

# **VHDL Wishbone Intercon Generator Geschrieben in Python, generiert VHDL**

HARALD HECKMANN

Studiengang Angewandte Informatik, Hochschule RheinMain

Wintersemester 2015/2016

Projekt bei Prof. Dr. Steffen Reith

**VHDL Wishbone Intercon Generator**  
**Geschrieben in Python, generiert VHDL**

**Zusammenfassung**

*Kurzfassung*

## Inhaltsverzeichnis

<b>1</b>	<b>Einleitung</b>	<b>4</b>
1.1	Wishbone . . . . .	4
1.2	Aufgaben des Wishbone Intercon . . . . .	4
1.3	Wishbone Beispiele . . . . .	4
<b>2</b>	<b>Das Projekt</b>	<b>4</b>
2.1	Projektaufbau . . . . .	4
2.2	Aufgaben der einzelnen Module . . . . .	5
2.3	Features . . . . .	5
2.3.1	Implementiert . . . . .	5
2.3.2	Im Generator nicht Berücksichtigt . . . . .	5
2.3.3	Nachträglich entfernt . . . . .	5
2.4	Arbeitsweise des Generators . . . . .	6
2.4.1	VHDL Templates . . . . .	6
2.5	Aufbau der Konfigurationsdatei . . . . .	6
2.6	Probleme / Schwierigkeiten . . . . .	6
<b>3</b>	<b>Schlusswort</b>	<b>6</b>

# 1 Einleitung

## 1.1 Wishbone

Wishbone ist ein Opensource Bussystem für System-on-Chip (SoC) Systeme. Wishbone gibt ein Regelwerk vor, das bei der Entwicklung von IP-Cores berücksichtigt werden kann, so dass alle Module, die diese Regeln berücksichtigen, in der Lage sind bei geeigneter Verschaltung miteinander zu kommunizieren. Die Teilnehmer eines Wishbone Bussystems werden in Master und Slave Komponenten unterteilt. Lediglich ein Master initiiert einen Datenaustausch und gibt dabei vor, ob es ein lesender oder schreibender Zugriff ist und an welcher Adresse die Daten abgelegt oder ausgelesen werden. Wishbone erlaubt Multimaster Systeme (mehrere Master Komponenten). Die Verschaltungseinheit, welche im weiteren Verlauf dieses Dokumentes mit dem Begriff „Intercon“ referenziert wird, verschaltet bei einer Anfrage des Masters mit Hilfe der vom Master angegebenen Adresse dessen Leitungen mit den Leitungen des für die Adresse zuständigen Slaves.

## 1.2 Aufgaben des Wishbone Intercon

- Adressdekodierer - den für die angegebene Adresse zuständigen Slave wählen
- (Nur bei Multimaster Systemen) Ein Arbiter, der vom Benutzer definiert wird
- Verbindung der Komponenten, sodass
  - variable Adress- und Datenbusbreiten berücksichtigt werden
  - Byte- und Wordadressierung berücksichtigen
  - bei unterschiedlicher Endianess einer Konvertierung durchgeführt wird

## 1.3 Wishbone Beispiele

Beispiel 1, Übungsszenario (Master) Button-Controller

(Slave1) LED-Controller

(Slave2) RGB-LED-Controller Beispiel 2, Reales Szenario (Master) MCU

(Slave1) VGA-Controller

(Slave2) SRAM-Controller

# 2 Das Projekt

Projektziele

## 2.1 Projektaufbau

Alle Module mit Assoziationspfeilen zeichnen, module im nächsten kapitel erklären (AKTIVITÄTSDIAGRAMM?)

Dateipfade angeben

## 2.2 Aufgaben der einzelnen Module

**libs/wb\_component.py** enthält Wishbone Component klasse mit: - wishbone componenten als superklasse -> generelle informationen die sich master als auch slave teilen -> unterklasse wishbone master mit masterspezifischen infos -> unterklasse wishbone slave mit slavespezifischen infos

**libs/wb\_intercon.py** enthält intercon klasse mit: - generellen intercon infos - einem master objekt (aus wb\_component->master) - beliebig vielen slave objekten (aus wb\_component->slave)

**libs/wb\_file\_manager.py** - parsen der config - ausgabe der aus der konfiguration gelesenen und in intercon objekt eingetragenen werte - generierung des intercon in VHDL

**libs/main.py** - ausführen der funktionen in richtiger reifolge - exitstatus - berechnungsdauer

## 2.3 Features

### 2.3.1 Implementiert

### 2.3.2 Im Generator nicht Berücksichtigt

Einige Konfigurationsmöglichkeiten werden mit in die Objekte übernommen, allerdings nicht vom Generator berücksichtigt. Dazu zählen:

Für Slaves:

- **addressing\_granularity** und **word\_size**  
Beschreibt, in welchem Abstand Adressen angesprochen werden können. Die **addressing\_granularity** kann byte oder word als Wert annehmen. Sollte word ausgewählt sein, muss man den Konfigurationsparameter **word\_size** setzen, der den Abstand in Bits angibt. Beispiel (Adressraum: 0x0 - 0x100):  
**addressing\_granularity** = byte, Adressen 0x0, 0x1, 0x2, 0x3, 0x4, 0x5, etc. können angesprochen werden  
**addressing\_granularity** = word und **word\_size** = 32, dann können Adressen in  $\log_2(32)=4$ er Schritten angesprochen werden : 0x0, 0x4, 0x8, etc.

Für Master und Slaves:

- **data flow**  
Gibt an, ob eine Komponente auf den Datenbus nur lesen, nur schreiben oder lesen und schreiben darf

### 2.3.3 Nachträglich entfernt

- **datatransfer** kann die Werte single, burst und rmw (read, modify, write) annehmen. Dieser Wert wurde auf Grund eines Missverständnisses zu Beginn des Projektes anschließend im Verlauf des Projektes wieder entfernt, da das Verhalten des Intercon sich nicht dadurch beeinflusst, ob die Daten einzeln, mit bursts oder mittels rmw übertragen werden

## 2.4 Arbeitsweise des Generators

Der Generator verwendet ein WishboneIntercon Objekt (aus wb\_intercon.py), dass wie bereits bekannt alle notwendigen Informationen aus der Konfigurationsdatei enthält. Anschließend wird eine Templatedatei geöffnet, die den gesamten VHDL-Code und Platzhalter enthält. Einige Platzhalter werden immer durch Text ersetzt, da diese Zeilen im Template bei jeder Konfiguration im generierten VHDL Code enthalten sein werden. Nicht der gesamte VHDL Code kann so generiert werden, da es VHDL Code gibt der nur bei bestimmten Konfigurationen erzeugt wird (Beispielsweise 0-6 zusätzliche Signale) oder sich bei unterschiedlichen Konfigurationen unterscheidet (Beispielsweise Verschaltung der Datenleitung mit gleicher und unterschiedlicher Endianess). Der Code, der von der Struktur her nicht immer gleich bleibt, wird in Python (im Textsegment) erstellt und ersetzt anschließend die restlichen Platzhalter.

### 2.4.1 VHDL Templates

`template_intercon.tmpl` wofür ist das gut?

`template_slave.tmpl` wofür ist das gut?

## 2.5 Aufbau der Konfigurationsdatei

Erkläre sections, keys wurden bereits erklärt

## 2.6 Probleme / Schwierigkeiten

beschreibe hier das kombinatorischer schaltkreis / latch problem

## 3 Schlusswort

War die vorgehensweise eine gute idee? was lief gut? was lief schlecht? was könnte man in zukunft ändern? gibt es bugs?