Hochschule
Augsburg University of
Applied Sciences

Fakultät für
Informatik

Bachelorarbeit

Desgin und Implementierung eines FPGA-Event-Recorders mithilfe der freien IceStorm-Toolchain

Studienrichtung: Technische Informatik

Domenik Müller

Hochschule für angewandte
Wissenschaften Augsburg

An der Hochschule 1
D-86161 Augsburg

Telefon +49 821 55 86-0
Fax +49 821 55 86-3222
www.hs-augsburg.de
info@hs-augsburg.de

Fakultät für Informatik
Telefon +49 821 55 86-3450
Fax +49 821 55 86-3499

Verfasser der Bachelorarbeit
Domenik Müller
Am Eser 3
86150 Augsburg
Telefon +49 821 44 92 57 54
domenikmueller@gmx.net

Prüfer: Prof. Dr. Hubert Högl
Zweitprüfer: Prof. Dr. Alexander von Bodisco

Abgabedatum: 20.06.2018

Zusammenfassung

In der vorliegenden Arbeit wird der Entwurf und die Implementierung eines FPGA-basierten Event-Recorders mithilfe der Open-Source-Toolchain IceStorm beschrieben. Ähnlich wie bei einem Logikanalysator werden digitale Signale an den Eingängen erfasst, allerdings wird der Signalzustand an den Eingängen nicht wie bei einem Logikanalysator kontinuierlich übertragen, sondern es wird bereits zur Erfassungszeit nach relevanten Signaländerungen und Eingangskombinationen gefiltert. Eine Filterung nach benutzerdefinierten Events bietet sich vor allem für die Analyse von bekannten Signalen an, deren Ablauf mit möglichst hoher zeitlicher Auflösung abgebildet werden soll. Die Implementierung des Event-Recorders erfolgt auf einem Raspberry Pi Zero mit einem Lattice iCE40-basierten FPGA-Shield und wird vollständig mit den von der IceStorm-Toolchain zur Verfügung gestellten Open-Source-Tools und Komponenten umgesetzt.

Abstract

The work in hand describes the design and implementation of a FPGA based event recorder using the open source toolchain IceStorm. Similar to a logic analyzer it records logic level input signals, but in contrast to a logic analyzer it does not provide a continuous stream of the signal state. Instead the signals are filtered at capture time to match only relevant input combinations and signal changes. Using predefined events to filter the input stream is especially suitable for the analysis of known signals that are to be captured and examined at high temporal resolution. The implementation of the event recorder is done on a Raspberry Pi Zero with a Lattice iCE40-based FPGA shield and is realized completely with the tools and components provided by the open source toolchain IceStorm.

Inhaltsverzeichnis

1. Einführung

Mikrocontroller werden heute in Gebrauchsgegenständen aller Art verbaut und werden den Anforderungen entsprechend immer leistungsstärker und damit unter anderem auch schneller. Selbst einfache Mikrocontroller arbeiten oft mit einer Geschwindigkeit im mehrstelligen Megaherz Bereich (sprich: mehrere Millionen Takte pro Sekunde). Im Unterschied zu klassischen PC-Systemen werden an Mikrocontroller oft Echtzeit-Anforderungen gestellt, das heißt Ergebnisse müssen zuverlässig innerhalb einer vorbestimmten Zeitspanne geliefert werden[1]. Dabei wird auch die Hardware von Mikrocontrollern zunehmend komplexer und es werden vermehrt Mehrprozessor-Systeme verwendet, die angepasste und mitunter unübersichtlichere Programmiertechniken erfordern.
Um Echtzeit-Anforderung erfüllen zu können werden für die Entwicklung von Mikrocontroller-Systemen (aber auch von digitalen Systemen im allgemeinen) Werkzeuge benötigt, mit denen Signale mit hoher zeitlicher Auflösung erfasst und analysiert werden können.
Diese Aufgabe wird unter anderem von Logikanalysatoren erfüllt, die die an den Eingängen anliegenden Spannungen mit einer festen Frequenz erfassen und die Daten dann zum Beispiel an einen PC übertragen, an dem sie ausgewertet werden können.
In der vorliegenden Arbeit wird eine spezielle Form von Logikanalysator entworfen, bei der ein Teil der Auswertung bereits zur Erfassungszeit durchgeführt wird. Das Eingangssignal wird auf bestimmte - vom Benutzer definierte - Signaländerungen untersucht und nur relevante Signaländerungen ("Events") werden an den Benutzer weitergereicht.
Dieses Vorgehen bietet sich vor allem dann an, wenn der Signalverlauf grundsätzlich bekannt ist und der Fokus der Analyse auf der exakten zeitlichen Abbildung des erwarteten Signalverlaufs liegt.
Ein Beispiel wären die Ausgänge eines Mikrocontrollers, bei dem bestimmte Kombinationen gesetzt werden um den Start und das Ende von Funktionen im Quellcode zu signalisieren. Da das Setzen von GPIO-Pins meist in einem einzigen CPU-Takt ausgeführt werden kann, können so zuverlässige Aussagen zur Laufzeit von Funktionen, oder bei periodischer Ausführung auch zur zeitlichen Fluktuation der Funktionsausführung ("Jitter-Analyse") getroffen werden.

1.1 Motivation

Die vorliegende Arbeit schließt thematisch an die Bachelorarbeit "Ein universales, rekonfigurierbares und freies USB-Gerät zur Timing-, Protokoll-, Logik- und Eventanalyse von digitalen Signalen" von Andreas Müller (2010) und einer darauf folgenden, an der Hochschule Augsburg durchgeführten, Projektarbeit im Wintersemester 2013/14 an.
In der Bachelorarbeit wurde eine Hardware-Platine namens "USB-TPLE" mit USB-Schnittstelle, einem CPLD-Chip von Altera und einem Atmega Mikrocontroller für die selbe Zielsetzung entworfen, und mit der Software-Implementierung begonnen[2].
Im nachfolgenden Semester-Projekt "Logikanalysator mit AVR Mega32U4 und Altera MAX CPLD" wurde die Software-Implementierung ausgebaut und eine erste funktionsfähige Konfiguration für den CPLD-Chip entwickelt.

Im Folgenden soll ein anderer Ansatz für die Umsetzung verfolgt werden:

Verwendung von käuflich verfügbarer Hardware
: Anstatt der selbst entworfenen Platine soll aus Gründen der Vefügbarkeit und um die Einstiegshürde für Benutzer zu verringern ein käuflich erwerbbares Produkt verwendet werden.

Verwendung eines FPGAs
: Um größere Flexibilität bei der Implementierung zu ermöglichen wird ein Field Programmable Gate Array (FPGA) anstatt des Complex Programmable Logic Device (CPLD) verwendet (eine detailliertere Erklärung findet sich im Kapitel "Design").

Neben Verfügbarkeit und Flexibilität des Designs wird vor allem ein weiterer Grundsatz bei der Implementierung verfolgt:

Verwendung von Open-Source Software und Hardware
: Bereits die Arbeit von Andreas Müller wurde unter einer Open-Source-Lizenz veröffentlicht und es wurden alle Projekt-Quellen und Ressourcen (einschließlich des Hardwaredesigns) öffentlich verfügbar gemacht.
Dieser Ansatz soll hier weiter verfolgt werden, dementsprechend steht der im Rahmen dieser Arbeit entstandene Quelltext (wie große Teile der IceStorm-Toolchain) unter der ISC-Lizenz zur Verfügung. Die Bachelorarbeit selbst wird unter Creative Commons Attribution-ShareAlike 3.0 International Lizenz veröffentlicht.
Ausserdem steht mit dem Projekt "IceStorm" erstmals auch eine Open-Source Software-Toolchain zur Programmierung von FPGA-Chips zur Verfügung, wodurch eine vollständige Open-Source Implementierung möglich wird. (In der vorliegenden Arbeit mit Ausnahme der proprietären Komponenten des Raspberry Pi Zero).

In Kombination mit der preisgünstigen[1] Hardware bietet die IceStorm-Toolchain eine interessante Alternative zu den Angeboten der großen FPGA-Hersteller wie Xilinx oder Intel (ehemalig Altera), insbesondere für Lehrzwecke und kleinere Projekte.

[1] Der Preis des verwendeten IceZero-Boards lag zum Zeitpunkt der Erstellung der Arbeit zum Beipiel bei ca. 40€, das zur Programmierung verwendete Raspberry Pi Zero W bei unter 15€

1.2 Abgrenzung von bestehenden Lösungen zur Logikanalyse

Es ist eine Vielzahl von kommerziellen Logikanalysatoren am Markt verfügbar. Allerdings bieten selbst flexible Geräte wie zum Beispiel die Discovery Serie von Digilent nicht die gewünschte Funktionalität der Event-Filterung zur Erfassungszeit und die Möglichkeit die so gewonnenen Daten in einen Text- bzw. Kommandozeilen-basierten Workflow einzubetten [2].

Von kommerziellen Produkten abgesehen gibt es auch einige Open-Source Logikanalysatoren. Für diese Arbeit relevant sind hier vor allem:

SUMP2 ist eine Verilog-basierte Logikanalysator-Implementierung mit einer zugehörigen - in Python implementierten - grafischen Benutzeroberfläche. Es existieren angepasste Varianten von SUMP2 die ohne weitere Modifikationen auf dem auch in dieser Arbeit verwendeten iCE40-FPGA-Chip lauffähig sind[4].

Open Bench Logic Sniffer ist ein Open-Source Hardware-Produkt das auf einem Xilinx Spartan 3E FPGA basiert und eine weiterentwickelte Variante von SUMP2 verwendet. Die "Demon core" betitelte Weiterentwicklung ist unter anderem deshalb interessant, da mit ihr komplexere Triggerbedingungen definiert werden können, und so zum Beispiel zeitliche und logische Abläufe von Eingangssignalen als Trigger abgebildet werden können. Hierauf soll im Kaptiel "Aussicht" noch einmal eingegangen werden.

Das verwendete SUMP2 Datenübertragungsformat wird zum Teil auch von anderen Anwendungen unterstützt, so kann zum Beispiel der Java-Client JaWi[5] oder Pulseview[6] (ein Qt-Frontend der libsigrok-Biliothek) als grafische Benutzeroberfläche verwendet werden.
Beide Varianten verwenden zur Datenübertragung eine serielle Schnittstelle (UART), die - zumindest bei Verwendung von geläufigen Baud-Raten - die Übertragungsgeschwindigkeit stark einschränkt. Ebenso sind beide Varianten konzeptionell für die Aufnahme festgelegter und relativ kurzer Sampling-Zeiten ausgelegt und unterstützen — wie die kommerziellen Produkte — keine Event-Filterung zur Erfassungszeit.

Eine Anpassung des SUMP2 Projektes wurde in Erwägung gezogen, aber aufgrund der zum Teil recht hohen Code-Komplexität und der strukturellen Unterschiede nicht durchgeführt.

1.3 Zielsetzung

Für die Implementierung des Event-Recorders wurden folgende technische Ziele angestrebt:

- Es soll der logische Pegel von 8 bis 16 Eingangs-Pins abgefragt werden und die Eingangsdaten sollen mit einem stabilen Zeitstempel versehen werden.
- Die zeitliche Auflösung der Aufnahme soll im mehrstelligen Megaherz-Bereich liegen
- Bestimmte Eingangskombinationen sollen in Textform definiert, und bei der Aufnahme als Events erkannt werden
- Zur Steuerung der Aufnahme soll ein Kommandozeilentool zur Verfügung stehen, mit dem auch die aufgenommenen Daten in Textform abgespeichert werden können.

[2] Geräte der Discovery-Serie können durch eine API z.B. in Python geskriptet werden, eine kontinuierliche "Event-Erkennung" ist aber nicht ohne Eeiteres möglich (siehe z.B. folgender Foreneintrag[3])

1.4 Aufbau der Arbeit

Im folgenen Kapitel “Design” werden zunächst die nötigen technischen Grundlagen für die Umsetzung des Projekts besprochen, anschließend wird auf getroffene Desginentscheidungen bei der Auswahl der Hardware und Software eingegangen.Das Kaptiel “Implementierung” beschreibt die nötigen Anpassungen bestehender Software und die Entwicklung neuer Softwarekomponenenten bei der Durchführung des Projekts. Im Kapitel “Anwendungsfall: Jitter-Analyse von Software-generierten MIDI-Clock Signalen” wird die Benutzung des Event-Recorder und die Analyse der aufgenommenen Daten anhand eines konkreten Beispiels besprochen. Es folgt ein “Fazit” in dem der Status des Projekts und die Umsetzung rekapituliert werden und abschließend wird im Kapitel “Aussicht” auf weiteres Vorgehen und Optimierungsmöglichkeiten eingegangen.

2. Design

2.1 Theoretische Grundlagen

Zunächst sollen die grundlegenden Eigenschaften der erwarteten Eingangssignale definiert und näher beschrieben werden.

2.1.1 Zeit- und wertdiskrete digitale Signale

Bei den Eingangssignalen des Event-Recorders handelt es sich um die Ausgänge von Mikrocontrollern oder anderer digitaler Schaltungen und damit um digitale Signal. Digitale Signale sind durch zwei grundlegende Eigenschaften charakterisiert, sie sind:

- **zeitdiskret**, und
- **wertdiskret**

Wertdiskret bedeutet, dass das Signal nur genau einen Wert aus einer festgelegten Anzahl möglicher Werte-Zustände annehmen kann. In den meisten Fällen beschränkt sich der Wertebereich auf die Binärwerte “1” oder “0”, das heißt ein digitales Signal ist zum Beispiel bei 0,1 Volt Spannung “0” und bei 3,2 Volt “1”, wohingegen ein analoges Signal alle möglichen Werte zwischen 0 und 3,3 Volt annehmen kann.

Zeitdiskret bedeutet, dass ein digitales Signal “nur zu bestimmten periodischen Zeitpunkten definiert ist beziehungsweise nur dann eine Veränderung im Signalwert aufweist”[7], das heißt dass das Signal bei der Erfassung mit einem festen “Zeitraster” abgetastet wird und zwischen den Rasterpunkten einen festen Wert behält.

Bei der Erfassung digitaler Signale ist es nötig, dass die Abtastfrequenz nach Nyquist-Shannon-Abtasttheorem mindestens doppelt so hoch als die maximale Frequenz des untersuchten Signals sein muss, damit das Ausgangssignal ohne Informationsverlust abgebildet werden kann (vgl. [7]).

Es ist zu beachten, dass es sich bei der Definition von digitalen Signalen um eine idealisierte Ansicht handelt und dass sich bei realen digitalen Signalen oft – vor allem bei hohen Frequenzen – Störeffekte zeigen. Ein Beispiel für einen Störeffekt wäre das “Prellen” eines mechanischen Schalters, bei dem es durch den mechanischen Kontakt zu einem mehrfachen Signalwechsel kommen kann, bevor ein stabiles Signal anliegt. Wenn der Schalter-Zustand dabei mit vergleichsweise langsamer Geschwindigkeit ausgelesen wird, gehen die

deutlich schnelleren Signalwechsel gemäß dem Nyquist-Shannon-Abstasttheorem nicht in das Ausgangssignal ein, bei entsprechend hoher Abtastrate werden sie allerdings mit in Ausgangssignal übernommen und können dann unerwünschte Effekte auslösen.
Bei der in dieser Arbeit verwendeten Hardware sind keine speziellen Vorrichtungen vorhanden um solchen Effekten entgegen zu wirken, das heißt es wird am Eingang ein für die Analyse ausreichend stabiles Signal erwartet.

Damit die Ergebnisse des Event-Recorders richtig auswertet werden können, muss zu jeder Signaländerung der "diskrete" Zeitpunkt bekannt sein, das heißt das Zeitraster der Abtastung muss numeriert werden, so dass jeder Signaländerung ein eindeutiger Zeitstempel zugeordnet werden kann. Diese "Numerierung" wird umgesetzt, in dem der von einem Oszillator[1] generierte Schaltungstakt mit einer Zählerschaltung aufaddiert wird. Der Zeitstempel entspricht dann einfach dem aktuellen Zählerstand.

EVENT_ID [8 Bit]	INPUT_DATA [8 Bit]	TIMESTAMP [16 Bit]
0x02	0x01	0x0EF8

Tabelle 2.1: Beispielhafte Darstellung eines aufgenommenen Events mit 16-Bit Zeitstempel

Die Bit-Breite des Zählers legt zusammen mit der Geschwindigkeit des Takts die maximal mögliche Aufnahmelänge fest. Der in dieser Arbeit verwendete Zähler hat eine Breite von 46 Bit, womit sich bei einer Taktfrequenz von 100 Mhz zum Beispiel eine Laufzeit von ca. 195 Stunden ergibt[2] (was für die meisten Anwendungsfälle mehr als ausreichend ist).

