FPGA
Design und Verifikation

DE1-SoC Framework

Ziele
- Mit dem DE1-SoC Framework werden anhand einem einfachen Beispiel verschiedene Grundfunktionen, wie sie bei den meisten Designs vorkommen, demonstriert.

Aktivitäten
- Kopieren der Starthilfe in ein eigenes Projektverzeichnis
- Analyse von vorgegebenem VHDL-Code mit einem **Editor**
- Funktionale Verifikation mit **QuestaSim**
- Implementieren auf ein FPGA mit **Quartus**
- Funktionstest auf dem **FPGA Entwicklungsboard**
- Lernkontrolle mit **Kontrollfragen**

Zeitbedarf
3 Stunden

Notwendiges Material
- VHDL Entwicklungsumgebung
- FPGA Entwicklungsboard

Inhaltsverzeichnis

1 Allgemeines 3
1.1 Filestruktur 3
1.2 Tools 4
1.3 Entwicklungsboard 5
2 Code Analyse 6
2.1 Struktur des Demodesigns 6
2.2 Analyse des Designs 6
2.3 Analyse der Testbench 8
3 Verifikation 9
3.1 Setup 9
3.2 Simulation 9
3.3 Analyse im Wave-Window 10
4 Implementation 11
4.1 Setup 11
4.2 Quartus 11
4.3 Überprüfung ihrer Ressourcenabschätzung 12
5 Test 13
5.1 USB-Blaster generell 13
5.2 Quartus Programmer 13
5.3 Funktionstest 14

1 Allgemeines

1.1 Filestruktur

Für diese Übung brauchen Sie Tools (VHDL Editor, Simulator und Synthesetool) und Daten (Ausgangslage gem. Abbildung 1).

Unter Linux können Sie sich zuerst eine spezielle Umgebung einrichten mit

Linux-Befehl

```
msh digital
```

und anschiessen die Daten mit der Ausgangslage zu sich in Home-Verzeichnis kopieren

```
start_fpga_framework
```

Unter Windows müssen Sie die Tools lokal installiert haben und die Ausgangslage in ihren Arbeitsbereich kopieren.

Verzeichnisstruktur für unsere Projekte

de1_soc_framework	**framework**	**Top-Directory**
2_vhdl	**2_vhdl**	**VHDL Sourcefiles**
ip	ip	Standardmodule
3_questasim	**3_questasim**	**Simulationsumgebung**
scripts	scripts	Simulationsskripts
work	work	Kompilierte Designdaten
4_quartus	**4_quartus**	**Implementationsumgebung**
de1_soc_top	de1_soc_top	Projektverzeichnis
output_files	output_files	Generierte Files, Resultate
scripts	scripts	Syntheseskripts
doc	**doc**	**Projektdokumentation**

Abbildung 1: Verzeichnisstruktur

1.2 Tools

Hilfsmittel für das Strukturierte Design ist ein *VHDL Editor*. Mit *QuestaSim* können wir die Funktion verifizieren, mit *Quartus* machen wir die Synthese und erstellen eine Konfiguration für den FPGA.

