technische universität
dortmund

fakultät für informatik
informatik 12

Synthese Eingebetteter Systeme

Sommersemester 2011

2 – SystemC–Überblick

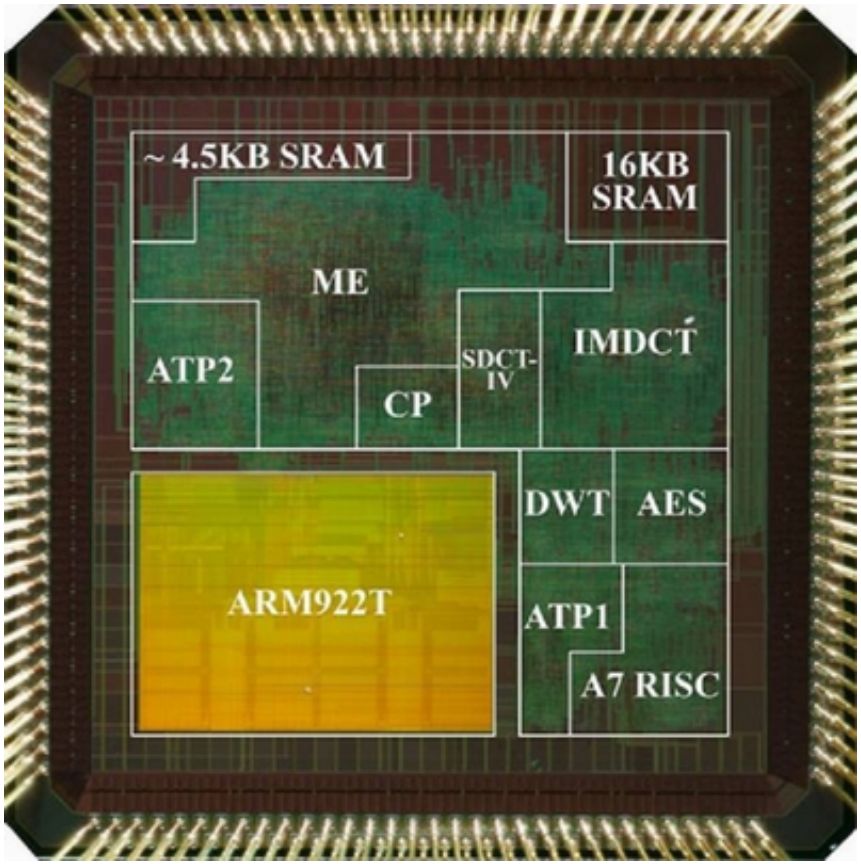
Michael Engel
Informatik 12
TU Dortmund

2011/04/08

Überblick

- ESL-Sprache: SystemC
- SystemC-Struktur und Entwurfsflüsse
- Codebeispiel
- SystemC-Komponenten
  - Zeitverhalten
  - Datentypen
  - Instanziierung

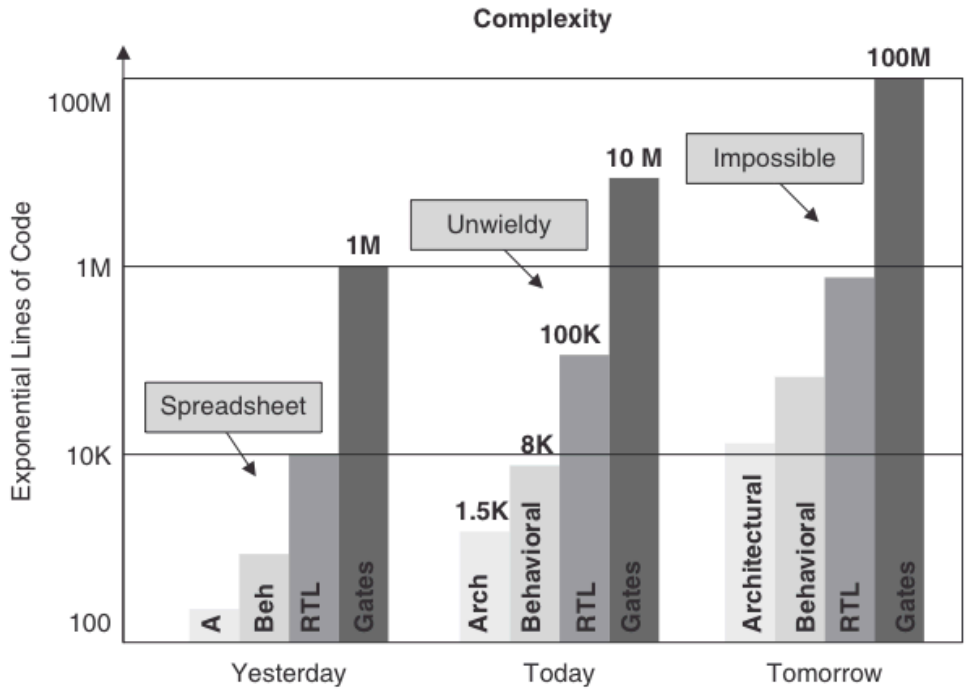
Motivation



- Aktuelle komplexe Systeme bestehen aus HW & SW
- Ein Modell für Hardware und Software!

Komplexität von Entwürfen

- Aktuelle Schaltungen: ~10 Millionen Gatter
- Zukünftige Schaltungen: ~100 Millionen Gatter
  - Diese werden heute entworfen…
  - Größe als RTL: ~1 Million Zeilen!
- Komplexität steigt weiter durch…
  - Mehrere Chips
  - MPSoCs
  - Software!



