

Synthese Eingebetteter Systeme

Sommersemester 2011

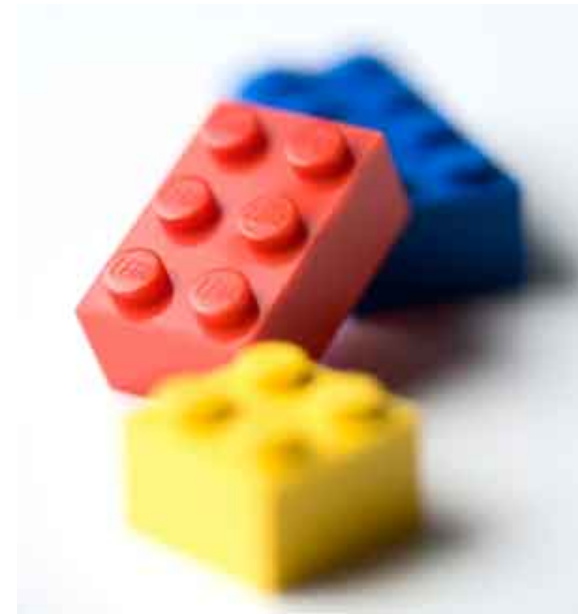
7 – SystemC-Details und Testbenches

Michael Engel
Informatik 12
TU Dortmund

2011/05/02

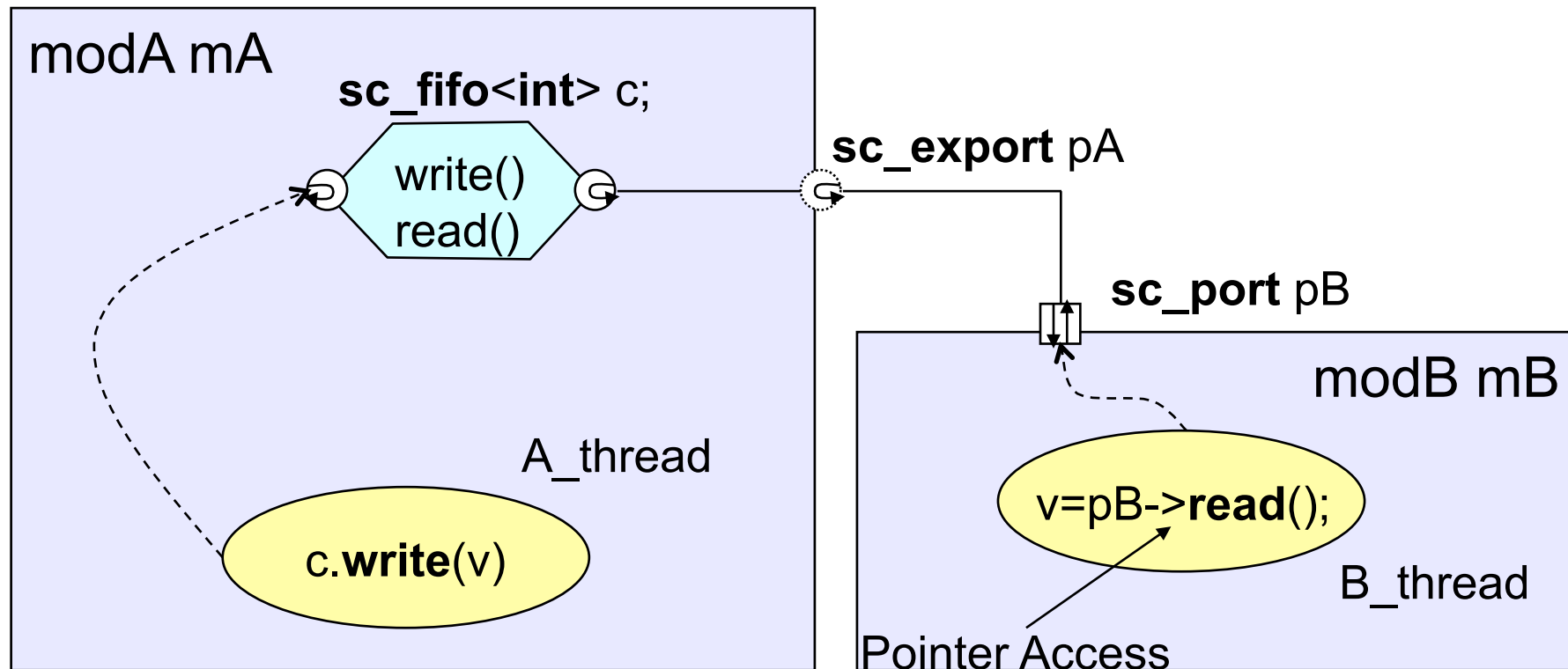
SystemC-Details

- sc_export
- Clocks



sc_export (ab SystemC 2.1)

- Neue Art von Port
 - Ähnlich Standard-Port, Channel innerhalb v. Modulen
 - Ermöglicht einfachere Verbindungen:



Syntax von `sc_export`

- **Syntax:**

```
sc_export<interface> portname;
```

Channel-Verbindungen jetzt innerhalb des Moduls:

```
SC_MODULE(modulename) {  
    sc_export<interface> portname;  
    channel c_instance;  
    SC_CTOR(modulename) {  
        portname(c_instance);  
    }  
};
```

Beispiel: Ausgabe umschalten

```
SC_MODULE(clock_gen) {
    sc_export<sc_signal<bool> > clock_xp;
    sc_signal<bool> oscillator;
    SC_CTOR(clock_gen) {
        SC_METHOD(clock_method);
        clock_xp(oscillator); // Verbinde sc_signal channel mit export clock_xp
        oscillator.write(false);
    }
    void clock_method() {
        oscillator.write(!oscillator.read());
