Fachhochschule Dortmund
University of Applied Sciences and Arts

Cologne Chip

FPGA Implementierung einer SRAM basierten Physically Unclonable Function

Bachelorthesis

im Studiengang „Elektrotechnik Automation & Antriebe“

vorgelegt von

Idriss Karmadi

am 15.07.2022

an der Fachhochschule Dortmund

Erstprüfer: Herr Prof. Dr. Karagounis

Zweitprüfer: Herr Felix Schneider

Abstract

Die vorliegende Arbeit befasst sich mit der FPGA Implementierung einer SRAM basierten Physically Unclonable Function, welche unter Verwendung der Synthesesoftware Yosys umgesetzt werden soll. Nach einer notwendigen Einführung in das GateMate FPGA 1A1 wird ein umfassender Überblick auf ein Block RAM (BRAM) gegeben. Basierend auf der VHDL Sprache wird ein Modul erstellt, das aus verschiedenen Untermodulen besteht, um die Daten des BRAMs über eine serielle Schnittstelle zu transferieren. Als Ergebnis werden die Daten ausgelesen und ausgewertet.

Abstract

This thesis deals with the FPGA implementation of an SRAM-based Physically Unclonable Function, which is to be implemented using the synthesis software Yosys. After a necessary introduction to the GateMate FPGA 1A1, a comprehensive overview of a block RAM is given. Based on the VHDL language a module is designed consisting of different sub-modules to transfer the data bits through a serial port. As a result, the data is read out and evaluated.

Inhalt

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

FH: Fachhochschule Dortmund
PUF: Physically Unclonable Function
FPGA: Field Programmable Gate Array
GPIO: General Purpose Input/Output
BRAM: Block Random Access Memory
Pmod: Peripheral Modul
VHDL: Very High Hardware Description Language
SPI: Serial Peripheral Interfac
USB: Universal Serial Bus
UART: Universal Asynchronous Receiver Transmitter
DC: direct current
AC: alternating current
PCB: Power Supply
SLP: Super Low Power
SRAM: Static random-access memory
DRAM: Dynamic Random Access Memory
CPE: combining programmable element

1. Einleitung

Die vorliegende Arbeit befasst sich mit der GateMate 1A1 FPGA Implementierung einer SRAM basierten Physically Unclonable Function. SRAM ist ein statisches Random Access Memory, das Datenbits im Speicher hält, solange die Versorgungsspannung anliegt. Es gewährleistet einen schnelleren Zugriff auf Daten als ein Dynamisches Random Access Memory (DRAM).

Das GateMate™ FPGA ist eine Familie von kleinen bis mittelgroßen FPGAs, die alle Anforderungen von typischen Anwendungen abdeckt. Das programmierte Silizium kann mit geringem Stromverbrauch bis hin zu Hochgeschwindigkeitsanwendungen verwendet und damit in einem breiten Spektrum von Bereichen wie Industrie, Automatisierung, Kommunikation, Sicherheit, Automotive, Internet of Things (IoT), Künstliche Intelligenz (KI) und vieles mehr eingesetzt werden. Dazu kombinieren diese FPGAs verschiedene Funktionen mit den niedrigsten Kosten auf dem Markt. Das GateMate FPGA-Programm wird vom deutschen Bundesministerium für Wirtschaft und Energie im Rahmen des Projekts Important Project of Common European Interest on Microelectronics von der Europäischen Kommission gefördert. Die Gatemate FPGA Familie wird von der Firma Cologne Chip AG mit Sitz in Köln vermarktet. Das Unternehmen wurde im Jahr 1996 gegründet und verfügt über exzellente Branchenkenntnisse und ein erfahrenes Team von Entwicklern. Der Design- und Fertigungsstandort **Made in Germany** stellt ein Alleinstellungsmerkmal der weltweit wettbewerbsfähigen FPGAs auf dem Markt dar. In dieser Thesis wird zunächst der Begriff der Physically Unclonable Function geklärt und das Funktionsprinzip SRAM basierter Physically Unclonable Function vorgestellt. Anschließend wird die verwendete Hardware insbesondere das FPGA und benötigte Zusatzmodule beschrieben. Beim Entwurf von digitalen Schaltungen spielt auch die Software zur Electronic Design Automation (EDA) eine wichtige Rolle. Eine Besonderheit in diesem Projekt ist die Verwendung der Open-Source Synthese Software Yosys, welche zusammen mit einer proprietären Software zum Place & Route für die Implementierung von digitalen Schaltungen auf dem Gatemate FPGA genutzt wird. Daraufhin wird der Schaltungsentwurf in VHDL vorgestellt, der die Auslesung von Daten aus einem BRAM des FPGAs und die Übertragung dieser Daten über eine UART Schnittstelle zur Aufgabe hat. Schließlich werden die Ergebnisse der Versuche mit und ohne vorinitialisiertem BRAM vorgestellt.

2. Aufgabenbeschreibung

Ziel ist die Implementierung einer SRAM basierten Physically Unclonable Function in einem FPGA das angesteuert, ausgelesen und zu Testzwecken initialisiert wird. Genutzt wird das GateMate™ FPGA der Firma Chip Cologne, welches auf einem Evaluationsboard in Betrieb genommen wird. Die aus dem BRAM ausgelesenen Daten werden einzeln, d.h. byteweise übertragen. Dafür wird ein USB Modul der Firma FTDI genutzt, welches über eine UART Schnittstelle verfügt. Das FTDI Modul ist auf einer kleinen Platine integriert, die über einen PMOD Stecker an den FPGA angeschlossen werden kann. Außerdem wurde eine weitere Platine mit Tastern verwendet, die ebenfalls über PMOD Stecker mit dem FPGA verbunden werden kann.

Zur Umsetzung dieser Implementierung werden Programme wie Yosys, Ghdl, und openFPGALoader unter Ubuntu (Linux) eingesetzt. Außerdem wurden die Hardware Module mithilfe der VHDL-Sprache entworfen, simuliert und auf dem FPGA implementiert.

3. PUFs

Der Begriff Physical Unclonable Function (PUF) lässt sich nur schwer in einer einzigen geschlossenen Definition erfassen. Eine PUF führt eine funktionelle Operation durch. Diese funktionelle Operation erzeugt eine messbare Ausgabe, wenn sie mit einer bestimmten Eingabe abgefragt wird. In den meisten Fällen ist eine PUF keine echte Funktion im mathematischen Sinne, da eine Eingabe in eine PUF mehr als eine mögliche Ausgabe haben kann. Es ist angemessener, eine PUF als eine Funktion im technischen Sinne zu betrachten, d. h. als eine Prozedur, die von einem bestimmten physikalischen System ausgeführt wird oder auf dieses System einwirkt. Im Zusammenhang mit PUFs wird die Eingabe als Challenge und die Ausgabe als Response bezeichnet. Eine angewandte Anforderung und die gemessene Antwort wird im Allgemeinen als Challenge-Response-Paar oder CRP, und die von einer bestimmten PUF erzwungene Beziehung zwischen Anforderung und als ihr CRP-Verhalten bezeichnet [1]. Bei der Charakterisierung von PUFs werden die folgenden beiden Metriken verwendet.

- Für eine bestimmte Eingabe ist die Inter-Distanz zwischen zwei verschiedenen PUF Instanziierungen der Abstand zwischen den beiden Antworten, die sich aus der einmaligen Anwendung dieser Eingabe auf beide PUFs ergeben.
- Für eine bestimmte Eingabe ist die Intra-Distanz zwischen zwei Bewertungen auf einer einzigen PUF-Instanziierung die Distanz zwischen den beiden Antworten, die sich aus der zweimaligen Anwendung dieser Eingabe auf eine PUF ergeben.

Die erste Metrik beschreibt die Einzigartigkeit des generierten Bitmusters. Die Einzigartigkeit des Bitmusters setzt voraus, dass nur zufällige Fertigungsschwankungen Schlüsselmerkmale bestimmen,

während der Einfluss systematischer Ursachen ausgeschlossen werden kann. Zur quantitativen Bewertung der Einzigartigkeit der generierten Schlüssel wird die Inter-Hamming Distanz herangezogen, die wie folgt definiert ist:

$$D_{P_i,P_j}^{inter} = \frac{2}{k(k-1)} \sum_{j=1}^{k-1} \sum_{j=i+1}^{k} \frac{HD(P_i, P_j)}{n} \cdot 100\%$$

Die Formelgröße n ist dabei die Bitlänge des Schlüssels und k die Anzahl der Primitive, welche miteinander verglichen werden. Die Funktion HD bezeichnet die Hamming-Distanz zwischen den Schlüsseln, welche von den PUV Primitiven Pi bzw. P_j generiert werden und der Anzahl der Bitstellen entspricht, an denen sich zwei Codewörter voneinander unterscheiden. Im Idealfall ist die Inter-Hamming Distanz binomial verteilt und besitzt einen Erwartungswert von $E(Di^{inter}P_i;P_j) = 50\%$.

Die zweite Metrik beschreibt die Reproduzierbarkeit des Bitmusters, die besonders wichtig ist, weil bei der Verwendung von PUF Zellen das Bitmuster bei Bedarf immer wieder neu generiert wird. Zur Bewertung der Reproduzierbarkeit der generierten Schlüssel wird die Intra-Hamming Distanz herangezogen, die wie folgt definiert ist:

$$D_{P_i,P_i'}^{intra} = \frac{1}{k} \sum_{j=i+1}^{k} \frac{HD(P_i, P_i')}{n} \cdot 100\%$$

Die Formelgröße n ist dabei wiederum die Bitlänge des Schlüssels und k die Anzahl der Vergleiche, welche zwischen dem Schlüssel Pi und dem zu einem späteren Zeitpunkt aber von der gleichen Zelle generierten Schlüssel P'i durchgeführt wurde. Im Idealfall besitzt die Intra-Hamming Distanz den Wert $D^{intra}\ P_i;P'_i = 0$, was bedeutet, dass einmal generierte Schlüssel jederzeit reproduziert, werden können.

3.1 Digitale intrinsische physikalisch nicht klonierbare Funktionen

PUFs gelten als Alternative zu On-Chip Speichern für Schlüssel, die in der Kryptographie eingesetzt werden. Kryptographischen Verfahren wie z.B. RSA, SHA und AES werden verwendet, um die Firmware von eingebetteten Systemen vor unauthorisiertem Zugriff und Manipulation zu schützen. Die Sicherheit des Systems wird erhöht, wenn wie im ISO/IEC 11889 Standard spezifiziert, der Schlüssel in einem nichtflüchtigen Speicher On-Chip abgelegt ist und keine I/O Schnittstellen genutzt werden, um den Schlüssel aus externen Speicherbausteinen einzulesen. Bei der Nutzung von PUV Primitiven wird der Schlüssel nicht gespeichert, sondern das Bitmuster immer wieder bei Bedarf neu generiert. Dadurch können teure Produktionsschritte für integrierte ROM, EEPROM oder Flash-Module eingespart werden. PUF Primitive nutzen dabei stochastische Prozessvariationen, die sich inhärent bei der Produktion von Halbleiterbauelementen ergeben. Obwohl moderne CMOS Technologien einer strengen

Prozesskontrolle unterliegen, können Hersteller zufällige Variationen auf der mikroskopischen Ebene weder verhindern noch beeinflussen, was beispielsweise zu lokalen Änderungen der Oxiddicken und Dotierungskonzentrationen der Bauelemente führt. Für ein hohes Maß in Sicherheit ist die Verwendung von intrinsischen PUFs anzustreben. Hierbei werden zwei Voraussetzungen unterschieden, damit ein PUF als intrinsisch bezeichnet werden kann:

1. Die PUF, einschließlich aller notwendigen Messeinrichtungen, sollten vollständig in der Einbettungsvorrichtung z.B. einem Mikrochips integriert sein.

2. Die vollständige PUF-Konstruktion sollte aus Primitiven bestehen, die für den Herstellungsprozess des Einbettungsgeräts natürlich verfügbar sind.

Die erste Bedingung bedeutet, dass das Gerät seine eigene PUF abfragen und auslesen kann, ohne dass externe Instrumente erforderlich sind, und ohne dass die Anfrage und die Antwort das Gerät verlassen müssen. Die zweite Bedingung besagt, dass die komplette PUF- Konstruktion außer dem vom PUF belegten Platz praktisch keine zusätzlichen Kosten verursacht, d. h. es sind keine zusätzlichen Fertigungsschritte oder speziellen Komponenten erforderlich. Diese Bedingung wird beispielsweise von Beschichtungs-PUF und integrierten optischen PUFs nicht erfüllt, da beide hochspezialisierte Verarbeitungsschritte erfordern.[2]

3.2 Speicherbasierte intrinsische PUFs

Im Rahmen dieser Thesis wird eine Art von intrinsischer PUFs erörtert, die auf dem Einschwingzustand digitaler Speicherprimitive basiert. Eine digitale Speicherzelle ist typischerweise eine digitale Schaltung mit mehr als einem logischen stabilen Zustand. Wenn sie sich in einem ihrer stabilen Zustände befindet, kann sie Informationen speichern, z. B. eine Binärziffer im Fall von zwei möglichen stabilen Zuständen. Wenn das Element jedoch in einen instabilen Zustand gebracht wird, ist nicht klar, was passiert. Es könnte anfangen, zwischen instabilen Zuständen zu oszillieren, oder es könnte wieder in einen seiner stabilen Zustände konvergieren. Im letzteren Fall ist zu beobachten, dass bestimmte Zellen bestimmte stabile Zustände gegenüber anderen stark bevorzugen. Dieser Effekt kann oft nicht durch die logische Implementierung der Zelle erklärt werden. Stattdessen liegt eine interne physikalische Fehlanpassung vor, die z. B. durch Fertigungsschwankungen verursacht wird. Aus diesem Grund ist der stabile Einschwingzustand einer destabilisierten Speicherzelle ein guter Kandidat für eine PUF-Antwort.[3]

3.2.1 SRAM PUFs

SRAM-PUFs wurden in [5] vorgeschlagen und ein sehr ähnliches Konzept wurde in [6] zeitgleich vorgestellt. SRAM oder statischer Direktzugriffsspeicher ist eine Art digitaler, aus Zellen bestehender Speicher, wobei jede Zelle jeweils eine Binärziffer speichern kann. Eine SRAM-Zelle, wie in Abb. 1 gezeigt, ist aus zwei kreuzgekoppelten Invertern aufgebaut, was zu zwei stabilen Zuständen führt. In normaler CMOS-Technologie wird diese Schaltung mit 4 MOSFETs implementiert, wobei zwei zusätzliche MOSFETs ,wie in Abbildung 2 gezeigt, für den Lese- und Schreibzugriff verwendet werden. Aus diesen Gründen wird die physikalische Diskrepanz zwischen den beiden symmetrischen Hälften der Schaltung, die jeweils einen Inverter implementieren, so gering wie möglich gehalten. Aus der logischen Beschreibung der Zellen geht nicht hervor, in welchem Zustand sich der Speicher direkt nach dem Einschalten befindet, d. h. was passiert, wenn die Versorgungsspannung angelegt wird. Es ist zu beobachten, dass einige Zellen beim Einschalten vorzugsweise eine Null speichern, andere vorzugsweise eine Eins, und einige Zellen haben keine wirkliche Präferenz, aber die Verteilung dieser drei Arten von Zellen über den gesamten Speicher ist zufällig. Wie sich herausstellt, bestimmt die zufällige physikalische Fehlanpassung in der Zelle, welche durch Fertigungsschwankungen verursacht wird, das Einschaltverhalten. Sie zwingt eine Zelle je nach Vorzeichen der Fehlanpassung während des Einschaltens auf Null oder Eins. Ist die Fehlanpassung sehr gering, wird der Einschaltzustand durch stochastisches Rauschen in der Schaltung bestimmt und ist zufällig, ohne dass eine echte Präferenz vorliegt.

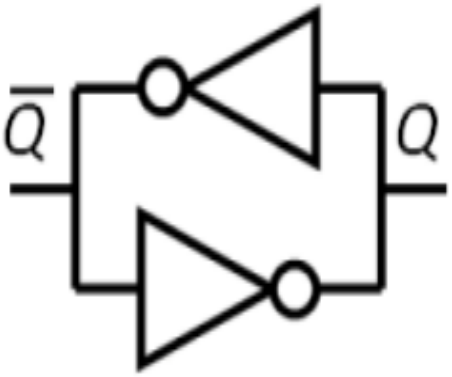


Abbildung 1: Logische Schaltung einer SRAM PUF-Zelle [20]

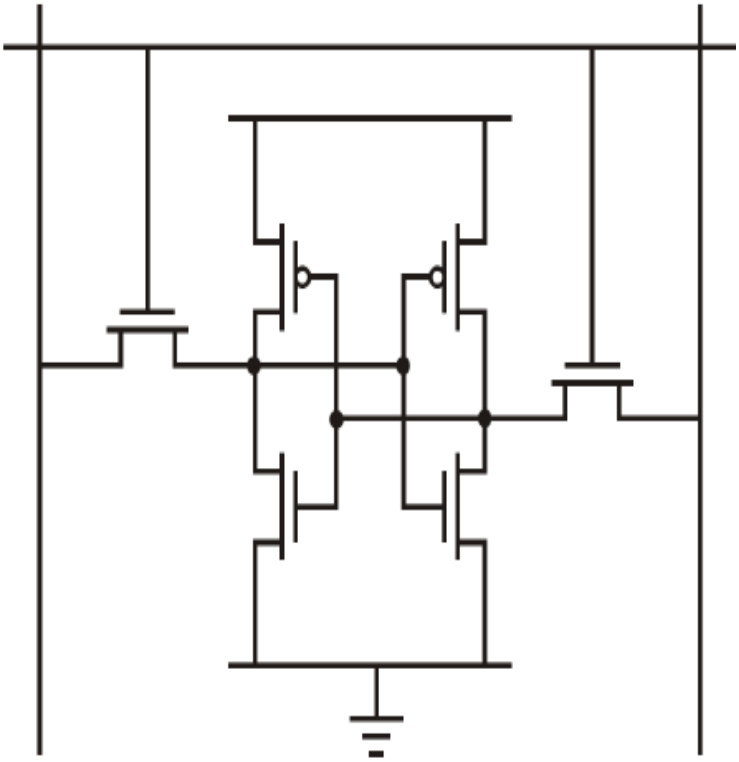


Abbildung 2: Elektrische Schaltung einer SRAM (PUF)-Zelle in Standard-CMOS-Technologie. [20]

In [5] wurden umfangreiche Experimente mit SRAM-PUFs durchgeführt. Sie erfassten den Einschaltzustand von 8190 Byte SRAM aus verschiedenen Speicherblöcken auf verschiedenen FPGAs. Die Ergebnisse zeigen eine durchschnittliche Inter-Distanz zwischen zwei verschiedenen Blöcken von $\mathbf{D}^{\mathbf{inter}}$ = 49.97% und die durchschnittliche Intra-Distanz innerhalb mehrerer Messungen eines einzelnen Blocks beträgt $\mathbf{D}^{\mathbf{intra}}$ = 3,57% für eine feste Umgebung und $\mathbf{D}^{\mathbf{intra}}$ < 12% für große Temperaturabweichungen. In [7] schätzen die Autoren den Entropiegehalt der SRAM-Einschaltzustände auf 0,76 Bit pro SRAM-Zelle. In [6, 8] wurde das Einschaltverhalten von SRAM auf zwei verschiedenen Plattformen untersucht. Für 5120 Blöcke von 64 SRAM-Zellen, die auf 8 kommerziellen SRAM-Chips gemessen wurden, erhielten sie $\mathbf{D}^{\text{inter}}$ = 43.16% und $\mathbf{D}^{\mathbf{intra}}$ =3.8%, und für 15 Blöcke von 64 SRAM-Zellen aus dem eingebetteten Speicher in 3 Mikrocontroller-Chips erhielten sie $\mathbf{D}^{\text{inter}}$ = 49.34% und $\mathbf{D}^{\mathbf{intra}}$ = 6.5%.

4. Verwendete Hardware

Wie in der Einleitung der Arbeit bereits beschrieben wurde, soll eine SRAM basierte PUF in einem Block RAM des GateMate™ FPGAs implementiert werden. Hierzu wird ein Evaluationsboard und weitere Komponenten benötigt, die im Folgenden vorgestellt werden.

4.1 GateMate™ FPGA Evaluation Board

Das GateMateTM Evaluation Board ist eine funktionsreiche, sofort einsatzbereite Entwicklungsplattform für den CCGM1A1 FPGA. Es dient als Referenzdesign und ermöglicht einen direkten Einstieg in die Anwendungsentwicklung. Benutzeranwendungen können auf jede der sechs verfügbaren I/O-Bänke zugeschnitten werden. Das Anschließen zusätzlicher Hardware ist dank vorhandener Pmod-Anschlüsse ebenfalls einfach. Für diesen standardisierten Steckert-Typ existieren einer Vielzahl von Peripherieplatinen, einschließlich Motorsteuerungen, Sensoren, Displays und mehr.

