
# On Chip Signal Processing - RTL Implementierung RX

Daniel Ferrari
Nikolaus Haminger
Florian Kitzer
Matthäus Kücher
Alexander Traxler

12. Februar 2019

## Inhaltsverzeichnis

# 1 Übersicht

## 1.1 Spezifikation

In diesem Projekt sollte eine OFDM-RX-Architektur in VHDL implementiert werden. Ein grobes Blockdiagramm dieser RX-Komponente war vorgegeben und ist in Abbildung 1 sichtbar.

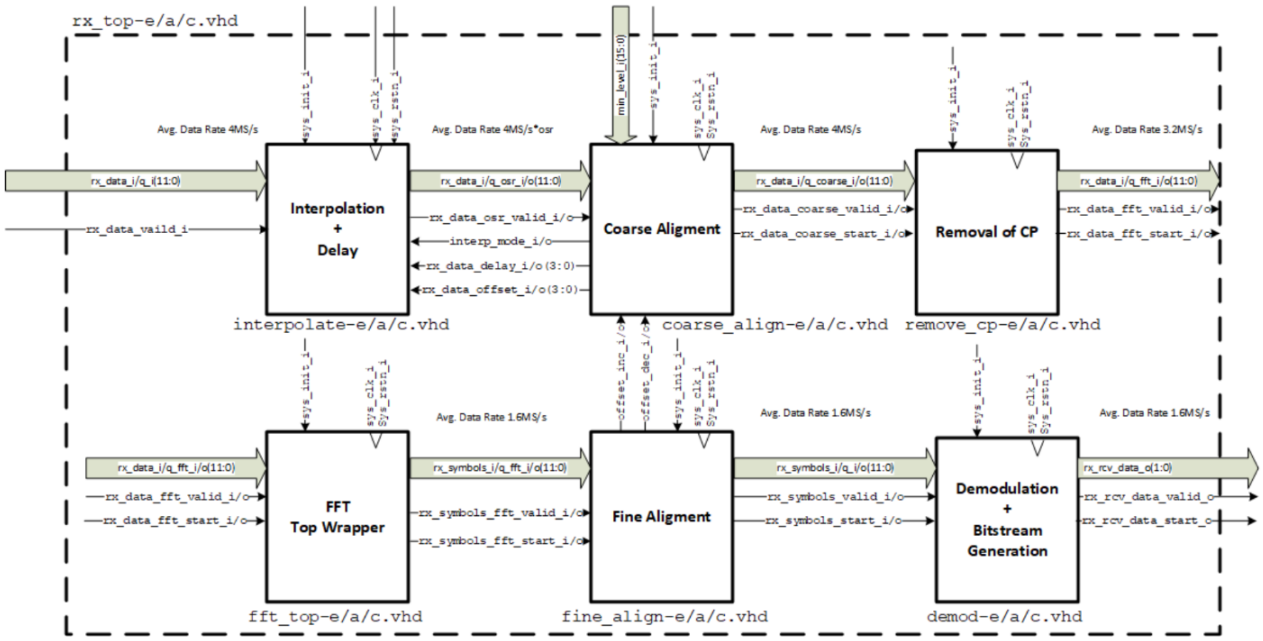


Abbildung 1: *Blockschaltbild der gesamten RX-Kette.*

## 1.2 Implementierung

Die Implementierung wurde weitgehend nach dem Vorbild des spezifizierten Blockschaltbildes vollzogen. Allerdings wird der Offset und das Delay in der Entity *Coarse Alignment* gehandhabt, weshalb die Leitungen `rx_data_delay`, `rx_data_offset` und `interp_mode` zwischen *Interpolation* und *Coarse Alignment* eliminiert wurden.

## 1.3 Toplevel-Simulation

Zur Verifikation der gesamten RX-Kette wurde eine Toplevel-Simulation erstellt. Für den Vergleich der Ergebnisse der Hardware-Kette wurde ein Golden Model in Python implementiert. Diese Implementierung berechnet die Input-Signale für die Hardware-Simulation (`rx_data_i`), speichert diese in CSV-Dateien ab und simuliert die gesamte RX-Kette in einer Hardware-ähnlichen Implementierung. Die Ergebnisse des Golden Models werden ebenfalls abgespeichert und am Ende der Hardware-Simulation mit eben jenen Ergebnissen verglichen. Außerdem werden *BER* und *EVM* berechnet sowie Konstellationdiagramme jeweils für beide Simulationen erstellt.