        next_trigger(10,SC_NS);
    };
};

#include "clock_gen.h"
clock_gen clock_gen_i;
collision_detector collision_detector_i;
// Verbinde clock
collision_detector_i.clock(clock_gen_i.clock_xp);
```

Einschränkungen



- **sc_export** kann nicht in einer statischen Sensitivitätsliste verwendet werden
 - Folgendes ist aber zulässig:
wait(xportname->event()).
- Es existieren keine Arrays von Verbindungen wie bei **sc_port**.

Clocks

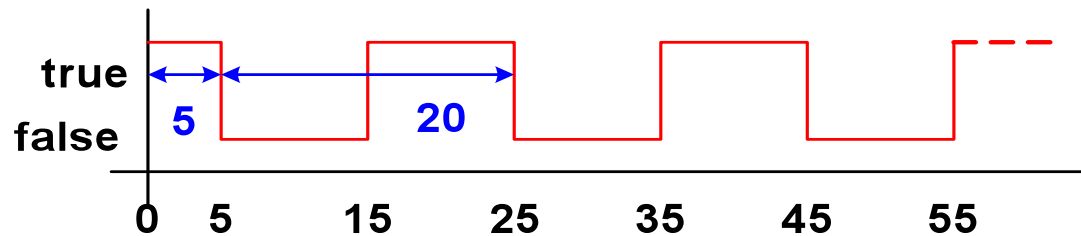


- SystemC besitzt den speziellen Channel **sc_clock**
 - Implementiert das **sc_signal_in_if<>**-Interface.
- Clocks erzeugen Timing-Signale
 - Verwendet, um Events zu synchronisieren
- Mehrere Clocks mit beliebigen Phasen sind möglich
- **Syntax:**
 - **sc_clock** *clk*("clk", *period*, *duty*, *start*, *pos_first*);
- mit:
 - **sc_time/double** *period*: Δ zwischen zwei aufeinander folgenden Taktflanken in gleicher Richtung,
 - **double** *duty*=0.5: % der Periode, in der Clock "True" ist
 - **sc_time/double** *start* = 0: Zeitpunkt der ersten Flanke
 - **boolean** *pos_first*=**true**: ist erste Flanke positiv?

