

Modelling with SystemC

Andrej Eisfeld

May 22, 2011

Inhaltsverzeichnis

Grundlagen

Die eigene CPU

Synthese

Fakten

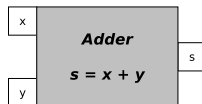
- Open Source C++ Klassenbibliothek
- Mehrere Versionen seit 1997
 - Release 1.0: Fokus auf RTL-Modellierung
 - Release 2.0.X: Systemmodellierung hinzugefügt
 - Standardisiert als IEEE std 1666
 - Künftiger Release 3: RTOS Modellierung

SystemC – Komponenten

- Module (SC_MODULE)
- Ports
 - sc_in, sc_out, scinout
 - sc_fifo_in, sc_fifo_out
- Prozesse
 - SC_METHOD
 - SC_THREAD
 - SC_CTHREAD
- Kanäle
 - sc_signal
 - sc_buffer
- Datentypen

Ein kleines Beispiel: SC_METHOD

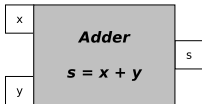
```
1 // file adder.h
2 #include "systemc.h"
3 SC_MODULE(Adder)
4 {
5     sc_in<int> x;
6     sc_in<int> y;
7     sc_out<int> s;
8     void add();
9     SC_CTOR(Adder)
10    {
11        SC_METHOD(add);
12        sensitive << x << y;
13    }
14 };
```



Ein kleines Beispiel: SC_METHOD

```

1 // file adder.h
2 #include "systemc.h"
3 SC_MODULE(Adder)
4 {
5     sc_in<int> x;
6     sc_in<int> y;
7     sc_out<int> s;
8     void add();
9     SC_CTOR(Adder)
10    {
11        SC_METHOD(add);
12        sensitive << x << y;
13    }
14 };
  
```



```

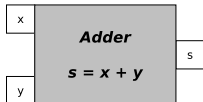
1 // file adder.cpp
2 #include "adder.h"
3 void Adder::add()
4 {
5     s = x + y;
6 }
  
```

Alternativ: `s.write(x.read() + y.read());`

Ein kleines Beispiel: SC_THREAD

```

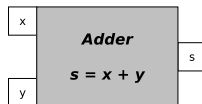
1 // file adder.h
2 #include "systemc.h"
3 SC_MODULE(Adder)
4 {
5     sc_in<int> x;
6     sc_in<int> y;
7     sc_out<int> s;
8     void add();
9     SC_CTOR(Adder)
10    {
11        SC_THREAD(add);
12        sensitive << x << y;
13    }
14 };
  
```



Ein kleines Beispiel: SC_THREAD

```

1 // file adder.h
2 #include "systemc.h"
3 SC_MODULE(Adder)
4 {
5     sc_in<int> x;
6     sc_in<int> y;
7     sc_out<int> s;
8     void add();
9     SC_CTOR(Adder)
10    {
11        SC_THREAD(add);
12        sensitive << x << y;
13    }
14 };
  
```

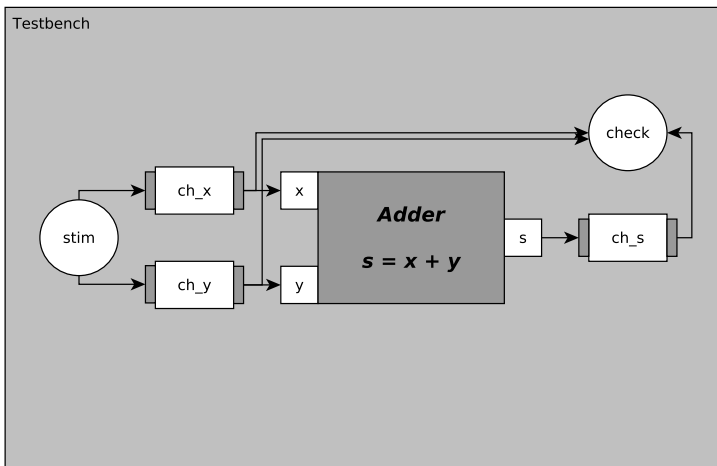


```

1 // file adder.cpp
2 #include "adder.h"
3 void Adder::add()
4 {
5     for(;;) {
6         wait();
7         s = x + y;
8     }
9
10 }
  
```

SC_METHOD	SC_THREAD
Ausführung, wenn Aktivierungsbedingung eintritt (sensitive)	Einmalige Ausführung bei Simulationsbeginn
Blockierende Anweisungen verboten	Oft blockierende Anweisungen
	Beinhaltet oft Endlosschleife