**Achtung**: Matlab wird nicht benötigt, da das gesamte Golden Model sowie die Verifikation in Python implementiert wurde (mit Fixed-Point-Repräsentation und Taylor-Approximation der

Interpolation). Die Gründe für diese Maßnahme sind die bessere Performance von Python, die einfachere Interaktion aus der Modelsim-Simulation und die bessere Strukturierungsmöglichkeit des Codes.

### 1.3.1 Anleitung zur Toplevel-Simulation

Eine ausführliche Anleitung befinden sich in der `README.md` des Repositories. Hier kurz noch einmal zusammengefasst:

**Anforderungen**:

- Modelsim oder Questasim
- Python3 (https://www.python.org/downloads/)
- numpy und matplotlib (`pip install numpy matplotlib`)

**Ausführung**:
Um die Simulation zu starten, einfach `start_rx_simulation.bat <cmd|gui|clean>` oder `start_rx_simulation.sh <cmd|gui|clean>` im Root-Folder des Repositories ausführen.

### 1.3.2 Simulations-Output

Nachfolgend ist der Output einer Simulation von 3 unterschiedlichen RX-Sequenzen zu sehen.

```
# ========================================
# Test the OFDM RX path
# ========================================
# [      0.00 us] INFO    : Working in folder src/grpRx/unitTopLevel/sim
#
# Testing RX chain with symbol sequence #0
# ----------------------------------------
# [      0.12 us] INFO    : Generating input data and expected bitstream by calling ../src/
    golden_model.py
# [      0.12 us] INFO    : Loading input data and expected bitstream from files rx_in_signal0
    .csv and result_bits0.csv
# [      0.12 us] INFO    : Feeding system with RX symbols, this might take a while
# [    243.38 us] INFO    : Received  256 of 5120 output bits ( 1 of 20 chips)
# [    323.37 us] INFO    : Received  512 of 5120 output bits ( 2 of 20 chips)
# [    403.26 us] INFO    : Received  768 of 5120 output bits ( 3 of 20 chips)
# [    483.26 us] INFO    : Received 1024 of 5120 output bits ( 4 of 20 chips)
# [    563.26 us] INFO    : Received 1280 of 5120 output bits ( 5 of 20 chips)
# [    643.37 us] INFO    : Received 1536 of 5120 output bits ( 6 of 20 chips)
# [    723.38 us] INFO    : Received 1792 of 5120 output bits ( 7 of 20 chips)
# [    803.37 us] INFO    : Received 2048 of 5120 output bits ( 8 of 20 chips)
# [    883.26 us] INFO    : Received 2304 of 5120 output bits ( 9 of 20 chips)
# [    963.26 us] INFO    : Received 2560 of 5120 output bits (10 of 20 chips)
# [   1043.27 us] INFO    : Received 2816 of 5120 output bits (11 of 20 chips)
# [   1123.37 us] INFO    : Received 3072 of 5120 output bits (12 of 20 chips)
# [   1203.38 us] INFO    : Received 3328 of 5120 output bits (13 of 20 chips)
# [   1283.37 us] INFO    : Received 3584 of 5120 output bits (14 of 20 chips)
# [   1363.27 us] INFO    : Received 3840 of 5120 output bits (15 of 20 chips)
# [   1443.26 us] INFO    : Received 4096 of 5120 output bits (16 of 20 chips)
# [   1523.27 us] INFO    : Received 4352 of 5120 output bits (17 of 20 chips)
# [   1603.37 us] INFO    : Received 4608 of 5120 output bits (18 of 20 chips)
# [   1680.11 us] INFO    : Received enough output bits
# [   1680.11 us] INFO    : Verifying RX chain output
# [   1680.11 us] SUCCESS : Validation successful: BER=0.000, EVM=20.895db
#
# [   1680.11 us] INFO    : Scatter plot file written: scatter_plot0.png
#
# Testing RX chain with symbol sequence #1
# ----------------------------------------
# [   1680.13 us] INFO    : Generating input data and expected bitstream by calling ../src/
    golden_model.py
```

```
# [   1680.13 us] INFO    : Loading input data and expected bitstream from files rx_in_signal1
    .csv and result_bits1.csv
