technische universität
dortmund

fakultät für informatik
informatik 12

Synthese Eingebetteter Systeme

Sommersemester 2011

Übung 3

Michael Engel
Informatik 12
TU Dortmund

2011/05/25

Übung 3

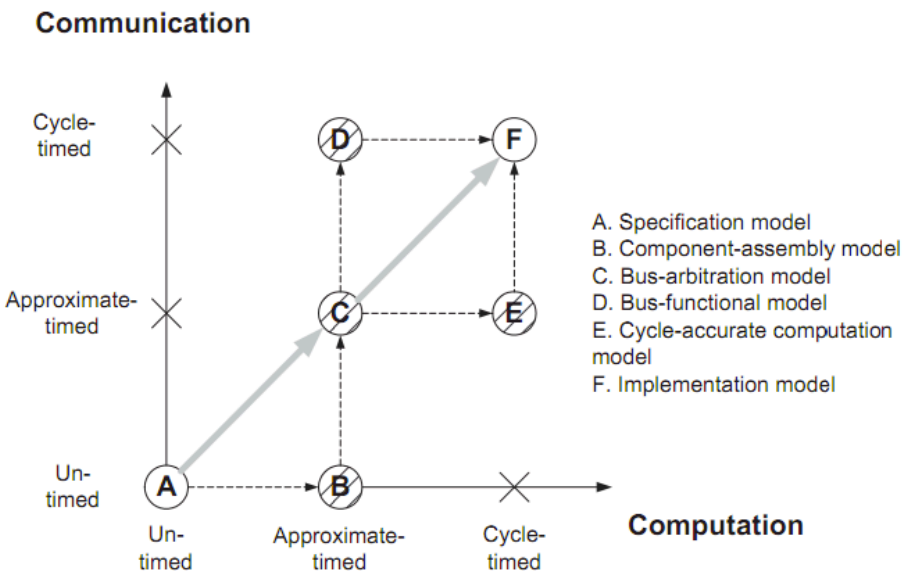
- SystemC-TLM
- TLM-Modellierung: TokenRing-Netzwerk
- Testbenches

SystemC-TLM

- Transaction Level Modeling erweitert SystemC um Möglichkeiten, Kommunikation abstrakt zu beschreiben. Erklären Sie mit eigenen Worten,
  - 1. warum Spezifikationsmodelle und Implementationsmodelle keine TLM-Modelle darstellen

SystemC-TLM

- Transaction Level Modeling erweitert SystemC um Möglichkeiten, Kommunikation abstrakt zu beschreiben. Erklären Sie mit eigenen Worten,
  - 1. warum Spezifikationsmodelle und Implementationsmodelle keine TLM-Modelle darstellen
- Spezifikationsmodelle: Keine Zeitangaben für
  - Berechnung **und**
  - Kommunikation
  - => Zu ungenau!
- Implementationsmodelle: Feste Zeitangaben für
  - Berechnung **und**
  - Kommunikation
  - => Zu präzise!



SystemC-TLM

- Transaction Level Modeling erweitert SystemC um Möglichkeiten, Kommunikation abstrakt zu beschreiben. Erklären Sie mit eigenen Worten,
  - 2. warum zwischen Berechnungs- und Kommunikationsmodellen unterschieden wird

SystemC-TLM

- Transaction Level Modeling erweitert SystemC um Möglichkeiten, Kommunikation abstrakt zu beschreiben. Erklären Sie mit eigenen Worten,
  - 2. warum zwischen Berechnungs- und Kommunikationsmodellen unterschieden wird
- Berechnung und Kommunikation sind verzahnt
  - Gemeinsame Modellierung würde Möglichkeiten beim Systementwurf einschränken
- Verschiedene Abstraktion der Modellierung gewünscht