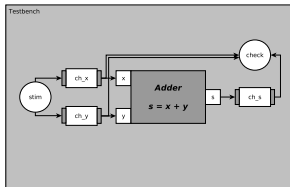
Ein kleines Beispiel: Testbench



Ein kleines Beispiel: Testbench

```

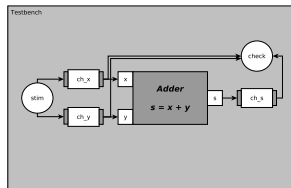
1 SC_MODULE(Testbench)
2 {
3   sc_signal<int> ch_x, ch_y, ch_s;
4   Adder dut;
5
6   void stim(); // stimuli process
7   void check(); // checking process
8
9   SC_CTOR(Testbench)
10  : dut("dut"), ch_x("ch_x")
11  ch_y("ch_y"), ch_s("ch_s")
12  {
13    SC_THREAD(stim);
14    SC_METHOD(check);
15    sensitive << ch_s;
16  }
17 };
  
```



Ein kleines Beispiel: Testbench

```

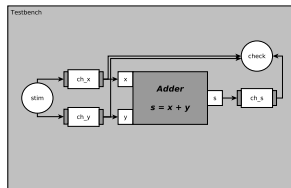
1 SC_CTOR(Testbench)
2 : dut("dut"), ch_x("ch_x")
3 {
4   SC_THREAD(stim);
5   SC_METHOD(check);
6   sensitive << ch_s;
7
8   dut.x(ch_x);
9   // port x of dut bound to ch_x
10  dut.y(ch_y);
11  // port y of dut bound to ch_y
12  dut.s(ch_s);
13  // port s of dut bound to ch_s
14 }
  
```



Ein kleines Beispiel: Testbench

```

1 #include "testbench.h"
2 void Testbench::stim() //SC_THREAD
3 {
4     ch_x = 3; ch_y = 4;
5     wait(10, SC_NS); // wait 10 ns
6     ch_x = 7; ch_y = 0;
7     wait(10, SC_NS);
8 }
9 void Testbench::check() //SC_METHOD
10 {
11     cout << ch_x << ch_y << ch_s <<
12         endl;
13     if(ch_s != ch_x+ch_y) sc_stop();
14     else cout << "-> OK" << endl;
15 }
  
```



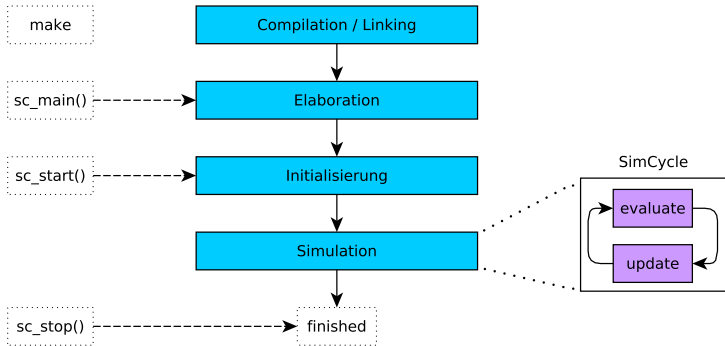
Ein kleines Beispiel: Main

```
1 // file main.cpp
2 int sc_main(int argc, char *argv[]) // C++ main()
3 {
4     Testbench tb("tb");
5     sc_start();
6     cout << "simulation finished" << endl;
7 }
```

Ein kleines Beispiel: Main

```
1 // file main.cpp
2 #include "testbench.h"
3 int sc_main(int argc, char *argv[])
4 {
5     Testbench tb("tb");
6
7     sc_trace_file *handle; // file handle declaration
8     handle = sc_create_vcd_trace_file("waveforms");
9     sc_trace(handle, tb.ch_x, "ch_x");
10    sc_trace(handle, tb.ch_y, "ch_y");
11    sc_trace(handle, tb.ch_s, "s");
12
13    sc_start();
14    cout << "simulation finished" << endl;
15    sc_close_vcd_trace_file(handle);
16 }
```

Phasen in SystemC



CPU-Aufbau

- Single Instruction Computer (SIC)
- Harvard-Architektur: je ein Modul für
 - Codespeicher
 - Datenspeicher
- State-Machine mit 2 Zuständen
- Program Counter

Unsere Single-Instruction

Instruction: SuBtract, jump if Negative (**SBN**):