Standardwerte

Clocks: Beispiele

Example: `sc_clock my_clk ("CLK", 20, 0.5, 5, false);`



```
sc_time t(10,SC_NS), t2 (5, SC_NS);  
sc_clock clk1("clk1", t, 0.5, t2); // 10 ns Periode, 50% duty, Start @ 5ns.
```

```
sc_clock clk2("clk2"); // Periode=1, 50% duty, start @ 0ns; 1st Flanke pos.
```

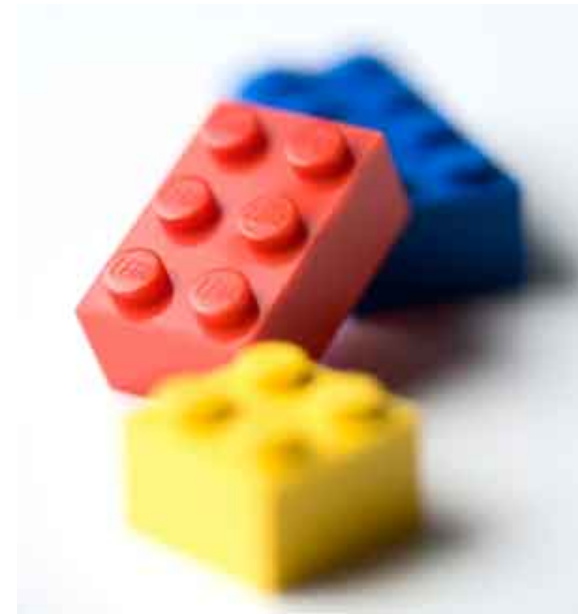
```
sc_clock clk3("clk3",20, 0.1,0, false); // Periode?20, 10% duty, Start@0ns
```

Export von Clocks aus Modulen

```
SC_MODULE(CLOCK_GEN) {
    sc_port<sc_signal_out_if<bool> > clk_p1;
    sc_export<sc_signal_in_if<bool> > clk_p2; // SystemC 2.1
    sc_clock clk1;
    sc_clock clk2;
    SC_CTOR(clock_gen)
        : clk1("clk1",4,SC_NS), clk2("clk2",6,SC_NS)
    {
        SC_METHOD(clk1_method);
        sensitive << clk1;
        clk_p2(clk2); // schneller
    }
    void clk1_method() {
        clk_p1->write(clk1);
    }
};
```

SystemC-Testbenches

- Dynamische Prozesse
- Debugging und Signalverfolgung
- Testbenches



Dynamische Prozesse (ab SystemC 2.1)

- Funktionen und Methoden können als dynamische Prozesse verwendet werden

Voraussetzung:

```
#define SC_INCLUDE_DYNAMIC_PROCESSES
```

Beispiele für als Prozesse verwendbare Funktionen:

```
void inject(); // normale Fkt. ohne Argumente/Rückgabewert  
int count_changes(sc_signal<int>& sig) // normale Fkt.  
bool TestChan::Track(sc_signal<packet>& pkt); // Methode  
bool TestChan::Errors(int maxwarn, int maxerr); // Methode
```

Syntax zur Erzeugung dynamischer Prozesse

- Registrierung dynamischer Prozesse ohne Rückgabewert

- **sc_process_handle** *hname* = **sc_spawn** (
 /* void*/ **sc_bind**(&*funcName*, *ARGS*),
 processName,
 SpawnOptions
);

Standard:
call by value

- **sc_process_handle** *hname* = **sc_spawn**(
 /*void*/ **sc_bind**(&*methName*, *object*, *ARGS*)
 processName,
 SpawnOptions
);

