEC**
- Rozdzielczość czasu:
  - **sc_set_time_resolution(**wartość, jednostka**)**
  - **sc_time sc_get_time_resolution()**

Przykład 1: testowanie modułu „euclid”

```
#include "systemc.h"
#include "euclid.h"
class top : public sc_module  {
public:
  SC_HAS_PROCESS(top);
  top(sc_module_name name) : div("d1"),
          clk("c1", 1, SC_NS)
  {
    SC_THREAD(t1);
    SC_THREAD(t2);
    SC_METHOD(t3);
    sensitive << c;
    div.A(a);    div.B(b);   div.C(c);
    div.CLOCK(clk);
    div.RESET(reset);
    div.READY(ready);
  }

  void t1() {
    reset.write(false);
    wait(10, SC_NS);
    a = 40;
    wait(10, SC_NS);
    a = 64;
  }
```

```
void t2() {
    wait(10, SC_NS);
    b = 10;
    wait(10, SC_NS);
    b = 8;
  }

  void t3() {
    cout << "at time: " << sc_time_stamp() << "
          output: " << c.read() << endl;
  }

  euclid_gcd div;
  sc_signal<unsigned> a, b, c;
  sc_signal<bool> reset, ready;
  sc_clock clk;
};

int sc_main (int argc , char *argv[])
{
  top top1("Top1");
  sc_start(100, SC_NS);
  cout << endl << endl;
  return 0;
}
```


Przykład 2: śledzenie sygnałów i zmiennych

```
#include "systemc.h"
SC_MODULE(some_module) {
   sc_signal<int> sig;
   int var;
   SC_CTOR(some_module) {
        sig = var = 0;
        SC_METHOD(proc);
   }
   void proc() {
     sig = - ++var;
     next_trigger(sc_time(10, SC_NS));
   }
};
```

```
int sc_main(int argc, char** argv){
    sc_trace_file* trace_file =
         sc_create_vcd_trace_file("trace");
    some_module XYZ("XYZ");
    sc_trace(trace_file, XYZ.sig, "XYZ.sig");
    sc_trace(trace_file, XYZ.var, "XYZ.var");
    sc_start(sc_time(500, SC_NS));
    sc_close_vcd_trace_file(trace_file);
    return 0;
}
```

Koniec