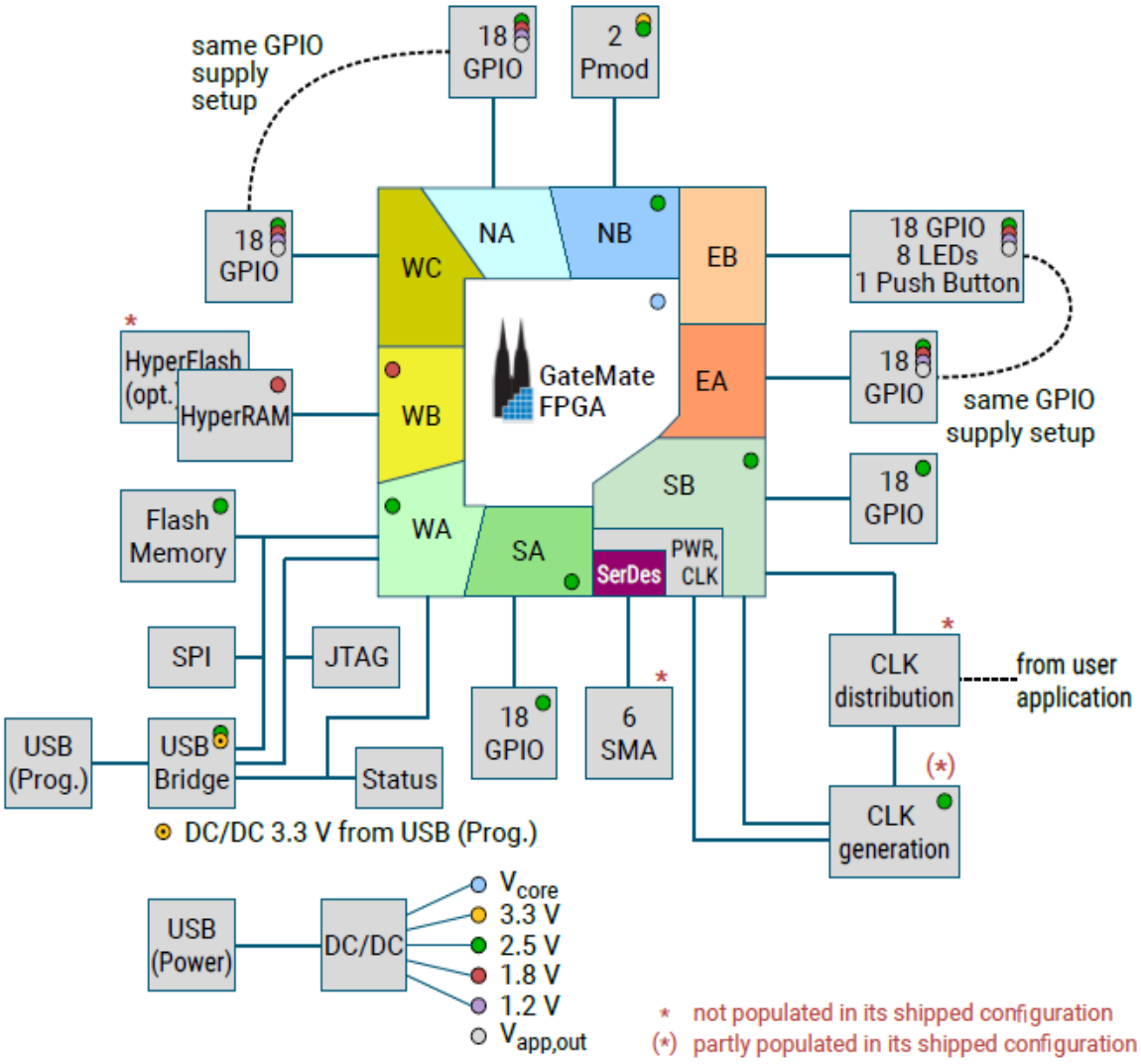


Abbildung 3: Überblick über das GateMate™ FPGA Evaluation Board Version 3.1 [9]

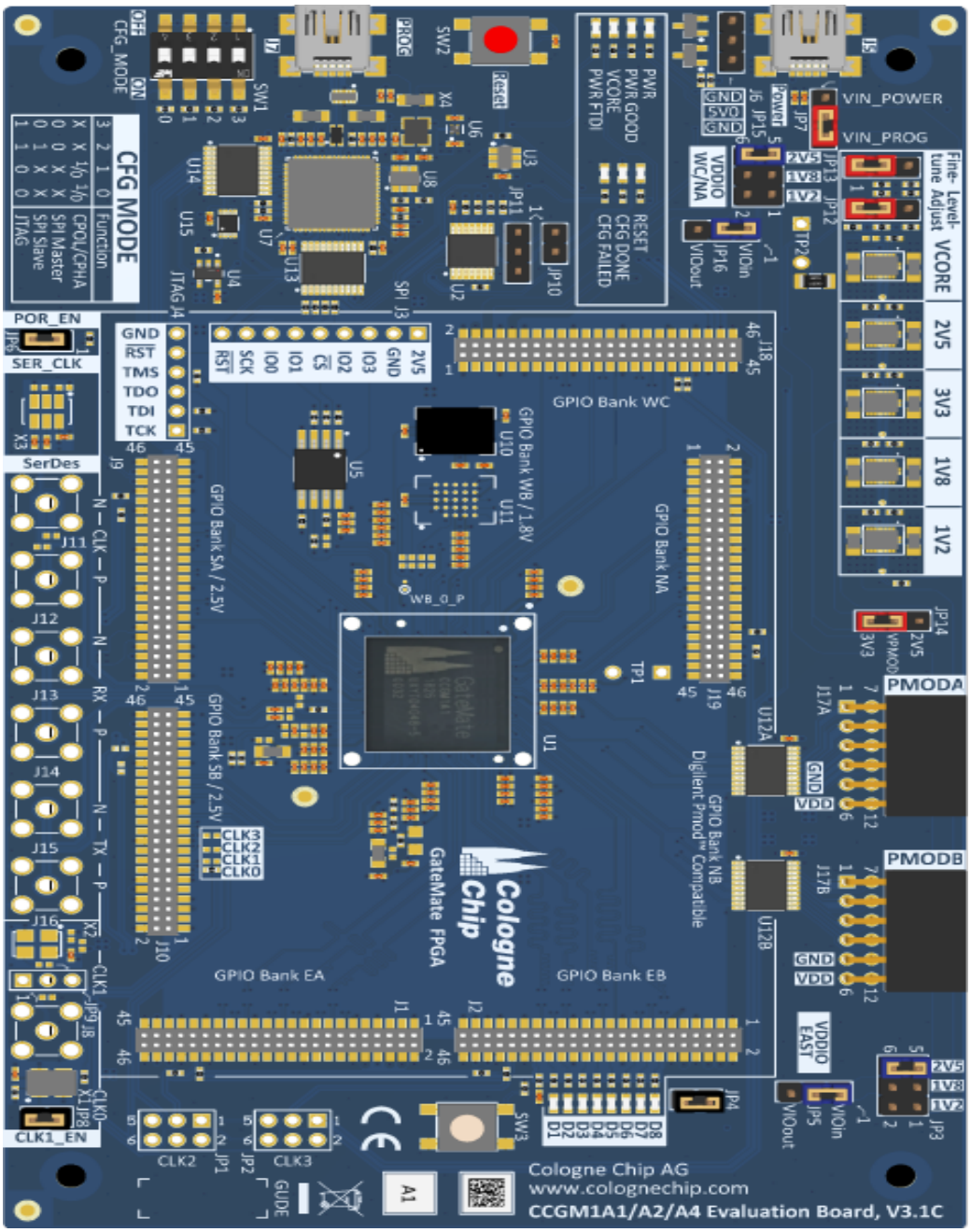


Abbildung 4: GateMate™ FPGA Evaluation Board Version 3.1 [10]

In Abb. 3 wird ein Überblick auf die Funktionen des Evaluation Boards gegeben. Einige General Purpose Input/Output (GPIO)-Bänke haben je nach ihrer Funktion einen festen Spannungspegel. Andere können auf unterschiedliche Spannungen nach den Anforderungen der Benutzeranwendung konfiguriert werden. Die Draufsicht auf die Leiterplatte (PCB) ist in Abb. 4 dargestellt. Es ist zu beachten, dass einige Komponenten wie z.B. die SMA-Stecker auf der linken Seite der Leiterplatte für erweiterte Funktionen nur vorbereitet sind und bei Bedarf vom Benutzer selbst bestückt werden müssen. [5]

4.1.1 Blockdiagramm

Das GateMateTM FPGA-Evaluierungsboard ermöglicht einen effektiven Zugang zu den Funktionen des CCGM1A1. In Abb.5 wird mit Hilfe eines Blockdiagramms ein vereinfachter Überblick auf die Leiterplatte (PCB) gegeben. Typischerweise wird eine einzige Versorgungsspannung von einem beliebigen USB-Bus abgeleitet und für die Stromversorgung des PCBs und der Benutzeranwendung verwendet. Die FPGA-Konfiguration kann auf zwei verschiedenen Arten geladen werden. Mit Hilfe des On-Board-Flash-Speichers kann die FPGA-Konfiguration nach einem Reset automatisch erfolgen. Alternativ kann auch ein Host-Controller zum Laden der FPGA-Konfiguration verwendet werden. Die Anwenderapplikation kann die General Purpose Input / Output (GPIO) Bänke des GateMate™ FPGAs frei konfigurieren und auf einige zusätzliche spezialisierte Signale wie Reset und Clock zurückgreifen, um gestellte Anwendungsanforderungen zu erfüllen. [11]

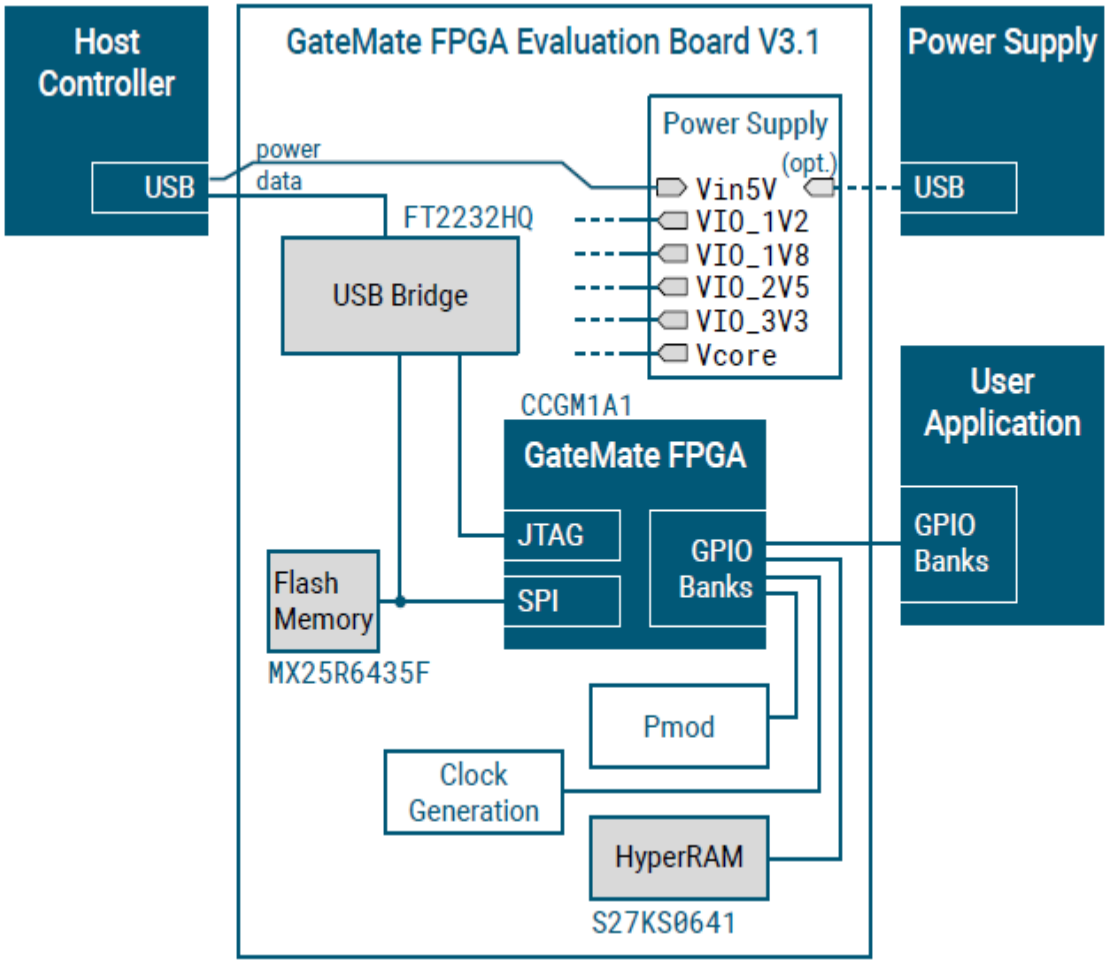


Abbildung 5: Blockdiagramm des GateMate™ FPGA Evaluation Board [12]

4.1.2 Funktionen des Evaluation-Boards

In diesem Kapitel werden die Funktionen des GateMateTM FPGA Evaluation Board beschrieben.

4.1.2.1 PCB Power Supply

Die Stromversorgungseinheit besteht aus fünf DC-DC Wandlern, die alle Spannungen bieten, um eine breite Palette an Anforderungen von Benutzeranwendungen zu erfüllen. Alle Wandler werden aus einer einzigen Quelle gespeist. Typischerweise kann ein USB-Netzteil verwendet werden. [13]

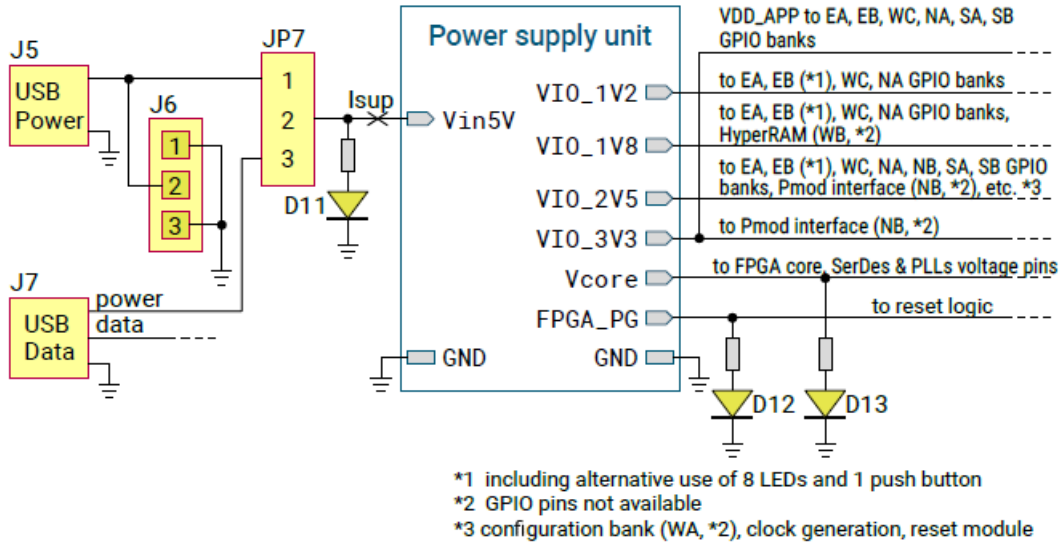


Abbildung 6: PCB Stromversorgung [13]

4.1.2.2 GPIO Power Supply

Das CCGM1A1 FPGA bietet neun GPIO-Bänke (General Purpose Input / Output). In Tabelle 1 wird ein Überblick gegeben, wie die GPIO-Versorgungsspannung auf dem Evaluation Board auf unterschiedliche Weise eingestellt wird.

Figure	Page	GPIO Bänke	Eigenschaften des Systems
3.3	23	WC, NA, EA	GPIO- Spannung, ausgewählt aus drei On-Board-Quellen oder der Anwendungsspannung
3.4	24	EB	GPIO-Spannung wählbar aus drei On-Board-Quellen oder der Applikationsspannung, Zusatzfunktionen (LEDs und User-Button)
3.5	25	SA, SB	Einzelne 2,5 V Spannungsversorgung
-	-	WA	Konfigurationsbank, 2,5 V Versorgung
3.16	35	WB	HyperBusSpeicher, 1,8 V Versorgung
3.15	34	NB	Pmod-Schnittstelle, 2,5 V Versorgung

Tabelle 1: Zuordnung des GPIO-Stromversorgungsschemas zu den GPIO-Bänken [14]

In Abb. 7 wird die konfigurierbare GPIO-Versorgung für die GPIO-Bänke WC, NA und EA gezeigt. Für diese Bänke können eine von drei On-Board-Spannungen sowie die (VIO_OUT)-Versorgung aus der Benutzeranwendung ausgewählt werden. Die ausgewählte On-Board-Spannung wird auch in die Benutzeranwendung eingespeist (VIO_IN) und kann zur Ergänzung der separaten Spannung (VDD_APP) verwendet werden. [14]

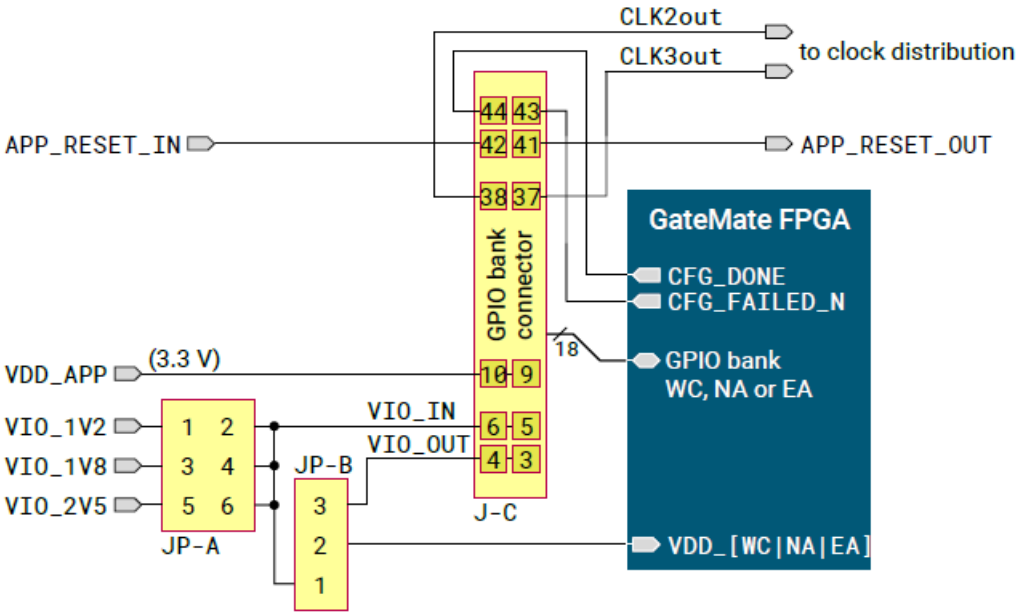


Abbildung 7: GPIO-Versorgung mit Spannungsauswahl [10]

4.1.2.3 SPI and JTAG Data Busses

Auf den SPI-Bus kann von jedem Anwendergerät zugegriffen werden. Dazu muss der Stecker J3 (2,54 mm Pinabstand, 9 Pins) bestückt werden. In Abb. 8 wird die Pinbelegung des SPI-Steckers dargestellt.

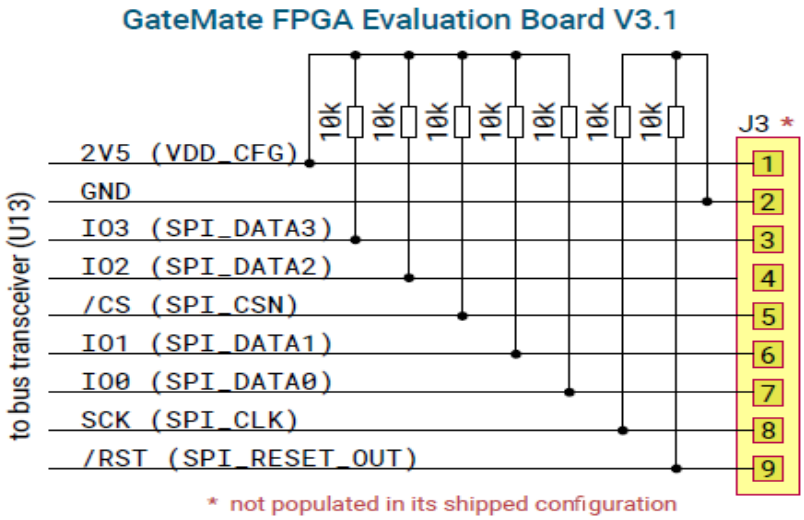


Abbildung 8: Optionaler SPI-Schnittstellenanschluss J3 [21]

Auf die JTAG-Schnittstelle kann ebenfalls von jedem beliebigen Anwendergerät zugegriffen werden. Dazu muss der Stecker J4 (2,54mm Pinabstand, 6 Pins) ebenfalls bestückt werden. In Abb. 9 wird die Pinbelegung des JTAG Steckers gezeigt.