Stackgröße,
don't_initialize,
sensivity

Referenz auf
aufrufendes Modul,
z.B. **this**

Debugging und Signalverläufe



- SystemC hat keine Anzeige für Signalverläufe
- Erzeugung von Standard-VCD-Dateien
 - VCD = “value change dump”
- SystemC unterstützt auch proprietäre Formate

```
sc_tracefile* tracefile;  
tracefile = sc_create_vcd_trace_file(tracefile_name);  
if (!tracefile) cout << "There was an error" << endl;  
...  
sc_trace(tracefile, signal_name, "signal_name");  
...  
sc_start();  
...  
sc_close_vcd_trace_file(tracefile);
```

Debugging und Signalverläufe (2)

- Signalname muss vor Aufruf von **sc_trace** deklariert werden
- Variablen, Signale und Ports können verfolgt werden
- Dateierweiterung (z.B. “.vcd”) muss explizit angegeben werden
- Tracing kann in Modulkonstruktoren definiert werden

Debugging und Signalverläufe: Beispiel

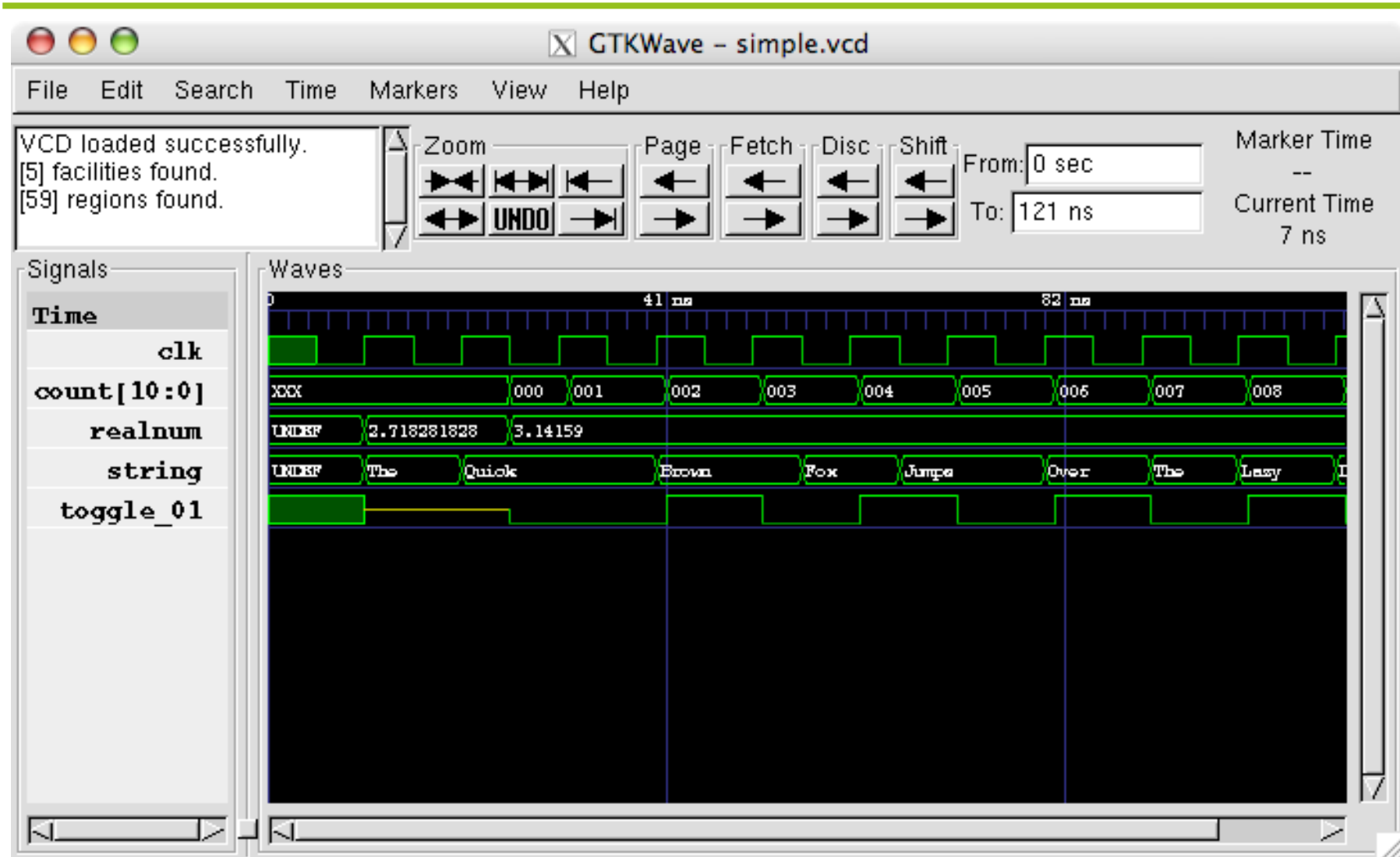
```
// Datei wave.h
```

```
SC_MODULE(wave) {  
    sc_signal<bool> brake;  
    sc_trace_file* tracefile  
    double temperature;  
};
```

