Projekterstellung und -verwaltung in der Entwicklungsumgebung Vivado

Handout zum 0. Praktikumstermin

Hardwarepraktikum SoSe 21

Diese Anleitung soll die Grundlagen zur Arbeit mit der FPGA-Entwicklungsumgebung Vivado der Firma Xilinx sowie des Simulationswerkzeugs QuestaSim von Mentor vermitteln. Sie umfasst das Anlegen und Verwalten von Projekten, Erstellung und Test einfacher HDL-Beschreibungen, die Synthese in eine FPGA-geeignete Darstellungsform, die Simulation des eigenen Designs sowie das Übertragen dieser Konfiguration in den FPGA.

Hinweis: im regulären Praktikumsbetrieb wäre die nachfolgende Beispielaufgabe (so wie auch einige Aufgaben später im Praktikum) auf dem FPGA getestet worden. Dies ist aufgrund der aktuellen Situation leider nicht möglich. Die dafür relevanten Abschnitte 7 und 9 können Sie bei Interesse dennoch nachvollziehen, auch wenn sie mangels Hardware wenig zielführend sind.

1 Arbeitsumgebung

Folgen Sie der Änleitung zur Nutzung der Toolchain", die Sie auf ISIS finden. Verbinden Sie sich mit den Uni-Servern oder laden Sie die nötigen Tools herunter. In diesem Handout wird davon ausgegangen, dass Sie auf den Uni-Servern arbeiten.

Im Praktikum werden die Werkzeuge QuestaSim von Mentor Graphics zur Simulation bzw. Xilinx Vivado Synthesis zur Logiksynthese verwendet.

Um die Werkzeuge nutzen zu können, müssen ein paar Umgebungsvariablen[1] gesetzt werden. Dies erfolgt per Script. Öffnen Sie ein Terminal und geben Sie folgenden Befehl ein:

```
source /afs/tu-berlin.de/units/Fak_IV/aes/scripts/hwptienv_basys
```

Danach sollten alle Werkzeuge die im Rahmen des Praktikums genutzt werden aus diesem Terminal aufrufbar sein.
Sie können diese Zeile auch in ihre `.bashrc` eintragen, beachten Sie jedoch das dies Einfluss auf andere Werkzeuge wie zum Beispiel den gcc hat! Fügen Sie dafür die oben genannte Zeile mit *nano* an das Ende der `.bashrc`, also mit `nano ~/.bashrc`.

[1] Pfade der Werkzeuge (PATH) sowie License-Server (LM_LICENSE_FILE)

2 Anlegen eines neuen Projekts

Legen Sie ein Verzeichnis für Vivado-Projekte in Ihrem Homeverzeichnis an und vergeben Sie einen aussagekräftigen Namen ohne Leerzeichen[2] (beispielsweise `Vivado_WORK`). Legen Sie darin einen zweiten Ordner an, der das heutige Projekt enthalten wird. Vergeben Sie einen Namen, z.B. `Tutorial`.

Starten Sie die Entwicklungsumgebung Vivado durch Eingabe von `vivado&` in einem Terminal. Klicken Sie im Menü *Quickstart* auf *Create Project* und geben Sie den Speicherort des Projektverzeichnisses an. Achten Sie unbedingt darauf, dass weder Projektname noch Projektpfad Leerzeichen enthalten, so wie es in Abbildung **??** dargestellt ist. Lassen Sie Vivado dazu auch ein Unterverzeichnis erstellen.

