

# HOCHSCHULE OSNABRÜCK

UNIVERSITY OF APPLIED SCIENCES

Projektbericht:

CRC-DMA

Modul:

Elektronische Systeme

Betreuer:

Prof. Dr. Winfried Gehrke

Jan Oliver Schöwerling

Autoren:

Matthias Biermann

Sebastian Meyer

Datum: 18. Februar 2025

# Inhaltsverzeichnis

<b>1</b>	<b>Einleitung</b>	<b>1</b>
1.1	Projektbeschreibung und Zielsetzung . . . . .	1
1.2	Anforderungen . . . . .	1
1.3	CRC - Theoretische Grundlagen . . . . .	1
<b>2</b>	<b>Konzept und Umsetzung des CRC-DMA-IP</b>	<b>2</b>
2.1	Entity / Schnittstellen . . . . .	2
2.2	Architecture . . . . .	4
2.2.1	DMA Komponente - axis_dma . . . . .	4
2.2.2	CRC-Berechnungskomponente - axis_crc . . . . .	5
2.3	AXI-CRC-DMA-IP . . . . .	6
2.4	Einbindung in das Gesamtsystem . . . . .	7
<b>3</b>	<b>Software - Treiberfunktion</b>	<b>8</b>
<b>4</b>	<b>Testkonzept und Ergebnisse</b>	<b>9</b>
4.1	Verifikation der CRC-Berechnungskomponente . . . . .	9
4.1.1	Referenzprogramm . . . . .	9
4.1.2	Simulation / Testbench . . . . .	9
4.2	Simulation des CRC-DMA-IPs . . . . .	10
4.3	Verifikation des Gesamtsystems . . . . .	11
4.3.1	Bare-Metal-Programm . . . . .	11
4.3.2	Testprogramm . . . . .	12
<b>5</b>	<b>Fazit und Ausblick</b>	<b>13</b>
<b>6</b>	<b>Anahng</b>	<b>14</b>
6.1	Hardware . . . . .	14
6.2	Software . . . . .	14
<b>7</b>	<b>Quellen</b>	<b>15</b>

# 1 Einleitung

## 1.1 Projektbeschreibung und Zielsetzung

Dieses Projekt umfasst die Konzeptionierung und Implementierung eines IPs zur Berechnung einer 32 Bit Cyclic Redundancy Check (CRC)-Prüfsumme. Die Verarbeitung erfolgt paketweise, wobei die Daten aus dem Speicher gelesen, verarbeitet und mit angehängter Prüfsumme wieder zurückgeschrieben werden. Der Speicherzugriff erfolgt über eine AXI-Master-Schnittstelle.

Nach Implementierung der Hardware erfolgt die Erstellung von Software, die die Konfiguration und Steuerung des IP-Cores übernimmt. Der Softwarepart dieser Arbeit besteht aus zwei Programmen. Beim ersten handelt es sich um ein C-Programm, das ohne Betriebssystem auf dem Mikroprozessor des hier verwendeten Entwicklungsboards [2] ausgeführt wird. Das zweite Programm wird im Kontext eines Linux Betriebssystems ausgeführt und beinhaltet die Implementierung eines ausführlichen Testprogramms, das die Funktionalität des IP-Cores überprüft.

## 1.2 Anforderungen

Die Anforderungen an das Projekt und an das zu erstellende IP umfassen folgende Punkte:

- Implementierung eines **CRC-Berechnungsmoduls** als **IP-Core**
- **Paketweises Einlesen** von Daten aus dem Speicher über einen **Full-AXI-Master**
- Berechnung einer korrekten CRC-Prüfsumme mit **konfigurierbaren Parametern** (InitialValue, Polynomial, FinalXOR, Input/OutputReflected)
- **Anfügen der Prüfsumme** an das ursprüngliche Datenpaket
- **Zurückschreiben** der Daten über einen **Full-AXI-Master**
- Unterstützung von **Interrupts** zur Synchronisation mit der CPU
- Validierung der Hardwareimplementierung durch **Simulation und Messungen**
- Implementierung eines **Testprogramms** zur Überprüfung der Funktionalität
- Möglichst hoher **Datendurchsatz** bei einer Taktfrequenz von 100 MHz

## 1.3 CRC - Theoretische Grundlagen

Der Cyclic Redundancy Check (CRC) ist ein Verfahren zur Fehlererkennung, das in der Regel in digitalen Kommunikationssystemen eingesetzt wird. Die CRC-Berechnung basiert auf der Division eines Datenwortes durch ein Generatorpolynom. Der Rest der Division ist die CRC-Prüfsumme.

Die Berechnung der CRC-Prüfsumme erfolgt dabei mit verschiedenen Parametern [3]. Ein spezifischer Satz von Parametern definiert einen standard CRC-Algorithmus. Folgende Parameter sind definiert:

- **CRC-Länge / Width:** Die CRC-Länge definiert die Länge der Prüfsumme in Bit. In dieser Arbeit ist sie auf 32 Bit festgelegt.
- **Generatorpolynom / Polynomial:** Das verwendete Generatorpolynom für die Polynomdivision.
- **Initialwert / Initial Value:** Der Initialwert ist der Wert, mit dem das CRC-Register initialisiert wird.

- **Eingangsreflexion / Input Reflected:** Die Bits des Eingangsdatenwortes werden vor der Berechnung umgekehrt.
- **Ergebnisreflexion / Output Reflected:** Das Ergebnis wird nach der Berechnung über die gesamte Bitbreite reflektiert.
- **Finales XOR / Final XOR:** Der finale CRC-Wert wird mit diesem Wert XOR-verknüpft, bevor er zurückgegeben wird.

Mit diesen Parametern lässt sich jeder standardisierte CRC-Algorithmus definieren.

## 2 Konzept und Umsetzung des CRC-DMA-IP

Im Rahmen dieses Projektes wurde ein IP-Block entwickelt, der die komplette CRC-DMA Funktionalität beinhaltet. Dieser kann anschließend in ein größeres System eingebunden werden, um die CRC-Prüfsummen von Datenpaketen zu berechnen und diese wieder zurückzuschreiben. Die Einbindung in ein Gesamtsystem wird in Abschnitt 2.4 beschrieben.

