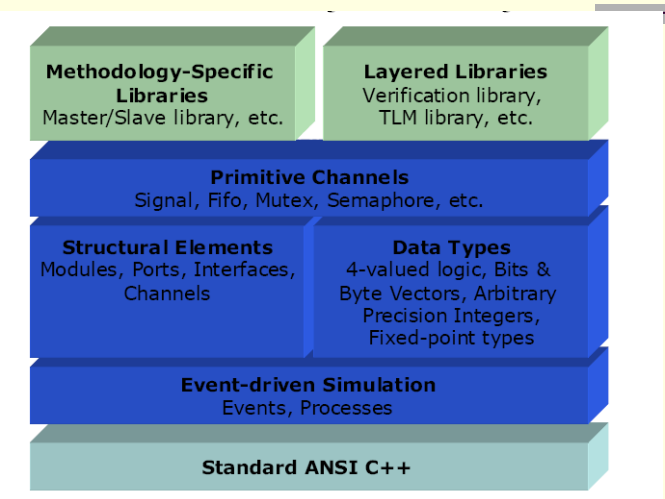
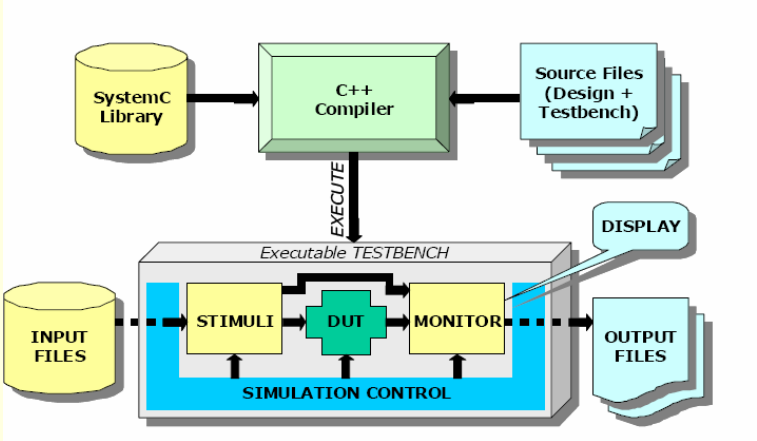
Systemy wbudowane

Wykład 8: SystemC – zasady modelowania systemów wbudowanych

Architektura środowiska SystemC



Tworzenie modelu w SystemC



Algorytm symulacji

1. Tsim=0. Inicjalizacja wszystkich procesów (z wyjątkiem procesów z wykluczoną inicjalizacją).
2. Tsim=czas wystąpienia najbliższego zdarzenia.
3. Realizacja wszystkich zdarzeń zachodzących dla Tsim.
4. Wykonanie procesów na wejściach których zaszło zdarzenie w chwili Tsim:
   - Procesy SC_METHOD: wykonywane do końca,
   - Procesy SC_THREAD wykonywane do najbliższej instrykcji wait(),
   - Wygenerowane zdarzenia są wstawiane do kolejki zdarzeń, zdarzenia wygenerowane dla chwili Tsim są wstawiane do kolejki z czasem Tsim+Δ
5. Jeśli Tsim=max czas symulacji to STOP w przeciwnym przypadku GOTO 2.

Stosowanie własnych typów

Przykład:

```
typedef struct pt {
        long info;
        int seq;
        int retry;
} packet_type;
```


```
sc_signal<packet_type>   chan;
```


Śledzenie niestandardowych danych

Plik xxx.h:

```
extern  void sc_trace(sc_trace_file *tf, const packet_type& v,
          const sc_string& NAME);
```

Plik xxx.cpp:

```
void sc_trace(sc_trace_file *tf, const packet_type& v,
          const sc_string& NAME) {
sc_trace(tf,v.info, NAME + ".info");
sc_trace(tf,v.seq, NAME + ".seq");
sc_trace(tf,v.retry, NAME + ".retry");
}
```

Operowanie na niestandardowych danych – odczyt /zapis

Operator =

const T& **operator=** (const T&);

Przykład:

```
struct packet_type {
long info;
int seq;
int retry;
inline void operator = (const
        packet_type& s)
{
        info = s.info;
        seq = s.seq;
        retry = s.retry;
}
};
```