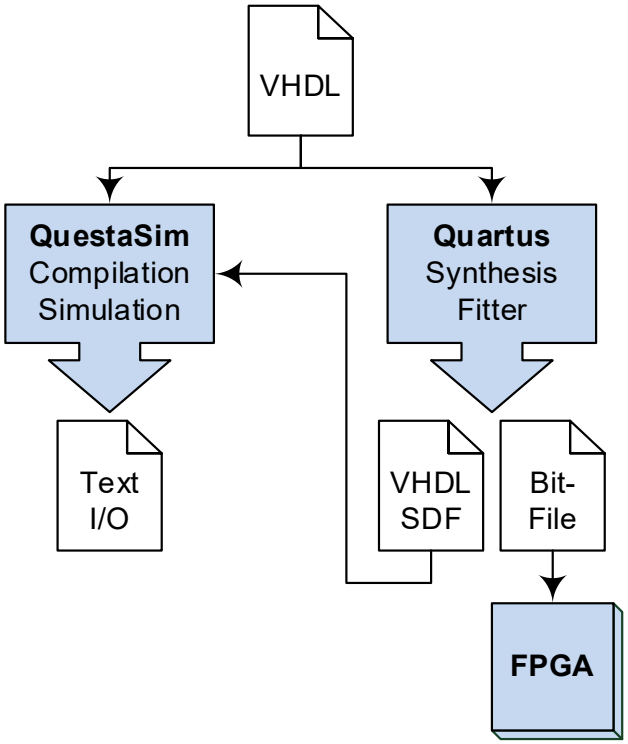


Abbildung 2: Designflow mit den Tools von Mentor und Altera

VHDL Editor

Es gibt verschiedene Editoren mit VHDL Unterstützung:
- Visual Studio Code
- Notepad++

Unter Linux startet man VS Code aus der Konsole mit

```
code &
```

QuestaSim

Unter Linux startet man das Tool aus der Konsole mit

```
vsim &
```

Quartus

Unter Linux startet man das Tool aus der Konsole mit

```
quartus &
```

1.3 Entwicklungsboard

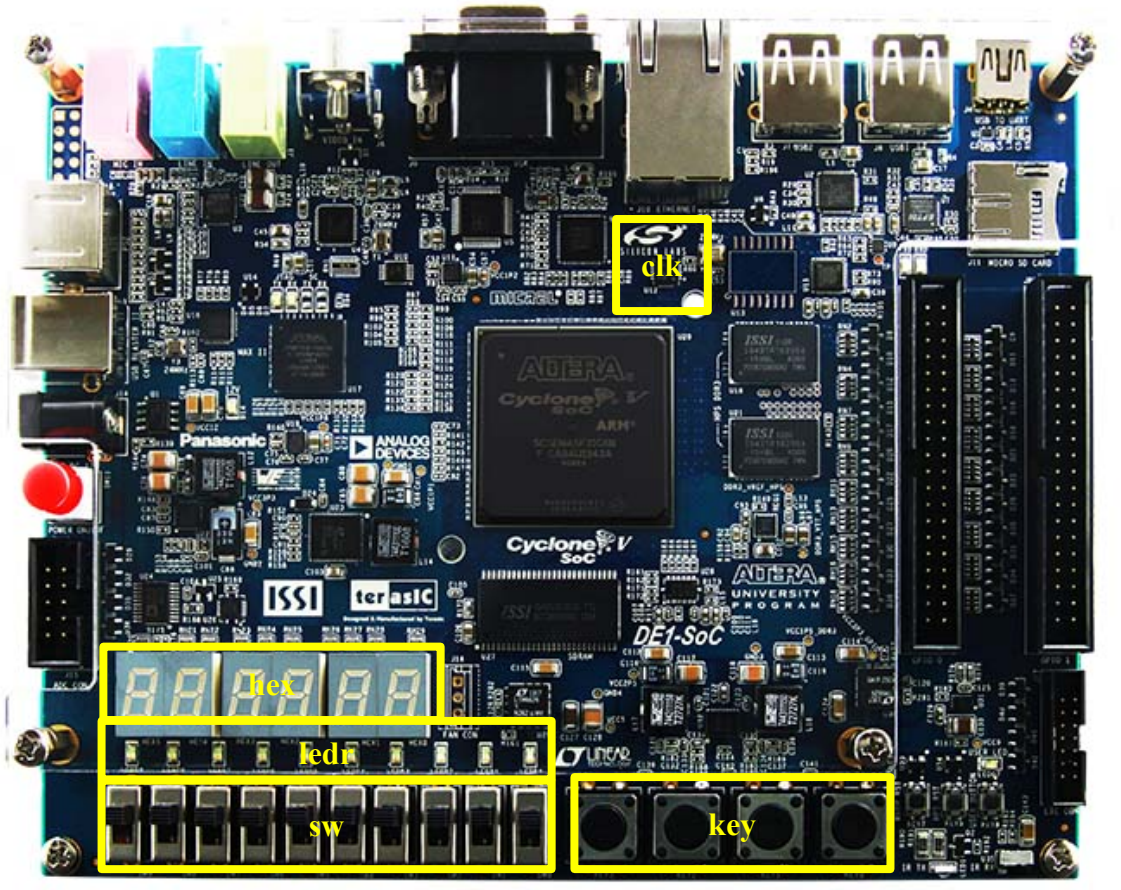


Abbildung 3: DE1-SOC Development Board (Cyclone V). Markiert sind neben der Clock-Generierung die Siebensegmentanzeigen (hex), die LEDs (ledr), die Switches (sw) und die Drucktasten (key).