Alle im Folgenden dargestellten Blockschaltbilder und VHDL-Quelldateien befinden sich im Vivado-projekt `axi_crc_dma` in den beigefügten Dateien.

### 2.1 Entity / Schnittstellen

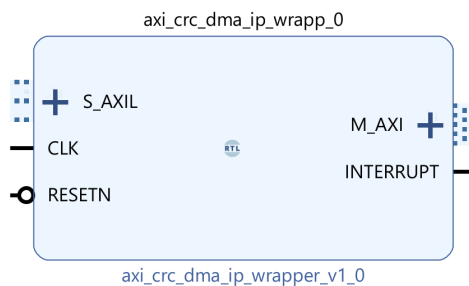


Abbildung 1: Entity des IP `axi_crc_dma_ip`

In Abbildung 1 ist die Entity des IPs `axi_crc_dma_ip` dargestellt.

Die AXI-Master Schnittstelle `M_AXI` dient zum Lesen und Schreiben der Daten aus dem Speicher.

Über die `S_AXIL` Schnittstelle kann das IP gesteuert und konfiguriert werden. In Tabelle 1 sind die dafür zur Verfügung stehenden Register aufgelistet.

Adresse	Registername	Beschreibung
0x00	<b>Control</b>	[0] Run
		[1] Interrupt Enable
		[31:2] Reserved
0x04	<b>Interrupt Status</b>	[0] Interrupt Status
		[31:1] Reserved
0x08	<b>Read Address</b>	[31:0] Startadresse der zu lesenden Datenpakete
0x0C	<b>Write Address</b>	[31:0] Zieladresse für Daten + Prüfsumme
0x10	<b>Packet Size</b>	[15:0] Paketgröße Minus 1 in Worte
		[31:16] Reserved
0x14	<b>Number Packets</b>	[15:0] Anzahl der Pakete Minus 1
		[31:16] Reserved
0x18	<b>Polynomial</b>	[31:0] Generatorpolynom für CRC-Berechnung
0x1C	<b>Initial Value</b>	[31:0] Initialwert der CRC-Berechnung
0x20	<b>Final XOR</b>	[31:0] Wert für Finale XOR-Verknüpfung
0x24	<b>InOutReflected</b>	[0] Input Reflected
		[1] Output Reflected
		[31:2] Reserved
0x28	<b>AxCache</b>	[3:0] M_AXI AWCACHE
		[7:4] M_AXI ARCCACHE
		[31:8] Reserved

Tabelle 1: Registerübersicht der CRC-DMA IP

Mit dem Register **Control** kann das IP gesteuert werden. Um die CRC-Berechnung zu starten, muss Bit 0 gesetzt werden. Solange das IP aktive ist, bleibt das Bit 0 gesetzt. Das Löschen des Bit 0 hat keine Wirkung. Ein gestarteter Vorgang kann also nicht vorzeitig abgebrochen werden.

Mit dem Register **Interrupt Status** kann der Status des Interrupts ausgelesen werden. Wenn der Interrupt aktiviert ist, wird dieser gesetzt, sobald die CRC-Berechnung abgeschlossen ist. Zeitgleich wird der Interruptausgang des IPs auf High gesetzt. Bevor das IP erneut gestartet werden kann, muss der Interrupt durch das Löschen des Bits 0 zurückgesetzt werden.

Die Register **Read Address** und **Write Address** dienen zur Konfiguration der Speicheradressen, von denen die Daten gelesen und wieder zurückgeschrieben werden. Die Register **Packet Size** und **Number Packets** dienen zur Konfiguration der Paketgröße und der Anzahl der zu verarbeitenden Pakete. Das IP arbeitet wort-weise. Es können keine einzelnen Bytes adressiert werden. Das heißt, dass die konfigurierten Adressen word-aligned sein müssen und die Paketgröße und -anzahl in Worten angegeben werden müssen.

Die Register **Polynomial**, **Initial Value**, **Final XOR** und **InOutReflected** dienen zur Konfiguration der CRC-Berechnung. Während eine CRC-Berechnung aktiv ist, dürfen diese Register nicht geändert werden, da es sich direkt auf die CRC-Berechnung auswirkt.

Mit dem Register **AxCache** können die Werte für die *ARCCache* und *AWCCache* Signale der AXI-Master Schnittstelle konfiguriert werden. Das Verändern dieses Registers ist nicht notwendig, da der Standardwert bereits eine korrekte Funktionalität des IPs gewährleistet. Es kann im Folgenden vernachlässigt werden.

## 2.2 Architecture

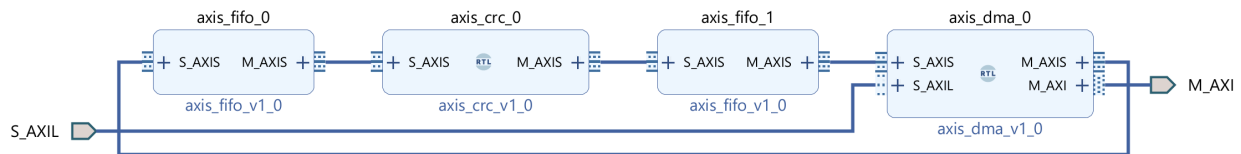


Abbildung 2: Architecture des IP `axi_crc_dma_ip` im Interfaces-View

In Abbildung 2 ist der grundlegende Aufbau des IPs `axi_crc_dma_ip` dargestellt. Es ist als Block-Design in Vivado unter der Bezeichnung `axi_crc_dma_ip` zu finden. Das IP besteht aus insgesamt vier Komponenten, die untereinander über AXI-Stream Schnittstellen kommunizieren und Daten weitergeben. In den folgenden Abschnitten werden die einzelnen Komponenten genauer beschrieben.

