

# Berechnungsmodelle

*models of computation (MOC)*

Jürgen Ruf

Jürgen Ruf

Systembeschreibungssprachen SS 2002

## Motivation

- Wir haben mehrere Systembeschreibungssprachen kennen gelernt
- ABER: wie modelliert man nun Systeme mit diesen Sprachen
- Es gibt unterschiedliche Berechnungsmodelle, die für unterschiedliche
  - Sprachen
  - Systeme
  - was man modellieren möchtemehr oder weniger gut geeignet sind

Jürgen Ruf

Systembeschreibungssprachen SS 2002

## Berechnungsmodelle

### Ein MOC ist definiert durch

- das Zeitmodell
- die Kommunikationsmechanismen
- die Regeln zur Prozessaktivierung

### einige MOCs:

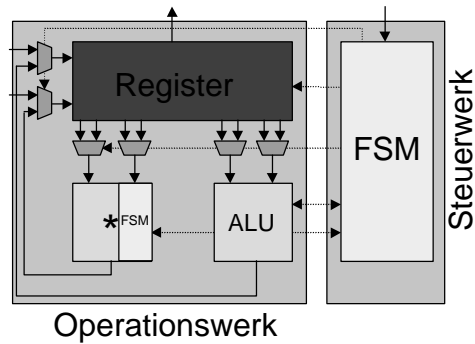
- discrete events
  - RTL/behavioral models
  - Transaction level models
- Kahn process networks
- Static dataflow, dynamic dataflow

## RTL

- Zur Beschreibung digitaler HW
- Synchronisation durch Takte (HW ist verzögerungsfrei)
- Kommunikation über Signale
- Prozesse/Threads repräsentieren
  - sequentielle Schaltungen, d.h. sensitiv auf die Taktleitung
  - kombinatorische Schaltungen, d.h. sensitiv auf alle Eingänge
- *pin-/zyklusakkurat*

## RTL-Architektur

- Aufteilung der HW in
  - Datenpfad (Operationswerk)
  - Kontrollogik (Steuerwerk)



Jürgen Ruf

Systembeschreibungssprachen SS 2002

## RTL Beispiel - robot controller

```
SC_MODULE(robot_controller) {  
  
    sc_in<bool>          CLOCK;  
    sc_in<bool>          RESET;  
    sc_in<sc_bv<8> >    uSEQ_BUS;  
    sc_in<bool>          CLRMRDY;  
    sc_in<bool>          uSW_ZERO;  
  
    sc_inout<bool>      MRDY;  
    sc_inout<bool>      REPOS;  
    sc_inout<bool>      MAGNET;  
    sc_out<bool>        XY;  
    sc_out<bool>        REVERSE;  
    sc_out<bool>        LDDIR;  
    sc_out<bool>        LSB_CNTR;  
  
    /* Internal variables and signals */  
    enum ctrl_state { IDLE, INST, DIST, RECAL, DIR, MOVE };  
    sc_signal<ctrl_state> curr_state, next_state;  
};
```

Jürgen Ruf

Systembeschreibungssprachen SS 2002

## RTL Beispiel - robot controller

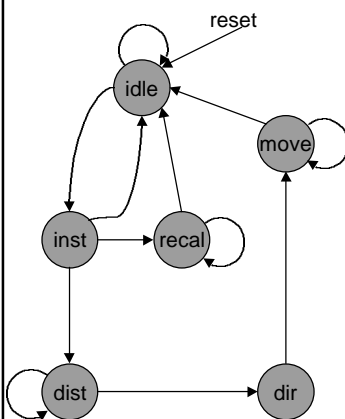
```
void ctrl_fsm_state();
void ctrl_fsm();

/* Constructor */
SC_CTOR(robot_controller) {
    SC_METHOD(counter_proc);    sensitive << CLOCK.pos();
    SC_METHOD(inst_reg_proc);   sensitive << CLOCK.pos();
    SC_METHOD(mrdy_proc);       sensitive << CLOCK.pos();
    SC_METHOD(ctrl_fsm_state);  sensitive << CLOCK.pos();
    SC_METHOD(ctrl_fsm);
        sensitive << RESET << REPOS << MAGNET
            << DONE << uSW_ZERO << MRDY << curr_state;
    }
};
```