Operowanie na niestandardowych danych – porównywanie

Operator ==

bool T::**operator==** (const T&);

Przykład:

```
struct packet_type {
long info;
int seq;
int retry;
inline bool operator == (const
        packet_type& rhs) const
{
return (rhs.info == info &&
        rhs.seq == seq &&
        rhs.retry == retry);
}
};
```

Operowanie na niestandardowych danych – funkcje diagnostyczne

Operator <<

```
std::ostream& operator<< (
        std::ostream&, const T& );
```

Metody: print(), dump()

Przykład:

```
inline  ostream&
operator << ( ostream& os, const pkt& a )
{
   os <<  a.info;
   return os;
}
```


Przykład 3: wykrywanie zmian sygnałów

```
#include "systemc.h"
SC_MODULE(toggle) {
   sc_in<bool> clk;
   sc_out<bool> output;
   bool state;
   void toggle_it() {
      output = state = !state;
   }
   SC_CTOR(toggle) {
      state = true;
      SC_METHOD(toggle_it);
      sensitive << clk.pos();
   }
};

template <class T> SC_MODULE(hs_checker) {
   sc_in<T> data;
   sc_in<bool> data_valid;
 void check_it() {
    if (data_valid.read() && !data_valid.event())
      cerr << name() << ": glitch at t="
           << sc_simulation_time() << endl;
   }
```

```
SC_CTOR(hs_checker) {
      SC_METHOD(check_it);
      sensitive << data;
   }
};

int sc_main(int argc, char** argv)
{
   sc_signal<bool> data("data");
   sc_signal<bool> data_valid("data_valid");
   sc_clock data_clock("data_clock", 10);
   sc_clock data_valid_clock("data_val_clk", 100);
   toggle toggle_data("toggle_data");
   toggle toggle_data_valid("toggle_data_valid");
   toggle_data.clk(data_clock);
   toggle_data.output(data);
   toggle_data_valid.clk(data_valid_clock);
   toggle_data_valid.output(data_valid);
   hs_checker<bool> checker("checker");
   checker.data(data);
   checker.data_valid(data_valid);
   sc_start(1000);
   return 0;
}
```

Przykład 4: sprawdzanie relacji czasowych

```
#include "systemc.h"
SC_MODULE(min_max_checker) {
   sc_in<bool> first, second;
   SC_HAS_PROCESS(min_max_checker);
   sc_time min_time_, max_time_;
    min_max_checker(sc_module_name name,
                const sc_time& min_time,
                const sc_time& max_time)
      : sc_module(name),
        min_time_(min_time),
        max_time_(max_time)
      {
         SC_THREAD(check_it);
      }

 void print_error(const char* message) {
  cerr << name() << ": protocol violation at t="
          << sc_simulation_time() << " ("
          << message << ")" << endl;
   }
```

```
void check_it() {
   while (true) {
      wait(first->posedge_event());
      if (second.read()) {
         print_error("second signal is already high");
         continue;
      }
   sc_time before = sc_time_stamp();
    wait(max_time_, first->value_changed_event() |
              second->value_changed_event());
   if (first.event()) {
      print_error("first signal went low too early");
      continue;
   }
   if (!second.event())
        print_error("max delay overrun");
        sc_time delta = sc_time_stamp() - before;
        if (delta < min_time_)
            print_error("min delay underrun");
      }
   }
};
```



```
SC_MODULE(bool_wave_gen) {
   sc_out<bool> output;
   sc_time* w_;
   bool state_;
   void toggle_it() {
      for (;;) {
        for (int i=0;;i++) {
          if (w_[i].value() == 0)  break;
          wait(w_[i]);
          output = state_ = !state_;
          cout << name() << ": output=" << state_
             << " at t="  << sc_time_stamp() << endl;
          }
      }
   }

   SC_HAS_PROCESS(bool_wave_gen);
   bool_wave_gen(sc_module_name name,
              sc_time waveform[],
              bool start_value)
      : sc_module(name), w_(waveform),
         state_(start_value)
      {
         SC_THREAD(toggle_it);
      }
};
```

```
int sc_main(int argc, char** argv) {
  sc_signal<bool> first("first");
  sc_signal<bool> second("second");
   sc_time first_waveform[] = {
    sc_time( 10, SC_NS ), // t=10
    sc_time( 10, SC_NS ),
    sc_time( 10, SC_NS ), // t=30
    sc_time( 10, SC_NS ),
    sc_time( 10, SC_NS ), // t=50
    sc_time( 10, SC_NS ),
    sc_time( 10, SC_NS ), // t=70
    sc_time( 10, SC_NS ),
    sc_time( 10, SC_NS ), // t=90
    sc_time(  2, SC_NS ), // goes down too early
    sc_time( 18, SC_NS ),
    sc_time( 10, SC_NS ),
    sc_time( 0, SC_NS )
   };
```

```
sc_time second_waveform[] = {
      sc_time( 13, SC_NS ), // OK
      sc_time(  1, SC_NS ),
      sc_time( 17, SC_NS ), // too early (31ns)
      sc_time(  1, SC_NS ),
      sc_time( 24, SC_NS ), // too late (56ns)
      sc_time(  1, SC_NS ),
      sc_time(  1, SC_NS ), // already high (58ns)
      sc_time( 22, SC_NS ), // -> back to normal
      sc_time( 14, SC_NS ), // OK (94ns)
      sc_time(  6, SC_NS ),
      sc_time( 0, SC_NS )
  };
bool_wave_gen first_wave("first_wave",
        first_waveform, false);
bool_wave_gen second_wave("second_wave",
        second_waveform, false);
```

