

FPGA Development

Sven Gregori, CN8
<gregori@hs-furtwangen.de>

UnFUG SS 2008
Hochschule Furtwangen

29. Mai 2008

Überblick

- ① FPGA
- ② VHDL
- ③ IP Cores
- ④ praktische Umsetzung
- ⑤ Entwicklungs-Software
- ⑥ Fazit
- ⑦ Links

FPGA



FPGA

- Field Programmable Gate Array
→ "vor Ort programmierbarer Logikbaustein"
- frei programmierbarer Baustein
- Chip allein macht an sich eigentlich garnichts
- erst durch Beschreibung ("Programmierung") wird er zu dem, was er sein soll



frei programmierbar

- theoretisch kann man alles aus der Digitalwelt in einem FPGA realisieren
- zum Beispiel
 - simpelste Logikgatter
 - Adress Decoder, 7 Segment Decoder
 - digitale Audio Filter, Bildverarbeitung
 - Arithmetikeinheiten (FPU, elliptische Kurven Arithmetik, ...)
 - Crypto-Einheiten
 - RAID Controller, GPU, CPU, μ C, ...



Möglichkeiten

- echte Parallelisierung
- optimale Anpassung an Anwendung
- z.B. Busbreiten können direkt auf Daten abgestimmt werden
- komplette Anwendung auf einem Chip - alle Komponenten in FPGA implementieren, selbst Speicher, CPU etc. möglich



Anwendungsgebiete

- Generische Bausteine
 - Microcontroller, Mikroprozessoren, Co-Prozessoren, Controller, Accelerators, ...
- im Bereich
 - Digitale Signalverarbeitung, Kryptographie, Medizintechnik, Rüstung, Luft- und Raumfahrt, Unterhaltungselektronik, ...
- Parallelisierung
- ASIC Prototyping



Aufbau

- Configurable Logic Blocks (CLBs)
- I/O Blöcke
- RAM Blöcke
- Multiplizierer
- Digital Clock Manager (DCM)
- je nach Modell noch zusätzlich
 - eingebetteter Microprozessor oder -controller
 - Ethernet MAC(s)
 - DSP Einheiten
 - ...



Configurable Logic Block

- Lookup Tables (LUTs)
 - z.B. 4 Eingänge, 1 Ausgang
 - direkte Abbildung einer Wahrheitstabelle
 - auch als RAM (16x1) verwendbar
 - oder als Schieberegister
- Speicherelemente (Flip Flops)
- Logikgatter und Multiplexer fürs Zusammenspiel



Block RAM

- z.B. 20 Blöcke mit je 16kBit Daten
- Single- und Dual Port möglich
- Wortbreiten einstellbar
z.B. 4Kx4 oder 8Kx2 oder 16Kx1



Digital Clock Manager

- Generierung verschiedener Taktfrequenzen
- Taktverdopplung
- Takthalbierung
- Taktphasendrehung
- Ausgleich von Taktverzögerungen physikalischer Natur



vom nackten FPGA zum Baustein

- System wird mit Hardwarebeschreibungssprache definiert
- VHDL oder Verilog als "klassische Sprachen"
- SystemC
 - eher zur Systemmodellierung und Simulation
 - an sich keine Sprache sondern C++ Bibliothek
 - hat komplett C++ im Hintergrund und dessen Syntax
 - dadurch höhere Abstraktion
 - bisher hauptsächlich für Simulation verwendet

VHDL



VHDL

- **V**ery High Speed Integrated Circuit **H**ardware **D**escription Language
- in den 80er Jahren entwickelt und vom DoD gefördert
- genormt durch IEEE 1076
- verwendet Ada Konzepte und Syntax
- ermöglicht Beschreibung ("Programmierung") von digitalen Bausteinen



Aufbau eines Bausteins in VHDL

- Entity
 - beschreibt äußere Struktur des Bausteins
 - Deklaration von Eingabe und Ausgabe Ports
 - optionale definition generischer Attribute
- Architecture
 - eigentliches Verhalten des Bausteins
 - verwendet intern *Signale*

```

○○○○
○○○○
○

```

```

○○●
○○○○
○○

```

```

○○○

```

```

○
○
○
○○○
○

```

```

○
○○
○○
○
○

```

```

○○○

```

```

○○

```

Beispiel

```

entity gate is
  port (
    a: in  bit;
    b: in  bit;
    y: out bit;
  );
end gate;

```

```

architecture behaviour of gate is
begin
  y <= a and b;
end behaviour;