Jürgen Ruf

Systembeschreibungssprachen SS 2002

## RTL Beispiel - robot controller



```
Void robot_controller::ctrl_fsm_state()
{
    curr_state.write(next_state.read());
}
```

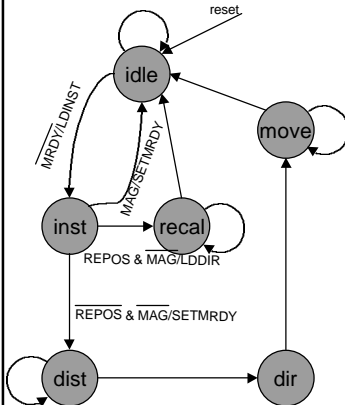
```
void robot_controller::ctrl_fsm()
{
    ctrl_state ns = curr_state;
```

```
    LDDIST.write(false);
    COUNT.write(false);
    LDINST.write(false);
    SETMRDY.write(false);
    ...
```

Jürgen Ruf

Systembeschreibungssprachen SS 2002

## RTL Beispiel - robot controller



```
if (RESET.read()) {
    ns = IDLE;
} else {
    switch (curr_state.read()) {
    case IDLE:
        if (!MRDY.read()) {
            LDINST.write(true);
            ns = INST;
        }
        break;
    case INST:
        if (MAG.read()) {
            SETMRDY.write(true);
            ns = IDLE;
        } else if (REPOS.read()) {
            LDDIR.write(true);
            ns = RECAL;
        } else {
            SETMRDY.write(true);
            ns = DIST;
        }
        break;
    case RECAL:
        .....
    }
```

Jürgen Ruf

Systembeschreibungssprachen SS 2002

## Verhaltensbeschreibung - Motivation

In RTL muß der Designer festlegen:

- die Zustände der FSMs
- die Zustandsübergänge
- welche Operation wann ausgeführt wird
- wieviele Ressourcen es gibt (z.B. 1 Mult, 2 ALUs)

→ Wenig Flexibilität, keine Alternativen möglich

→ Deshalb Beschreibung auf höherer Ebene:

Verhaltensbeschreibung, algorithmische Ebene

Jürgen Ruf

Systembeschreibungssprachen SS 2002

## Verhaltensbeschreibung

- Ein-/Ausgaben werden als geordnete Sequenz aufgefaßt
  - durch Takt synchronisiert
  - aber ein Takt auf algorithmischer Ebene kann mehreren auf RT-Ebene entsprechen
- Verhalten wird durch Programmfluß beschrieben (auf Modulebene)
- Zuordnung von Operation auf Takte und funktionale Blöcke übernimmt Synthesetool
- Modulegrenzen sind immer noch *pinakkurat*
  - Verbindung von Modulen durch Signale

Jürgen Ruf

Systembeschreibungssprachen SS 2002

## Verhaltensbeschreibung - Beispiel

```
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());
        watching(RESET.delayed() == true);
    }
};
```

Jürgen Ruf

Systembeschreibungssprachen SS 2002

## Verhaltensbeschreibung - Beispiel

```
void
euclid_gcd::compute()
{
    // reset section
    unsigned tmp_a = 0, tmp_b;

    while (true) { // main loop

        // Here is an I/O cycle
        C.write(tmp_a);
        READY.write(true);
        wait();

        // Another I/O cycle
        tmp_a = A.read();
        tmp_b = B.read();
        READY.write(false);
        wait();

        // No I/O takes place during
        // the computation of GCD.
        // Computation and
        // communication are separated
        while (tmp_b != 0) {
            unsigned r = tmp_a;
            tmp_a = tmp_b;
            while (r >= tmp_b) {
                r = r - tmp_b;
            }
            tmp_b = r;
        }
    }
}
```

Jürgen Ruf

Systembeschreibungssprachen SS 2002

## Funktionale Beschreibung auf Systemebene

### Bisher Beschreibung von Hardware

- Struktur, Interface, Timing und Funktion sind (teilweise) festgelegt

Aber auf höheren Ebenen (vor HW/SW-Partitionierung) will man oft nur die Funktionalität beschreiben und von diesen Details abstrahieren

