Dokumentation

Marc de Bever

July 11, 2020

Contents

1 Einleitung

Dieses Dokument ist ein Teil der Dokumentation des Projektes Imager-Emulator. Dieses Projekt besteht aus drei Teilprojekten. Genauer gesagt besteht es aus zwei Bachelor-Thesen und einem Projekt 5 (P5) des Studienganges Elektro- und Informationstechnik der Fachhochschule Nordwestschweiz. Das P5 und die erste Thesis laufen parallel, die zweite Thesis wird ein Semester später durchgeführt. Diese Dokumentation ist diejenige von der ersten Thesis, welche parallel mit dem Projekt 5 durchgeführt wird. Das Projekt 5 und die zweite

Figure 1: Überblick über das Projekt

Thesis wird von Fabio Nardo geschrieben und diese Dokumentation ist von mir, Marc de Bever, geschrieben.

Das Projekt Imager-Emulator wurde von der Firma Varian Medical Systems ausgeschrieben. Das Ziel is es, einen Röntgensensor zu emulieren. Der Sensor wird auch Imager genannt, wenn auch die ganze zugehörige Elektronik gemeint ist. Daher kommt der Name Imager-Emulator. Der Sensor gibt Bilder über eine nicht standardisierte Schnittstelle an einen Computer weiter. Der Computer, welcher die Bilder des Sensors entgegen nimmt, rechnet diverse Algorithmen, die unteranderem Pixelfehler erkennen und korrigieren. Jedoch kann dem Sensor nicht gesagt werden, *mach mal 'nen Pixelfehler*. Daher braucht es ein Hardware-Emulator, um Bilder mit Pixelfehlern zu generieren und an den Computer zu senden. Mit Hilfe von diesem Emulator können die Algorithmen des Computers in einem realitätsnahen System getestet werden. Der Emulator soll über USB Konfiguriert werden. Somit ergibt sich das Blockdiagramm in Abbildung 1. Ein Computer konfiguriert den Emulator über USB, dieser gibt sich als Imager aus und überträgt die gewünschten Bilder mit den Pixelfehlern auf den XI-Computer. Der XI-Computer ist der Computer, auf dem die Algorithmen laufen. Um die Algorithmen zu testen, bekommt der Computer via Ethernet die berechneten Bilder wiederum und kann diese mit den erwarteten Bilder verglichen.

Da dieses Projekt eine spezielle Konstellation hat, ist die Aufteilung der Teilprojekte wie folgt. Das P5 soll die Hardware entwickeln. Und die erste Thesis, also diese Arbeit, soll die Entwicklungsumgebungen und das FPGA-Modul in Betrieb genehmen, sowie auch die Schnittstellen programmieren. Die zweite Thesis soll schlussendlich die Kommunikation mit dem Computer implementieren, es ermöglichen Pixelfehler in die Bilder einzubauen und das ganze Projekt abzuschliessen.

Dieser Bericht dient zum einen dazu, dass im weiterführenden Projekt verstanden wird, wie die Entwicklungsumgebung und Schnittstellen funktionieren, wo deren Möglichkeiten und Grenzen liegen, und zum anderen dient er, um zu diese Arbeit als Thesis zu dokumentieren. Der Bericht ist folgendermassen aufgebaut.

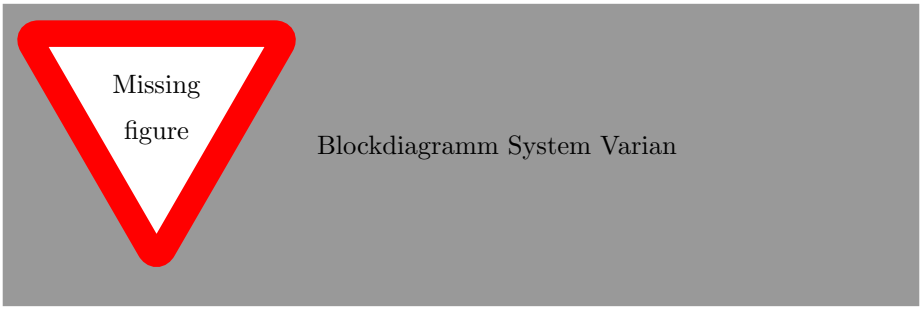

Figure 2: Blockdiagramm des Systems von Varian

Das zweite Kapitel beschreibt wie die Systeme aussehen. Dafür beschreibt es das Projekt mit drei verschiedenen Systemen. Als erstes das, welches von der Varian schon vorhanden ist. Dieses soll durch dieses Projekt emuliert werden. Das zweite System ist eine Zwischenlösung, damit gleichzeitig die Hardware und die Software entwickelt werden kann. Und das letzte System ist das endgültige Gerät, welches den Sensor der Varian emuliert. Diese Kapitel soll einen Top-Down-Blick auf das Projekt geben. Zusätzlich beschreibt es die Wahl des wichtigsten Komponenten, des FPGA-Moduls.

