

Demoprojekt Direct Memory Access (DMA) anhand einer Fast Fourier Transform (FFT)

Sebastian Wagner

February 15, 2025

Contents

| | | |
|----------|--|-----------|
| 1 | Aufgabenstellung | 2 |
| 2 | Verwendung / Quick-Start guide | 3 |
| 2.1 | Verwendungsablauf | 3 |
| 2.2 | Mitgelieferte Dateien | 3 |
| 2.3 | Erstellung eigener Signaldateien und Visualisierung der FFT Dateien | 5 |
| 3 | Direct Memory Access | 6 |
| 4 | Fast Fourier Transform | 7 |
| 5 | Blockaufbau der benutzten IP.Cores | 7 |
| 5.1 | FFT IP-Core | 7 |
| 5.1.1 | FFT-Implementation | 9 |
| 5.1.2 | FFT Konfigurationsanschluss über Konstanten | 11 |
| 5.2 | DMA IP.Core | 12 |
| 5.2.1 | DMA-Implementation | 13 |
| 6 | Gesamtes Vivado Hardware Design | 14 |
| 7 | Vitis Software Design | 16 |

List of Figures

| | | |
|---|--|---|
| 1 | Wezterm Befehl um sich mit einem Port zu verbinden | 3 |
|---|--|---|

| | | |
|----|--|----|
| 2 | Erwartete Ausgabe; Einlesen der Signaldatei auf SD Karte . . . | 4 |
| 3 | Erwartete Ausgabe; FFT Rohausgabe | 4 |
| 4 | Erzeugte Beispielssignaldatei | 5 |
| 5 | Einseitige FFT-Transformation | 6 |
| 6 | Port-Beschreibung FFT IP-Core | 8 |
| 7 | Wirklich vergebene Anschlüsse | 9 |
| 8 | Konfiguration FFT IP-Core | 10 |
| 9 | Implementation FFT IP-Core | 10 |
| 10 | Konfigurations Feld FFT IP Block | 11 |
| 11 | Konstanten für die Konfigurationsfelder der FFT | 12 |
| 12 | Block Diagram des AXI DMA IP-Cores [AMD24a, p. 7] . . . | 12 |
| 13 | Implementations DMA Vivado | 13 |
| 14 | Konfiguration des DMA IP-Cores | 14 |

List of Equations

1 Aufgabenstellung

Das in diesem Dokument beschriebene Projekt befasst sich mit der Erstellung eines Demoprojekts zur *Hardware basierten Berechnung einer FFT, mittels direktem Speicherzugriff, über einen DMA-Controller*. Der dazu benötigte Code wurde in **Vivado** (für die Beschreibung der Hardware) und **Vitis** (für die Beschreibung der Software) erstellt. Das Projekt wurde für ein Cora-Z7-7 (!) Entwicklungsboard erstellt. Falls ein anderes Entwicklungsboard verwendet werden soll muss möglicherweise die Hardwareplattform in Vivado neu erstellt werden. Es wurden folgende Teilziele behandelt:

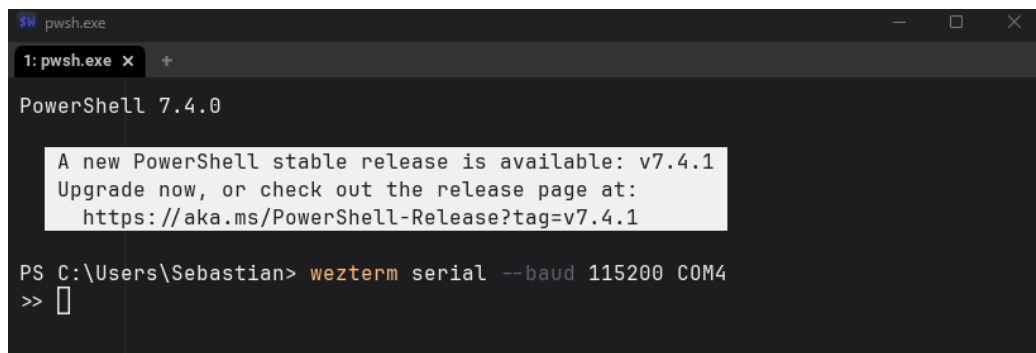
- Erstellen eines Blockdesigns in Vivado zur Beschreibung der Hardware Plattform.
- Generierung eines Signals im Zeitbereich durch ein Python Skript, mit entsprechender Visualisierung.
- Erstellen des Standalone Codes in Vitis, der auf dem ARM Kern des Coras läuft.
- Auslesen des generierten FFT Signals und anschließende Visualisierung über ein Python Skript.