### 2.2.1 DMA Komponente - `axis_dma`

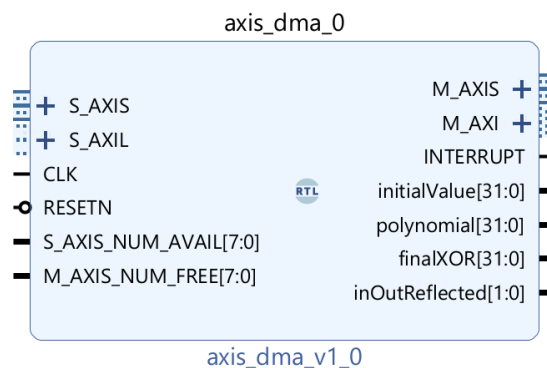


Abbildung 3: Entity der Komponente `axis_dma`

Die Komponente `axis_dma` ist die steuernde Komponente in diesem System. Sie ist für das Lesen und Schreiben der Daten aus dem Speicher über die `M_AXI` Schnittstelle zuständig. Dabei werden die gelesenen Datenpakete zunächst an einen FIFO-Speicher über die `M_AXIS` Schnittstelle weitergegeben. Über die `M_AXIS` Schnittstelle werden die Datenpakete mit angehängter Prüfsumme wieder angenommen und zurück in den Speicher geschrieben.

Die Komponente arbeitet bei den Speicherzugriffen über die `M_AXI` Schnittstelle immer mit möglichst großen Burstlängen, um möglichst effiziente Speicherzugriffe zu gewährleisten. Um sicherzustellen, dass das System dabei genug Daten aufnehmen bzw. zur Verfügung stellen kann, werden zwei FIFO-Speicher verwendet. Die maximale Burstlänge in diesem System beträgt 16 Worte. Die FIFO-Speicher haben eine Tiefe von 256 Worten, um ausreichend Daten puffern zu können, damit bei Verzögerungen der `M_AXI` Lesezugriffe die CRC-Berechnungskomponente trotzdem ununterbrochen Daten verarbeiten kann.

Die Eingänge *S\_AXIS\_NUM\_AVIL* und *M\_AXIS\_NUM\_FREE* dienen zur Überwachung der Füllstände der FIFO-Speicher. Abhängig davon, ob genug Daten zum Schreiben oder genug freier Speicher zu Verfügung steht, werden die *M\_AXI* Zugriffe gestartet.

Beim Ausgeben des Datenstroms über die *M\_AXIS* Schnittstelle markiert das *M\_AXIS\_TLAST* Signal das letzte Datenwort eines Pakets.

Die Komponente ist bei Schreib- und Lesebursts in der Lage, 4kB-Adressgrenzen zu erkennen und passt die Burstlängen und die Adressen entsprechend des nächsten Zugriffs an.

Die Schreib- und Leseburst sind in zwei separaten Prozessen bzw. endlichen Automaten aufgeteilt, die weitgehend unabhängig voneinander arbeiten. Daher kann die Komponente parallel Daten lesen und schreiben.

Des Weiteren hält diese Komponente alle AXIL-Register des IPs, wofür sie eine *S\_AXIL* Schnittstelle besitzt. Die Register, die zur Konfiguration der CRC-Berechnung dienen, sind als Ausgänge nach Außen geführt.

### 2.2.2 CRC-Berechnungskomponente - axis\_crc

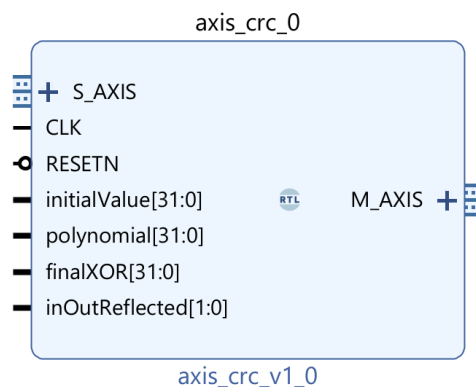


Abbildung 4: Entity der Komponente **axis\_crc**

In Abbildung 4 ist die Entity der Komponente **axis\_crc** dargestellt. Die Eingänge *initialValue*, *polynomial*, *finalXOR* und *inOutReflected* dienen zur Konfiguration der CRC-Berechnung. Sie fließen in die Kombinatoriken der CRC-Berechnung ein.

Die Berechnung der CRC-Prüfsumme erfolgt wort-weise. Der Datenstrom von 32-Bit Worten wird über die *S\_AXIS* Schnittstelle eingelesen und wieder über die *M\_AXIS* Schnittstelle ausgegeben. Dabei kann maximal alle 3 Takte ein Wort eingelesen werden, das in die Berechnung der CRC-Prüfsumme einfließt. Das Ende eines Datenpaketes wird durch das *S\_AXIS\_TLAST* Signal markiert. Wenn die Komponente dieses Signal erhält, beendet sie die CRC-Berechnung und gibt das berechnete CRC-Wort über die *M\_AXIS* Schnittstelle aus. Die CRC-Prüfsumme wird dadurch an den Datenstrom des ursprünglichen Datenpaketes angehängt. Das *LAST*-Signal wird nicht mehr beim letzten Wort des Datenpaketes, sondern bei der CRC-Prüfsumme gesetzt.

Nachdem die CRC-Prüfsumme ausgegeben wurde, wird intern das CRC-Register wieder mit dem Initialwert initialisiert. Die Komponente ist dann wieder bereit, eine neue CRC-Prüfsumme zu berechnen.

Intern erfolgt die Berechnung der CRC-Prüfsumme mithilfe eines endlichen Automaten. Um keine Timingprobleme zu verursachen, wurde so die Berechnung der CRC-Prüfsumme auf drei Takte aufgeteilt.