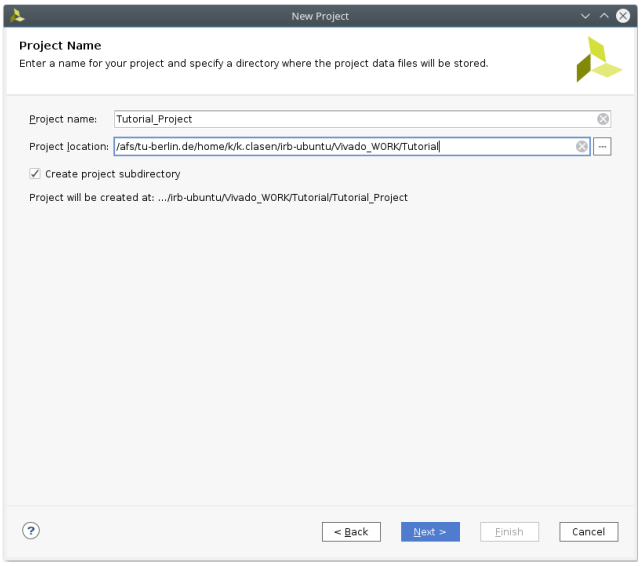

Abbildung 1: Projekterstellung - Name und Pfad

Als nächstes, wählen Sie als *Project Type* den Typ *RTL Project* und drücken Sie auf *Next*. Wählen sie hier unten als *Target language VHDL* statt *Verilog* aus und klicken Sie auf *Next*. Sie könnten hier neue oder bereits existierende Dateien zum Projekt hinzufügen. Bestätigen Sie das *Constraints-Fenster* ohne Änderungen.

Im nächsten Fenster, siehe Abbildung **??**, wird der FPGA-Typ des Testboards ausgewählt. Diese Optionen sind auch später noch veränderbar.

Geben Sie folgende Vorgaben ein:

Family:	Artix-7
Package:	cpg236
Speed Grade:	-1

Wählen Sie dann im unteren Teil des Fensters das Board mit dem Bezeichner `xc7a35tcpg236-1` aus.

[2]Leerzeichen oder sonstige nicht-alphanumerische Zeichen außer '_' im Pfad führen zur Arbeitsverweigerung von sämtlichen Xilinx-Tools. Zusätzlich müssen Projektnamen mit einem Buchstaben anfangen.

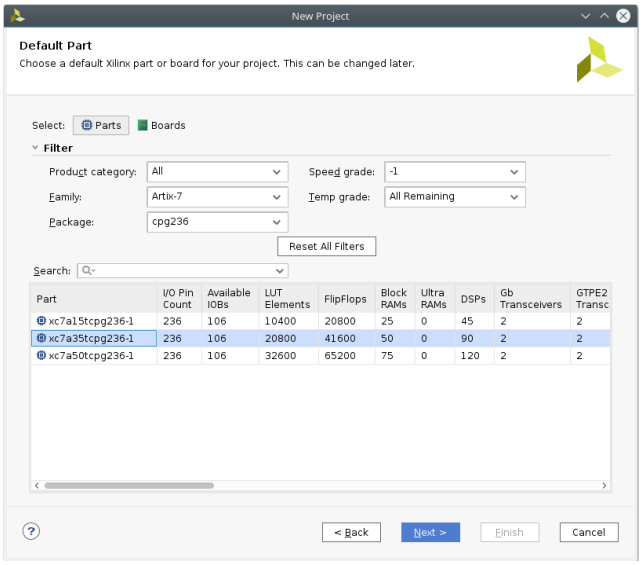

Abbildung 2: Projekterstellung - FPGA Auswahl

Abschließend erhalten Sie eine Zusammenfassung der gewählten Einstellungen, bestätigen Sie diese durch einen Klick auf *Finish*. Sie sehen nun ein Fenster, ähnlich dem aus Abbildung **??**, mit einer Übersicht über den Projektstatus, ein Fenster für die zum aktuellen Projekt gehörenden Quellen (*Sources*), in der Seitenleiste die auf das Projekt und seine Quellen anwendbaren Aktionen (*Flow Navigator*) und einen Bereich für die Textausgaben der Werkzeuge (*TCL Console*, *Messages*, etc).