2 Verwendung / Quick-Start guide

2.1 Verwendungsablauf

Der grundsätzliche Anwendungsablauf des Programms sieht wie folgt aus: Eine Signaldatei mit dem Namen *signal.bin*, die sich auf einer SD Karte befindet wird über den SD Kartenslot des Cora-Z7-7 Entwicklungsboards eingelesen. Anschließend wird die Signaldatei in dem internen DDR Speicher des Coras gespeichert. Der DMA Controller greift auf diesen Speicher zu und übergibt die Daten dem FFT Core. Nachdem der FFT Core fertig ist, wird das fertige Signal in einer Datei, mit dem Namen *FFT.bin* auf der SD Karte abgespeichert.

Debug Nachrichten, sowie Rohdaten der SD Karte und des FFT Cores werden während des Programms ausgegeben. Um auf den entsprechenden Port auf Windows zuzugreifen kann z.B. ein Terminalemulator wie *Wezterm* verwendet werden. Der Befehl, um sich beispielsweise mit Port 4 zu verbinden, wird in Abbildung 1 gezeigt. Die Baudrate ist hierbei 115200.



```
pwsh.exe
1: pwsh.exe x +
PowerShell 7.4.0

A new PowerShell stable release is available: v7.4.1
Upgrade now, or check out the release page at:
https://aka.ms/PowerShell-Release?tag=v7.4.1

PS C:\Users\Sebastian> wezterm serial --baud 115200 COM4
>> █
```

Figure 1: Wezterm Befehl um sich mit einem Port zu verbinden

Das Programm kann innerhalb der Vitis Entwicklungsumgebung gestartet werden. Die Plattform und die Applikation sollten bereits gebildet sein. In den folgenden beiden Abbildungen, findet man die erwarteten Terminalausgaben, insofern keine Fehler auftreten.

2.2 Mitgelieferte Dateien

In dem Projektordner lassen sich das Vivado Projekt, das Vitis Projekt (mit Hardware Plattform & Standalone Application) und zwei Python Skripts zur Signalgenerierung und Visualisierung finden. Außerdem sollte ein bereits erstelltes Signal mit dem Namen **signal.bin** und zwei Ausgabedateien mit den

Namen **FFT.bin** und **FFT2.bin** mitgeliefert sein.

```
! weston x +
Data read from SD-card at index 2: -0.861988+0.000000+1
Data read from SD-card at index 3: 0.307330+0.000000+1
Data read from SD-card at index 4: 0.291816+0.000000+1
Data read from SD-card at index 5: -1.199771+0.000000+1
Data read from SD-card at index 6: -0.412377+0.000000+1
Data read from SD-card at index 7: 1.888410+0.000000+1
Data read from SD-card at index 8: -0.240720+0.000000+1
Data read from SD-card at index 9: -0.544083+0.000000+1
Data read from SD-card at index 10: 0.439483+0.000000+1
Data read from SD-card at index 11: 0.411155+0.000000+1
Data read from SD-card at index 12: -1.471505+0.000000+1
Data read from SD-card at index 13: -0.524151+0.000000+1
Data read from SD-card at index 14: 1.650810+0.000000+1
Data read from SD-card at index 15: -0.277625+0.000000+1
Data read from SD-card at index 16: -0.319723+0.000000+1
Data read from SD-card at index 17: 0.635778+0.000000+1
Data read from SD-card at index 18: 0.641051+0.000000+1
Data read from SD-card at index 19: -1.620274+0.000000+1
Data read from SD-card at index 20: -0.568085+0.000000+1
Data read from SD-card at index 21: 1.325708+0.000000+1
Data read from SD-card at index 22: -0.406455+0.000000+1
Data read from SD-card at index 23: -0.249417+0.000000+1
Data read from SD-card at index 24: 0.028173+0.000000+1
Data read from SD-card at index 25: 0.919850+0.000000+1
Data read from SD-card at index 26: -1.626934+0.000000+1
Data read from SD-card at index 27: -0.746058+0.000000+1
Data read from SD-card at index 28: 0.997275+0.000000+1
Data read from SD-card at index 29: -0.588277+0.000000+1
Data read from SD-card at index 30: -0.351605+0.000000+1
Data read from SD-card at index 31: 0.956195+0.000000+1
Data read from SD-card at index 32: 1.165720+0.000000+1
Data read from SD-card at index 33: -1.515461+0.000000+1
Data read from SD-card at index 34: -0.287909+0.000000+1
Data read from SD-card at index 35: 0.746058+0.000000+1
Data read from SD-card at index 36: -0.762887+0.000000+1
Data read from SD-card at index 37: -0.593341+0.000000+1
Data read from SD-card at index 38: 0.986946+0.000000+1
Data read from SD-card at index 39: 1.300681+0.000000+1
Data read from SD-card at index 40: -1.342640+0.000000+1
Data read from SD-card at index 41: 0.016704+0.000000+1
Data read from SD-card at index 42: 0.625900+0.000000+1
Data read from SD-card at index 43: -0.864533+0.000000+1
Data read from SD-card at index 44: -0.906058+0.000000+1
Data read from SD-card at index 45: 0.925608+0.000000+1
Data read from SD-card at index 46: 1.274100+0.000000+1
```