## 2.3 AXI-CRC-DMA-IP

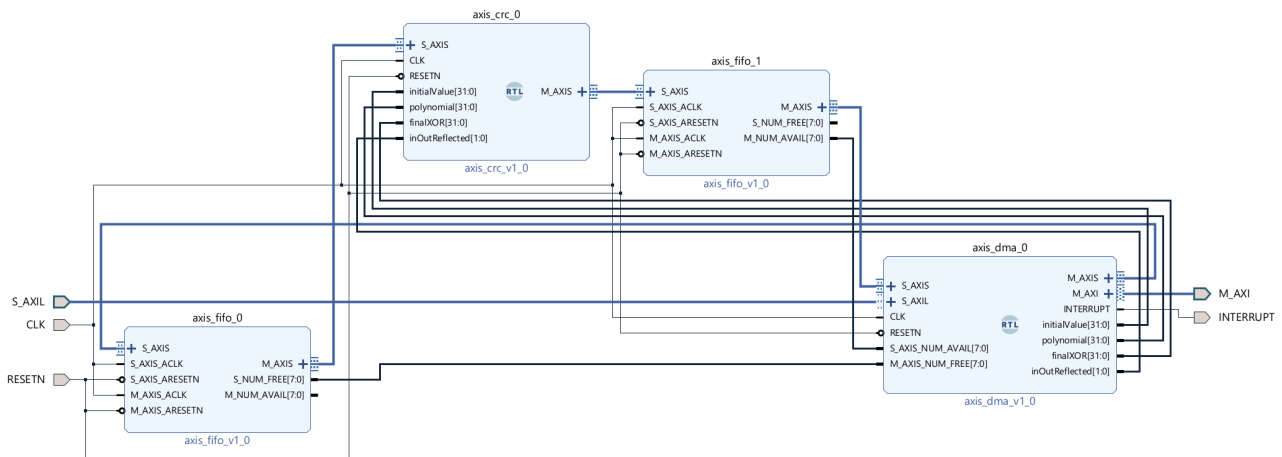


Abbildung 5: Architecture des IP `axi_crc_dma_ip`

In Abbildung 5 ist die Gesamtarchitektur des IPs `axi_crc_dma_ip` im Detail dargestellt. Da die Bestandteile des IPs in den vorherigen Abschnitten beschrieben wurden, erfolgt hier noch eine kurze Erläuterung der Gesamtarchitektur des CRC-DMA-IPs.

Die Komponente `axis_dma` ist für das Lesen und Schreiben der Daten aus dem Speicher zuständig. Sie liest so viele Daten, wie zuvor über die AXIL-Register **Packet Size** und **Number Packets** konfiguriert wurden. Die Daten, die aus dem Speicher gelesen werden, werden als AXI-Stream-Datenstrom an das erste FIFO weitergegeben. Dieses puffert den Datenstrom und gibt ihn an die CRC-Berechnungskomponente `axis_crc` weiter, wo aus dem Datenstrom eine bzw. mehrere CRC-Prüfsummen berechnet werden. Die CRC-Prüfsumme wird an den Datenstrom angehängt und über das zweite FIFO an die DMA-Komponente `axis_dma` zurückgegeben, um sie zurück in den Speicher zu schreiben.

Sobald alle Datenpakete verarbeitet und wieder in den Speicher geschrieben wurden, wird ein Interrupt ausgelöst, um die CPU zu informieren, dass die CRC-Berechnung abgeschlossen ist.



## 2.4 Einbindung in das Gesamtsystem

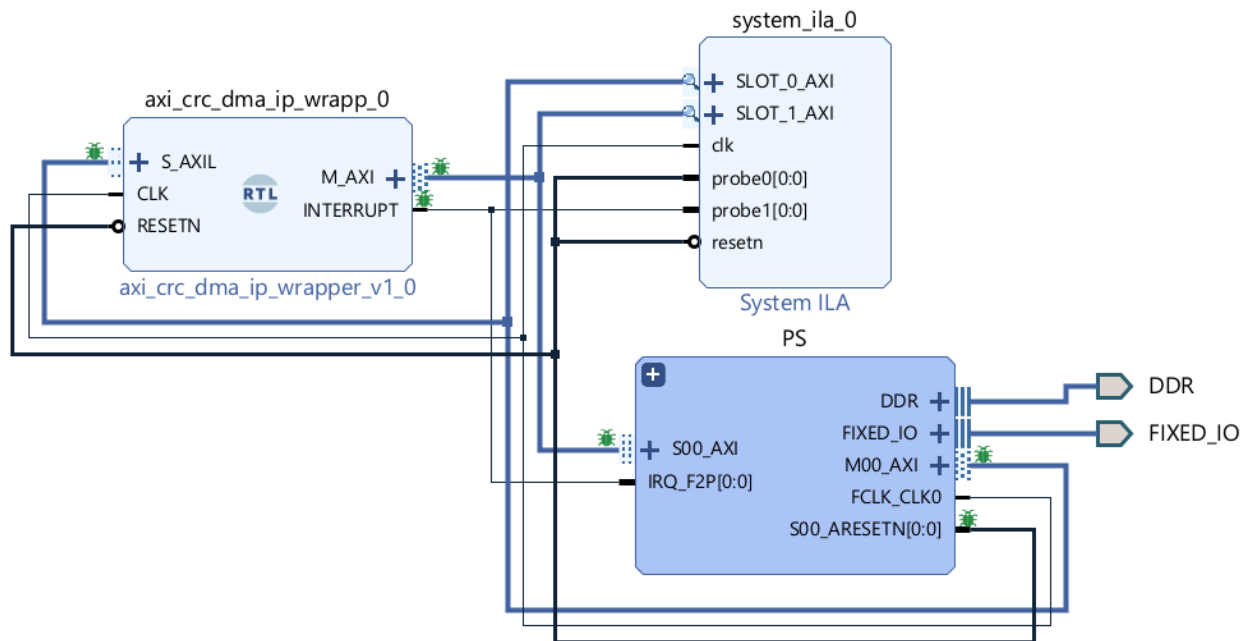


Abbildung 6: Block-Design des Gesamtsystems: *axi\_crc\_dma\_syn\_1*

Um das CRC-DMA-IP in ein Gesamtsystem einzubinden, müssen die AXI-Lite und Full-AXI Schnittstellen mit dem Processing System (PS) des Zynq-SoC verbunden werden. Im hier erstellten Gesamtsystem wurde die Full-AXI-Master Schnittstelle mit der ACP-Schnittstelle verbunden. Da Speicherzugriffe auf das SDRAM mit der ACP-Schnittstelle über den Cache erfolgen, sind die Daten so stets mit der CPU synchronisiert.

Die GP-Master-AXI Schnittstelle ist mit der AXIL-Slave Schnittstelle des IPs verbunden. Damit ist die CPU in der Lage, die Konfigurationsregister des IPs zu schreiben und das IP zu starten.