SystemC-TLM

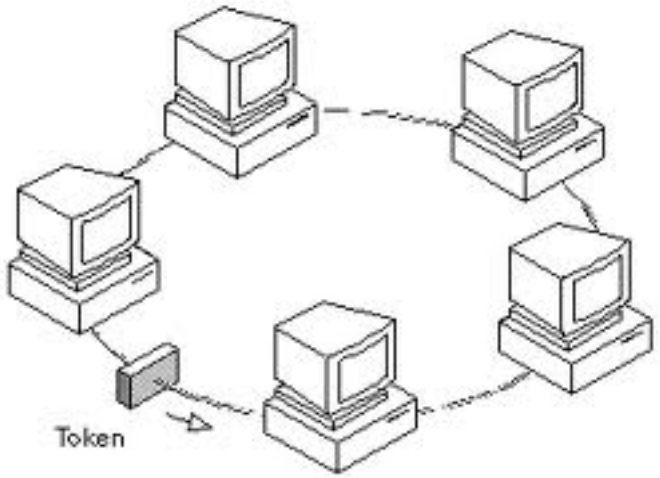
- Transaction Level Modeling erweitert SystemC um Möglichkeiten, Kommunikation abstrakt zu beschreiben. Erklären Sie mit eigenen Worten,
  - 3. warum es sinnvoll ist, die Zeitgranularität (approximate oder cycle timed) für Berechnungs- und Kommunikationsmodelle separat bestimmen zu können

SystemC-TLM

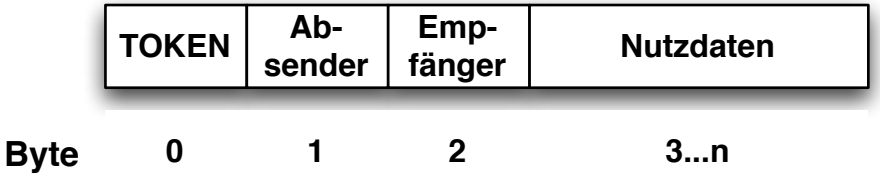
- Transaction Level Modeling erweitert SystemC um Möglichkeiten, Kommunikation abstrakt zu beschreiben. Erklären Sie mit eigenen Worten,
  - 3. warum es sinnvoll ist, die Zeitgranularität (approximate oder cycle timed) für Berechnungs- und Kommunikationsmodelle separat bestimmen zu können
- Sinnvoll z.B. für Design Space Exploration
  - Kommunikation (z.B. Bustyp als TDMA) oder aber Prozessormodell (z.B. festgelegter IP-Core) ist fest vorgegeben
  - Die jeweils andere Dimension lässt Freiraum für Optimierungen

TLM-Modellierung

- Modellierung eines Tokenring-Netzwerks
  - Kommunikation nur in eine Richtung
  - Knoten empfängt Paket
    - Weiterleitung, falls Paket nicht an Knoten adressiert
    - “Einbehaltung”, falls Paket an Knoten adressiert
    - “Einbehaltung” im Fehlerfall: empfangender Knoten = Absenderknoten
  - Token impliziert Sendeerlaubnis
    - Immer nur ein Token im System
    - Folge: <= 1 Paket gleichzeitig im Netz
    - …also auch pro Knoten nur für ein Paket Speicherplatz erforderlich