Figure 2: Erwartete Ausgabe; Einlesen der Signaldatei auf SD Karte

```
! weston x +
FFT Data at 23: -0.247863-1.88843+1
FFT Data at 24: -0.095372-1.385670+1
FFT Data at 25: -0.040120-1.058575+1
FFT Data at 26: -0.113428-0.767227+1
FFT Data at 27: -0.594234-0.399783+1
FFT Data at 28: 11.006500-3.665380+1
FFT Data at 29: 1.458121-0.632704+1
FFT Data at 30: 1.084802-0.355384+1
FFT Data at 31: 0.983556-0.166344+1
FFT Data at 32: 0.934400-0.000000+1
FFT Data at 33: 0.983554+0.166344+1
FFT Data at 34: 1.084803+0.355384+1
FFT Data at 35: 1.458121+0.632704+1
FFT Data at 36: 11.006500+3.665380+1
FFT Data at 37: -0.594234+0.399783+1
FFT Data at 38: -0.113429+0.767228+1
FFT Data at 39: -0.040120+1.058575+1
FFT Data at 40: -0.095372+1.385670+1
FFT Data at 41: -0.247863+1.88843+1
FFT Data at 42: -0.527091+2.419802+1
FFT Data at 43: -1.046472+3.432506+1
FFT Data at 44: -2.219238+5.576513+1
FFT Data at 45: -6.966218+13.931931+1
FFT Data at 46: 16.757252-27.310413+1
FFT Data at 47: 4.913390-6.610668+1
FFT Data at 48: 3.200885-3.570557+1
FFT Data at 49: 2.497140-2.284511+1
FFT Data at 50: 2.094248-1.521100+1
FFT Data at 51: 1.808263-0.962335+1
FFT Data at 52: 1.554739-0.469115+1
FFT Data at 53: 1.256261+0.079380+1
FFT Data at 54: 0.741890+0.945804+1
FFT Data at 55: -1.080301+3.746585+1
FFT Data at 56: 8.225870-10.240263+1
FFT Data at 57: 3.339600-2.737110+1
FFT Data at 58: 2.662361-1.613320+1
FFT Data at 59: 2.394430-1.102993+1
FFT Data at 60: 2.252733-0.783397+1
FFT Data at 61: 2.167600-0.546621+1
FFT Data at 62: 2.114181-0.346495+1
FFT Data at 63: 2.083427-0.271311+1
Successfully ran AXI DMA Interrupt Data Exchange
Successfully wrote to SD-Card
--- Exiting main() ---
```

Figure 3: Erwartete Ausgabe; FFT Rohausgabe

2.3 Erstellung eigener Signaldateien und Visualisierung der FFT Dateien

Um eigene *signal.bin* Dateien zu erstellen, findet man in dem Projektverzeichnis ein Python Skript, mit dem Namen *GenerateSignal.ipynb*. Dieses Programm erstellt Signaldateien. Dabei werden eine zufällige Menge an Cosinus Funktionen, mit entsprechend zufällig gewählten Frequenzen, aufaddiert und anhand von Parametern wie Signallänge und Abtastrate evaluiert. Der Verlauf des Signals wird zudem geplottet und als *RealValueInput.png* & *ImaginaryValueInput.png* gespeichert. Für eine, beispielhaft erzeugte, Signaldatei wird dies in Abbildung 4 veranschaulicht.