2 Code Analyse

2.1 Struktur des Demodesigns

Den VHDL Code für das Demodesign finden Sie im Ordner 2_vhdl.

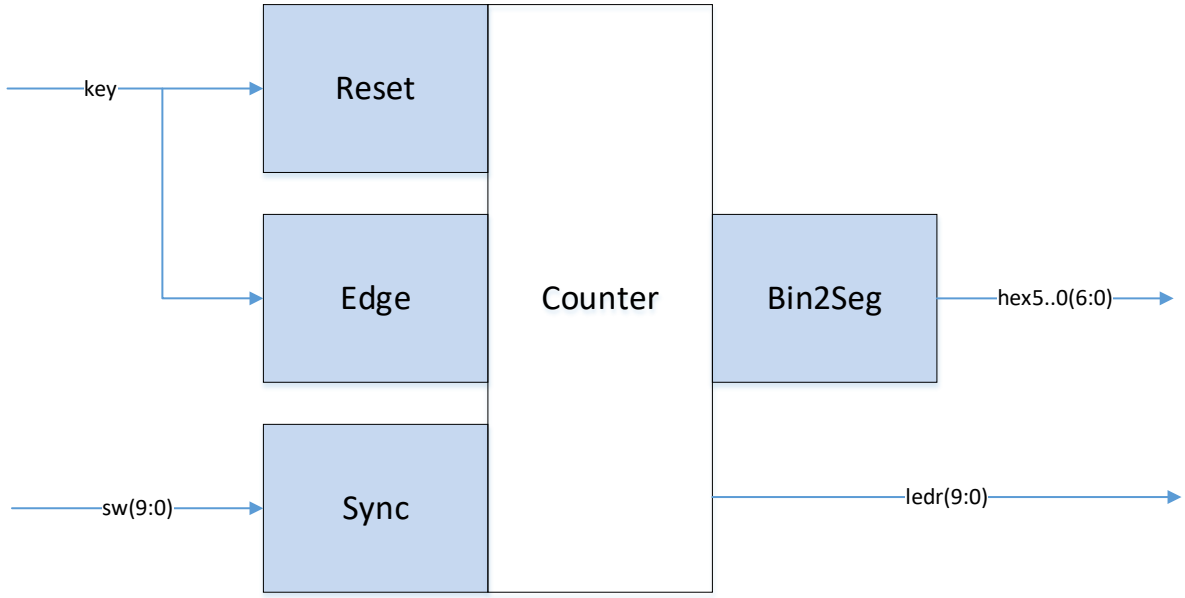


Abbildung 4: Demodesign mit den einfachen Schnittstellen

Eingänge

Reset	Synchronisation des Resets nach der Methode "Asynchronous Assertion – Synchronous Deassertion"
Edge	Flankenerkennung der Drucktasten – Pulserzeugung bei der fallenden Flanke (Tasten sind low-activ).
Sync	Synchronisation der Switches.

Ausgänge

Bin2Seg	Ansteuerung der Siebensegmentanzeigen. Die Ausgabe auf die 10 LEDs geschieht direkt

System

Counter	Eigentliche Funktion: Bei diesem Demodesign ist das ein Zähler mit verschiedenen Funktionen.

2.2 Analyse des Designs

Analysieren Sie alle Blöcke und versuchen Sie, die folgenden Fragen zu beantworten.

- Verstehen Sie die Funktion des jeweiligen Blocks?
- Wie viele Flipflops werden synthetisiert?