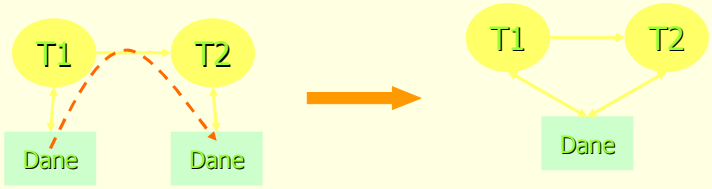
```
min_max_checker the_checker("the_checker",
   sc_time(2, SC_NS),  sc_time(5, SC_NS));
first_wave.output(first);
second_wave.output(second);
the_checker.first(first);
the_checker.second(second);
sc_start(500);
 return 0;
}
```


Tworzenie specyfikacji w SystemC (1)

1. Algorytm → zadania
   - Max. zrównoleglenie: niezależne obliczenia w różnych zadaniach
   - Zadania sekwencyjne: możliwość przetwarzania potokowego

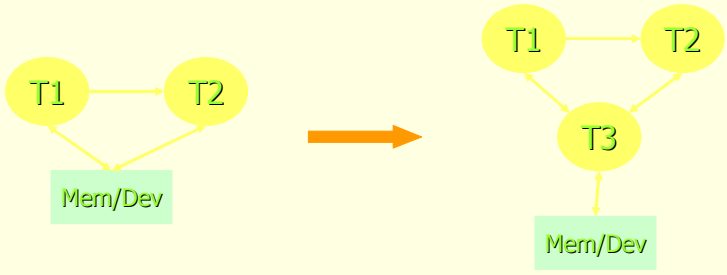
Tworzenie specyfikacji w SystemC (2)

2. Minimalizacja transmisji
   - Reorganizacja obliczeń
   - Wspólna pamięć

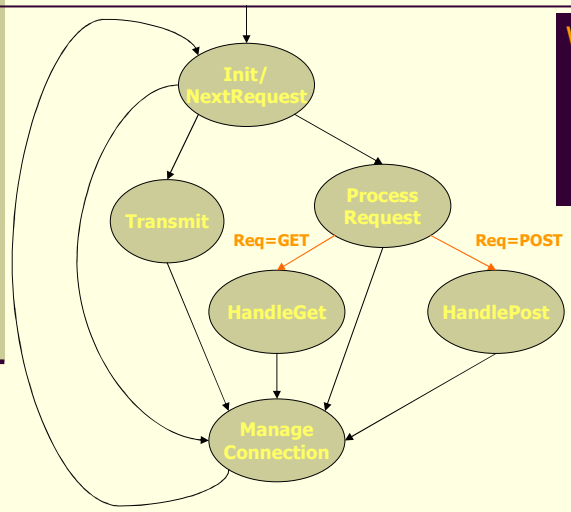


Tworzenie specyfikacji w SystemC (3)