TLM-Modellierung



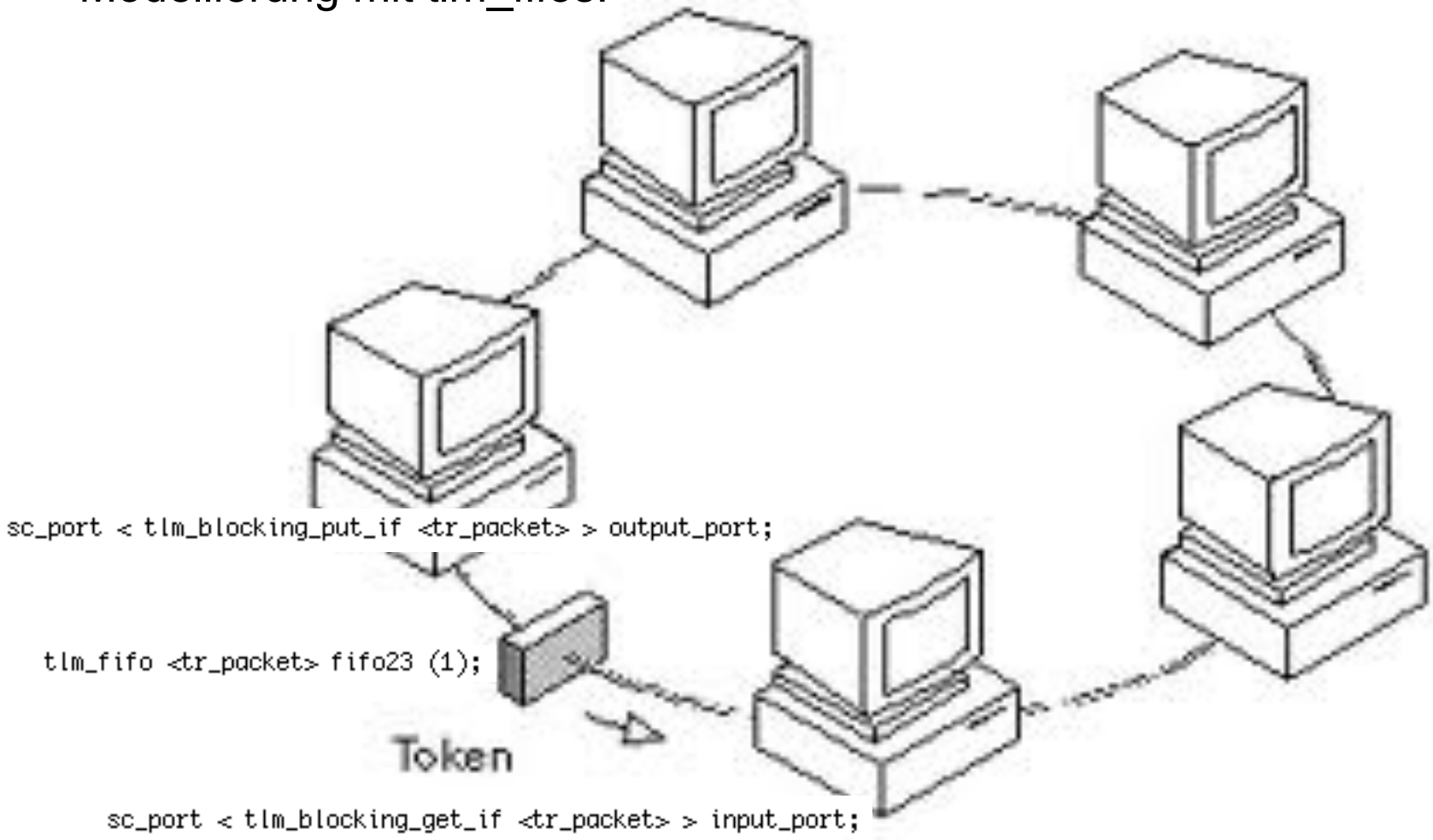
- Paketformat als struct (“n” muss nicht variabel sein, feste maximale Paketlänge ist realistisch):

```
struct tr_packet {
    sc_uint<8> token;    // 0=FREI oder 1=BELEGT
    sc_uint<8> sender;
    sc_uint<8> receiver;
    sc_uint<8> data[8];
}
```

- Templates ermöglichen die Übertragung von structs via tlm_fifos!

TLM-Modellierung

- Modellierung mit tlm_fifos:



Tokenring: Stationen Headerfile

- SystemC-Simulation für Token Ring-Netzwerk mit vier Stationen S0–S3 basierend auf ~~SystemC TLM tlm_uni_channels~~ unidirektionalen TLM-Kanälen wie tlm_fifos.
- Alle vier tr_stations sind identisch:

```
#include <systemc.h>

SC_MODULE(tr_station)
{
  sc_port < tlm_blocking_get_if <tr_packet> > input_port;
  sc_port < tlm_blocking_put_if <tr_packet> > output_port;

  int id; // meine netzwerkadresse
  tr_packet p; // ankommendes paket
  tr_packet neu; // zu sendendes paket
  bool sent; // ist zu sendendes paket rausgegangen?

  bool packet_to_send(void);
  void tr_station_thread(void);
```

Kein SC_CTOR-Makro!
So ist die Übergabe von Parametern an SC_MODULES möglich

```
  SC_HAS_PROCESS(tr_station);
  tr_station(sc_module_name nm, int configured_id) : sc_module(nm) {
      cout << "Station " << configured_id << " laeuft!" << endl;
      id = configured_id;
      sent = true;
      SC_THREAD(tr_station_thread);
  }
};
```

Tokenring: Stationen Implementierung

- …***fast*** identisch:

```
void tr_station::tr_station_thread() {
        // eine Station muss erstmal was schicken, sonst geht nichts!
        if (id == 1) {
                neu.receiver = 3;
                neu.sender = id;
                neu.token = true;
                cout << "Station " << id << " sendet initiales Paket" << endl;
                output_port->put(neu);
        }
```

Eine Station muss initiales Paket senden.
Wenn nicht, endet Simulation nach Initialisierung!

