Diplomarbeit

Jan-Heiner Dreschhoff

FPGA-Prototyp der Signalverarbeitung für ABS-Sensoren mit Diagnosefunktion

Fakultät Technik und Informatik
Department Informations- und Elektrotechnik

Faculty of Engineering and Computer Science
Department of Information and Electrical Engineering

Jan-Heiner Dreschhoff

FPGA-Prototyp der Signalverarbeitung für ABS-Sensoren mit Diagnosefunktion

Diplomarbeit eingereicht im Rahmen der Diplomprüfung
im Studiengang Informations- und Elektrotechnik
Studienrichtung Informationstechnik
am Department Informations- und Elektrotechnik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr.-Ing. Karl-Ragmar Riemschneider
Zweitgutachter: Prof. Dr. Franz Schubert

Abgegeben am 28.07.2010

Jan-Heiner Dreschhoff

Thema der Diplomarbeit

FPGA-Prototyp der Signalverarbeitung für ABS-Sensoren mit Diagnosefunktion

Stichworte

Automotive, Signalverarbeitung, ABS-Sensor, AMR-Effekt, DFT, THD, Harmonische, FPGA

Kurzzusammenfassung

In dieser Arbeit geht es um die Zustandserkennung von ABS Sensoren. Diese Zustandserkennung wird durch Analyse der Harmonischen des Sensorsignals durchgeführt. Maßgröße für die Signalqualität ist ein Quotient aus Störanteil zu Gesamtsignal, der sogenannte THD (Total Harmonic Distortion). Abgeschätzt werden die dafür benötigten Frequenzanteile durch die Berechnung der diskreten Fourier-Transformation. Die Ergebnisse werden zur Auswertung über RS232 an einen Messrechner gesendet.

Jan-Heiner Dreschhoff

Title of the paper

FPGA Prototype of the Signalprocessing of a Antilock Breaking Sensor with Diagnosics

Keywords

automotive, signal processing, anti-lock-breaking sensor, AMR-effect, DFT, THD, harmonic, FPGA

Abstract

This paper is about state detection for Anti-Lock Breaking sensors. For this, the harmonics in the signal of the sensor are being analyzed. The Benchmark for the signal's quality is a ratio of the interference over the full signal. This benchmark is called THD (Total Harmonic Distortion). The needed frequencies are estimated using the discrete fourier-transformation. The results are sent to a PC for analysis via RS232

Inhaltsverzeichnis

1 Einführung

Diese Arbeit beschäftigt sich mit der Implementierung eines Algorithmusses zur Abschätzung der gesamtharmonischen Störung (THD) eines Signals in VHDL. Sie ist Teil des Projektes ESZ-ABS (Experimentelle Signalverarbeitung und Zustandserkennung) der HAW-Hamburg. Das Projekt analysiert die Ausgangssignale eines ABS Sensors.

ABS Sensoren werden in jedem modernen Automobil verwendet.
Vier Sensoren messen die Drehzahl der Räder, und übertragen mindestens ein Rechtecksignal an die Bremssteuerung. Die Periodendauer dieser Signale erlaubt der Bremssteuerung Annahmen über die Drehzahl eines jeden Rades. Je nach Ausführung der Steuerung können Bremsen bei Blockieren des Rades gelöst werden (ABS), es werden einzelne Räder bei Traktionsverlust gebremst oder beides kann zur Fahrdynamikregelung kombiniert werden. Die Sensoren interagieren mit einem an der Radnabe angebrachten Encoderrad, welches entweder aus einem ferromagnetischen Zahnrad (siehe Bild 1.2) oder einem Rad mit magnetisierter Oberfläche besteht (siehe Bild 1.1).
Der Sensor befindet sich in einer Einbaulage mit einem Abstand $d < 3,5mm$ von dem Encoderrad. Wie auf den Bildern zu erkennen, bedeutet dies, dass der Sensor sich in unmittelbarer Nähe zur Bremsscheibe befindet. Diese kann sehr heiß werden, so dass ABS Sensoren auch bei Temperaturen bis $150°C$ und im Winter bei bis zu $-40°C$ zuverlässig arbeiten müssen [22]. Die großen Temperaturunterschiede sind allerdings nicht die einzig hohe Anforderung an das System. Der Sensor ist mechanischem Stress ausgesetzt, muss eine hohe EMV haben und einen Überspannungsschutz. Zusätzlich kommen immer höhere Höchstgeschwindigkeiten. Deshalb gilt die Spezifikation bis zu einer Encoder-Zahnfrequenz von $2,5kHz$, was laut Daimler AG [4] für Geschwindigkeiten von bis zu $375km/h$ reichen soll.
Diese Angabe ist aber Typabhängig, da schon die im Projekt zur Verfügung stehenden Encoderräder (z.B. VW Golf IV mit 43 Zähnen, BMW 330i Touring mit 48) sehr unterschiedlich sind. Sollten vom Verbraucher verstärkt Systeme wie etwa die Elektromechanische Bremse nachgefragt werden, könnte die Zahl der Zähne sogar steigen.