Gründe für eine ESL-Sprache

Fragestellungen beim Systementwurf:

- Soll eine Funktion in Hardware, in Software oder mit einem besseren Algorithmus implementiert werden?
- Benötigt diese Funktion als Hardware- oder als Software-Implementierung weniger Energie?
- Ist genügend Verbindungsbandbreite für den Algorithmus verfügbar?
- Was ist die minimal benötigte Präzision, damit der Algorithmus funktioniert?

➜ RTL-Beschreibungen sind hier zu unflexibel!

Idee: Verwendung einer *höheren Abstraktionsebene*

- Eingebettete Software meist in C/C++ entwickelt
- Ist C/C++ eine sinnvolle Sprache für ESL-Modellierung?

Nachteile eines C/C++-basierten Entwurfsflusses

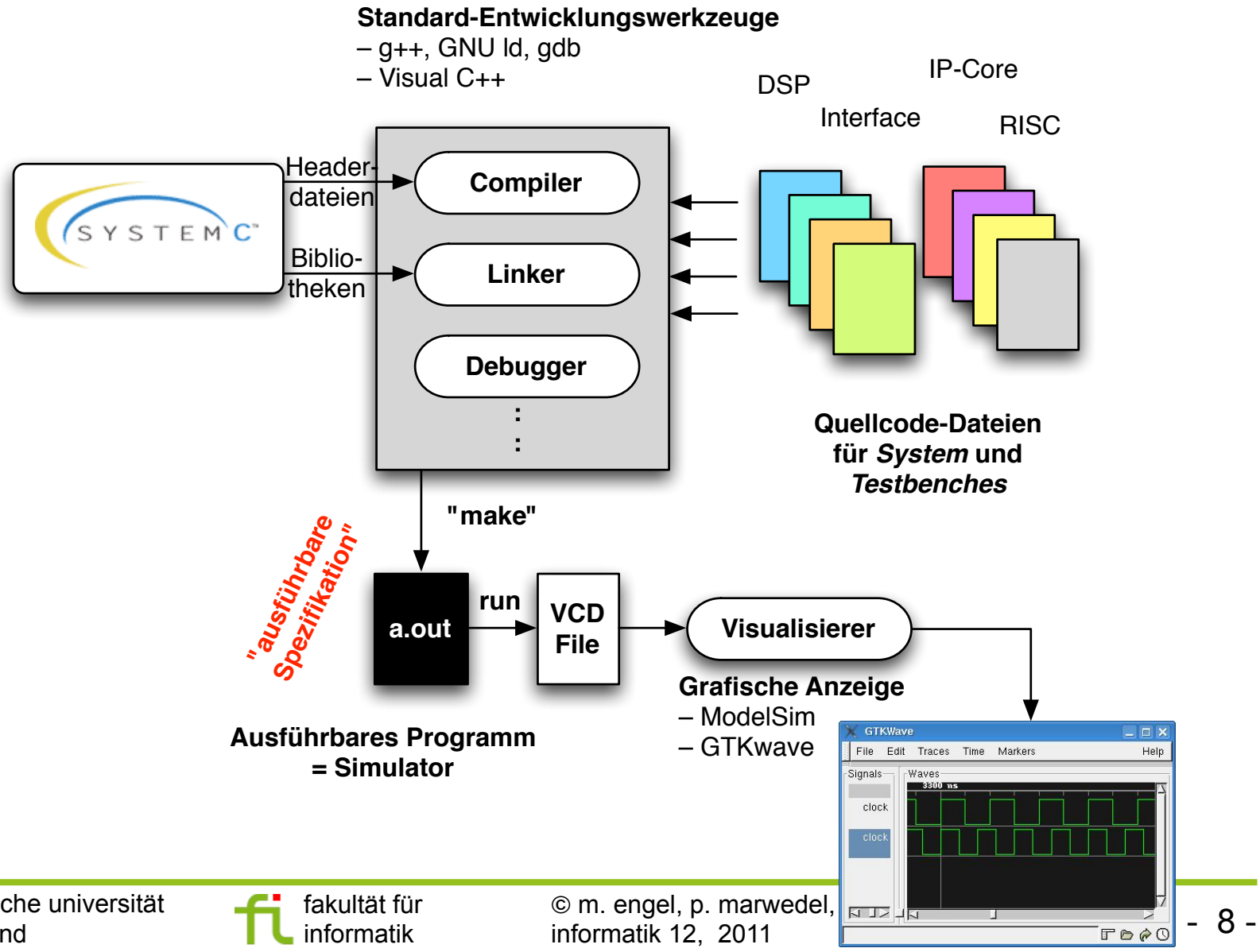
- C/C++ war *nicht* für den Hardwareentwurf gedacht!
- C/C++ mangelt es an
  - Hardware-naher Kommunikation - Signale, Protokolle
  - Zeitdarstellung - Takte, Zeitsequentielle Operationen
  - Nebenläufigkeit - Hardware ist inhärent nebenläufig und arbeitet grundsätzlich parallel
  - Reaktivität - Hardware ist grundsätzlich reaktiv
    - Reagiert auf *Stimuli*
    - Interagiert mit der Umgebung (erfordert Ausnahmebehandlung)
  - Hardware-Datentypen
    - Bits, Bit-Vektoren, Mehrwertige Logiktypen, Festkommatypen
- Fehlende Verbindung zur Hardware beim Debugging