Das dritte Kapitel beschreibt den geschrieben Code. Dieser besteht aus drei Teilen, der Firmware für den Mikrocontroller, der für den FPGA und der Software für die PC-Applikation. Dieses Kapitel erwähnt zum Teil Sachen des zweiten Kapitels nocheinmal. Aber anstatt einen Überblick zu geben, soll dieses Kapitel sich auf die Details konzentrieren.

Das vierte Kapitel beschreibt wie der Imager-Emulator konfiguriert wird und wie zusätzliche Konfigurationen nachträglich hinzugefügt werden können.

Und zu guter Letzt, das sechste Kapitel beschreibt die Entwicklungsumgebungen (IDEs), mit welchen der Code entwickelt wurde. Dieses Kapitel beschreibt zuerst wie diese aufgebaut sind, was sie können und wie sie zusammenhängen. Als nächstes wird beschrieben, wie man mit den IDEs die Hardware debuggen kann und als letztes die Grenzen der Entwicklungsumgebungen.

2 System

Das Projekt kann in drei Systeme unterteilt werden. Das vorhandene System von Varian ist das erste. Das zweite System ist eine Zwischenlösung um mit der Entwicklung der Firmware anzufangen, ohne das die eigene Hardware schon vorhanden ist. Und das dritte System ist das endgültige System, welches das System von Varian emuliert.

2.1 System Varian

Das System der Varian besteht aus dem Imager und dem XI-Computer. Wobei diese beiden Blöcke Teil eines Röntgengerätes für medizinische Zwecke sind. Der Imager ist ein Bildsensor für Röntgenstrahlen und sendet die die gemessenen Bilder über eine LWL-Schnittstelle zum XI-Computer. Der XI-Computer empfängt die Bilder und lässt diverse Algorithmen darüber laufen. Unter anderem auch Algorithmen zur Erkennung von Pixelfehlern. Das mit den Algorithmen bearbeitete Bild wird wiederum über Ethernet an einen externen Computer gesendet.

Um die Bilder vom Sensor auszulesen, zu digitalisieren und zu übertragen, sitzt auf dem Imager ein FPGA. Dieser sendet die Bilder zum XI-Computer entweder über einen vom FPGA internen oder externen Serialisierer/Deserialisierer, kurz SerDes, zu einem LWL-Modul. Der FPGA kommuniziert über einen parallelen Bus mit dem externen SerDes. Die beiden SerDes senden ein LVDS Signal zum LWL-Modul. Das LWL-Modul ist ein standardisiertes Small Form-factor Pluggable (SFP). Um zwischen den beiden SerDes auszuwählen, gehen die Tx-Signale auf einen Multiplexer und die Rx-Signale auf einen Demultiplexer, dieser wird der Einfachheitshalber nur Mux genannt. Das beschriebene System ist im Blockschaltbild in Figure 2 zu sehen.

Der Vorteil des internen SerDes ist die Geschwindigkeit. Der externe SerDes Daten mit einer Übertragungsrate von 1.25Gbps übertragen und der interne mit 6Gbps. Die älteren Versionen des Imagers arbeiten nur mit dem externen SerDes und die neueren mit beiden. Da stellt sich die Frage, wieso überhaupt den externe SerDes brauchen? Beim Einschalten des Imagers ist der FPGA noch nicht konfiguriert und solange der FPGA nicht konfiguriert ist, ist auch der interne SerDes nicht konfiguriert. Somit braucht es den externen SerDes, um den FPGA vom XI-Computer aus zu konfigurieren. Sobald der FPGA konfiguriert ist, schaltet dieser auf den internen SerDes um und kommuniziert mit dem XI-Computer über diesen. Welche SerDes die verschiedenen Versionen des Imagers besitzen ist in Tabelle 1 ersichtlich.

Unterschied der verschiedenen Versionen erklären

Bootvorgang

Das Bitfile um den FPGA zu konfigurieren wird bei jedem Start vom XI-Computer über den externen SerDes auf den FPGA geladen. Dazu müssen sich zuerst die beiden SerDes, also der auf dem Imager und der auf im XI-Computer, synchronisieren. Damit der XI-Computer weiss, welcher FPGA auf dem Imager vorhanden ist, liegt ein Bitmuster mittels Pull-Up und Pull-Down Widerständen an den Tx-Leitungen des SerDes auf dem Imager. Somit ergibt sich folgende Bootsequenz.