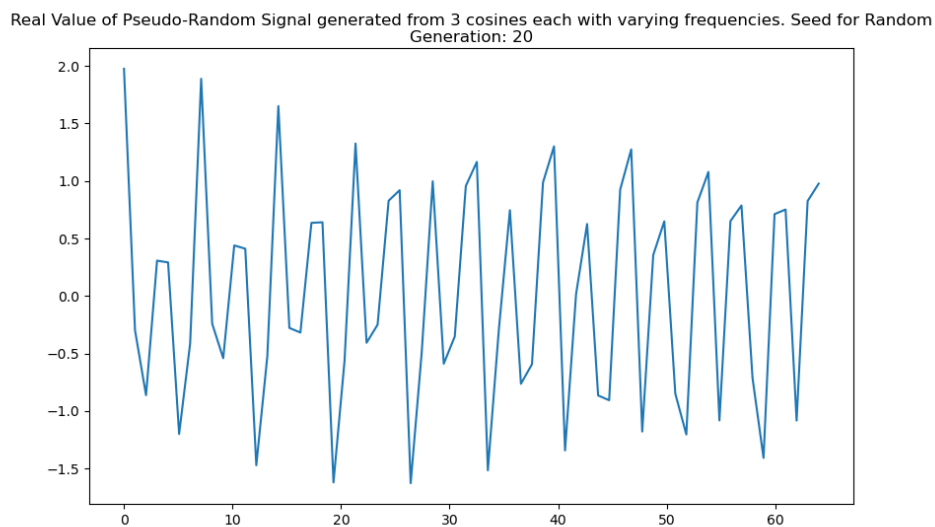


Figure 4: Erzeugte Beispielssignaldatei

Ähnlich dazu gibt es auch zur Visualisierung der FFT-Ausgabedatei ein Python Skript. Das Skript mit dem Namen *PlotFFT.ipynb*, plottet das einseitige Spektrum der FFT und speichert dieses wieder als png Datei. Das einseitige FFT Spektrum, des in Abbildung 4 gezeigten Spektrums wird in der nachfolgenden Abbildung veranschaulicht.

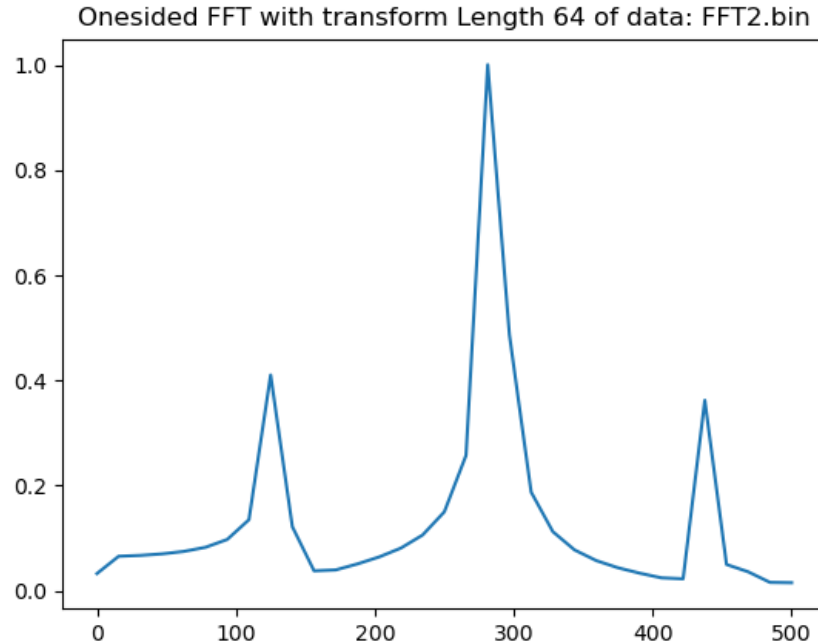


Figure 5: Einseitige FFT-Transformation

3 Direct Memory Access

Direct Memory Access beschreibt die Möglichkeit einer Komponente direkt auf ein gewisses Speichersegment zuzugreifen, ohne direkte Beteiligung der CPU. Großer Vorteil dieses Features ist, dass dadurch die CPU nicht ausgelastet wird. In unserem Fall wird die Berechnung einer 64 Point FFT nicht von der CPU selber durchgeführt, sondern die hierfür benötigte Arbeit wird auf einen dafür ausgelegten Hardwarebaustein durchgeführt. Der ARM CPU Kern muss dadurch nurnoch die jeweilige Aufgabe an eine Hardware Adresse delegieren. Um DMA verwenden zu können muss ein IP-Core in die PL eingebunden werden. Der dafür konkret verwendete IP-Core, lautet AXI DMA Controller.

4 Fast Fourier Transform

Eine Fast Fourier Transform ist ein Algorithmus zur schnelleren Berechnung einer Diskreten Fourier Transformation. Im Vergleich zur normalen DFT beträgt die Berechnungskomplexität der FFT nur $O(\log(N)*N)$, anstatt $O(N^2)$. Aufgrund der Bedeutsamkeit, und der vielzähligen Anwendungen der FFT, gilt diese als einer der wichtigsten Algorithmen.

5 Blockaufbau der benutzten IP.Cores

In diesem Abschnitt wird eine Übersicht der verwendeten IP-Cores und deren Anschluss- sowie Konfigurationsmöglichkeiten gegeben.

5.1 FFT IP-Core

Es wird der standardmäßig vorhanden IP-Core zur Berechnung von FFTs verwendet. Die folgende Abbildung zeigt das Blockbild des IP-Cores.