# [   1680.13 us] INFO    : Feeding system with RX symbols, this might take a while
# [   1923.38 us] INFO    : Received  256 of 5120 output bits ( 1 of 20 chips)
# [   2003.37 us] INFO    : Received  512 of 5120 output bits ( 2 of 20 chips)
# [   2083.26 us] INFO    : Received  768 of 5120 output bits ( 3 of 20 chips)
# [   2163.26 us] INFO    : Received 1024 of 5120 output bits ( 4 of 20 chips)
# [   2243.26 us] INFO    : Received 1280 of 5120 output bits ( 5 of 20 chips)
# [   2323.36 us] INFO    : Received 1536 of 5120 output bits ( 6 of 20 chips)
# [   2403.38 us] INFO    : Received 1792 of 5120 output bits ( 7 of 20 chips)
# [   2483.36 us] INFO    : Received 2048 of 5120 output bits ( 8 of 20 chips)
# [   2563.26 us] INFO    : Received 2304 of 5120 output bits ( 9 of 20 chips)
# [   2643.26 us] INFO    : Received 2560 of 5120 output bits (10 of 20 chips)
# [   2723.26 us] INFO    : Received 2816 of 5120 output bits (11 of 20 chips)
# [   2803.36 us] INFO    : Received 3072 of 5120 output bits (12 of 20 chips)
# [   2883.38 us] INFO    : Received 3328 of 5120 output bits (13 of 20 chips)
# [   2963.36 us] INFO    : Received 3584 of 5120 output bits (14 of 20 chips)
# [   3043.26 us] INFO    : Received 3840 of 5120 output bits (15 of 20 chips)
# [   3123.26 us] INFO    : Received 4096 of 5120 output bits (16 of 20 chips)
# [   3203.26 us] INFO    : Received 4352 of 5120 output bits (17 of 20 chips)
# [   3283.36 us] INFO    : Received 4608 of 5120 output bits (18 of 20 chips)
# [   3360.11 us] INFO    : Received enough output bits
# [   3360.11 us] INFO    : Verifying RX chain output
# [   3360.11 us] SUCCESS : Validation successful: BER=0.000, EVM=21.016db
#
# [   3360.11 us] INFO    : Scatter plot file written: scatter_plot1.png
#
# Testing RX chain with symbol sequence #2
# ----------------------------------------
# [   3360.13 us] INFO    : Generating input data and expected bitstream by calling ../src/
    golden_model.py
# [   3360.13 us] INFO    : Loading input data and expected bitstream from files rx_in_signal2
    .csv and result_bits2.csv
# [   3360.13 us] INFO    : Feeding system with RX symbols, this might take a while
# [   3603.24 us] INFO    : Received  256 of 5120 output bits ( 1 of 20 chips)
# [   3683.24 us] INFO    : Received  512 of 5120 output bits ( 2 of 20 chips)
# [   3763.26 us] INFO    : Received  768 of 5120 output bits ( 3 of 20 chips)
# [   3843.26 us] INFO    : Received 1024 of 5120 output bits ( 4 of 20 chips)
# [   3923.36 us] INFO    : Received 1280 of 5120 output bits ( 5 of 20 chips)
# [   4003.38 us] INFO    : Received 1536 of 5120 output bits ( 6 of 20 chips)
# [   4083.36 us] INFO    : Received 1792 of 5120 output bits ( 7 of 20 chips)
# [   4163.27 us] INFO    : Received 2048 of 5120 output bits ( 8 of 20 chips)
# [   4243.26 us] INFO    : Received 2304 of 5120 output bits ( 9 of 20 chips)
# [   4323.27 us] INFO    : Received 2560 of 5120 output bits (10 of 20 chips)
# [   4403.36 us] INFO    : Received 2816 of 5120 output bits (11 of 20 chips)
# [   4483.38 us] INFO    : Received 3072 of 5120 output bits (12 of 20 chips)
# [   4563.36 us] INFO    : Received 3328 of 5120 output bits (13 of 20 chips)
# [   4643.27 us] INFO    : Received 3584 of 5120 output bits (14 of 20 chips)
# [   4723.26 us] INFO    : Received 3840 of 5120 output bits (15 of 20 chips)
# [   4803.27 us] INFO    : Received 4096 of 5120 output bits (16 of 20 chips)
# [   4883.36 us] INFO    : Received 4352 of 5120 output bits (17 of 20 chips)
# [   4963.38 us] INFO    : Received 4608 of 5120 output bits (18 of 20 chips)
# [   5040.10 us] INFO    : Received enough output bits
# [   5040.10 us] INFO    : Verifying RX chain output
# [   5040.10 us] SUCCESS : Validation successful: BER=0.000, EVM=22.675db
#
# [   5040.10 us] INFO    : Scatter plot file written: scatter_plot2.png
#
# ========================================
# Simulation end @5040.10 us
# ========================================
# Number of checks :           3
# Successful       :           3
# Warnings         :           0
# Errors           :           0
# ========================================
# SIMULATION SUCCESS
# ========================================
```