➜ Entwicklung von SystemC (seit 1999)

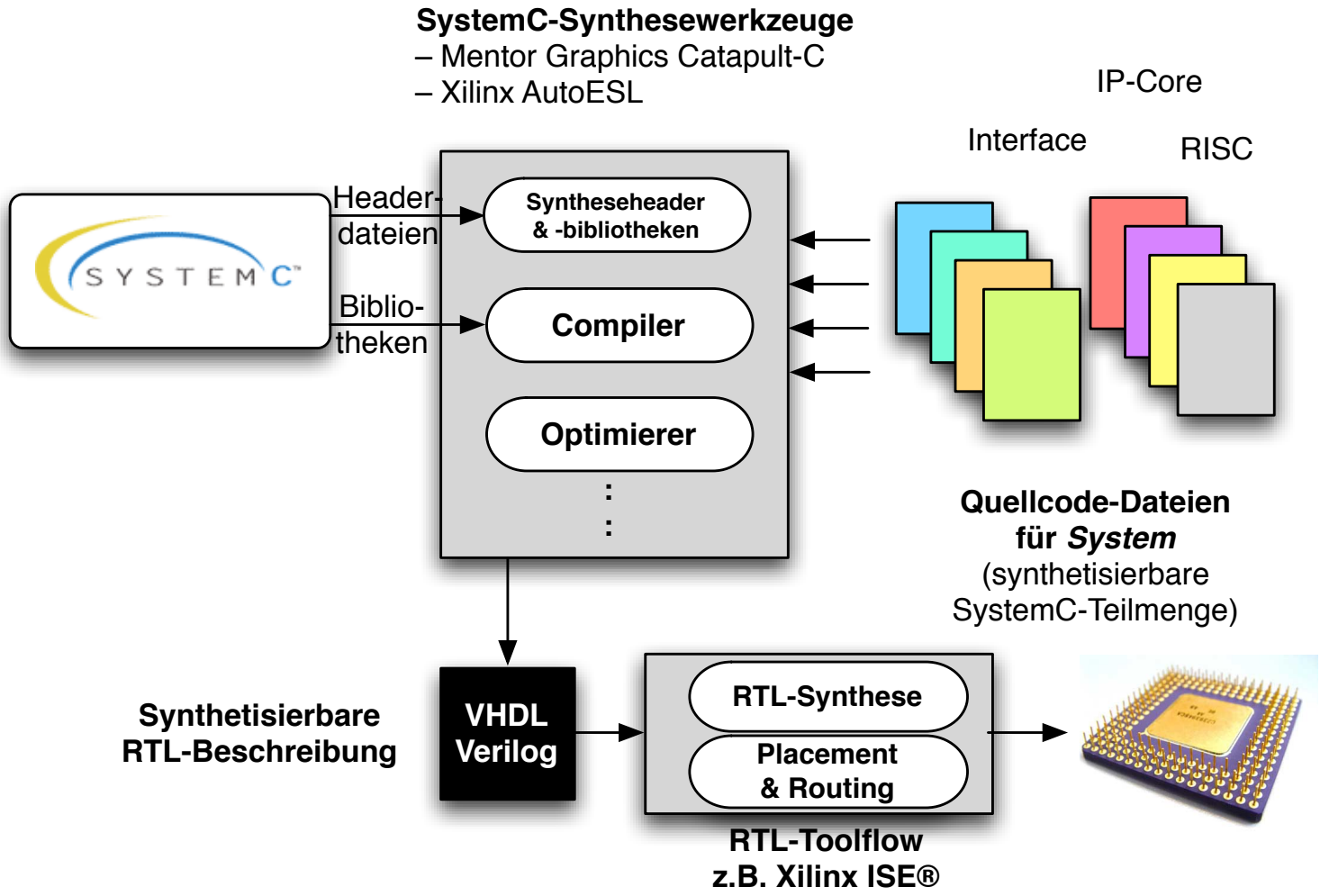
Was ist SystemC®?

- Systementwurfssprache
  - Systeme aus Hard- und Software-Komponenten
- Bibliothek für C++ mit Unterstützung für
  - Funktionalität & Kommunikation,
  - Software & Hardware,
  - Verschiedene Abstraktionsebenen,
  - Nebenläufigkeit,
  - Daten und Kontrollflüsse
- Verwendbar für zyklengenaue Modelle von SW-Algorithmen, HW-Architekturen und Schnittstellen