2.1.2 Definition "Event"

Für diese Arbeit soll unter dem Begriff "Event" ein vom Benutzer festgelegter Signalzustand oder eine Signaländerung an den Eingangspins des Event-Recorders verstanden werden. In den meisten Fällen macht es mehr Sinn eine Signaländerung zu definieren, als einen Zustand, da bei einem anhaltenden Zustand auch das Event kontinuierlich "ausgelöst" wird, und ein dementsprechend hohes Datenvolumen erzeugt.
Es ergeben sich folgende Definitionsmöglichkeiten für die einzelnen Eingangs-Pins:

- "0": niedriger logischer Pegel
- "1": hoher logischer Pegel
- "u": steigender Pegel
- "d": fallender Pegel
- "x": beliebiger Zustand ("don't care")

Zusätzlich wäre eine Kombination von "u" und "d" denkbar, die auf beliebige Signaländerungen reagiert. Eine Verkettung dieser Möglichkeiten bei der jedem Eingangspins ein Zustand zugewiesen wird soll als "Event-Trigger" – also als Auslöser eines bestimmten Events – bezeichnet werden.
Dies entspricht im Wesentlichen der Definition von *Trigger* die bei "traditionellen" Logikanalysatoren verwendet wird, allerdings mit dem Unterschied dass der Trigger bei einem "traditionellen" Logikanalysator die eigentliche Aufnahme einmalig auslöst, und dann bis

[1] Eigener Hardware-Baustein, der eine konstant auf- und abschwingende Spannung erzeugt und damit zur Taktung digitaler Schaltung verwendet werden kann

[2] Berechnung: $2^{46} * \frac{1}{100Mhz} = 2^{46} * 10ns =$ 195 Stunden, 28 Minuten und 7,422 Sekunden

zum Ende der Aufnahme nicht mehr von Bedeutung ist, während ein Event-Trigger als Teil der Aufnahme kontinuierlich überprüft werden muss und erst bei Erfüllung der Trigger-Bedingung überhaupt Ausgangsdaten generiert werden.
Es bietet sich an einem Event zusätzlich bestimme Funktionen zuweisen zu können, wie zum Beispiel das Starten[3] und Stoppen der Event-Erkennung, oder das Wechseln in einen zusätzlichen Modus, bei dem alle Signaländerungen ein Ausgangs-Event produzieren ("Dump"-Modus).
Wie in der Einführung erwähnt soll die Definition der Events in Text-Form möglich sein und kann dann zum Beispiel für 8 Eingangspins folgendermaßen aussehen:

```
# event configuration
events:
  - start:
          trigger: uxxxxxxx
          function: start

  - stop:
          trigger: dxxxxxxx
          function: stop

  - event1:
          trigger: 1uxxxxxx

  - event2:
          trigger: 1xuxxxxx
          function: dump_begin
...
```

[3] Die Erkennung eines Start-Events erfordert folglich, dass auch im "Ruhezustand" eine Event-Erkennung durchgeführt wird

2.2 Überblick der benötigten Hard- und Software-Komponenten

2.2.1 Datenerfassung: FPGA

Wie im vorherigen Kapitel beschrieben wird für die Datenerfassung eine Zählerschaltung benötigt, die einen stabilen Zeitstempel liefern kann. Voraussetzung dafür ist, dass der Zähler kontinuierlich läuft und nicht durch andere Vorgänge unterbrochen werden kann. Dies ist bei PC-Systemen oder auch Mikrocontrollern nicht ohne weiteres möglich, da die Programmausführung zu jedem Zeitpunkt von Betriebssystem-Funktionen oder Interrupt-Routinen[4] pausiert werden kann.
Deswegen bietet sich hier die Verwendung einer programmierbaren logischen Schaltung wie zum Beispiel eines CPLDs (Complex Programmable Logic Device) oder FPGAs (Field Programmable Gate Array) an, bei dem eine von anderen Komponenten zeitlich vollkommen unabhängige parallele Ausführung des Zählers möglich ist.
Sowohl CPLD- als auch FPGA-Chips bestehen aus einer Vielzahl von einheitlichen Blöcken die einfache logische Funktionen abbilden können (in der folgenden Abbildung "PLBs", also "Programmable Logic Blocks" bezeichnet). Im Vergleich zu logischen Gattern sind die die Blöcke in ihrer Funktion aber frei konfigurierbar. Durch die Vernetzung der so konfigurierten Blöcke können umfangreiche digitale Schaltungen realisiert werden.
FPGAs sind etwas komplexer aufgebaut als CPLDs, dabei aber auch flexibler bei der Vernetzung und enthalten oft zusätzliche Funktionsblöcke wie den in der nachfolgenden Abbildung erkennbaren Block-RAM als Zwischenspeicher für größere Datenmengen oder die PLL-Einheit zur Erzeugung von Taktsignalen mit konfigurierbarer Geschwindigkeit (vgl. [8]).

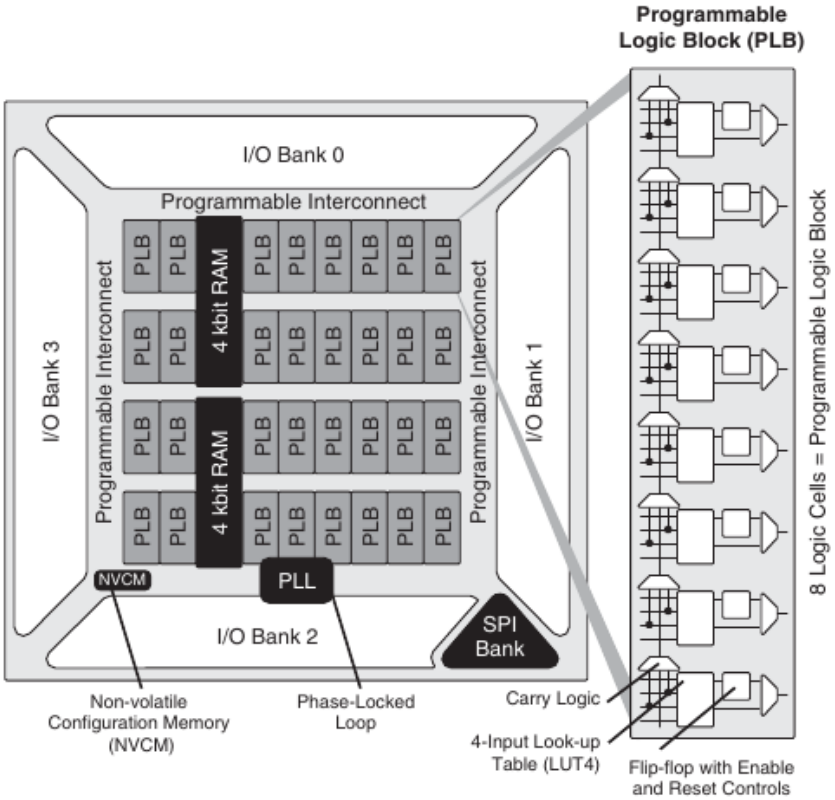


Abbildung 2.1: Aufbau des verwendeten iCE40 FPGAs (Quelle: iCE40 Datasheet[9])

[4]Bei einer Interrupt-Routine wird die aktuelle Programmausführung auf CPU-Ebene pausiert, zum Beispiel um auf Ereignisse externer Hardwaregeräte reagieren zu können

Zusätzlich ist die Anzahl von Funktionsblöcken bei FPGAs üblicherweise deutlich höher als als bei CPLDs, weswegen in dieser Arbeit ein FPGA-Chip verwendet werden soll.

Die in der Abbildung erkennbaren I/O-Bänke beinhalten GPIO-Pins die als Ein- oder Ausgänge konfiguriert und direkt mit den Logikblöcken ("PLBs") im FPGA verbunden werden können.

2.2.2 Datenzwischenspeicher: SRAM

Nach der Erfassung müssen die Daten zwischengespeichert werden. Dabei sind vor allem zwei Faktoren entscheidend:

- Die **Geschwindigkeit** des Speichers, die meist den maximalen Datendurchsatz der Gesamtschaltung bestimmt
- Die **Größe** des Speichers, die festlegt wie lange bei hohem Datendurchsatz aufgenommen werden kann

Die meisten nicht-flüchtigen Speicher sind aufgrund der unzureichenden Lese- und Schreib-Geschwindigkeiten nicht für diesen Verwendungszweck geeignet, weswegen sich die Verwendung von RAM-Speicher anbietet.
Bei Mikrocontrollern und kleineren FPGA-Boards werden aus Kostengründen und wegen der unkomplizierten Ansteuerung oft SRAM-Speichereinheiten verbaut. SRAM-Speicher hat meist sehr kurze Zugriffszeiten, allerdings bei einer geringen Speichergröße von wenigen Mbit. Der in dieser Arbeit verwendete SRAM-Speicher hat beispielsweise eine Größe von 4 Mbit (512 KB) und könnte damit maximal 62500 Events zwischenspeichern[5].
Bei kontinuierlicher Erfassung von Eingangssignalen mit einer Abtastrate von 100 Mhz entspricht dies einer Aufnahmezeit von unter einer Millisekunde[6]. Da allerdings keine Eingangssignale erwartet werden, bei denen Events mit einer Frequenz von 100 Mhz auftreten und außerdem bereits bei laufender Aufnahme Events aus dem RAM-Speicher entnommen und weiter übertragen werden, kann der SRAM-Speicher trotzdem im Sinne der Aufgabenstellung als Zwischenspeicher verwendet werden.
Als Alternative könnte DRAM-Speicher verwendet werden, der ein Vielfaches der Speicherkapazität bietet und damit auch längere Aufnahmen bei hoher Signaldichte ermöglichen würde. DRAM-Speicher erfordert allerdings im Vergleich zu SRAM ein kontinuierliches "Auffrischen" der Speicher-Inhalte durch eine Controller-Einheit und bedeutet dementsprechend deutlich mehr Aufwand und hohe Timinganforderungen bei der Implementierung.

2.2.3 Datenübertragung: SPI

Nach der Datenerfassung und Zwischenspeicherung werden die Daten an ein externes System übertragen, an dem sie weiterverarbeitet oder ausgewertet werden können. Zur Datenübertragung gibt es eine Vielzahl von Schnittstellen und Protokollen. Dabei werden die Daten im Normalfall "serialisiert", das heißt wenn ein Event aus 64 Bits besteht, werden die Bits nacheinander über eine einzige Datenleitung übertragen. Geläufige serielle Datenübertragungsverfahren sind vor allem UART, I^2C und SPI.

Bei einem **UART (Universal Asynchronous Receiver Transmitter)** werden die Daten über eine Empfangs- und eine Sendeleitung ausgetauscht. Es wird kein eigenes Taktsignal übertragen, weswegen auf beiden Seiten eine feste Übertragungsgeschwindigkeit ("Baud-Rate") eingestellt werden muss. UART-Schnittstellen sind weit

[5] Bei einer angenommenen Event-Größe von 64-Bit, und unter der Annahme dass der SRAM-Speicher ausschließlich zum Zwischenspeichern von Events verwendet wird

[6] $62500 * 10ns = 0.625ms$

verbreitet und verhältnismäßig einfach zu implementieren, dabei allerdings recht fehleranfällig und bei üblichen Baud-Raten ist die Übertragungsgeschwindigkeit stark limitiert (vgl. [10]).

I²C (Inter-Integrated Circuit) ist ein synchroner Datenbus, bei dem eine eigene Leitung für den Takt und eine Datenleitung verwendet wird. I²C arbeitet nach dem Master-Slave-Prinzip, das heißt es können auch mehrere Geräte miteinander kommunizieren. I²C untertützt Übertragungsraten bis zu 5 Mbit/s (unidirektional, vgl. [11]).

SPI (Serial Peripheral Interface) ist wie I²C ein synchroner Datenbus nach dem Master-Slave-Prinzip. Neben der Leitung für den Takt wird eine Daten-Leitung in Senderichtung und eine Datenleitung in Empfangsrichtung verwendet. Zusätzlich wird pro Gerät eine "Chip-Select" Leitung benötigt, um die Geräte adressieren zu können. SPI kann Übertragungsraten bis in den mehrstelligen Megabit-Bereich ermöglichen (vgl. [12]).

Aufgrund der hohen Übertragungsrate und der relativ unkomplizierten Implementierungsmöglichkeiten soll SPI für die Datenübertragung verwendet werden.

2.2.4 Steuerung der Aufnahme und sequentielle Programmabläufe

Neben der reinen Datenerfassung und -Übertragung werden noch Komponenten zur Steuerung und zur Kontrolle des Aufnahmevorgangs benötigt.
Grundsätzlich können die benötigten Vorgänge und Zustände (zum Beispiel das Starten und Stoppen der Aufnahme) direkt im FPGA umgesetzt werden. Als Kommunikationsweg zur Steuerung und Konfiguration kann dann wiederum SPI verwendet werden.
Bei längeren oder komplexeren Programmabläufen die keine zeitkritische Ausführung erfordern bietet sich die Verwendung eines zusätzlichen Mikrocontrollers an. Da die meisten PC-Systeme keine programmierbaren GPIO-Pins (als SPI-Schnittstelle) zur Verfügung stellen, kann ein Mikrocontroller außerdem als Brücke zwischen FPGA und Anwendersystem fungieren, und zum Beispiel die vom FPGA erfassten Daten über einen USB-Port zur Verfügung stellen.
Davon abgesehen wird unter Umständen noch zusätzliche Hardware- und Software-Infrastruktur benötigt um das FPGA und den Mikrocontroller zu programmieren und zu konfigurieren.

2.3 Auswahl der Software-Toolchain: IceStorm

Im Normalfall werden FPGAs mit den vom Hersteller angebotenen – kommerziellen – Software-Tools programmiert. Da der Quelltext dieser Tools nicht öffentlich zugänglich ist und FPGA-Hersteller keine relevanten Informationen zur Hardwarestruktur von FPGA-Chips veröffentlichen war es für Opensource-Tools bisher nicht möglich eine Alternative zu den kommerziellen Toolchains der Hersteller zu bieten.
Mit der IceStorm-Toolchain wurde erstmals die Hardwarestruktur der – relativ einfach aufgebauten – Lattice iCE40 Produkt-Serie[7] durch Reverse Engineering rekonstruiert, so dass ein vollständiger Arbeitsablauf für die Entwicklung von FPGA-Designs zur Verfügung steht.
Der Arbeitsablauf lässt sich in mehrere grundlegende Schritte unterteilen:

- Erstellen des Quelltexts für ein FPGA-Design in einer Hardwarebeschreibungssprache wie Verilog oder VHDL
- Prüfung der Funktion des Designs mit einem Software-Simulator
- Synthese des Designs: Der Quelltext wird in Funktionsblöcke umgewandelt die durch Hardware-Komponenten des FPGAs (wie Ein- und Ausgänge, Lookup-Tabellen für logische Funktionen, FlipFlops oder BRAM als Speichereinheiten, etc.) realisiert werden können
- "Place-and-Route": Es wird eine geometrische Anordnung der Funktionsblöcke auf dem FPGA-Chip entworfen ("place") und die Einzelkomponenten werden miteinander verbunden ("route"). Dabei müssen auch die zeitlichen Anforderungen der Schaltung berücksichtigt werden.
- Abschließend wird das Ergebnis des "Place-and-Route"-Vorgangs in ein Format gebracht mit dem direkt die Hardware konfiguriert werden kann – dem sogenannten "Bitstream" – und das Bitstream wird auf den FPGA-Chip übertragen

Die IceStrom-Toolchain bildet diesen Ablauf folgendermaßen ab:

- Als Hardwarebeschreibungssprache wird Verilog unterstützt[8]
- Die IceStorm-Toolchain enthält kein eigenes Tool für die Simulation, es können aber existierende Lösungen wie zum Beispiel `iverilog`[15] und `gtkwave`[16] (zur grafischen Darstellung der Simulationsergebnisse) verwendet werden
- Als Synthese-Tool dient `yosys`[17]
- Für den Pace-and-Route-Vorgang wird `arachne-pnr`[18] verwendet
- Für die Generierung und das Übertragen des Bitstreams stehen als Teil der IceStorm-Tools[19] `icepack` und `iceprog` zur Verfügung

[7] Es werden nicht alle Chips der iCE40-Serie unterstützt, die verwendbaren Produktvarianten sind auf der IceStorm-Website aufgelistet[13]

[8] Zum Zeitpunkt der Erstellung dieser Arbeit gibt es kein vollständiges VHDL-Frontend für das Synthese-Tool `yosys`. Mit dem Projekt "YODL" wurde eine Implementierung begonnen, die aber seit längerem nicht mehr weiterentwickelt wird (siehe [14]). Davon abgesehen können VHDL-zu-Verilog Konverter verwendet werden, wobei allerdings nicht immer von einer problemfreien Konvertierung ausgegangen werden kann

2.3.1 IcoSoc als Prototyping- und Implementierungsplattform

Vom selben Autoren[9] wie das Synthese-Tools `yosys` gibt es mit dem `picorv32` auch eine RISC-V CPU-Implementierung in Verilog und mit dem `IcoSoc`-Projekt eine Softwareumgebung die auf die einfache Verwendung des `picorv32`-Prozessors auf einem iCE40-basierten Entwicklungs-Board ("IcoBoard") ausgerichtet ist. Das `IcoSoc`-Projekt liefert eine Reihe von Modulen, mit denen zum Beispiel die Daten der GPIO-Pins direkt in den SRAM-Speicher gemappt werden können, oder die die Kommunikation mit externer Hardware (zum Beispiel über SPI oder UART) ermöglichen. Es werden also typische Mikrocontroller-Funktionalitäten direkt auf dem FPGA umgesetzt und es kann mit RISC-V kompatiblem C-Quellcode gearbeitet werden, wodurch viele Implementierungs-Aufgaben deutlich erleichtert werden. Dabei allerdings mit dem Nachteil dass die "Soft-CPU" auf dem FPGA-Board nur mit der vergleichsweise langsamen Geschwindigkeit von circa 20 Mhz läuft, und dementsprechend keine unmittelbar zeitkritischen Vorgänge übernehmen kann.
Das `IcoSoc`-Projekt wurde für diese Arbeit auf das verwendete FPGA-Board portiert, um Funktionsabläufe zu testen und weniger zeitkritische Abläufe zu implementieren.

2.4 Auswahl der Hardware

2.4.1 IceZero FPGA-Shield (iCE40HX4K)

Die Auswahl der Toolchain schränkt die verwendbare FPGA-Hardware auf die unterstützten iCE40-Modelle ein. Aufgrund des kompakten Formfaktors und des vergleichsweise günstigen Preises wurde das IceZero-FPGA-Shield für die Aufgabenstellung ausgewählt. Der auf dem IceZero-Board verwendete "ÌCE40HX4K"-Chip (im TQ144-Formfaktor) bietet 107 programmierbare Ein- bzw. Ausgangspins und laut Datenblatt 3520 "Logic Cells". Hardware-seitig hat der Chip allerdings den gleichen Funktionsumfang wie das größere "8K"-Model – die Limitierung auf 3520 Logic Cells wird durch eine Software-Sperre der vom Hersteller zur Verfügung gestellten "iCEcube2"-Toolchain erreicht.
Das heißt bei Verwendung mit der IceStorm-Toolchain bietet der Chip den gleichen Funktionsumfang wie die im Datenblatt abgebildete 8K-Variante:

Part Number	[...]	**HX4K**	**HX8K**
Logic Cells (LUT + Flip-Flop)		3,520	7,680
RAM4K Memory Blocks		20	32
RAM4K RAM bits		80K	128K
Phase-Locked Loops (PLLs)		2	2
Maximum Programmable I/O Pins		95	206
Maximum Differential Input Pairs		12	26
High Current LED Drivers		0	0

Tabelle 2.2: Hardware-Spezifikationen des verwendeten iCE40HX4K-Chips (vgl. [9])

Vom FPGA-Chip selbst abgesehen beinhaltet das IceZero-Board einen 4MBit (512KB) großen SRAM-Speicher und einen 64MBit (8MB) großen SPI-Baustein, der zur Speicherung des Bitstreams und der Konfiguration des FPGAs verwendet werden kann. Es ist ein 100 Mhz schneller Oszillator zur Erzeugung des Systemtakts verbaut und es sind drei benutzerprogrammierbare LEDs vorhanden. Die vorhandenen Ein- und Ausgänge liegen an 4 Pmod-Schnittstellen und einer 40-Pin-Buchsenleisten für den Anschluss an ein Raspberry Pi an (siehe [20]).