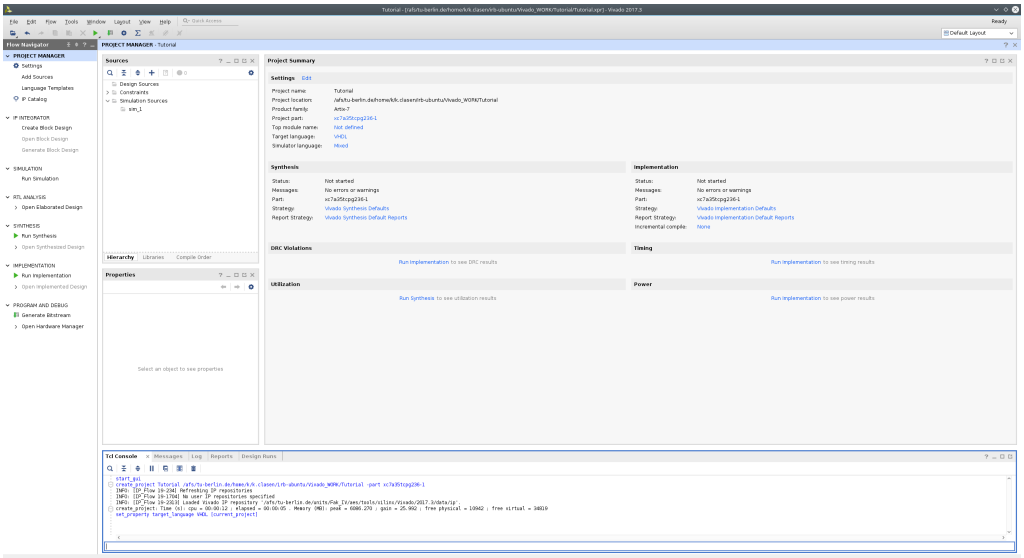

Abbildung 3: Vivado nach der Projekterstellung

3 Anlegen und Einbinden einer VHDL-Datei

Vivado kann Projektquellen an zwei Orten speichern: lokal im Projektverzeichnis oder in einem getrennt verwalteten Ordner. Da wir im Rahmen des Praktikums zusätzlich mit QuestaSim arbeiten möchten ist es ratsam die Quelldateien in einem separaten Ordner anzulegen. Falls Sie später nur mit den in Vivado integrierten Xilinx-Tools arbeiten sollten, können Sie die Option *Local to Project* wählen und Vivado die Quellen verwalten lassen. Wenn Sie jedoch weitere Tools wie z.B. QuestaSim zur Simulation verwenden möchten, empfiehlt es sich, die Quellen in einen eigenen Ordner außerhalb des Projektverzeichnisses zu speichern, um einfacher darauf zugreifen zu können und Konflikte zu vermeiden.

Für das Hardwarepraktikum wird daher für jedes Projekt folgende Verzeichnisstruktur empfohlen:

- ▽ **Vivado_WORK**
  - ▽ **Tutorial**
    - ▷ **Tutorial_Project** - *Xilinx Projektverzeichnis*
    - ▷ **src** - *Quellenverzeichnis*
    - ▷ **work** - *QuestaSim Projektverzeichnis*

Über den *Add Sources* Button in der Seitenleiste können Sie nun neue oder bereits existierende Dateien zu Ihrem Projekt hinzufügen. Für synthetisierbare VHDL-Dateien wählen Sie die Option *Add or Create Design Sources* im ersten Dialogfenster. Für Testbenches, die nicht synthetisiert werden, ist hier später *Add or Create Simulation Sources* zu wählen.

Eine leere Datei im Quellenverzeichnis anzulegen, die entity von Hand oder mit Unterstützung eines geeigneten Editors zu schreiben und die Quelle mit *Add Sources* dem Projekt hinzuzufügen dürfte langfristig der schnellste und unkomplizierteste Weg der Projektverwaltung sein.

Legen Sie in Ihrem Quellenordner die leere VHDL-Datei `or_gate.vhd` an. Öffnen Sie die Datei in einem Editor Ihrer Wahl und übernehmen Sie die Entitydeklaration[3] aus Listing **??**.

Hinweis: Kopieren Sie bitte solche Listings nie direkt aus der PDF, da immer wieder Fehler beim Kopieren auftreten. Zum Beispiel werden zusätzliche Leerzeichen um Punkte gesetzt oder Hochkommatas als Akzente interpretiert.

```
library ieee;
use ieee.std_logic_1164.all;

entity or_gate is
    port(   INPUT_1:  in std_logic;
            INPUT_2:  in std_logic;
            OUTPUT : out std_logic );
end entity or_gate;

architecture structure of or_gate is
begin
```

[3] Der architecture-Teil gehört natürlich nicht zur entity-Deklaration. Vivado bindet die VHDL-Datei aber nur mit gültiger architecture-Deklaration als Quelle ins Projekt ein.

```
end architecture structure;
```

Listing 1: Entity `or_gate`

Speichern Sie die Datei und fügen Sie sie anschließend in Vivado per *Add Sources* → *Add or Create Design Sources* → *Add Files...* dem Projekt hinzu.

Wichtig: Achten Sie dabei immer darauf, das Häkchen *copy Sources into projekt* zu entfernen, ansonsten arbeitet Vivado mit einer Kopie und nimmt Änderungen an der Originaldatei nicht mehr wahr.

Wählt man nun unten im Sources-Fenster *Libraries* aus, so taucht die Komponente unterhalb der default-Library `xil_defaultlib` auf, wie Abbildung **??** verdeutlicht.

