Disposition

Marc de Bever

July 7, 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). Das P5 und eine Thesis laufen parallel und 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 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 Sensor zu emulieren. Der Sensor gibt Bilder über eine nicht standardisierte LWL 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 Gerät, um Bilder mit Pixelfehlern zu generieren und an den Computer zu senden. Und dieses Gerät soll in diesem Projekt entwickelt werden.

Da dieses Projekt eine spezielle Konstellation hat, ist die Aufteilung der Teilprojekte wie folgt. Im P5 soll die Hardware entwickelt werden. In der ersten Thesis, also dieser Arbeit, sollen die Entwicklungsumgebungen und das FPGA-Modul in Betrieb genommen werden. Sowie sollen auch die Schnittstellen programmiert werden. Die zweite Thesis soll schlussendlich die Kommunikation mit dem Computer implementieren, die Pixelfehler in die Bilder einbauen und das ganze Projekt abschliessen.

Dieser Bericht dient zum einen dazu, dass im weiterführenden Projekt verstanden werden kann, wie die Entwicklungsumgebung und Schnittstellen funktioniert, deren Möglichkeiten und Grenzen. Zum anderen Dient er um zu diese Arbeit als Thesis zu dokumentieren. Der Bericht ist folgendermassen aufgebaut.

Das zweite Kapitel dieses Berichtes beschreibt wie das Sensorsystem der Varian aufgebaut ist. Zuerst gibt es einen groben Überblick und danach geht es genauer auf die Teilbereiche ein. Das dritte Kapitel beschreibt wie unser System aussehen soll. Es gibt einen top-down Blick auf das Projekt. Das heisst es beinhaltet die Blockschaltbilder. Zusätzlich beschreibt es die Wahl der wichtigsten Komponenten, wie das FPGA-Modul.

Das vierte Kapitel beschreibt die Hardware, da dies die Arbeit des Projektes 5 ist, werden in diesem Kapitel nur die wichtigsten Punkte erwähnt, welche für das Programmieren relevant sind.

Das fünfte Kapitel beschreibt die Software. Diese besteht aus drei Teilen, dem Code für den Mikrocontroller, dem Code für den FPGA und dem Code für die PC-Applikation. Das sechste Kapitel beschreibt wie zusätzliche Konfigurationen nachträglich hinzugefügt werden können. Das siebte Kapitel beschreibt die Entwicklungsumgebung.

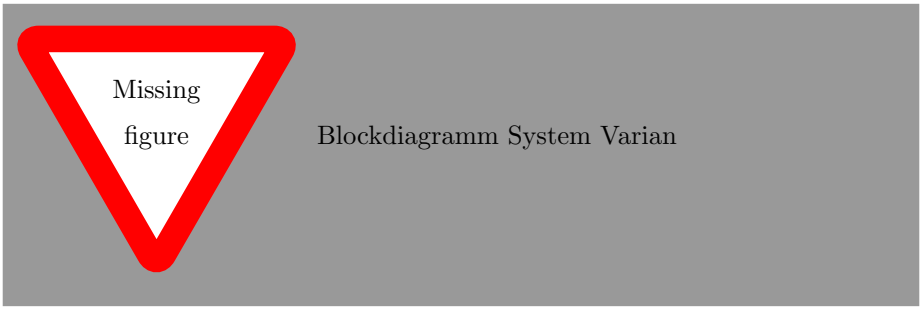

Figure 1: Blockdiagramm des Systems von Varian

Und das letzte Kapitel beschreibt Funktionalitäten, welche nicht von uns entwickelt worden sind, sondern schon auf dem Modul vorhanden sind.

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 Firmware Entwicklung 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 ein LWL-Kabel 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 kann wiederum über Ethernet an einen PC gesendet werden.

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 und der die beiden SerDes senden ein LVDS Signal zum LWL-Modul. 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 1 zu sehen.

Der Vorteil des internen SerDes ist die Geschwindigkeit. Der externe SerDes arbeitet 1.25Gbps und der interne mit 6Gbps. Die älteren Versionen des Imagers arbeiten nur mit dem externen SerDes und die neueren mit internen. Da

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.

stellt sich die Frage, wieso externe SerDes überhaupt gebraucht wird. Beim Einschalten des Imagers ist der noch nicht konfiguriert. Somit wird der externe SerDes gebraucht um den FPGA vom XI-Computer 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.

Bootvorgang

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. Dies hat Vorrang vor Write- und Read-Befehlen. Die genaue Auflistung aller Enstellungsregister 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 Werte 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 endlich 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. Somit bleibt zwischen den zu übertragenden Zeilen noch Zeit um etwaige Read-Befehle auszuführen.