[9] Die Entwicklung der IceStorm-Toolchain wurde vor allem von Clifford Wolf vorangetrieben, das Projekt hat aber auch eine aktive Community die regelmäßige Beiträge leistet

2.4.2 Raspberry Pi Zero W

Das IceZero-Board ist für die Verwendung mit einem Raspberry Pi-Mikrocontroller vorgesehen und hat den gleichen Formfaktor wie das Raspberry Pi Zero (mit circa 5 x 3 cm das kleinste Modell der Raspberry Pi Serie).
Das Raspberry Pi dient für diese Arbeit hauptsächlich dem Zweck über die mit dem FPGA verbundenen GPIO-Pins eine Schnittstelle für die Konfiguration und den Datenaustausch mit dem FPGA zu liefern.
Prinzipiell steht mit dem Raspberry Pi ein Mikrocontroller mit großem Funktionsumfang und Leistungsreserven zur Verfügung, mit dem weitergehende Funktionen umgesetzt werden könnten. Das Raspberry Pi kann per USB direkt an einen Anwender-PC angeschlossen werden und könnte mithilfe verschiedener USB-Modi zum Beispiel als Netzerkgerät eine Web-Seite mit einer grafischen Darstellung der Ergebnisse anzeigen, oder als serielle Schnittstelle das SUMP2-Protokol zur Weitergabe der Aufnahme-Daten an den Anwender-PC implementieren.

3. Implementierung

Grundlage für die Arbeit mit dem FPGA-Board ist eine Möglichkeit generierte Bitstreams auf das FPGA-Board zu übertragen. Auf der Produkt-Seite des Herstellers des IceZero-Boards (Trenz Electronic)[20]wird für diesen Zweck auf das Git-Repository des `icotools`-Projekts verwiesen, in dem sich ein einfaches Testbeispiel zur Überprüfung des SRAM-Speichers befindet – und das C-Programm `icezprog` zum Flashen des Bitstreams.
Das `icezprog`-Programm implementiert über direktes Ansprechen von GPIO-Pins ("bit-banging") eine SPI-Schnittstelle zum Flash-Speicher des IceZero-Boards, und nach einem Neustart[1] konfiguriert sich das FPGA selbständig mit dem Inhalt des auf dem Flash-Speicher abgelegten Bitstreams.
Für die spätere Verwendung des `IcoSoc`-Projekts wird allerdings noch weitere Funktionalität benötigt. Wichtig ist hier vor allem die Möglichkeit Speicherinhalte mit einem "Offset" in den Flash-Speicher zu schreiben, da später zusätzlich zum Bitstream auch ein "appimage" auf dem Flash-Speicher abgelegt wird, das den auszuführenden C-Code für den `IcoSoc` enthält.
Das `icotools`-Projekt enthält mit dem Tool `icoprog` ein Programm das (unter anderem) Offsets unterstützt und in erster Linie zum Programmieren von Bitstreams für das Ico-Board gedacht ist. IcoSoc unterstützt außerdem Debugging-Funktionalitäten die über eine eigene 8 Bit breite parallele Schnittstelle zum Raspberry Pi realisiert sind. Diese Schnittstelle kann auch verwendet werden um ein `appimage` beim Start des IcoSocs direkt zu laden, ohne dass es vorher im Flash-Speicher abgelegt werden muss. Die parallele Schnittstelle wird ebenfalls über das `icoprog`-Tool angesprochen.
Um die selbe Funktionalität auch für das IceZero-Board zur Verfügung zu stellen wurde als erstes das `icoprog`-Tool entsprechend angepasst.

3.1 Anpassung des Tools zum Programmieren des Bitstreams (icoprog)

Das IcoBoard verfügt wie das IceZero-Board über einen 40-Pin-Header zum Anschluss an ein Raspberry Pi. Davon abgesehen kann das IcoBoard aber auch mit einem zusätzlichen FTDI-Interfaceboard über USB programmiert werden. Das `icoprog`-Programm enthält deswegen sowohl Methoden zur Programmierung via USB als auch über die GPIO-Pins des Raspberry Pi's, und noch eine zusätzliche GPIO-basierte Variante für ein anderes Board.

[1] Der Neustart wird durch setzten des mit dem FPGA verbundenen GPIO-Pins "CONFIG_RESET" ausgeführt

Ursprünglich war geplant eine Konfiguration für das IceZero-Board direkt in das `icoprog`-Tool zu integrieren, aufgrund der ohnehin schon eher unübersichtlichen Code-Basis wurden die benötigten Komponenten aber in ein eigenes Programm namens `icozctl` ausgelagert, dass dann später um Funktionen zur Steuerung der Aufnahme und Datenübertragung erweitert wurde.
Für die Anpassung mussten einige GPIO-Pins geändert werden, da beim IceZero-Board nicht alle Pins des GPIO-Headers mit dem FPGA verbunden sind. (Das IcoBoard verwendet einen iCE40-Chip im CT256-Formfaktor, weswegen deutlich mehr Pins zur Verfügung stehen als beim IceZero-Board[21]).
Davon abgesehen ist auf dem IcoBoard ein zusätzlicher MachXO2-Chip von Lattice verbaut, der (unter anderem) das SPI-Signal vom Raspberry Pi je nach gesetztem Chip-Select-Signal entweder direkt an das FPGA oder den Flash-Speicher weiterleitet. Beim IceZero-Board sind das FPGA und der Flashspeicher direkt an die gleichen GPIO-Pins des Raspberry Pi's angeschlossen (inklusive Chip-Select-Signal), ein gezieltes Ansprechen der einzelnen Komponenten ist deswegen nicht möglich, und die entsprechenden Chip-Select-Signale wurden aus dem Code entfernt. Im späteren Verlauf der Implementierung wurde entschieden die parallele Schnittstelle zu entfernen, da mit ihr keine bidirektional Kommunikation mit der nötigen Geschwindigkeit realisiert werden konnte. Stattdessen wurde mit den frei-gewordenen Pins eine SPI-Schnittstelle und ein zusätzlicher UART für einfache Debug-Ausgaben umgesetzt.
Ein direktes Flashen von `appimages` beim Starten des IcoSoc ist dementsprechend bei aktuellem Stand nicht möglich und die erweiterten Debug-Funktionen des IcoSoc sind nicht verfügbar. Davon abgesehen kann das `icozctl`-Tool auf dem IceZero-Board mit der gleichen Syntax wie `icoprog` für das IcoBoard verwendet werden. Zu Debug-Zwecken wurde `icoprog` außerdem um eine Option "-N" erweitert, mit der ein Offset für das Auslesen des aktuellen Flash-Speicher-Inhalts angegeben werden kann.

3.2 Portierung und nötige Anpassungen des Verilog-SoCs (IcoSoc)

Als nächstes wurde das IcoSoc-Projekt auf das IceZero-Board portiert.

3.2.1 Struktur des IcoSoc-Projekts

Zum Überblick soll erst ein kurzer Blick auf die Verzeichnisstruktur des Projekts geworfen werden:

```
./common
./common/firmware.c
./common/firmware.S
./common/icosoc_debugger.v
./common/icosoc_raspif.v
./common/picorv32.v
./common/riscv_flash.ld
[...]
./examples
./examples/event_recorder
./examples/hello
[...]
./mod_gpio
./mod_rs232
./mod_spi
```

```
[...]
./README
./icosoc.py
```

Im Verzeichnis `common/` befinden sich die Grundbausteine des IcoSoc-Systems. Dazu gehört die Verilog-Implementierung des PicoRV32-Prozessors und weitere grundlegende Systemkomponenten wie zum Beispiel die erwähnte Raspberry Pi Schnittstelle ("raspif") und außerdem die nötige Firmware zum Starten des IcoSocs und zum Laden der `appimages`. Zusätzlich findet sich hier auch das Linker-Skript für den C-Code der `appimages`.
Im Verzeichnis `examples/` liegen verschiedene Beispiel-Projekte, unter anderem ein ausführliches "Hello world"-Beispiel in dem die für IcoSoc-Projekte verfügbaren Komponenten vorgestellt werden — und das in dieser Arbeit entworfene Event-Recorder Projekt.
Darauf folgen die verfügbaren IcoSoc-Module jeweils in einem eigenen Unterverzeichnis. Das Modul "mod_gpio" stellt zum Beispiel die Funktionalität zur Verfügung, dass im C-Code GPIO-Pins als Ausgänge definiert werden können und ihr Ausgangspegel anschließend über das Schreiben an eine bestimmte Speicheradresse gesetzt werden kann. Zuerst muss das Modul in der Projekt-Konfigurationsdatei `icosoc.cfg` aktiviert werden:

```
board icoboard
mod gpio gpios
    address 2
    connect IO pmod2 pmod1
```

Die Benutzung im C-Code würde dann folgendermaßen aussehen:

```
// Alle GPIO Pins als Ausgang verwenden
icosoc_gpios_dir(0xffff);

// Ersten Pin auf '1' setzen
icosoc_gpios_set(0x0001);
```

Wobei der "set"-Befehl des Moduls als einfacher Schreibbefehl an eine Speicheradresse implementiert ist:

```
static inline void icosoc_@name@_set(uint32_t bitmask) {
    *(volatile uint32_t*)(0x20000000 + @addr@ * 0x10000) = bitmask;
}
```

Die Module selbst sind in Verilog geschrieben und bilden somit eine Brücke zwischen dem Timing-stabilen Verilog-Teil und dem langsameren C-Code.

Direkt im Projekt-Verzeichnis findet sich eine README-Datei und das Python-Skript `icosoc.py`. Das Python-Skript generiert aus der Projekt-Konfigurationsdatei mithilfe der verfügbaren Module und Grundkomponenten den eigentlichen Verilog-Code für den IcoSoc und kümmert sich um die ordnungsgemäße Verbindung und Konfiguration der Module. (Die Platzhalter "@name@" und "@addr@" in der Definition der `gpio_get()`-Methode werden hier zum Beispiel auch durch die in der Konfigurationsdatei festgelegten Werte ersetzt). Zusätzlich wird ein `Makefile` generiert, mit dem das Projekt synthetisiert, der C-Code kompiliert, und abschließend der erzeugte Bitstream und das `appimage` auf das FPGA-Board geflasht werden können.

3.2.2 Portierung des IcoSoc-Projekts

Für die Portierung waren gemäß der Struktur des IcoSoc-Projekts hauptsächlich Anpassungen im `icosoc.py` Skript nötig.
Das Python-Skript generiert unter anderem auch die `icosoc.pcf`-Datei, in der die Pin-Definitionen für die Synthese festgelegt werden. Da auf dem IceZero-Board ein anderes iCE40-FPGA-Modell verwendet wird, mussten hier alle Pin-Definitionen gemäß den Informationen aus dem IceZero-Board-Schaltplan[22] angepasst werden. Ein einfaches Beispiel wäre der Eingang des Systemtakts und die LED-Pins:

```
if board == "icezero":
    icosoc_pcf["10-std"].append("""
    set_io CLKIN  49  // icoboard: R9
    set_io LED1   110 // icoboard: C8
    set_io LED2   93  // icoboard: F7
    set_io LED3   94  // icoboard: K9
[...]
""")
```

Wie im Beispiel ersichtlich ist wurden die meisten durchgeführten Änderungen mit der Variable "board" parameterisiert, so dass die Unterstützung mehrerer Board mit vergleichsweise geringem Aufwand möglich wäre[2].

Die Pmod-Pins werden den verwendeten Modulen nach Bedarf zugewiesen und nach einem festen Namensschema benannt (siehe hierzu auch "Detaillierte Pinbelegung aller Pmod-Header").
Auch hier wurden die entsprechenden Pin-Definitionen angepasst.
Für IcoSoc-Module sind grundsätzlich nur Verbindungen zu Pmod-Pins vorgesehen. Für die geplante Funktionalität werden allerdings auch Verbindungen zu anderen FPGA-Pins und Verilog-Signalen (wie zum Beispiel dem CLKIN Pin) benötigt. Das Skript wurde so angepasst, dass Moduldefinitionen auch andere – in einer Liste festgelegte – Signale verwenden dürfen.

```
def make_pins(pname):
    [...]
    allowed_signals = ['CLKIN']
    if pname in allowed_signals:
        return [pname]
    [...]
```

Wie schon erwähnt wurde außerdem die "RASPIF"-Schnittstelle entfernt, und stattdessen ein einfacher UART für Debug-Ausgaben integriert.
Die Änderungen wurden analog zu einem `icotools`-Fork[3] auf Github durchgeführt, bei dem die gleichen Anpassungen für ein anderes iCE40-basiertes Board durchgeführt wurden ("BlackIce II"-Board, siehe [23]).
Davon abgesehen wurde ein unbenutztes "HRAM"-Signal aus dem Design entfernt und kleinere Änderungen an der Generierung des Makefiles durchgeführt. Der `icoprog`-Befehl wurde durch `icozctl` ersetzt und das Log-Level wurde auf die Stufe "-v2" reduziert. Außerdem muss dem Place-and-Route-Befehl (`arachne-pnr`) und der durchgeführten Timing-Analyse (`icetime`) mit der Option `-P tq144:4k` der Formfaktor des iCE40-Chips mitgegeben werden, damit die Pindefinitionen richtig zugeordnet werden können. Mit den

[2] Für die Untersützung mehrerer Boards wäre eine tiefergehende Umstrukturierung des IcoSoc-Projekts sinnvoll, die aber über den Umfang dieser Arbeit hinausgeht

[3] Der Fork enthält außerdem zusätzliche Beispiel-Projekte und unter anderem ein I²C-Modul

genannten Änderungen liegt ein voll funktionsfähiger Port des IcoSoc-Projekts für das IceZero-Board vor.

3.3 Implementierung des Event-Recorder Moduls

Zur Implementierung der Signalerfassung wurde ein eigenes IcoSoc-Modul namens `mod_triggerrec` erstellt.

3.3.1 GPIO-Eingänge

Die Erfassung der an den Pmod-Headern anliegenden Eingangssignale erfolgt analog zum GPIO-Modul, allerdings mit dem Unterschied dass die verwendeten Pins fest als Eingänge definiert werden. Dafür wird bei der Instantiierung des SB_IO-Primitves für den PIN_TYPE die Bit-Kombination `0000_01` gesetzt (siehe "Input Pin Function Table" aus der SiliconBlue ICE Technology Library[24, S. 73]).
Auf den Registerinhalt des so definierten SB_IO-Primitives kann dann in Verilog mit einem einfachen "wire" der entsprechenden Bit-Breite zugegriffen werden:

```
// gpio input
wire [IO_LENGTH-1:0] io_in;
```

3.3.2 Bus-Schnittstelle

Um Daten aus dem Verilog-Teil in den C-Teil transferieren zu können muss die Bus-Schnittstelle des IcoSoc in Verilog implementiert werden. Dafür werden – wie im GPIO-Modul – die Signale `ctrl_rd`, `ctrl_wr`, `ctr_addr` und `ctrl_wdat` als Eingänge `ctrl_rdat` und `ctrl_done` als Ausgänge des Moduls zur Verfügung gestellt. Die Bus-Schnittstelle wird (vereinfacht) folgendermaßen umgesetzt:

```
always @(posedge clk) begin
        if (!ctrl_done) begin
                ctrl_done <= 1;
                // write from bus to local register associated with address 4
                if (|ctrl_wr) begin
                        if (ctrl_addr == 4) local_register <= ctrl_wdat;
                end
                // read from local registers to bus
                if (ctrl_rd) begin
                        if (ctrl_addr == 4) ctrl_rdat <= local_register;
                end
        end
end
```

Für einige Register wird eine Länge von 64 Bit verwendet. Da der Bus eine Breite von 32 Bit hat (also nur 32 Bit in einem Takt übertragen kann), wird für diese Register eine zusätzliche Zustandsvariable (`ctrl_state`) verwendet, so dass beim ersten Zugriff immer die höheren 32 Bit übertragen werden und beim zweiten Zugriff (und zweiten Zustand) die unteren 32 Bit.

3.3.3 BRAM-Speicher für die Event-Definitionen

Da aus dem Verilog-Modul ein direkter Zugriff auf die Event-Definitionen und -Trigger möglich sein soll werden diese in einem Register-Array abgelegt.
Die Länge des Arrays ist parametrisiert, so dass die Anzahl der gewünschten Event-Trigger bei der Synthese angegeben werden kann. Bei einfacher Lese- und Schreib-Logik auf das Array wird dieses als BRAM synthetisiert, und es werden somit keine zusätzlichen 'Logic Slices' in Anspruch genommen.
Im Trigger-Array werden die 64 Bit langen Event-Definitionen jeweils in zwei aufeinanderfolgende 32-Bit Werte aufgeteilt, da dies die Bus-Logik etwas vereinfacht. Bei der Bus-Kommunikation wird die angeforderte Adresse (`ctrl_addr`) mit einem Offset[4] in den Index des Event-Registers-Arrays umgesetzt, so dass alle Array-Elemente direkt vom Bus ansprechbar sind.