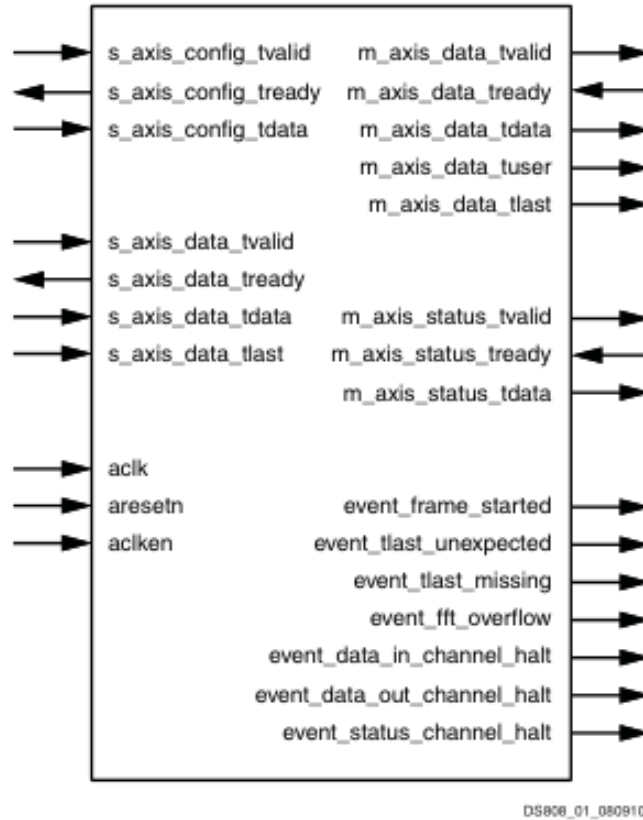


Figure 2-1: Core Schematic Symbol

Figure 6: Port-Beschreibung FFT IP-Core

Zusätzlich zu den hier gezeigten Anschlussmöglichkeiten, lässt sich der IP-Core auch über eine AXI-Stream Schnittstelle steuern. Diese Herangehensweise wurde in diesem Projekt auch gewählt, da man ohne größeren Aufwand die FFT- und DMA IP-Cores mittels der AXI Schnittstelle verknüpfen kann. Die dadurch, wirklich zu verwendenden Anschlüsse, sind in der nachfolgenden Abbildung 7 zu sehen. Außer dem eigentlichen Datenanschluss, müssen noch Clock, Reset, sowie zwei Config Ports verbunden werden.

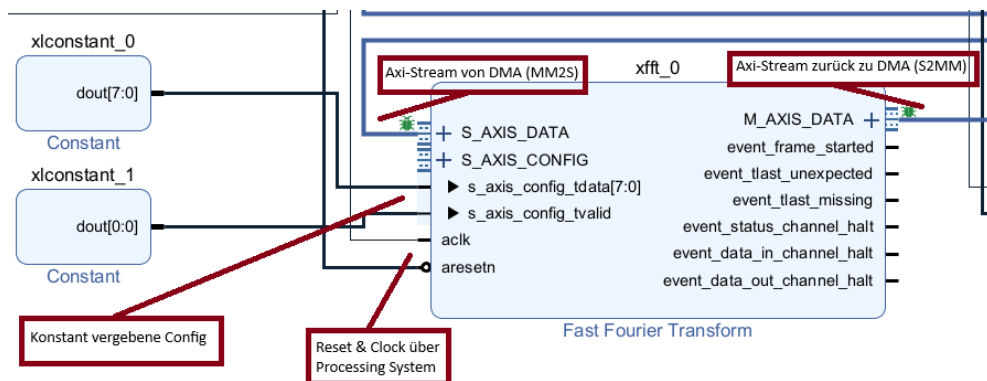


Figure 7: Wirklich vergebene Anschlüsse

5.1.1 FFT-Implementation

Der FFT IP-Core wurde wie folgt konfiguriert. (Tab Configuration)

- Number of Channels : 1
- Transform Length : 64
- Target Clock : 250 Mhz
- Target Data Throughput : 50 MSPS
- Architecture Choice : Automatically Select

Der FFT IP-Core wurde hauptsächlich auf seinen standardmäßig Einstellungen gelassen. Sofern in späteren Arbeiten eine konfigurierbare Transformlänge einstellbar sein soll, kann man dies in diesem Reiter auch einstellen. Dabei zu beachten ist, dass dann eine Transformlänge von $2^N \dots MaxTransformlength$ einstellbar ist. [AMD24b, p.16] Außerdem erhöht sich dann auch das Konfigurationsfeld. Unter dem Reiter Implementation des IP-Cores wurde als Datenformat Floating Point ausgewählt. Als Input Data Width 32 und als Phase Factor Width 24. Unter Control Signals wurde ARESETn ausgewählt. Als Output Ordering wurde Bit Reversed ausgewählt. Die beiden Einstellungen zu Konfiguration und Implementation können in der folgenden Abbildung abgelesen werden.

Figure 8: Konfiguration FFT IP-Core

Figure 9: Implementation FFT IP-Core

Wie sich die einzelnen Konfigurationsfelder zusammen setzen kann in [AMD24b, p.16] nachgelesen werden. Allgemein müssen nur Ports angeschlossen werden, die nach der Konfiguration des FFT IP-Cores auch wirklich Verwendung finden. Diese sind auch unter *Implementation Details* innerhalb des *customize IP* Fensters zu sehen. Die folgende Abbildung zeigt welche Daten konfiguriert werden müssen.