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 den Zeilen und 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, welches vor den Daten des Read-Befehls gesendet wird. Die Daten eines Read-Befehls bestehen aus den Daten aus dem Einstellungsregister.

Wahl des FPGAs

Das oben beschriebene System soll emuliert werden. Die Vorgabe ist es 100 Bilder speichern zu können. Mit einer maximalen Bildgrösse von 6144x3072x16 ergibt dies etwa 3.5GB Daten. Die Daten sollen via USB auf das Gerät geladen werden und über die LWL-Schnittstelle weiter zum XI-Computer. Daher braucht unser System eine integrierte Hardware für das USB. Die LWL-Schnittstelle ist nicht standardisiert, daher braucht der Emulator eine selber designte Platine. Um Übertragungsgeschwindigkeiten bis 6GBps zu unterstützen muss der Speicher an einer entsprechend schnellen Schnittstelle angeschlossen sein. Da das Layouten einer solchen Schnittstellen zusätzliche Komplikationen, schwierigkeiten und tests verursacht, haben wir uns entschieden ein Modul zu verwenden, auf welchem der Speicher schon angeschlossen ist. 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.

Mit diesen Anforderungen haben wir fünf verschiedene Varianten von den Firmen Enclustra und Trenz-Electronic herausgesucht. Diese sind in der Tabelle 2 aufgelistet.

Die erste Variante ist die einzige Variante, welche genug Speicherplatz für alle 100 Bilder hat. Dafür ist sie auch die teuerste. Zusätzlich ist das Modul mit USB ausgerüstet. Variante zwei ist die günstigste, aber hat dafür nur 1GB RAM und kein USB. Dies müsste auf mit einem vom Modul seperaten Chip implementiert werden. Varianten drei und vier sind Mittellösungen, sie haben nur 2GB RAM, kosten dafür auch nicht so viel. Wobei Variante vier auch kein USB auf dem Modul besitzt. 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 weit-

Table 2: Verschiedene Vorschläge für ein 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
SDRAM	4GB	1GB	2GB
FPGA	Zynq Ultrascale+ XCZU5EV-2SFVC784I	Artix-7	Zynq Ultrascale+ XCZU4EV-1
USB	2x USB 3.0	-	2x USB 2.0
MGT	4 MGT 12.5	4 GTP	4 MGT 12.5
Preis	1010 CHF	225CHF	680CHF
Anz. IO	178		

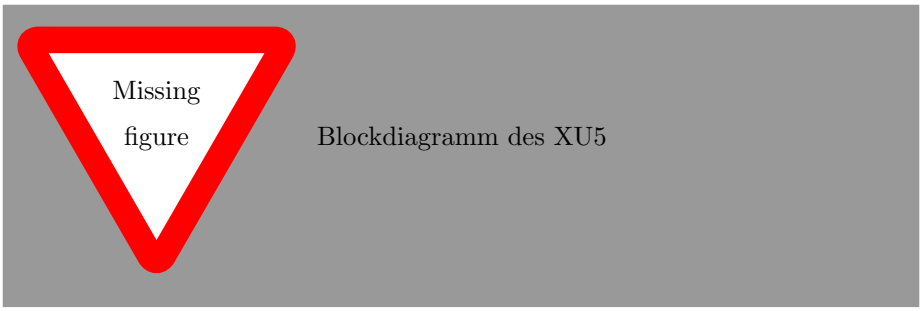

Figure 2: Blockdiagramm SoC Moduls XU5 von Enclustra

eres Entwicklungsboard mit einem Artix 7 bringt uns nichts und ein Modul mit einem Zyncscale 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. Und somit hat dieses 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 Lines auf die Stecker und die zweite Variante führt mehr IOs anstatt diese Lines auf die Stecker. Dies war uns auch so klar. Jedoch braucht das USB 3.0 die GTR Lines. Daher kann nur das USB 2.0 gebraucht werden, welche seperat heraus geführt werden. Dies war jedoch, obwohl erst im Nachhinein gesehen, auch in Ordung und daher sind wir bei deiser 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 und ein ARM System hat. Das Modul hat zwei SDRAMs, wobei einer am ARM System hängt und der andere am FPGA. Zusätzlich hat es noch vier LEDs. Das USB 2 vom Ultrascale ist per ULPI mit einem externen

Figure 3: Blockdiagramm des Zynq Ultrascale+ SoCs von Xilinx

Chip verbunden, welcher wiederum auf die Stecker geht. Das Blockschaltbild ist in Abbildung 2 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 system und einem FPGA integriert in einem Chip. Das Blockschaltbild ist in Abbildung 3 zu sehen. Auch hier sind nur die für das Projekt relevante Blöcke abgebildet.