### 1.3.3 Konstellationsdiagramm

Abbildung 2 zeigt die Konstellationsdiagramme einer Symbol-Sequenz für die Golden-Model- und die Hardware-Simulation. Die Plots sind nach einer Simulation im Ordner `src/grpRx/unitTopLevel/sim` zu finden.

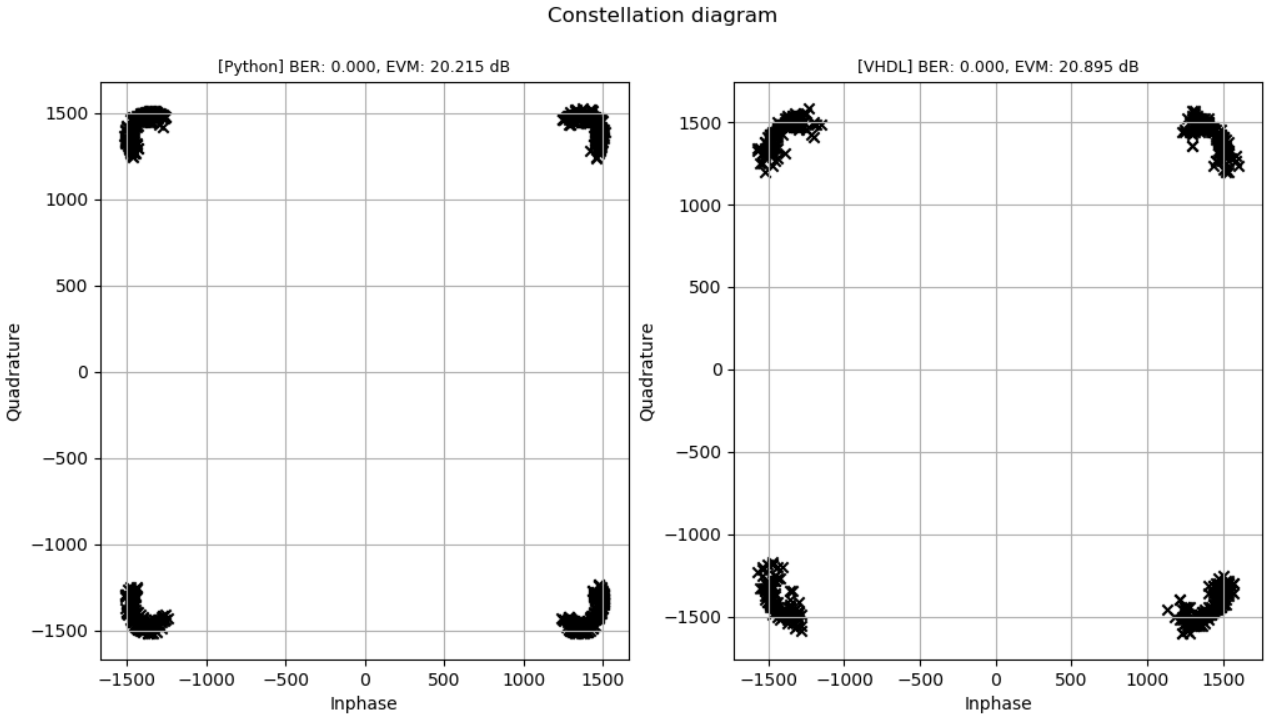


Abbildung 2: *Konstellationsdiagramme der Golden-Model- und Hardware-Simulationen*