- effizientere Modellierung
- schnellere Simulation

Jürgen Ruf

Systembeschreibungssprachen SS 2002

## Zeitfreie funktionale Modelle - Datenfluß

### Kahn Prozessnetzwerke

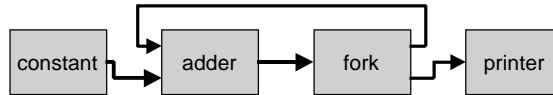
- Zur Beschreibung von Signalverarbeitungs Anwendungen (Multimedia/Telekommunikation)
- Module arbeiten parallel
- Kommunikation über unendliche FIFOs (blocking-read, nonblocking-write)
- Verarbeitung häufig zeitfrei modelliert

## Zeitfreie funktionale Modelle - Datenfluß

### Statische Datenfluß Netzwerke

- spezielle Kahn-Prozess-Netzwerke
- Jeder Prozeß ist aufgeteilt in drei Schritte
  - Eingabetokens lesen
  - Berechnung durchführen
  - Ausgabetokens schreiben
- Anzahl der Knoten die in jedem Schritt geschrieben und gelesen werden ist für jeden Prozeß fest und zur Compilezeit bekannt

## Datenfluß Beispiel



- „constant“ liefert einen konstanten Wert
- „adder“ nimmt seine Eingaben und addiert diese
- „fork“ nimmt seine Eingabe und gibt sie auf zwei Ports aus
- „printer“ gibt die Ergebnisse aus

Jürgen Ruf

Systembeschreibungssprachen SS 2002

## Datenfluß Beispiel Sourcen

```
template <class T>
  SC_MODULE(DF_Const){
  sc_fifo_out<T> output;
  void process() {
  while (1);
  output.write(constant_);
  }

  SC_HAS_PROCESS(DF_Const);
  DF_Const(sc_module_name NAME,
  const T& CONSTANT) :
  sc_module(NAME),
  constant_(CONSTANT) {
  SC_THREAD(process);
  }

  T constant_;
};

template <class T>
  SC_MODULE(DF_Adder) {
  sc_fifo_in<T> input1, input2;
  sc_fifo_out<T> output;
  void process() {
  while (1)
  output.write(input1.read()
  + input2.read());
  }

  SC_CTOR(DF_Adder) {
  SC_THREAD(process);
  }
};
```

Jürgen Ruf

Systembeschreibungssprachen SS 2002

## Datenfluß Beispiel Sourcen

```
template <class T> SC_MODULE(DF_Fork) {
    sc_fifo_in<T> input;
    sc_fifo_out<T> output1, output2;
    void process() {
        while(1) {
            T value = input.read();
            output1.write(value);
            output2.write(value);
        }
    }
    SC_CTOR(DF_Fork) { SC_THREAD(process); }
};
```

Jürgen Ruf

Systembeschreibungssprachen SS 2002

## Datenfluß Beispiel Sourcen

```
template <class T> SC_MODULE(DF_Printer) {
    sc_fifo_in<T> input;
    SC_HAS_PROCESS(DF_Printer);

    DF_Printer(sc_module_name NAME,
               unsigned N_ITERATIONS) :
        sc_module(NAME),
        n_iterations_(N_ITERATIONS),
        done_(false)
    { SC_THREAD(process); }

    void process() {
        for (unsigned i=0; i<n_iterations_; i++) {
            T value = input.read();
            cout << name() << " " << value << endl;
        }
        done_ = true;
        return;
    }

    ~DF_Printer() {
        if (!done_)
            cout << name()
                 << " not done yet" << endl;
    }

    unsigned n_iterations_;
    bool done_;
};
```