Block	Funktion	Flipflops
Reset rsync.vhd	jenach generic modus selection voll Synchroner oder asyncron reset und syncroner Clk.	2 Flipflops
Edge isync.vhd	input daten synchronisierung und Flanken detektion jenach configuration der generics. In diesem	2*4 Flipflops
Sync isync.vhd		2*10 Flipflops
Bin2Seg bin2seg7.vhd		
counter counter.vhd		
Total		292

Analysieren Sie den **counter** detaillierter:
- Zeichnen Sie das Zustandsdiagramm.
- Handelt es sich um eine Moore- oder Mealy FSM?

Mealy Automat

Für Block Bin2Seg:

regs: 32 Flipflops
count: (2**25-1) --> 24Bits
clk_1Hz: 1Flipflop
oseg: 6*7=42 Flipflops

-->Total = 99 FlipFlops

Für Block counter:

cnt_en: 1
cnt_prescale : 25
state_cs: 7
int_reg: 3*32=96
newvalue: 1
write_en: 1
data_reg: 32

-->Total = 163 FlipFlops

- Machen Sie eine Tabelle und ordnen Sie die Ein- und Ausgänge des **counter** den Peripherien des DE1-SoC Boards zu (z.B. key(0) – Reset).

system	**de1_soc_top**
clk	clk_50
rst_n	key(0)
start/stop Button	key(1)
load button	key(2)
set max value button	key(3)
format Switch	sw(9)
cnt speed switch	sw(8)
cnt modus switch	sw(7)
cnt direction switch	sw(6)
max value value (switch 0 to 5)	sw(5:0)
Siebensegmentanzeige 5 und 6	hex5:hex4
Siebensegmentanzeige 3 und 4	hex3:hex2
Siebensegment anzeige 1 und 2	hex1:hex0
Status leds des zähler	ledr

Abbildung 5: Mapping der logischen Namen zu den board-spezifischen Namen

2.3 Analyse der Testbench

Wir verifizieren mittels Testbench unseren **counter**, nicht das gesamte Framework!

Öffnen Sie nun den Block **counter_tb**, hier handelt es sich um eine strukturale Beschreibung, welche das DUV und den Verify Block enthält.
Öffnen Sie den Verify Block **counter_verify**. Analysieren Sie den VHDL Code und beantworten Sie folgende Fragen:

- Was fällt auf bei den Ports?

  alle Inputs des Counter sind im verify block Outputs. und umgekehrt

- Verstehen Sie, wie der Clock und Reset stimuliert werden?
  rst nur die ersten 3 clk zyklen danach clk Generierung durch loop function
- Erkennen Sie, wie der **counter** auf die Stimulis reagieren soll?

3 Verifikation

3.1 Setup

Im Verzeichnis **3_questasim/scripts** finden Sie zwei vorbereitete Files:

- sim_setup.tcl Enthält nützliche, Projekt-spezifische Befehle
- sim.do Enthält Einstellungen für die Simulation

3.2 Simulation

Wechsel Sie in das Verzeichnis **3_questasim** und starten dort den Simulator. Führen Sie in der Konsole (Transcript) folgenden Befehl aus:

```
do  scripts/sim_setup.tcl
```

Nun stehen Ihnen verschiedene Befehle zur Auswahl:

- `com` Kompiliert alle Sourcefiles
- `ld` Kompiliert alle Sourcefiles und lädt die Simulation
- `sim` Führt die Simulation aus