3.3.4 Erkennung von Signaländerungen und Events

Der erste Schritt bei der Auswertung des GPIO-Registers ist die kontinuierliche Erkennung von Signaländerungen.
Dies wird im Verilog-Code durch einen einfachen Vergleich und ein Zwischenspeichern des alten Signalzustands erreicht:

```
// input capture
always @(posedge clk_fast) begin
        // increment counter if active
        if (status[0])
                counter <= counter + 1;

        // on input changes ..
        if ((io_buf2 != io_buf1)) begin
                data_in_fast <= { io_buf1, 1'b0, counter[46:0] };
                shift_in_fast <= 1;
        end
end

io_buf1 <= io_in;
io_buf2 <= io_buf1;
[...]
```

An dieser Stelle wird auch der Zähler erhöht, wenn das Modul in aktiviertem Zustand ist. Darüber hinaus wurde auch eine Event-Erkennung in Verilog begonnen. Bei der vorhandenen Implementierung wird bei der Synthese allerdings eine große Menge kombinatorischer Logik generiert, und es können nicht ausreichend viele Event-Definitionen umgesetzt werden (Die Kapazität des iCE40-Chips wird bei circa 8 Event-Definitionen erreicht, bei großer Auslastung der Logic Slices erhöhen sich auch die Place-and-Route-Zeiten auf Werte von einer halben Stunde oder mehr).
Ohne weitere Optimierungen ist die Event-Erkennung in Verilog deswegen nicht praktikabel und wurde bei aktuellem Stand der Arbeit wieder deaktiviert.
Es ist anzumerken, dass die Erkennung von Signaländerungen das Datenvolumens bei der Aufnahme bereits stark reduziert – insbesondere bei geringer Signaldichte.

[4] Die Adresse hat um der Bitbreite der Registerelemente zu entsprechen einen "Multiplikator" und wird deswegen zusätzlich per Rechts-Shift verkleinert bevor der Index generiert werden kann

3.3.5 Cross-Clock BRAM-Puffer

Da die Event-Erkennung im Verilog-Teil noch nicht abgeschlossen ist, aber auch um schnell aufeinander-folgende Signale ohne Verlust erfassen zu können, macht es Sinn bereits direkt auf dem FPGA ein Puffer für die Aufnahmedaten zu implementieren. Der Puffer soll wiederum als BRAM synthetisiert werden, da es sich um größere Datenmengen handelt und keine kombinatorische Logik verbraucht werden soll.
Bei FPGA-Implementierungen werden derartige Puffer häufig benötigt – und dabei oft mit der zusätzlichen Funktionalität, dass Daten zwischen unterschiedlich schnell getakteten Komponenten ausgetauscht werden. Der Austausch von Daten zwischen verschiedenen Taktdomänen ist dabei kein triviales Problem. Das `IcoSoc`-Projekt enthält in der Datei `icosoc_crossclockfifo.v` allerdings bereits eine Implementierung eines solchen Cross-Clock-Puffers, der für das `raspif`-Modul verwendet wird. Da keine weitere Dokumentation zum Puffer-Modul vorhanden war, wurde zuerst eine `iverilog`-Testbench geschrieben, die die erwartete Funktionsweise bestätigt hat. Das `crossclockfifo`-Modul konnte so erfolgreich als BRAM-Puffer in das Event-Recorder-Modul eingebunden werden.
IcoSoc-Module laufen im Normalfall mit dem Systemtakt des IcoSocs. Dieser wird durch eine PLL-Konfiguration generiert und liegt bei (circa) 20 Mhz.
Zusammen mit der bei der Portierung des `IcoSoc`-Projekts durchgeführten Anpassung, dass IcoSoc-Module auch andere Ein- und Ausgänge als die Pmod-Pins haben können, konnte so das 100 Mhz schnelle CLKIN-Signal an das Event-Recorder-Modul weitergereicht werden und damit eine Steigerung der Erfassungsgeschwindkeit auf 100 Mhz erreicht werden.

3.3.6 Simulation des Event-Recorder-Moduls mit iverilog

Das Verhalten des Event-Recorder-Moduls lässt sich mit einer `iverilog`-Testbench simulieren.
Durch die Bus-Schnittstelle kann dabei die erwartete Funktionalität geprüft werden ohne dass die anderen Komponenten des `IcoSoc`-Systems mitsimuliert werden müssen.

In der Testbench werden zwei “tasks” für den schreibenden und lesenden Bus-Zugriff erstellt, deren Verhalten einem tatsächlichen Bus-Zugriffs entspricht:

```
task ctrl_write(input [15:0] addr, input [31:0] data); begin
        ctrl_wr <= 1;
        ctrl_rd <= 0;
        ctrl_addr <= addr;
        ctrl_wdat <= data;
        @(posedge clk);
        while (!ctrl_done) @(posedge clk);
        ctrl_wr <= 0;
end endtask

task ctrl_read(input [15:0] addr); begin
        [...]
end endtask
```

Anschließend können in der Testbench Speicher-Zugriffe – analog zu den vom Modul definierten C-Funktionen – simuliert werden und ihr Ergebnis geprüft werden.

```
        // test write to status register (address 0x04)
        ctrl_write('h4, 'h01);
```

```
repeat (2) @(posedge clk);

// read status register
ctrl_read('h4);
repeat (2) @(posedge clk);

$display("Status: %s", (ctrl_rdat == 'h01)? "OK" : "Not OK");
```

Für die Ausführung der Testbenches ist jeweils ein kurzes Bash-Skript vorhanden, so kann im Modul-Verzeichnis `mod_triggerrec` zum Beispiel der Befehl

```
sh testbench.sh
```

ausgeführt werden, und die erzeugte VCD-Datei mit `gtkwave` geöffnet werden:

```
gtkwave testbench.vcd
```

Der folgende Auszug zeigt zum Beispiel einen Signalübergang von '0' auf '1' am ersten Pin des Event-Recorders, wobei der Signal-Wert (0x0001) und der entsprechende Zähler-Wert (0) über das Register `data_in_fast` in den Cross-Clock-FIFO übertragen werden. Es folgt ein Bus-Zugriff auf die Adresse 0x0C, bei dem der im FIFO gespeicherte Wert abgefragt und über das Register `data_out` an den Bus übertragen wird. Der Lese-Zugriff erfolgt in zwei Einzelschritten, bei den zuerst die oberen und dann die unteren 32 Bit übertragen werden (siehe `ctrl_stat` und `ctrl_done`-Signal).
(Der Signalübergang wird in der Simulation auch als Start-Event erkannt, weswegen der Zähler erst nach dem Signalwechsel aktiv wird).

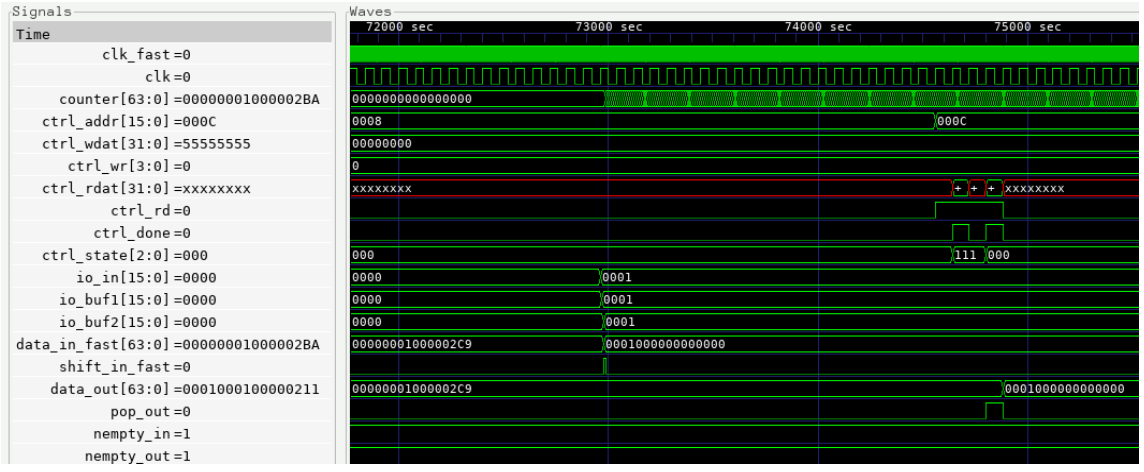


Abbildung 3.1: Auszug aus der Simulation des Event-Recorder-Moduls

3.4 Implementierung eines SPI-Slave-Moduls

Für den Austausch der Daten mit dem Raspberry Pi wird noch eine Komponente für die SPI-Kommunikation benötigt. Diese Komponente soll wieder als `IcoSoc`-Modul umgesetzt sein, um die Ausführung des C-Codes nicht unnötig zu verlangsamen. Das `IcoSoc`-Projekt enthält bereits ein SPI-Modul, allerdings handelt es sich hier um eine reine Master-Implementierung (vor allem zum Ansprechen externer Hardware wie zum Beispiel kleinerer

LCDs). Deswegen wurde noch ein zusätzliches SPI-Slave-Modul entworfen.
Die Bus-Logik funktioniert analog zum Event-Recorder-Modul, wobei im C-Teil eine "bi-direktionale" Übertragungsfunktion zur Verfügung gestellt wird. Es wird ein 32-Bit-Wert als Argument übergeben und gleichzeitig einen 32-Bit-Wert als Rückgabe geliefert.

```
static inline uint32_t icosoc_@name@_xfer(uint32_t value)
{
    *(volatile uint32_t*)(0x20000004 + @addr@ * 0x10000) = value;
    return *(volatile uint32_t*)(0x20000008 + @addr@ * 0x10000);
}
```

Das Argument sind dabei die über SPI zu sendenden Daten, und der Rückgabewert die empfangenen Daten.
Der SPI-Transfer ist dabei ähnlich einem Schieberegister implementiert, bei dem bei jeder steigenden Taktflanke ein Bit der Sende-Daten am MISO-Ausgang anliegt, und (nach einem Schiebevorgang) durch das am MOSI-Eingang anliegende Bit ersetzt wird. Bei den meisten SPI-Implementierungen werden dabei 8 Bit am Stück übertragen (vgl. [12]), durch die Übertragung von 32-Bit kann im vorliegenden Fall die Übertragungsrate etwas erhöht werden. Vom C-Teil aus ist zu beachten, dass es sich hier um eine blockierende Funktion handelt, das heißt nach einem Funktionsaufruf kann kein anderer Code ausgeführt werden, bis vom SPI-Modul 32 steigende Taktflanken am `PI_SCLK`-Eingang verarbeitet worden sind.

3.5 Zusammenführung der Module als Icosoc-Projekt

Die Steuerung des Programmablaufs für die Event-Aufnahme soll dann unter Verwendung des Event-Recorder- und SPI-Slave-Moduls im C-Teil der IcoSoc-Anwendung realisiert werden.

3.5.1 Projektkonfiguration

Dafür wird zuerst ein neues Projekt-Verzeichnis mit einer `icosoc.cfg`-Datei angelegt. In der Konfigurationsdatei wird das Ziel-Board festgelegt und die gewünschten Module definiert.

```
# event recorder
board icezero

# load appimage from flash
flashpmem

# capture input
mod triggerrec tr0
  address 2
  connect clk_fast CLKIN
  connect IO pmod4 pmod3

# spi slave
mod spi_slave spi0
  address 1
  connect cs PI_CE0
```

```
connect miso PI_MISO
connect mosi PI_MOSI
connect sclk PI_SCLK
```

Das `Makefile` kann aus einem anderen Projekt übernommen werden und benötigt keine besonderen Anpassungen.

3.5.2 SRAM-Puffer und Programmablauf

Der SRAM-Puffer und der gewünschte Programmablauf werden dann in der Datei `main.c` umgesetzt. Für den SRAM-Puffer wurde eine existierende FIFO-Implementierung von `mikrocontroller.net`(siehe [25]) verwendet und für 64-Bit-Inhalte angepasst.
`IcoSoc` führt den Code des `appimages` im Normalfall direkt vom Flash-Speicher aus, wodurch keine SRAM-Kapazitäten verbraucht werden, was aber auch mit einer verhältnismäßig schlechten Performance einhergeht. Performance-kritische Sektionen sollten aus dem SRAM-Speicher ausgeführt werden, weswegen dem SRAM-Puffer eine eigene Linker-Sektion zugewiesen wird, die dann im Linker-Skript in den SRAM platziert wird.

```
#define BUFFER_SIZE 8192
__attribute__ ((section (".text.buffer"))) struct Buffer {
  uint64_t data[BUFFER_SIZE];
  uint64_t read;  // always points to the oldest entry
  uint64_t write; // always point to a empty entry
} buffer = {{}, 0, 0};
```

Ähnlich verhält es sich auch mit der Hauptschleife `main()`, für die die vordefinierte `.text.sram`-Sektion verwendet wird.
Die Main-Schleife lässt sich vereinfacht folgendermaßen beschreiben:

```
__attribute__ ((section (".text.sram"))) int main(void)
{
        /* status:
         * 0: not running
         * 1: running with event detection
         * 2: running in "dump" mode
         */
        uint8_t status = 0;

        while (true) {
                // push events from BRAM buffer to SRAM buffer
                event = icosoc_tr0_get_fifo();
                if (event != 0) {
                        // add all events in dump mode
                        if (status == 3) {
                                BufferIn(event);
                                continue;
                        }
                        // event detection
                        for (uint8_t t = 0; t < MAX_EVENTS; t++) {
                                if (match) {
                                        // apply event function
```

```
                                        if (function)
                                                status = function;

                                        // push into SRAM buffer
                                        if ((status == 1) || function)
                                                BufferIn(event);
                                }
                        }
                        continue; // continue until BRAM buffer is empty
                }

                // init SPI transfer ..
                cmd = icosoc_spi0_xfer(0);

                // SPI commands
                switch (cmd)
                {
                        case 1-5:
                                // receive new event configuration
                                // reset counter, start/start
                                // recording / dump, etc
                                [...]

                        default:
                                // send data
                                if (BufferOut(&event)) {
                                        icosoc_spi0_xfer(event);
                                }
                                break;
                }
        }
}
```

Es werden also in immer erst alle Events aus dem BRAM-Puffer abgeholt – und dabei die Event-Erkennung durchgeführt wenn diese aktiviert ist. Falls ein Event erkannt wurde oder der "Dump"-Modus aktiv ist wird das Event im SRAM-Puffer abgelegt.
Wenn keine Daten im BRAM-Puffer liegen wird ein SPI-Transfer initiiert, der vom Raspberry Pi einen "Befehl" anfordert. Dies kann ein Kontrollbefehl zum Starten oder Stoppen der Aufnahme sein, oder eine neue Event-Definition oder aber ein Befehl zum Abholen der gespeicherten Daten – in diesem Fall wird ein Event aus dem SRAM-Puffer entnommen und per SPI an das Raspberry Pi übertragen.

3.5.3 Event-Erkennung

Für die Event-Erkennung kann aufgrund der "don't care"-Option für einzelne Trigger-Bits kein einfacher Vergleich verwendet werden.
Es ist aber trotzdem möglich eine Erkennung durch die Bitoperationen `Und` und einem `Exklusiv-Oder` direkt auf den Daten des Events und eines Event-Triggers durchzuführen. Als Beispiel soll folgender Event-Trigger mit einer Breite von 4 Bit angenommen werden:

Event-Trigger
1uxx

Tabelle 3.1: Beispielhafter Event-Trigger

Aus der Text-Definition des Event-Triggers wird nun ein binär-kodierter Trigger erzeugt der aus 3 mal 4 Bit besteht:

"Don't care"-Maske	Vorheriger Wert	Aktueller Wert
1100	1000	1100

Tabelle 3.2: Binärkodierung des Beispiel-Triggers

Für die "Don't care"-Maske werden alle "x" mit einer '0' eingetragen, alle anderen Werte mit einer '1'. Beim vorherigen und aktuellen Wert werden "Don't care"-Bits ebenfalls mit einer '0' eingetragen, alle anderen Trigger-Optionen mit dem jeweilig erwarteten Wert – für ein "u" (up) also beim vorherigen Wert eine '0' und beim aktuellen Wert eine '1'.

Die Event-Erkennung kann dann, wie im folgenden für zwei Beispiele veranschaulicht, duchgeführt werden:

	Beispiel 1		Beispiel 2	
	Vorheriger Wert	Aktueller Wert	Vorheriger Wert	Aktueller Wert
	1010	1110	1010	0110
Verundung mit "Don't care"-Maske	1010 & 1100 = 1000	1110 & 1100 = 1100	1010 & 1100 = 1000	0110 & 1100 = 0100
Exklusiv-Oder mit der Trigger-Definition	1000 ^1000 = 0000	1100 ^1100 = 0000	1000 ^1000 = 0000	0100 ^1100 = 1000
Beide Werte gleich 0?	wahr		falsch	

Tabelle 3.3: Beispielhafte Event-Erkennung auf Basis von Binäroperationen

Ein Event liegt genau dann vor, wenn beide Werte nach der XOR-Operation gleich Null sind.
Das beschriebene Vorgehen entspricht der Trigger-Erkennung beim SUMP2-Logikanalysator (vgl. zum Beispiel [26]). Die Implementierung der Event-Erkennung konnte aus zeitlichen Gründen nur exemplarisch getestet werden.