# 2 Komponenten

## 2.1 Interpolation

## 2.2 Coarse Alignment

Die Aufgabe des `Coarse Alignment` ist das Finden des Beginns einer OFDM-Übertragung. Für das Finden der Beginns wird die Methode nach Schmidl und Cox verwendet. Dazu wird in Hardware die Korrelation über zwei identische Halbsymbole (= Trainingssymbol) gerechnet und das Maximum detektiert. Danach werden fortlaufend die Steuersignale `start_of_symbol` und `data_valid` des Datenstromes gesetzt und nach den Vorgaben des `Fine Alignment` korrigiert.

### 2.2.1 Architektur

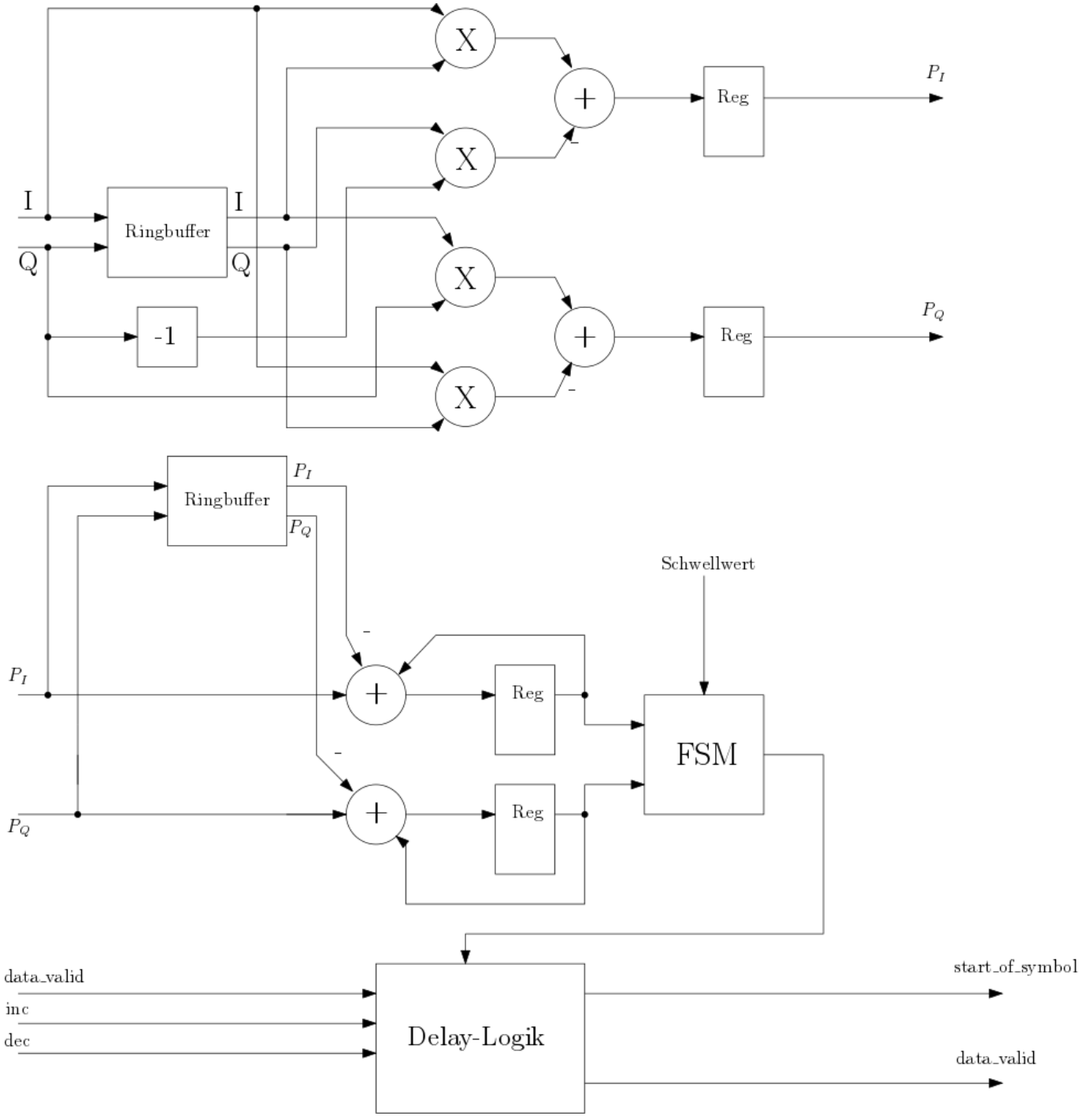


Abbildung 3: *Architektur des* `Coarse Alignment`*-Blocks. Links oben sind die Eingangsdaten als I und Q Anteil zu sehen. Darauf folgt die Korrelationsberechnung, gefolgt von der FSM zum Erkennen des Maximums. Unten im Bild ist die Logik zur Steuersignalgenerierung zu sehen.*