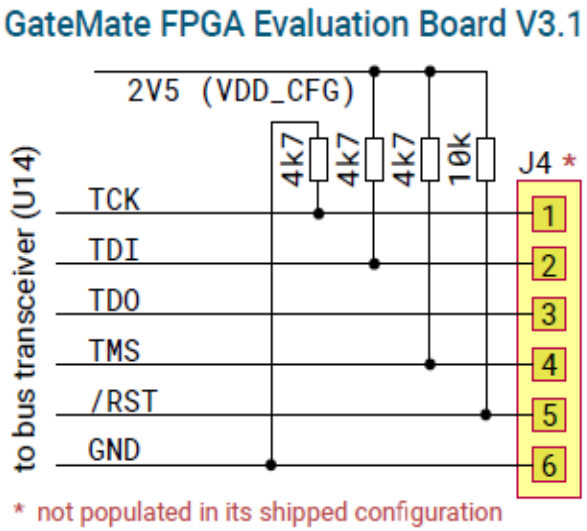


Abbildung 9: Optionaler JTAG-Schnittstellenanschluss J4 [21]

4.1.2.4 Konfiguration Mode und Reset

Es gibt mehrere Möglichkeiten, die FPGA-Konfiguration zu laden. Der Konfigurationsmode wird mit dem Schalter SW1 gewählt. Der FPGA-Reset hängt von verschiedenen Bedingungen ab:

- Die Taste SW2, die an der oberen Kante des Boards in Abbildung 4 zu sehen ist, kann vom Benutzer gedrückt werden, um einen FPGA-Reset auszulösen.
- Alle DC-DC-Wandler müssen nach dem Einschalten "Power Good" melden, um den Reset-Status zu beenden.
- Der Reset kann vom Host-Controller über die SPI- oder JTAG-Schnittstelle ausgelöst werden. In diesem Fall muss entweder Level Shifter U13 oder U14 das Reset-Signal des Hosts an das GateMate™ FPGA weiterleiten.
- Ein Reset-Signal kann von einem externen SPI-Gerät über den SPI-Anschluss J3 Pin 9 an das FPGA weitergeleitet werden (siehe Abb. 7).
- Ein Reset-Signal kann von einem externen JTAG-Gerät über den JTAG-Anschluss J4 Pin 5 an das FPGA geleitet werden (siehe Abb. 8).
- Die Benutzeranwendung kann den FPGA-Reset über den GPIO-Bankanschluss J-C Pin 41 (APP_RESET_OUT) auslösen. Dieses Signal kann über Jumper JP10 deaktiviert werden.

Über den GPIO-Bankanschluss wird ein Reset-Signal APP_RESET_IN an die Benutzeranwendung weitergeleitet. Dieses Signal wird entweder durch den Reset-Taster SW2 oder eine der oben genannten Reset-Bedingungen gesetzt (siehe Abb. 10).

Jeder GPIO-Bankanschluss hat ein APP_RESET_OUT-Signal. Diese sind alle miteinander verbunden und werden in Abb. 9 als "User application Reset out" bezeichnet. [15]

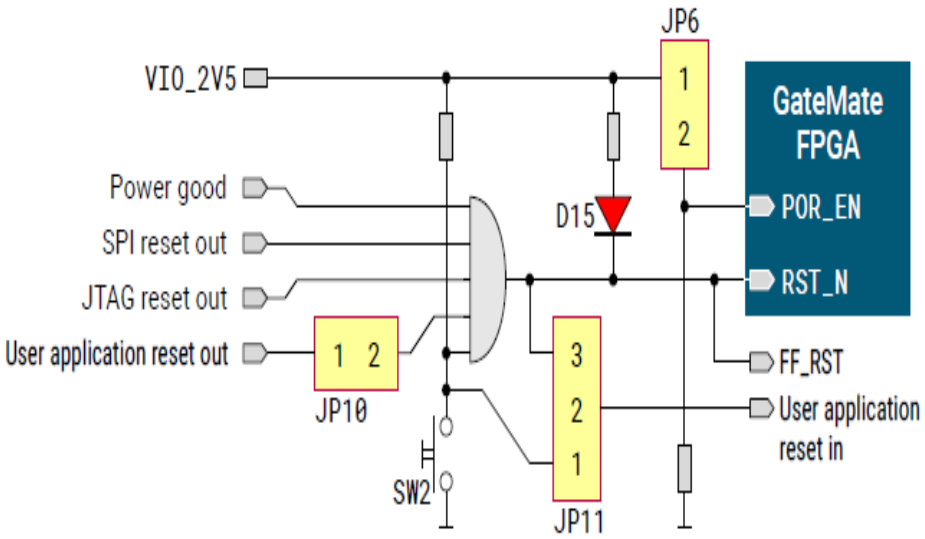


Abbildung 10: Reset Modul [16]

4.1.2.5 Clock Generation and Distribution

Das Evaluation Board hat einen On-Board-Oszillator X1 mit einer Frequenz von 10 MHz. Dieses Signal ist mit dem GPIO IO_SB_A8 verbunden, welcher dem Eingang für Clock 0 entspricht (siehe Abb. 11). Mit Jumper 8 kann der Oszillator deaktiviert werden.

Das GateMate™ FPGA hat drei weitere Takteingänge, die mit der GPIO WA Bank verbunden sind. Das Evaluation Board hat diese Clocks zur Verwendung vorbereitet, aber einige Komponenten müssen vom Benutzer bestückt werden, wie in Abb. 11 gezeigt.

Clock 1: Ein anderer Oszillator kann bestückt werden oder eine externe Taktquelle kann über die SMA-Buchse J8 eingekoppelt werden.

Clock 2 und 3: Die Benutzeranwendung kann zwei Taktsignale in die FPGA-Eingänge einspeisen. Die Jumper JP1 und JP2 müssen bestückt werden, um die Taktquelle auszuwählen.

Schließlich kann auch ein LVDS-Taktoszillator (Low-Voltage Differential Signaling) bestückt werden, um einen SerDes-Eingangstakt (Serializer/Deserializer) zu erzeugen (siehe Abb. 12). Dabei ist zu beachten, dass auch die Vorwiderstände R73 und R74 bestückt werden müssen. [17]

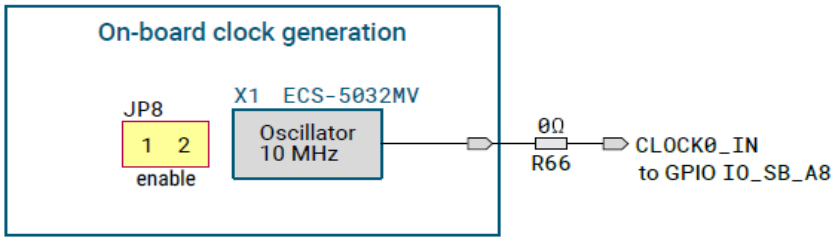


Abbildung 11: 10MHz On-Board Clock Oszillator [17]

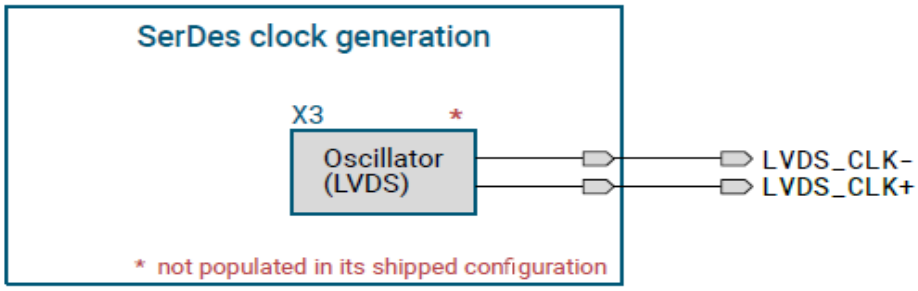


Abbildung 12: Optional SerDes Clock (LVDS Clock Oszillator) [17]

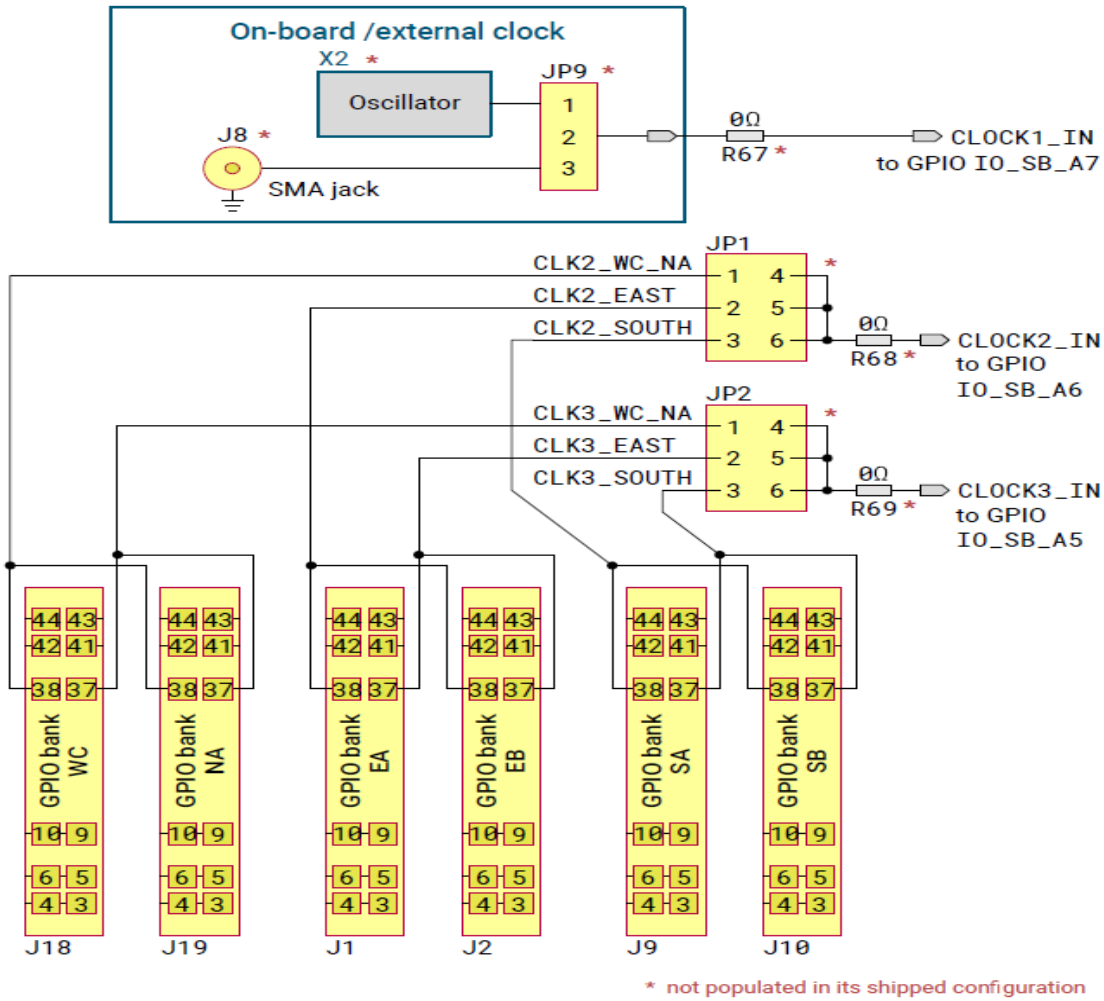


Abbildung 13: Optional Clock Signals [17]

4.1.2.6 GPIO Verbindungen zur User Applikation

Die Benutzeranwendung kann mit einer oder mehreren GPIO-Banken verbunden werden. Es stehen bis zu sechs Bänke zur Verfügung. Die anderen GPIO-Bänke werden anderweitig zugewiesen. Alle GPIO-Bank-Anschlüsse haben die in Abb. 14 dargestellte Pinbelegung. Zu beachten ist, dass die GPIOs auf zwei verschiedene Arten mit umgedrehtem Signalvektor angeordnet sind. Dies wurde so umgesetzt, um die freie Wahl einer GPIO-Bank für 1-Bank-Benutzeranwendungen, wie in Abb. 6 gezeigt, zu gewährleisten, und gleichzeitig eine differenzielle Signalführung auf dem Evaluation Board zu ermöglichen. Das PCB-Routing aller GPIO-Signale der Bänke EA und EB ist sowohl inter-pair als auch intra-pair längenangepasst. Die Benutzeranwendung kann über den 3,3-V-DC-DC-Wandler des Boards mit Strom versorgt werden. Darüber hinaus kann die GPIO-Spannung vom Evaluation Board zur Benutzeranwendung oder umgekehrt geleitet werden.

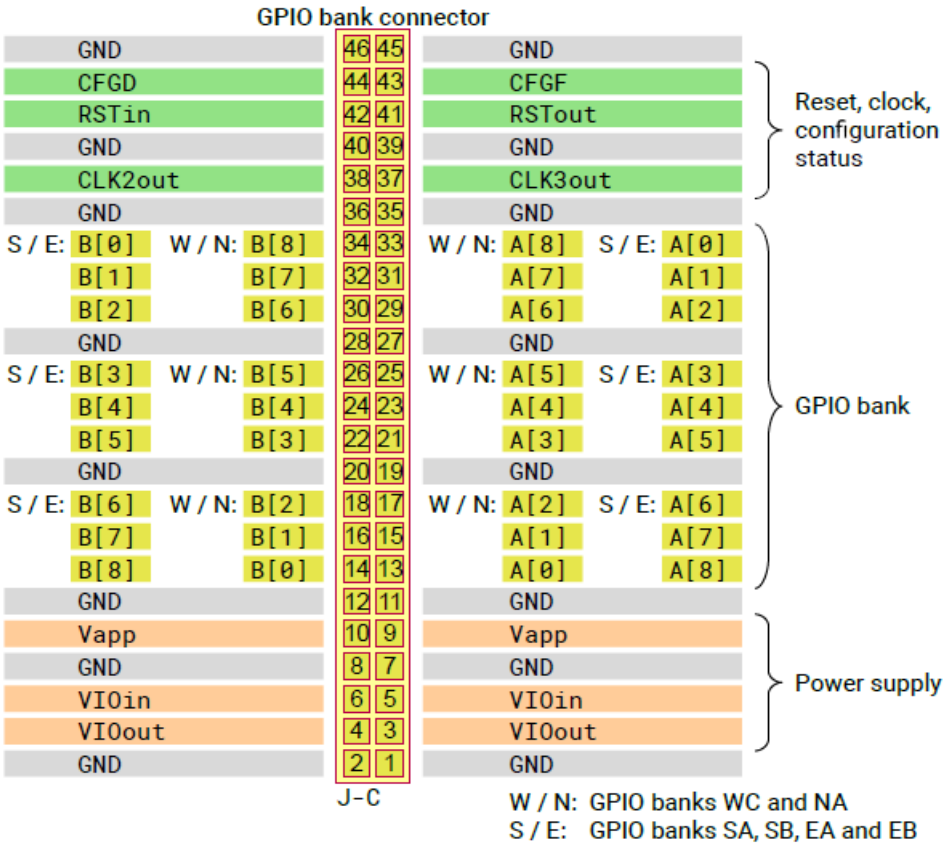


Abbildung 14: GPIO-Bank-Anschluss J-C [18]

4.1.2.7 Pmod Interface

Die GPIO-Bank NB stellt eine Pmod-Schnittstelle mit den zwei 12-Pin Steckern J17A und J17B dar. Eine Versorgungsspannung von 3,3 V kann über den Jumper JP14 gewählt werden, um die Pmod-Spezifikation zu erfüllen. Falls es für bestimmte Anwendungen gewünscht ist, kann außerdem auch eine 2,5-V-

Versorgung verwendet werden. Allerdings funktionieren viele Pmod-Geräte in diesem Fall nicht korrekt. [19]

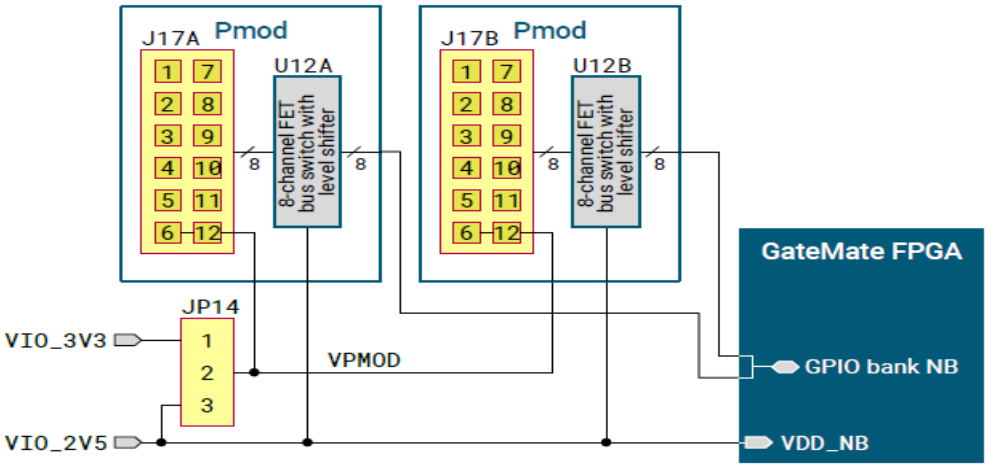


Abbildung 15: GPIO-Bank NB mit Pmod-Schnittstelle [19]

Im Allgemeinen können Pmod-Module über 6-, 8- oder 12-polige Stecker an Hauptplatinen angeschlossen werden. Über die Anschlüsse können mehrere digitale Steuersignale und auch serielle Protokolle wie SPI oder I2C übertragen werden. Pmod-Module ermöglichen bei Bedarf die modulare Erweiterung von Systemen und erlauben dadurch effektivere Designs. Darüber hinaus ist es mit Hilfe von PMOD Modulen möglich, analoge Signale und Stromversorgungen nur dorthin zu leiten, wo sie benötigt werden und einen Abstand zu den digitalen Komponenten einzuhalten, um die Einkopplung von Störungen zu reduzieren.

4.1.2.8 HyperRAM Device

Die GPIO-Bank WB wird verwendet, um einen On-Board-HyperRAM-Speicher bereitzustellen. Das Bauteil U10 ist ein 64-MB-Baustein, der, wie in Abbildung 16 dargestellt, mit der GPIO-Bank verbunden ist (siehe Tabelle 2).

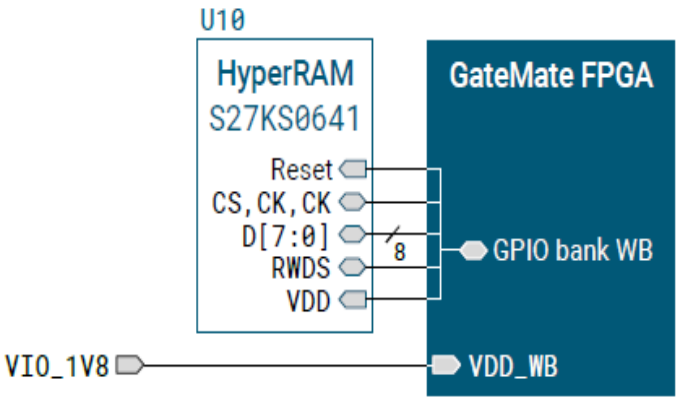


Abbildung 16: GPIO-Bank WB mit HyperRAM Baustein [22]

GPIO	HyperRAM Signal	GPIO	HyperRAM Signal
IO_WB_A0	–	IO_WB_B0	CS
IO_WB_A1	–	IO_WB_B1	–
IO_WB_A2	RESET	IO_WB_B2	–
IO_WB_A3	CK	IO_WB_B3	CK
IO_WB_A4	–	IO_WB_B4	RWDS
IO_WB_A5	DQ0	IO_WB_B5	DQ1
IO_WB_A6	DQ2	IO_WB_B6	DQ3
IO_WB_A7	DQ4	IO_WB_B7	DQ5
IO_WB_A8	DQ6	IO_WB_B8	DQ7

Tabelle 2: GPIO-Zuweisung an das On-Board-HyperRAM-Baustein. [22]

4.2 SERDES Interface

Das GateMate™ Evaluation Board ist auf die Verwendung einer SerDes-Schnittstelle ausgerichtet. Um Zugriff auf die CCGM1A1 SerDes-Schnittstelle zu erhalten, müssen die SMA-Buchsen J13...J16 vom Benutzer bestückt werden (siehe Abb. 17).

Es gibt zwei Möglichkeiten, das SerDes-Taktsignal einzuspeisen:

- ✓ Die SMA-Buchsen J11 und J12 werden bestückt, um einen externen LVDS-Takt einzuspeisen.
- ✓ Ein integrierter LVDS-Oszillator und die Vorwiderstände R73 und R74 werden bestückt (siehe auch Abb. 12).

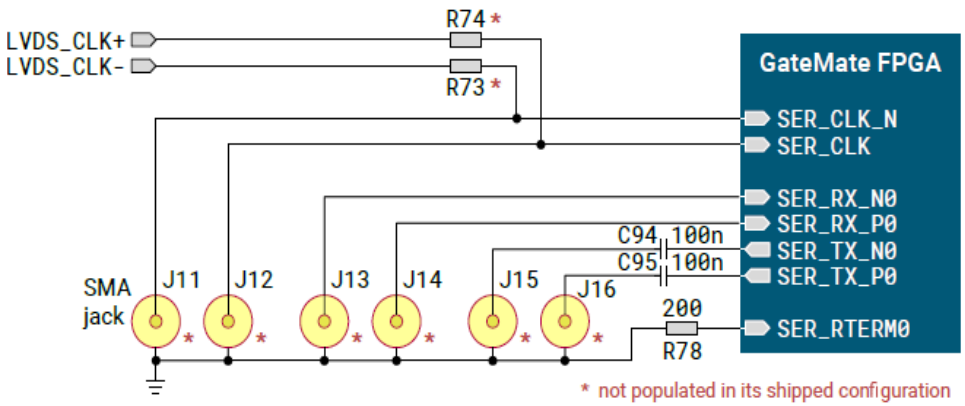


Abbildung 17: Optionale SerDes-Schnittstelle [23]

4.3 Pmod USBUART

In diesem Projekt sollen Daten aus dem FPGA ausgelesen werden und über eine USB Schnittstelle an den PC gesendet werden. Es wurde entschieden, für die Übertragung der Daten ein USBUART PMOD Modul zu verwenden. Das USBUART Pmod Modul bietet eine USB-zu-UART-Kreuzkonvertierung durch

den FTDI FT232R Baustein (siehe Abbildung 18 und 19). Benutzer können mit dem Pmod Daten in beide Richtungen senden und die konvertierten Daten im entsprechenden Format empfangen.