Jürgen Ruf

Systembeschreibungssprachen SS 2002

## Datenfluß Beispiel Sourcen

```
int sc_main(int, char**) {
    DF_Const<int> constant("constant", 1);
    DF_Adder<int> adder("adder");
    DF_Fork<int> fork("fork");
    DF_Printer<int> printer("printer", 10);

    sc_fifo<int> const_out("const_out", 5);
    sc_fifo<int> adder_out("adder_out", 1);
    sc_fifo<int> feedback("feedback", 1);
    sc_fifo<int> to_printer("to_printer", 1);

    feedback.write(42);

    // interconnect
    constant.output(const_out);
    adder.input1(const_out);
    adder.input2(feedback);

    adder.output(adder_out);
    fork.input(adder_out);
    fork.output1(feedback);
    fork.output2(to_printer);
    printer.input(to_printer);

    sc_start(-1);

    return 0;
}
```

Jürgen Ruf

Systembeschreibungssprachen SS 2002

## Hinzufügen von Zeitinformationen

```
void process() {
    while (1) {
        wait(200, SC_NS);
        output.write(constant_);
    }
}

void process() {
    while (1) {
        T data = input1.read() + input2.read();
        wait(200, SC_NS);
        output.write(data);
    }
}

...
```

Jürgen Ruf

Systembeschreibungssprachen SS 2002

## Transaktionsbasierte Modelle

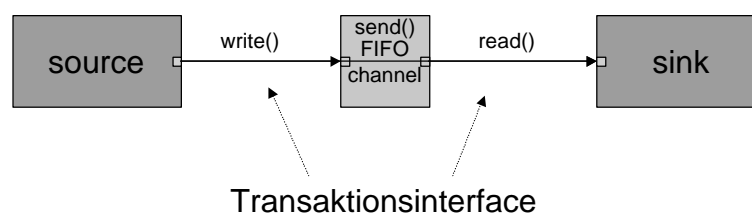
- Diskretes Ereignis-Modell
- Kommunikation via Funktionsaufrufen (Transaktionen) mit
  - Startzeit, Endzeit, Daten
- Modell betont die Funktionalität des Datentransfers: welche Daten, Quelle-Ziel,...
- Schnellere Simulation/Entwurfsraumexploration
- Abstraktion von Details wie Kommunikationsprotokoll, etc.

Jürgen Ruf

Systembeschreibungssprachen SS 2002

## Transaktionsbasierte Modelle

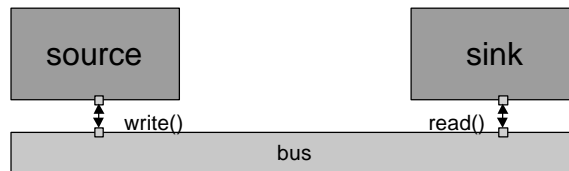
- Einsatz zur Modellierung von
  - zeitfreien funktionalen Modellen
  - zeitbehafteten funktionalen Modellen
  - Testbenches



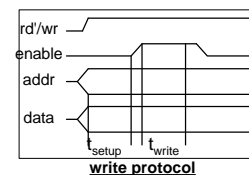
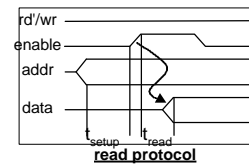
Jürgen Ruf

Systembeschreibungssprachen SS 2002

## Transaktionsbasierte Modelle



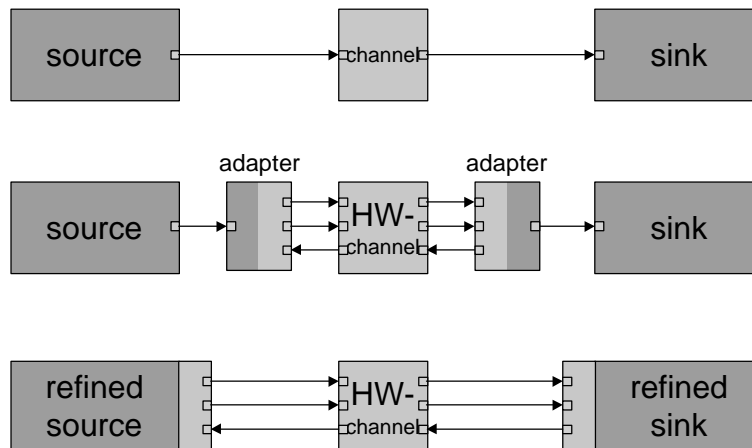
- Hinter den Transaktionsaufrufen können sich komplexe Protokolle verstecken, die aber nicht implementiert werden, aber Einfügen von Verzögerungszeiten möglich



