Präsentation Diplomarbeit

Portierung eines Java Frameworks zur Erzeugung von parametrisierten Hardware Strukturen auf eine neue Generation von rekonfigurierbaren Logikbausteinen

Bearbeiter: Sven Mitlehner

Prüfer: Prof. Dr. Andreas Koch

Technische Universität Darmstadt

Fachbereich Informatik

Fachgebiet Eingebettete Systeme und ihre Anwendungen

Agenda

Motivation

JHDL Überblick

Xilinx Virtex Überblick

Aufgabenstellung

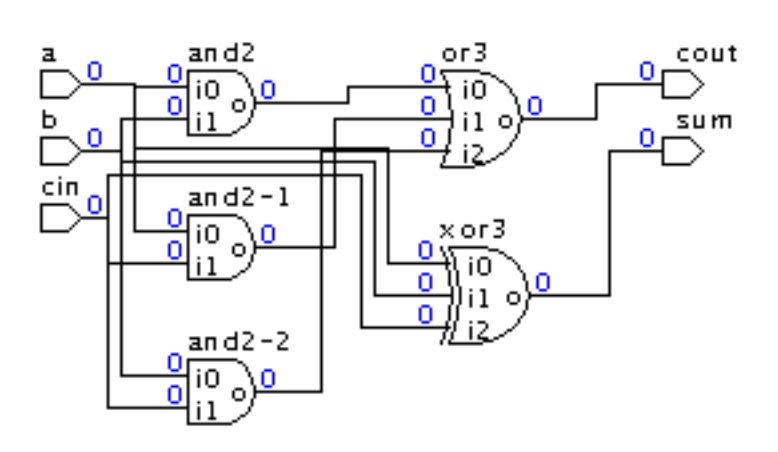
Erweiterung von JHDL

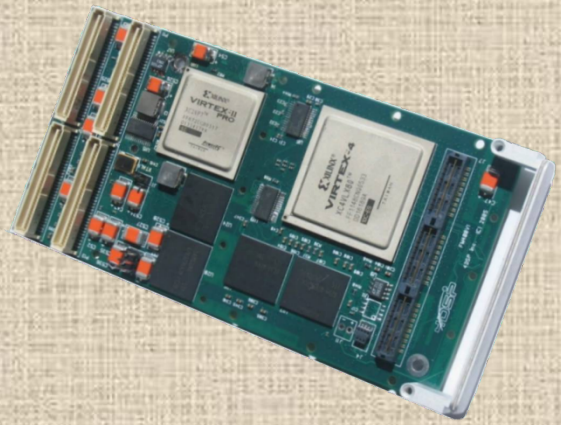
JHDL Tools: Code Generator

JHDL Tools: Testframework

Fragen / Diskussion

Motivation





Motivation

Programmierung von rekonfigurierbaren Logikbausteinen (FPGAs)

Traditionelle Vorgehensweise: Programmierung mittels einer Hardware Description Language (z.B. VHDL oder Verilog)

Nachteile:

- Hohe Komplexität des zu schreibenden Codes
- Niedriges Abstraktionslevel

Lösung:

- Verwendung von „Hochsprachen“ zur Hardwareentwicklung

JHDL Überblick

JHDL ist eine Hardware Description Language aufbauend auf Java

Vorteile:

- Höheres Abstraktionslevel
- Bessere Übersichtlichkeit
- Schnellere Entwurfszyklen
- Schaltkreise werden dargestellt als Java Klassen
- Verwendung von bekannten Java Sprachkonstrukten und Bibliotheken zur Implementierung und Simulation von Schaltkreisen
- Unterstützt die Programmierung verschiedener FPGAs
- Frei erweiterbar um neue Module
- Enthält Tools zur Simulation und zum Debuggen von Schaltkreisen

Xilinx Virtex Überblick

Xilinx ist Marktführer bei der Entwicklung von High-End FPGAs

Einführung der Virtex Gerätefamilie im Jahr 1998

Das aktuelle Modell Virtex-5 wurde 2006 vorgestellt

Features:

- Hohe Performance
- Hohe Kapazität (bis zu 330,000 logische Zellen)
- Enthält komplexe Module für verschiedene Aufgabenbereiche, z.B. Digital Signal Processing, Ethernet Kommunikation, PCI Express Kommunikation

Aufgabenstellung

Die JHDL Bibliothek enthält bereits Module für die Programmierung von Xilinx Virtex und Xilinx Virtex 2

Aufgaben:

- Erweiterung von JHDL um ein Modul zur Unterstützung der neuen Virtex 5 Gerätegeneration
- Unterstützung der neu eingeführten Primitive sowie der im Virtex 2 Modul nicht implementierten Primitive
- Erstellen eines Benutzerhandbuchs zur Implementierung von Modulen für weitere Geräte
- Programmierung von Bibliotheken zur vereinfachten Implementierung und Verifikation von Modulen

Erweiterung von JHDL

Erstellung jeweils einer Klasse für jedes Primitiv

Festlegung der Eigenschaften eines Primitivs

- Eingänge / Ausgänge
- Name
- Schematisches Symbol
- Simulationsverhalten

```
public final void propagate() {
    if (s.get(this) == 0) {
        o.put(this, i0.get(this));
    }
    else {
        o.put(this, i1.get(this));
    }
}
```

Erweiterung von JHDL

Problematik: Die Erstellung von Klassen beinhaltet einen großen Anteil mechanisch zu wiederholender Tätigkeiten

- Konstruktoren
- Definition von Ein- und Ausgängen
- Definition von Attributen
- Prüfen von Wertebereichen für Attribute

Das wiederholte Programmieren von Programmteilen mit fast identischer Funktion für verschiedene Primitive verzögert die Entwicklung erheblich!

Lösung: JHDL Tools

JHDL Tools: Code Generator

Idee:

- Einmalige Definition aller zu einem Primitiv gehörenden Ein- und Ausgänge sowie Attribute mit zugehörigem Wertebereich
- Entwurf eines Simulationsmodells
- Alle Funktionen, welche auf die oben definierten Eigenschaften zugreifen, werden automatisch generiert
- Bei einer Änderung der Spezifikationen muss der Quellcode nur zentral an einer Stelle modifiziert werden

JHDL Tools: Code Generator

Beispiel: bufgmux.java

```
public static CellInterface[] cell_interface = {
    in("i0", 1),
    in("i1", 1),
    in("s", 1),
    out("o", 1)
};
```

```
public bufgmux(Node parent, Wire i0, Wire i1, Wire s, Wire o) {
    super(parent);
    ArgBlockList abl = new ArgBlockList();
    abl.insert("i0", i0);
    abl.insert("i1", i1);
    abl.insert("s", s);
    abl.insert("o", o);
    connectAllWires(abl);
    init();
}
```

JHDL Tools: Testframework

Bisher:

- Jede Primitive-Klasse enthält eine test Methode.
- Jede Primitive-Klasse ist ausführbar. Die main Methode ruft die test Methode auf
- Auf der Konsole wird ausgegeben, ob er Test erfolgreich war

Probleme:

- Viel redundanter Code (Initialisieren des Testframeworks sowie der zu testenden Klasse)
- Programmcode und Testfälle nicht voneinander getrennt
- Komplizierte Implementierung „dynamischer Testfälle“

JHDL Tools: Testframework

- Nur die für einen Testfall wesentlichen Informationen müssen vom Entwickler programmiert werden
- Testfälle werden zentral in einem Package oder einer Klasse definiert.
- Bei Bedarf können diese Testfälle auch mit JUnit oder ähnlichen Frameworks verwendet werden

```
t.runTest(bufgce.class, new String[] {"i", "ce", "o"}, new long[][] {
    {0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1},
    {0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0}
});
```

Vielen Dank für ihre Aufmerksamkeit!

Fragen?