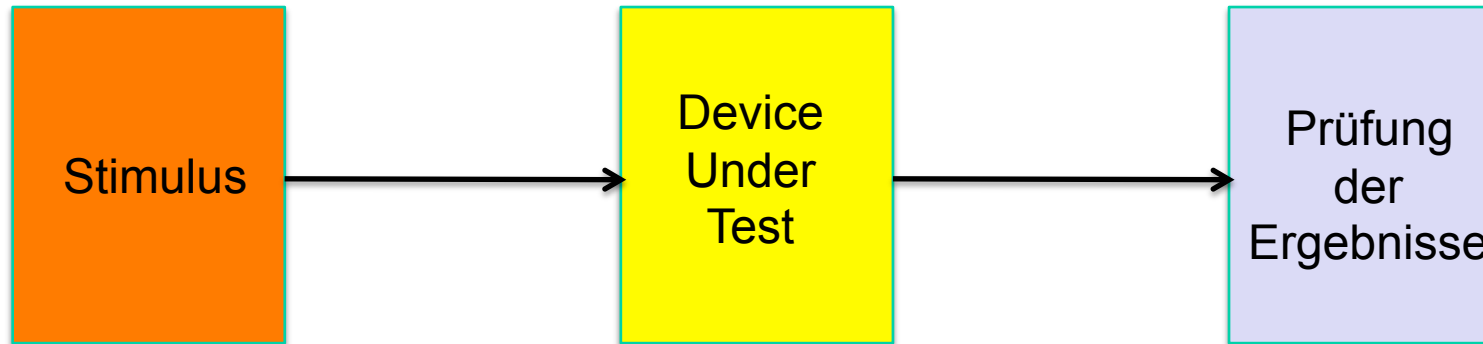
```
// Datei wave.cpp
```

```
wave::wave(sc_module_name nm); // Konstruktor  
:module(nm) { ...  
    tracefile = sc_create_vcd_file("wave");  
    sc_trace(tracefile, brake, "brake");  
    sc_trace(tracefile, temperature, "temperature");  
} // Ende Konstruktor  
wave::~~wave() {  
    sc_close_vcd_file(tracefile);  
    cout << "wave.vcd erzeugt" << endl;  
}
```

Beispielausgabe: GTKWave



Typische Testbench



- Das Stimulus-Modul erzeugt *Stimuli* für den zu testenden Systementwurf (*Design under Test, DUT*)
- Die Ergebnisprüfung kontrolliert die Ausgabe des Systems und überprüft, ob die Ausgabe korrekt ist