Tokenring: Stationen Implementierung

- Warten auf Paketempfang und prüfen des Inhalts:

```
while (1) {
        // cout << "Station " << id << " wartet auf Paket" << endl;
        input_port->get(p);     // blockierend

        // eingehendes paket anschauen...
        // bin ich empfaenger oder sender des pakets?
        if (p.receiver == id || p.sender == id) {
                // paket runternehmen
                // ...und neues, leeres verwenden
                if (p.sender == id) {
                        cout << "FEHLER: Station " << id
                             << " hat eigenes Paket empfangen!" << endl;
                } else {
                        cout << "Station " << id << " hat Paket von "
                             << "Station " << p.sender << " erhalten." << endl;
                }
                // leeres Paket auf die Reise schicken
                cout << "Station " << id << " sendet freies Token" << endl;
                neu.receiver = 0;
                neu.sender = 0;
                neu.token = false;
                output_port->put(neu);
        } else {
```

Tokenring: Stationen Implementierung

- Wenn freies Token ankommt, dieses weiterleiten…
  - ausser wir haben ein eigenes Paket zu versenden…

```
        } else {
                if (packet_to_send() && p.token == false) { // freies token angekommen und paket bereit
                        neu.receiver = (rand() % 4) + 1;
                        neu.sender = id;
                        neu.token = true;
                        cout << "Station " << id << " sendet Paket an " << neu.receiver << endl;
                        output_port->put(neu);
                        sent = true;
                } else { // angekommenes paket weitersenden
                        cout << "Station " << id << " leitet Paket weiter an " << p.receiver << endl;
                        output_port->put(p);
                }
        }
    }
}
```

- Die hier vorgestellte Lösung ist suboptimal – warum?

Tokenring: Stationen Implementierung

- Die hier vorgestellte Lösung ist suboptimal – warum?

Tokenring: Stationen Implementierung

- Die hier vorgestellte Lösung ist suboptimal – *deshalb*:

```
while (1) {
        // cout << "Station " << id << " wartet auf Paket" << endl;
        input_port->get(p);     // blockierend

        // eingehendes paket anschauen...
        // bin ich empfaenger oder sender des pakets?
        if (p.receiver == id || p.sender == id) {
                // paket runternehmen
                // ...und neues, leeres verwenden
                if (p.sender == id) {
                        cout << "FEHLER: Station " << id
                             << " hat eigenes Paket empfangen!" << endl;
                } else {
                        cout << "Station " << id << " hat Paket von "
                             << "Station " << p.sender << " erhalten." << endl;
                }
                // leeres Paket auf die Reise schicken
                cout << "Station " << id << " sendet freies Token" << endl;
                neu.receiver = 0;
                neu.sender = 0;
                neu.token = false;
                output_port->put(neu);
        } else {
```

Wenn Empfänger eines Pakets neues Paket zu versenden hat, wartet er, bis ein freies Token ankommt…

Tokenring: Top Level

- Fitting it all together:

```
#include <tr.h>
#include <tr_station.h>

int sc_main(int argc, char **argv)
{
  tlm_fifo <tr_packet> fifo12 (1);
  tlm_fifo <tr_packet> fifo23 (1);
  tlm_fifo <tr_packet> fifo34 (1);
  tlm_fifo <tr_packet> fifo41 (1);

  tr_station t1("t1", 1);
  tr_station t2("t2", 2);
  tr_station t3("t3", 3);
  tr_station t4("t4", 4);

  // t1 -> t2
  t1.output_port(fifo12);
  t2.input_port(fifo12);

  // t2 -> t3
  t2.output_port(fifo23);
  t3.input_port(fifo23);

  // t3 -> t4
  t3.output_port(fifo34);
  t4.input_port(fifo34);

  // t4 -> t1
  t4.output_port(fifo41);
  t1.input_port(fifo41);

  sc_start();
}
```