Die Interruptleitung des IPs ist mit dem Interrupt-Controller des Zynq-SoC verbunden.

Der System ILA dient zur genaueren Betrachtung der Full-AXI-Bursts IPs auf der Hardware. Dies wird in Abschnitt 4.3 genauer beschrieben.

Dieses Gesamtsystem kann nun synthetisiert und implementiert werden.

### 3 Software - Treiberfunktion

```

1  typedef struct
2  {
3      volatile uint32_t Control;           // [0] Run, [1] INT Enable
4      volatile uint32_t InterruptStatus;   // [0] INT Status
5      volatile uint32_t ReadAddress;       // [31:0] Read Address of Data
6      volatile uint32_t WriteAddress;      // [31:0] Write Address of Data
7      + CRC Checksums
8      volatile uint32_t PacketSize;        // [15:0] Size of Packets Minus
9      1 in 32 Bit words
10     volatile uint32_t NumberPackets;     // [15:0] Number of Packets
11     Minus 1
12     volatile uint32_t Polynomial;        // [31:0] Polynomial for CRC
13     Calculation
14     volatile uint32_t InitialValue;       // [31:0] Initial Value of CRC
15     Calculation
16     volatile uint32_t FinalXOR;          // [31:0] Final XOR Value
17     volatile uint32_t InOutReflected;    // [0] Input Reflected, [1]
18     Output Reflected
19     volatile uint32_t AxCache;           // [3:0] M_AXI AWCACHE, [7:4]
20     M_AXI ARCACHE
21 } CRC_DMA_Typedef;
22
23 typedef CRC_DMA_Typedef *PCRC_DMA_Typedef;

```

Codeausschnitt 1: Struct für CRC-DMA-IP

Zur vereinfachten Handhabung des CRC-DMA-IPs wurde ein Struct und eine Treiberfunktion erstellt. In Codeausschnitt 1 ist das Struct für das CRC-DMA-IP dargestellt.

```

1  typedef struct
2  {
3      uint32_t Polynomial;
4      uint32_t InitialValue;
5      uint32_t FinalXOR;
6      bool     InputReflected;
7      bool     OutputReflected;
8  } CrcParameterSet;
9
10 // load a specific set of CRC parameters into Hardware
11 void CRC_DMA_set_parameters(PCRC_DMA_Typedef baseAddr, const
    CrcParameterSet* parameterSet);

```

Codeausschnitt 2: Struct für CRC-Parameter

Für einen einfacheren Umgang mit den CRC-Parametern befindet sich zusätzlich ein Struct für einen Satz an CRC-Parametern. Diese Struct ist in Codeausschnitt 2 dargestellt. Zum einfachen Setzen der CRC-Parameter im IP kann die Funktion `void CRC_DMA_set_parameters(...)` genutzt werden.

Beide gezeigten Codeausschnitte sind in der Headerdatei `axi_crc_dma.h` zu finden.

## 4 Testkonzept und Ergebnisse

### 4.1 Verifikation der CRC-Berechnungskomponente

Die Korrektheit der von der CRC-Berechnungskomponente berechneten CRC-Prüfsumme ist für das Gesamtsystem von essentieller Bedeutung. Daher liegt ein besonderes Augenmerk auf der Verifikation dieser Komponente.

#### 4.1.1 Referenzprogramm

Zunächst wurde das C-Programm *crc.c* geschrieben, das CRC-Prüfsummen in Software berechnet. Dieses Programm dient bei der Entwicklung der Hardware als Referenz für die berechneten CRC-Prüfsummen. Der CRC-Algorithmus wurde hauptsächlich mithilfe von Sunshine2k.de [3] erstellt. Parallel wurden die Ergebnisse mit anderen CRC-Online Rechnern verglichen, um die Korrektheit der Berechnung zu überprüfen [1].

Kern dieses Programms ist die Funktion `uint32_t calcCRC32(...)`, die die CRC-Prüfsumme über ein Datenpaket berechnet. Mithilfe dieser Funktion können CRC-Prüfsummen für beliebige Datenpakete und für eine beliebige Kombination an CRC-Parametern berechnet werden.

Neben dem testweise Berechnen von Prüfsummen werden in der `main`-Funktion auch die CRC-Prüfsummen für die Testbench der CRC-Berechnungskomponente berechnet. Dies geschieht in der Funktion `void calc_axis_crc_tb()`.

#### 4.1.2 Simulation / Testbench

Um die Korrektheit der CRC-Berechnungskomponente *axis\_crc* zu gewährleisten, wurde eine Testbench erstellt. In dieser Testbench werden von der CRC-Berechnungskomponente insgesamt 24 Prüfsummen über dasselbe Datenpaket von drei 32-Bit-Worten mit verschiedenen Parametern berechnet. Da auch nur ein kleinster Fehler in der Implementierung des CRC-Algorithmus völlig andere Ergebnisse zur Folge hätte, genügt es zunächst nur ein Datenpaket für die Verifikation zu nutzen.

Die Testbench wurde in VHDL geschrieben und überprüft automatisch die Korrektheit der berechneten CRC-Prüfsummen sowie das korrekte Ausgeben der Daten.

