technische universität
dortmund

fakultät für informatik
informatik 12

Synthese Eingebetteter Systeme

Sommersemester 2011

Übung 2

Michael Engel
Informatik 12
TU Dortmund

2011/05/06

Übung 2

- SystemC-Simulation
- Flip-Flop
- Cabrioverdeck

1. SystemC-Simulation

- SystemC: Simulationskern
  - Stellt sicher, dass parallele Abläufe (Nebenläufigkeit) korrekt modelliert werden
  - Verhalten der Simulation soll nicht von Aufrufreihenfolge von Prozessen abhängen
- Beispiel: 2 SC_THREADs:

```
SC_THREAD(proc_1);
    sensitive << Trig.pos();
SC_THREAD(proc_2);
    sensitive << Trig.pos();
```

- Welcher Thread startet beim Übergang von Trig von low nach high zuerst?
  - Darf das eine Rolle spielen?

1. SystemC-Simulation

- Kommunikation zwischen Prozessen via Signalen
  - Prozessausführung und Signalaktualisierung sind zwei separate Phasen
- Getätigte Zuweisungen an sc_signals finden *in der Zukunft* statt
  - Sobald alle gerade aktiven Prozesse evaluiert wurden und an einem Punkt sind, an dem sie suspendiert werden müssen oder auf ein Ereignis warten
- Es kann also vorkommen, dass keine Simulationszeit vergeht
  - Wenn noch Signalaktualisierungen ausstehen, laufen zugehörige Prozesse erneut, ohne dass die Zeit fortschreitet
  - Dies ist der ∂-Zyklus!

1. SystemC: ∂-Zyklen

- ∂-Zyklen bestehen also aus zwei Phasen:
  - Evaluationsphase
  - Aktualisierungsphase (*update phase*)
  - Deterministisches Verhalten für sc_signals
- Software-Modellierung ohne ∂-Zyklen sinnvoll
  - Keine Ausführung der Aktualisierungsphase
  - Erfordert *sofortige* Benachrichtigung über Ereignisse
  - Kann *nichtdeterministisches Verhalten* hervorrufen

1. SystemC: Events

Event-Benachrichtigung über die notify()-Methode der sc_event-Klasse

- notify() ohne Argumente: Sofortige Benachrichtigung
  - Prozesse, die sensitiv auf das Event sind, laufen noch in der aktuellen Evaluationsphase
- notify() mit Zeit=0: ∂-Benachrichtigung.
  - Prozesse, die sensitiv auf das Event sind, laufen während der Evaluationsphase des folgenden ∂-Zyklus
- notify() mit Zeit != 0: zeitgesteuerte Benachrichtigung
  - Prozesse, die sensitiv auf das Event sind, laufen während der Evaluationsphase eines zukünftigen Simulationszeitpunktes

1. SystemC: Ablauf

- Verhalten des Simulationskerns

1. **Initialisierung:** führe alle Prozesse (ausser SC_THREADS) in unbestimmter Reihenfolge aus
2. **Evaluierung:** wähle einen ablaufbereiten (“ready”) Prozess aus und führe seine Ausführung fort

   -> führt evtl. zu sofortigen Event-Benachrichtigungen und damit zur Ausführung weiterer Prozesse in der selben Phase
3. Wiederhole Schritt 2, bis keine Prozesse mehr ablaufbereit sind
4. **Aktualisierung:** führe alle in vorigen Schritten 2 angeforderten Aktualisierungen (Aufrufe von update() durch request_update()) aus
5. Wenn es ∂-Event-Benachrichtigungen in Schritt 2 oder 4 gab, bestimme die in Folge “ready” werdenden Prozesse und springe zu Schritt 2

1. SystemC: Ablauf

6. Wenn keine zeitanhängigen Events anstehen, ist der Simulationszyklus beendet
7. Setze Simulationzeit auf den Zeitpunkt der frühesten ausstehenden zeitgesteuerten Event-Benachrichtigung
8. Bestimme, welche Prozesse aufgrund der aktuellen zeitgesteuerten Events zum neuen Zeitpunkt ablaufbereit sind und fahre mit Punkt 2 fort

1. SystemC-Nichtdeterminismus

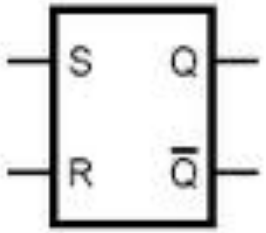
- SystemC-Simulation ist deterministisch für sc_signals
- Sprache erlaubt *Nichtdeterminismus* über Variablen
  - Für Hardwaremodellierung unbrauchbar…
  - aber evtl. für Softwaremodellierung nützlich

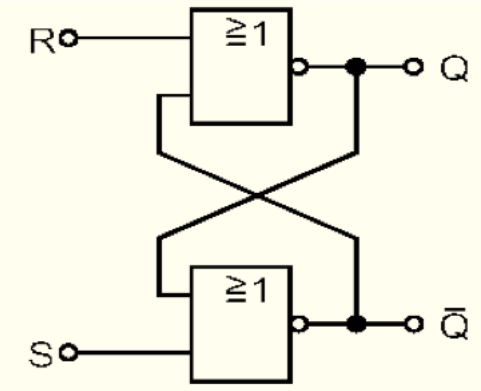
```
SC_MODULE(nondet)
{
  sc_in Trig;

  int SharedVariable;
  void proc_1()
  {
    SharedVariable = 1;
    cout << SharedVariable << endl;
  }
```

```
  void proc_2()
  {
    SharedVariable = 2;
    cout << SharedVariable << endl;
  }

  SC_CTOR(nondet)
  {
    SC_THREAD(proc_1);
    sensitive << Trig.pos();
    SC_THREAD(proc_2);
    sensitive << Trig.pos();
  }
};
```