3. Eliminacja konfliktów dostępu do wspólnych zasobów

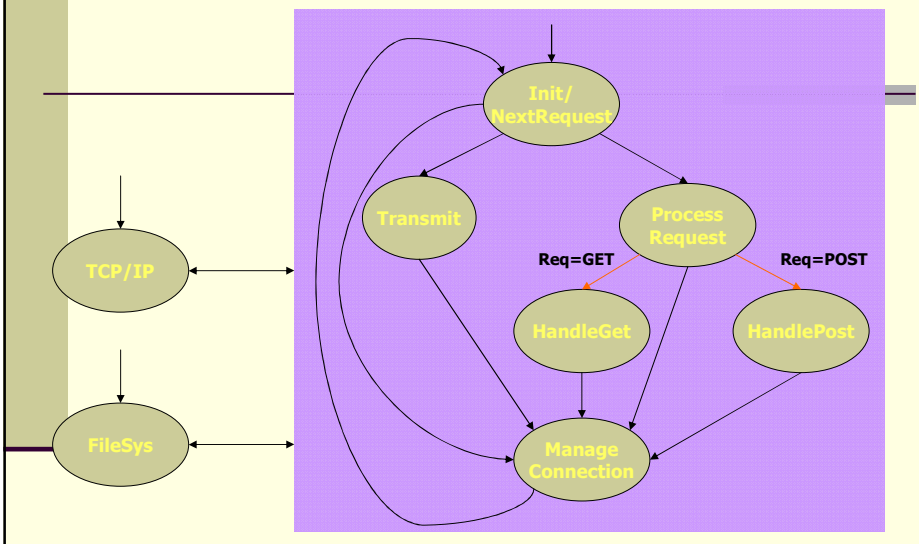


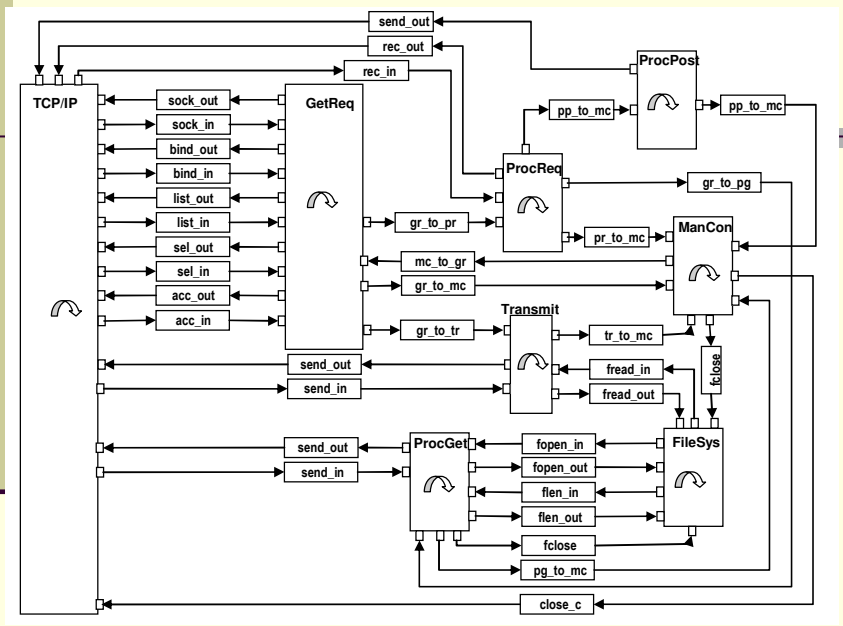
Przykład: wbudowany serwer internetowy



Wspólne zasoby:

- interfejs sieciowy
- system plików




send_out
rec_out
rec_in
ProcPost
TCP/IP
sock_out
sock_in
bind_out
bind_in
list_out
list_in
sel_out
sel_in
acc_out
acc_in
GetReq
pp_to_mc
pp_to_mc
ProcReq
gr_to_pg
gr_to_pr
pr_to_mc
ManCon
mc_to_gr
gr_to_mc
Transmit
gr_to_tr
tr_to_mc
fread_in
fread_out
fclose
send_out
send_in
send_out
send_in
ProcGet
fopen_in
fopen_out
flen_in
flen_out
fclose
pg_to_mc
FileSys
close_c

Koniec