technische universität
dortmund

fakultät für informatik
informatik 12

Synthese Eingebetteter Systeme

Sommersemester 2011

Übung 1

Michael Engel
Informatik 12
TU Dortmund

2011/04/29

Übung 1

- SystemC-Datentypen
- Simulationsperformance
- SystemC-Makros
- Abbildung auf RTL-Ebene

1. SystemC-Datentypen

- Installieren Sie die aktuelle SystemC-Version 2.2 (von http://www.systemc.org) und übersetzen Sie das “Hello, World”-Beispiel aus der Vorlesung
  - Probleme?
- Modifizieren Sie das Beispiel so, dass Sie das Verhalten von sc_logic-Variablen untersuchen können

Hello_SC.cpp

```
void Hello_SC::main_method(void)
{
  std::cout << "0 AND Z = " << ((sc_logic)'0' & (sc_logic)'Z') << std::endl;
  std::cout << "1 OR X = " << ((sc_logic)'1' | (sc_logic)'X') << std::endl;
  std::cout << "X XOR Z = " << ((sc_logic)'X' ^ (sc_logic)'Z') << std::endl;
}
```

```
0 AND Z = 0
1 OR X = 1
X XOR Z = X
```

2. Simulationsperformance

- Finden Sie mit einem geeigneten SystemC-Modell den Unterschied in der Simulationsgeschwindigkeit für
  - **bool**-Variablen vs. **sc_bit** und **sc_logic**

Hello_SC.cpp

```
#include <time.h>

void Hello_SC::main_method(void)
{
  clock_t start, end;
  bool x = 0;

  start = clock();
  // 100.000 Durchläufe
  for(int i = 0; i < 100000; i++){
    x != x;
  }
  end = clock();
  std::cout << "100000x Bool: " << end - start << " Clocks" << endl;

  // entsprechend für sc_bit, sc_logic
```

2. Simulationsperformance

- Finden Sie mit einem geeigneten SystemC-Modell den Unterschied in der Simulationsgeschwindigkeit für
  - **sc_int**- vs. **sc_bigint**-Variablen mit 64 Bit Breite

Hello_SC.cpp

```
#include <time.h>

void Hello_SC::main_method(void)
{
  clock_t start, end;

sc_int<64> a = 0;
  start = clock();
  // 100.000 Durchläufe
  for(int i = 0; i < 100000; i++){
    a++;
  }
  end = clock();
  std::cout << "100000x Bool: " << end - start << " Clocks" << endl;

  // entsprechend für sc_bigint
```

2. Simulationsperformance

- Ergebnisse:

```
Info: (I804) /IEEE_Std_1666/deprecated: sc_bit is deprecated, use bool instead

100000x Bool: 318 Clocks
100000x sc_bit: 581 Clocks              => Faktor 1,5
100000x sc_logic: 578 Clocks            => Faktor 1,5
100000x sc_int<64>: 9073 Clocks
100000x sc_bigint<64>: 35357 Clocks     => Faktor 38,9 (!)

100000x Bool: 329 Clocks
100000x sc_bit: 578 Clocks
100000x sc_logic: 579 Clocks
100000x sc_int<64>: 9199 Clocks
100000x sc_bigint<64>: 35324 Clocks
```

- Was verursacht die Abweichungen bei den beiden Läufen?

3. SystemC-Makros

- Finden Sie heraus, welchen C++- Konstrukten die Makros SC_MODULE, SC_METHOD und SC_CTOR entsprechen!
  - Header-Files lesen (aufwendig) oder gcc –E aufrufen!

```
SC_MODULE(test_module) {
  int i;
};
```

```
=> struct test_module : ::sc_core::sc_module {
     int i;
   };
```

```
SC_METHOD(test_method) {
  int i;
};
```