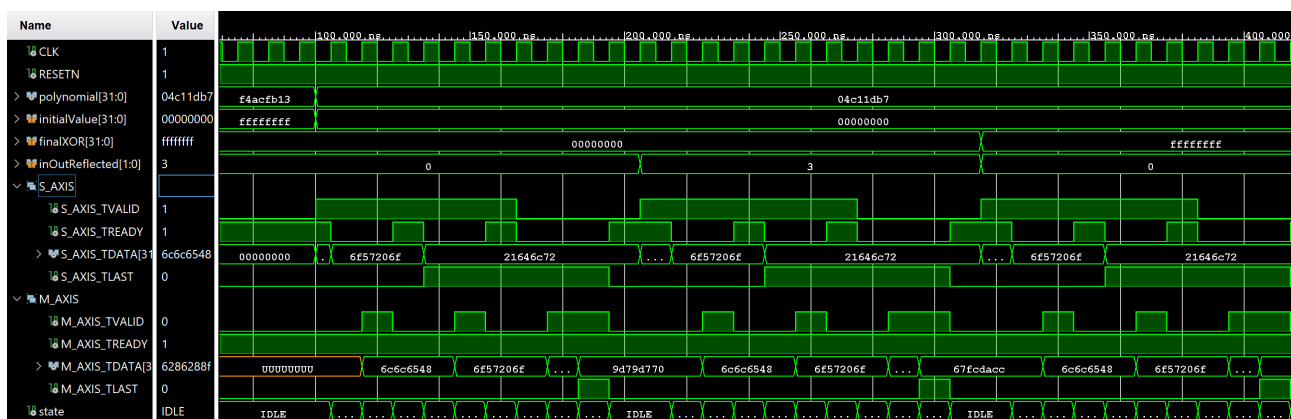


Abbildung 7: Wave-Diagramm der *axis\_crc* Testbench

In Abbildung 7 ist ein Ausschnitt des Wave-Diagramms der Testbench dargestellt. Falls einer der 24 Testfälle fehlschlägt, das heißt, wenn entweder die zugeführten Daten nicht wieder korrekt ausgegeben

werden oder wenn die CRC-Prüfsumme falsch ist, wird eine Fehlermeldung ausgegeben. Somit kann die korrekte Funktionalität der CRC-Berechnungskomponente gewährleistet werden.

Die in dieser Arbeit erstellte CRC-Berechnungskomponente wurde mithilfe dieser Testbench erfolgreich verifiziert.

## 4.2 Simulation des CRC-DMA-IPs

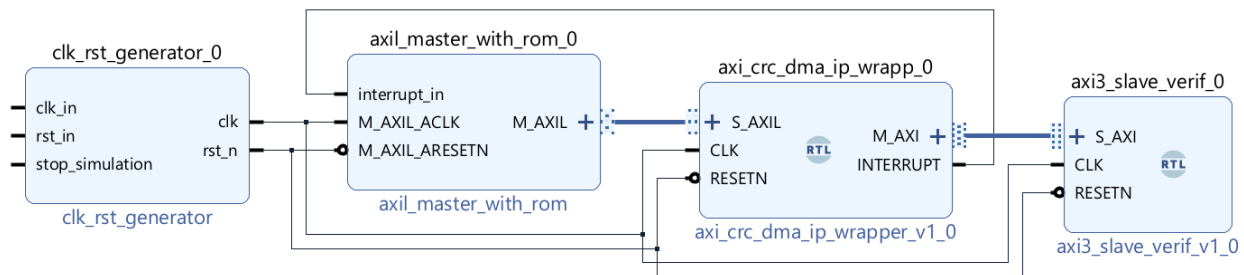


Abbildung 8: Simulation des CRC-DMA-IPs

Zur Simulation des gesamten CRC-DMA-IPs wird das Block-Design *axi\_crc\_dma\_sim\_01* verwendet, das in Abbildung 8 dargestellt ist. Das IP *axil\_master\_with\_rom* ist ein AXIL-Master, der die nötige Konfiguration des CRC-DMA-IPs durchführt und den Vorgang anschließend startet. Er stammt aus dem IP-Verzeichnis dieser Vorlesung. Die AXIL-Schreibbefehle für den AXIL-Master sind im Stimuli-Script *axi\_crc\_dma\_sim.stm* definiert.

Das IP *axi3\_slave\_verif* ist ein Baustein, der als Full-AXI-Slave fungiert. Er akzeptiert Schreibbursts und gibt bei Lesebursts selbst erzeugte Daten wieder. Dabei hat diese Komponente jedoch keinen internen Speicher, sondern dient nur dazu, korrekte Reaktionen auf Lese- und Schreibbursts zu erzeugen. Die Adressen und die eigentlichen Daten werden von dieser Komponente nicht verarbeitet. Das Grundgerüst dieser Komponente wurde mithilfe von ChatGPT erstellt und anschließend in der Simulation etwas angepasst, sodass das Verhalten möglichst dem korrekten Verhalten einer Full-AXI-Slave Schnittstelle entspricht.

In diesem Simulations-Block-Design wird zum einen die Funktionsweise der AXIL-Schnittstelle überprüft. Das heißt, dass die AXIL-Register richtig geschrieben und gelesen werden können und dass der CRC-DMA-Vorgang über das Kontrollregister gestartet werden kann. Zum anderen wird vornehmlich das richtige DMA-Verhalten des IPs überprüft. Dabei sind unter anderem folgende Punkte von Bedeutung:

- Werden die Valid/Ready-Handshakes korrekt durchgeführt?
- Werden die korrekten Adressen übergeben?
- Werden die korrekten Daten übergeben?
- Wird die korrekte Menge an Daten entsprechend der Konfiguration gelesen?
- Wird die korrekte Menge an Daten entsprechend der Konfiguration geschrieben?
- Werden stets maximal mögliche Burstlängen verwendet?
- Werden 4kB-Adressgrenzen korrekt behandelt?

- Wird die Interruptleitung nach dem Vorgang korrekt gesetzt?

Da es sich um eine manuelle Testbench handelt, werden all diese Punkte durch das Betrachten des Wave-Diagramms überprüft. Zusätzlich wurden dabei verschiedene Konfigurationen des IPs getestet, um auch die Funktionalität bei unterschiedlichen Parametern zu überprüfen und Edge-Cases mit abzudecken.