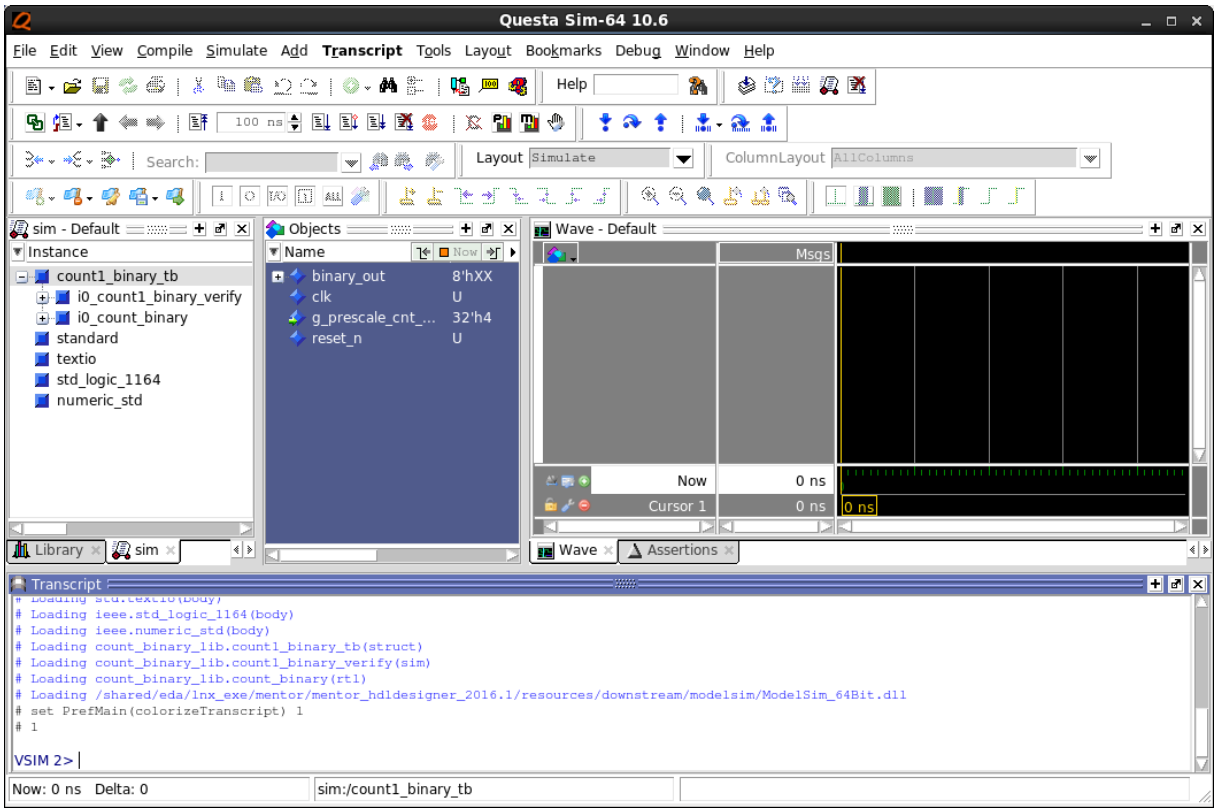


Abbildung 6: Im weissen Fenster sehen Sie die Struktur, im blauen die Signale und im schwarzen den Signalverlauf.

Führen Sie die Befehle `com`, `ld` und `sim` aus und verifizieren Sie die Funktion im Wave- und Transcript-Window.

3.3 Analyse im Wave-Window

Zu welcher Zeit

- ist das System initialisiert und welches ist der Max-Wert des Zählers?

  @70ns wechsel zu S_idle & @70ns max value = 0

- und auf welchen Wert wird der Max-Wert verändert?

  @71ns 0x3F

- und mit welchem Wert wird der Zähler geladen?

  @110ns 9

- springt der Zähler vom Maximal-Wert zurück auf null?

4 Implementation

4.1 Setup

Im Verzeichnis **4_quartus/de1_soc_top** finden Sie vorbereitete Files:
- de1_soc_top.qpf Quartus Project File
- de1_soc_top.qsf Quartus Setting File
- de1_soc_top.qip Quartus IP File (verweist auf alle Source Files)
- de1_soc_top.sdc Synopsys Design Constraints (enthält alle Timing Vorgaben)

und im Unterverzeichnis **scripts**
- set_location_assignments.tcl Definiert alle Pin-Zuordnungen

Info: Informationen zum DE1-SOC Board finden Sie im User Manual.