Zuerst aktiviert der XI-Computer den Sync-Eingang seines SerDes. Somit kann sich der Rx-Pfad des SerDes auf dem Imager mit dem Tx-Takt des SerDes auf dem XI-Computer synchronisieren. Der Sync-Eingang des SerDes auf Imagers ist auf einen Rx-Kanal von sich selber angeschlossen. Dieser wird, sobald das Rx-Pfade des SerDes auf dem Imager synchronisiert ist, aktiviert. Nun sind

Table 1: Eckdaten der verschiedenen Sensorversionen der Varian

Imager	**Auflösung**	**Pattern**[1]	**FPGA**	**Schnittstelle**
DMI[2]	1280x1280	b10'1010'1010'1010'1010 XI: 0xAAA	Spartan 3 XC3S200	externer SerDes
RTI 1.0[2] (RTI4343L)	1280x1280	b10'1010'1010'1010'1010 XI: 0xAAA	Spartan 3 XC3S200	externer SerDes
RTI 2.0 (RTI4343iL)	3072x3072	b10'1010'1001'0101'0101 XI: 0x955	Artix 7 XC7A100T	aktuell: externer Serdes zukünftig: MGT(GTP)
RTIXL 1.0 (RTI8643L)	6144x3072	b10'1010'1001'1001'0101 XI: 0x995	Artix 7 XC7A200T	externer SerDes und MGT(GTP)

[1]Das XI wertet nur die letzten 12 bit des Pattern aus. Um das DC-Balancing zu erreichen, müssen trotzdem alle 18 bits korrekt gesendet werden.
[2]DMI und RTI 1.0 besitzen exakt die gleiche Elektronik.

beide Pfade der SerDes synchronisiert.

Als nächstes liest der XI-Computer das Versionmuster auf dem Imager aus. Aufgrund von diesem konfiguriert er den FPGA auf dem Imager mit dem entsprechenden Bitfile. Nun ist der FPGA konfiguriert und der Imager kann auf die Befehle vom XI-Computer reagieren.

Kommunikation zwischen Imager und XI-Computer

Man kann zwischen dem XI-Computer und dem Imager von einer Master-Slave Beziehung reden. Der XI-Computer hat die Möglichkeit dem Imager verschiedene Befehle zu senden, welche der Imager daraufhin ausführt. Die Befehle können in drei Gruppen eingeteilt werden. Es gibt die Write-Befehle, welche dem Imager Einstellungen übermitteln, die der Imager in einem Register speichert. Mit den Read-Befehlen kann der XI-Computer diese Einstellungsregister wieder auslesen. Und als letztes gibt es noch den Trigger-Befehl. Die Write-Befehle beinhalten eine Adresse und einen Wert, die Read-Befehle nur einen Wert. Mit dem Trigger-Befehl kann der XI-Computer ein Bild anfordern. Ein Trigger-Befehl hat Vorrang vor Write- und Read-Befehlen. Die genaue Auflistung aller Einstellungsregister und deren Aufgabe kann in der Dokumentation des Imagers entnommen werden.

auf Imager Dokumentation referenzieren

Der FPGA liest immer Zeile für Zeile die Werte des Sensors aus. Solange kein Trigger-Befehl vom XI-Computer gesendet wird, werden die Zeilen verworfen. Sobald ein Trigger-Befehl kommt, wartet der FPGA bis er die letzte Zeile ausgelesen hat und verwirft auch diese noch. Wenn der FPGA nun wieder bei der ersten Zeile angelangt, fängt er an das Bild Pixel für Pixel, Zeile für Zeile zu übertragen. Am Anfang des Bildes vor der ersten Zeile sendet der FPGA ein *Frame-Start* Signal. Und vor jeder Zeile überträgt er ein *Line-Start* Signal. Vor der ersten Zeile muss kein zusätzliches Line-Start Signal gesendet werden. Pro Takt wird ein Pixel übertragen. Der FPGA überträgt eine Zeile erst wenn

er sie vollständig vom Sensor ausgelesen hat. Dafür überträgt er die Zeile Pixel für Pixel ohne unterbruch. Ein Pixel ist 16 Bit gross und somit genau gleichgross, wie die parallele Schnittstelle zum SerDes. Um immer eine Zeile von Sensor auszulesen und eine Zeile zu übertragen speichert ein dual-port RAM die Pixelwerte.