3.6 Implementierung des textbasierten Benutzerinterfaces

Abschließend werden noch Methoden zur Programmsteuerung und zum Datenaustausch mit dem Raspberry Pi benötigt.
Diese wurden in das Kommandozeilen-Tool `icozctl` integriert.
Das Einlesen der Konfigurationsdatei wurde mit der C++-Bibliothek `yaml-cpp` realisiert. Es wird für jedes definierte Event der Event-Trigger und (falls vorhanden) die Funktion ausgelesen, und eine entsprechende Binärkodierung erzeugt. Die so erzeugten Event-Konfigurationen können dann mit dem SPI-Befehl "1" zum FPGA übertragen werden.
Für die SPI-Kommunikation kann hier direkt das vom Linux-Kernel zur Verfügung gestellte `spidev`-Interface verwendet werden. Die benötigten Funktionen wurden ähnlich wie in dem in der Kernel-Dokumentation bereitgestellten `spidev_test`-Programm implementiert (siehe [27]).
Der SPI-Transfer ist auf eine Geschwindigkeit von 2 Mhz eingestellt, allerdings muss die Programmausführung zwischen den Transferfunktionen immer wieder pausiert werden, um dem C-Code des IcoSocs ausreichend Zeit zu geben die SPI-Daten zu verarbeiten (Insbesondere wenn Debug-Ausgaben auf den UART ausgeführt werden).
Ein einfacher "Start"-Befehl kann dann zum Beispiel folgendermaßen implementiert sein:

```
// send start command (0x04)
uint8_t rx[4], tx[4] = {0,0,0,4};
spi_xfer(tx,rx);
```

Die Byte-Reihenfolge muss hierbei im "Little-Endian"-Format angegeben werden, damit sie vom IcoSoc richtig interpretiert wird.
Für die Übertragung der aufgenommen Events wurde eine Funktion implementiert, die in einem Loop neue Daten vom Event-Recorder abholt und diese abspeichert. Das Speichern wird im Moment auf zweierlei Arten parallel ausgeführt. Zum einen werden die Daten im CSV-Format direkt auf die Standardausgabe geschrieben. (Alle anderen Ausgaben des `icozctl`-Tools werden wie im ursprünglichen `icoprog`-Tool auf die Fehlerausgabe weitergeleitet, so dass die Standardausgabe direkt in eine Datei geschrieben werden kann). Das Format der Ausgabe sieht dann folgendermaßen aus:

```
time_in_ns,pin_15,pin_14,pin_13,pin_12,pin_11,pin_10,pin_9,pin_8,
↪  pin_7,pin_6,pin_5,pin_4,pin_3,pin_2,pin_1,pin_0
206490,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1
[...]
```

Der Zeitstempel wird dabei in Nanosekunden angegeben.
Gleichzeitig erfolgt die Ausgabe zusätzlich im VCD-Format[5]. Die Ausgabedatei wird mit der Option `-o <Dateiname.vcd>` angegeben. Die Implementierung der VCD-Ausgabe orientiert sich an einem Konverter-Projekt auf Github (siehe [28]). Im VCD-Format wird jedem Pin ein Symbol als Abkürzung zugewiesen (vgl. [29]). Die entsprechende Zuordnung findet sich im Anhang unter "Pinbelegung der Pmod-Header des Icezero-Boards".

Die zusätzlichen Funktionen von `icozctl` werden auch in der Programmhilfe kurz beschrieben:

```
# icozctl -h
[...]

Start recording in event detection mode:
    icozctl -s [-o events.vcd]

Start recording in dump mode:
    icozctl -S [-o events.vcd]

Read a event configuration file and send it to the FPGA:
    icozctl -c config.yaml

Write the recorded events to a VCD file:
    icozctl -o events.vcd

Additional options:
    -v      verbose output
    -x      don't check flash contents after writing them
            (slightly faster operation with -f)
    -O n    offset (in 64 kB pages) for -f, -F and -e
    -N n    number of events to record for -o
```

[5] Die Angabe einer VCD-Ausgabedatei ist im Moment obligatorisch

4. Anwendungsfall: Jitter-Analyse von Software-generierten MIDI-Clock Signalen

Als Beispiel für ein zeitkritisches Signal sollen im Folgenden mehrere MIDI-Clock Signale mit dem Event-Recorder untersucht werden, und dabei der allgemeine Ablauf einer Event-Aufnahme und der nachfolgenden Analyse geschildert werden. Das MIDI-Clock Signal wird in den untersuchten Fällen software-basiert (auf einem normalen Anwender-System) generiert, deswegen ist von einer messbaren zeitlichen Fluktuation ("Jitter") auszugehen. Das heißt die Clock-Signale treten nicht exakt zum erwarteten Zeitpunkt sondern leicht zeitlich verfrüht oder verspätetet auf.
MIDI ist ein 1983 spezifizierter Standard für den Austausch von Steuerinformationen zwischen elektronischen Musikinstrumenten und wird trotz einiger signifikanter Limitierungen seit über 35 Jahren nahezu unverändert für praktisch alle elektronischen Musikinstrumente als Standardschnittstelle verwendet. MIDI bietet mit der "MIDI-Clock" eine Möglichkeit mehrere Geräte zeitlich zu synchronisieren. So könnte zum Beispiel ein Synthesizer mit der Musiksoftware auf einem PC synchronisiert werden, um tempo-abhängige Arpeggios[1] zu spielen.
Dabei verwendet MIDI zur Datenübertragung das gleiche Protokoll wie ein UART bei einer Übertragungsgeschwindigkeit von 31250 Bit/s (vgl. [30]). Die meisten MIDI-Nachrichten setzen sich aus einem Statusbyte und zwei darauf folgenden Datenbytes zusammen, für die MIDI-Clock werden allerdings nur einzelne Bytes versendet.
Zuerst wird ein Start-Byte (0xFA) übertragen, dann wird – abhängig vom Tempo – 24 mal pro Viertelnote ein "Clock-Tick" (0xF8) gesendet, und abschließend ein Stopp-Bit (0xFC) (siehe [31]).

[1] Ein Akkord bei dem die einzelnen Töne nicht gleichzeitig, sondern zeitversetzt nacheinander gespielt oder ausgelöst werden

Eine einfache Implementierung in Python mithilfe des `python-rtmidi`-Moduls könnte folgendermaßen aussehen:

```
# MIDI CLOCK TEMPO (beats per minute)
BPM = 80
# define clock messages
clock_start = [0xFA]
clock_tick  = [0xF8]
clock_stop  = [0xFC]
[...]
    # calculate clock period
    clock_period = 60 / (BPM * 24)

    # send start byte
    midiout.send_message(clock_start)

    # run
    while (True):
        midiout.send_message(clock_tick)
        time.sleep(clock_period)
```

4.1 Test-Setup: USB-Midi mit Teensy LC

Um das MIDI-Signal auswerten zu können wird ein Mikrocontroller ("Teensy LC") verwendet, der per USB an einen PC angeschlossen werden kann, und über die USB-Schnittstelle ein MIDI-Gerät simuliert. Das heißt am PC wird eine virtuelle MIDI-Schnittstelle erzeugt, und alle ankommenden MIDI-Daten werden direkt zum Mikrocontroller übertragen. Der Mikrocontroller wartet auf das Start-Byte einer MIDI-Clock und setzt einen GPIO-Pin auf "1" um dem Event-Recorder den Anfang der Aufnahme zu signalisieren. Danach wird bei jedem empfangenen Clock-Tick ein zweiter GPIO-Pin kurz auf "1" gesetzt und anschließend wieder auf "0". Dies soll beim Event-Recorder als Event erkannt werden. Beim Empfangen des Stopp-Bytes wird der erste GPIO-Pin auf "0" gesetzt und die Aufnahme wird beendet.
Der Teensy-Mikrocontroller kann mit der Arduino-IDE programmiert werden und es steht eine MIDI-Bibliothek zur Verfügung, was eine kurze und unkomplizierte Umsetzung erlaubt:

```
[...]
// midi clock start handler (0xFA)
void onStart() {
  digitalWrite(0, HIGH);
}

// midi clock tick handler (0xF8)
void onClock() {
    digitalWrite(1, HIGH);
    digitalWrite(1, LOW);
}

// midi clock stop handler (0xFC)
void onStop() {
  digitalWrite(0, LOW);
```

```
}

void setup() {
  // pin setup
  [...]
  // register midi handlers
  usbMIDI.setHandleStart(onStart);
  usbMIDI.setHandleStop(onStop);
  usbMIDI.setHandleClock(onClock);
}

void loop() {
  usbMIDI.read();
}
```

4.2 Einrichten des Projekts

Die vollständige Einrichtung des Raspberry Pi Zero W soll hier aus Platzgründen nicht beschrieben werden (es steht aber eine detailliertere Anleitung im Projekt-Wiki zur Verfügung). Der grundsätzliche Ablauf ist wie folgt:

1. Download, Kompilieren und Installation der `IceStorm`-Toolchain auf dem Anwender-PC (Linux-System)

   ```
   # siehe http://www.clifford.at/icestorm/#install
   ```

2. Download, Kompilieren und Installation der RISC-V Toolchain auf dem Anwender-PC

   ```
   git clone git@github.com:cliffordwolf/picorv32.git
   cd picorv32
   make download-tools
   make -j$(nproc) build-tools
   ```

   Das Kompilieren der RISC-V `gcc`-Toolchain kann je nach Rechenleistung mehrere Stunden dauern.

3. Download des Git-Projekts auf dem Anwender-PC (Linux-System)

   ```
   cd ..
   git clone https://github.com/dm7h/icozsoc.git
   ```

4. Einrichten der SSH-Verbindung zum Raspberry Pi Zero W

   ```
   # siehe zum Beispiel:
   ↪  https://www.raspberrypi.org/documentation/remote-
   ↪  access/ssh/passwordless.md

   # Exportieren des SSH-Hosts, z.B. pi@zero oder pi@192.168.1.43:
   export SSH_RASPI=pi@zero
   # eine SSH-Verbindung sollte dann ohne Passwort-Abfrage möglich sein:
   ssh $SSH_RASPI
   ```

5. Download, Kompilieren und Installation des `icozctl` Git-Projekts auf dem Raspberry Pi
   Zur Installation des `icozctl`-Tools wird auf dem Raspberry Pi folgendes ausgeführt:

   ```
   git clone https://github.com/dm7h/icozctl
   cd icozctl
   sudo make install
   ```

6. Generieren und Programmieren des Bitfiles mithilfe des zur Verfügung gestellten `Makefiles`
   Wenn `icozctl` erfolgreich auf dem Raspberry Pi installiert wurde, kann auf dem Anwender-PC das FPGA-Bitstream und die IcoSoc-Anwendung kompiliert und via Raspberry Pi auf die Hardware geflasht werden:

   ```
   cd ../icosoc/examples/event-recorder/
   make prog_flash
   ```

4.3 Konfiguration der Event-Trigger

Um die Events zu konfigurieren wird zunächst per SSH auf das Raspberry Pi gewechselt. Im Ordner des `icozctl`-Tools befindet sich die Event-Konfigurationsdatei `config.yml`. Für die Aufnahme wird nicht zwangsläufig eine Event-Erkennung benötigt, es kann aber aber trotzdem ein Start-, Stopp- und Clock-Event definiert werden um unnötige Aufnahmedaten zu minimieren:

```
# event configuration
events:
  - start:
          trigger: u
          funcion: start

  - stop:
          trigger: d
          function: stop

  - clk_up:
          trigger: 1u

  - clk_down:
          trigger: 1d
```

4.4 Vorbereiten der Hardware

Bevor die Event-Aufnahme gestartet werden kann müssen die am Mikrocontroller konfigurierten Ausgangs-Pins mit den Eingangs-Pins des Event-Recorders verbunden werden. In unserem Fall wird der Ausgang “0” und “1” des Teensy LC mit den Eingängen “0” und “1” am Pmod-Header P3 des IceZero-Boards verbunden und zusätzlich wird mit einen “GND”-Pin eine Masse-Verbindung hergestellt. Siehe zur Pin-Belegung auch den Anhang “Pmod-Pinbelegung für die Event-Aufnahme”.

4.5 Durchführen der Event-Aufnahme

Die Auswertung soll zu Anschauungszwecken das VCD-Dateiformat verwenden. Die Aufnahme kann dann mit folgendem Befehl gestartet werden:

```
icozctl -c config.yml -o midi_clock.vcd
```

Das Programm wartet nun auf Eingangsdaten, die zum Beispiel durch das Starten der in Python implementieren Midi-Clock oder durch ein Musik-Programm geliefert werden können. Die Aufnahme kann zu jedem Zeitpunkt durch die Tastenkombination Strg+c beendet werden. Alternativ kann `icozctl` auch mit der Option "-N" gestartet werden, um die Aufnahme nach einer festgelegten Anzahl von Samples zu beenden.

```
icozctl -N 512 -c config.yml -o midi_clock.vcd
```

Das Ergebnis der Aufnahme kann dann zum Beispiel mit dem Programm `gtkwave`[16] oder mit `pulseview`[32] grafisch dargestellt und überprüft werden:

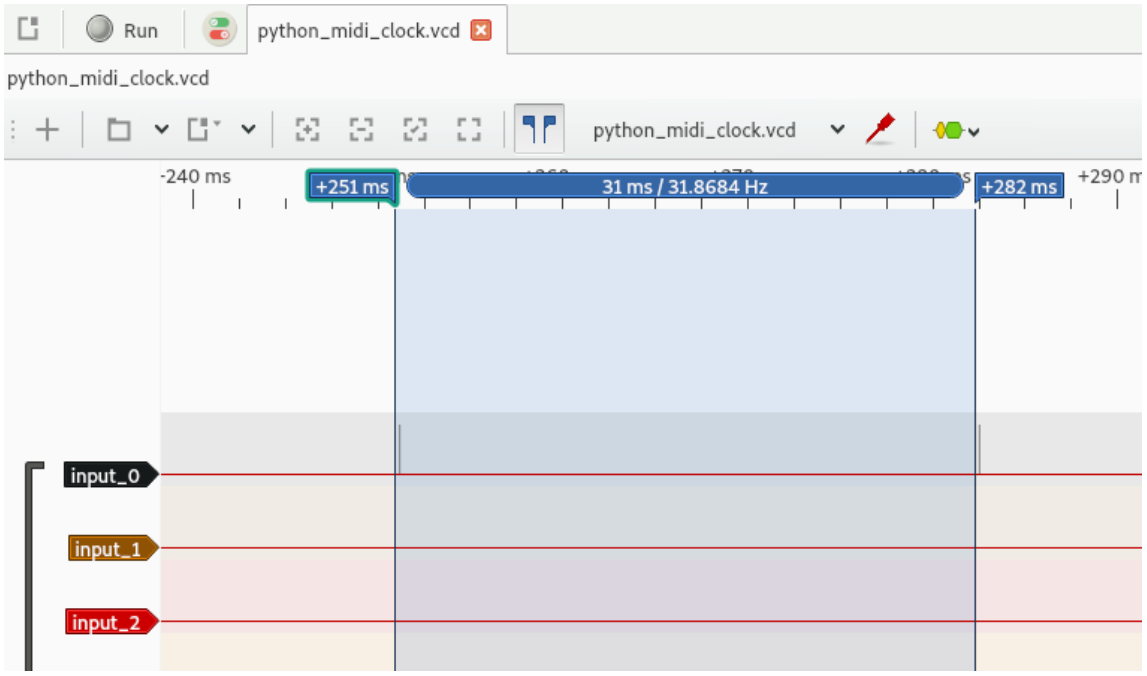


Abbildung 4.1: Darstellung eines aufgenommenen
Midi-Clock-Signals mit Pulseview (Quelle: Pulseview-Screenshot)

4.6 Analyse der Ergebnisse

Für die Analyse der Daten kann zum Beispiel ein Python-Skript verwendet werden. Um Aussagen über den Jitter des Signals machen zu können bietet sich die Generierung eines Histogramms an, bei dem auf der x-Achse die Abweichung zum erwarteten Clock-Signal und auf der y-Achse die Häufigkeit der Abweichung dargestellt wird.

Für das Einlesen der VCD-Datei wird das Python-Modul `Verilog_VCD`[33] verwendet:

```
import Verilog_VCD
vcd = Verilog_VCD.parse_vcd('midi_clock.vcd')
```

Im VCD-Dateiformat wird jedem Signal ein Symbol als Abkürzung zugewiesen, auf die Daten des ersten Signals der Datei kann zum Beispiel mit der Abkürzung "!" zugegriffen werden (siehe "Pmod-Pinbelegung für die Event-Aufnahme" für eine Zuordnung der Signal-Abkürzungen). Die Zeitstempel sind in der Python-Liste unter dem Index "tv"("time value") abrufbar. Der zeitliche Abstand zum jeweils vorhergehenden Event (Zeit-Delta) kann folgendermaßen berechnet werden:

```
# get the first time value
last = vcd["!"]["tv"][0][0]

# calculate time deltas
deltas = []
for item in vcd["!"]["tv"][1:]:
    deltas.append(item[0]-last)
    last = item[0]
```

Die Daten werden in ein numpy-Array umgewandelt und es wird die Differenz zum erwarteten Zeit-Delta gebildet. Anschließend werden sie in eine pandas-Zeitserie konvertiert, wodurch automatisch statistisch relevante Kennwerte wie der Mittelwert und die Standardabweichung berechnet und angezeigt werden können:

```
np_deltas = np.array(deltas)
expected_delta = 31250000 # expected clock period in nanoseconds
np_deltas -= expected_delta
print(deltas_series.describe())
```

Die Ausgabe sieht zum Beispiel[2] folgendermaßen aus:

```
count                         86
mean     0 days 00:00:00.000125
std      0 days 00:00:00.000026
min      0 days 00:00:00.000053
25%      0 days 00:00:00.000109
50%      0 days 00:00:00.000117
75%      0 days 00:00:00.000131
max      0 days 00:00:00.000227
```

Abschließend kann mithilfe der matplotlib-Bibliothek ein Histogramm in Millisekunden-Auflösung erzeugt und angezeigt werden:

```
(deltas_series/pd.Timedelta(milliseconds = 1)).hist()
[...]
plt.show()
```

[2]Die Werte stammen von der oben beschriebenen Python-Implementierung, das Zeit-Delta wird hier in Mikrosekunden-Auflösung angezeigt

4.6.1 Software MIDI-Clock: Python-Implementierung

Für die erste Messung wurde die Python-Implementierung verwendet.