4.2 Quartus

Wechsel Sie in das Verzeichnis **4_quartus/de1_soc_top** und starten dort Quartus. Öffnen Sie in Quartus das Projektfile **de1_soc_top.qpf**

Starten Sie nun den Compile-Vorgang:

Quartus::`Processing→Start Compilation`

oder einfach den Play-Button drücken. Dabei werden folgende Schritte ausgeführt:

- Analysis & Synthesis HDL Syntax Check und Synthese in eine RTL-Netzliste
- Fitter Abbilden der Meta-Logik auf reale FPGA Elemente
- Assembler Erstellen der Konfigurationsdatei
- Timing Analyzer Statische Timing Analyse des Designs

Überprüfen Sie die Statusmeldungen auf allfällige Errors und Warnungen (gewisse Warnungen können akzeptiert werden – welche das sind weiss man mit ein bisschen Erfahrung☺).

Schauen Sie sich das Resultat nach Schritt 1 **Analysis & Synthesis** mit dem Netlist Viewer an.

Quartus::`Tools→Netlist Viewers→RTL Viewer`

Öffnen Sie auch das Resultat nach Schritt 2 **Fitter**.

Quartus::`Tools→Netlist Viewers→Technology Map Viewer (Post-Fitting)`

Frage: Wie unterscheiden sich die beiden Netzlisten?

4.3 Überprüfung ihrer Ressourcenabschätzung

Öffnen Sie den **Compilation Report** und überprüfen Sie Ihre Schätzungen bezüglich Flipflops und LUTs.

Quartus::`Processing→Compilation Report`

Block	**Flipflops**	**Abweichung gegenüber der Schätzung mit Begründung.**
rsync_1		
isync_1		
isync_2		
system_1		
bin2seg7_1		
Total		

5 Test

5.1 USB-Blaster generell

Altera FPGAs werden entweder über ihre JTAG Schnittstelle oder aus einem seriellen EPROM programmiert.

Um die FPGAs bequem vom PC aus zu programmieren hat Altera ein "USB-Blaster" Modul geschaffen.

Weil das Modul bei dedizierten FPGAs nur einmal zum Programmieren benötigt wird, ist es aus Effizienz- und Kosten-Gründen ein separates Modul.

Im unserem Falle ist jedoch das Modul komplett auf dem Board integriert. Dies vereinfacht die Handhabung und ist kein Nachteil, da das Evaluation-Board in dieser Art sowieso in erster Linie für die Schulung und Entwicklung verwendet wird.

Der Anschluss des USB-Blasters ist auf dem Evaluation-Board beschriftet.

5.2 Quartus Programmer

Verbinden Sie Ihren PC und den Blaster-Eingang ihres DE1-SOC-Developmentboards mit einem USB-Kabel und öffen Sie den *Programmer*.

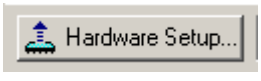
Quartus::`Tools→Programmer`

oder einfach den Programmer-Button drücken.

Mit dem Programmer können Sie das SOF-File im JTAG-Mode auf das FPGA herunterladen und anschliessend die Funktion ihrer Stoppuhr verifizieren.

Hardware Setup

Wenn der Treiber für den USB-Blaster installiert ist, muss noch die Quartus Software entsprechend konfiguriert werden.



Dazu muss durch Drücken des "Hardware Setup…" Knopfes das entsprechende Fenster geöffnet werden.

Nun können Sie mit "Add Hardware" zuerst die USB-Blaster Schnittstelle zur Auswahl hinzufügen. Anschliessend muss unter der Rubrik "Currently selected hardware" diese Schnittstelle auch als Programmier-Verbindung ausgewählt werden.

Auto Detect

Nach dem Schliessen des "Hardware Setup" Fensters muss nun auch noch im Programmier Fenster der "Mode" auf JTAG gesetzt werden und danach ein Auto-Detektierung durchgeführt werden. Es sollte eine Auswahl für den FPGA kommen (wählen Sie den 5CSEMA5) – es sollte danach eine JTAG-Chain mit HPS und FPGA erscheinen (wie im Bild unten). Nun müssen Sie noch das SOF-File dem FPGA zuordnen und die "Check-Box" in der Kolonne für "Program/Configure" auswählen.