- $A = A - B$
- *if*(($A - B < 0$) *jump* C instructions

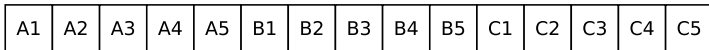
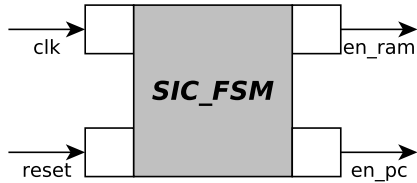
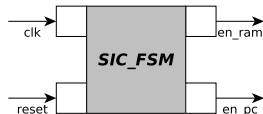


Figure: Befehlsaufbau

SIC_FSM

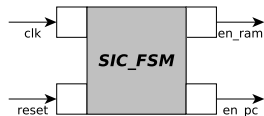


```
1 enum state { FETCH, WRITE_BACK };
2
3 SC_MODULE(SIC_FSM) {
4     sc_in<bool> clk, reset;
5     sc_out<bool> en_pc_o, en_ram_o;
6     sc_signal<state> c_state, n_state;
7
8     void updState();
9     void getNextState();
10
11     SC_CTOR(SIC_FSM) {
12         SC_METHOD(updState);
13         sensitive << clk.pos() << reset;
14         SC_METHOD(getNextState);
15         sensitive << current_state;
16     }
17 };
```



```

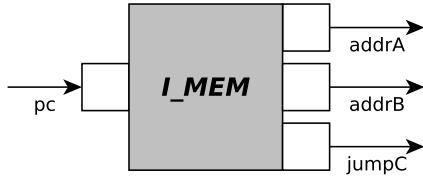
1 void SIC_FSM::getNextState() {
2     switch (c_state) {
3         case FETCH:
4             en_pc_o.write( false );
5             en_ram_o.write( false );
6             n_state = WRITE_BACK;
7             break;
8         case WRITE_BACK:
9             en_pc_o.write( true );
10            en_ram_o.write( true );
11            n_state = FETCH;
12            break;
13     }
14 }
```



```

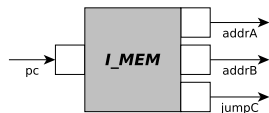
1 void SIC_FSM::updState() {
2     if( reset.read() )
3         c_state = FETCH;
4     else
5         c_state = n_state;
6 }
```

I_MEM



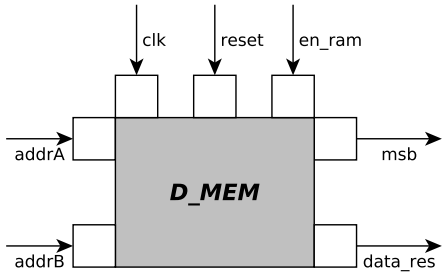
```

1 SC_MODULE(I_MEM) {
2     sc_in<sc_uint<5>> pc_i;
3     sc_out<sc_uint<5>> addrA_o;
4     sc_out<sc_uint<5>> addrB_o;
5     sc_out<sc_int<5>> jumpC_o;
6     sc_int<8> mem[32];
7
8     void readInstruction();
9
10    SC_CTOR(I_MEM) {
11        SC_METHOD(readInstruction);
12        sensitive << pc_i;
13    }
14 };
  
```

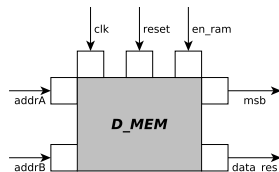


```
1 void I_MEM :: readInstruction() {
2
3     sc_uint<15> mem[32];
4
5     // fill instruction memory
6
7     addrA_o.write( mem[pc_i.read()].range(14, 10) );
8     addrB_o.write( mem[pc_i.read()].range(9, 5) );
9     jumpC_o.write( mem[pc_i.read()].range(4, 0) );
10 }
```

D_MEM

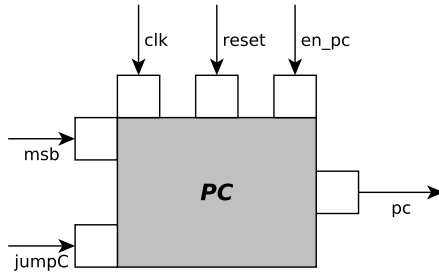


```
1 SC_MODULE(D_MEM) {  
2  
3     sc_in<bool> clk , reset , en_ram ;  
4     sc_in<sc_uint<5>> a ;  
5     sc_in<sc_uint<5>> b ;  
6  
7     sc_out<bool> msb_o ;  
8     sc_out<sc_int<8>> data_res_o ;  
9  
10    sc_int<8> mem[32];  
11  
12    void doJob();  
13  
14    SC_CTOR(D_MEM) {  
15        SC_METHOD(doJob);  
16        sensitive << clk.pos() << reset ;  
17    }  
18 };
```