[9] [12] [19] [20]

Bild 1.1: *Aktives Encoderrad des VW Golf V, an der Stirnseite magnetisiert [9]*

Bild 1.2: *Passives Encoderrad des BMW 330i touring, radialer ferromagnetischer Zahnkranz [9]*

1.1 Das Projekt ESZ-ABS

Moderne ABS-Sensoren nutzen den anisotropen Magnetoresistiven (AMR) Effekt, um Beschleunigungen der Räder zu erkennen. Anhand dieser Beschleunigungen kann die Bremsensteuerung das Blockieren eines Rades erkennen und Bremsen wieder lösen, um den Reibungskoeffizienten zwischen Rad und Untergrund zu erhöhen. Dazu ist an jedem Rad ein Sensor und ein Encoderrad angebracht, welches ein Magnetfeld verwirbelt und so ein Signal am AMR-Sensor erzeugt. Der Aufbau des Sensors ist vereinfacht dargestellt auf Bild 1.3.

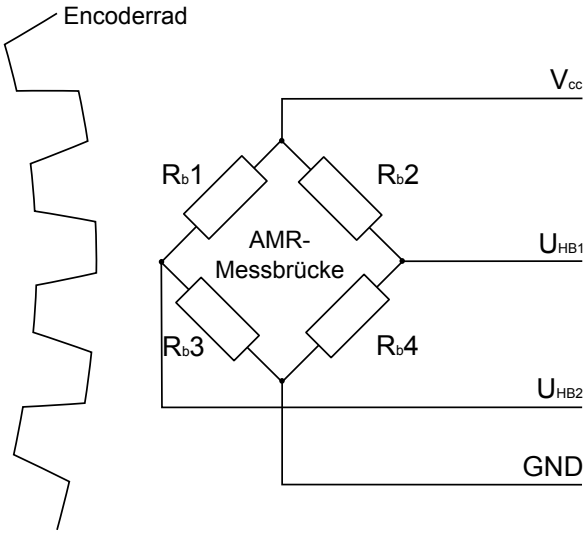


Bild 1.3: *AMR Sensorkopf vor passiven Encoderrad*

Aus den Signalen der Halbbrücken (U_{HB1} und U_{HB1}) wird ein Differenzsignal U_{diff} gebildet und analysiert. Dieses Signal ist bei optimaler Einbaulage sinusförmig. Abweichungen von dieser optimalen Einbaulage führen auf Grund der nichtlinearen Übertragungsfunktion der AMR-Messbrücke zu Verzerrungen (siehe 4.2). Ob diese Verzerrungen Einfluss auf die Interpretierbarkeit des Signals haben, soll von der Projektgruppe ESZ-ABS genauer untersucht werden.
Man erkennt auf Bild 1.3 die schematische Darstellung eines Zahnrades zur Magnetfeldablenkung. Es gibt dieses Rad in passiver (Bild 1.2) und in aktiver (Bild 1.1) Bauweise; beide stehen im Projekt zur Verfügung. Beide nutzen den AMR Effekt, sie unterscheiden sich aber in der Quelle des Magnetfeldes. Man unterscheidet:

- Passives Encoderrad
  Das Encoderrad sieht aus, wie in Bild 1.3; es besteht aus Zähnen und Lücken. Der Sensor hat einen starken Festmagneten, dessen Magnetfeld abgelenkt wird. Bild 1.5 zeigt eine Skizze des Sensors KMI22 von *NXP Semiconductors* mit Festmagnet. Auf Bild 1.4 ist rechts ein Sensor mit Magnetfeldlinien über einem Passiven Encoderrad abgebildet.
- Aktives Encoderrad
  Das Encoderrad ist magnetisiert worden, es wechseln sich Nord- und Südpol auf ebener Fläche ab. Dieser Polwechsel wird vom Sensor aufgenommen. Auf Bild 1.4 ist links ein Sensor mit Magnetfeldlinien über einem aktiven Encoderrad abgebildet.