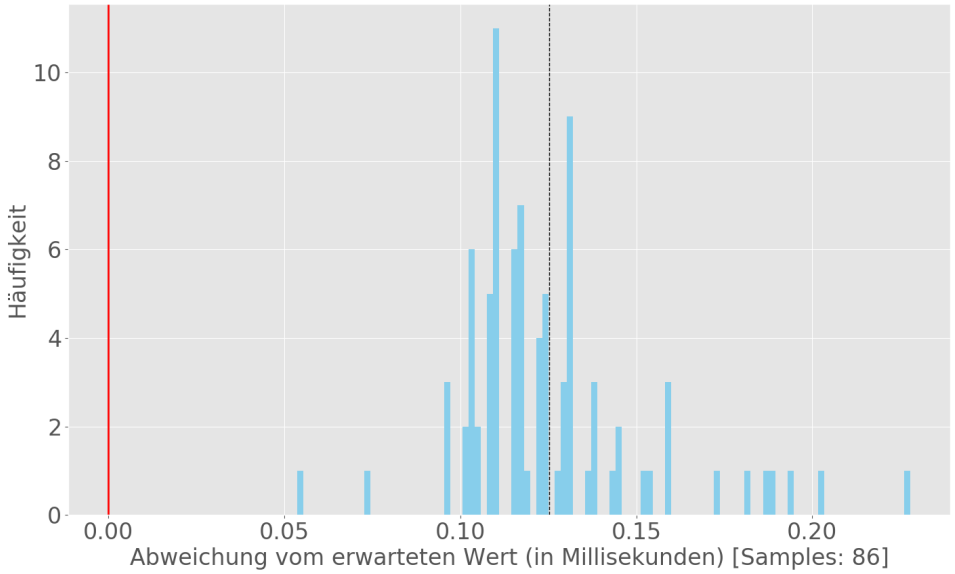


Abbildung 4.2: Histogramm der Python-generierten MIDI-Clock

Die erwartete Abweichung (0) ist mit der roten Linie markiert, der Mittelwert mit der gestrichelten schwarzen Linie.
Auffällig ist hier vor allem dass die MIDI-Clock langsamer läuft als erwartet, da eine konstante positive Abweichung von ca. 0,125 Millisekunden vorliegt. Dies lässt sich dadurch erklären, dass bei jedem Ausführen des `midiout.sendmessage(clock_tick)`-Befehls zusätzlich zur berechneten Clock-Periode Zeit verbraucht wird. Der Jitter liegt bei circa 0,2 Millisekunden und ist damit vergleichsweise niedrig.

4.6.2 Software MIDI-Clock: Renoise und Reaper

Anschließend wurde die MIDI-Clock von den Musikprogrammen Renoise und Reaper aufgenommen[3].

[3]Es wurden (von den Vorlieben des Autors abgesehen) keine besonderen Auswahlkriterien verfolgt, beide Programme haben eine kostenlose Demo-Version mit der die Funktionalität geprüft werden kann

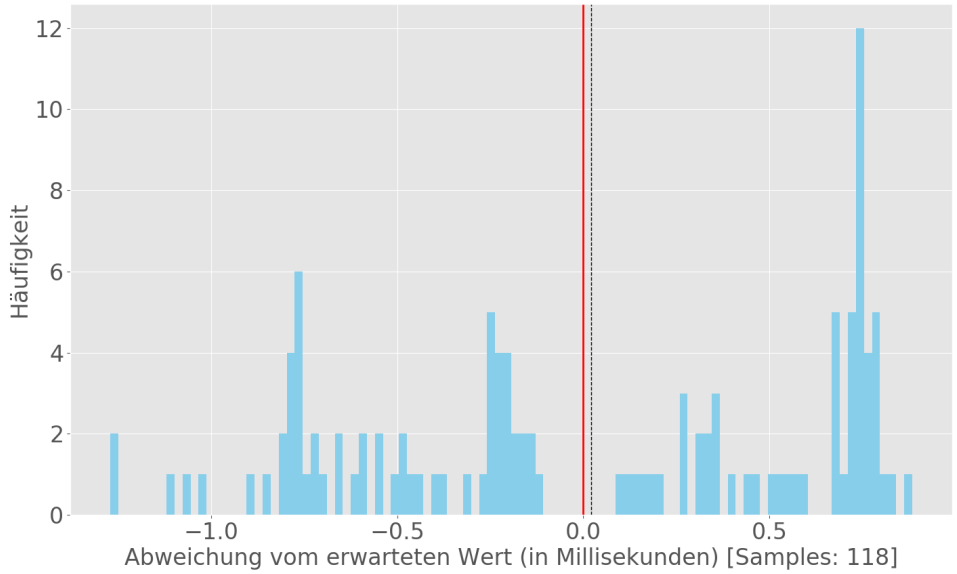


Abbildung 4.3: Histogramm der MIDI-Clock des Musik-Programms Renoise

Bei der MIDI-Clock von Renoise zeigt sich ein Jitter von fast 2 Millisekunden – ein eher schlechter Wert bei dem sogar hörbare Konsequenzen auftreten könnten. Die Wahrnehmung von zeitlichen Verzögerungen ist stark vom betroffenen Audio-Material abhängig. Bei einem zeitlichen Versatz von zwei identischen Signalen kann das zweite Signal bereits ab 0,6 Millisekunden als Reflexion wahrgenommen werden, und beeinflusst dementsprechend den Charakter des Audio-Materials. Zu einem differenzierbaren Echo-Effekt kommt es erst bei Zeitverschiebungen von 20 Millisekunden oder mehr (vgl. [34, S. 29]).

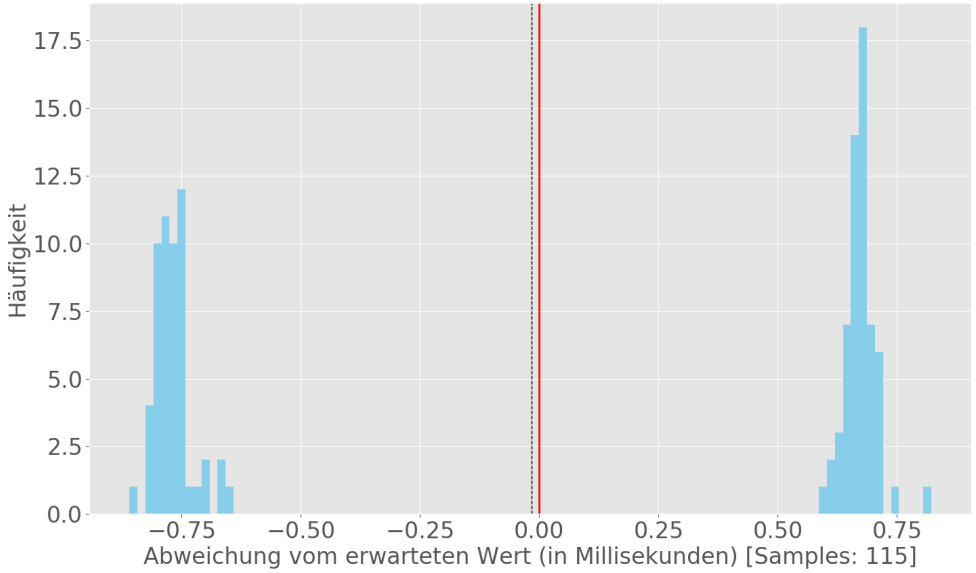


Abbildung 4.4: Histogramm der MIDI-Clock des Musik-Programms Reaper

Bei der MIDI-Clock von Reaper fällt der Jitter mit circa 1,5 Millisekunden etwas geringer aus und zeigt eine starke ausgeprägte bimodale Verteilung. Auch dies ist – inbesondere im Vergleich zur "naiven" Python-Implementierung – ein eher schlechter Wert.

4.6.3 Hardware MIDI-Clock: Midipal

Abschließend wurde noch die auf einem ATMega328p-Mikrocontroller basierende Hardware-MIDI-Clock des "MIDIpal"[4] getestet. Das Clock-Signal wurde über den MIDI-Eingang einer RME Digiface PCI-Soundkarte erfasst und mit einer Software-Loopback auf das virtuelle MIDI-Interface des Teensy LC geroutet.

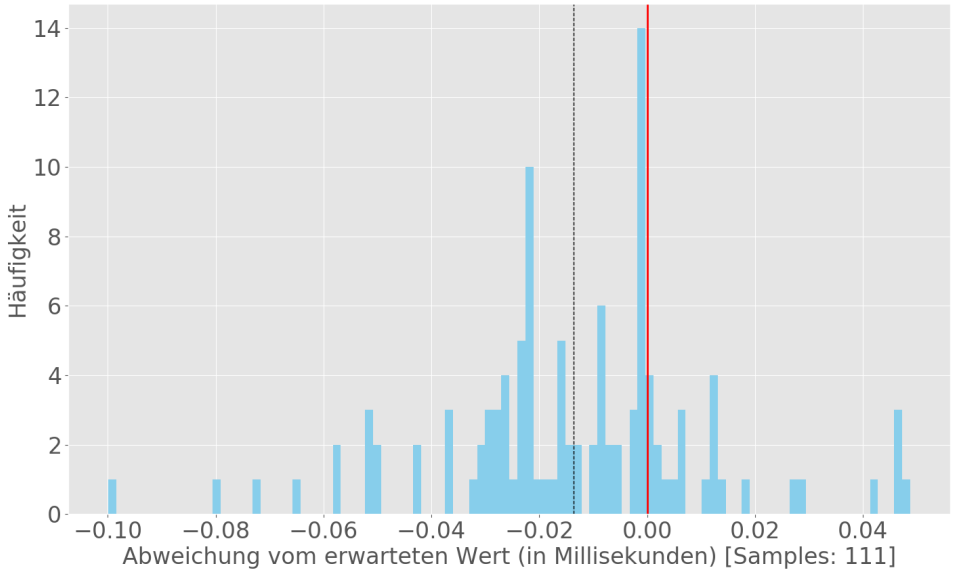


Abbildung 4.5: Histogramm einer Mikrocontroller-generierten Hardware-MIDI-Clock (Mutable Instruments MidiPal)

Die Messung liefert ein deutlich stabileres Ergebnis mit einem Jitter von circa 0,15 Millisekunden. Es zeigt sich, dass die Mikrocontroller-basierte Lösung den beiden Software-Produkten in Hinblick auf die Stabilität des Clock-Signals überlegen ist.

Zu allen Messungen ist anzumerken, dass der Teensy-Mikrocontroller (der mit einer Taktfrequenz von 40 Mhz läuft) einen – nicht näher bestimmten – verfälschenden Einfluss auf das Endergebnis hat. Alle Messungen wurden wiederholt mit variierenden Samplezahlen durchgeführt, und die beschriebenen Charakteristika sind ohne nennenswerte Abweichungen reproduzierbar. Dementsprechend kann davon ausgegangen werden, dass die vom Teensy verursachte Messverfälschung für die Analyse vernachlässigbar ist.

[4] Das "MIDIpal" ist ein Open-Source-Hardwareprodukt der Firma Mutable Instruments, das allerdings nicht mehr kommerziell angeboten wird (siehe [35])

5. Fazit

In der vorliegenden Arbeit wurde das IcoSoc-Projekt auf das IceZero-FPGA-Shield portiert, und somit eine flexible Plattform für die Entwicklung von Open-Source FPGA-Projekten zur Verfügung gestellt.
Auf Basis der IcoSoc-Komponenten wurde ein System zur Logikanalyse entworfen, mit dem benutzerdefinierte Events in hoher zeitlicher Auflösung aufgenommen werden können.
Die zeitkritischeren Komponenten wurden dabei als Verilog-Module implementiert, wohingegen die Steuerung des Programmablaufs mithilfe der RISC-V CPU "PicoRV32" als C-Code auf dem FPGA umgesetzt wurde. Die Event-Erkennung wurde dabei im langsameren C-Code umgesetzt, da die Verilog-Implementierung zu viel FPGA-Ressourcen verbraucht hat und aus zeitlichen Gründen keine Optimierung mehr durchgeführt werden konnte. Am Beispiel von verschieden MIDI-Clock-Signalen wurde der Ablauf einer Event-Messung erläutert und eine beispielhafte Analyse der aufgenommenen Daten durchgeführt.
In Hinblick auf die Event-Erkennung, aber auch auf die Gesamt-Performance des Event-Recorders besteht noch Optimierungspotential.

6. Aussicht

Die Implementierung der Event-Erkennung im Verilog-Teil sollte beim weiteren Vorgehen mit Priorität behandelt werden. Dies ist insbesondere für die Verwendung der eingeführten Event-Funktionen (Starten und Stoppen der Aufnahme und des Dump-Moduses) von Bedeutung, da diese Funktionen im C-Teil mit deutlicher Verzögerung ausgeführt werden. Wenn zum Beispiel ein bestimmtes Event den Dump-Modus auslösen soll, um direkt darauf eine Folge von relevanten Signaländerungen zu untersuchen, werden diese Signale unter Umständen noch gar nicht erfasst, da die Funktion im C-Teil nicht rechtzeitig ausgeführt wurde.
Davon abgesehen würde aber auch der Gesamt-Datendurchsatz des Event-Recorders stark von einer Verilog-Implementierung profitieren.
Ähnliches gilt auch für den SRAM-Puffer und die restlichen Programm-Komponenten. Es konnten keine detaillierteren Performance-Analysen mehr durchgeführt werden, es ist aber davon auszugehen, dass die Verwendung des Soft-Cores den Datendurchsatz um ein Vielfaches verringert. Es wäre also zu überlegen, ob die Funktionalität des C-Codes nicht komplett in Verilog umgesetzt wird.
Eine weitere Optimierung bietet sich bei der der Erfassungsgeschwindigkeit der Eingangsdaten an: Das `SB_IO`-Primitiv der Lattice Technology Library bietet die Funktion Eingänge sowohl bei steigender als auch bei fallender Taktflanke abzufragen und jeweils in einem eigenen Register abzulegen (siehe [24]). Die Erfassungsgeschwindigkeit könnte so ohne größere Code-Änderungen auf 200 Mhz verdoppelt werden.
Einen Ansatz für die Optimierung der Event-Erkennung bietet das Trigger-Modul des SUMP2-Analysators (in der "Demon core"-Version). Dabei werden die Bit-Operationen der Trigger-Logik in Blöcke von je 4 Bits Breite aufgeteilt, die dann bei der Synthese direkt als Lookup-Tabellen umgesetzt werden können (Das Vorgehen ist ausführlich im Quellcode dokumentiert, siehe [26]). Dieses Vorgehen sollte ohne größere Modifikationen auf die Event-Trigger angewandt werden können.
Davon abgesehen bietet die "Demon core"-Version eine interessante Erweiterung für komplexere Trigger-Bedingungen. Es können mehrere Trigger-Bedingungen mit logischen Funktionen verknüpft werden und Sequenzen von Trigger-Bedingungen definiert werden (vgl. [36]). Dies würde es zum Beispiel ermöglichen ein serielles Midi-Clock-Signal ohne einen zusätzlichen Mikrocontroller als Event zu erkennen und damit auch ohne die zeitlich Verfälschung des Mikrocontrollers aufnehmen zu können.
Auch die erweiterten Trigger-Bedingungen sollten sich in den Event-Recorder integrieren lassen, und würden damit eine wertvolle Ergänzung zum bestehenden Funktionsumfang liefern.

Abkürzungen

I²C Inter-Integrated Circuit. 9, 10, 38, *Glossary:* I²C

API Application Programming Interface. 3, 38, *Glossary:* API

CPLD Complex Programmable Logic Device. 2, 8, 38, *Glossary:* CPLD

CSV Comma-seperated values. 26, 38, *Glossary:* CSV

FPGA Field Programmable Gate Array. 2, 8, 38, *Glossary:* FPGA

GPIO General Purpose Input / Output. 10, 38, *Glossary:* GPIO

MIDI Musical Instrument Digital Interface. 27

PLL Phase-locked loop. 8, 38, *Glossary:* PLL

RAM Random-Access Memory. 8

SPI Serial Peripheral Interface. 9, 10, 38, *Glossary:* SPI

UART Universal Asynchronous Receiver Transmitter. 3, 9, 27, 38, *Glossary:* UART

VCD Value Change Dump. 26, 30, 38, *Glossary:* VCD

VHDL Very High Speed Integrated Circuit Hardware Description Language. 11, 38, *Glossary:* VHDL

Glossar

I²C Synchroner, serieller Datenbus nach dem Master-Slave-Prinzip, der Übertragunsraten bis zu 5 Mbit/s ermöglicht (vgl. [11]). 10, 38

API Schnittstelle zur Anwendungsprogrammierung. Erlaubt zum Beispiel die Verwendung von Programmkomponeneten durch eine exteren Skript-Sprache wie Python (vgl. **wiki:API**). 38

CPLD Im Vergleich zu FPGAs einfacher aufgebaute programmierbare logische Schaltungen (vgl. [37]). 2, 8, 38

CSV Text-basiertes Dateiformat bei dem die Daten durch ein Trennzeichen (zum Beispiel einem Komma) strukturiert werden (vgl. **wiki:CSV**). 38

FPGA Ein rekonfigurierbarer Chip dessen Schaltungsstruktur in einer Hardwarebeschreibungssprache (wie VHDL oder Verilog) frei programmierbar ist (vgl. [38]). 2, 8, 38

GPIO Frei programmierbarer Kontakt der zum Beispiel für das Ansprechen externer Hardwarekomponenten verwendet werden kann (vgl. **wiki:GPIO**). 38

PLL Elektronische Schaltungsanordnung die die Frequenz eines veränderbaren Oszillators beeinflussen kann. Wird bei FPGAs meist zur Erzeugung von Taktsignalen mit einstellbarer Geschwindkeit verwendet (vgl. **wiki:PLL**). 38

Pmod Offener Standard für eine Mikrokontroller- und FPGA-Schnittstelle bei der 6 Pins mit einer Masse-, Stromleitung und 4 Datenleitgungen zusammengefasst werden. 12

Reverse Engineering "Reverse Engineering [...] bezeichnet den Vorgang, aus einem bestehenden fertigen System oder einem meistens industriell gefertigten Produkt durch Untersuchung der Strukturen, Zustände und Verhaltensweisen die Konstruktionselemente zu extrahieren."[39]. 11

SPI Synchroner, serieller Datenbus für Datenübertragungen nach dem Master-Slave-Prinzip mit dem vergleichsweise hohe Datendurchsätze möglich sind (vgl. [12]). 10, 38

UART Schnittstelle zur asynchronen, seriellen Datenübertragung (vgl. [10]). 9, 38

VCD Text-basiertes Dateiformat zur Speicherung von zeitlichen Signalabfolgen (vgl. [29]). 38

Verilog Wortkreuzung aus "Verification" und "Logic". Hardwarebeschreibungssparche für Programmierung und Simulation von FPGAs und CPLDs die in den USA geläufiger ist als VHDL (vgl. [40], siehe auch [41] zur Namesnherkunft). 3, 11

VHDL Vor allem in Europa verbeitetete Hardwarebeschreibungssprache für die Simulation und Programmierung von FPGAs und CPLDs (vgl. **wiki:VHDL**). 38

Abbildungsverzeichnis

Tabellenverzeichnis

Literatur

[1] *Echtzeit – Wikipedia*, [Online; accessed 1. Jun. 2018], Mai 2018. Adresse: https://de.wikipedia.org/wiki/Echtzeit.