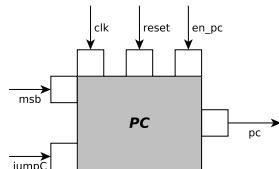


```
1 void D_MEM :: doJob() {
2   sc_int<8> res;
3   if (reset.read()) {
4     for (int i=0; i<32; i++) {
5       mem[i] = 0;
6     }
7     msb_o.write(false);
8     data_res_o.write(0);
9     mem[2] = 1;
10  }
11  else {
12    res = mem[a.read()] - mem[b.read()];
13    if (en_ram.read()) {
14      mem[a_i.read()] = res;
15    }
16    msb_o.write( res[7] );
17    data_res_o.write( res );
18  }
19 }
```

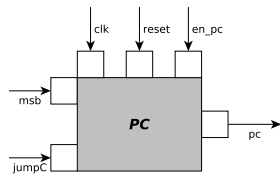
Program Counter



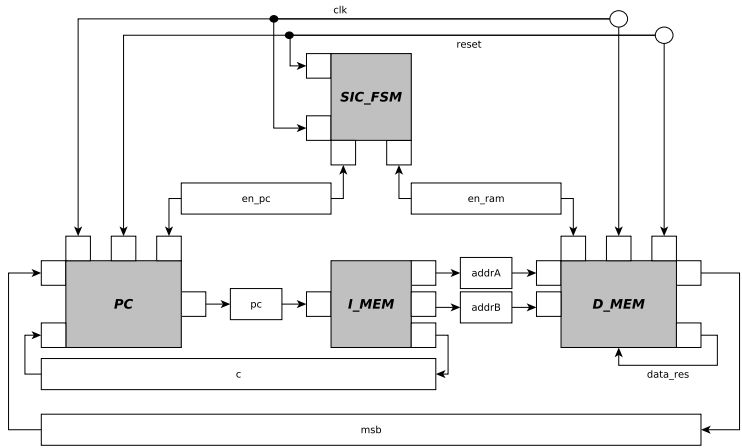
```
1 SC_MODULE(PC) {
2     sc_in<bool> reset , clk , en_pc;
3     sc_in<bool> msb_i;
4     sc_in< sc_int<5> > c_i;
5     sc_out< sc_uint<5> > pc_o;
6     sc_signal<sc_uint<5>> pc;
7
8     void calculatePC ();
9     void assign_PCout ();
10
11     SC_CTOR(PC) {
12         SC_METHOD(calculatePC);
13         sensitive << clk.pos() << reset;
14         SC_METHOD(assign_PCout);
15         sensitive << pc_v;
16     }
17 };
```



```
1 void PC::calculatePC() {  
2  
3     // check en, reset  
4     if (msb_i.read()) {  
5         pc.write( pc.read() + c_i.read  
6                 ());  
7     }  
8     else {  
9         pc.write( pc.read() + 1);  
10    }  
11 };  
12 void PC::assign_P Cout() {  
13     pc_o.write(pc);  
14 }
```



SIC_TOP





HARDWARE/SOFTWARE-COSIMULATION

Software: Fibonacci-Folge

- SBN mem[0] mem[0] 1 // reset
- SBN mem[0] mem[2] 1
- mem[0] = temp. Speicher
- mem[1] = temp. Speicher
- mem[2] = letzte Fibonacci-Zahl
- mem[3] = aktuelle Fib.-Zahl

Software: Fibonacci-Folge

- SBN mem[0] mem[0] 1 // reset
- SBN mem[0] mem[2] 1
- SBN mem[0] mem[3] 1 // nächste Zahl(neg)
- mem[0] = temp. Speicher
- mem[1] = temp. Speicher
- mem[2] = letzte Fibonacci-Zahl
- mem[3] = aktuelle Fib.-Zahl

Software: Fibonacci-Folge

- SBN mem[0] mem[0] 1 // reset
- SBN mem[0] mem[2] 1
- SBN mem[0] mem[3] 1 // nächste Zahl(neg)
- SBN mem[1] mem[3] 1 // alten Wert sichern(neg)
- mem[0] = temp. Speicher
- mem[1] = temp. Speicher
- mem[2] = letzte Fibonacci-Zahl
- mem[3] = aktuelle Fib.-Zahl

Software: Fibonacci-Folge

- SBN mem[0] mem[0] 1 // reset
- SBN mem[0] mem[2] 1
- SBN mem[0] mem[3] 1 // nächste Zahl(neg)
- SBN mem[1] mem[3] 1 // alten Wert sichern(neg)
- SBN mem[2] mem[2] 1 // reset
- mem[0] = temp. Speicher
- mem[1] = temp. Speicher
- mem[2] = letzte Fibonacci-Zahl
- mem[3] = aktuelle Fib.-Zahl