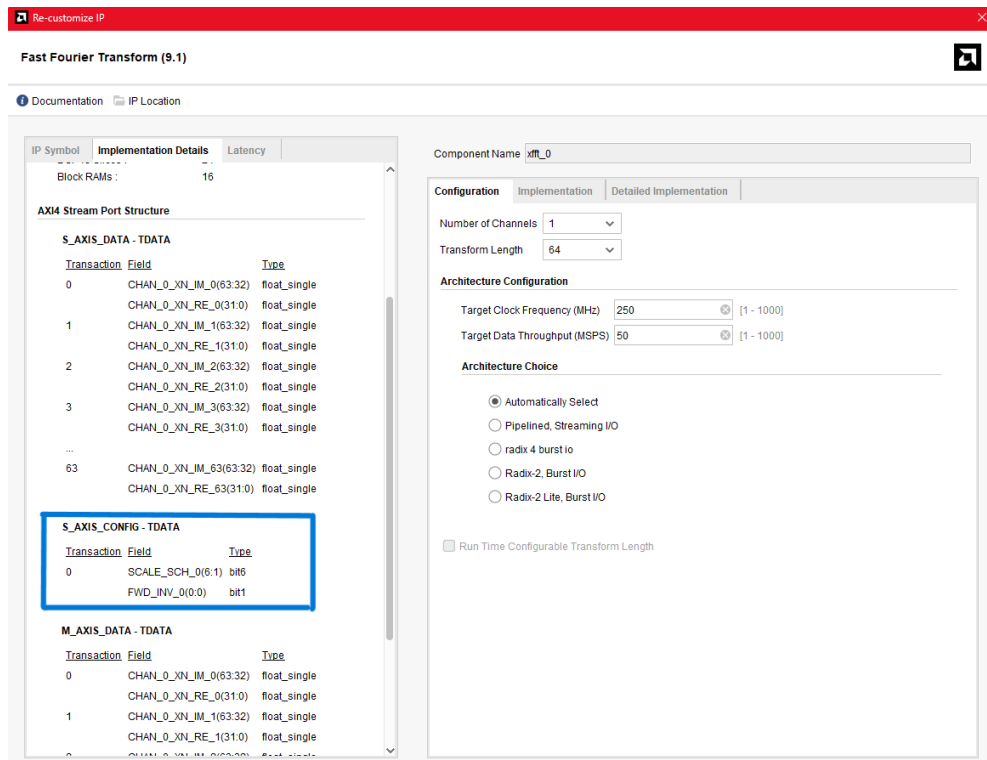


Figure 10: Konfigurations Feld FFT IP Block

5.1.2 FFT Konfigurationsanschluss über Konstanten

Da sich während der Laufzeit der Anwendung keine config-Daten ändern, wurden diese hier durch Konstanten realisiert. Beide Konstanten haben den Wert 1, was bei FWD_INV eine Forwärts FFT bedeutet und bei SCALE_SCH eine Skalierung von 1.

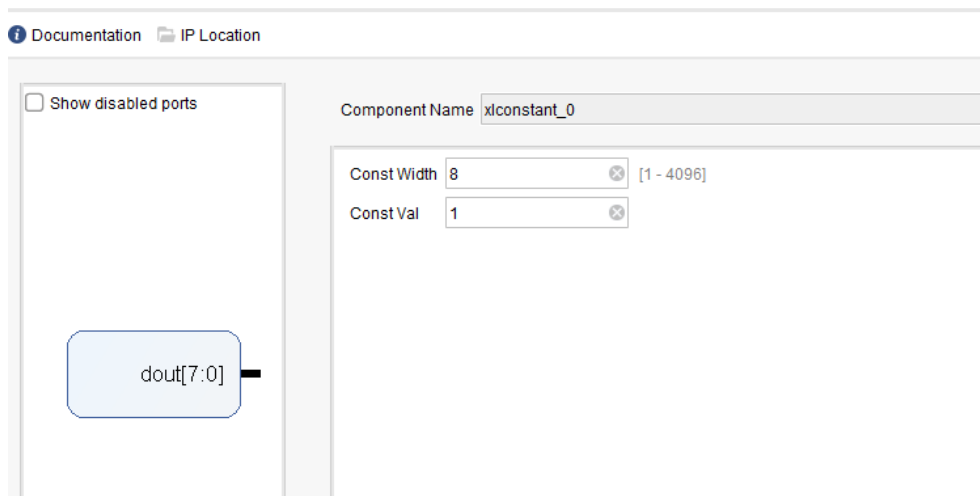


Figure 11: Konstanten für die Konfigurationsfelder der FFT

5.2 DMA IP.Core

In der folgenden Abbildung ist das Blockdiagramm des DMA IP-Cores zu sehen.