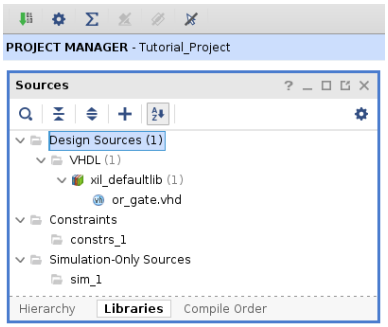

Abbildung 4: Entity und Architecture einer Projektquelle

Ein Doppelklick auf die neue Datei im *Sources*-Fenster öffnet den Texteditor.

4 Vivado Texteditor anpassen oder anderen Editor einbinden

Die VHDL-Dateien können natürlich unabhängig von der Entwicklungsumgebung in Ihrem bevorzugten Editor bearbeitet werden. Doch auch beim Öffnen von Dateien aus Vivado sind Sie nicht an den mitgelieferten Texteditor gebunden. Um Quellen aus der Entwicklungsumgebung heraus mit dem Editor der Wahl öffnen zu können, gehen Sie folgendermaßen vor: Öffnen Sie die Vivado-Voreinstellungen durch einen Klick auf *Tools* → *Settings*. Auf der linken Seite finden Sie ein Feld *Text Editor*, dass der Anpassung des eingebauten Editors dient, siehe Abb. **??**. Hier kann ein anderer Standardeditor angegeben werden. Taucht im DropDown-Menü der von ihnen gewünschte Editor nicht auf, wählen Sie *Custom Editor...* und geben den Befehl zur Ausführung Ihres Editors entsprechend den dargestellten Erläuterungen ein.

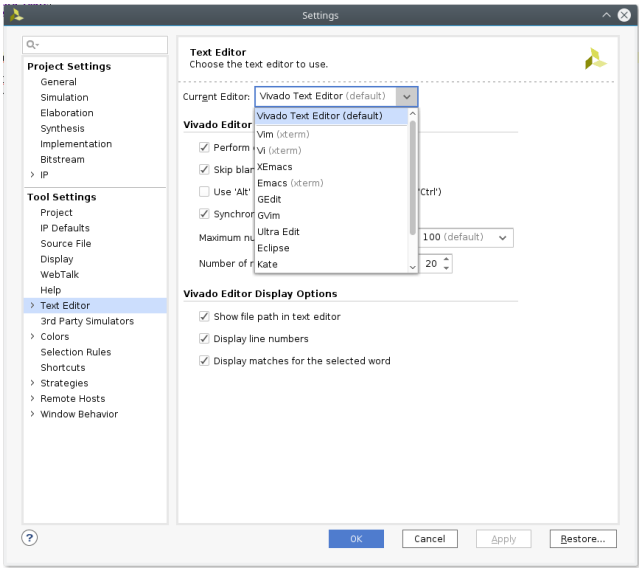

Abbildung 5: Festlegen des Standardtexteditors

5 Syntaxcheck und RTL Übersicht

Vervollständigen Sie die architecture von `or_gate.vhd`, z.B. folgendermaßen[4] wie in Listing **??** vorgeschlagen.

```
architecture structure of or_gate is
begin
    OUTPUT <= INPUT_1 or INPUT_2;
end architecture structure;
```

Listing 2: Mögliche architecture für das Oder-Gatter

Elaborate ist der erste Schritt zur Überprüfung Ihres VHDL Codes. Dabei wird Ihre Hardwarebeschreibung auf eine Register Transfer Level (RTL) Struktur abgebildet. Klicken Sie dafür auf der linken Seite im *Flow Navigator* auf *Open Elaborated Design*, welches sich unter der Gruppe *RTL Analysis* befindet. Die dabei verwendeten Bausteine (Gatter, Register, Multiplexer etc.) sind noch unabhängig von der tatsächlich verwendeten Zieltechnologie. In diesem Fall wird ein Schaltungssymbol angezeigt. Ein Doppelklick auf dieses Symbol öffnet die nächst tiefere Ebene. Wie in Abbildung **??** dargestellt wird ein Oder-Gatter angezeigt, obwohl das FPGA nicht direkt über ein derartiges Gatter verfügt.