Die Read-Befehle werden nur ausgeführt, wenn keine Bilddaten zu senden sind. Daher werden sie in einem FIFO zwischengespeichert und bei Möglichkeit übertragen. Die Read-Befehle dürfen nur zwischen Zeilen oder Bildern übertragen werden und nicht zwischen einzelnen Pixeln. Da es länger dauert eine Zeile von Sensor zu lesen, als eine Zeile an den XI-Computer zu senden, bleibt immer etwas Zeit um die Read-Befehle vom FIFO abzuarbeiten. Wie vor einem neuen Bild und einer neuen Zeile gibt es das *Command* Signal, dieses wird immer vor der Antwort auf einen Read-Befehls gesendet. Die Antwort eines Read-Befehls bestehen aus den Daten aus dem Einstellungsregister.

Wahl des FPGAs

Das oben beschriebene System der Varian soll emuliert werden. Die Vorgabe ist es 100 Bilder speichern zu können. Mit einer maximalen Bildgrösse von 6144x3072x16 ergibt dies etwa 3.5GB Daten. Als Recheneinheit ist die Vorgabe einen FPGA von Xilinx zu nehmen. Dies ist vom Auftragsgeber, Varian, vorgegeben, da sie mit diesen FPGAs arbeiten und wir somit auch vorhandenen Code für die LWL-Schnittstelle übernehmen können. Die Daten sollen via USB auf das Gerät geladen werden und über die LWL-Schnittstelle weiter zum XI-Computer. Daher braucht das System eine Hardware für das USB und eine für die LWL-Schnittstelle. Für das USB kommt entweder ein externer Chip infrage oder der FPGA muss ein integriertes USB-Modul besitzen. Die LWL-Schnittstell der Varian ist nicht vollständig standardisiert, daher muss diese mit externen Bauteilen aufgebaut werden. Der interne SerDes ist ein Multi-Gigabit-Transiever (MGT). Die FPGAs von Xilinx haben verschiedene MGTs, welche nicht alle die nötigen Geschwindigkeit unterstützen. https://www.xilinx.com/products/technology/high-speed-serial.html#overview Das Schema der LWL-Schnittstelle kann von der ref Varian übernommen werden. Um Übertragungsgeschwindigkeiten bis 6GBps zu unterstützen, muss der Speicher und der FPGA über eine entsprechend schnelle Schnittstelle kommunizieren können. Da das Layouten einer solchen Schnittstellen eine Komplexe angelegenheit ist und bei der Inbetriebnahme zusätzliche Tests benötigt, haben wir uns entschieden ein FPGA-Modul zu verwenden, auf welchem der Speicher schon vorhanden ist.

Mit diesen Anforderungen haben wir fünf verschiedene Varianten von den Firmen Enclustra und Trenz-Electronic gefunden. Da FPGA-Module mit 4GB RAM sehr teuer sind, haben nicht alle die Kapazität 100 Bilder bei voller Auflösung zu speichern. Die Varianten sind in der Tabelle 2 aufgelistet.

Die erste Variante ist die einzige Variante, welches mit 4GB genug Speicherplatz für alle 100 Bilder hat. Dafür ist es auch die teuerste. Das Modul besitzt ein Zynq Ultrascale+ SoC von Xilinx. Der Soc besitzt ein integriertes FPGA-System und ein integriertes ARM Prozessorsystem. Das USB 3.0 ist auch in

Table 2: Die verschiedenen Vorschläge für das FPGA-Modul

	Variante 1	**Variante 2**	**Variante 3**
Hersteller	Enclustra	Trenz-Electronic	Enclustra
Hersteller	ME-XU5-5EV-2I-D12E	TE0712-02-35-2I	ME-XU5-4EV-1I-D11E-G1
RAM	4GB	1GB	2GB
FPGA	Zynq Ultrascale+ XCZU5EV	Artix-7	Zynq Ultrascale+ XCZU4EV
USB	USB 3.0	-	USB 2.0
MGT	4 MGT 12.5	4 GTP	4 MGT 12.5
Preis	1010 CHF	225CHF	680CHF
	Variante 4	**Variante 5**	
Hersteller	Trenz-Electronic	Enclustra	
Hersteller	TE0803-03-4AE11-A	ME-KX1-160-1C-D10	
RAM	2GB	1GB	
FPGA	Zynq UltraScale+ XCZU4CG	Kintex-7	
USB	USB 3.0	USB 3.0	
MGT	4 GTH 16.3	4 MGT 6.6	
Preis	561CHF	638CHF	

dem SoC integriert. Somit kann das Prozessorsystem das USB Protokoll abarbeiten und die Bilder auf das RAM laden. Und das FPGA die Bilder mit der gewünschten Geschwindigkeit zum XI-Computer übertragen.