```



Sprachkonzepte

- Syntax ist case insensitive
- Flusskontrolle (if, for, case, ...)
- Funktionen
- Code innerhalb eines process Blocks wird sequenziell ausgeführt, Code ausserhalb parallel
- Event gesteuerte Ausführung (z.B. Änderung eines Eingangsignals)
- Zusammenschalten und Zusammenfassen mehrerer Bauteile zu komplexeren Blöcken



Datentypen

- Bit, Bit Vector, Integer, ...
- IEEE 1164 std_logic
- Arrays
- physikalische Datentypen
- selbstdefinierte Datentypen



Signale

- digitale Signale innerhalb des FPGAs
- I/O Signale an die FPGA Pins
- Verbindungen zwischen einzelnen Komponenten
- auf Bit Ebene, können aber zu Bus zusammengefasst werden
- Signal als Integer im Bereich 0-100 deklariert wird z.B. auf 7 Bit breiten Bus abgebildet

process Block

- hat in der Regel eine *sensitivity* Liste mit Signalnamen
→ diese Signale triggern die Ausführung des Prozesses
- sequenzielle Ausführung der Befehle
- ermöglicht Flusskontrolle wie in bekannten Sprachen
- Signalzuweisungen werden erst beim nächsten Event übernommen

Beispiel - Synchron/Asynchron

```

architecture beh of foo is
begin
  process(clk)
  begin
    if rising_edge(clk) then
      x <= a and b and c;      -- 1
    end if;
  end process;
  y <= a and c;              -- 2
  z <= a xor b;              -- 3
end beh;

```

Beispiel - Signalzuweisung

```

architecture beh of foo is
    signal a, b, c: std_logic;
begin
    process(clk)
    begin
        if rising_edge(clk) then
            a <= '1';
            b <= a;
            c <= b;
        end if;
    end process;
end beh;

```

IP Cores

IP Cores

- fertige Komponenten
- werden in Form von Netzlisten oder Source Code vertrieben
- opencores.org - Sammlung von Open Source HDL Projekten
- FPGA Herstellerspezifische IP Cores
- Third Party Vendor IP Cores
- können direkt ins eigene Design mit aufgenommen werden
- <buzzword>code reusability</buzzword> \o/



opencores.org

- Sammlung von Open Source HDL Projekten
- VHDL, Verilog, SystemC
- meist GPL oder BSD License
- Kategorien: CPUs, Crypto Cores, Signalverarbeitung, Communication Controller, Video Controller, ...



Soft Processors

- in HDL geschriebene CPUs
- können in FPGA eingebettet werden
- Beispiele
 - MicroBlaze (Xilinx, 32 Bit RISC (DLX), Linux unterstützt)
 - PicoBlaze (Xilinx, 8 Bit RISC)
 - Nios II (Altera, 32 Bit RISC, Linux unterstützt)
 - OpenSPARC (Sun Microsystems, 64 Bit SPARC, GPL)
 - OpenRISC (opencores.org, 32 Bit RISC, GPL)
 - JOP (opencores.org, native Java CPU, GPL)
 - sich andere auf opencores.org

praktische Umsetzung

Testbench

- Test- und Simulationsumgebung
- auch in z.B. VHDL geschrieben
- ermöglicht eben Test und Simulation von Entities
- Device Under Test (DUT) wird in Testbench eingebettet
- Input von DUT wird an Output von Testbench angeschlossen und umgekehrt
- Testbench speist Signale in DUT ein und liest Ausgabewerte

Simulation

- Design in z.B. Testbench laufen lassen
- oder spezielle Simulationsprogramme verwenden
→ on-the-fly Signalzuweisung
- Untersuchung von Timing und Funktionalität
- eventuelle Fehler vorzeitig erkennen

Constraints

- Pin Zuordnung
 - I/O Signale werden an die FPGA Pins zugewiesen
 - einbringen von Pullup/Pulldown Widerständen möglich
 - Pinspezifisch Spannungspegelstandard und Ausgangsstrom definieren
- Constraints
 - Timing Constraints
 - Placement Constraints
 - ...
- Initialwerte für z.B. Speicherelemente angeben

Synthese

- Übersetzung des HDL Codes in Netzliste
- Netzliste enthält textuelle Beschreibung der Komponenten und ihre Verbindungen
- (kommerzielle) IP Cores werden oft als Netzliste ausgeliefert

Implementierung

- Mapping
 - Komponenten werden in native FPGA Elemente übersetzt
- Place & Route
 - Herstellen der Verbindung zwischen den Komponenten

Upload

- zuerst Bitstream Generierung
- Bitstream enthält nötige Konfigurationsinformationen für FPGA
- Upload meist via JTAG (IEEE 1149.1)
- FPGA lädt beim Booten diese Informationen
- Speicherung des Bitstreams meist auf externem PROM