2. Flipflop: RS

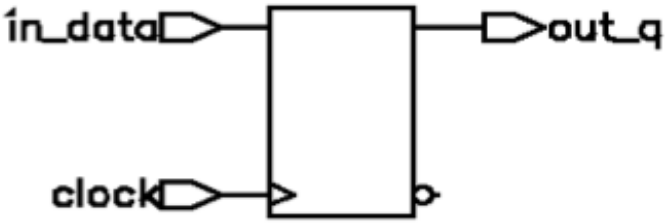
```
/* Einfaches RS-FF */

#include "systemc.h"
SC_MODULE (rsff1) {
  sc_in<bool> r, s;
  sc_out<bool> out_q, out_notq;
  void do_rsff();                  // Methode für RS-Flipflop
  SC_CTOR (rsff1) {                // Konstruktor
      SC_METHOD (do_rsff);
      sensitive << r << s;
   }
};
void rsff1::do_rsff() {            // Verhalten!
      if (s.read() == 1)
            { out_q.write(1); out_notq.write(0); }
      else if (r.read() == 1)
            { out_q.write(0); out_notq.write(1); }
}
```





2. D-Flipflop

```
/* Positiv flankengetriggertes DFF */

#include "systemc.h"
SC_MODULE (dff1) {
  sc_in<bool> in_data;
  sc_out<bool> out_q;
  sc_in<bool> clock;          // Clock port
  void do_dff_pos ();         // Methode für D-Flipflop
  SC_CTOR (dff1) {            // Konstruktor
      SC_METHOD (do_dff_pos);
      sensitive_pos << clock;
   }
};
void dff1::do_dff_pos() {
      out_q.write(in_data.read());        // Verhalten!
}
```

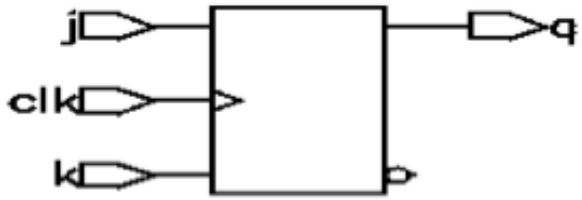


2. JK-Flipflop

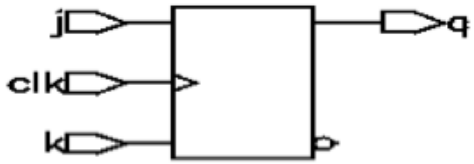
- Funktion

J	K	CLK	Qn+1
0	0	Rising	Qn
0	1	Rising	0
1	0	Rising	1
1	1	Rising	$\overline{Qn}$
X	X	Falling	Qn

- Schaltsymbol



2. JK-Flipflop



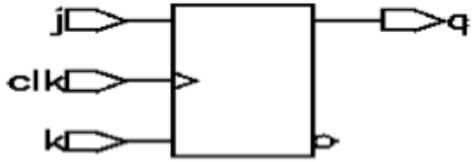
Verhaltensbeschreibung
– wird zu JK-FF synthetisiert!

```
void jkff1::jk_flop() {
    sc_uint<2> temp;
    // temp erzeugt Vektor
    temp[1] = j.read();
    temp[0] = k.read();
    switch(temp) {
        case 0x1: q.write(0);
            break;
        case 0x2: q.write(1);
            break;
        case 0x3: q.write(!q.read());
            break;
        default: break;
    }
}
```

```
/* Positiv flankengetriggertes JK-FF */
#include "systemc.h"
SC_MODULE (jkff1) {
    sc_in<bool> j, k;
    sc_inout<bool> q;   // inout: Q wieder lesen
    sc_in<bool> clk;    // Clock Port
    void jk_flop ();    // Methode
    SC_CTOR (jkff1) {   // Konstruktor
        SC_METHOD (jk_flop);
            sensitive << clk.pos();
    }
};
```

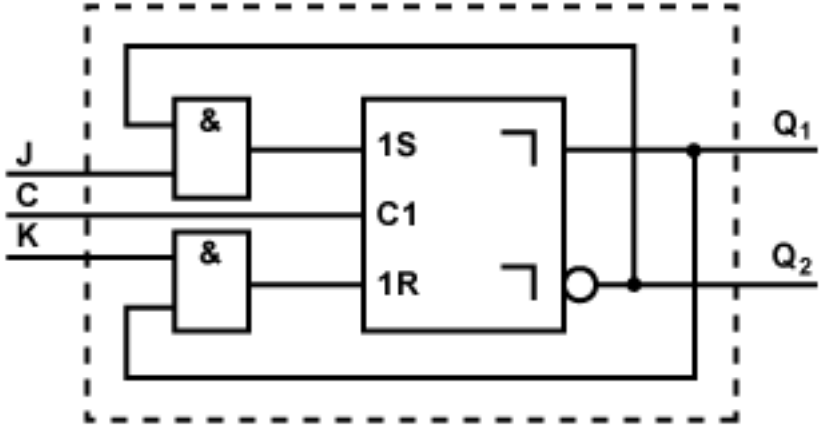
J	K	CLK	Qn+1
0	0	Rising	Qn
0	1	Rising	0
1	0	Rising	1
1	1	Rising	$\overline{Qn}$
X	X	Falling	Qn

2. JK-Flipflop



Alternativer Ansatz:

- Hierarchische Beschreibung
  - Getaktetes RS-FF
  - Plus zwei UND-Gatter
- Baut auf (taktgesteuertem) RS-FF auf



3. Cabrioverdeck

- *YOU!* ☺

Fragen?

- *Fragen!*