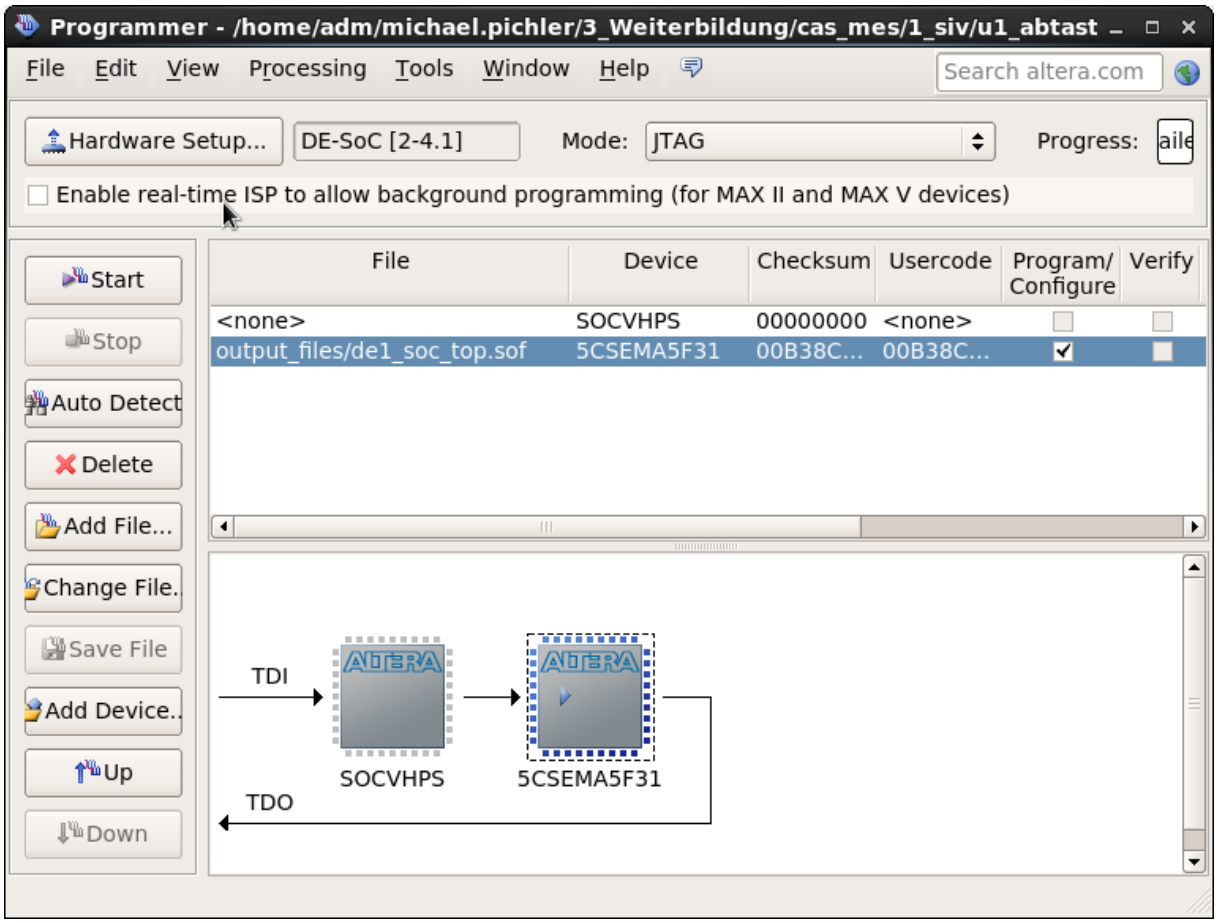
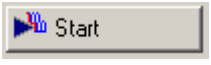


Abbildung 7: Quartus Programmer mit HPS und FPGA in der JTAG-Chain

FPGA Programmierung

Ausser dem "Check-Box" Eintrag sind alle anderen Konfigurationen nur einmal Notwendig.

Zum wiederholten Programmieren gehören die folgenden Schritte:

- Verbinden und Einschalten des FPGA Boards
- Auswählen des Files durch Klick auf die Check-Box
- Drücken des Start-Knopfes 

5.3 Funktionstest

Überprüfen Sie nun auf dem Board, ob Ihr System richtig funktioniert. Alles OK?