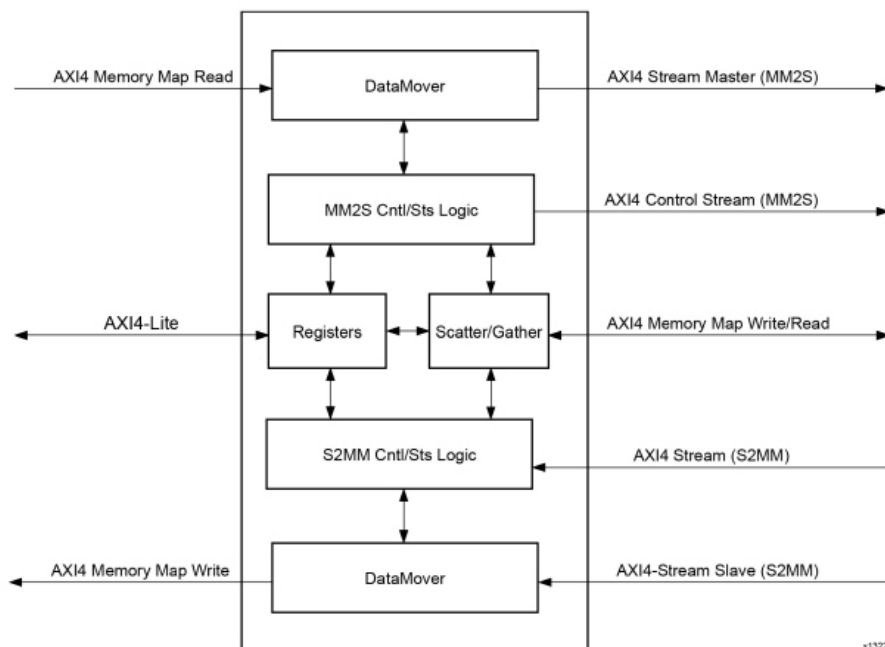


Figure 12: Block Diagram des AXI DMA IP-Cores [AMD24a, p. 7]

5.2.1 DMA-Implementation

Konzeptuell erhält das DMA Modul jeweils einen *gemappten* Speicherbereich für die Inputs des FFTs (in unserem Fall 64 complex floats) und einen Speicherbereich für die Ausgänge des FFTs (auch wieder 64 complex floats), eine *Control* Axi Schnittstelle und zwei Streams für die Daten für und von der FFT. Diese Zusammenhänge sollen in der folgenden Abbildung nochmal veranschaulicht werden.

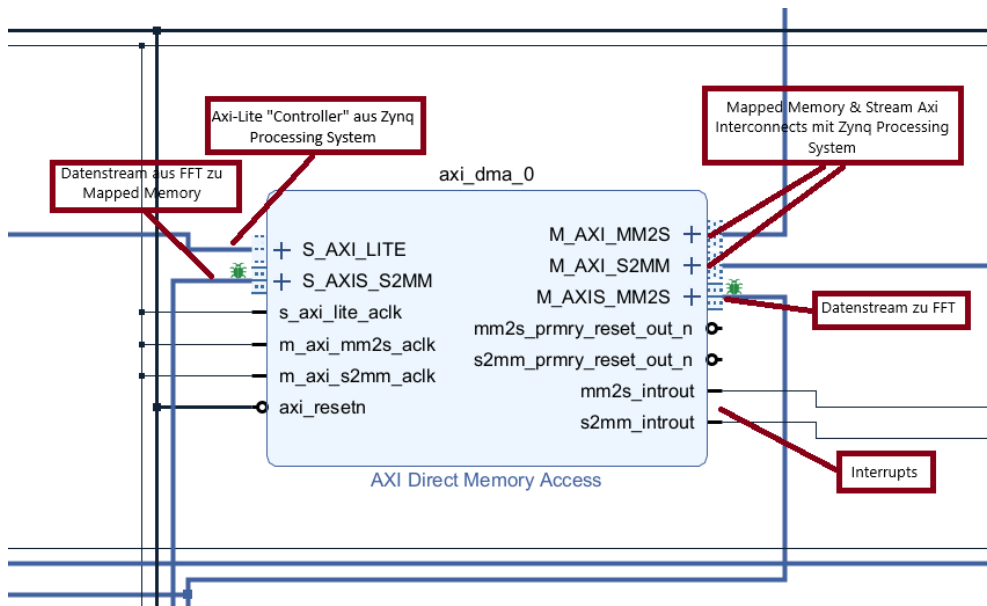


Figure 13: Implementations DMA Vivado

Die Konfiguration des DMAs ist der nächsten Abbildung abzulesen.