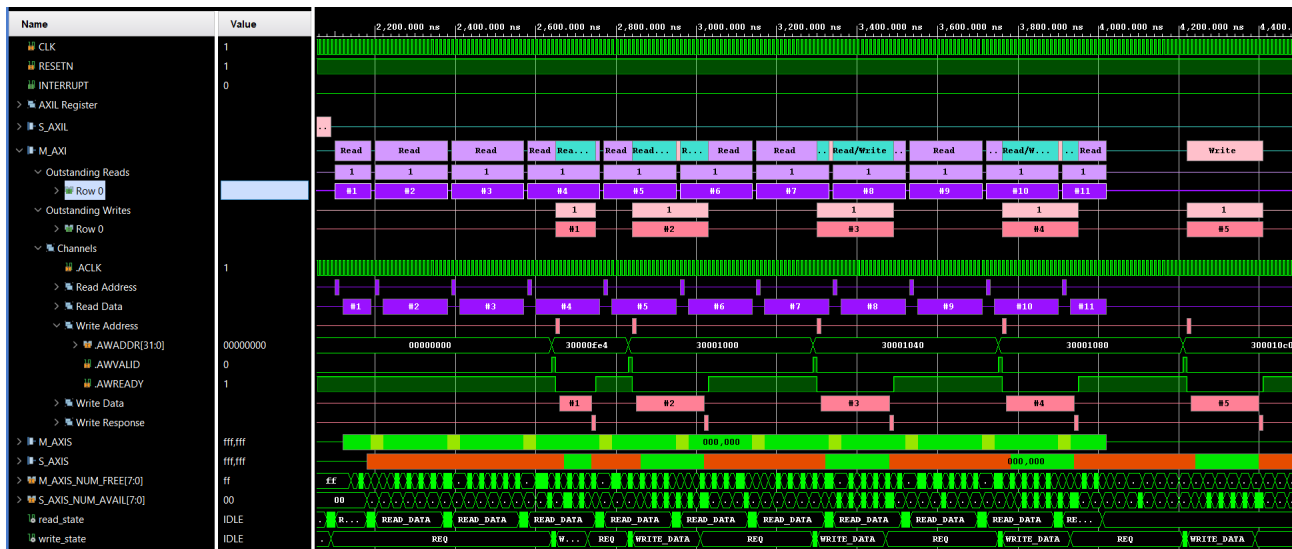


Abbildung 9: Exemplarisches Wave-Diagramm der Simulation des CRC-DMA-IPs

Abbildung 9 zeigt exemplarisch das Wave-Diagramm der Simulation. Dort ist zu sehen, dass das IP keine Schreibbursts über die 4kB-Grenze hinaus ausführt, sondern die Burstlänge entsprechend anpasst.

## 4.3 Verifikation des Gesamtsystems

### 4.3.1 Bare-Metal-Programm

Zum Testen des Gesamtsystems wurde zunächst das Programm `axi_crc_dma_bare_metal.c` erstellt, das auf dem Mikroprozessor ohne Betriebssystem, also bare-metal, ausgeführt wird. In diesem Testprogramm wird ein einfacher Anwendungsfall für das CRC-DMA-IP durchgeführt. Das beinhaltet das anlegen von Datenpaketen im Speicher, das Konfigurieren des CRC-DMA-IPs und das Starten des Vorgangs.

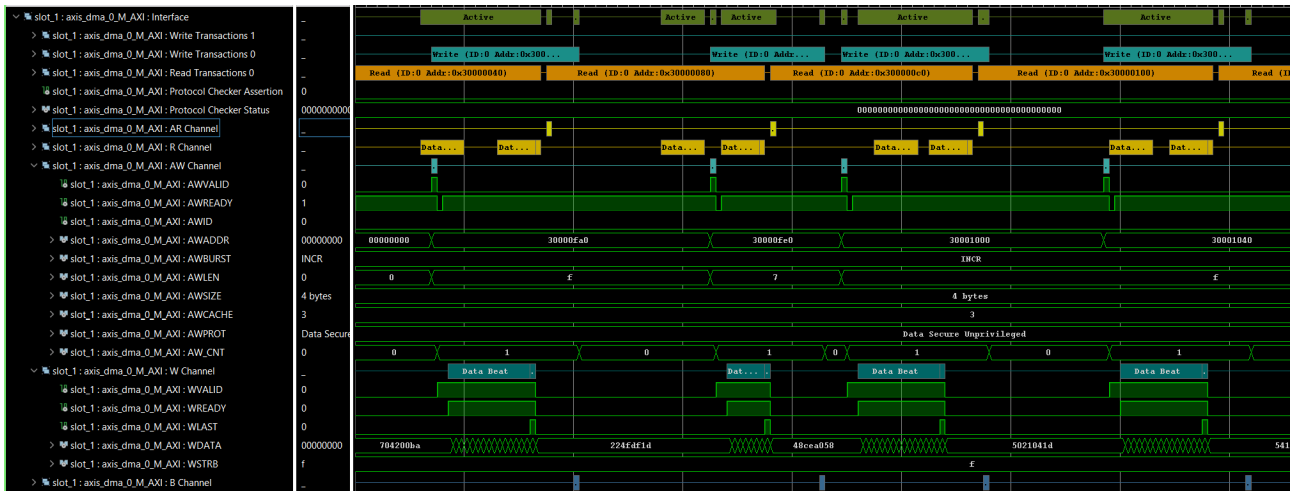


Abbildung 10: ILA-Messung der Full-AXI-Master Schnittstelle

Dieses Programm wurde vor allem dafür verwendet, mithilfe von ILA-Messungen die Funktionalität der Full-AXI-Master Schnittstelle zu überprüfen. Eine Beispielmessung ist in Abbildung 10 dargestellt. So konnte final die korrekte Kommunikation des IPs mit dem Restsystem festgestellt werden.

#### 4.3.2 Testprogramm

Für die finale Verifikation des CRC-DMA-IPs bzw. des Gesamtsystems wurde das Testprogramm `CRC_DMA.cpp` erstellt, das im Kontext eines Linux Betriebssystems ausgeführt wird. Das Testprogramm ist in mehrere Testläufe unterteilt. Der Ablauf eines Testlaufs ist wie folgt:

1. Auswählen eines neuen CRC-Parameter-Satzes
2. Zufällige Datenpakete erzeugen mit einer zufälligen Paketanzahl und -größe
3. Konfiguration des CRC-DMA-IPs
4. Starten des CRC-DMA-IPs
5. Warten auf den Interrupt des IPs
6. Berechnen der CRC-Prüfsummen in Software zur Verifikation
7. Überprüfen der vom IP geschriebenen Daten und CRC-Prüfsummen