## Kommunikationsverfeinerung

- Eine manuelle Technik um Kommunikation auf hoher Abstraktionsebene (transaktionsbasiert) auf Interfaces auf niedrigen Abstraktionsebenen Abzubilden

## Kommunikationsverfeinerung



Jürgen Ruf

Systembeschreibungssprachen SS 2002

## Kommunikationsverfeinerung

- **Kanalverfeinerung**
  - Ersetzen des abstrakten Kanals durch einen HW-nahen Kanal
- **Adapter erzeugen und einfügen**
  - Adapter bildet Kommunikation auf HW-nahe Interfaces ab
- **Validation**
  - Verifizieren, ob die Kommunikation wie gewünscht funktioniert
- **Interfaceverfeinerung**
  - inlinen der Adapterfunktion in den Sender/Empfänger (protocol inlining)
- **Wiederholte Validation**

Jürgen Ruf

Systembeschreibungssprachen SS 2002

## Beispiel



```
SC_MODULE(source) {
  sc_port<sc_fifo_write_if<int> > out;

  void main_action() {
    while (true) {
      ...
      out->write( data );
      ...
    }
    ...
  }
}
```

```
SC_MODULE(sink) {
  sc_port<sc_fifo_read_if<int> > in;

  void main_action() {
    while (true) {
      ...
      data = in->read();
      ...
    }
    ...
  }
}
```

Jürgen Ruf

Systembeschreibungssprachen SS 2002

## HW FIFO

- Die Kommunikation soll in HW realisiert werden  
→ Kopplung der abstrakten Kommunikation und der HW-Realisierung (im Beispiel Handshaking)

```
SC_MODULE(hw_fifo) {
  sc_in<int> data_in;
  sc_in<bool> data_in_valid;
  sc_out<bool> data_in_ready;

  sc_out<int> data_out;
  sc_out<bool> data_out_valid;
  sc_in<bool> data_out_was_read;

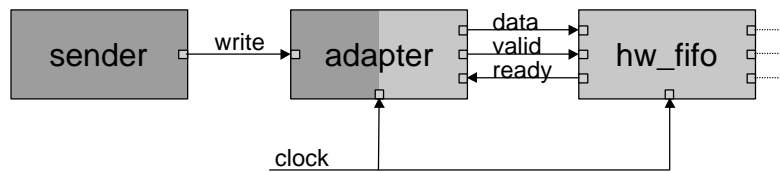
  sc_in_clk clk;
  ...
}
```

Jürgen Ruf

Systembeschreibungssprachen SS 2002

## Refinement durch einführen von Adaptern

- Adapter bildet die high-level Interfacebefehle auf low-level HW-Signale ab



Jürgen Ruf

Systembeschreibungssprachen SS 2002

## Der Adapter

```
class w_adapter : public sc_module,
                 public sc_fifo_write_if<int> {

    sc_out<int> data_port;
    sc_out<bool> valid_port;
    sc_in<bool> ready_port;
    sc_in_clk clk;

    virtual void write( const int& data ) {
        while (ready_port == false ) wait( clk.pos_edge());
        data_port = data;
        valid_port = true;
        wait( clk.pos_edge() );
        valid_port = false;
    }
}
```

Systembeschreibung  
nun mit Adaptern  
verifizieren

Jürgen Ruf

Systembeschreibungssprachen SS 2002

## Protocol inlining

```
SC_MODULE(refined_source) {  
  // sc_port<sc_fifo_write_if<int> > out;  
  sc_out<int> data_port;  
  sc_out<bool> valid_port;  
  sc_in<bool> ready_port;  
  sc_in_clk clk;  
  
  void main_action() {  
    while (true) {  
      ...  
      //out->write( data );  
      write( data );  
      ...  
    }  
    ...  
  }  
}
```

```
virtual void write( const int& data ) {  
  while (ready_port == false ) wait (clk.pos_edge()  
  data_port = data;  
  valid_port = true;  
  wait( clk.pos_edge() );  
  valid_port = false;  
}
```