Component Name

☐ Enable Asynchronous Clocks (Auto)

☐ Enable Scatter Gather Engine

☐ Enable Micro DMA

☐ Enable Multi Channel Support

☐ Enable Control / Status Stream

Width of Buffer Length Register (8-26) bits

Address Width (32-64) bits

☒ Enable Read Channel

Number of Channels
 Memory Map Data Width
 Stream Data Width
 Max Burst Size
☐ Allow Unaligned Transfers

☒ Enable Write Channel

Number of Channels
 Memory Map Data Width
 Stream Data Width
 Max Burst Size
☐ Allow Unaligned Transfers

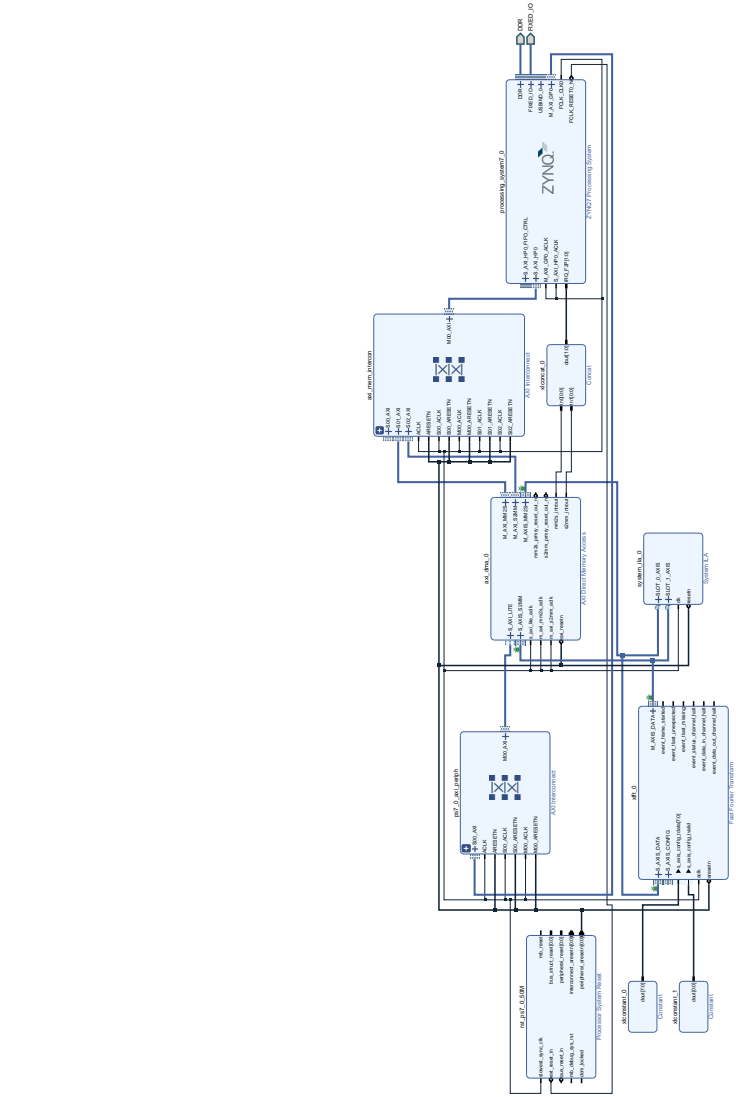
☐ Use Rlength In Status Stream

☐ Enable Single AXI4 Data Interface

Figure 14: Konfiguration des DMA IP-Cores

6 Gesamtes Vivado Hardware Design

Im folgenden Abschnitt ist der Block Design der Hardware zu als pdf zu sehen.



7 Vitis Software Design

In Vitis wurden hauptsächlich die bereits vorhandenen Beispielprojekte zu DMAs verwendet. Die eigentlichen FFT Rohdaten werden von einer SD-Karte ausgelesen und dann wieder abgespeichert. Aufschlussreicher ist hierbei das konkrete Vitis Projekt einzusehen. Oberflächlich betrachtet werden folgende Schritte abgearbeitet:

- Einstellen von Baseadressen aufsetzen von Interrupt Controller, DMA-Controller und SD-Karten Controller
- Auslesen einer Rohdatei aus der SD-Karte (BIN Datei Größe $64(\text{Anzahl Dateneinträge}) * \text{sizeof}(\text{complex float})$) -> in TxBuffer für DMA abspeichern
- Simple DMA-Transfer (ohne Scatter Gather siehe mehr [AMD24a]) Zynq -> DMA und DMA -> Zynq
- Warten auf Interrupts von DMA (Lesen und Schreiben vollendet)
- Abspeichern des RxBuffers (fertige FFT Transformation) auf SD-Karte

References

- [AMD24a] AMD. Axi dma logicore ip product guide (pg021). https://docs.amd.com/r/en-US/pg021_axi_dma, 2024.
- [AMD24b] AMD. Fast fourier transform logicore ip product guide (pg109). <https://docs.amd.com/r/en-US/pg109-xfft>, 2024.