Abbildung 18: Pmod USBUART [24]

Abbildung 19: Pmod USBUART [24]

Merkmale:

- USB-zu-Seriell-UART-Schnittstelle
- Micro-USB-Anschluss
- Option zur Stromversorgung der Systemplatine über den FTDI-Chip
- 6-poliger Pmod-Anschluss mit UART-Schnittstelle

Elektrisch:

- Bus-UART
- Spezifikationsversion 1.2.0
- Logikpegel 3,3 V

Physisch:

- Breite 2,54 cm (1,0 Zoll)
- Länge 2,03 cm (0,80 Zoll)

UART steht für Universal Asynchronous Receiver und Transmitter, ist eine elektronische Schnittstelle, die zur seriellen Übertragung digitaler Daten dient. Bei der UART Protokolleinheit kann es sich sowohl um ein eigenständiges elektronisches Bauelement (ein UART-Chip bzw. -Baustein) oder um einen Funktionsblock eines höherintegrierten Bauteils z. B. eines Mikrocontrollers handeln. Eine UART-Schnittstelle dient zum Senden und Empfangen von Daten über eine Datenleitung und bildet den Standard der seriellen Schnittstellen an PCs und Mikrocontrollern. Auch im industriellen Bereich ist die Schnittstelle mit verschiedenen Interfaces (z. B. RS-232 oder EIA-485) sehr verbreitet. Die Daten werden als serieller digitaler Datenstrom mit einem fixen Rahmen übertragen, der aus einem Start-Bit, fünf bis maximal acht oder neun Datenbits (abhängig von der Anwendung), einem optionalen Parity-Bit zur Erkennung von Übertragungsfehlern und einem oder zwei Stopp-Bits besteht.

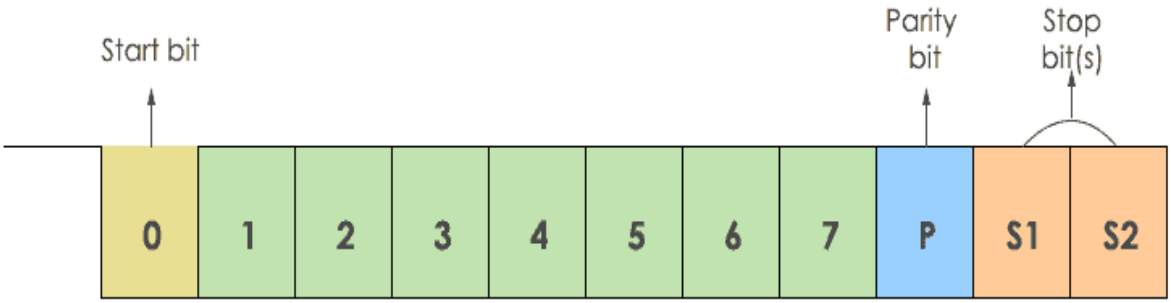


Abbildung 20: UART Schnittstelle. [31]

Die Besonderheit bei der asynchronen Betriebsweise besteht darin, dass der Sender dem Empfänger kein eigenes Taktsignal auf einer eigenen Steuerleitung überträgt. [25] Stattdessen synchronisiert sich der Empfänger über die Länge des Rahmens, vermittelt durch die Vorderflanke des neuen Start-Bits nach dem letzten empfangenen Stopp-Bit, sowie die eingestellte Baudrate, welche in diesem Fall der Bitrate entspricht. Weil der Beginn einer Übertragung mit dem Start-Bit zu beliebigen Zeitpunkten erfolgen kann, wird diese serielle Schnittstelle als asynchron bezeichnet. Um eine Synchronisation gewährleisten zu können, ist die Anzahl der übertragbaren Datenbits innerhalb eines Rahmens eingeschränkt. Würde mehr als ein Byte in einen Rahmen verpackt, könnte die Synchronisation verloren gehen, was zu Fehlinterpretationen des Datenstromes und somit zu einer fehlerhaften Übertragung führen könnte. Wenn in einer Sendepause keine Daten zu übertragen sind, so legt der Sender die Leitung auf die Polarität des Stopp-Bits. Weil der Empfänger sich mit jedem übertragenen Rahmen neu synchronisiert, ist es nicht notwendig, dass zwischen den übertragenen Rahmen ein zeitlicher Zusammenhang besteht. Nur für die Dauer eines einzelnen Rahmens müssen Sender und Empfänger synchron arbeiten, nicht länger. Das nennt man „bytesynchron“ oder zeichensynchron. [25]

Die Geschichte des UART ist eng verbunden mit der Standardisierung der RS-232 Datenkommunikation. Waren die ersten UARTs für Datenübertragungsraten weniger hundert Bit/s und den Anschluss

an Teletypes mit Stromschnittstelle oder Modems vorgesehen, so erreichten sie in späteren Jahren als eigenständige Chips mehrere Megabit pro Sekunde Bei der Datenkommunikation über die RS-232-Schnittstelle findet. Die asynchrone Übertragung findet Verwendung. Diese Schnittstelle weist eine vergleichsweise große Verbreitung auf. Ein UART erzeugt die auf der RS-232-Schnittstelle zu übertragenden Datenbits und den dazu notwendigen Datenrahmen. Die eigentliche RS-232-Schnittstelle besteht zusätzlich noch aus Pegelumsetzern und weiteren Bauelementen wie Steckern, welche nicht mehr Teil eines UART sind. Damit UART-Baugruppen kommunizieren können, müssen die Empfangsleitung (Rx) der einen und die Sendungsleitung (Tx) der anderen Baugruppe am Stecker gegenüberstehen.

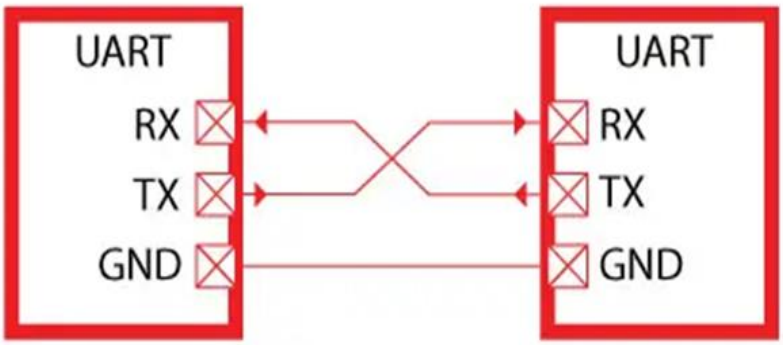


Abbildung 21Kommunikationsprotokoll. ***[32]***

Damit sind stets zwei Steckerbelegungstypen (Master und Slave) erforderlich, auch wenn die Geräte vollkommen gleichberechtigt kommunizieren. Sollen Master mit Master oder Slave mit Slaven kommunizieren können, sind Kreuzverbinder (analog dem Null-Modem-Kabel der seriellen Schnittstelle oder dem Cross-Over-Kabel des Ethernets) erforderlich. Eine Modifikation, die Single-Wire UART (SWART), vermeidet dieses Verpolungsproblem. Werden Rx und Tx zusammen auf einem Pin vereint, ist zwar nur eine Simplex-Übertragung möglich, dafür aber können dann beliebige Module miteinander kommunizieren. Es können sogar mehrere UART-Module auf einem Draht kommunizieren (SWART-Bus). Die SWART ist insbesondere für kurze Entfernungen und für Datenraten bis 115.200 Baud geeignet. [25]

Neben der Realisierung von UART als Kommunikationsbaustein in Mikrocontrollern, oder als eigenständige integrierte Schaltung ist der Entwurf dieser Schnittstelle auch in Form von Hardwarebeschreibungssprachen für die Integration oder als sogenanntes „Software-UART“, das nur durch eine Programmabfolge vorliegt und bestimmte Ein-/Ausgabepins direkt ansteuert (Bit-Banging).

4.4 Pmod BTN für 4 Benutzerdrucktasten:

Der Pmod BTN bietet dem Benutzer vier Drucktasten für einfache Benutzereingaben auf einer Systemplatine. In diesem Projekt wird der Pmod BTN für das Signal RST benötigt. Die Drucktaste BTN1 wird als Pin_in mit der GPIO Bank IO_NB_A0 des FPGAs Gatemate™ verbunden.

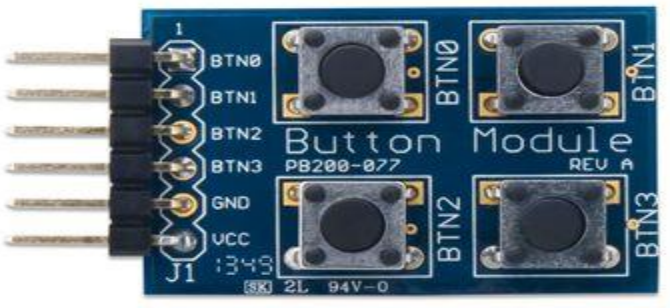


Abbildung 22: Pmod_Button. [26]

Die Merkmale des Pmod BTN sind:

- ✓ Vier Drucktasten mit Momentanwert
- ✓ Entprellende Filter
- ✓ Invertierende Schmitt-Trigger
- ✓ Kleine Leiterplattengröße für flexible Entwürfe 1,2" × 0,8" (3,0 cm × 2,0 cm)
- ✓ 6-poliger Pmod-Anschluss mit GPIO-Schnittstelle
- ✓ Entspricht der Digilent Pmod-Schnittstellenspezifikation Typ 1

5. Verwendete Software-Pakete

Beim Entwurf digitaler Schaltungen spielt EDA Software eine wichtige Rolle. In diesem Kapitel werden alle verwendete Software-Pakete vorgestellt. Außerdem wird die Vorgehensweise bei der Implementierung digitaler Schaltungen auf dem Gatemate FPGA beschrieben.

5.1Ubuntu

Ubuntu, ist eine GNU/Linux-Distribution, die auf Debian basiert. Der Name Ubuntu bedeutet auf Zulu etwa „Menschlichkeit“ und bezeichnet eine afrikanische Philosophie. Die Entwickler verfolgen mit dem System das Ziel, ein einfach zu installierendes und leicht zu bedienendes Betriebssystem mit aufeinander abgestimmter Software zu schaffen. Das Projekt wird vom Software-Hersteller Canonical gesponsert, der vom südafrikanischen Unternehmer Mark Shuttleworth gegründet wurde. Ubuntu konnte seit dem Erscheinen der ersten Version im Oktober 2004 seine Bekanntheit stetig steigern und ist inzwischen eine der meistgenutzten GNU/Linux-Distributionen. Die Nutzerzahl wird auf etwa 25 Millionen geschätzt. Ubuntu setzte in den Versionem 11.04 bis 17.04 standardmäßig die von der Ubuntu-Entwicklergemeinschaft selbst entwickelte Desktop-Umgebung Unity ein, aber seit der Version 17.10 basiert die graphische Bedienoberfläche von Ubuntu wieder auf Gnome. Von Ubuntu existieren verschiedene Abwandlungen. Zu den offiziellen Unterprojekten gehören unter anderem Kubuntu mit KDE, Xubuntu mit Xfce, Ubuntu MATE mit MATE, Ubuntu Budgie mit Budgie sowie Ubuntu Studio, das speziell auf die Anforderungen von Audio-, Grafik- und Videobearbeitung ausgerichtet ist. Neue Ubuntu-Versionen erscheinen jedes halbe Jahr im April (04er-Versionen) und im Oktober (10er-Versionen). Die derzeit aktuelle Version 22.04 LTS „Jammy Jellyfish“ erschien am 21. April 2022. [27]

5.2 VHDL

Very High Speed Integrated Circuit Hardware Description Language (auch VHSIC Hardware Description Language), kurz VHDL, ist eine Hardwarebeschreibungssprache, mit der es möglich ist, digitale Systeme textbasiert zu beschreiben. VHDL ist seit 1987 als IEEE-Standard festgelegt und es gibt inzwischen einige ebenfalls standardisierte Spracherweiterungen. Darüber hinaus gibt es Sprachderivate wie zum Beispiel VHDL-AMS, mit deren Hilfe auch analoge oder Mixed-Signal-Systeme beschrieben werden können. VHDL ist als Beschreibungssprache keine Programmiersprache; da sie jedoch Objekte beschreibt, deren Aufgabe meist die Informationsverarbeitung ist, kann über deren Simulation dennoch Datenverarbeitung stattfinden, indem für diesen Simulationslauf mitgegebene „Eingangsdaten“ von der (simulierten) Hardware zu „Ergebnisdaten“ verarbeitet werden. Durch diesen Umweg kann VHDL (in Kombination mit einem Simulator) wie eine Programmiersprache Turing-vollständige Datenverarbeitung beschreiben. Durch fortschrittliche Schaltungsgeneratoren ist es mitunter sogar möglich, anstatt des Hardwareaufbaus für einen Algorithmus nur den Algorithmus selbst anzugeben; die dazugehörige Schaltung wird vollautomatisch erzeugt. Dies nähert VHDL einer Programmiersprache weiter an. [28]

5.3. HTerm

HTerm ist ein Terminalprogramm für die serielle Schnittstelle. HTerm wird in diesem Projekt eingesetzt, um die aus dem BRAM ausgelesenen Daten zu empfangen und darzustellen. Es ist unter anderem möglich, die Baud-Rate und die Parität für das Versenden und den Empfang über die Schnittstelle zu konfigurieren. Außerdem bietet HTerm die Möglichkeit Daten als Dezimal-, Hexadezimal- und Binärzahlen sowie im ASCII-Code darzustellen.

5.3 Yosys

5.3.1 Definition

Yosys ist ein Softwarepaket für die logische Synthese (engl. RTL synthesis), mit dem man eine logische Schaltung aus ihrer Beschreibung in der Hardwarebeschreibungssprache (engl. Hardware Description Language, HDL) Verilog in eine Netzliste aus technologiespezifischen Gattern umwandeln kann. Eine derartige Gatternetzliste kann bei der Implementierung einer digitalen Schaltung als anwendungsspezifischer integrierter Schaltkreis (engl. Application Specific Integrated Circuit, ASIC) einem Place & Route Tool zur Platzierung und Verdrahtung und Generierung eines Layouts übergeben werden. Bei der Implementierung auf einem FPGA kann die Netzliste auf die vorhandenen Ressourcen abgebildet werden und zur Generierung eines Konfigurationsdatenstrom herangezogen werden. Yosys führt auch formale Verifikationsaufgaben durch. Er wurde von Clifford Wolf entwickelt. Er kann mit Hilfe des Plugins ghdl-yosys-plugin die Funktionalität von GHDL einbinden und damit auch VHDL Designs synthetisieren. [29]

5.3.2 Ausgewählte Funktionen und typische Anwendungen

- ✓ Verarbeitung nahezu aller synthetisierbarer Verilog-2005-Designs
- ✓ Konvertierung von Verilog nach BLIF / EDIF/ BTOR / SMT-LIB / einfaches RTL Verilog / usw.
- ✓ Eingebaute formale Methoden zur Überprüfung von Eigenschaften und Äquivalenz
- ✓ ASIC-Standard-Zellbibliotheken (im Liberty File Format)
- ✓ zu Xilinx 7-Series und Lattice iCE40 und ECP5 FPGAs
- ✓ Basis und/oder Front-End für kundenspezifische Abläufe

Yosys kann an jeden beliebigen Syntheseauftrag angepasst werden, indem die vorhandenen Durchläufe (Algorithmen) mit Hilfe von Synthese-Skripten kombiniert und zusätzliche Durchläufe nach Bedarf hinzugefügt werden, indem die Yosys C++-Codebasis erweitert wird.

5.3.3 GHDL

GHDL ist eine Abkürzung für G Hardware Design Language (derzeit hat G keine Bedeutung). Es ist ein VHDL-Analysator, Compiler, Simulator und experimenteller Synthesizer, der fast jedes VHDL-Design verarbeiten kann. Er übersetzt eine VHDL-Datei direkt in Maschinencode, indem er das GCC-Backend verwendet und keine Zwischensprache wie C oder C++ einsetzt. Daher sollte der kompilierte Code schneller und die Analysezeit kürzer sein als bei einem Compiler, der eine Zwischensprache verwendet. In Yosys wird GHDL als Plugin eingebunden, um die Synthese von VHDL Entwürfen zu ermöglichen.

5.3.4 Place & Route

Place and Route ist ein Schritt beim Entwurf von Leiterplatten, integrierten Schaltungen und feldprogrammierbaren Gate-Arrays. Wie der Name schon andeutet, besteht sie aus zwei Schritten, der Platzierung und dem Routing. Beim ersten Schritt, der Platzierung, wird entschieden, wo alle elektronischen Komponenten, Schaltkreise und Logikelemente auf einem im Allgemeinen begrenzten Platz platziert werden sollen. Danach folgt die Entflechtung, bei der das genaue Design aller Drähte festgelegt wird, die zur Verbindung der platzierten Komponenten benötigt werden. In diesem Schritt müssen alle gewünschten Verbindungen unter Einhaltung der Regeln und Beschränkungen des Fertigungsprozesses realisiert werden. Im Zusammenhang mit dem GateMate FPGA wird ein proprietäres Programm von Cologne Chip für das Place & Route verwendet. Das Place & Route Tool erhält die Gatternetzliste, die Yosys während es Synthesedurchlaufs generiert hat, als Eingabe. Zusätzlich können Anwenderwünsche wie z.B. die Position der Pins und Timing Constraints übergeben werden. Die Place & Route Software führt Algorithmen aus, mit denen die Gatter aus der Netzliste auf die FPGA Ressourcen verteilt werden. Die ersten Schritte der Prozessierung umfassen dabei Verfahren zur Geschwindigkeits- oder Flächenoptimierung vor dem Mapping. Nach der Platzierung und dem Routing kann die statische Zeitanalyse (STA) zu weiteren Optimierungsschritten führen und macht die Place & Route Software zu einem iterativen Prozess von beschränkungsgesteuerten Re-Placement- und Re-Routing-Schritten, um schließlich die Benutzeranforderungen zu erfüllen.

5.3.5 openFPGALoader7

openFPGALoader ist ein universelles Dienstprogramm für die Programmierung bzw. die Übertragung des Konfigurationsdatenstroms von FPGAs. Es ist kompatibel zu vielen Boards, Kabeln und FPGAs der wichtigsten Hersteller (Xilinx, Altera/Intel, Lattice, Gowin, Efi-nix, Anlogic, Cologne Chip). openFPGALoader kann unter Linux, Windows und macOS ausgeführt werden.

5.3.6 Zadig USB Installer

Zadig ist eine Windows-Anwendung, die generische USB-Treiber wie WinUSB, libusb-win32/libusb0.sys oder libusbK installiert. Diese ermöglichen den Zugriff auf USB-Geräte. Dies kann besonders dann nützlich sein, wenn:

- Zugriff auf ein Gerät mit einer libusb-basierten Anwendung erforderlich ist,
- ein generischer USB-Treiber aktualisiert werden soll, oder
- mit WinUSB auf ein Gerät zugegriffen werden muss.

In diesem Projekt wird der Zadig USB Installer verwendet, um den Treiber zu installieren, mit dem der Zugriff auf das Gatemate Evaluationboard möglich ist.