Variante zwei ist die günstigste, aber hat dafür nur 1GB RAM und kein USB. Als FPGA hat es einen Artix-7. Dies ist ein reiner FPGA und besitzt kein integriertes Prozessorsystem, wie Variante 1. Das USB müsste auf einem vom Modul separaten Chip implementiert werden.

Varianten drei und vier sind Mittellösungen, sie haben jeweils 2GB RAM und einen Zynq Ultrascale+ SoC. Vom Preis her liegen sie auch in der Mitte. Beide Varianten besitzen ein im SoC integriertes USB. Sie unterscheiden sich im Hersteller des Moduls und in der genauen SoC Version.

Variante fünf ist ist etwa gleich teuer wie die Mittellösungen, jedoch hat sie nur 1GB RAM. Dafür ist sie mit einem einfacheren FPGA und einem externen Chip für das USB ausgestattet. Diese Variante wäre einfacher zu programmieren, da es ein weniger komplexes System ist.

Die Entscheidung haben wir der Varian überlassen. Die Varian hat sich für die Variante 3 entschieden. Grund für diese Entscheidung ist: "Ein weiteres Entwicklungsboard mit einem Artix-7 bringt uns nichts und ein Modul mit einem Zync Ultrascale+ bringt und auch für später etwas. Ebenfalls gefällt uns, dass bereits DDR4 Memory und USB3.0 vorhanden ist und somit das Preis-Leistungsverhältnis für uns stimmt."

Jedoch ist uns da noch ein kleiner Fehler unterlaufen. Nämlich hat das Modul der Variante 3 Modul nur USB 2 und kein USB 3.0. Das Problem ist, das Enclustra Aufgrund von Kompatibilitäten bei den Mercury XU5 Modulen zwei verschiedene Bestückungsvarianten hat. Die einen führen die GTR-Pins auf die Stecker und die zweite Variante führt mehr IO-Pins anstatt die GTR-

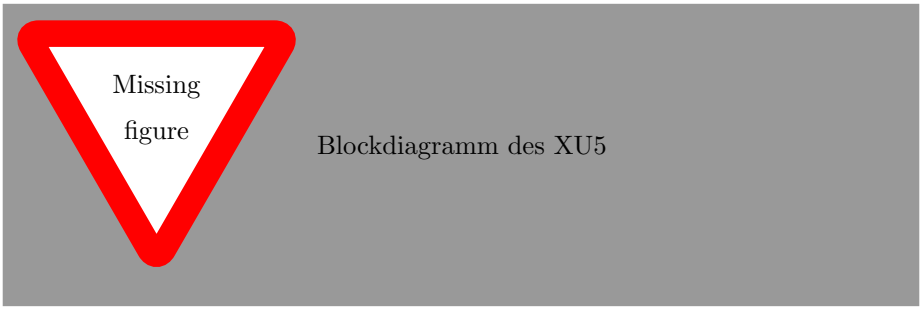

Figure 3: Blockdiagramm SoC Moduls XU5 von Enclustra

Pins auf die Modulstecker. Dies war uns auch so klar. Jedoch braucht das USB 3.0 auch die GTR-Pins, was wir zu diesem Zeitpunkt nicht wussten. Da USB 3.0 auf anderen Leitungen, wie USB 2.0 läuft, aber trotzdem die USB 2.0 Leitung besitzt und unterstützt, können die USB 2.0 Leitungen des SoCs, welche auf anderen Pins heraus geführt werden, trotzdem gebraucht werden. Dies war, obwohl erst nach dem Bestellen gesehen, trotzdem in Ordnung und daher sind wir bei dieser Variante geblieben.