Testbenches

- Testbenches erzeugen Stimuli für den zu testenden Entwurf und prüfen die erzeugten Ausgaben
- Eine Testbench kann auf verschiedene Weisen implementiert werden
- **Der Stimulus kann...**
 - von einem Prozess erzeugt, Ergebnisse von einem anderen geprüft werden
 - im Hauptprogramm integriert werden und die Ergebnisse von anderem Prozess überprüft werden
- **Die Überprüfung kann auch...**
 - z.B. im Hauptprogramm integriert werden usw.
- Es gibt keine kanonische Methode zum Entwurf
 - Abhängig von der Anwendung

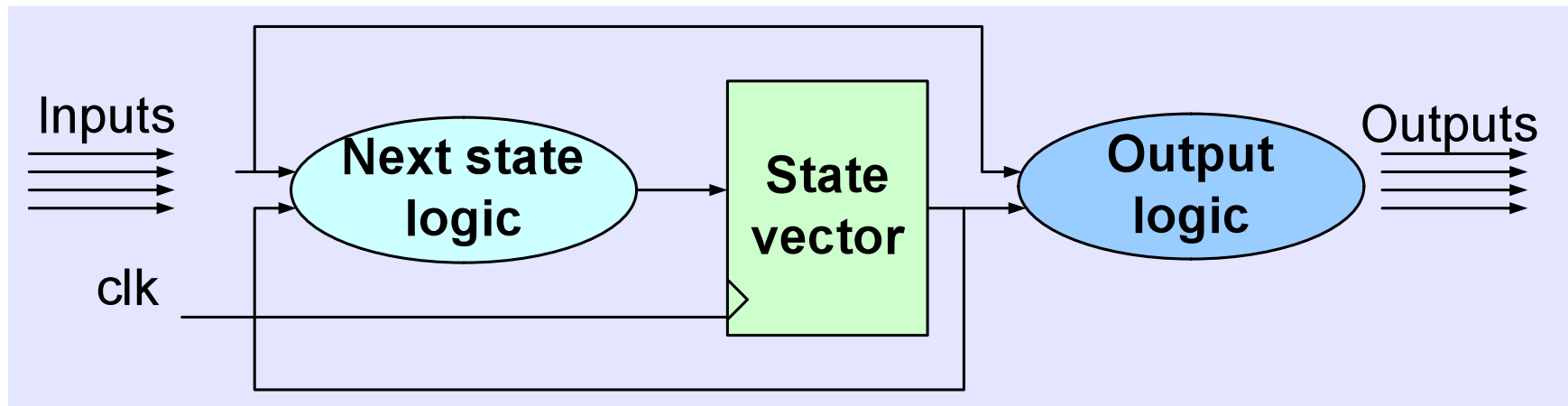
Typische Testbench

- Das Stimulus-Modul kann Stimuli aus einer Datei lesen oder sie als **SC_THREAD** Prozess realisieren
- Entsprechendes gilt für die Überprüfung der Ergebnisse
- Oft werden Stimuli-Erzeugung und Ergebnisüberprüfung in einem Modul vereint
- Ergebnisüberprüfung kann entfallen, wenn die Ausgabe manuell überprüft wird
 - VCD-Datei => GTKwave