5.3.7 Entwurfs-Flow des GateMate FPGAs

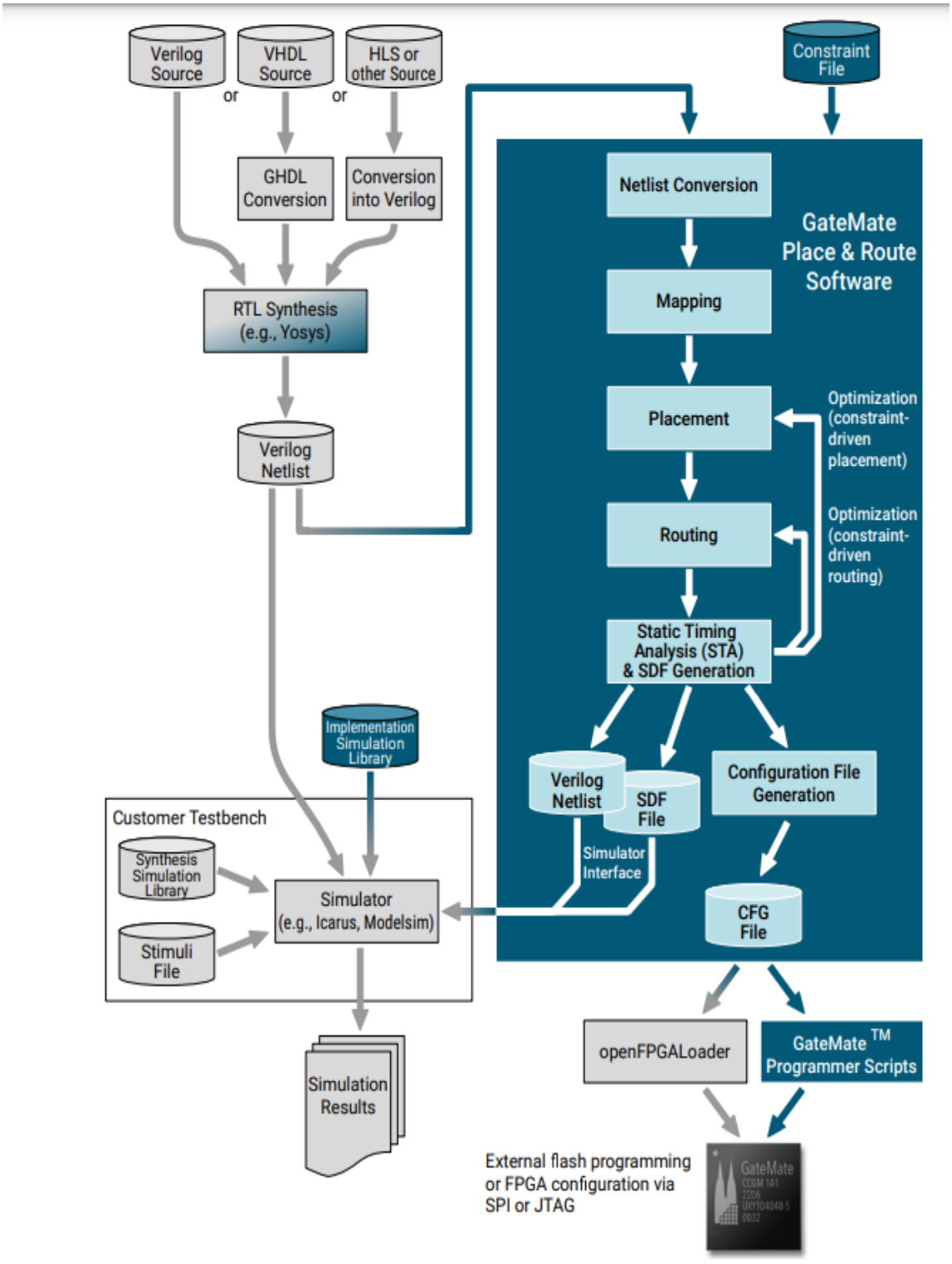


Abbildung 23: Ablauf der Synthese und Implementierung. [30]

Dieser Abschnitt beschreibt den Entwurfsvorgang bei der Implementierung digitaler Schaltungen auf Gatemate^{TM} FPGAs. Der Entwurfsflow ist in Abbildung 21 dargestellt. Als Synthesewerkzeug wird das Open-Source Tool Yosys verwendet, was Verilog Module verarbeiten kann. Yosys verfügt aber auch

über eine umfangreiche VHDL-Unterstützung. VHDL-Quellen werden in Yosys über GHDL bzw. das ghdl-yosys-plugin ohne zusätzlichen Aufwand für den Nutzer synthetisiert. Die Steuerung von Yosys und den anderen EDA Werkzeugen erfolgt auf der Kommandozeilenebene oder skriptgesteuert. Zur Vereinfachung der Bedienung werden vorgefertigte Skripte von Cologne Chip geliefert, deren Ausführung mit Hilfe eines Makefiles in kurze Kommandos abgebildet werden. In der Datei PUF_BAT_Synth.tcl , die sich im Verzeichnis **/tcl** befindet, werden alle Design Files in der Form: src/File.vhd eingetragen. Der für dieses Projekt vollständige Eintrag in der PUF_BAT_Synth.tcl Datei lautet

ghdl --warn-no-binding -C --ieee=synopsys src/BRAM_BA.vhd src/PUF_BAT.vhd src/counter_neu.vhd src/PUF_BA.vhd src/Counter_BA.vhd src/FSM_BA.vhd src/MUX_BA.vhd src/serial_tx.vhd -e ${top}

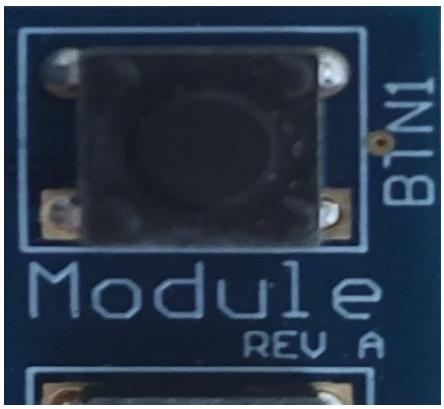
Im Makefile muss dabei die Variable top auf das Toplevel Modul gesetzt werden. In diesem Projekt lautet der Name des Toplevel Moduls PUF_BAT. Im Allgemeinen erkennt Yosys automatisch das Top-Level-Modul in der Menge der Eingabedateien. Zur Synthese wird der Befehl **„make synth“** verwendet. Nachdem man den Befehl **„make synth“** eingegeben hat, wird unter Anderem das Skript **tcl/PUF_BAT.tcl** ausgeführt und alle notwendigen Parameter übergeben, um die Designdatein zu öffnen und die Synthese zu starten und das Syntheseergebnis in ein Unterverzeichnis zu schreiben. Die Synthese erzeugt eine Gate-Level-Darstellung des Schaltungsentwurfs in Form einer Verilog Netzliste von architekturspezifischen Primitiven. Nachdem die Synthese erfolgreich durchgelaufen ist, wird in diesem Projekt die Datei **PUF_BAT_synth.v** im Unterverzeichnis **/net** erstellt. Die synthetisierte Netzliste ist für dieses Projekt von besonderer Bedeutung, weil Sie die BRAM_20K Instanz mit ihren Vorinitialisierungswerten enthält. Wurde im VHDL Modul, welches das BRAM beschreibt, das Array Signal, das dem Speicher entspricht, vorinitialisiert, finden sich diese Initialisierungswerte in den INIT_xx Attributen der BRAM_20K Instanz in der synthetisierten Netzliste wieder. Andernfalls wird die BRAM_20K Instanz mit Nullen vorinitialisiert. Um die zufälligen BRAM Inhalte auszulesen, die sich nach dem Einschaltmoment ergeben, müssen die INIT_xx Attribute mit den jeweiligen Initialisierungswerte aus der Deklaration der BRAM_20K Instanz manuell entfernt werden.

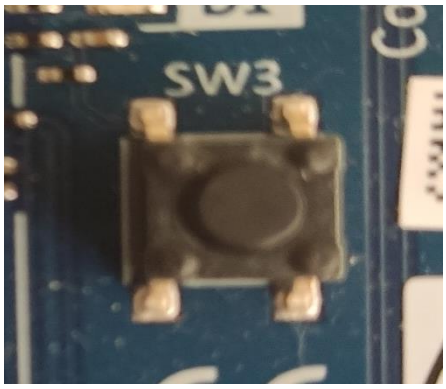
Als nächstes wird der Befehl **„make impl“** für die Implementierung verwendet. Der Befehl **„make impl“** ruft das Cologne Chip Place & Route Tool auf und übergibt alle notwendigen Parameter, um die Netzliste des Designs zu öffnen und die Abbildung auf die FPGA Ressourcen durchzuführen. Im Unterverzeichnis **/src** befindet sich auch die PUF_BAT.ccf Datei. Sie ermöglicht die Zuordnung zwischen den Ein- und Ausgängen des Toplevel-Moduls des Schaltungsentwurfs mit den Ports des GateMate™ FPGAs. Die Einträge im CCF-File haben das folgende Format:

<pin-direction> "<pin-name>" Loc = "<pin-location>" | <opt.-constraints>;

Abhängig von den Parametern erzeugt das Tool mindestens die folgenden Ausgabedateien:

- ✓ Konfigurations_Bitsream: **/PUF_BAT_vhdl/PUF_BAT.cfg.bit**
- ✓ Verilog-Netzliste für die Nachimplementierungssimulation: **/net/PUF_BAT.v**
- ✓ SDF-Verzögerungsdatei für die Simulation nach der Implementierung: **/PUF_BAT_vhdl/PUF_BAT.SDF**
- ✓ Pin-Datei (falls aktiviert): PUF_BAT.pin
- ✓ Platzierungsdatei (falls aktiviert): **/PUF_BAT_vhdl/PUF_BAT.place**
- ✓ Querverweisdatei (falls aktiviert): **/PUF_BAT_vhdl/PUF_BAT.crf**





.

5.3.8 Verzeichnis Strukturen des Projekts

Für die Durchführung von Implementierungsprojekten empfiehlt Cologne Chip die in Abbildung 26 dargestellte Ordnerstruktur. Im Verzeichnis **/cc-tool-win** befinden sich die zwei Unterverzeichnisse **bin** und **workspace**. Im Unterverzeichnis **/bin** befinden sich die drei weiteren Ordner **openFPGALoader**, **p_r** und **Yosys**, in der die verwendeten EDA Tools installiert sind. Diese Tools werden während der Synthese, Implementierung und Konfiguration des Projekts aufgerufen. Im Unterverzeichnis **/workspace** werden die unterschiedlichen Anwendungsprojekte in Unterverzeichnisse abgelegt. In diesem Projekt wurde ein Verzeichnis **PUF_BAT** erstellt. Darüber hinaus befindet sich im Verzeichnis die Datei

config.mk. Hierbei handelt es sich um eine Datei, in der projektübergreifende Konfigurationen für die Verwendung mit dem Befehl make vorhanden sind. Unter anderem werden make Regeln mit Kommandozeilenbefehlen und verwendeten Option und Argumenten formuliert. Im Unterverzeichnis /**PUF_BAT** sind die vier weiteren Verzeichnisse **log**, **net**, **src** und **tcl** und das eigentliche **Makefile** vorhanden. Im Unterverzeichnis **/net** werden die Syntheseergebnisse in der Datei **PUF_BAT_synth.v** abgelegt. Im Unterverzeichnis /**src** befinden sich alle VHDL Sourcen und auch das **CCF** File mit der die verwendeten FPGA Pins konfiguriert werden. Im Unterverzeichnis **/tcl** befindet sich die Datei PUF_BAT_Synth.tcl. In dieser Datei müssen unter anderem alle Module des Schaltungsentwurfs in einer bestimmten Form eingetragen werden**.** Im Makefile wird die Datei config.mk eingebunden und weitere projektspezifische Konfigurationen abgelegt. Beispielsweise wie das Hauptmodul **PUF_BAT** der Variable Top zugewiesen, damit Yosys das Hauptmodul und die dazugehörigen Eingänge und Ausgänge erkennen kann.

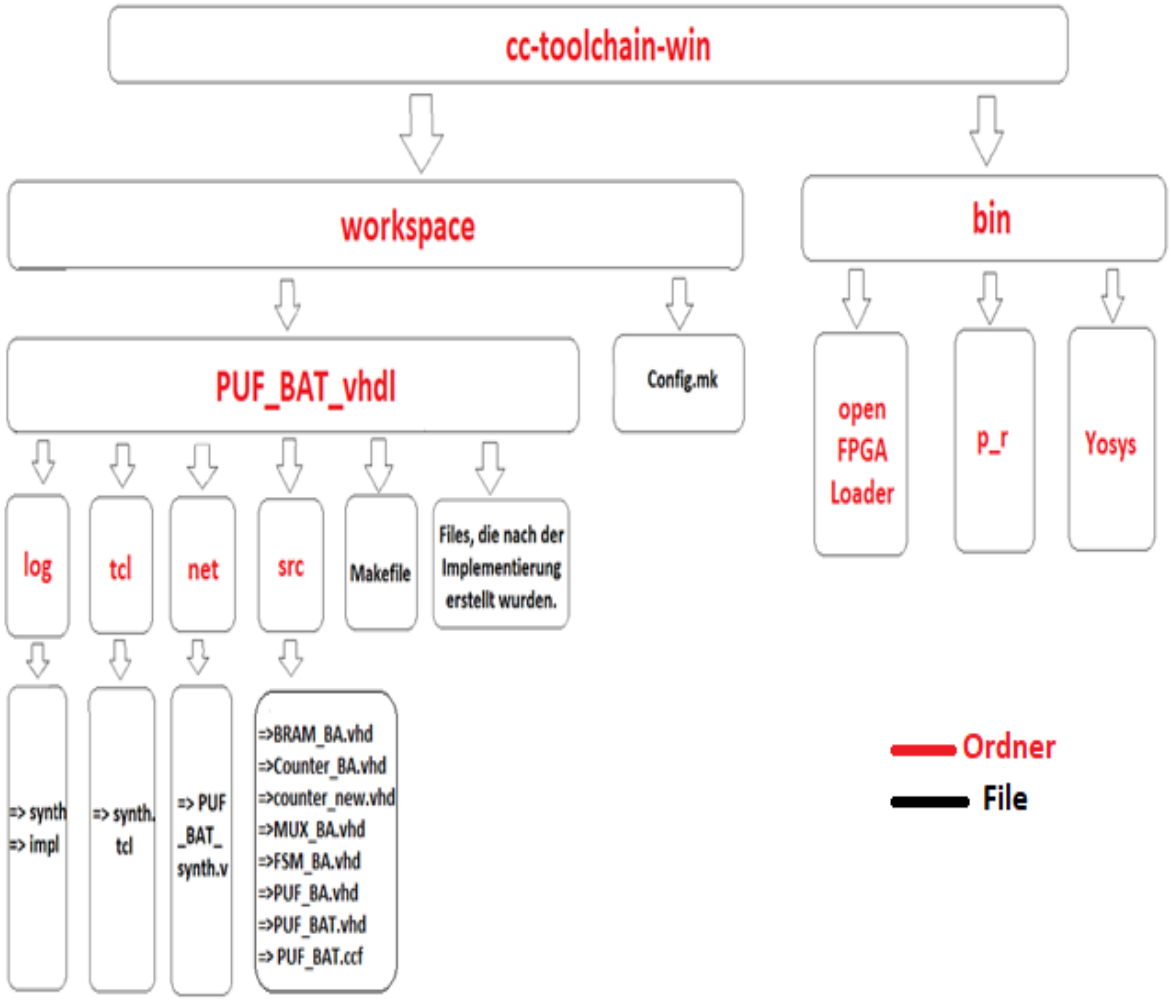


Abbildung 24: Baumstruktur des Unterverzeichnis cc-tool-win

6. Beschreibung des Schaltungsentwurfs

In diesem Teil der Arbeit soll der Schaltungsentwurf beschrieben werden, mit dem ein BRAM instanziiert und ausgelesen und die Daten über eine UART Schnittstelle versendet werden sollen. Zunächst wir die Struktur des Entwurfs mit Hilfe eines Blockdiagramms erläutert. Anschließend werden die einzelnen Module, die in VHDL entworfen worden sind, näher beschrieben. Der vollständige VHDL Source-Code ist im Anhang beigefügt.

6.1 Blockdiagramm

Die Struktur des Schaltungsentwurfs ist im Blockdiagramm in Abbildung 27 dargestellt. Als zentrales Element befindet sich am linken Rand des Blockschaltbildes eine Instanz eines BRAMS. Die Adresse, welche bestimmt, welches Datenwort des BRAMS ausgelesen werden soll, wird durch einen Coutner generiert. Ein aus dem BRAM ausgelesenen Datenwort besitzt eine Wortbreite von 18Bit, während über die UART Schnittstelle nur 8 Bit bzw. Bytes übertragen werden können. Aus diesem Grund befindet sich zwischen dem BRAM Ausgang und dem Eingang des UART Moduls ein Multiplexer, der das Datenwort des SRAMS in Bytes unterteilt. Welches Byte des ausgelesenen BRAM Datenworts an den UART übergeben werden soll, entscheiden eine Zustandsmaschine, die am unteren Teil des Blockschaltbildes zu sehen ist. Sobald ein neues Byte am Ausgang des Mutliplexers anliegt, initiiert die Zustandsmaschine die Übermittlung des Bytes über die UART Schnittstelle. Wenn die UART Schnittstelle die Versendung des Bytes vollzogen hat, wird das nächste Byte des aktuellen Datenwortes angestoßen. Sobald das Datenwort vollständig versendet worden ist, löst die Zustandsmaschine die Inkrementierung der BRAM Adresse aus und liest das nächste Datenwort aus dem BRAM aus. Der Vorgang endet, sobald das BRAM vollständig ausgelesen und alle Daten über die UART Schnittstelle übertragen worden ist. Durch ein externes Signal kann der Vorgang von Neuem begonnen werden.

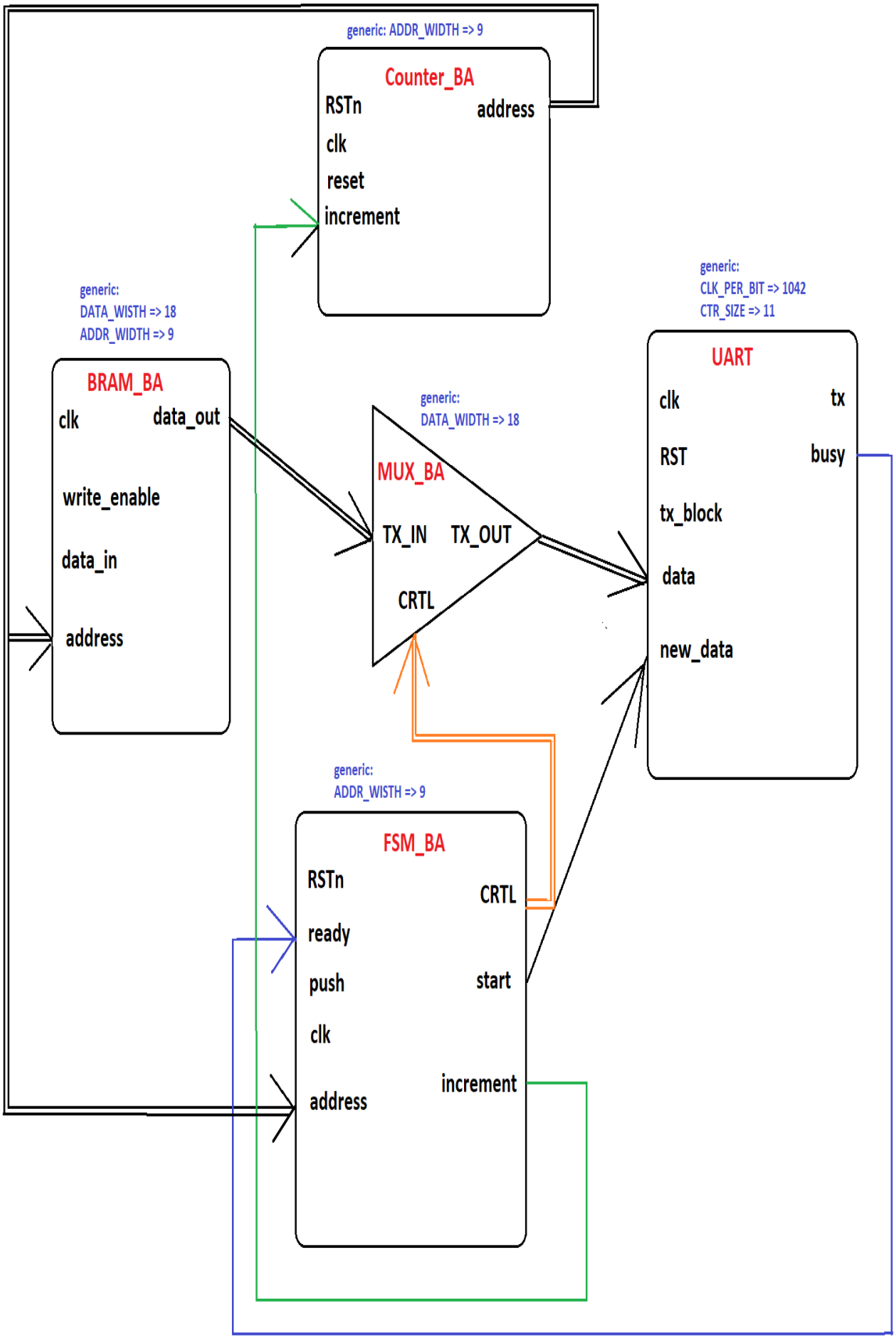


Abbildung 25: Blockdiagramm

6.2 Module

6.2.1 BRAM_BA

Bei dem Modul BRAM_BA handelt es sich um eine Instanz der BRAM Primitive, die einem im FPGA integrierten Block SRAM entspricht. Sie stellt die Physically Unclonable Function dar, die im Rahmen dieses Projekts untersucht werden soll.

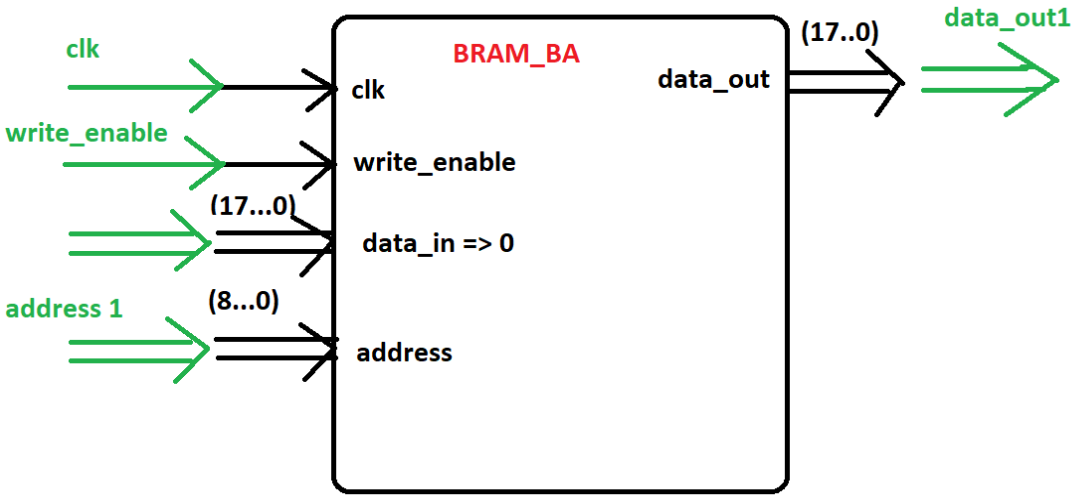


Abbildung 26:BRAM_BA Modul

Das BRAM_BA Modul besteht aus den vier Eingängen clk, write_enable, data_in, address und einem Ausgang data_out. Das Modul erhält an dem Port address die Adresse des Datenworts, das ausgelesen werden soll. Am Port data_in werden Daten angelegt, die in den Speicher hinein geschrieben werden sollen. Das Signal write_enable entscheidet, ob Daten in den BRAM hineingeschrieben (write_enable=1) oder ausgelesen (write_enable=0) werden sollen. Da in diesem Projekt nur Daten vom BRAM ausgelesen und keine Daten in den BRAM hineingeschrieben werden, wird das Signal write_enable auf Null gesetzt und am Port data_in liegen ebenfalls nur Nullen an. data_out entspricht dem Ausgangsport, an dem die addressierten Datenworte mit einer Wortbreite von 18 Bit mit der steigenden Flanke des Tatktsignals clk angelegt werden.