Enclustra Mercury XU5

Der Enclustra Mercury XU5 Soc ist ein SoC Modul, auf welchem ein Xilinx Zynq Ultrascale+ ZU4EV SoC sitzt. Der Ultrascale ist ein System on Chip (SoC), welcher ein FPGA-System und ein ARM Prozessorsystem in einem Chip integriert. Das Modul hat zwei SDRAMs, wobei einer am ARM System hängt und der andere am FPGA. Beide sind über DDR4 an den SoC angeschlossen. Zusätzlich hat es sechs LEDs vier GPIOs und zwei Status-LEDs, welche auf die Pins *PS_ERROR* und *PS_STATUS* des SoCs geführt sind. Das USB 2.0 vom SoC ist per ULPI mit einem externen Chip auf dem Modul verbunden, welcher wiederum auf die Modulstecker geht. Über zwei Pins des Modulsteckers kann die Bootvariante ausgewählt werden. Somit sind mit vier verschiedenen Boot-Möglichkeiten nicht alle Möglichkeiten des SoCs vorhanden. Das Blockschaltbild ist in Abbildung 3 zu sehen. Das XU5 hat noch weitere Funktionen. Diese sind jedoch für dieses Projekt nicht relevant und sind daher weder auf dem Blockdiagramm noch im Text beschrieben. Für genauere Informationen kann die Dokumentation des Moduls angeschaut werden.

ref auf Doku von XU5

Xilinx Zynq Ultrascale+

Der Ultrascale ist ein SoC mit einem ARM Prozessorsystem und einem FPGA-System integriert in einem Chip. Das ARM-System wird auch Programmable System (PS) genannt und des FPGA-System auch Programmable Logic (PL). Das Blockschaltbild ist in Abbildung 4 zu sehen. Auch hier sind nur die für das Projekt relevante Blöcke abgebildet.

Figure 4: Blockdiagramm des Zynq Ultrascale+ SoCs von Xilinx

Das ARM-System besteht aus zwei ARM Cortex-R5 Kernen, vier Cortex-A53 Kernen und einem Grafikkern. Der Grafikkern ist für das Projekt irrelevant und wird daher komplet verschwiegen. Zusätzlich enthält das ARM-System noch eine Configuration Security Unit (CSU) und eine Platform Management Unit (PMU). Diese beiden Blöcke sind für das Booten zuständig und während dem Betrieb für Sicherheitsrelevante Funktionen und das Power Management.

Als Peripherie besitzt der SoC eine Schnittstelle für USB 3.0 und 2.0, UART, SDIO, DDR und JTAG. Zusätzlich können über GPIOs Pins als Ein- oder Ausgänge verwendet werden. Die verschiedenen Prozessorkerne. die Peripherien, der Speicher und das FPGA-System sind über AXI-Interconnect-Switches zusammengeschlossen. Das JTAG ist mit drei JTAG-Controllern verbunden. Einer, welcher auf das ganze System Zugriff hat, einer welcher mit dem Coresight des ARM-Systems verbunden ist und einer, welches das FPGA-System debuggen kann. Der FPGA ist ein Xilinx FPGA von der Ultrascale+ Familie.

Es gibt vier Hauptdokumentationen über das SoC. Das Datenblatt, das Technical Referenz Manual(TRM), die Software Developer Guide(SDG) und das Packaging and Pinouts Guide.

ref auf Dokus

2.2 Entwicklungssystem

Da das XU5 nur ein Modul ist, welches nicht selbstständig läuft, braucht es ein zusätzliches Board. Im Verlauf dieses Projektes wird dieses Board entwickelt. Damit jedoch direkt von Anfang an mit dem Entwickeln auf dem SoC begonnen werden kann, haben wir zusätzlich noch das Mercury+ PE1-200 Board von Enclustra gekauft.

Mercury+ PE1-200 Baseboard

Das PE1-Board ist ein Baseboard welches alle zusätzliche nötigen Funktionen für das XU5-Modul zu Verfügung stellt. Das Blockdiagramm ist in Abbildung 5 zu sehen. Auch hier sind wieder nur die für das Projekt relevanten Blöcke zu sehen. Das PE1-Board hat zwei USB Ports. Über den einen ist das JTAG und das UART geführt und den zweiten das USB Interface des SoCs. Das Routing