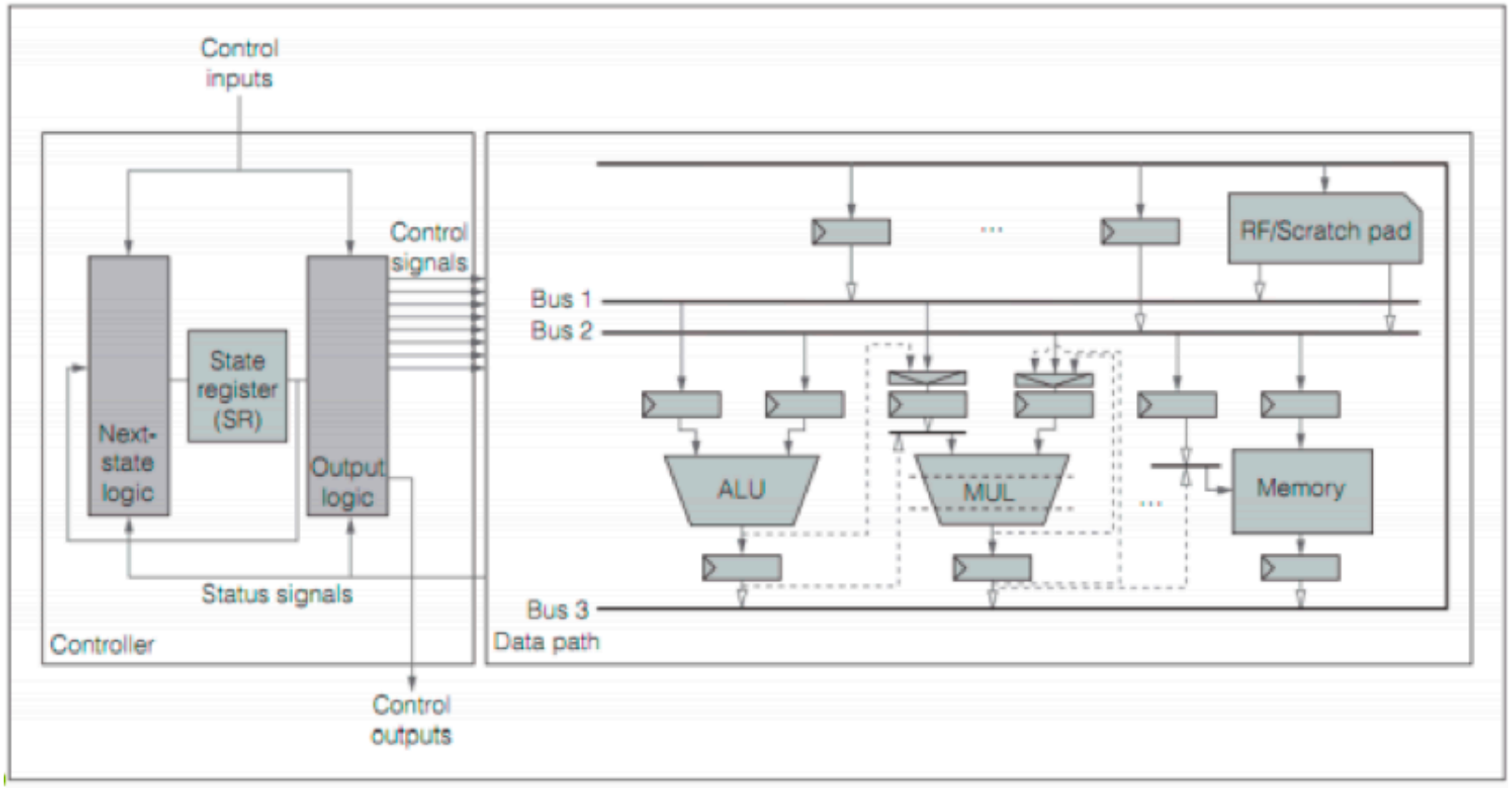
```
=> { ::sc_core::sc_process_handle test_method_handle =
       sc_core::sc_get_curr_simcontext()->
        create_method_process("test_method",
          false, static_cast<sc_core::SC_ENTRY_FUNC>
            (&SC_CURRENT_USER_MODULE::test_method),
          this, 0 );
     this->sensitive << test_method_handle;
     this->sensitive_pos << test_method_handle;
     this->sensitive_neg << test_method_handle; } {
     int i;
   };
```

```
SC_CTOR(test_ctor) {
  int i;
};
```

```
=> typedef test_ctor SC_CURRENT_USER_MODULE;
   test_ctor( ::sc_core::sc_module_name ) {
      int i;
   };
```

4. Abbildung auf RTL-Ebene

- Die folgenden Instruktionen sollen abgebildet werden:
  - a = b – c; d = a AND c;



4. Abbildung auf RTL-Ebene

- *Ohne Bindung:*
  - **state(n):**
  - a=b+c;
  - go to state(n+1);
  - **state(n+1):**
  - d=a AND c;
  - go to state(n+2);
- *Bindung an Speicherelemente:*
  - **state(n):**
  - RF(1)=RF(2)+RF(3);
  - go to state(n+1);
  - **state(n+1):**
  - RF(4)=RF(1)+RF(3);
  - go to state(n+2);
- *Bindung an Funktionseinheiten:*
  - **state(n):**
  - a=ALU1(-, b, c);
  - go to state(n+1);
  - **state(n+1):**
  - d=ALU1(AND, a, c);
  - go to state(n+2);
- *Bindung an FE & Speicherelemente:*
  - **state(n):**
  - RF(1)=ALU1(-, RF(2), RF(3));
  - go to state(n+1);
  - **state(n+1):**
  - RF(4)=ALU1(AND, RF(1), RF(3));
  - go to state(n+2)

4. Abbildung auf RTL-Ebene

- *Mit Bindung an Funktionseinheiten und Speicherelementen und Angabe der Verbindungen:*
  - **state(n):**
  - BUS(1) = RF(2);
  - BUS(2) = RF(3);
  - BUS(3)=ALU(-, BUS(1), BUS(2));
  - RF(1) = BUS(3);
  - go to state(n+1);
  - **state(n+1):**
  - BUS(1) = RF(1);
  - BUS(3)=ALU(AND, BUS(1), BUS(2));
  - RF(4) = BUS(3);
  - go to state(n+2);

Fragen?

- *Fragen!*