6.2.2 Counter_BA

Der Counter_BA hat die Aufgabe von 0 bis 511zu zählen. Der Zählerstand wird inkrementiert, wenn das Signal increment von der Zustandsmaschine gesetzt wird. Das Ausgangssignal address mit einer Bitbreite von 9 Bit entspricht der Adresse des BRAM Datenwortes, das als nächstes ausgelesen werden soll.

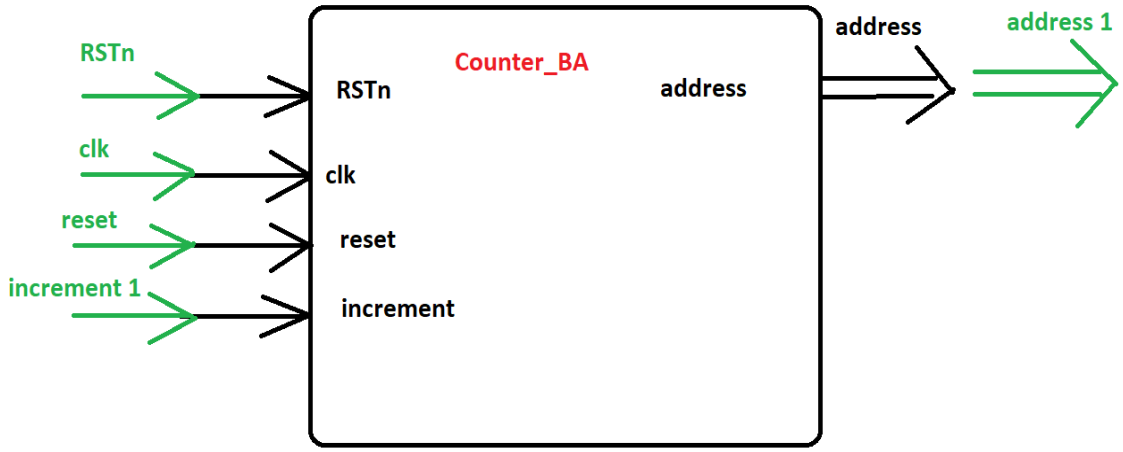


Abbildung 27: Counter_BA Modul

Der Port CLK entspricht dem Systemtakt und führt bei einer steigenden Flanke und entsprechender Ansteuerung zur Aktivierung des Zählers. Das Signal RSTn entspricht einem Low-Aktiven asynchronen Reset.

6.2.3 MUX_BA

Das Modul MUX_BA hat die Aufgabe das aktuell ausgelesene 18-Bit Datenwort des BRAMs, was am Eingang TX_IN anliegt, in drei Bytes zu unterteilen und über den Port TX_OUT an den Eingang der UART Schnittstelle anzulegen. Welcher Teil des BRAM Datenwortes weitergeleitet werden soll, wird durch den Steuereingang CRTL festgelegt. Der Port CTRL besitzt zwei Bits, wobei der Wert "00" zur Weiterleitung des niederwertigsten Bytes [7:0] und der Wert "01" zur Weiterleitung des nächst höherwertigen Bytes [15:8] führt. Beim Wert "10" werden die hochwertigsten zwei Bits ds Datenwortes [17:16] rechtsbündig weitergeleitet, wobei die MSB Bits des Bytes mit Nullen aufgefüllt werden.

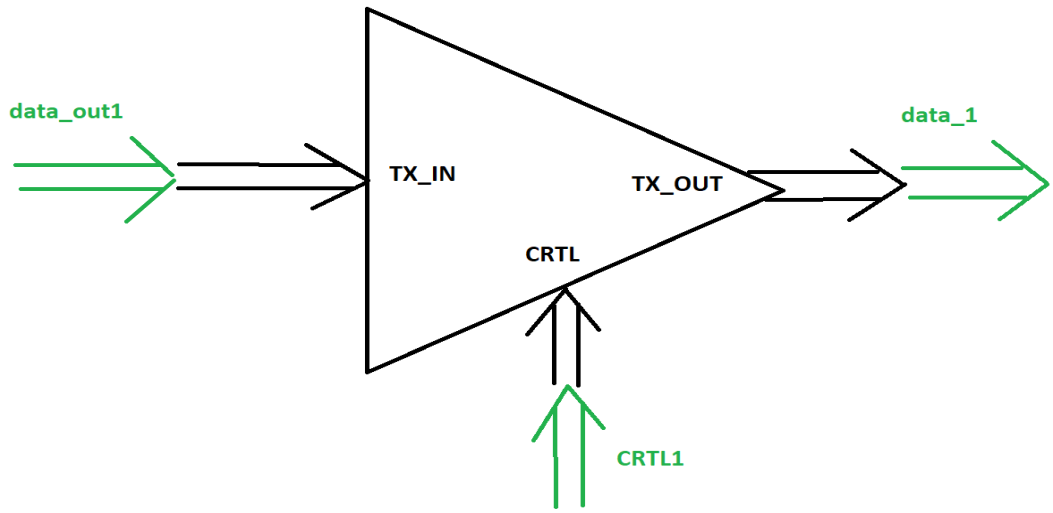


Abbildung 28: MUX_BA Modul

6.2.4 FSM_BA

Die Zustandsmaschine hat die Aufgabe die vollständige Auslesung des BRAM und Übermittlung der Daten über die UART Schnittstelle zu steuern. Wie in Abbildung 32 dargestellt ist, wartet die State Maschine zunächst auf den Knopfdruck „Push“. Dann werden das Modul MUX_BA und UART so angesteuert, dass die 3 Bytes, die dem Datenwort des BRAMS an der aktuellen Adresse entsprechen, übertragen werden. Die Versendung eines Bytes über die UART Schnittstelle geschieht mit Hilfe des Signals start. Die vollständige Versendung eines Bytes über die UART Schnittstelle signalisiert der UART über das Signal ready. Nach der Übertragung des dritten Bytes wird dem Modul Counter_BA das Signal gegeben, die Adresse zu inkrementieren. Wenn alle 512 Adressen abgearbeitet worden sind, geht die State Maschine in den Ausgangszustand „idle“.

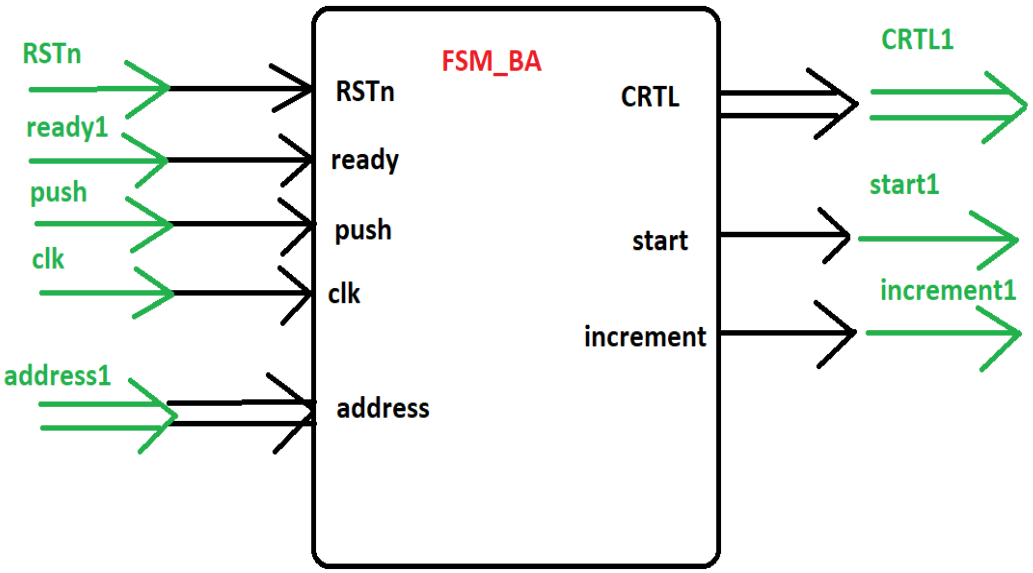


Abbildung 29: FSM_BA Modul

Das FSM_BA Modul besitzt fünf Eingänge RSTn, ready, push, clk und address und die drei Ausgänge CRTL, start und increment.

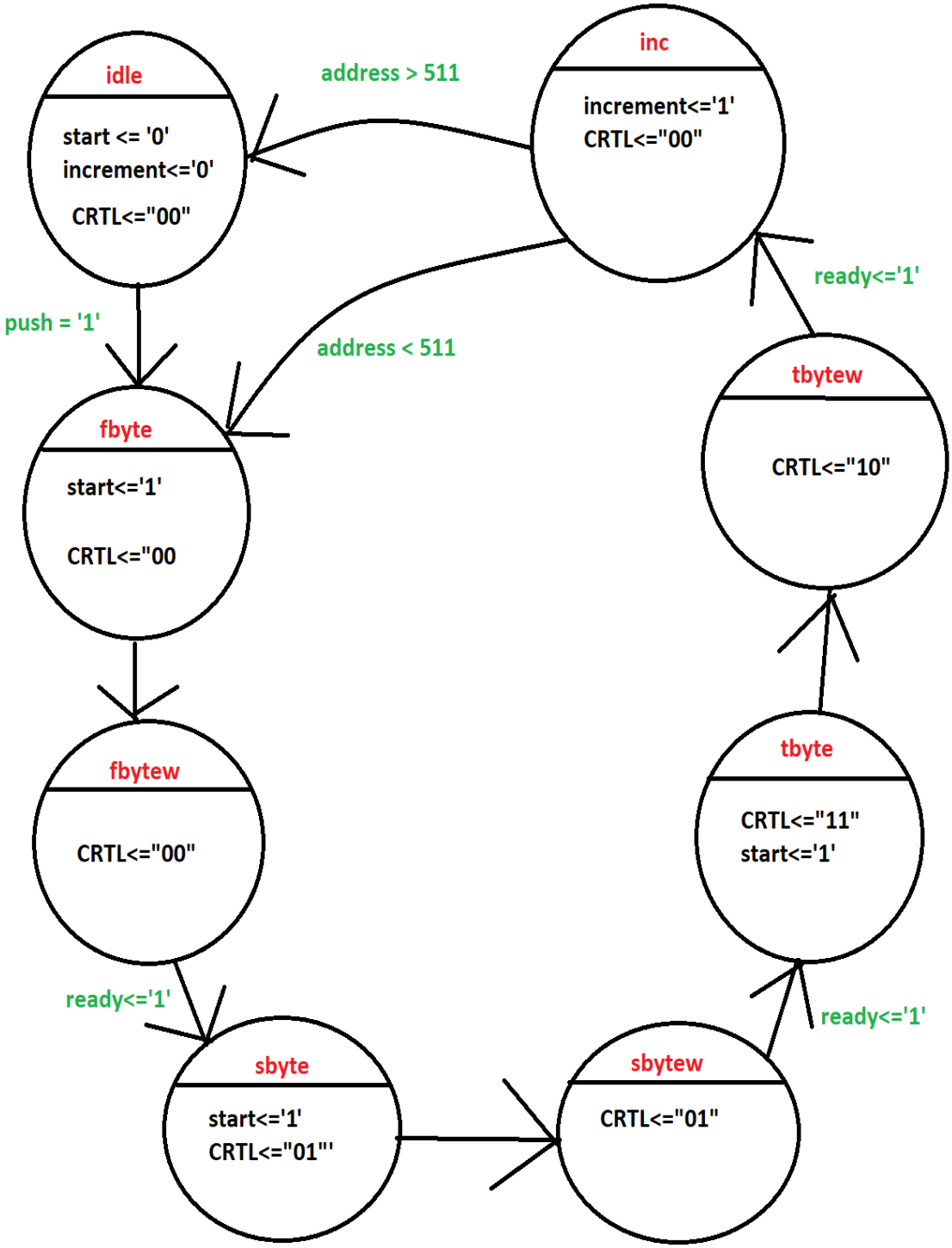


Abbildung 30:Zustandsdiagramm.

Das Zustandsdiagramm in Abbildung 32 stellt die Zustände der Zustandsmaschine grafisch dar. Der initiale Zustand ist idle. Nachdem der PUSH-Button gedrückt wurde, geht die Zustandsmaschine in den zweiten Zustand fbyte über, in dem für einen Takt das Signal start gesetzt, um das niederwertigste Byte des aktuellen Datenwort zu versenden. Der Zustand wird danach gewechselt und im fbytew Zustand wird auf die vollständige Versendung des Bytes gewartet. Wenn ready =`1´, wechselt die Zustandsmaschine in den Zustand sbyte, um die Versendung des nächst höherwertigen Bytes zu

initiieren. Der Zustand wird danach gewechselt und im sbytew Zustand wird wieder auf die vollständige Versendung des Bytes gewartet. Wenn ready =`1´, wechselt die Zustandsmaschine in den Zustand tbyt und initiiert die Versendung des dritten Bytes des aktuellen Datenworts. Der Zustand wird danach auf tbytew gewechselt und im tbytew Zustand wird drauf gewartet, dass das dritte Byte vollständig versendet worden ist. Wenn ready <=`1´, wechselt die Zustandsmaschine in den Zustand inc und löst dort die Inkrementierung der Adresse aus. Im Zustand inc wird verglichen, ob die aktuelle Adresse kleiner oder grösser als 511. Wenn die Adresse > 511 ist, ist der Auslesevorgang beendet und die Zustandsmaschine wechselt in den Zustand idle. Wenn die Adresse < 511 ist, ist der Auslesevorgang noch nicht vollständig vollzogen und die Zustandsmaschine wechselt in den Zustand fbyte.

6.2.5 Serial_tx

Das Modul serial_tx führt die Versendung serieller Daten nach dem UART Standard durch. Es beinhaltet eine Zustandsmaschine und ein Schieberegister. Da das Modul aus dem Mojo Projekt übernommen worden ist, wird es an dieser Stelle nicht weiter beschrieben. Im Kontext des Mojo Boards, wird das Modul verwendet, um den Datenaustausch zwischen einem FPGA und einem Mikrocontroller auf dem Mojo Board über eine UART Schnittstelle zu implementieren. Das Modul serial_tx enthält 8 Bits vom Modul MUX_BA und überträgt über den Ausgangspin tx die Daten an das USBUART Modul, welches per PMOD Stecker an das Gatemate Evaluation-Board angeschlossen worden ist.

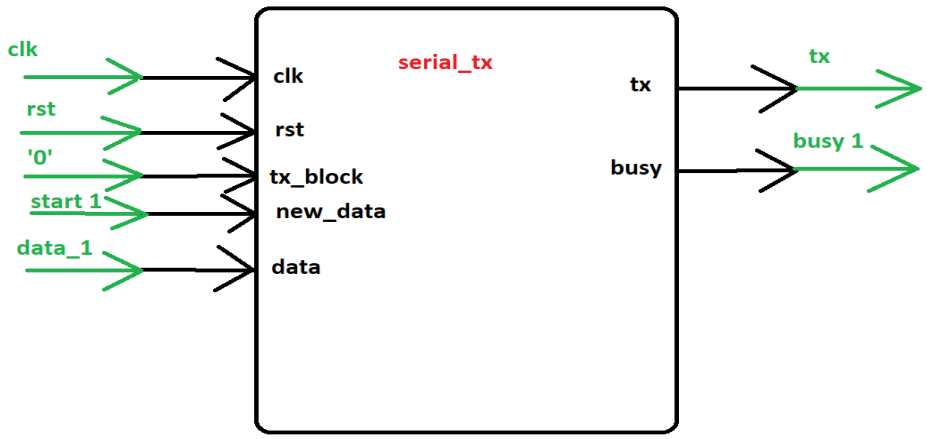


Abbildung 31: serial_tx Modul

Das serial_tx Modul besitzt die fünf Eingänge clk, rst, tx_block, new_data und data und die zwei Ausgänge tx und busy. Gestartet wird der Sendevorgang über das Signal new_data. Solange das Signal busy gesetzt ist, ist das Modul mit der Versendung eines Bytes beschäftigt.

6.2.6 PUF_BA

Der PUF_BA besteht aus den fünf Modulen Counter_BA, BRAM_BA, FSM_BA, MUX_BA und serial_tx. Das Modul hat die drei Eingänge clk, RSTn und push und den Ausgang tx. An den Port clk wird dabei der Systemtakt angelegt. RSTn entspricht einem asynchronen low-aktiven Reset und push entspricht einem entprellten Signal, mit dem der Nutzer über einen Taster die Auslesung des BRAM starten kann. Die UART Daten liegen dann am Signal tx an. Ursprünglich entsprach dieses Modul dem Toplevel. Es wurde jedoch eine weitere Hierarchieebene eingeführt, um ein weiteres Modul für die Entprellung des Tasters zu integrieren.

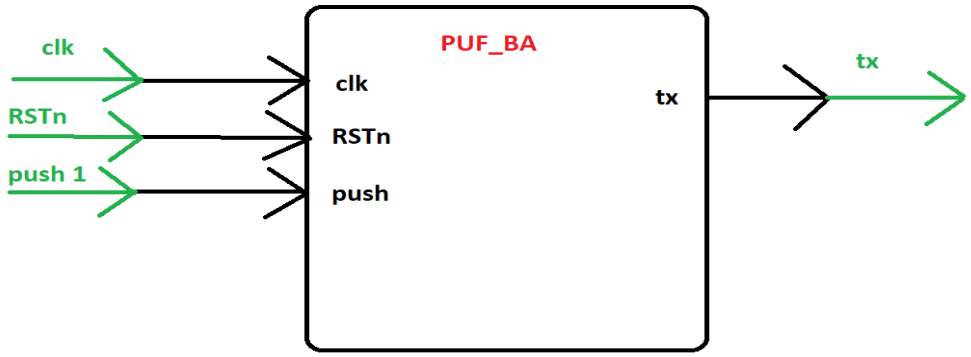


Abbildung 32: PUF_BA Modul

6.2.7 counter_neu

Das counter_neu Modul wurde für die Entprellung des Tastersignals push erstellt und besitzt die drei Eingänge clk, push_button_in und RSTn und den Ausgang push_button_out. Wenn der Taster gedrückt wird, der an das Signal push_button_in angeschlossen ist, wird das Signal push_button_out unabhängig vom Zustand des Signals push_button_in für mindesten 500 Takte auf high gehalten.

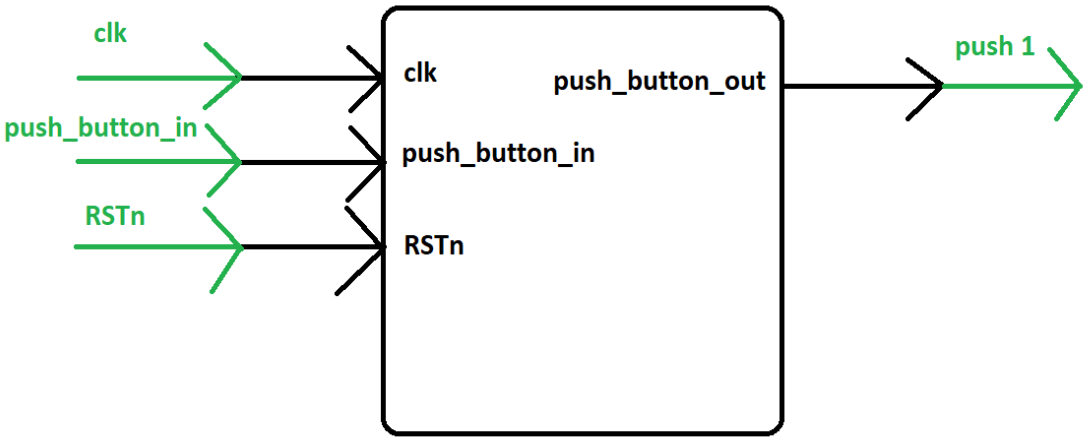


Abbildung 33: counter_neu Modul

6.2 PUF_BAT

PUF_BAT entspricht dem Toplevel Modul des Schaltungsentwurfs und besteht aus den zwei Modulen PUF_BA und counter_neu. Es hat die drei Eingänge clk, RSTn und push und den Ausgang tx.

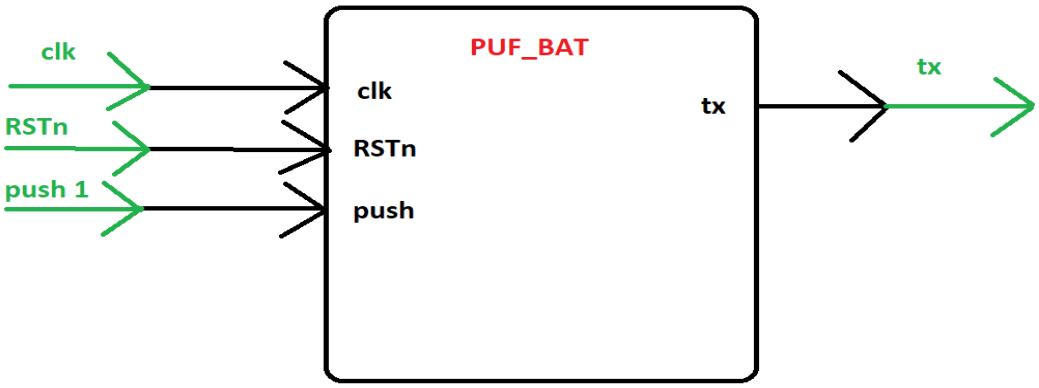


Abbildung 34: PUF_BAT Modul

6.3 Versuchsbeschreibung

6.3.1 Experiment 1

Nach der Verifikation der grundsätzlichen Funktionalität, wird das Block-RAM angesteuert, und die Daten werden via UART übertragen, Zunächst wird mit Initialisierungswerten im VHDL-Code synthetisiert und implementiert. Schließlich werden Daten übertragen und mit den Initialisierungswerten verglichen.