[2] A. Müller, „Ein universales, rekonfigurierbares und freies USB-Gerät zur Timing-, Protokoll-, Logik- und Eventanalyse von digitalen Signalen“, Bachelorarbeit, Hochschule Augsburg, 2010.

[3] *Advanced triggering and buffer filling*, [Online; accessed 31. May 2018], Mai 2018. Adresse: https://forum.digilentinc.com/topic/9488-advanced-triggering-and-buffer-filling.

[4] *SUMP2 – 96 MSPS Logic Analyzer for $22*, [Online; accessed 31. May 2018], Okt. 2016. Adresse: https://blackmesalabs.wordpress.com/2016/10/24/sump2-96-msps-logic-analyzer-for-22.

[5] *Open Bench Logic Sniffer - DP*, [Online; accessed 31. May 2018], Juni 2016. Adresse: http://dangerousprototypes.com/docs/Open_Bench_Logic_Sniffer.

[6] *Openbench Logic Sniffer - sigrok*, [Online; accessed 31. May 2018], Mai 2018. Adresse: https://sigrok.org/wiki/Openbench_Logic_Sniffer.

[7] *Digitalsignal – Wikipedia*, [Online; accessed 4. Jun. 2018], Mai 2018. Adresse: https://de.wikipedia.org/wiki/Digitalsignal.

[8] *Programmierbare logische Schaltung – Wikipedia*, [Online; accessed 6. Jun. 2018], Mai 2018. Adresse: https://de.wikipedia.org/wiki/Programmierbare_logische_Schaltung.

[9] *iCE40 LP/HX Family Data Sheet*, Lattice Semiconductor Corp., 2017. Adresse: http://www.latticesemi.com/view_document?document_id=49312.

[10] *Universal Asynchronous Receiver Transmitter – Wikipedia*, [Online; accessed 31. May 2018], Mai 2018. Adresse: https://de.wikipedia.org/wiki/Universal_Asynchronous_Receiver_Transmitter.

[11] *I^2C – Wikipedia*, [Online; accessed 7. Jun. 2018], Juni 2018. Adresse: https://de.wikipedia.org/wiki/I%C2%B2C.

[12] *Serial Peripheral Interface – Wikipedia*, [Online; accessed 31. May 2018], Mai 2018. Adresse: https://de.wikipedia.org/wiki/Serial_Peripheral_Interface.

[13] C. Wolf und M. Lasser, *Project IceStorm*, [Online; accessed 6. Jun. 2018], Mai 2018. Adresse: http://www.clifford.at/icestorm.

[14] *forflo/yodl*, [Online; accessed 16. Jun. 2018], Juni 2018. Adresse: https://github.com/forflo/yodl.

[15] *Icarus Verilog*, [Online; accessed 16. Jun. 2018], Juni 2018. Adresse: http://iverilog.icarus.com.

[16] *GTKWave*, [Online; accessed 16. Jun. 2018], Juni 2018. Adresse: http://gtkwave.sourceforge.net.

[17] *YosysHQ/yosys*, [Online; accessed 16. Jun. 2018], Juni 2018. Adresse: https://github.com/YosysHQ/yosys.

[18] *cseed/arachne-pnr*, [Online; accessed 16. Jun. 2018], Juni 2018. Adresse: https://github.com/cseed/arachne-pnr.

[19] *cliffordwolf/icestorm*, [Online; accessed 16. Jun. 2018], Juni 2018. Adresse: https://github.com/cliffordwolf/icestorm.

[20] T. E. GmbH, *IceZero mit Lattice ICE40HX*, [Online; accessed 17. Jun. 2018], Juni 2018. Adresse: https://shop.trenz-electronic.de/de/TE0876-02-IceZero-mit-Lattice-ICE40HX?path=Trenz_Electronic/Modules_and_Module_Carriers/3.05x6.5/TE0876/Reference_Design.

[21] ——, *icoBoard Version 1.1 mit 8 MBit SRAM*, [Online; accessed 17. Jun. 2018], Juni 2018. Adresse: https://shop.trenz-electronic.de/de/TE0887-03M-icoBoard-Version-1.1-mit-8-MBit-SRAM?c=459.

[22] *SCH-TE0876-01*, Lattice Semiconductor Corp., 2017. Adresse: http://www.trenz-electronic.de/fileadmin/docs/Trenz_Electronic/Modules_and_Module_Carriers/3.05x6.5/TE0876/REV01/Documents/SCH-TE0876-01.PDF.

[23] *lawrie/icotools*, [Online; accessed 17. Jun. 2018], Juni 2018. Adresse: https://github.com/lawrie/icotools.

[24] *SiliconBlue ICE Technology Library*, Lattice Semiconductor Corp. Adresse: http://www.latticesemi.com/-/media/LatticeSemi/Documents/TechnicalBriefs/iCETechnologyLibrary.ashx?document_id=44572.

[25] *FIFO – Mikrocontroller.net*, [Online; accessed 18. Jun. 2018], Juni 2018. Adresse: https://www.mikrocontroller.net/articles/FIFO.

[26] *jhol/demon-core-import*, [Online; accessed 19. Jun. 2018], Juni 2018. Adresse: https://github.com/jhol/demon-core-import/blob/master/trigger.v.

[27] *torvalds/linux*, [Online; accessed 19. Jun. 2018], Juni 2018. Adresse: https://github.com/torvalds/linux/blob/master/tools/spi/spidev_test.c.

[28] *rickyrockrat/la2vcd2*, [Online; accessed 19. Jun. 2018], Juni 2018. Adresse: https://github.com/rickyrockrat/la2vcd2.

[29] *Value Change Dump – Wikipedia*, [Online; accessed 18. Jun. 2018], Juni 2018. Adresse: https://de.wikipedia.org/wiki/Value_Change_Dump.

[30] *MIDI - Wikipedia*, [Online; accessed 18. Jun. 2018], Juni 2018. Adresse: https://en.wikipedia.org/wiki/MIDI.

[31] *MIDI beat clock - Wikipedia*, [Online; accessed 18. Jun. 2018], Juni 2018. Adresse: https://en.wikipedia.org/wiki/MIDI_beat_clock.

[32] *PulseView - sigrok*, [Online; accessed 16. Jun. 2018], Mai 2018. Adresse: https://sigrok.org/wiki/PulseView.

[33] *zylin/Verilog_VCD*, [Online; accessed 18. Jun. 2018], Juni 2018. Adresse: https://github.com/zylin/Verilog_VCD.

[34] T. Falke, „Dreidimensionale Lokalisierbarkeit und Differenzierbarkeit von Hörereignissen“, Masterarbeit, Hochschule Hamburg, 2016.

[35] *Mutable Instruments | MIDIpal - User manual*, [Online; accessed 18. Jun. 2018], Juni 2018. Adresse: https://mutable-instruments.net/archive/midipal/manual.

[36] *SUMP logic analyzer Verilog Demon core documentation - DP*, [Online; accessed 19. Jun. 2018], Juni 2016. Adresse: http://dangerousprototypes.com/docs/SUMP_logic_analyzer_Verilog_Demon_core_documentation.

[37] *Complex Programmable Logic Device – Wikipedia*, [Online; accessed 26. May 2018], Mai 2018. Adresse: https://de.wikipedia.org/wiki/Complex_Programmable_Logic_Device.

[38] *Field Programmable Gate Array – Wikipedia*, [Online; accessed 26. May 2018], Mai 2018. Adresse: https://de.wikipedia.org/wiki/Field_Programmable_Gate_Array.

[39] *Reverse Engineering – Wikipedia*, [Online; accessed 16. Jun. 2018], Mai 2018. Adresse: https://de.wikipedia.org/wiki/Reverse_Engineering.

[40] *Verilog – Wikipedia*, [Online; accessed 26. May 2018], Mai 2018. Adresse: https://de.wikipedia.org/wiki/Verilog.

[41] S. Golson, *Oral History of Philip Raymond "Phil" Moorby*. Computer History Museum, 2013, S. 23–25. Adresse: http://archive.computerhistory.org/resources/access/text/2013/11/102746653-05-01-acc.pdf.

A. Material

A.1 Pmod-Pinbelegung für die Event-Aufnahme

Für die Event-Aufnahme sind die Pmod-Header P3 und P4 als Eingänge konfiguriert.

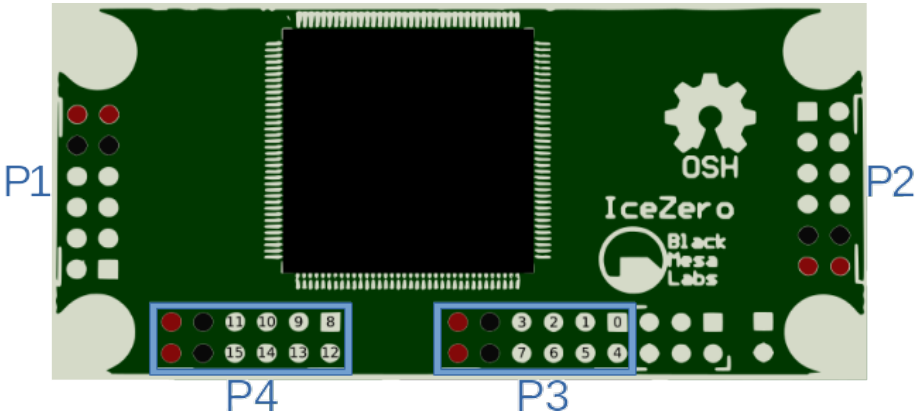


Abbildung A.1: Pinbelegung der Pmod-Header des IceZero-Boards

Bei der Pin-Belegung wurde das selbe Schema wie beim GPIO-Modul des IcoSoc verwendet. Die Numerierung läuft dementsprechend von rechts nach links und für die einzelnen Pmod-Header "zeilenweise" von oben nach unten.
Für die Zuordnung der Signalnamen (und der zugehörigen VCD-Symbole) bei der Ausgabe im CSV- oder VCD-Format kann folgende Tabelle verwendet werden:

		,	+	*	)
3.3V	GND	pin_11	pin_10	pin_9	pin_8
3.3V	GND	pin_15	pin_14	pin_13	pin_12
		0	/	.	-

Pmod - P4

		$	#	”	!
3.3V	GND	pin_3	pin_2	pin_1	pin_0
3.3V	GND	pin_7	pin_6	pin_5	pin_4
		(	’	&	%

Pmod - P3

Tabelle A.1: Pinbelegung der Pmod-Header des Icezero-Boards

A.2 Pinverbindungen Raspberry Pi und IceZero FPGA-Shield

Bei den mit einem “X” markierten Pins besteht Hardware-seitig keine Verbindung zwischen dem Raspberry Pi und dem IceZero-Board.

Funktion	iCE40	IceZero	WiringPi	Name	Physical		Name	WiringPi	IceZero	iCE40	Funktion
				3.3V	1	2	5V				
unused	115	PI_I2C_SDA	8	SDA.1	3	4	5V				
unused	114	PI_I2C_SCL	9	SCL.1	5	6	GND				
		X	7	1-Wire	7	8	TxD	15	PI_UART_WI	113	DEBUG_UART
				GND	9	10	RxD	16	PI_UART_RO	112	DEBUG_UART
		X	0	GPIO.0	11	12	GPIO.1	1	X		
		X	2	GPIO.2	13	14	GND				
unused	101	PI_GPIO_2	3	GPIO.3	15	16	GPIO.4	4	X		
				3.3V	17	18	GPIO.5	5	PI_GPIO_1	99	unused
DATA_MOSI	90	PI_SPI_MOSI	12	MOSI	19	20	GND				
DATA_MISO	87	PI_SPI_MISO	13	MISO	21	22	GPIO.6	6	PI_GPIO_0	88	unused
DATA_SCK	79	PI_SPI_SCK	14	SCLK	23	24	CE0	10	PI_SPI_CE_0	85	DATA_SS
				GND	25	26	CE1	11	PI_SPI_CE_1	78	unused
unused	73	PI_ID_0	30	SDA.0	27	28	SCL.0	31	PI_ID_1	74	unused
CONFIG_DONE	65	CFG_DONE	21	GPIO.21	29	30	GND				
CONFIG_MOSI	68	CFG_SI	22	GPIO.22	31	32	GPIO.26	26	CFG_SS	71	CONFIG_SS
CONFIG_MISO	67	CFG_SO	23	GPIO.23	33	34	GND				
		X	24	GPIO.24	35	36	GPIO.27	27	CFG_SCK	70	CONFIG_SCK
CONFIG_RESET	66	CFG_RST_1	25	GPIO.25	37	38	GPIO.28	28	X		
				GND	39	40	GPIO.29	29	X		

Tabelle A.2: Pinverbindungen Raspberry Pi und IceZero FPGA-Shield

A.3 Detaillierte Pinbelegung aller Pmod-Header

Tabelle A.3: Pinbelegung Pmod-P1

130	135	137	139	iCE40 PCF Pin
GPIO_PIN_7	GPIO_PIN_5	GPIO_PIN_3	GPIO_PIN_1	IceZero Name
pmod1_4	pmod1_3	pmod1_2	pmod1_1	IcoSoc Name
pmod1_10	pmod1_9	pmod1_8	pmod1_7	IcoSoc Name
GPIO_PIN_6	GPIO_PIN_4	GPIO_PIN_2	GPIO_PIN_0	IceZero Name
134	136	138	141	iCE40 PCF Pin
Pmod P1				

Tabelle A.4: Pinbelegung Pmod-P2

43	45	48	56	iCE40 PCF Pin
GPIO_PIN_15	GPIO_PIN_13	GPIO_PIN_11	GPIO_PIN_9	IceZero Name
pmod2_4	pmod2_3	pmod2_2	pmod2_1	IcoSoc Name
pmod2_10	pmod2_9	pmod2_8	pmod2_7	IcoSoc Name
GPIO_PIN_14	GPIO_PIN_12	GPIO_PIN_10	GPIO_PIN_8	IceZero Name
42	44	47	55	iCE40 PCF Pin
Pmod P2				

Tabelle A.5: Pinbelegung Pmod-P3

52	28	29	26	iCE40 PCF Pin
GPIO_PIN_31	GPIO_PIN_30	GPIO_PIN_29	GPIO_PIN_28	IceZero Name
pmod3_4	pmod3_3	pmod3_2	pmod3_1	IcoSoc Name
pmod3_10	pmod3_9	pmod3_8	pmod3_7	IcoSoc Name
GPIO_PIN_19	GPIO_PIN_18	GPIO_PIN_17	GPIO_PIN_16	IceZero Name
37	38	39	41	iCE40 PCF Pin
Pmod P3				

Tabelle A.6: Pinbelegung Pmod-P4

7	8	20	21	iCE40 PCF Pin
GPIO_PIN_27	GPIO_PIN_26	GPIO_PIN_25	GPIO_PIN_24	IceZero Name
pmod4_4	pmod4_3	pmod4_2	pmod4_1	IcoSoc Name
pmod4_10	pmod4_9	pmod4_8	pmod4_7	IcoSoc Name
GPIO_PIN_23	GPIO_PIN_22	GPIO_PIN_21	GPIO_PIN_20	IceZero Name
142	143	144	1	iCE40 PCF Pin
Pmod P4				

A.4 Inhalt der CD

Die beiliegende CD enthält den Inhalt des Github-Repositories

`https://github.com/dm7h/fpga-event-recorder`

zum Zeitpunkt der Abgabe.

Die folgende Tabelle bietet einen Überblick über die Inhalte des Repositories:

Verz.	Unterverzeichnis	Inhaltsbeschreibung
thesis/		Finale Version der Bachelorarbeit als PDF
thesis/	src/	Latex-Sourcen der Bachelorarbeit
doc/	ref/	iCE40 Manuals und relevante Hardwaredokumentation
misc/		Nicht direkt Event-Recorder bezogene Informationen und Code (u.a. für die Analyse)
src/		
	Logikanalysator/	Dateien des Semesterprojekts "Logikanalysator mit AVR Mega32U4 und Altera MAX CPLD" inkl. Dateien der Bachelorarbeit von Andreas Müller (USB-TPLE)
	icotools/	Portierung des icotools Projekts für das IceZero-Board. Original-Repository: https://github.com/cliffordwolf/icotools. Relevant sind hier vor allem die Unterverzeichnisse "icosoc" und "icozprog"
	icozctl/	Fork des icoprog-Tools (icotools/icoporg) mit zusätzlichen Funktionen zur Steuerung des Event-Recorders
	picosoc/	Portierung des PicoSoc-Projekts auf das IceZero-Board (nicht direkt projekt-relevant). PicoSoc ist eine minimale Variante des IcoSocs. Original-Repository: https://github.com/cliffordwolf/picorv32/tree/master/picosoc
	sump2_ice40/	SUMP2 Variante für das IceZero-Board. Quelle: https://blackmesalabs.wordpress.com/2017/02/07/icezero-fpga-board-for-rasppi/
	sump2_pipistrello_ftdi_fifo/	SUMP2 Vairante für das Pipistrello-Board bei dem der UART durch einen FTDI-Fifo ersetzt wurde, um einen Höheren Datenduchsatz zu ermöglichen. Quelle: http://forum.gadgetfactory.net/topic/1748-open-bench-logic-sniffer-with-64mb-capture-buffer/
	demon-core-import/	SUMP2 Weiterenwicklung für den Open Bench Logic Sniffer. Quelle: https://github.com/jhol/demon-core-import

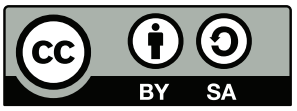
Tabelle A.7: Überblick über den Inhalt des Git-Repositories

B. Lizenz

Alle im Rahmen dieser Arbeit erstellten Text-Dokumente stehen unter der folgenden Creative Commons BY-SA 3.0 Lizenz.

B.1 Creative Commons Attribution-ShareAlike 3.0 International

Dieses Werk ist lizenziert unter einer Creative Commons „Namensnennung – Weitergabe unter gleichen Bedingungen 3.0 Deutschland“ Lizenz.