Debugging

- wenn möglich über Simulation
- aber Hardware-Verhalten muss nicht deterministisch sein
- Oszilloskop, Logic Analyzer
 - Debug Informationen z.B. auf I/O Pins ausgeben
 - Debugging innerhalb des FPGAs aber nicht möglich
- Herstellerspezifische Debugging Unterstützung
 - z.B. Xilinx ChipScope Pro
 - packt Logic Analyzer etc. ins Design mit rein
 - Debugging von FPGA Innenleben

Entwicklungs-Software

Werkzeuge

- Entwicklung
 - Vim :)
 - Eclipse Plugin
 - Mentor Graphics HDL Designer
 - herstellerspezifische Entwicklungsumgebungen
- Simulation
 - Mentor Graphics ModelSim
 - GHDL + GTKWave
 - herstellerspezifisch
- Synthese
 - Mentor Graphics LeonardoSpectrum
 - herstellerspezifisch

GHDL

- Open Source VHDL Compiler und Simulator
- kompiliert VHDL Source Code in ausführbare Datei
- liegt auch als .o Datei vor
- direkte Ausführung des VHDL Designs, inkl. Shell I/O
- erzeugt VCD oder GHW Waveform
- kann aber keine Netzlisten erzeugen und nicht synthetisieren

GTKWave

- Open Source Waveform Viewer
- zeigt die von (z.B.) GHDL erzeugten Signale graphisch an
- SystemC unterstützt z.B. auch VCD Output
- für Verilog auch Anzeige auf Code-Ebene möglich

Xilinx ISE

- Entwicklungsumgebung von Xilinx
- ermöglicht kompletten FPGA Entwicklungszyklus
- natürlich volle Unterstützung für Xilinx FPGAs
- Freeware Version "Xilinx ISE WebPACK"
Einschränkung: unterstützt nicht alle FPGAs
- native Linux Unterstützung
- einzige verfügbare Linux Entwicklungsumgebung (Stand 2007)

Xilinx ISE

- bietet
 - Editor
 - Synthese und Implementierung
 - Pinbelegung grafisch editieren
 - Zeitverhalten bestimmen
 - PROM Generierung und Upload
 - Xilinx Core Komponenten Wizard
 - ...
- GUI ist nur Frontend zu einzelnen CLI Programmen
→ können auch direkt von der Shell aus gestartet werden

Xilinx Linux Makefile & Tools

- Nutzung von Xilinx ISE Funktionalität auf Kommandozeile
- ermöglicht gewohnte Entwicklungsumgebung
→ screen, vim, make
- komplette Synthese, Implementierung, PROM Datei
Generierung und Upload möglich

kommerzielle Software

- Mentor Graphics HDL Designer
 - Entwicklungsumgebung auch auf grafischer Ebene
 - Linux und UNIX Unterstützung
- Mentor Graphics ModelSim
 - Simulationsumgebung
 - Linux Unterstützung nur für Verilog bisher (Stand 2008)
- Xilinx ChipScope Pro
 - vorhin erwähnte Debugging Umgebung
 - ebenfalls für Linux verfügbar

Fazit

Fazit

- "general purpose chip for special purpose applications"
- Möglichkeit Hardware nach eigenem Geschmack und Gebrauch selbst entwerfen
- nettes Spielzeug :)

Vorteile FPGA

- echte Parallelität
- Änderungen möglich ohne Hardware anfragen zu müssen
- geringerer Kostenfaktor gegenüber ASICs (bei geringen Stückzahlen)
- einfacherer und schnellerer ASIC Entwicklungszyklus durch Prototyping mit FPGA
- System kann direkt auf Problem angepasst werden

Nachteile FPGA

- manchmal ist Software Lösung sinnvoller
- lädt Konfiguration bei jedem Start von zusätzlich benötigtem EEPROM oder Flash
- nur relativ geringe Taktraten möglich
- bei Massenproduktion besser ASIC verwenden

Links

Links und Literatur

Open Source IP Cores

- <http://www.opencores.org>

VHDL Tutorial (Peter J. Ashenden)

- http://www.tutground.net/Files/VHDL_TUTORIAL.pdf

Peter J. Ashenden, The Designer's Guide to VHDL

Links - Entwicklung

GHDL

- <http://ghdl.free.fr/>

GTKWave

- <http://home.nc.rr.com/gtkwave/>

Xilinx ISE WebPACK

- http://www.xilinx.com/ise/logic_design_prod/webpack.htm

Xilinx Linux Makefiles

- http://avr.auctionant.de/vhdl/xilinx_fpga_makefile_linux.html

Fragen?