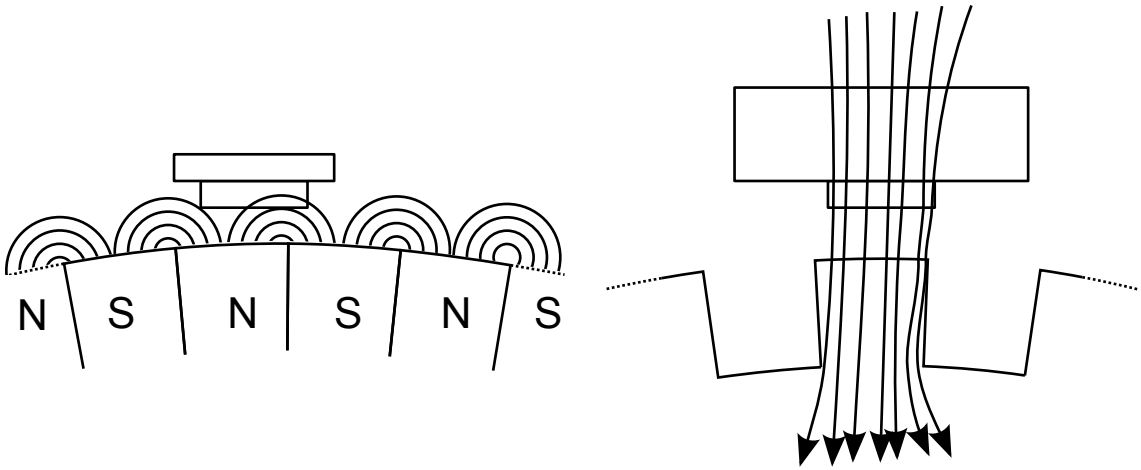


Bild 1.4: *Sensoren über aktivem (links) und passiven (rechts) Encoderrad*

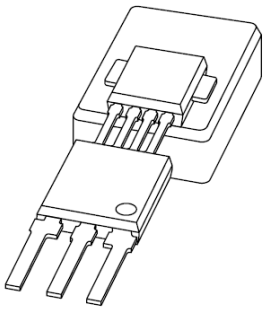


Bild 1.5: *Skizze des KMI22 von NXP mit Festmagnet*

1.1.1 Stand des Projekts

Momentan kann noch keine qualitative Aussage bezüglich der Signifikanz des Analyseansatzes gemacht werden, da die Messmittel unzureichend genau sind, und ihre beschränkten Freiheitsgrade eher Stichproben messen als umfangreiche Messreihen. Aber die Messungen, die möglich sind, untermauern die obige Annahme.

Für Messungen stehen momentan vier Aufbauten zur Verfügung (drei Radmessplätze und ein Kreuzspulenmessplatz) sowie ein Nachbau des Sensors zum Testen von Algorithmen.
Im Kreuzspulenmessplatz wird ein Sensor in ein Magnetfeld getaucht. Dieses Magnetfeld wird dann in Stärke und Richtung verändert. So wird die Kennlinie des Sensors genauer Bestimmt, und sein Übertragungsverhalten analysiert.