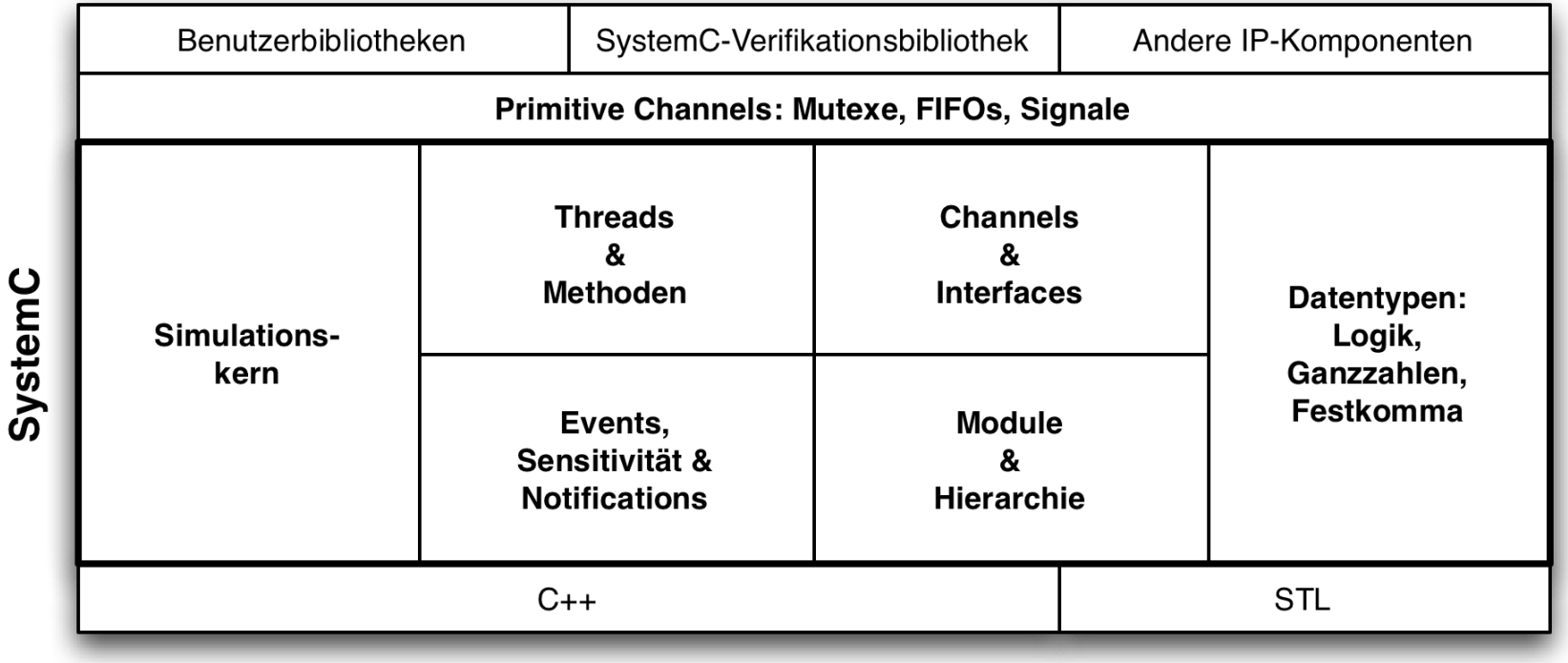
SystemC-Entwurfsmethode: Simulation



SystemC-Entwurfsmethode: Synthese



SystemC-Architektur



SystemC-Spracharchitektur

- Modellierung von Hardware und Software
  - SystemC-Schwerpunkt liegt auf Hardwareentwurf
  - C++-Erweiterungen, um Hardware zu modellieren
- Hardwarenahe Eigenschaften von SystemC:
  - Zeitmodell
  - Hardware-Datentypen
  - Modulhierarchie:
    - Darstellung von Struktur und Hierarchie
  - Nebenläufigkeitsmodell
  - Kommunikationsverwaltung zwischen nebenläufigen Vorgängen
- Auf SystemC-Basis von SystemC sind bekannte Berechnungsmodelle (Models of Computation, MOC) implementierbar
- Untere Ebenen von SystemC können unabhängig von den oberen Ebenen verwendet werden, z.B. für RTL-nahe Modelle

SystemC-Beispiel: Hello, World!

Main.cpp

```
#include <Hello_SC.h>

int sc_main(int argc, char* argv[]) {
  const sc_time t_PERIOD(8,SC_NS);
  sc_clock clk("clk",t_PERIOD);
  Hello_SC iHello_SystemC
   ("iHello_SystemC");
  iHello_SystemC.clk_pi(clk);
  sc_start(10);
  return 0;
}
```

Hello_SC.h

```
#ifndef HELLO_SYSTEMC_H
#define HELLO_SYSTEMC_H

#include <systemc.h>
#include <iostream>

SC_MODULE(Hello_SC) {
  sc_in_clk clk_pi;
  void main_method(void);

  SC_CTOR(Hello_SC) {
    SC_METHOD(main_method);
    sensitive << clk_pi.neg();
    dont_initialize();
  }
};
#endif
```

Hello_SC.cpp

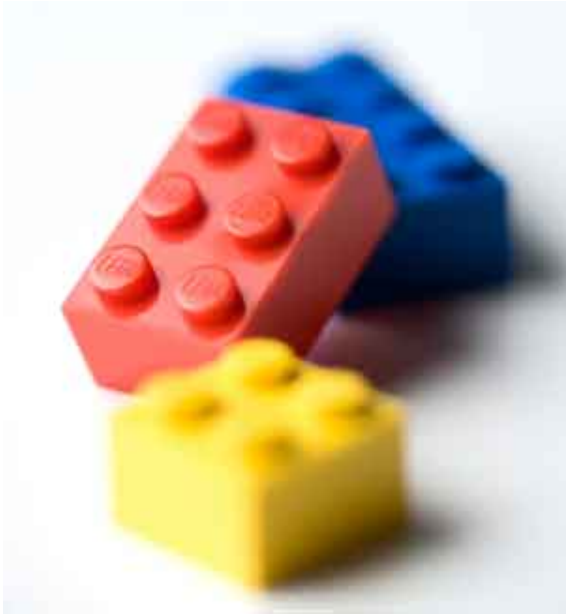
```
#include <Hello_SC.h>

void Hello_SC::main_method(void)
{
  std::cout << sc_time_stamp() <<
  "Hello world!" << std::endl;
}
```