Figure 5: Blockdiagramm des Mercury+ PE1-200 Baseboards von Enclustra

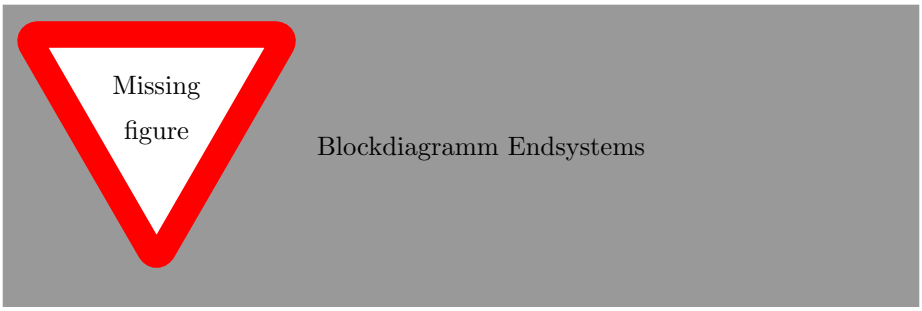

Figure 6: Blockdiagramm des Imager-Emulators

der USB Schnittstellen kann über die Schalter auf dem Board eingestellt werden. Das JTAG wird auf einen Chip auf dem Board zu UART umgewandelt, damit mit den Entwicklungsumgebungen direkt ohne zusätzliche Hardware per USB auf die Debug-Schnittstelle zugegriffen werden kann. Ein von Enclustra programmierter Chip konvertiert das JTAG auf UART und ein FTDI-Chip konvertiert die beiden UART-Signale auf USB. Um zu booten wird die SD Karte gebraucht, für welche das PE1-Board eine Halterung besitzt. Zusätzlich hat das Board noch LEDs und Schalter. Die LEDs und Schalter können als IOs des SoCs gebraucht werden. Mit den Schaltern können zusätzlich Einstellungen für den Bootmodus und das Routing der USB-Pfade gesetzt werden. Um von der SD-Karte zu booten müssen die Schalter *A1* und *B2* abgeschaltet sein. Um das JTAG und UART auf die Mikro-USB-Büchse zu führen, muss Schalter *B2* abgeschaltet sein. Und um das USB-Interface des SoCs auf die USB-B-Buchse zu führen, muss Schalter *B1* abgeschalten sein. ref auf Doku

2.3 Endsystem

In Abbildung 6 ist das Blockschaltbild des endgültigen Systems zu sehen. Das selber entwickelte Board hat einen Stecker für das XU5-Modul. Das Modul kann über den SerDes mit dem XI-Computer kommunizieren. Mit dem Mux kann

zwischen dem internen und dem externen SerDes gewechselt werden. Und das SFP Modul wandelt das elektrische Signal in ein optisches um. Die Daten können über den Mikro-USB-Stecker auf den SoC geladen werden. Um zu booten hat es ein SD-Kartenhalter. Zusätzlich hat es noch LEDs und Taster, um falls nötig einfache Ein- und Ausgaben zu betätigen. Um zu debuggen hat es einen Stecker für die JTAG-Schnittstelle. Um diese am Computer anzuschliessen braucht es noch eine externe Hardware, welche das JTAG auf USB konvertiert. Die JTAG-Schnittstelle ist nicht auf dem Blockdiagramm, da es nur zu Debug-Zwecken dient und während des Betriebs nicht verwendet wird.

Dieses Board ist die Arbeit des Projekt 5, welches gleichzeitig mit diesem Projekt durchgeführt. Für eine genauere Erklärung der Hardware und Designentscheidungen wird auf den Bericht jenes Projektes verwiesen.

ref Doku Fabio

3 Software

Dieses Kapitel ist der Hauptteil der Arbeit. Es beschreibt den FPGA und Mikrocontroller Code, welcher auf dem SoC läuft und wie dieser getestet wurde. Es ist in die Unterkapitel ARM, FPGA und PC Software aufgeteilt. - FPGA Code von Varian mit Blockschaltbild erklären und sagen, welche Module übernommen werden können. Sagen, das der Bootsequenz zuerst noch emuliert wird. Blockdiagramm von unserem System zeigen. und in den Unterkapiteln genauer auf die Module eingehen. FPGA code von Varian zuerst nur oberflächlich beschreiben und erst im Unterkapitel FPGA genauer auf die Relevanten Blöcke eingehen.

3.1 ARM

Dieses Kapitel beschreibt, welche Komponenten des ARM Cores gebraucht werden und wie sie konfiguriert sind, wie die Module des ARM Cores aufgebaut sind und auf welchem Core sie laufen.

3.2 FPGA

Dieses Kapitel beschreibt die Module, welche auf dem FPGA laufen.

3.3 PC Applikation

Dieses Kapitel beschreibt die Software, welche auf dem PC läuft, welche den Imager konfiguriert.

4 Konfiguration

Dieses Kapitel beschreibt, wie zusätzliche Konfigurationen hinzugefügt werden können.

5 Entwicklungsumgebung

Es gibt zwei Entwicklungsumgebungen (IDE) von Xilinx, um den Zynq Ultrascale+ SoC zu programmieren und zu konfigurieren. Die eine ist Vivado. Vivado ist die IDE um Code für den FPGA zu entwickeln. Mit Vivado kann man IPs im Projekt einfügen, FPGA Code simulieren, syntetisieren, implementieren, den FPGA Konfigurieren und direkt auf dem FPGA debuggen.