Beim ersten Betätigen von *Open Elaborated Design* kann Vivado Sie nach dem Topmodul fragen. Innerhalb eines Projekts stellt immer eine Komponente das Topmodul dar. Diese Komponente und alle in der Hierarchie darunter liegenden (durch Instantiierung im Topmodul) werden im FPGA realisiert. Andere Komponenten, die nicht in der Hierarchie unterhalb des Topmoduls liegen, werden ignoriert. Geben Sie hier, falls nötig, `or_gate` ein. Falls Sie dies später noch einmal ändern möchten, kann eine Designkomponente im Sources-Fenster in der Hierarchy-Ansicht durch auswählen der Komponente und anschließendem (!) Rechtsklick zum Topmodul erklärt werden (Menüpunkt ”Set as Top”).

[4] Der Name der architecture ist im Prinzip beliebig und für die Funktionsbeschreibung des Oder-Gatters existieren zahlreiche Möglichkeiten

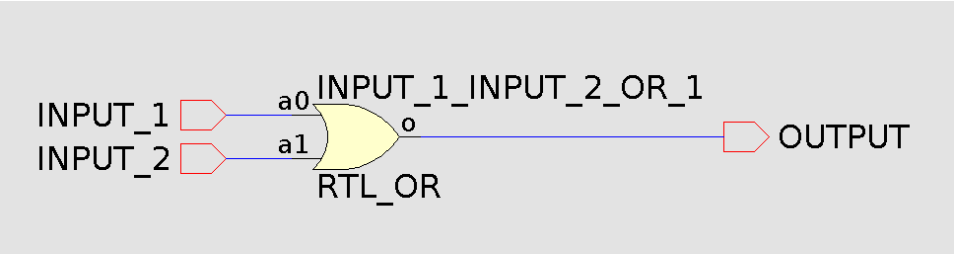

Abbildung 6: RTL Schematic für das Oder-Gatter

Sie sollten sich angewöhnen, gelegentlich einen Blick auf das RTL Schematic zu werfen. Fehler in Ihrem Design fallen hier oft frühzeitig auf. Vergessen Sie beispielsweise, Ihre Modulausgänge zu beschalten, bleibt das Schematic vollständig leer.

6 Simulation mit Testbench in QuestaSim

Der Test von Komponenten wird mit einer Testbench durchgeführt. Dies ist eine spezielle VHDL Entity, die keine Eingänge oder Ausgänge enthält und nicht synthesefähig ist. In dieser wird die zu testende Komponente instanziiert und Ihre Eingangssignale mithilfe der Simulationskonstrukte `wait` und `after` generiert.

Erstellen Sie eine Datei namens `or_gate_tw.vhd` in Ihrem Quellenverzeichnis. Fügen Sie den Code aus Listing **??** ein. Bitte beachten Sie, dass Copy&Paste zu Syntaxfehlern führen kann, weshalb Abtippen zu bevorzugen ist.

```
library ieee;
use ieee.std_logic_1164.all;

entity or_gate_tw is

end or_gate_tw;

architecture testbench of or_gate_tw is

    signal A : std_logic := '0';
    signal B : std_logic := '0';
    signal C : std_logic;

    component or_gate
        port ( INPUT_1 : in std_logic;
        INPUT_2 : in std_logic;
        OUTPUT : out std_logic );
    end component;

    begin
        A <= '1' after 20 ns, '0' after 60 ns;

        B <= '1' after 40 ns, '0' after 80 ns;

        uut : or_gate
            port map ( INPUT_1 => A,
            INPUT_2 => B,
            OUTPUT => C );


end testbench;
```

Listing 3: Testbench für das Oder-Gatter

Wechseln Sie in einem Terminal in das Verzeichnis ~/Vivado_WORK/Tutorial/ und starten Sie QuestaSim mit dem Befehl `vsim&`.

QuestaSim benötigt ein Arbeitsverzeichnis namens `work`. Normalerweise existiert diese Bibliothek nicht und wird als *unavailable* gekennzeichnet. Um dies zu korrigieren tragen Sie im *Transcript*-Fenster (die „Konsole“ von QuestaSim im unteren Bildschirmteil) `vlib work` gefolgt von `vmap work work` ein. Ignorieren Sie die Warning, falls `work` bereits existiert. Das bedeutet Sie haben bereits einen Ordner mit dem Namen `work` angelegt gehabt. Führen Sie aber trotzdem den Befehl `vmap work work` aus. Das Arbeitsverzeichnis bleibt zukünftig erhalten.