Compilieren mit:

```
g++ -I. -o Hello_SC Hello_SC.cpp \
   Main.cpp -lsystemc
```

SystemC-Komponenten

- Zeitmodell
- Datentypen
- Instanziierung
- Stolpersteine
- Hierarchie und Struktur
- Ports, Interfaces und Kanäle
- Kommunikation
- Nebenläufigkeit
- Module
- Ereignisse, Sensitivität und Notifications



Zeitmodell und Einheiten

- SystemC verwendet ganzzahlige absolute Zeitangaben
- Intern: unsigned integer, ≥ 64 Bit
- Zeit Beginnt bei 0, monoton steigend
- **`sc_time`**-Typ
  - stellt Zeit oder Zeitintervall dar
- Zeitobjekte sind Tupel
  - (Zeitwert, Zeiteinheit)
- Modelle, die einen Takt benötigen, können **`sc_clock`** verwenden

Einheit	Größe
SC_SEC	Sekunden
SC_MS	Millisekunden
SC_US	Mikrosekunden
SC_NS	Nanosekunden
SC_PS	Picosekunden
SC_FS	Femtosekunden

Deklaration und Verwendung

Syntax:

- **sc_time** *name,…;* // keine Initialisierung!
- **sc_time** *name*(*Groesse, Zeiteinheit*),…;

Deklarationen:

- **sc_time** t_PERIOD (5, **SC_NS**);
- **sc_time** t_TIMEOUT(100, **SC_MS**);

Verwendung:

- t_MEASURE = (t_CURRENT – t_LAST_CLOCK);
- **if** (t_MEASURE > t_HOLD) { error (”Zeitfehler"); };
- **wait**(t_HOLD) // Innerhalb **SC_THREAD** erlaubt (s.u.)

sc_start()

`sc_start()` startet die Simulationsphase

- Optionale Zeitangabe als **`sc_time`** begrenzt Simulationsdauer

Beispiele:

- **`sc_start();`** // keine maximale Dauer
- **`sc_start(t_TIMEOUT);`** // Simulation bis t_TIMEOUT

Anzeigen der Zeit

- **`sc_time_stamp()`** liefert aktuelle Simulationszeit
- **`sc_simulation_time()`** liefert aktuelle Zeit als **double**
- Zeit kann mit dem iostream “<<“-Operator angezeigt werden. Beispiele:
  - **`cout << sc_time_stamp() << endl;`**
  - **`std::cout`** `<< " Timeout: " << t_TIMEOUT <<` **`std::endl;`**

Zeitauflösung

- **Zeitauflösung:** kleinste Zeitspanne, die von allen **`sc_time`**–Objekten dargestellt werden kann
  Standardauflösung: 1 ps
  - Standard änderbar vor weiterer Verwendung von **`sc_time`**: **`sc_set_time_resolution`**(*value*,*unit*)
  - Aktuelle Zeitauflösung abfragbar durch Aufruf von **`sc_get_time_resolution`**()
- **Standardzeiteinheit:** wird bei fehlender Einheitenangabe verwendet
  - Änderbar vor weiterer Verwendung von **`sc_time`**: **`sc_set_default_time_unit`**(*value*,*unit*)

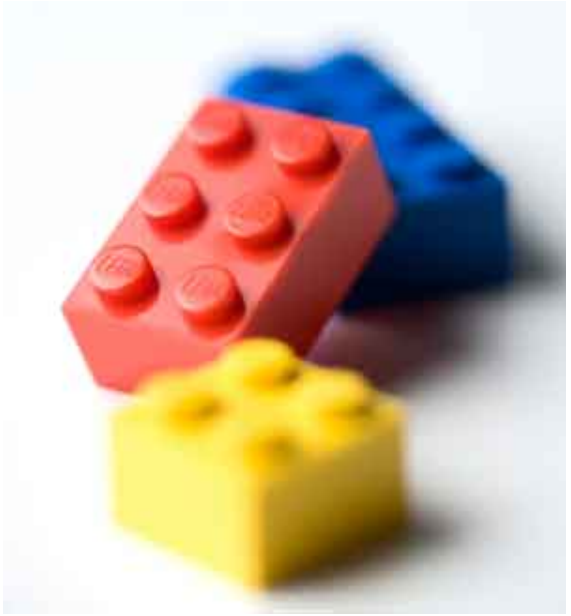
Beispiele

```
int sc_main …
sc_set_time_resolution(1,SC_MS);
sc_set_default_time_unit (1,SC_SEC);
simple_process instance("instance");
sc_start(7200, SC_SEC); // maximale Laufzeit 2h
double t = sc_simulation_time();
unsigned hours = int (t/3600.0);
t–=3600.0*hours;
unsigned minutes = int (t/60.0);
```

Source & ©: D. Black, J. Donovan: SystemC from the ground up, Springer, 2004

SystemC-Komponenten

- Zeitmodell
- Datentypen
- Instanziierung
- Stolpersteine
- Hierarchie und Struktur
- Ports, Interfaces und Kanäle
- Kommunikation
- Nebenläufigkeit
- Module
- Ereignisse, Sensitivität und Notifications



Datentypen

SystemC unterstützt:

- C++-eigene Datentypen
- SystemC-Typen
- Benutzerdefinierte Typen

SystemC-Typen:

- Zweiwertige (‘0’/’1’) Logik und Logikvektoren
- Vierwertige (‘0’/’1’/’Z’/’X’) Logik und Logikvektoren
- Ganzzahlen beliebiger Größe (mit/ohne Vorzeichen)
- Festkomma-Typen
  - Mit/ohne Vorzeichen, mit/ohne Templates

C++-Datentypen

Bits:

- bool

Ganzzahlen:

- (unsigned) char, (unsigned) short, (unsigned) int, (unsigned) long

Gleitkomma:

- float, double, long double

In Simulation: C++-Datentypen *schneller* als die SystemC-Äquivalente

- “bool” häufig zur Darstellung einzelner Bits verwendet

SystemC-Datentypen

- **sc_bit** – Zweiwertige einzelne Bits
- **sc_logic** – Vierwertige einzelne Bits
- **sc_int** – Ganzzahl mit Vorzeichen, 1–64 Bit
- **sc_uint** – Ganzzahl ohne Vorzeichen, 1–64 Bit
- **sc_bigint** – Ganzzahl beliebiger Größe
- **sc_biguint** – Ganzzahl beliebiger Größe (ohne VZ.)
- **sc_bv** – Beliebig großer zweiwertiger Vektor
- **sc_lv** – Beliebig großer vierwertiger Vektor
- **sc_fixed** – Festkommatyp (Template) mit VZ.
- **sc_ufixed** – Festkommatyp (Template) ohneVZ.
- **sc_fix** – Festkommatyp mit VZ.
- **sc_ufix** – Festkommatyp ohneVZ.

Typ sc_bit

sc_bit ist ein zweiwertiger Datentyp

- Stellt ein einzelnes Bit dar
- Eine Variable vom Typ **sc_bit** kann nur die Werte '0'(false) oder '1'(true) annehmen

sc_bit-Operatoren

Bitweise	&(und)	|(oder)	^(xor)	~(nicht)
Zuweisung	=	&=	|=	^=
Gleichheit	==	!=		

Deklaration eines Objekts vom Typ **sc_bit**:

```
sc_bit s;
```

- Aus Performancegründen (Simulation) wird oft der Typ **bool** anstelle von **sc_bit** verwendet!

Typ sc_logic

Einzelnes Bit mit vier möglichen Werten:

- ’0’(false), ’1’(true)
- ’X’ (unbekannt) und ’Z’ (hochohmig oder offener Kontakt)

Modelliert Busse mit mehreren Teilnehmern, Weitergabe von ‘X’-Werten, Initialisierungswerte und offene Busse

`sc_logic`-Operatoren: wie bei Typ **`sc_bit`**

Beispielzuweisung:

```
sc_logic x;          // Deklaration
x = ’1’;             // assign a 1 value
x = ’Z’;             // assign a Z value
```

- Verhalten der Operatoren für ‘X’- und ‘Z’-Werte? ➜ Übung!

Vorzeichenlose Ganzzahlen mit fester Breite

SystemC-Ganzzahlen sind vorzeichenlos und vorzeichenbehaftet mit Breiten von 1 bis 64 Bit verfügbar

- **`sc_int<n>`** VZ-behaftete Ganzzahl mit **n** Bit Breite
- **`sc_uint<n>`** Vorzeichenlose Ganzzahl mit **n** Bit Breite

Vorzeichenbehaftete Typen als Zweierkomplement dargestellt

Beispiele:

```
sc_int<64> x;      // deklariert 64-Bit Ganzzahl mit VZ.
sc_uint<48> y;     // deklariert 48-Bit Ganzzahl ohne VZ.
```

Ganzzahlen fester Breite

Ganzzahloperatoren

Bitweise	~	&	|	^	>>	<<			
Arithmetisch	+	-	*	/	%				
Zuweisung	=	+=	-=	*=	/=	%=	&=	|=	^=
Gleichheit	==	!=							
Relational	<	<=	>	>=					
Autoinkrement	++								
Autodekrement	--								
Bitselektion	[x] ➜ überladener Array-Operator								
Teilbitfolge	range()								
Konkatenation	(,) ➜ überladener Reihenfolge-Operator								

Ganzzahlen mit beliebiger Breite

- Für Operanden mit mehr als 64 Bit:
  - **`sc_biguint<n>`** (VZ-lose Ganzzahl bel. Breite)
  - **`sc_bigint<n>`** (Ganzzahl bel. Breite mit VZ.)
- Für Ganzzahlen beliebiger Größe verwendbar, nur durch Systemeigenschaften begrenzt
  - Simulation deutlich **langsamer** als **`sc_(u)int`**!

Operatoren identisch zu denen für Ganzzahlen fester Breite

Die Typen **sc_biguint, sc_bigint, sc_int, sc_uint** und normale C++-Typen können in Ausdrücken zusammen verwendet werden.
Typkonversion durch Zuweisungsoperator “=“

Bitvektoren beliebiger Länge

- Zweiwertiger Vektor beliebiger Länge: `sc_bv`-Typ für die Manipulation großer Bitvektoren

Operatoren auf Bitvektoren beliebiger Länge						
Bitweise	~	&	|	^	<<	>>
Zuweisung	=	&=	|=	^=		
Gleichheit	==	!=				
Bitselektion	[x]					
Teilbitfolge	range()					
Konkatenation	(,)					
Reduktion	and_reduce()	or_reduce()	xor_reduce()			