Testbench-Beispiel: Endlicher Automat

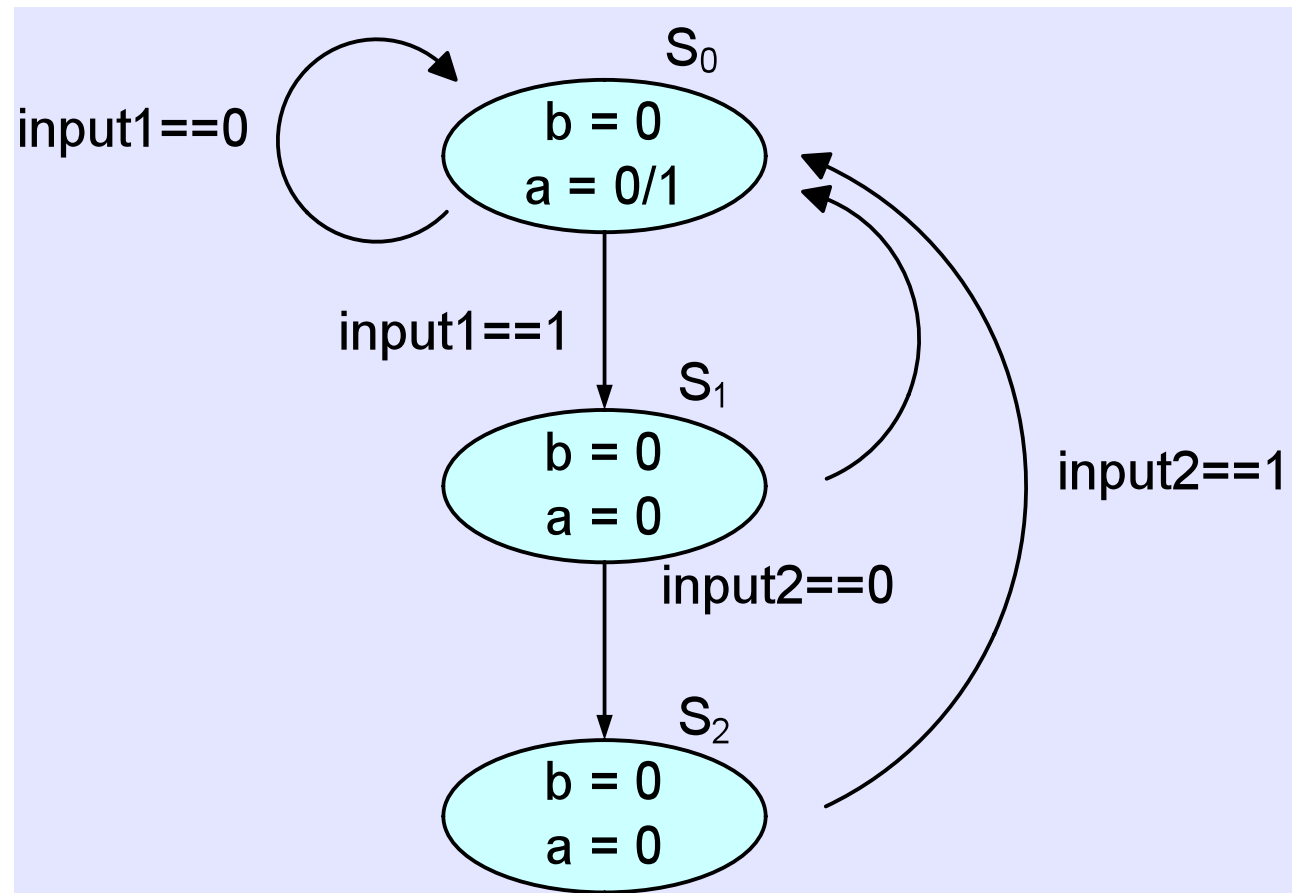
■ Mealy-Automat:

- Ein sequentielles Element: Statusvektor
- Zwei kombinatorische Elemente
 - Ausgabelogik
 - Logik für den Folgezustand



Testbench-Beispiel: Endlicher Automat

- Zustandsdiagramm für einen Automaten
- Ausgänge a und b



Testbench-Beispiel: Endlicher Automat



Testbench-Beispiel: Endlicher Automat

```
// main.cpp
```

```
#include <systemc.h>
#include "fsm.h"
#include "stimulus.h"
#include "display.h"
int sc_main (int argc , char *argv[])
{
    sc_clock clock;
    sc_signal<bool> rst;
    sc_signal<bool> in1, in2;
    sc_signal<bool> out_a, out_b;

    stimulus
stimulus1("stimulus_block");
    stimulus1.rst(rst);
    stimulus1.input1(in1);
    stimulus1.input2(in2);
    stimulus1.clk(clock.signal());
```

```
fsm fsm1("MyFSM");
    fsm1.rst(rst);
    fsm1.input1(in1);
    fsm1.input2(in2);
    fsm1.a(out_a);
    fsm1.b(out_b);
    fsm1.clk(clock);

    display display1 ("display");
    display1.a(out_a);
    display1.b(out_b);
    display1.clk(clock);

    sc_start(clock, -1); // Endlos

    return 0;
}
```

