Zusammenfassung Architektur Eingebetteter System

Paul Nykiel

26. Juli 2019

This page is intentionally left blank.

Inhaltsverzeichnis

Kapitel 1

Einführung

Ein eingettetes System ist in einen technischen Kontext oder Prozess eingebettet.
Im wesentlichen kann ein eingebettetes System als ein Computer, der einen technischen Prozess steuert oder regelt, betrachtet werden. Grafik

1.1 Architektur eines Eingebetteten Systems

1.1.1 Eigenschaften eines Eingebetteten Systems

- Enge Verzahnung zwischen Hard- und Software
- Strenge funktionale und zeitliche Randbedinungen
- Zusätzlich zum Prozessor wird I/O Hard- und Software benötigt
- Oftmals wird Anwendungsspezifische Hardware benötigt

⇒ Keine „General-Purpose“ Lösung möglich
Zusätzliche Probleme:

- Wenig Platz
- Nur beschränkte Energiekapazität
- System darf nicht warm werden
- Kostengünstig

1.1.2 Zusätzliche Herausforderungen beim Entwurf

Die Entwicklung eines eingebetteten Systems ist kein reines Software-Problem, zusätzlich muss beachtet werden:

- Auswahl eines Prozessors, Signalprozessors, Microcontrollers

- Ein-/Ausgabe Konzept&Komponenten
  - Sensoren und Aktoren
  - Kommunikationsschnittstellen
- Speichertechnologien und Anbindung
- Systempartitionierung: Aufteilen der Funktionen der Komponente
- Logik- und Schaltungsentwurf
- Auswahl geeigneter Halbleitertechnologien
- Entwicklung von Treibersoftware
- Wahl eines Laufzeits-/Betriebssystems
- Die eigentliche Softwareentwicklung

⇒ Aufteilung des Entwurfs auf mehrer Entwurfsebene

1.1.3 Entwurfsebenen

Verhalten	Syntheseschritt	Entscheidungen	Test
System Specification	Systemsynthese	HW/SW/OS	Modelsimulator / Checker
Behavioural Specification	Verhalten / Architektursynthese	Verarbeitungseinheiten	HW/SW-Simulation
Register-Transfer-Specification	RT-Synthese	Register, Addierer, Mux	HDL-Simulation
Logic-Specification	Logiksynthese	Gatter	Gate-Simulation

Tabelle 1.1: Entwurfsebenen

Graphik

1.2 Hardwarespezifikationssprachen

- Verilog
- VHDL (Very High Speed Integrated Circuit Description Language)

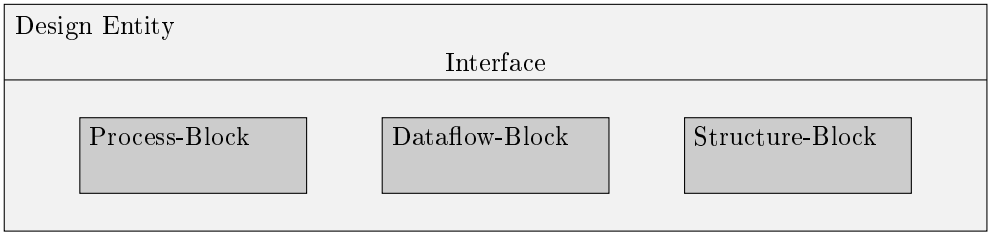

Abbildung 1.1: Aufbau einer Design-Entity

Process-Block Sequentiell abgearbeitete Logik:

```
process(clk)
begin
    ...
end
```

Dataflow-Block Konkurrent abgearbeitete Logik:

```
begin
    ...
end
```

Structure-Block Zusammenschalten weiterer Design-Entitys:

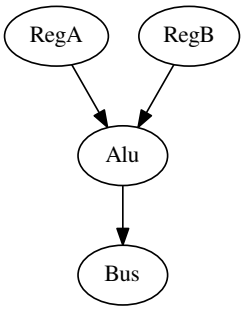

Abbildung 1.2: Structure-Block

1.2.1 Aufbau von VHDL-Beschreibungen

- use: Import von Bibliotheken

- `entity`: Schnittstellenbeschreibung
- `architecture`: Implementierung der Entity
- `configuration`: `architecture` zu `entity` auswählen

1.2.2 Beispiel: Multiplexer

Entity-Deklaration:

```
entity MUX is
    port(a,b,sel: in Bit;
         f: out Bit);
end MUX;
```

Als Process-Block

```
architecture BEHAVIOUR_MUX of MUX is
begin
    process(a,b,sel)
    begin
        if sel = '1' then f <= a;
        else f <= b;
    end process;
end BEHAVIOUR_MUX;
```

Als Dataflow-Block

```
architecture DATAFLOW_MUX of MUX is
begin
    f <= a when sel = '1' else b;
end DATAFLOW_MUX;
```

alternativ geht auch:

```
architecture DATAFLOW_MUX of MUX is
begin
    f <= (a and sel) or (b and (not sel));
end DATAFLOW_MUX;
```

eine weitere Option:

```
architecture DATAFLOW_MUX of MUX is
signal nsel, f1, f2 : Bit;
begin
    nsel <= not sel;
    f1 <= a and sel;
    f2 <= b and nsel;
    f <= f1 or f2;
end DATAFLOW_MUX;
```

Alternativ: Mit Variablen

Laut Skript geht das so nicht, sollte aber eigentlich schon?

Als Structure-Block

```
architecture STRUCTURE of MUX is
    component NOT
        port(i: in Bit; o: out Bit);
    end component;
    component AND
        port(i1, i2: in Bit; o: out Bit);
    end component;
    component OR
        port(i1, i2: in Bit; o: out Bit);
    end component;
    signal nsel, f1, f2: Bit;
begin
    g1: AND port map(a, sel, f1);
    g2: AND port map(b, nsel, f2);
    g3: OR port map(f1, f2, f);
    g4: NOT port map(sel, nsel);
end STRUCTURE;
```

1.3 Configuration

Rekursive die Architektur für jede Entity auswählen:

```
configuration STRUCTURE_MUX of MUX is
    for STRUCURE
        for g1,g2: AND use entity MYAND(BEHAVIOUR_AND)
            ...
        end for;
    end for;
end STRUCTURE_MUX;
```

Was genau passiert da jetzt?

Dann muss die oben genutze Entity und Architektur natürlich noch definiert werden:

```
entity MYAND is
    port(i1, i2: in Bit;
         o: out Bit);
end MYAND;

architecture BEHAVIOUR_MYAND is
    o <= i1 and i2;
end BEHAVIOUR_MYAND;
```

1.4 VHDL-Simulationssemantik

Aufgaben des Simulators:

- Signal treiben/propagieren
- Rückkopplungen auflösen
- Verzögerungen modellieren

1.4.1 Signale Treiben