Logikvektoren beliebiger Länge

- **sc_lv<n>** ist ein Logikvektor beliebiger Länge, jedes Bit kann einen von vier Werten annehmen
- **sc_lv<n>** ist variabel großes Array von sc_logic-Objekten

Deklarationsbeispiel:

```
sc_signal<sc_lv<64> > databus;   // 64 Bit breites Signal
                                 // mit Namen "databus"
```

sc_lv verwendet die selben Operatoren wie **sc_bv**.
sc_bv wird wesentlich schneller als **sc_lv** simuliert!

sc_bv / sc_lv

- **Eigenschaften:**
  - Werte zwischen **sc_bv** und **sc_lv** zuweisbar
  - Zuweisung konstanter Werte über Stringkonstanten
  - Konversion zwischen **sc_bv/sc_lv** und SystemC-Ganzzahltypen ist verfügbar
  - Keine arithmetischen Operationen

Festkomma-Typen

- DSP-Anwendungen benötigen oft Festkommadatentypen
- SystemC besitzt Festkommatypen mit & ohne Vorzeichen, um Hardware akkurat zu modellieren

4 Basis-Festkommatypen:

- **sc_fixed:** statischer Wert (zur Compilezeit bekannt)
- **sc_ufixed:** dto.
- **sc_fix:** variable Werte (zur Laufzeit konfigurierbar)
- **sc_ufix:** dto.

Zusätzlich gibt es ”schnelle ” Versionen **sc_fix_fast**, ...

- Auf 53 Bit beschränkt
- **#define SC_INCLUDE_FX** vor **#include** <systemc.h>

Festkomma-Typen

Beispiel:

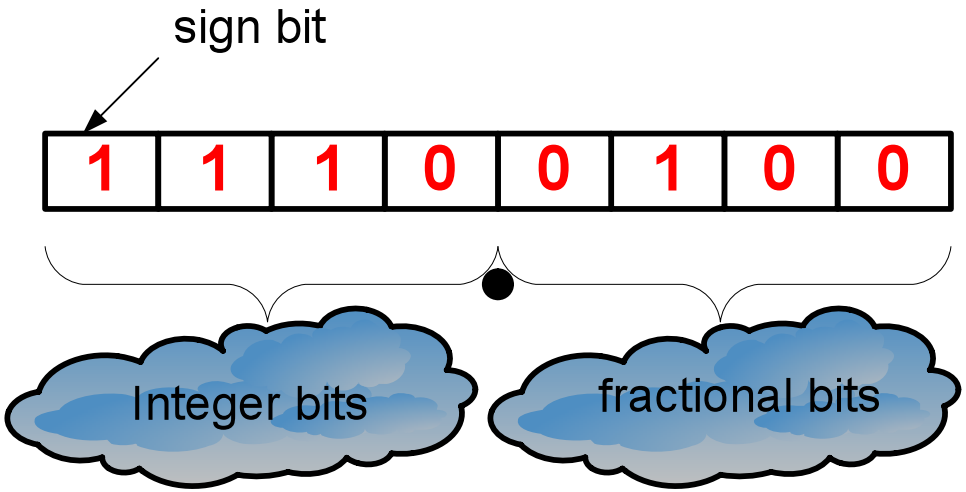
$(1.75) = (0001.1100)_2$

4

8

- 1er-Komplement von (0001.1100)$_2$ = (1110.0011)$_2$
- 2er-Komplement von (0001.1100)$_2$ = (1110.0100)$_2$

- *my_var*:



Festkomma-Typen

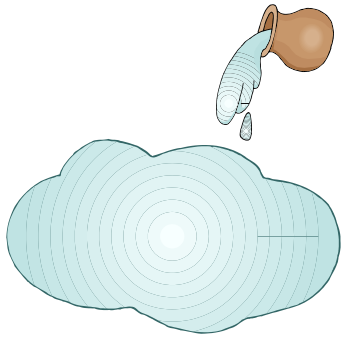
- Syntax zur Deklaration von Festkomma-Objekten:
  - `sc_fixed<wl, iwl, q_mode, o_mode, n_bits> x;`
  - `sc_ufixed<wl, iwl, q_mode, o_mode, n_bits> y;`
  - `sc_fix x(Liste der Optionen);`
  - `sc_ufix y(Liste der Optionen);`

Mit

- wl Gesamte Wortlänge
- iwl # Bits links vom Dezimalkomma
- q_mode Quantisierungsmodus
- o_mode Überlaufmodus
- n_bits Anzahl saturierter Bits
- x,y Objektname, Name des Festkomma-Objekts

Bedeutung der Optionen

Name	*Überlaufsteuerung*
SC_SAT	Sättigungsarithmetik
SC_WRAP	Überlauf
…	(weitere Optionen)



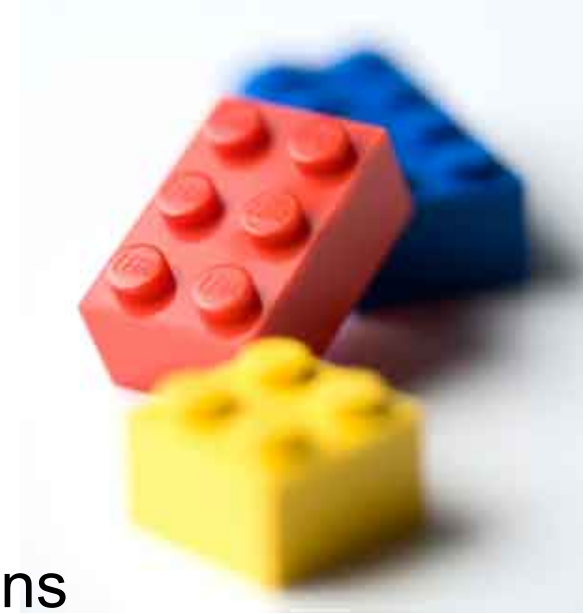
Name	*Quantisierungsmodus*
SC_RND	Runden
SC_RND_ZERO	Nach Null hin runden
SC_RND_MIN_INF	Nach –∞ hin runden
SC_RND_INF	Nach +∞ hin runden
SC_TRN	Abschneiden
…	(weitere Optionen)