Die Anzahl der Testläufe wird dem Programm als Argument beim Ausführen übergeben. Die Ausgabe des Programms erfolgt sowohl über die Konsole als auch in eine Logdatei, die automatisch erstellt wird.

```
----- Starten von Testdurchlauf 10 -----  
  
Paketgroesse: 280      Paketanzahl: 5  
  
Testdaten erzeugen  
CRC-Berechnung in Hardware starten  
Auf Interrupt warten...  
Interrupt erhalten  
CRC-Berechnung in Software durchfuehren  
Daten und Ergebnisse vergleichen:  
Paket 0:      Daten OK      CRC OK   CRC: 0x502af233  
Paket 1:      Daten OK      CRC OK   CRC: 0xbe97046e  
Paket 2:      Daten OK      CRC OK   CRC: 0x3f55df74  
Paket 3:      Daten OK      CRC OK   CRC: 0x10a70e76  
Paket 4:      Daten OK      CRC OK   CRC: 0xc8c17c04  
Alle Pakete OK  
Testlauf erfolgreich abgeschlossen!  
Gelesene Datenmenge in diesem Testlauf: 5600 Bytes  
  
Alle Testlaeufe erfolgreich abgeschlossen!  
Insgesamt gelesene Datenmenge: 105960 Bytes
```

Abbildung 11: Ausgabe des Testprogramms

In Abbildung 11 ist exemplarisch die Ausgabe des Testprogramms für einen Testlauf dargestellt. Zur Verifikation des CRC-DMA-IPs können nun mithilfe dieses Programms im großen Umfang CRC-Berechnungsvorgänge ausgeführt werden. Durch eine große Anzahl an Testläufen kann so sichergestellt werden, dass, wenn das IP fehlerhaft ist, diese Fehler auch entdeckt werden.

Anbei dieser Projektarbeit befindet sich eine Logdatei des Testprogramms, die die Ausgabe für 250 Testläufe enthält. Ergebnis dieses Tests ist, dass bei einer gelesenen Menge von ca. 1,4 GB Daten keine Fehler aufgetreten sind. Alle Daten wurden korrekt in den Speicher geschrieben.

Dabei ist anzumerken, dass nur bis zu einer Paketanzahl von 100 und einer Paketgröße von 10000 getestet wurde, auch wenn technisch gesehen sowohl Paketgröße als auch Paketanzahl auf einen Wert bis zu 65535 konfiguriert werden können.

## 5 Fazit und Ausblick

Es ist gelungen, ein CRC-DMA-IP gemäß den in Abschnitt 1.2 definierten Anforderungen zu erstellen und umfangreich mit einem Testprogramm zu testen.

Eine mögliche Verbesserung des IPs wäre die Optimierung der CRC-Berechnungskomponente. Diese ist aktuell der limitierende Faktor für den Datendurchsatz des IPs. Zum Beispiel könnte die CRC-Berechnung in mehrere Pipeline-Stufen aufgeteilt werden, wodurch der Datendurchsatz theoretisch auf ein 32-Bit-Wort pro Takt erhöht werden kann. Damit wäre der Datendurchsatz verdreifacht.

Des weiteren könnte eine auf ein Byte genaue Adressierung der Daten implementiert werden. Damit könnten Datenpakete auch innerhalb eines 32-Bit-Wortes beginnen und/oder enden.

Eine weitere Verbesserungsmöglichkeit wäre die Implementierung einer Stoppfunktion. Wenn ein Vorgang gestartet wurde, kann dieser aktuell nicht vorzeitig abgebrochen werden. Dies führt unter Umständen zu unnötigen Wartezeiten.

## 6 Anhang

### 6.1 Hardware

Jegliche Hardware Quelldateien sind im Vivadoprojekt `axi_crc_dma` zu finden. Dieses beinhaltet folgende VHDL-Quelldateien:

- `axis_crc.vhd`: CRC-Berechnungskomponente siehe Abschnitt 2.2.2
- `axis_crc_tb.vhd`: Testbench für die CRC-Berechnungskomponente siehe Abschnitt 4.1.2
- `axis_dma.vhd`: DMA-Komponente siehe Abschnitt 2.2.1
- `axi3_slave_verif.vhd`: Full-AXI-Slave zur Verifikation des Gesamtsystems siehe Abschnitt 4.2

Dazu beinhaltet das Vivadoprojekt folgende Block-Designs:

- `axi_crc_dma_ip`: Block-Design des CRC-DMA-IPs siehe Abschnitt 2.3
- `axi_crc_dma_sim_01`: Block-Design zur Simulation des CRC-DMA-IPs siehe Abschnitt 4.2
- `axi_crc_dma_syn_01`: Block-Design zur Synthese des Gesamtsystems mit CRC-DMA-IP siehe Abschnitt 2.4

### 6.2 Software

Folgende Programme bzw. Software-Quelldateien wurden erstellt:

- `axi_crc_dma.h`: Headerdatei für das CRC-DMA-IP siehe Abschnitt 3
- `axi_crc_dma.cpp`: Treiberfunktion für das CRC-DMA-IP siehe Abschnitt 3
- `crc.c`: Referenzprogramm zur Berechnung von CRC-Prüfsummen in Software siehe Abschnitt 4.1.1
- `axi_crc_dma_bare_metal.c`: Testprogramm für das CRC-DMA-IP siehe Abschnitt 4.3.1
- `CRC_DMA.cpp`: Testprogramm zur Verifikation des Gesamtsystems siehe Abschnitt 4.3.2



## 7 Quellen

- [1] *CRC Online Calculator*. URL: <https://crccalc.com/> (besucht am 13.02.2025).
- [2] *Digilent Reference - Zybo Z7*. URL: <https://digilent.com/reference/programmable-logic/zybo-z7/start> (besucht am 14.02.2025).
- [3] Sunshine2k. *Understanding and implementing CRC (Cyclic Redundancy Check) calculation*. URL: [https://www.sunshine2k.de/articles/coding/crc/understanding\\_crc.html](https://www.sunshine2k.de/articles/coding/crc/understanding_crc.html) (besucht am 13.02.2025).

## Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst und keine anderen Hilfsmittel als die angegebenen verwendet habe.

Insbesondere versichere ich, dass ich alle wörtlichen und sinngemäßen Übernahmen aus anderen Werken als solche kenntlich gemacht habe.

Ort, Datum: \_\_\_\_\_

Ort, Datum: \_\_\_\_\_

Unterschrift: \_\_\_\_\_

Unterschrift: \_\_\_\_\_