Jetzt müssen alle gewünschten Quelldateien nach `work` kompiliert werden. Unter der Voraussetzung, dass Ihre Quelldateien im Verzeichnis `src` im Projektverzeichnis liegen, gelingt dies durch die Eingabe von `vcom -work ./work/ ./src/*.vhd`
Falls Sie die grafische Oberfläche bevorzugen, öffnen Sie im Menü *Compile* → *Compile...*, wählen Sie alle Quelldateien aus und klicken Sie auf *Compile*. Schließen Sie das Fenster mit *Done*, wenn die Kompilation abgeschlossen ist.

Im *Workspace*-Fenster von QuestaSim finden Sie den Bibliothekseintrag work und darin die Datei `or_gate_tw`, ähnlich wie in Abbildung **??**.

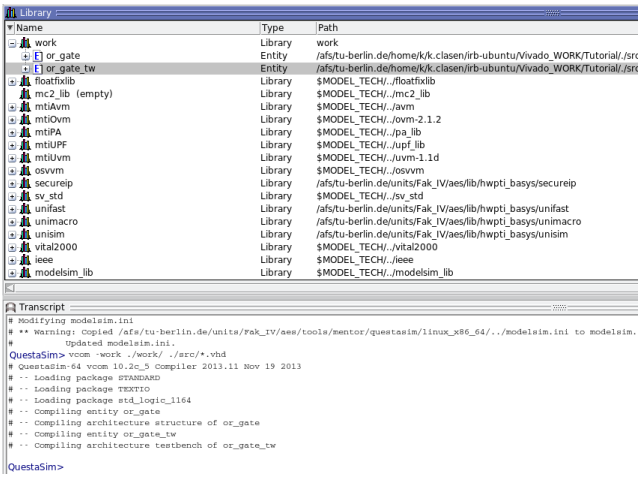

Abbildung 7: QuestaSim Workspace

Klickt man nun doppelt auf `or_gate_tw` öffnet sich unter anderem das *Objects*-Fenster von QuestaSim. Markieren Sie hier alle Einträge und wählen Sie in deren Kontextmenü *Add Wave*, wie in Abbildung **??** gezeigt.

Es öffnet sich das Wave-Fenster mit den hinzugefügten Signalen. Klicken Sie auf *Run* oder geben Sie in das *Transcript*-Fenster run 100 ns ein. Im *Wave*-Fenster sehen Sie die von Ihnen erzeugten Stimuli auf `INPUT_1` und `INPUT_2` sowie das daraus erzeugte Ausgangssignal `OUTPUT`, wie in Abbildung **??** gezeigt.

Beenden Sie QuestaSim wenn Sie sich von der korrekten Funktion der getesteten Komponente überzeugt haben. Dateien, die einmal per `vcom`[5] im QuestaSim -Arbeitsverzeichnis bekannt gemacht wurden, können weiterhin außerhalb oder innerhalb von QuestaSim bearbeitet werden. Im

[5] Die Dateien verbleiben an dem Ort, aus dem heraus Sie durch `vcom` erstmalig kompiliert wurden, hier also im Ordner `src` unterhalb des Projektverzeichnises.

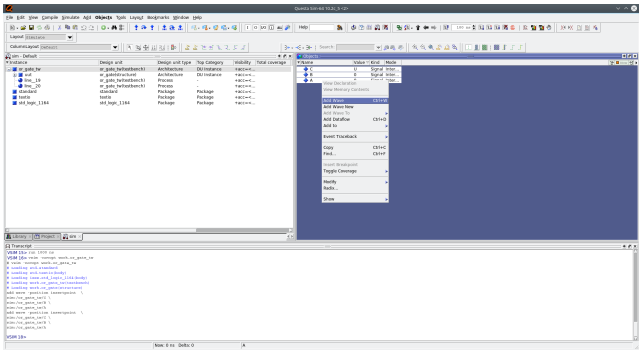

Abbildung 8: Hinzufügen von Signalen zur Waveform

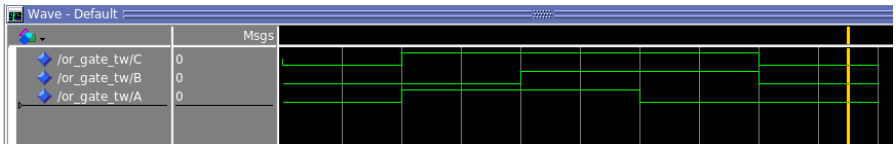

Abbildung 9: Wave-Fenster mit 100ns Simulation

Kontextmenü eines Moduls im *Workspace*-Fenster von QuestaSim können Sie durch *Recompile* eine veränderte Datei neu kompilieren.