Hohe Abstraktionsebenen und die STL

- STL-Containerklassen sind verwendbar
  - string
  - vector
  - map
  - list
  - deque
- Außerdem
  - for_each
  - count
  - min_element
  - reverse
  - sort
  - search
  - ….

SystemC-Komponenten

- Zeitmodell
- Datentypen
- Instanziierung
- Hierarchie und Struktur
- Ports, Interfaces und Kanäle
- Kommunikation
- Nebenläufigkeit
- Module
- Ereignisse, Sensitivität und Notifications
- Stolpersteine



sc_main()–Funktion

- **sc_main()** ist der Einsprungpunkt aus der SystemC-Bibliothek in die Simulation ➜ main() ist Teil von SystemC!
  - Prototyp identisch zu main:
    int sc_main(int argc**, char*** argv[] **);**
  - Argumente argc und argv[] sind die gewohnten Befehlszeilenparameter
    - An **sc_main()** von der **main()**–Funktion übergeben
  - Funktion **sc_main()** konfiguriert Simulationsvariablen (Zeiteinheit, Zeitauflösung etc.), instanziiert die Modulhierarchie und Kanäle, startet die Simulation start, räumt danach auf und liefert Statuscode zurück

sc_main()–Funktion

- Instanziierungs-Syntax:

  module_type module_instance_name(“string_name“);

- mit:
  - module_type: Typ des Moduls
    (von **sc_module** abgeleitete Klasse)
  - module_instance_name: Name der Modulinstanz
    (Objektname)
  - string_name: String, mit der die Modulinstanz initialisiert wird

sc_main()–Funktion

- Nach der Instanziierung in **sc_main()** können die Ports eines Moduls an Kanäle gebunden werden
- Explizites Binden von Ports:

  module_instance_name.port_name(channel_name);

- mit:
  - port_name Instanzname des zu bindenden Ports
  - channel_name Instanzname des Kanals, an welche der Port gebunden werden soll

Ausblick: SystemC–Beispiel

- „Echte“ Hardware
  - 8-Bit-Zähler
- Eingänge:
  - Enable
  - Clock
  - Reset
- Ausgänge:
  - Out
- Internes Signal:
  - Count

```
#include "systemc.h"

SC_MODULE (up_down_counter) {
  //----------Input Ports--------------
  sc_in     <bool> enable, clk, reset;
  //----------Output Ports-------------
  sc_out    <sc_uint<8> > out;
  //----------Internal Variables-------
  sc_uint<8>  count;

  //----------Code Starts Here---------
  void counter () {
    if (reset.read()) {
      count = 0 ;
    } else if (enable.read()) {
      count = count + 1;
    }
    out.write(count);
  }

  SC_CTOR(up_down_counter) {
    SC_METHOD (counter);
      sensitive << clk.pos();
  }

};
```

Stolpersteine

Den ersten Entwurf zu übersetzen kann nervenaufreibend sein – ein paar Tips:

- Häufigster Fehler:
  - Vergessen von “;” am Ende von SC_MODULE
  - SC_MODULE ist ein Makro, das eine Klasse darstellt (➜Übung)
- SystemC verwendet viele Templates
  - Problem (z.B. bei gcc): *geschachtelte Templates*
  - `sc_out<sc_uint<8>>` output;
  - Verwechslung mit Rechtsschiebe-Operator “>>”

```
design.cc:5: error: template argument 1 is invalid
```

  - Lösung – **Leerzeichen:** `sc_out<sc_uint<8> >`

Zusammenfassung

- Komplexere elektronische Systeme
  - ...erfordern Entwurf auf Systemebene
  - ...ESL-Sprache: SystemC
- SystemC
  - Library für C++
  - Modellierung von Hardware-Datentypen und -eigenschaften
  - Simulation und Synthese
    - nicht alle Konstrukte sind synthetisierbar!
- SystemC-Sprachkonstrukte
  - Zeitverhalten
  - Datentypen
  - Instanziierung
  - „Klassische“ Fehler von SystemC-Anfängern

Literatur

1. D. Black, J. Donovan: SystemC: From the Ground Up, Second Edition, *Springer,* 2010
2. http://www.doulos.com/knowhow/systemc
3. SystemC 2.2 Language Reference Manual
4. SystemC™ Version 2.2 User's Guide, http:// www.systemc.org
5. Joachim Gerlach: System-on-Chip Design with SystemC, U. Tübingen, Wilhelm-Schickard-Institut, Department of Computer Engineering
6. Thorsten Grötker, Stan Liao, Grant Martin, Stuart Swan, System Design with SystemC™, *Kluwer Academic Publishers*
7. SystemC Quick Reference Card:

http://comelec.enst.fr/hdl/sc_docs/systemc_quickreference.pdf