.

6.3.2 Experiment 2

Das BRAM wird als PUF verwendet. Außerdem werden die Initialisierungswerte aus der Post-Synthese Netzliste entfernt, die sich in der Datei PUF_BAT_synth.v und im Unterverzeichnis **/net** befindet. Abb. 35 zeigt die Netzliste mit den Initialisierungswerten.

```
CC_IBUF _277_ (
  .I(clk),
  .Y(_052_)
);
CC_IBUF _278_ (
  .I(push_button_in),
  .Y(\uuu.push_button_in )
);
CC_OBUF _279_ (
  .A(\u.tx ),
  .O(tx)
);
CC_BRAM_20K #(
  .A_CLK_INV(1'h0),
  .A_RD_WIDTH(32'd20),
  .A_WR_MODE("NO_CHANGE"),
  .A_WR_WIDTH(32'd20),
  .B_CLK_INV(1'h0),
  .B_RD_WIDTH(32'd0),
  .B_WR_MODE("WRITE_THROUGH"),
  .B_WR_WIDTH(32'd0),
  .INIT_00(320'h00fea00fea00fea00fea00fea00fea00fea00fea00fea00fea00fea00fea00fea00fea00fea),
  .INIT_01(320'h00fea00fea00fea00fea00fea00fea00fea00fea00fea00fea00fea00fea00fea00fea00fea),
  .INIT_02(320'h00fea00fea00fea00fea00fea00fea00fea00fea00fea00fea00fea00fea00fea00fea00fea),
  .INIT_03(320'h00fea00fea00fea00fea00fea00fea00fea00fea00fea00fea00fea00fea00fea00fea00fea),
  .INIT_04(320'h00fea00fea00fea00fea00fea00fea00fea00fea00fea00fea00fea00fea00fea00fea00fea),
  .INIT_05(320'h00fea00fea00fea00fea00fea00fea00fea00fea00fea00fea00fea00fea00fea00fea00fea),
  .INIT_06(320'h00fea00fea00fea00fea00fea00fea00fea00fea00fea00fea00fea00fea00fea00fea00fea),
  .INIT_07(320'h00fea00fea00fea00fea00fea00fea00fea00fea00fea00fea00fea00fea00fea00fea00fea),
  .INIT_08(320'h00fea00fea00fea00fea00fea00fea00fea00fea00fea00fea00fea00fea00fea00fea00fea),
  .INIT_09(320'h00fea00fea00fea00fea00fea00fea00fea00fea00fea00fea00fea00fea00fea00fea00fea),
  .INIT_0A(320'h00fea00fea00fea00fea00fea00fea00fea00fea00fea00fea00fea00fea00fea00fea00fea),
  .INIT_0B(320'h00fea00fea00fea00fea00fea00fea00fea00fea00fea00fea00fea00fea00fea00fea00fea),
  .INIT_0C(320'h00fea00fea00fea00fea00fea00fea00fea00fea00fea00fea00fea00fea00fea00fea00fea),
```

Abbildung 35: Die Instanziierung des BRAM-Moduls in der Netzliste.

Nachdem die **INIT_Parameter** manuell aus der synthetisierten Netzliste entfernt worden sind, wird die Post-Synthese Netzliste implementiert. Danach wird das Board 10-mal vom Strom getrennt, neu mit dem Bistream programmiert und die Daten aus dem RAM übertragen.

6.4 Ergebnisse

6.4.2 Experiment 1

Am Rechner, der über die USB-UART-Brücke angeschlossen ist, werden wie erwartet 1536 Bytes empfangen. Werden jeweils drei aufeinanderfolgende Bytes zu einem Wort zusammengefügt, ergeben sich die im VHDL-Code als Initialisierungswerte angegebenen 18-Bit-Wörter. Die Ansteuerung des Block-RAMs sowie die Datenübertragung an den PC sind damit verifiziert.

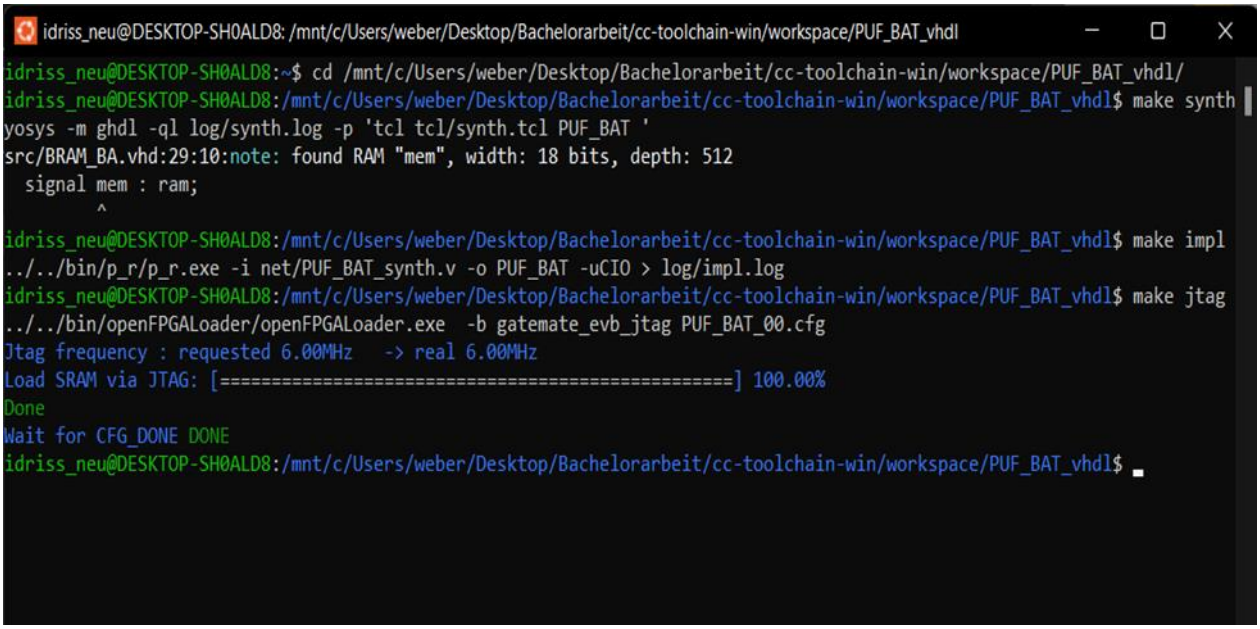


Abbildung 36: Screenshot der erfolgreichen Durchführung.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
10010010	00000010	00000000	10010010	10101010	00000000	00101001	00100010	00000010	10111111	00100000	00000000	10101111	00100010	00000000	10010110
00101000	00000010	00110111	00000010	00000010	10011001	10001010	00000010	10111001	00100000	00000010	00000000	00000000	00000000	00000000	10000000
00000000	10100001	10000000	00000010	00110000	00000000	00000010	00001000	00100000	00000000	10000111	10000000	00000000	10011001	00001010	00000000
10001000	10101000	00000010	00111111	10100010	00000010	10101110	10001010	00000000	10101011	10101000	00000010	10111100	10100010	00000010	00111110
10101010	00000010	00110111	00100010	00000010	00010111	10101010	00000010	10101110	00100000	00000000	00010001	00100000	00000000	10100111	00000000
00000000	10001100	10000000	00000010	00011011	10101010	00000010	10010100	00001000	00000010	10101000	00001000	00000000	10110100	10000000	00000010
10111010	10000010	00000000	00000111	10101010	00000000	00111101	10101010	00000000	10100111	10001010	00000010	10111011	00101000	00000010	00010101
10101010	00000010	00011110	00000000	00000010	10101110	10101010	00000000	00011111	10000000	00000000	00011111	00001000	00000000	00100010	10101010
00000000	00010100	00000000	00000000	10111001	00001000	00000000	10101100	00001000	00000000	10000100	00000000	00000000	00111001	10000000	00000010
00100000	00100000	00000000	10011110	10100000	00000000	00110111	00101010	00000010	10010101	00101000	00000010	10111010	00101000	00000010	00011111
00101000	00000000	10111110	10100010	00000010	10111111	10100000	00000010	00111111	00100000	00000010	00000001	00000000	00000000	00011100	00100000
00000010	10000000	00001000	00000010	10010001	00000000	00000000	00100010	00101010	00000000	10100000	10100000	00000000	00110100	00100010	00000010
00010110	10001000	00000000	00011111	00100000	00000000	10000111	00100010	00000000	10000011	10101010	00000000	00011111	10101000	00000000	10111100
10001010	00000000	10101111	00101010	00000000	10111111	10101010	00000010	00011010	00101010	00000000	10010001	00000000	00000000	00101110	10001010
00000010	10111001	00001010	00000000	00010001	00001000	00000000	00010010	00101000	00000010	00010000	00000000	00000010	10010010	10100010	00000010
00100000	10001010	00000000	10011001	00100010	00000010	00110010	00001000	00000000	10101111	10100010	00000010	00011111	10000010	00000000	10111111
00100000	00000000	10110101	00100010	00000010	10000111	10001010	00000000	10100110	00001000	00000010	00010100	10001000	00000000	10000100	00100000
00000000	10010100	00000000	00000010	00000000	00000010	00000010	00111001	00000000	00000000	10001010	00001000	00000000	00001111	00000010	00000010
10000000	10001010	00000010	10011001	10001010	00000000	10100011	00101010	00000010	10101110	10101010	00000010	00101110	00101010	00000010	10101010
10001010	00000010	10110111	10001010	00000000	10011010	10100010	00000000	10111110	00000010	00000000	10000001	00000000	00000000	10000100	10000010

Abbildung 37: die übertragenen Daten.

6.4.2 Experiment 2

Um abschätzen zu können, ob sich das betrachtete Block-RAM als PUF eignen würde, wird für die Adressen 1 bis 511 der in den zehn Versuchen am häufigsten auftretende Wert als erwarteter Wert definiert und anschließend für jeden Versuch der Hamming-Abstand zum erwarteten Wert bestimmt.

6.4.2.1 Betrachtung auf Wortebene

Wie Abbildung 36 zeigt, stellen sich über alle 512 Wörter und zehn Versuche Hamming-Abstände zwischen 0 und 4 ein; dabei entspricht die größte auftretende Distanz einem Anteil von 4/18≈22% abweichenden Bits. Über alle zehn Versuche wurden 5120 Wörter ausgelesen, Tabelle 3 zeigt die Häufigkeit der auftretenden Hamming-Abstände zum jeweiligen erwarteten Wert.

Hamming-Abstand	Anzahl	Anteil
0	3921	77%
1	1030	20%
2	160	3%
3	8	0,2%
4	1	≈0%

Tabelle 3: Häufigkeit der auftretenden Hamming-Abstände

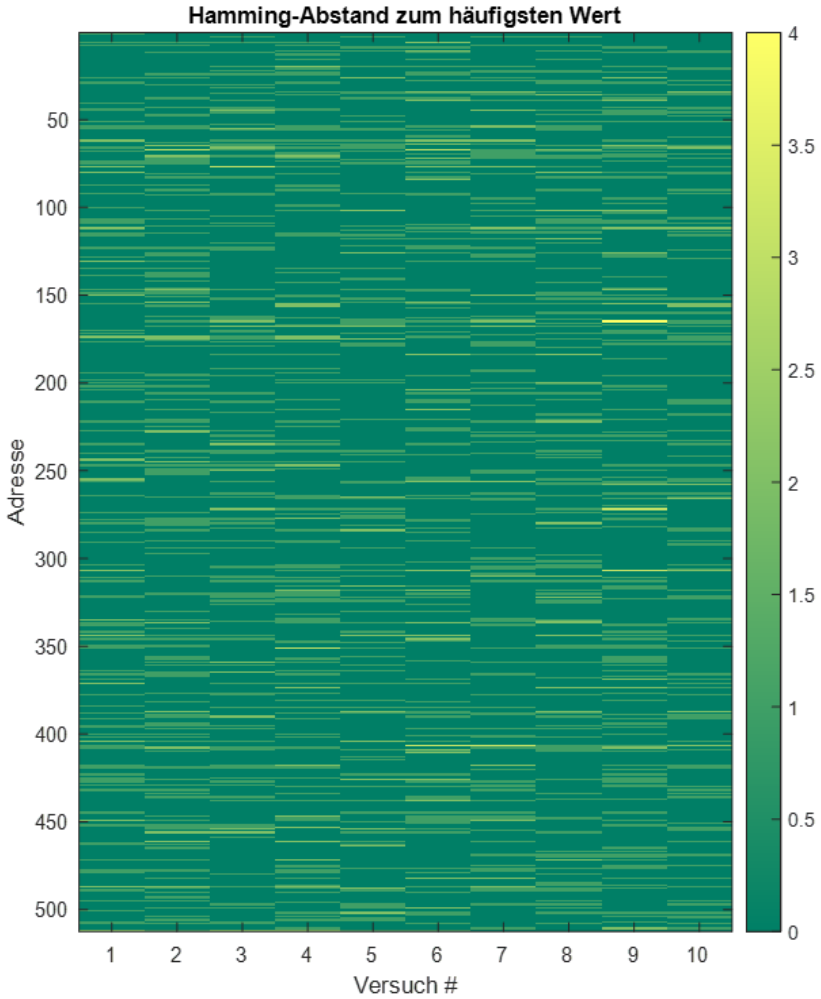


Abbildung 38: Hamming-Abstand zum häufigsten Wert.

Abbildung 36 zeigt die Häufigkeit der auftretenden Hamming-Abstände innerhalb der einzelnen Versuche. In jedem Versuch lag der Anteil der Wörter mit einer Hamming-Distanz von 0 zum erwarteten Wert bei mindestens 72,5% und höchstens 80%. Mit einer Hamming-Distanz von 0 oder 1 werden in jedem Versuch mindestens 95% und höchstens 98% der Wörter abgedeckt.

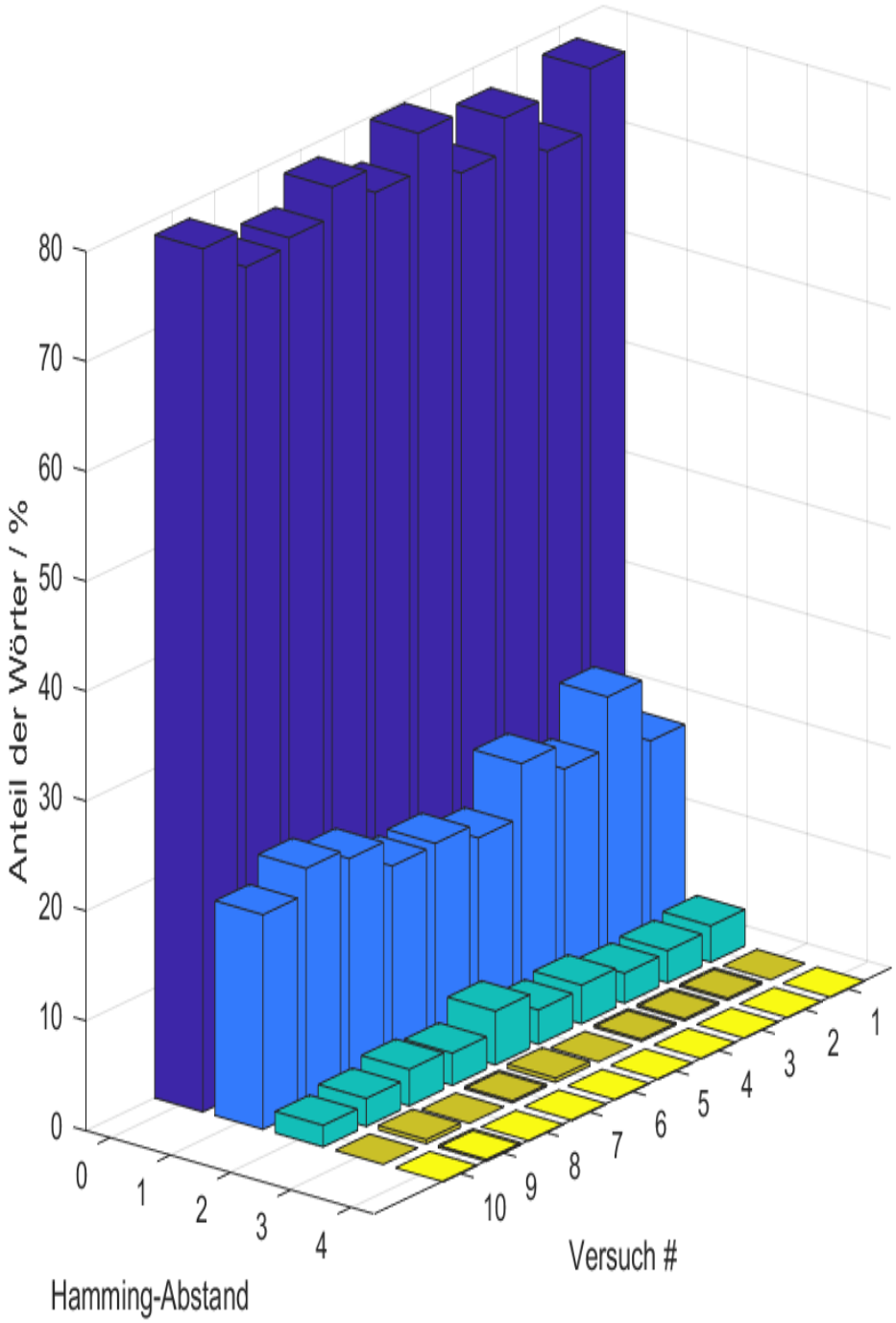


Abbildung 39: Hamming-Abstand pro Versuch.

6.4.2.2 Betrachtung auf Bitebene

In den zehn Versuchen wurden jeweils mindestens 32,45% und höchstens 32,85% 1-Bits ausgelesen. Insgesamt liegt die Wahrscheinlichkeit für das Auftreten eines 0-Bits damit höher als die Wahrscheinlichkeit für das Auftreten eines 1-Bits.

Von den $512 \cdot 18 = 9216$ Bits in den betrachteten 512 Wörtern des BRAMs nehmen 8657, was einem Anteil von 94% entspricht, in allen zehn Versuchen den gleichen Wert an.

Wird der Vektor aus den pro Bit am häufigsten auftretenden Werten als erwarteter Bitvektor definiert, ergeben sich in den 10 Versuchen die in Tabelle 4 gezeigten Hamming-Distanzen.

Versuch	Abs. Hamming-Abstand	Rel. Hamming-Abstand
1	132	1,4%
2	151	1,6%
3	129	1,4%
4	145	1,6%
5	121	1,3%
6	146	1,6%
7	130	1,4%
8	144	1,6%
9	151	1,6%
10	115	1,2%

Tabelle 4: Hamming-Distanzen.

7. Fazit

Ziel dieser Arbeit war die Implementierung einer SRAM-basierten Physically Unclonable Function mit einem FPGA und dem Open-Source Synthesewerkzeug Yosys. In der Einleitung wurde ein Überblick auf diese Arbeit und die Firma Cologne Chip gegeben. Im zweiten Kapitel wurde der PUF Begriff definiert und die Funktionsweise von SRAM basierten PUFs erläutert. Anschließend wurden in den Kapiteln 4 und 5 die verwendete Hardware und Software vorgestellt. Im 6 Kapitel wurden die Struktur des Schaltungsentwurfs und die Funktion der einzelnen Module vorgestellt. Zuletzt wurden die Funktionsweise des Schaltungsentwurfs durch praktische Versuche mit vorinitialisierten BRAM nachgewiesen und erste Versuche bei zufälligen Speicherinhalten im BRAM durchgeführt. Um festzustellen, ob die im Gatemate FPGAs vorhandenen BRAMs als PUFs verwendet werden können, sind weitere statistische Auswertung unter Variation der Versorgungsspannung und der Betriebstemperatur notwendig. Diese Analyse kann in einer nachfolgenden Bachelor-Arbeit durchgeführt werden.