Software: Fibonacci-Folge

- SBN mem[0] mem[0] 1 // reset
- SBN mem[0] mem[2] 1
- SBN mem[0] mem[3] 1 // nächste Zahl(neg)
- SBN mem[1] mem[3] 1 // alten Wert sichern(neg)
- SBN mem[2] mem[2] 1 // reset
- SBN mem[2] mem[1] 1 // alter Wert
- mem[0] = temp. Speicher
- mem[1] = temp. Speicher
- mem[2] = letzte Fibonacci-Zahl
- mem[3] = aktuelle Fib.-Zahl

Software: Fibonacci-Folge

- SBN mem[0] mem[0] 1 // reset
- SBN mem[0] mem[2] 1
- SBN mem[0] mem[3] 1 // nächste Zahl(neg)
- SBN mem[1] mem[3] 1 // alten Wert sichern(neg)
- SBN mem[2] mem[2] 1 // reset
- SBN mem[2] mem[1] 1 // alter Wert
- SBN mem[3] mem[3] 1 // reset
- mem[0] = temp. Speicher
- mem[1] = temp. Speicher
- mem[2] = letzte Fibonacci-Zahl
- mem[3] = aktuelle Fib.-Zahl

Software: Fibonacci-Folge

- SBN mem[0] mem[0] 1 // reset
- SBN mem[0] mem[2] 1
- SBN mem[0] mem[3] 1 // nächste Zahl(neg)
- SBN mem[1] mem[3] 1 // alten Wert sichern(neg)
- SBN mem[2] mem[2] 1 // reset
- SBN mem[2] mem[1] 1 // alter Wert
- SBN mem[3] mem[3] 1 // reset
- SBN mem[3] mem[0] 1 // neuer Wert
- mem[0] = temp. Speicher
- mem[1] = temp. Speicher
- mem[2] = letzte Fibonacci-Zahl
- mem[3] = aktuelle Fib.-Zahl

Software: Fibonacci-Folge

- SBN mem[0] mem[0] 1 // reset
- SBN mem[0] mem[2] 1
- SBN mem[0] mem[3] 1 // nächste Zahl(neg)
- SBN mem[1] mem[3] 1 // alten Wert sichern(neg)
- SBN mem[2] mem[2] 1 // reset
- SBN mem[2] mem[1] 1 // alter Wert
- SBN mem[3] mem[3] 1 // reset
- SBN mem[3] mem[0] 1 // neuer Wert
- Jump zur ersten Anweisung
- mem[0] = temp. Speicher
- mem[1] = temp. Speicher
- mem[2] = letzte Fibonacci-Zahl
- mem[3] = aktuelle Fib.-Zahl

Software: Fibonacci-Folge

- SBN mem[0] mem[0] 1 // reset
 - SBN mem[0] mem[2] 1
 - SBN mem[0] mem[3] 1 // nächste Zahl(neg)
 - SBN mem[1] mem[3] 1 // alten Wert sichern(neg)
 - SBN mem[2] mem[2] 1 // reset
 - SBN mem[2] mem[1] 1 // alter Wert
 - SBN mem[3] mem[3] 1 // reset
 - SBN mem[3] mem[0] 1 // neuer Wert
 - Jump zur ersten Anweisung
- mem[0] = temp. Speicher
 - mem[1] = temp. Speicher
 - mem[2] = letzte Fibonacci-Zahl
 - mem[3] = aktuelle Fib.-Zahl

Ergebnis der Simulation

0x91f95c012469cf51b3f2301676d57142af0216b58586ffbf
ve71423398033050ff6e792fe74dd0378e5ae71b340102babf
12e11a58d527f177fe2fa327e011d8

738. Fibonacci-Zahl

SystemC - to - Verilog

- Synopsys Cocentric SystemC Compiler (kommerziell)
- Cadence Catapult Compiler (kommerziell)
- GNU SystemC Compiler (gsc)
- SystemC to Verilog Synthesizable Subset Translator (sc2v)

Aktueller Synthesestand

- Keine Gleitkommatentypen !!
- SystemC-Datentypen synthetisierbar!!
- Kein SC_THREAD !!
- Keine sc_main !!
- Kein Typecast (bei gsc) !!

Literatur

- Vorlesung: Modelling, Simulation and Specification
Uni Stuttgart: INFOTECH-Master
- Structured Computer Organization, 5th. ed
Andrew S. Tanenbaum