### 2.2.2 Simulation

Für die Simulation wurde sowohl Matlab als auch Modelsim verwendet. Mit Hilfe von Matlab wurde ein Empfangssignal mit der Länge von 100 Symbolen generiert. In diesem Empfangssignal ist das Trainingssymbol an der zweiten Stelle. Der Beginn der Korrelation des generierten Empfangssignals ist in Abbildung 4 zu sehen. Der grün umrandete Signalteil stellt das Trainingssymbol dar. Das obere Diagramm zeigt das Ergebnis der iterativen Implementierung. Das untere Diagramm zeigt das Ergebnis der formalen Implementierung. Es ist zu erkennen, dass beide Diagramme bis auf eine Verschiebung das gleiche Ergebnis darstellen.

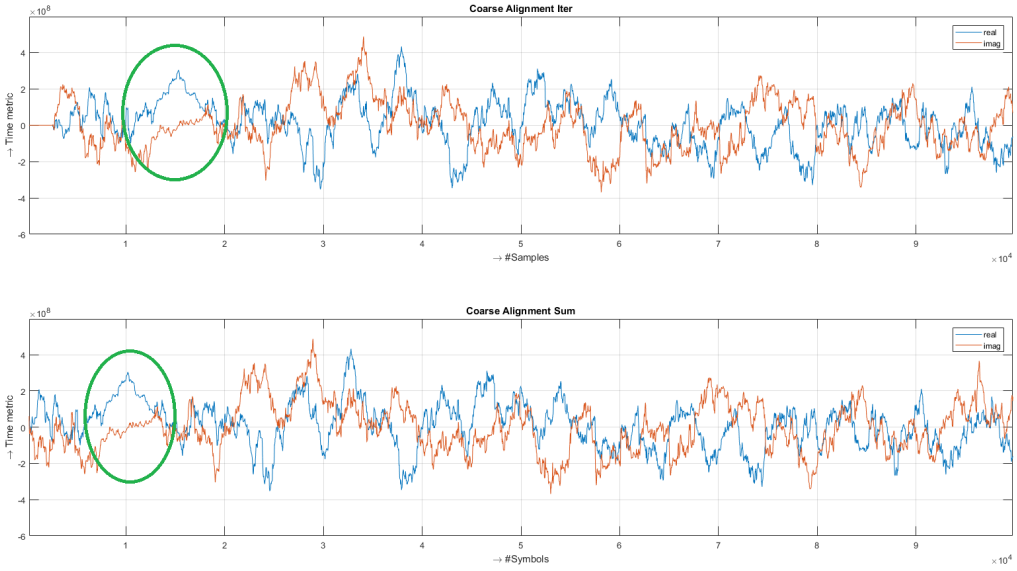


Abbildung 4: *Beginn des Empfangssignals. Oben ist die iterative und unten die formale Implementierung dargestellt. Der grün umrandete Signalteil stellt das Trainingssymbol dar.*