7 Abbildung von Signalen auf FPGA-Pins - Anlegen von Location Constraints

Als nächstes ist es notwendig festzulegen, wie die Ports Ihres Designs den Pins des FPGA zugeordnet werden sollen. Dazu verwenden wir wieder das Werkzeug Vivado, öffnen Sie dieses falls Sie es zuvor geschlossen haben. Klicken Sie auf *Add Sources* in der Seitenleiste und wählen Sie diesmal *Add or Create Constraints*. Sie können nun entweder wie vorher die VHDL-Dateien extern eine Datei namens `.xdc` im Ordner `src` anlegen und diese via *Add Files...* hinzufügen, oder Sie legen die Datei direkt im Vivado-Dialog an. Man sollte auch bei anderen Boards wie folgt vorgehen: Als erstes sucht man sich die Master.xdc heraus. Diese befindet sich für das Basys3 auf der ISIS Seite. Nachdem man diese Datei heruntergeladen und ins Quellverzeichnis kopiert hat, kann man sie dem Projekt hinzufügen, wie man es in Abb. **??** sehen kann.

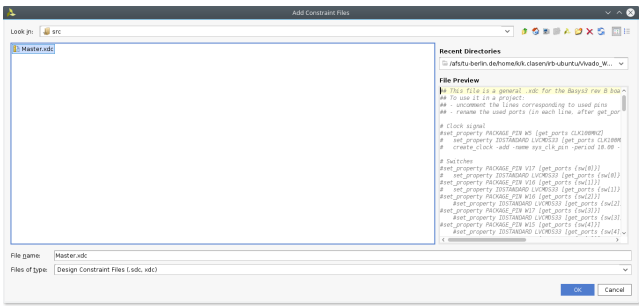

Abbildung 10: Hinzufügen von Location Constraints

Die Datei kann im *Sources* Fenster unter dem *Constraints*-Ordner gefunden werden. Öffnen Sie diese durch einen Doppelklick.

Sie können nun den Aufbau einer *Master.xdc*-Datei sehen. Oben steht eine kurze Beschreibung, wie man die .xdc-Datei korrekt verwendet. Danach sind alle Einstellungen auskommentiert und ganz zum Schluss kommen Einstellungen, die *nicht* verändert werden sollen. Diese beheben einige Warnings und sorgen dafür, dass die am Ende erstellte .bin-Datei nicht unnötig groß ist und sind von uns vorgegeben.

Kommentieren Sie nun alle benötigten Pins ein und ändern Sie den vorgegebenen Namen mit dem Namen des Signals Ihres Topmoduls.

Wir wollen die Wirkung des Oder-Gatters im FPGA visualisieren und testen. Unsere Experimentierplatinen verfügen beispielsweise über Schiebeschalter und LEDs. Zwei der Schalter werden die Eingänge des Gatters speisen, eine LED den Ausgangswert anzeigen. Die Zuordnung der FPGA-Pins zur Peripherie entnehmen Sie der *Master.xdc*-Datei. Wir wollen zum Beispiel den Switch 0 mit unserem Eingangssignal `INPUT_1` verbinden. Dafür ändern wir folgende Zeilen:

```
#set_property PACKAGE_PIN V17 [get_ports {sw[0]}]
#   set_property IOSTANDARD LVCMOS33 [get_ports {sw[0]}]
```

auf

```
set_property PACKAGE_PIN V17 [get_ports {INPUT_1}]
    set_property IOSTANDARD LVCMOS33 [get_ports {INPUT_1}]
```

Die Zeile ist nun einkommentiert und der generische Name `sw[0]` wurde durch den Namen des Signals `INPUT_1` ersetzt. Falls Ihr Signal ein Vektor ist, so muss wie bei dem generischen Namen, rechteckige Klammern genutzt werden, um auf die einzelnen Elemente zugreifen zu können.