8. Literaturverzeichnis

1. Physically Unclonable Functions: a Study on the State of the Art and Future Research Directions. Roel Maes, Ingrid Verbauwhede. Seite 2.
2. Physically Unclonable Functions: a Study on the State of the Art and Future Research Directions. Roel Maes, Ingrid Verbauwhede Seite 10, Seite 11.
3. Physically Unclonable Functions: a Study on the State of the Art and Future Research Directions. Roel Maes, Ingrid Verbauwhede Seite15, Seite 16.
4. Physically Unclonable Functions: a Study on the State of the Art and Future Research Directions. Roel Maes, Ingrid Verbauwhede Seite 15, Seite 16.
5. Guajardo, J., Kumar, S.S., Schrijen, G.J., Tuyls, P.: FPGA intrinsic PUFs and their use for IP protection. In: Cryptographic Hardware and Embedded Systems Workshop, LNCS, vol. 4727, pp. 63-80 (2007)
6. Herschel, Sir William J.: The origin of nger-printing. Oxford University Press (1916)
7. Guajardo, J., Kumar, S.S., Schrijen, G.J., Tuyls, P.: Physical unclonable functions and publickey cry to for FPGA IP protection. In: Field Programmable Logic and Applications, 2007, pp. 189-195 (2007)
8. Holcomb, D.E., Burleson, W.P., Fu, K.: Power-up sram state as an identifying ngerprint and source of true random numbers. IEEE Trans. Comput. 58(9), 1198-1210 (2009)
9. Cologne Chip. (April 2022) Köln. GateMateTM FPGA Evaluation Board and Schematics S 12 https://colognechip.com/docs/ds1003-gatemate1-evalboard-3v1-latest.pdf
10. Cologne Chip. (April 2022) Köln. GateMateTM FPGA Evaluation Board and Schematics S13 https://colognechip.com/docs/ds1003-gatemate1-evalboard-3v1-latest.pdf
11. Cologne Chip. (April 2022) Köln. GateMateTM FPGA Evaluation Board and Schematics S 16 https://colognechip.com/docs/ds1003-gatemate1-evalboard-3v1-latest.pdf
12. Cologne Chip. (April 2022) Köln. GateMateTM FPGA Evaluation Board and Schematics S 15 https://colognechip.com/docs/ds1003-gatemate1-evalboard-3v1-latest.pdf
13. Cologne Chip. (April 2022) Köln. GateMateTM FPGA Evaluation Board and Schematics S 21 https://colognechip.com/docs/ds1003-gatemate1-evalboard-3v1-latest.pdf
14. Cologne Chip. (April 2022) Köln. GateMateTM FPGA Evaluation Board and Schematics S 23 https://colognechip.com/docs/ds1003-gatemate1-evalboard-3v1-latest.pdf
15. Cologne Chip. (April 2022) Köln. GateMateTM FPGA Evaluation Board and Schematics S 28 https://colognechip.com/docs/ds1003-gatemate1-evalboard-3v1-latest.pdf
16. Cologne Chip. (April 2022) Köln. GateMateTM FPGA Evaluation Board and Schematics S 29 https://colognechip.com/docs/ds1003-gatemate1-evalboard-3v1-latest.pdf
17. Cologne Chip. (April 2022) Köln. GateMateTM FPGA Evaluation Board and Schematics S 30 https://colognechip.com/docs/ds1003-gatemate1-evalboard-3v1-latest.pdf

18. Cologne Chip. (April 2022) Köln. GateMateTM FPGA Evaluation Board and Schematics S 32
https://colognechip.com/docs/ds1003-gatemate1-evalboard-3v1-latest.pdf

19. Cologne Chip. (April 2022) Köln. GateMateTM FPGA Evaluation Board and Schematics S 34
https://colognechip.com/docs/ds1003-gatemate1-evalboard-3v1-latest.pdf

20. 1. Physically Unclonable Functions: a Study on the State of the Art and Future Research Directions. Roel Maes, Ingrid Verbauwhede. Seite 17.

21. Cologne Chip. (April 2022) Köln. GateMateTM FPGA Evaluation Board and Schematics S 27
https://colognechip.com/docs/ds1003-gatemate1-evalboard-3v1-latest.pdf

22. Cologne Chip. (April 2022) Köln. GateMateTM FPGA Evaluation Board and Schematics S 35
https://colognechip.com/docs/ds1003-gatemate1-evalboard-3v1-latest.pdf

23. Cologne Chip. (April 2022) Köln. GateMateTM FPGA Evaluation Board and Schematics S 36
https://colognechip.com/docs/ds1003-gatemate1-evalboard-3v1-latest.pdf

24. Pmod USBUART: USB-zu-UART-Schnittstelle, Digilent A National Instruments Company
https://shop.trenz-electronic.de/de/24242-Pmod-USBUART-USB-zu-UART-Schnittstelle4

25. Very High Speed Integrated Circuit Hardware Description Language
https://de.wikipedia.org/wiki/Very_High_Speed_Integrated_Circuit_Hardware_Description_Language

26. Pmod BTN: 4 User Pushbuttons
https://www.mirifica.pt/de/trenz-electronic-shop/pmod-btn-4-userpushbuttons_100217_1371/

27. Ubuntu – Wikipedia, 21. April 2022
https://de.wikipedia.org/wiki/Ubuntu

28. Yosys, 9 mai 2022
https://fr.wikipedia.org/wiki/Yosys

29. Very High Speed Integrated Circuit Hardware Description Language, 2. Juni 2021
https://de.wikipedia.org/wiki/Very_High_Speed_Integrated_Circuit_Hardware_Description_Language

30. Gatemate1-Datasheet-Latest, 22.Mai.2022
https://colognechip.com/docs/ds1001-gatemate1-datasheet-latest.pdf

31. Payato.com: UART RX Circuit - Bing images

32. UART-Kommunikationsprotokoll - Wie funktioniert es? - Codrey Electronics | IWOFR

9 Anhang

A.1 Codes

A.1.1 BRAM_BA

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
use IEEE.NUMERIC_STD.ALL;
-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;
entity BRAM_BA is
      generic (
                  DATA_WIDTH : natural := 18;
            ADDR_WIDTH : natural := 9
            );
       port (
            clk:          in  std_logic;
             data_in:      in  std_logic_vector (DATA_WIDTH-1 downto 0);
             address:      in  std_logic_vector (ADDR_WIDTH-1 downto 0);
             write_enable: in  std_logic; d
            ata_out:     out std_logic_vector (DATA_WIDTH-1 downto 0)
            );
end BRAM_BA;
architecture Behavioral of BRAM_BA is
type ram is array(0 to (2**ADDR_WIDTH)-1) of std_logic_vector(DATA_WIDTH-1 downto
0);
signal mem : ram :=
       (B"101010101010111111", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010",
```

```
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
```

```
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
```

```
 B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
B"101010101010101010", B"101010101010101010", B"101010101010101010",
```

```
B"1010101010101010", B"1010101010101010", B"1010101010101010",
B"1010101010101010", B"1010101010101010", B"1010101010101010",
B"1010101010101010", B"1010101010101010", B"1010101010101010",
B"1010101010101010");
begin
process (clk)
begin
       if rising_edge(clk) then
            if (write_enable = '1') then
             mem(to_integer(unsigned(address))) <= data_in;
             else
            data_out <= mem(to_integer(unsigned(address)));
             end if;
       end if;
end process;
end Behavioral;
```

A.1.2 Counter_BA

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.ALL;
entity Counter_BA is
	generic (
		ADDR_WIDTH : natural := 9
		);
		 Port (
			RSTn:			in std_logic;
			clk:			in std_logic;
			reset:			in std_logic; --high aktiv
			address:		out  std_logic_vector (ADDR_WIDTH-1 downto
0);
			increment:		in std_logic
			);
end Counter_BA;
architecture Behavioral of Counter_BA is
signal counter_up: unsigned(8 downto 0);
begin
-- UP COUNTER
process(clk,RSTn)
begin
		if RSTn='0' then
				counter_up <= (others=>'0');
			elsif rising_edge(clk) then
			 (reset = '1') then
			counter_up <= (others=>'0');
			elsif (increment='1') then
			counter_up <= Counter_up + "1";
			end if;
		 end if;
end process;
		address <= std_logic_vector(counter_up);
end Behavioral;
```

A.1.3 MUX_BA

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity MUX_BA is
	generic (
		DATA_WIDTH : natural := 18;
		 ADDR_WIDTH : natural := 9
		);
	port(
		TX_IN :	in std_logic_vector (DATA_WIDTH-1 downto 0);
		TX_OUT :	out std_logic_vector (7 downto 0);
		CRTL :		in std_logic_vector(1 downto 0)
		);
end MUX_BA;
architecture rtl of MUX_BA is
  -- declarative part: empty
begin
		p_MUX_BA : process(CRTL,TX_IN)
begin
	case CRTL is
		when "00"	=> TX_OUT <= TX_IN(7 downto 0);
		when "01"	=> TX_OUT <= TX_IN(15 downto 8);
		when "10"	=> TX_OUT <= "000000" & TX_IN(17 downto 16);
		when others => TX_OUT <= "00000000";
	end case;
end process p_MUX_BA;
end rtl;
```

A.1.4 FSM_BA

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.ALL;
entity FSM_BA is
	generic (
	ADDR_WIDTH : natural := 9
	 );
	 port (
		clk:		in std_logic;
		RSTn:		in std_logic;
		ready:			in std_logic;
		address:	in  std_logic_vector (ADDR_WIDTH-1 downto 0);
		push:		in std_logic;
		CRTL :			out std_logic_vector(1 downto 0);
		start:			out std_logic;
		increment:	out std_logic
		);
end FSM_BA;
architecture behavioral of FSM_BA is
type state_type is (idle, fbyte, fbytew, sbyte, sbytew, tbyte, tbytew, inc);
signal current_s, next_s: state_type;
begin
process (clk,RSTn)

begin
	if RSTn='0' then
			current_s <= idle;
		elsif (rising_edge(clk)) then
			current_s <= next_s;	--state change.
		end if;
 end process;
--state machine process.
process (current_s, push, ready, address)
begin
	case current_s is
		when idle =>
			if(push ='1') then
				next_s <= fbyte;
			else
				next_s  <= idle;
			end if;
		when fbyte =>
				next_s<= fbytew;
		when fbytew =>
			if(ready ='1') then
				next_s <= sbyte ;
			else
				next_s <= fbytew;
			end if;
		when sbyte =>
				next_s <= sbytew;
```

```
            when sbytew =>
                  if(ready ='1') then
                        next_s <= tbyte;
                  else
                        next_s <= sbytew;
                  end if;
            when tbyte =>
                        next_s <= tbytew;
            when tbytew =>
                  if(ready ='1') then
                        next_s <= inc;
                  else
                        next_s <= tbytew;
                  end if;
            when inc =>
                  if  (unsigned(address) < "111111111") then
      next_s <= fbyte;
                  else
                        next_s <= idle;
                         end if;
end case;
end process;
process(current_s)
begin
      start <= '0';
      increment<= '0';
      CRTL<= "00";
      case current_s is
            when idle =>
                        start <= '0';
                        increment<= '0';
                              CRTL<= "00";
            when fbyte =>
                  start <= '1';
                  CRTL <= "00";
                  increment<='0';
            when fbytew =>
                  CRTL <= "00";
            when sbyte =>
                  start <= '1';
                  CRTL <= "01";
            when sbytew =>
                  CRTL <= "01";
            when tbyte =>
                  start <= '1';
                  CRTL <= "10";
            when tbytew =>
                  CRTL <= "10";
            when inc =>
                  increment<='1';
      end case;
end process;
end behavioral;
```

A.1.5 PUF_BA

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.All;
entity PUF_BA is
       generic (
            DATA_WIDTH : natural := 18;
            ADDR_WIDTH : natural := 9
             );
       Port (      clk,RSTn,push: in std_logic;
                    tx: out std_logic
                   );
end PUF_BA;
architecture Behavioral of PUF_BA is
component BRAM_BA is
      generic (
                          DATA_WIDTH : natural := 18;
                          ADDR_WIDTH : natural := 9
                    );
      port (
                          clk          : in  std_logic;
                           data_in     : in  std_logic_vector (DATA_WIDTH-1
downto 0);
                           address     : in  std_logic_vector (ADDR_WIDTH-1
downto 0);
                          write_enable : in  std_logic;
                          data_out     : out std_logic_vector (DATA_WIDTH-1
downto 0           );
end component;
component MUX_BA is
      generic (
            DATA_WIDTH : natural := 18
            );
      port (
            TX_IN              :in std_logic_vector (DATA_WIDTH-1 downto 0);
            TX_OUT       :out std_logic_vector (7 downto 0);
            CRTL         :in std_logic_vector(1 downto 0)
              );
end component ;
component Counter_BA is
      generic (
             ADDR_WIDTH : natural := 9
            );
      Port (
            RSTn                :in std_logic;
            clk                :in std_logic;
            reset              :in std_logic;
            address             :out  std_logic_vector (ADDR_WIDTH-1 downto
0);
            increment          :in std_logic
              );
end component ;
```

```
component FSM_BA is
	generic (
		 ADDR_WIDTH : natural := 9
		);
	port (
		clk		:in std_logic;
		RSTn		:in std_logic;
		ready		:in std_logic;
		address		:in  std_logic_vector (ADDR_WIDTH-1 downto 0);
		push		:in std_logic;
		CRTL		:out std_logic_vector(1 downto 0);
		start		:out std_logic;
		increment	:out std_logic
		);
end component ;
component serial_tx is
	generic (
		CLK_PER_BIT  : natural := 1042;
		CTR_SIZE     : natural := 11
		);
	port (
		clk		: in  std_logic;
		rst		: in  std_logic;
		tx		: out std_logic;
		tx_block	: in  std_logic;
		busy		: out std_logic;
		new_data	: in  std_logic;
		data		: in  std_logic_vector(7 downto 0)
	);
end component;
signal address1 : std_logic_vector (ADDR_WIDTH-1 downto 0);
signal data_out1 :std_logic_vector (DATA_WIDTH-1 downto 0);
signal CRTL1 : std_logic_vector (1 downto 0);
signal reset: std_logic;
signal data_in: std_logic_vector(17 downto 0);
signal increment1: std_logic;
signal ready1: std_logic;
signal start1: std_logic;
signal tx1: std_logic;
signal busy1: std_logic;
signal tx_block1: std_logic;
signal data_1: std_logic_vector(7 downto 0);
signal rst:std_logic;
begin
tx<=tx1;
ready1<= not busy1;
rst<=not RSTn;
reset<='0';
U:BRAM_BA generic map (   DATA_WIDTH => 18
                          )
                      port map (        data_in => (others => '0'),
                                       clk => clk,
                                       write_enable => '0',
                                       data_out => data_out1,
                                        address => address1
                                 );
```

```
UU:MUX_BA generic map (   DATA_WIDTH => 18
                                    )
                      port map(        TX_IN => data_out1,
                                       TX_OUT => data_1,
                                  CRTL => CRTL1
                                 );
UUU:Counter_BA generic map (  ADDR_WIDTH => 9
                                     )
                           port map (         clk => clk,
                                              reset => reset, --synchron
                                              address => address1,
                                                increment => increment1,
                                                RSTn => RSTn –Asynchron
                                           );
UUUU:FSM_BA    generic map (
                                              ADDR_WIDTH => 9
                                                    )
               port map (              clk => clk,
                                       address => address1,
                                             CRTL => CRTL1,
                                             RSTn => RSTn,
                                             ready => ready1,
                                             push => push,
                                             start => start1,
                                             increment => increment1
                                  );
UUUUU: serial_tx generic map (  CLK_PER_BIT  =>  1042,
                                                    CTR_SIZE=>  11
                                     )
                    port map (   clk => clk,
                                       rst => rst,
                                       tx => tx1,
                                       tx_block => '0',
                                       new_data => start1,
                                       data => data_1,
                                       busy => busy1
                                    );
end Behavioral;
```

A.1.6 Serial_tx

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.NUMERIC_STD.all;
entity serial_tx is
	generic (
		CLK_PER_BIT  : natural := 1042;
		CTR_SIZE     : natural := 11
	);
	port (
		clk		: in  std_logic;
		rst		: in  std_logic;
		tx		: out std_logic;
		tx_block	: in  std_logic;
		busy		: out std_logic;
		new_data	: in  std_logic;
		data		: in  std_logic_vector(7 downto 0)
	);
end entity serial_tx;
architecture RTL of serial_tx is
type tx_state_type is (IDLE, START_BIT, DATA_BITS, STOP_BIT);
signal state_d, state_q		: tx_state_type;
signal ctr_d, ctr_q		: unsigned(CTR_SIZE-1 downto 0);
signal bit_ctr_d, bit_ctr_q	: unsigned(2 downto 0);
signal data_d, data_q		: std_logic_vector(7 downto 0);
signal tx_d, tx_q		: std_logic;
signal busy_r			: std_logic;
signal block_d, block_q		: std_logic;
begin
tx<= tx_q;
busy<= busy_r;
tx_comb: process(tx_block, block_q, ctr_q, bit_ctr_q, data_q, state_q, data,
new_data)
begin
block_d <= tx_block;
ctr_d  <= ctr_q;
bit_ctr_d <= bit_ctr_q;
data_d	<= data_q;
state_d <= state_q;
		case state_q is
		when IDLE =>
			if block_q = '1' then
				busy_r	<= '1';
				tx_d	<= '1';
			else
				busy_r	<=	'0';
				tx_d	<=	'1';
				bit_ctr_d	<= "000";
				ctr_d	<=	(others => '0');
				if new_data = '1' then
					data_d	<= data;
					state_d	<= START_BIT;
					--busy_r		<= '1';
				end if;
```

```
            when START_BIT =>
                  busy_r <= '1';
                  ctr_d  <= ctr_q + 1;
                  tx_d   <= '0';
                  if ctr_q = (CLK_PER_BIT-1) then
                        ctr_d  <= (others => '0');
                        state_d      <= DATA_BITS;

                  end if;
            when DATA_BITS =>
                  busy_r <= '1';
                  tx_d   <= data_q(0);
                  ctr_d  <= ctr_q + 1;
                  if ctr_q = (CLK_PER_BIT-1) then
                        data_d       <= "0" & data_q(7 downto 1);
                        ctr_d        <= (others => '0');
                        bit_ctr_d    <= bit_ctr_q + 1;
                        if bit_ctr_q = 7 then
                              state_d      <= STOP_BIT;
                        end if;
                  end if;
            when STOP_BIT =>
                  busy_r <= '1';
                  tx_d   <= '1';
                  ctr_d  <= ctr_q + 1;
                  if ctr_q = (CLK_PER_BIT-1) then
                        state_d      <= IDLE;
                  end if;
            when others =>

                  busy_r <= '1';
                  tx_d   <= '1';
                  state_d      <= IDLE;
      end case;
end process tx_comb;
tx_seq: process(clk, rst)
begin
      if rst = '1' then
            state_q      <= IDLE;
            tx_q   <= '1';
      elsif rising_edge(clk) then
            state_q      <= state_d;
            tx_q   <= tx_d;
            block_q            <= block_d;
            data_q       <= data_d;
            bit_ctr_q    <= bit_ctr_d;
            ctr_q        <= ctr_d;
      end if;
end process tx_seq;
end RTL;
```

A.1.7 PUF_BAT

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity PUF_BAT is
              Port (   clk                    : in std_logic;
                    RST                : in std_logic;
                           push_button_in     : in std_logic;
                          tx                  : out std_logic
                             );
end PUF_BAT;
architecture Behavioral of PUF_BAT is
component counter_neu is
       generic (
                    top : natural := 250000
             );
             Port (       clk, push_button_in,RSTn  : in std_logic;
                    push_button_out           : out std_logic
                      );
end component;
component PUF_BA is
       generic (
                           DATA_WIDTH : natural := 18;
                           ADDR_WIDTH : natural := 9
                     );
       Port (       clk,RSTn,push             : in std_logic;
                    tx                  : out std_logic
                    );
end component;
signal push1: std_logic;
signal RSTn: std_logic;
begin
RSTn<=not RST;
U:PUF_BA    generic map (   DATA_WIDTH =>18,
                                              ADDR_WIDTH => 9)
                      port map (                 clk => clk,
                                                RSTn => RSTn,
                                                          push => push1,
                           tx=>tx
                             );
UUU:counter_neu  generic map (   top => 250000
                           )
                           port map (      RSTn => RSTn,
                              clk => clk,
                                                 push_button_in => push_but-
ton_in,
                                                 push_button_out => push1
                                    );
end Behavioral;
```

A.1.8 Counter_neu

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
entity counter_neu is
	generic (
		top : natural := 250000
		);
		Port (	clk, push_button_in, RSTn	: in std_logic;
			push_button_out		: out std_logic
		);
end counter_neu;
architecture Behavioral of counter_neu is
signal counter_up: integer range 0 to top;
signal button_old: std_logic;
signal s_push_button_out:std_logic;
begin
	push_button_out<=s_push_button_out;
process(clk,RSTn)
	begin
		if RSTn='0' then
			counter_up <= 0;
			button_old <= '0';
			s_push_button_out <= '0';
		elsif (rising_edge(clk)) then
			if counter_up = top then
				counter_up <= 0;
			elsif push_button_in = '0' and button_old = '1' then
				s_push_button_out <= '1';
			elsif counter_up = 100 then
				s_push_button_out <= '0';
				counter_up <= counter_up + 1;
			else
				counter_up <= counter_up + 1;
			end if;
			button_old <= push_button_in;
		end if;
	end process;
end Behavioral;
```

A.1.9 PUF_BAT.ccf

```
## PUF_BAT.ccf

Pin_in   "clk"  Loc = "IO_SB_A8" | SCHMITT_TRIGGER=true;
Pin_in   "push_button_in"  Loc = "IO_EB_B0"; # SW3
Pin_out  "tx"  Loc = "IO_NB_A5"; # tx
Pin_in   "RST"  Loc = "IO_NB_A0"; #
```

A.1.10 PUF_BAT_Synth.tcl

```
## synth.tcl
## argparse
set top [lindex $argv 0]
set src [lrange $argv 1 end
## import yosys commands
yosys -import
## vhdl frontend
ghdl --warn-no-binding -C --ieee=synopsys src/BRAM_BA.vhd src/PUF_BAT.vhd
src/counter_neu.vhd src/PUF_BA.vhd src/Counter_BA.vhd src/FSM_BA.vhd
src/MUX_BA.vhd src/serial_tx.vhd -e ${top}
## synthesis
synth_gatemate -top ${top} -nomx8
#synth_xilinx -top ${top} -nolutram
## outputs
write_verilog -noattr net/${top}_synth.v
```