Für die Simulation wurde der Simulator Modelsim verwendet. Als Testframework wurde UVVM verwendet. Dadurch war es möglich ein besser strukturierte VHDL-Testbench zu entwerfen. In der Simulation wird das mit Matlab generierte Empfangssignal an den `Coarse Alignment`-Block angelegt und auf die Detektion des Trainingssymbols gewartet. Nach der Detektion wird das Timing der Steuersignale überprüft. Darauf wird überprüft, ob die Daten korrekt vom Block weiter gegeben werden. Am Ende wird das Verhalten im Zusammenhang mit den Steuerleitungen vom `Fine Alignment` und dem `init`-Signal überprüft. In Abbildung 5 ist ein Teil der Waveform der Simulation zu sehen. Die berechnete Korrelation in der Simulation stimmt mit der Berechnung in Matlab überein.

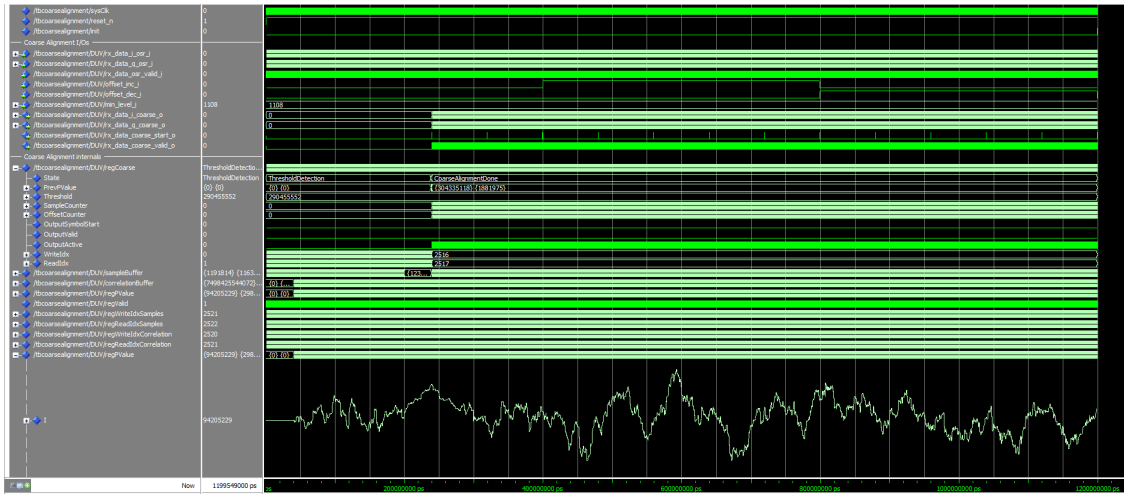

Abbildung 5: *Waveform der Simulation des* ***`Coarse Alignment`****. Im unteren Teil der Bildes ist die Korrelation des Empfangssignales zu erkennen.*

Um die Simulation starten zu können, müssen die Skripen `compile_uvvm.do`, `compile.do` und `run_simulation.do` in dieser Reihenfolge ausgeführt werden.

### 2.2.3 Synthese

Für den implementierten `Coarse Alignment`-Block wurde eine Synthese mit Quartus durchgeführt. Als Zielplattform wurde FPGA 5CSXFC6D6F31C6 der DE-10 Standard-Boards verwendet. Folgende Ressourcen werden für den `Coarse Alignment`-Block benötigt. Diese sind in der nachstehenden Tabelle 1 zu sehen.

| **Ressource** | **Menge** |
|---|---|
| Register | 180 |
| ALMs | 168 |
| Memory Bits | 15360 |
| DSP-Blöcke | 2 |
| Pin (theoretisch) | 72 |

Tabelle 1: Tabelle mit benötigten Ressourcen.

Die Timing-Analyse hat ergeben, dass der `Coarse Alignment`-Block im 1100mV-85C-Model mit einer maximalen Taktfrequenz von 111,68 MHz betrieben werden kann. Als Systemtakt sind 100 MHz vorgesehen.

## 2.3 Cyclic Prefix Removal

## 2.4 FFT Wrapper

## 2.5 Fine Alignment

## 2.6 Demodulation