Die zweite IDE ist Vitis. Mit Vitis kann man Code für das ARM-System schreibe, Bootdateien erstellen, das ARM-System programmieren, den FPGA konfigurieren und den ARM debuggen. Vitis basiert auf der Eclipse IDE mit den GNU GCC Tools.

In diesem Projekt wurden die Versionen Vivado 2019.2 und Vitis 2019 verwendet. Es ist die erste Version von Vitis. Vorher wurde die Xilinx SDK verwendet um Code für den ARM zu schreiben. Ab Vivado 2019.2 ist es jedoch nicht mehr möglich Projekte für die SDK zu exportieren, da die SDK und Vitis unterschiedliche Files brauchen um die Hardware, welche mit Vivado generiert wird, zu importieren. Für beide Tools können mit TCL Skripte geschrieben werden.

Mit den beiden Entwicklungsumgebungen kommt noch das Programm DocNav. Diesem Programm beinhaltet alle Dokumentationen von Xilinx.

5.1 Vivado

-Einleitung - Für was ist das Tool - Bennung der einzelnen Schaltflächen(in Bild) - Was kann es alles / aus welchen Teilsystemen besteht es - Doku - IP Integrator - Simulation - Syntetisieren - implementieren - Hardwaremanager

Vivado ist die IDE für die Entwicklung von FPGAs, sie versteht VHDL und Verilog. Vivado kann entweder im Projektmodus oder nicht im Projektmodus verwendet werden. In diesem Projekt verwenden wir Vivado nur im Projektmodus. Alle Befehle können enteder im GUI oder in der TCL Konsole eingegeben werden. Somit ist alles vollständig skriptbar.

Vivado besteht aus verschiedenen Teilprogrammen. Diese Sind der IP-Integrator, der Simulator, der Syntetisierer, der Implementierer und der Hardwaremanager.

Die Dokumentation von Vivado ist auf verschiedene Dokumente aufgeteilt. Zum starten ist das "Vivado Design Suite User Guide: Using the Vivado IDE" diese Dokumentation referenziert in Kapitel zwei auf die weiteren Dokumentationen der einzelnen Unterprogramme von Vivado.

5.2 Vitis

-Einleitung - Für was ist das Tool - Bennung der einzelnen Schaltflächen(in Bild) - Was kann es alles / aus welchen Teilsystemen besteht es - Doku - Plattform Projekt - System Projekt - Simples Projekt - Bootgen

Vitis ist die IDE von Xilinx, um die Processoren zu programmieren. Vitis bassiert auf der Eclipse IDE. Von Vivado kann man die Hardware exportieren

und in Vitis importieren. Mit der importierten Hardware wird ein Platform-Projekt generiert. auf Grund von diesen Platform-Projekten kann man dann Projekte erstellen. Hier gibt es auch zwei verschiedene Arten. Die System-Projekte und die normalen. Pro System-Projekt können mehrere normale Projekte sein. Jedoch pro Processor nur ein. Jedes System-Projekt ist an ein Platform-Projekt gebunden und jedes normale Projekt in einem System-Projekt und für einen bestimmten Processor.

Kompiliert wird der Code mit den GCC Compilern. Diese werden mit Vitis mitinstalliert und können von Vitis aus gestartet werden. Der kompilierte Code befindet sich in elf Dateien

Mit Hilfe vom Bootgen Tool kann aus den elf-Dateien und Bitfiles generiert werden. Mit diesen kann man den SoC booten.

Die Dokumentation von Vitis ist unter zu finden. ref auf doku

5.3 Schnittstelle zwischen Vivado und Vitis

5.4 Debugging

- Debugging auf dem Chip - Die debugging Tools - Hardware-Server - Vivado - ILA - Vitis - Gnu debugger

5.5 Workflow

- Files von Vivado - Von Vivado in Vitis exportieren - Files von Vitis - Do nots (Probleme mit Vitis beschreiben) - Skripte

6 Erläuterungen

Dieses Kapitel listet alle Abkürzungen mit den Bedeutungen und einer kurzen Erklärung auf. Zudem erklärt es, wie die verschiedenen Fachwörter zu verstehen sind.

ARM Core, Core, FPGA, Sensor, XI-Computer, SoC, Imager,
Imager-Emulator
P5 LWL FPGA Imager XI-Computer SerDes LVDS Rx Tx FIFO