Testbench-Beispiel: Endlicher Automat

```
// fsm.h
```

```
enum state_t
{ // enumerate FSM states
  S0, S1, S2
};

SC_MODULE(fsm){
  sc_in_clk clk;
  sc_in<bool> rst, input1, input2;
  sc_out<bool> a, b;
  sc_signal<state_t> state,
  next_state;

  void ns_op_logic();
  void update_state();

  SC_CTOR(fsm){
    SC_METHOD(update_state);
    sensitive_pos << clk;
    SC_METHOD(ns_op_logic);
    sensitive << state << input1 <<
input2;
  }
};
```

Deprecated...

Testbench-Beispiel: Endlicher Automat

```
#include "systemc.h"
#include "fsm.h"

void fsm::update_state() {
    if (rst.read() == true)
        state = S0;
    else
        state = next_state;
}

void fsm::ns_op_logic() {
    switch(state) {
        case S0: b.write(0);
                if (input1.read() ||
input2.read())
                    a.write(1);
                else
                    a.write(0);
                if (input1.read() == 1)
                    next_state = S1;
                else
                    next_state = S0;
                break;
    }
```

```
// fsm.cpp
```

```
case S1: a.write(0);
        b.write(1);
        if (input2.read() == 1)
            next_state = S2;
        else
            next_state = S0;
        break;

case S2: a.write(0);
        b.write(0);
        next_state = S0;
        break;

default: a.write(0);
        b.write(0);
        next_state = S0;
        break;
    }
}
```

Testbench-Beispiel: Endlicher Automat

stimulus.h

```
SC_MODULE(stimulus) {  
  
    sc_out<bool> rst;  
    sc_out<bool> input1, input2;  
    sc_in<bool> clk;  
  
    unsigned cycle;  
  
    SC_CTOR(stimulus)  
    {  
        SC_METHOD(entry);  
        dont_initialize();  
        sensitive_pos(clk);  
        cycle = 0;  
    }  
    void entry();  
};
```

Deprecated!

Testbench-Beispiel: Endlicher Automat

```
#include <systemc.h>
#include "stimulus.h"
void stimulus::entry() {

    cycle++;

    if (cycle<4) {
        rst.write(true);
        input1.write(false);
        input2.write(false);
    } else {
        rst.write(false);
        if (cycle%4==0) {
            input1.write(true);
            input2.write(true);
```

stimulus.cpp

```
        } else if(cycle%4==1) {
            input1.write(true);
            input2.write(false);
        } else if(cycle%4==2) {
            input1.write(false);
            input2.write(true);
        } else if(cycle%4==3) {
            input1.write(false);
            input2.write(false);
        }
    }
}
```

Testbench-Beispiel: Endlicher Automat

display.h

```
SC_MODULE(display) {  
  
    sc_in<bool> a, b;  
    sc_in<bool> clk;  
  
    unsigned i;  
  
    SC_CTOR(display)  
    {  
        SC_METHOD(entry);  
        dont_initialize();  
        sensitive_pos(clk);  
        i = 0;  
    }  
  
    void entry();  
};
```

Deprecated!

Testbench-Beispiel: Endlicher Automat

display.cpp

```
#include <systemc.h>
#include "display.h"

void display::entry(){

    cout<< "Display : " << "a=" << (int)a.read()
        << " b=" << (int)b.read() << " at time "
        << sc_simulation_time() << endl;

    i++;
    if(i == 24) {
        cout << "Simulation of " << i << " clock period"
            << " at time " << sc_simulation_time() << endl;
        sc_stop();
    };
}
```

Zusammenfassung

- Exports
- Clocks
- Testbenches