B.2 Creative Commons Legal Code

B.2.1 Attribution-ShareAlike 3.0 Unported

> CREATIVE COMMONS CORPORATION IS NOT A LAW FIRM AND DOES NOT PROVIDE LEGAL SERVICES. DISTRIBUTION OF THIS LICENSE DOES NOT CREATE AN ATTORNEY-CLIENT RELATIONSHIP. CREATIVE COMMONS PROVIDES THIS INFORMATION ON AN “AS-IS” BASIS. CREATIVE COMMONS MAKES NO WARRANTIES REGARDING THE INFORMATION PROVIDED, AND DISCLAIMS LIABILITY FOR DAMAGES RESULTING FROM ITS USE.

B.2.1.1 *License*

THE WORK (AS DEFINED BELOW) IS PROVIDED UNDER THE TERMS OF THIS CREATIVE COMMONS PUBLIC LICENSE (“CCPL” OR “LICENSE”). THE WORK IS PROTECTED BY COPYRIGHT AND/OR OTHER APPLICABLE LAW. ANY USE OF THE WORK OTHER THAN AS AUTHORIZED UNDER THIS LICENSE OR COPYRIGHT LAW IS PROHIBITED.

BY EXERCISING ANY RIGHTS TO THE WORK PROVIDED HERE, YOU ACCEPT AND AGREE TO BE BOUND BY THE TERMS OF THIS LICENSE. TO THE EXTENT THIS LICENSE MAY BE CONSIDERED TO BE A CONTRACT, THE LICENSOR GRANTS YOU THE RIGHTS CONTAINED HERE IN CONSIDERATION OF YOUR ACCEPTANCE OF SUCH TERMS AND CONDITIONS.

1. Definitions

(a) **"Adaptation"** means a work based upon the Work, or upon the Work and other pre-existing works, such as a translation, adaptation, derivative work, arrangement of music or other alterations of a literary or artistic work, or phonogram or performance and includes cinematographic adaptations or any other form in which the Work may be recast, transformed, or adapted including in any form recognizably derived from the original, except that a work that constitutes a Collection will not be considered an Adaptation for the purpose of this License. For the avoidance of doubt, where the Work is a musical work, performance or phonogram, the synchronization of the Work in timed-relation with a moving image ("synching") will be considered an Adaptation for the purpose of this License.

(b) **"Collection"** means a collection of literary or artistic works, such as encyclopedias and anthologies, or performances, phonograms or broadcasts, or other works or subject matter other than works listed in Section 1(f) below, which, by reason of the selection and arrangement of their contents, constitute intellectual creations, in which the Work is included in its entirety in unmodified form along with one or more other contributions, each constituting separate and independent works in themselves, which together are assembled into a collective whole. A work that constitutes a Collection will not be considered an Adaptation (as defined below) for the purposes of this License.

(c) **"Creative Commons Compatible License"** means a license that is listed at https://creativecommons.org/compatiblelicenses that has been approved by Creative Commons as being essentially equivalent to this License, including, at a minimum, because that license: (i) contains terms that have the same purpose, meaning and effect as the License Elements of this License; and, (ii) explicitly permits the relicensing of adaptations of works made available under that license under this License or a Creative Commons jurisdiction license with the same License Elements as this License.

(d) **"Distribute"** means to make available to the public the original and copies of the Work or Adaptation, as appropriate, through sale or other transfer of ownership.

(e) **"License Elements"** means the following high-level license attributes as selected by Licensor and indicated in the title of this License: Attribution, ShareAlike.

(f) **"Licensor"** means the individual, individuals, entity or entities that offer(s) the Work under the terms of this License.

(g) **"Original Author"** means, in the case of a literary or artistic work, the individual, individuals, entity or entities who created the Work or if no individual or entity can be identified, the publisher; and in addition (i) in the case of a performance the actors, singers, musicians, dancers, and other persons who act, sing, deliver, declaim, play in, interpret or otherwise perform literary or artistic works or expressions of folklore; (ii) in the case of a phonogram the producer being the person or legal entity who first fixes the sounds of a performance or other sounds; and, (iii) in the case of broadcasts, the organization that transmits the broadcast.

(h) **"Work"** means the literary and/or artistic work offered under the terms of this License including without limitation any production in the literary, scientific and artistic domain, whatever may be the mode or form of its expression including digital form, such as a book, pamphlet and other writing; a lecture, address, sermon or other work of the same nature; a dramatic or dramatico-musical work; a choreographic work or entertainment in dumb show; a musical composition with or without words; a cinematographic work to which are assimilated works expressed by a process analogous

to cinematography; a work of drawing, painting, architecture, sculpture, engraving or lithography; a photographic work to which are assimilated works expressed by a process analogous to photography; a work of applied art; an illustration, map, plan, sketch or three-dimensional work relative to geography, topography, architecture or science; a performance; a broadcast; a phonogram; a compilation of data to the extent it is protected as a copyrightable work; or a work performed by a variety or circus performer to the extent it is not otherwise considered a literary or artistic work.

(i) **"You"** means an individual or entity exercising rights under this License who has not previously violated the terms of this License with respect to the Work, or who has received express permission from the Licensor to exercise rights under this License despite a previous violation.

(j) **"Publicly Perform"** means to perform public recitations of the Work and to communicate to the public those public recitations, by any means or process, including by wire or wireless means or public digital performances; to make available to the public Works in such a way that members of the public may access these Works from a place and at a place individually chosen by them; to perform the Work to the public by any means or process and the communication to the public of the performances of the Work, including by public digital performance; to broadcast and rebroadcast the Work by any means including signs, sounds or images.

(k) **"Reproduce"** means to make copies of the Work by any means including without limitation by sound or visual recordings and the right of fixation and reproducing fixations of the Work, including storage of a protected performance or phonogram in digital form or other electronic medium.

2. Fair Dealing Rights. Nothing in this License is intended to reduce, limit, or restrict any uses free from copyright or rights arising from limitations or exceptions that are provided for in connection with the copyright protection under copyright law or other applicable laws.

3. License Grant. Subject to the terms and conditions of this License, Licensor hereby grants You a worldwide, royalty-free, non-exclusive, perpetual (for the duration of the applicable copyright) license to exercise the rights in the Work as stated below:

(a) to Reproduce the Work, to incorporate the Work into one or more Collections, and to Reproduce the Work as incorporated in the Collections;

(b) to create and Reproduce Adaptations provided that any such Adaptation, including any translation in any medium, takes reasonable steps to clearly label, demarcate or otherwise identify that changes were made to the original Work. For example, a translation could be marked "The original work was translated from English to Spanish," or a modification could indicate "The original work has been modified.";

(c) to Distribute and Publicly Perform the Work including as incorporated in Collections; and,

(d) to Distribute and Publicly Perform Adaptations.

(e) For the avoidance of doubt:

i. **Non-waivable Compulsory License Schemes**. In those jurisdictions in which the right to collect royalties through any statutory or compulsory licensing scheme cannot be waived, the Licensor reserves the exclusive right to

collect such royalties for any exercise by You of the rights granted under this License;

ii. **Waivable Compulsory License Schemes**. In those jurisdictions in which the right to collect royalties through any statutory or compulsory licensing scheme can be waived, the Licensor waives the exclusive right to collect such royalties for any exercise by You of the rights granted under this License; and,

iii. **Voluntary License Schemes**. The Licensor waives the right to collect royalties, whether individually or, in the event that the Licensor is a member of a collecting society that administers voluntary licensing schemes, via that society, from any exercise by You of the rights granted under this License.

The above rights may be exercised in all media and formats whether now known or hereafter devised. The above rights include the right to make such modifications as are technically necessary to exercise the rights in other media and formats. Subject to Section 8(f), all rights not expressly granted by Licensor are hereby reserved.

4. Restrictions. The license granted in Section 3 above is expressly made subject to and limited by the following restrictions:

(a) You may Distribute or Publicly Perform the Work only under the terms of this License. You must include a copy of, or the Uniform Resource Identifier (URI) for, this License with every copy of the Work You Distribute or Publicly Perform. You may not offer or impose any terms on the Work that restrict the terms of this License or the ability of the recipient of the Work to exercise the rights granted to that recipient under the terms of the License. You may not sublicense the Work. You must keep intact all notices that refer to this License and to the disclaimer of warranties with every copy of the Work You Distribute or Publicly Perform. When You Distribute or Publicly Perform the Work, You may not impose any effective technological measures on the Work that restrict the ability of a recipient of the Work from You to exercise the rights granted to that recipient under the terms of the License. This Section 4(a) applies to the Work as incorporated in a Collection, but this does not require the Collection apart from the Work itself to be made subject to the terms of this License. If You create a Collection, upon notice from any Licensor You must, to the extent practicable, remove from the Collection any credit as required by Section 4(c), as requested. If You create an Adaptation, upon notice from any Licensor You must, to the extent practicable, remove from the Adaptation any credit as required by Section 4(c), as requested.

(b) You may Distribute or Publicly Perform an Adaptation only under the terms of: (i) this License; (ii) a later version of this License with the same License Elements as this License; (iii) a Creative Commons jurisdiction license (either this or a later license version) that contains the same License Elements as this License (e.g., Attribution-ShareAlike 3.0 US)); (iv) a Creative Commons Compatible License. If you license the Adaptation under one of the licenses mentioned in (iv), you must comply with the terms of that license. If you license the Adaptation under the terms of any of the licenses mentioned in (i), (ii) or (iii) (the "Applicable License"), you must comply with the terms of the Applicable License generally and the following provisions: (I) You must include a copy of, or the URI for, the Applicable License with every copy of each Adaptation You Distribute or Publicly Perform; (II) You may not offer or impose any terms on the Adaptation that restrict the terms of the Applicable License or the ability of the recipient of the Adaptation to exercise the rights granted to that recipient under the terms of the Applicable License; (III) You must keep intact all

notices that refer to the Applicable License and to the disclaimer of warranties with every copy of the Work as included in the Adaptation You Distribute or Publicly Perform; (IV) when You Distribute or Publicly Perform the Adaptation, You may not impose any effective technological measures on the Adaptation that restrict the ability of a recipient of the Adaptation from You to exercise the rights granted to that recipient under the terms of the Applicable License. This Section 4(b) applies to the Adaptation as incorporated in a Collection, but this does not require the Collection apart from the Adaptation itself to be made subject to the terms of the Applicable License.

(c) If You Distribute, or Publicly Perform the Work or any Adaptations or Collections, You must, unless a request has been made pursuant to Section 4(a), keep intact all copyright notices for the Work and provide, reasonable to the medium or means You are utilizing: (i) the name of the Original Author (or pseudonym, if applicable) if supplied, and/or if the Original Author and/or Licensor designate another party or parties (e.g., a sponsor institute, publishing entity, journal) for attribution ("Attribution Parties") in Licensor's copyright notice, terms of service or by other reasonable means, the name of such party or parties; (ii) the title of the Work if supplied; (iii) to the extent reasonably practicable, the URI, if any, that Licensor specifies to be associated with the Work, unless such URI does not refer to the copyright notice or licensing information for the Work; and (iv) , consistent with Ssection 3(b), in the case of an Adaptation, a credit identifying the use of the Work in the Adaptation (e.g., "French translation of the Work by Original Author," or "Screenplay based on original Work by Original Author"). The credit required by this Section 4(c) may be implemented in any reasonable manner; provided, however, that in the case of a Adaptation or Collection, at a minimum such credit will appear, if a credit for all contributing authors of the Adaptation or Collection appears, then as part of these credits and in a manner at least as prominent as the credits for the other contributing authors. For the avoidance of doubt, You may only use the credit required by this Section for the purpose of attribution in the manner set out above and, by exercising Your rights under this License, You may not implicitly or explicitly assert or imply any connection with, sponsorship or endorsement by the Original Author, Licensor and/or Attribution Parties, as appropriate, of You or Your use of the Work, without the separate, express prior written permission of the Original Author, Licensor and/or Attribution Parties.

(d) Except as otherwise agreed in writing by the Licensor or as may be otherwise permitted by applicable law, if You Reproduce, Distribute or Publicly Perform the Work either by itself or as part of any Adaptations or Collections, You must not distort, mutilate, modify or take other derogatory action in relation to the Work which would be prejudicial to the Original Author's honor or reputation. Licensor agrees that in those jurisdictions (e.g. Japan), in which any exercise of the right granted in Section 3(b) of this License (the right to make Adaptations) would be deemed to be a distortion, mutilation, modification or other derogatory action prejudicial to the Original Author's honor and reputation, the Licensor will waive or not assert, as appropriate, this Section, to the fullest extent permitted by the applicable national law, to enable You to reasonably exercise Your right under Section 3(b) of this License (right to make Adaptations) but not otherwise.

5. Representations, Warranties and Disclaimer

UNLESS OTHERWISE MUTUALLY AGREED TO BY THE PARTIES IN WRITING, LICENSOR OFFERS THE WORK AS-IS AND MAKES NO REPRESENTATIONS OR

WARRANTIES OF ANY KIND CONCERNING THE WORK, EXPRESS, IMPLIED, STATUTORY OR OTHERWISE, INCLUDING, WITHOUT LIMITATION, WARRANTIES OF TITLE, MERCHANTIBILITY, FITNESS FOR A PARTICULAR PURPOSE, NONINFRINGEMENT, OR THE ABSENCE OF LATENT OR OTHER DEFECTS, ACCURACY, OR THE PRESENCE OF ABSENCE OF ERRORS, WHETHER OR NOT DISCOVERABLE. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO SUCH EXCLUSION MAY NOT APPLY TO YOU.

6. Limitation on Liability. EXCEPT TO THE EXTENT REQUIRED BY APPLICABLE LAW, IN NO EVENT WILL LICENSOR BE LIABLE TO YOU ON ANY LEGAL THEORY FOR ANY SPECIAL, INCIDENTAL, CONSEQUENTIAL, PUNITIVE OR EXEMPLARY DAMAGES ARISING OUT OF THIS LICENSE OR THE USE OF THE WORK, EVEN IF LICENSOR HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

7. Termination

(a) This License and the rights granted hereunder will terminate automatically upon any breach by You of the terms of this License. Individuals or entities who have received Adaptations or Collections from You under this License, however, will not have their licenses terminated provided such individuals or entities remain in full compliance with those licenses. Sections 1, 2, 5, 6, 7, and 8 will survive any termination of this License.

(b) Subject to the above terms and conditions, the license granted here is perpetual (for the duration of the applicable copyright in the Work). Notwithstanding the above, Licensor reserves the right to release the Work under different license terms or to stop distributing the Work at any time; provided, however that any such election will not serve to withdraw this License (or any other license that has been, or is required to be, granted under the terms of this License), and this License will continue in full force and effect unless terminated as stated above.

8. Miscellaneous

(a) Each time You Distribute or Publicly Perform the Work or a Collection, the Licensor offers to the recipient a license to the Work on the same terms and conditions as the license granted to You under this License.

(b) Each time You Distribute or Publicly Perform an Adaptation, Licensor offers to the recipient a license to the original Work on the same terms and conditions as the license granted to You under this License.

(c) If any provision of this License is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this License, and without further action by the parties to this agreement, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.

(d) No term or provision of this License shall be deemed waived and no breach consented to unless such waiver or consent shall be in writing and signed by the party to be charged with such waiver or consent.

(e) This License constitutes the entire agreement between the parties with respect to the Work licensed here. There are no understandings, agreements or representations with

respect to the Work not specified here. Licensor shall not be bound by any additional provisions that may appear in any communication from You. This License may not be modified without the mutual written agreement of the Licensor and You.

(f) The rights granted under, and the subject matter referenced, in this License were drafted utilizing the terminology of the Berne Convention for the Protection of Literary and Artistic Works (as amended on September 28, 1979), the Rome Convention of 1961, the WIPO Copyright Treaty of 1996, the WIPO Performances and Phonograms Treaty of 1996 and the Universal Copyright Convention (as revised on July 24, 1971). These rights and subject matter take effect in the relevant jurisdiction in which the License terms are sought to be enforced according to the corresponding provisions of the implementation of those treaty provisions in the applicable national law. If the standard suite of rights granted under applicable copyright law includes additional rights not granted under this License, such additional rights are deemed to be included in the License; this License is not intended to restrict the license of any rights under applicable law.

B.2.1.2 Creative Commons Notice

Creative Commons is not a party to this License, and makes no warranty whatsoever in connection with the Work. Creative Commons will not be liable to You or any party on any legal theory for any damages whatsoever, including without limitation any general, special, incidental or consequential damages arising in connection to this license. Notwithstanding the foregoing two (2) sentences, if Creative Commons has expressly identified itself as the Licensor hereunder, it shall have all rights and obligations of Licensor.

Except for the limited purpose of indicating to the public that the Work is licensed under the CCPL, Creative Commons does not authorize the use by either party of the trademark "Creative Commons" or any related trademark or logo of Creative Commons without the prior written consent of Creative Commons. Any permitted use will be in compliance with Creative Commons' then-current trademark usage guidelines, as may be published on its website or otherwise made available upon request from time to time. For the avoidance of doubt, this trademark restriction does not form part of the License.

Creative Commons may be contacted at https://creativecommons.org/.

B.3 ISC-Lizenz

Alle im Rahmen dieser Arbeit erstellten oder modifizierten Quelltext-Dokumente stehen soweit nicht anders vermerkt unter der nachfolgenden ISC-Lizenz.

Erklärung zum Abschlussbericht

Hiermit versichere ich, die eingereichte Abschlussarbeit selbständig verfasst und keine andere als die von mir angegebenen Quellen und Hilfsmittel benutzt zu haben. Wörtlich oder inhaltlich verwendete Quellen wurden entsprechend den anerkannten Regeln wissenschaftlichen Arbeitens zitiert. Ich erkläre weiterhin, dass die vorliegende Arbeit noch nicht anderweitig als Abschlussarbeit eingereicht wurde. Das Merkblatt zum Täuschungsverbot im Prüfungsverfahren der Hochschule Augsburg habe ich gelesen und zur Kenntnis genommen. Ich versichere, dass die von mir abgegebene Arbeit keinerlei Plagiate, Texte oder Bilder umfasst, die durch von mir beauftragte Dritte erstellt wurden.

Ort, Datum Unterschrift