Das ARM-System besteht aus zwei ARM Cortex-R5 Kernen, vier Cortex-A53 Kernen und einem Grafikkern. Der Grafikkern ist irrelevant für das Projekt 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 und während dem Betrieb für die Sicherheit und das Power Management zuständig.

Das Programm läuft auf einem der beiden R5 Kernen. Der Grund, wieso nicht der A53 Kern verwendet wird, ist, dass der A Kern eine MMU besitzt, welche den Adressraum virtualisiert und somit wesentlich komplexer ist.

Der FPGA ist ein Xilinx FPGA von der Ultrascale+ Familie. Er ist über AXI Busse mit dem ARM-System verbunden.

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

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 PE1 Board von Enclustra gekauft.

Figure 4: Blockdiagramm des Mercury+ PE1-200 Baseboards von Enclustra

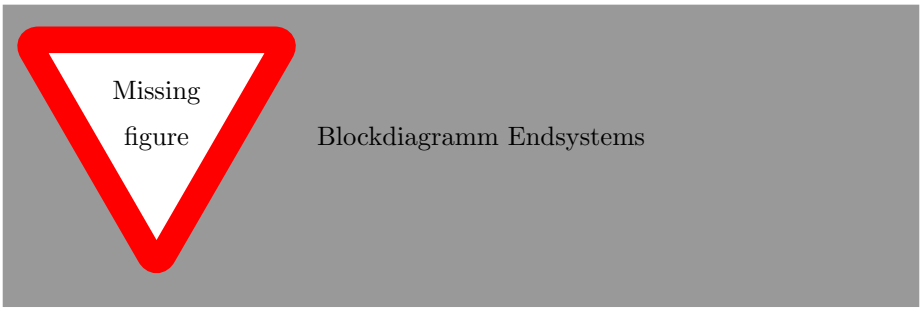

Figure 5: Blockdiagramm des Imager-Emulator

Mercury+ PE1-200 Baseboard

Das PE1 Board ist ein Baseboard welches alle zusätzliche nötigen funktionen für das XU5 zu verfügungstellt. Das Blockdiagramm ist in Abbildung 4 zu sehen. Auch hier sind wieder nur die für uns relevanten Blöcke zu sehen.

- Blockdiagram erklären - Nur Blöcke, welche wir auch brauchen - SD - USB - JTAG - UART - IO - Dip configuration - Boot - Referenz auf Doku

2.3 Endsystem

- Blockdiagram erklären - SFP -¿ Nicht auf Entwicklungssystem - SD - JTAG - IO - Referenz auf Schema und Liste mit Pin und FPGA belegung.

3 Hardware

Dieses Kapitel beschreibt die Hardware des Projektes. Es gibt einen kurzen Überblick und verweist auf die Dokumentation von Fabio und die anderen Dokumente für genauere Angaben. Es definiert die Angaben, welche für das Programmieren des SoCs wichtig sind. Und beschreibt die Inbetriebnahme des

SoC Teils der Hardware. – evt. weglassen, da teil in Kapitel System schon genug erläutert.

4 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.

4.1 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.

4.2 FPGA

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

4.3 PC Applikation

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

5 Konfiguration

Dieses Kapitel beschreibt, wie zusätzliche Konfigurationen hinzugefügt werden können.

6 Entwicklungsumgebung

Dieses Kapitel beschreibt, wie der Code für den SoC entwickelt werden kann und worden ist. Dies beinhaltet Vitis, Vivado und wie das Programm auf den SoC geladen werden kann.

6.1 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

6.2 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

6.3 Debugging

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

6.4 Workflow

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

7 Theorie

Dieses Kapitel beschreibt Funktionalitäten, welche nicht von mir entwickelt worden sind, sondern schon auf dem Modul vorhanden sind. In den einzelnen Unterkapiteln listet es die wichtigsten Konzepte und referenziert auf die Dokumentationen um genauere Informationen zu erhalten. Diese Unterkapitel sollen dazu dienen, dass in den anderen Kapiteln auf diese referenziert werden kann.

7.1 Enclustra Mercury PE5 Modul

7.2 Xilinx SoC

7.3 Bootvorgang

7.4 Standart Konfiguration der CPU

7.5 AXI

Eventuell werden noch folgende weitere Spezifikationen und Komponenten beschrieben.

7.6 ARM R5

7.7 USB Standart

7.8 USB Treiber

7.9 SerDes

7.10 MGT

7.11 Debugging

8 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