Tokenring: Beliebig große Netze

- Erweitern Sie die Simulation auf ein Token Ring-Netzwerk mit einer beliebig großen Anzahl an Stationen
- Lösung: Parameterübergabe über argv…

```
int sc_main(int argc, char **argv)
{
  int anzahl;
  int i;
  char name[7] = "tx..";

  if (argc < 2) {
        cout << "Aufruf: " << argv[0] << " <Anzahl>" << endl;
        exit(1);
  }

  anzahl = atoi(argv[1]);

  if (anzahl > 9999) {
        cout << "Max. 9999 Stationen!" << endl;
        exit(1);
  }

  cout << "Erzeuge TokenRing mit " << anzahl << " Stationen" << endl;
```

Tokenring: Beliebig grosse Netze

- Lösung: …Verwendung von Arrays und Schleifen!

Variabel dimensionierte Arrays…

```
tr_station* stationen[anzahl];
tlm_fifo<tr_packet>* fifos[anzahl];

for (i=1; i<=anzahl; i++) {
  snprintf(name, 7, "tx%04d", i);
  stationen[i-1] = new tr_station(name, i, anzahl);
  fifos[i-1] = new tlm_fifo<tr_packet>(1);
}

for (i=1; i<=anzahl; i++) {
  cout << "Verbinde Station " << i-1 << " mit Station " << i%anzahl << " ueber fifo " << i-1 << endl;
  cout << "output" << endl;
  stationen[i-1]->output_port(*(fifos[i-1]));
  cout << "input" << endl;
  stationen[i%anzahl  ]->input_port (*(fifos[i-1]));
}

sc_start();
}
```

Weiterer Parameter für Konstruktor

- Außerdem: Ersetzen der Konstante “4” in tr_station durch anzahl

Testbenches

- Welche Fälle sollten überprüft werden?

Testbenches

- Welche Fälle sollten überprüft werden?
  - Anzahl der Tokens/Pakete im Netz
    - Nur eine der tlm_fifos darf ein Paket enthalten!
  - Reihenfolge der Pakete
    - Richtung…
  - Holt Empfänger sein Paket ab?
  - Wird “Rundreise” von Paket nach einem Durchlauf beendet?
  - …mehr?

Testbenches

- Anzahl der Tokens/Pakete im Netz
  - Nur eine der tlm_fifos darf ein Paket enthalten!

```
tr_packet p;

while (1) {
  int count;

  // run for 1 delta cycle in each iteration
  sc_start(0);

  count=0;
  for (int i=0; i<anzahl; i++) {
      if(fifos[i]->nb_peek(p)) {
        cout << "Token: " << p.token << endl;
        cout << "Absender: " << p.sender << endl;
        cout << "Empfaenger: " << p.receiver << endl;
        count++;
      }
  }
  if (count != 1)
      cout << "Mehr als ein Paket im Netz!" << endl;
  else
      cout << "Exakt ein Paket im Netz - OK!" << endl;
}
```

Testbenches

- Reihenfolge der Pakete
- Überprüfen, ob Pakete von n -> n+1 (mod #) verschickt werden!

```
tr_packet p;

// run for 1 delta cycle in each iteration
sc_start(0);

// Erstes Paket finden
int paket_bei=0;
int paket_bei_neu=0;
for (int i=0; i<anzahl; i++) {
      if(fifos[i]->nb_peek(p)) {
        cout << "Paket bei " << i << endl;
        paket_bei = i;
    }
}

// Reihenfolge ueberpruefen
while (1) {
  int count;

  // run for 1 delta cycle in each iteration
  sc_start(0);

  for (int i=0; i<anzahl; i++) {
     if(fifos[i]->nb_peek(p)) {
        cout << "Paket bei " << i << endl;
        paket_bei_neu = i;
        if (paket_bei_neu != ((paket_bei+1) % anzahl))
           cout << "Reihenfolge falsch!" << endl;
        paket_bei = paket_bei_neu;
     }
  }
}
```

Fragen?

- *Fragen!*