Im Wesentlichen können Sie dem Constraint die Informationen entnehmen, dass das Signal `INPUT_1` jetzt den FPGA-Pin V17 treibt (der an den Switch angeschlossen ist) und dass der Signalpegel zwischen 0 und 3,3V (entspricht dem Standard „Low Voltage Complementary Metal Oxide Semiconductor 3,3V“ = LVCMOS33) liegt. Im Rahmen des Hardwarepraktikums ist keine nähere Beschäftigung mit der Funktionsweise der Constraints notwendig. Ergänzen Sie die xdc-Datei um das zweite Eingangssignal, welches ebenfalls mit einem Switch verbunden werden soll und um das Ausgangssignal, welches eine LED treiben soll. Speichern und schließen Sie das Constraintfile.

8 Synthese und Implementation

Nun können Sie Ihr Design auf die Zielhardware abbilden lassen. Dies geschieht in zwei Schritten: Synthese und Implementation. Der erste Schritt, den Sie durch einen Klick auf *Run Synthesis* in der Seitenleiste anstoßen können, übersetzt Ihre Hardware-Beschreibung in eine *Netlist*: eine graphförmige Beschreibung, bestehend aus LUTs, Puffern und Verbindungen (*Nets*). Nach der Synthese fragt Vivado, wie Sie fortfahren möchten: wählen Sie hier *View Reports*. Daraufhin öffnet sich im unteren Bereich ein Fenster namens *Reports*. Doppelklicken Sie auf *Vivado Synthesis Report* (unter *Synth Design*) und lesen Sie sich die Ausgaben durch. Achten Sie insbesondere auf Angaben zu

abgeleiteten Elementen im Abschnitt *RTL Component Statistics* und versichern Sie sich, dass diese Ihren Vorstellungen entsprechen. (Für dieses einfache Gatter, wird in dem Abschnitt nichts zu sehen sein. Achten Sie aber bei den nächsten Aufgaben darauf.) Sehen Sie sich auch die Ressourcen an, die Ihr Projekt benötigt (im Reports-Fenster *Utilization Report*, oder grafisch im *Project Summary* unter *Utilization*).

In einem zweiten Schritt, *Run Implementation* in der Seitenleiste, werden die Elemente der Netlist auf die Komponenten im FPGA abgebildet. Schauen Sie sich auch hier die Ausgaben an und versichern Sie sich, dass das implementierte Design alle Timing Constraints einhält. (Das Timing ist für das Oder Gatter nicht relevant)

9 Erzeugen von Bitfiles für die Konfiguration des FPGA

Als letztes muss das implementierte Design noch auf den FPGA übertragen werden. Wählen Sie unter *Program and Debug* in der Seitenleiste *Generate Bitstream* und bestätigen Sie im sich öffnenden Fenster die Vorgaben. Treten jetzt keine Fehler mehr auf, wird im Verzeichnis `<Projekt>/<Projekt>.runs/impl_1` die Datei `or_gate.bit` erzeugt, mit der das FPGA „programmiert“ werden kann.

Vivado enthält das Werkzeug *Hardware Manager*, mit dem das Bitfile direkt in den FPGA transferiert werden kann, wenn das Entwicklungsboard per USB an den Arbeitsplatzrechner angeschlossen ist. Für SunRay-Arbeitsplätze verwenden wir jedoch die folgende Lösung: Suchen Sie in Ihrem Projektverzeichnis die Datei `or_gate.bit`. Geben Sie der Datei den Namen `<Gruppennummer>_or_gate.bit` um sie von den Dateien der anderen Gruppen abzugrenzen.

Senden Sie die Datei an `aes-hardwprakt@lists.tu-berlin.de` und melden Sie sich bei einem der Betreuer um die Datei auf den FPGA zu übertragen.

10 WorkFlow Zusammenfassung

An dieser Stelle soll noch einmal deutlich gemacht werden, wie der (grobe) typische Arbeitsablauf für das Hardwarepraktikum aussieht.

1. Lesen Sie sich die Aufgaben durch und nutzen Sie die Quellen/Folien als Hilfe
2. Laden Sie sich den Vorgaben Ordner runter und kopieren Sie die VHDL-Dateien in ihren *src*-Ordner
3. Bearbeiten Sie die Vorgaben mit Ihrem Lieblingseditor
4. Prüfen Sie die Funktionalität mit einer vorgegebenen oder selbst erstellten Testbench in QuestaSim
5. Fügen Sie die Dateien dann in Vivado Ihrem Projekt hinzu
6. Synthetisieren Sie diese und schauen sich die *Reports* an
7. Wenn der Report in Ordnung aussieht, laden Sie Ihre Lösung gemäß der Abgabeformalitäten hoch