Der Radmessplatz, welcher als erstes in Betrieb genommen wurde (RMP1), konnte die Verzerrung durch Verfahren auf einer linearen Achse demonstrieren. Ein

Schrittmotor und dessen Matlabansteuerung ermöglichten das automatische Aufzeichnen von Messwerten.
Es wurde ein zweiter Radmessplatz entwickelt (RMP2). Das dort angebrachte Encoderrad kann auf Höhe des Sensors auf zwei Achsen linear verfahren, sowie in der Ebene, die diese Achsen aufspannen, geschwenkt werden. Durch Aktoren und ein Matlab Steuerskript wurde auch dieser Messplatz automatisiert. Durch die höheren Freiheitsgrade konnten mehr Daten gesammelt werden. So fand man z.B. heraus, dass ein verkippter Sensor vor dem Encoderrad zu einer (gemessenen) Verdoppelung der Encoderzahnfrequnz führen kann.
Der dritte Radmessplatz hat im Gegensatz zu RMP1 und RMP2 den Anspruch sehr genaue und umfassende Messreihen zu ermöglichen. Bei ihm werden keine Referenzmuster als Encoderrad verwendet, sondern an Radnaben montierte Orginalteile (Bild 1.2 und 1.1). Die Motoren, von denen die Encoderräder angetrieben werden, sind so dimensioniert, dass der vollständige spezifizierte Bereich [4] angefahren werden kann. Der Sensorkopf (welcher die AMR Messbrücke beinhaltet) ist, dank dreier linearer Verfahreinheiten, zweier Goniometer und eines Rotationselementes um sechs Achsen verfahrbar. Es kann jede erdenkliche Position, auf $10\mu m$ genau, vor dem Encoderrad eingenommen werden.

Um die Sensorfunktionen umzusetzen, wurde auf Basis des MSP430F1611 und des MSP430F1232, beide von *Texas Instruments*, eine Plattform entwickelt. Diese setzt alle Grundfunktionen des Sensors um und erweitert sie um eine neue Diagnosefunktion. Über eine RS232 Schnittstelle können die verarbeiteten Daten an einen PC gesendet werden, sie werden aber auch auf mehreren Siebensegmentanzeigen direkt auf der eigens entwickelten Hardware angezeigt. Erwähnenswert an dieser Arbeit ist, dass eine besondere Form der Abtastung verwendet wird (mehr dazu unter 2.3.5). Sie ermöglicht es auf dem relativ leistungsschwachen Controller die Signalanalyse durchzuführen. Die Verstärkung des Sensorsignals geschieht auf einer Verstärkerplatine, allerdings musste die Verstärkung noch mit der Hand, durch Jumper, verändert werden.
Eine weitere Platine schaft dort Abhilfe und automatisiert die Verstärkungsregelung und auch die Offsetkompensation. Auch sie verwendet einen MSP430f1611 von *TI*. Mit ihr wurde es dann möglich, längere Messreihen automatisch durchzuführen.
Diese Messreihen liefern bisher viel versprechende Ergebnisse. Trotzdem ist die Entwicklung noch nicht so weit abgeschlossen, dass man daran denken kann, das Projekt erfolgreich zu beenden. Die Analyse ist momentan noch aufwendig, es wurden also Algorithmen gesucht und gefunden, um den Aufwand zu minimieren [10]. Diese können aber auf Grund von fehlenden Erkenntnissen über die benötigte Genauigkeit noch nicht implementiert werden.

1.1.2 Ziel dieser Arbeit

Diese Arbeit setzt an der Weiterentwicklung der Analysefunktion an, im dem Sinne, dass diese auf einer neuen Entwicklungsplattform stattfinden wird. Die aktuelle Datenverarbeitung bei Autos findet natürlich nicht in Microcontrollern statt, sondern in dafür eigens entwickelten ICs. Sie unterliegen nicht den Nachteilen eines in einer Hochsprache programmierbaren Bauteils; vielmehr können mehrere Instruktion zur Zeit ausgeführt werden. Sie nutzen den Umstand, dass ihr „Programm“ aus Hardware besteht. Dadurch kann nahezu beliebig Parallel gearbeitet werden und man muss sich über Scheduling keine Gedanken machen.

Die Entwicklung eines „FPGA-Prototyps der Signalverarbeitung für ABS-Sensoren mit Diagnosefunktion“ ist der erste Ansatz im Projekt ESZ-ABS, Erfahrungen mit dieser Form der Signalverarbeitung zu sammeln. Zur Durchführung der Aufgabe sollen folgende Schritte erfolgen:

- **Konzeption**
  Recherche, Vorstudien, Analyse der Vorarbeiten, Abschätzungen von Rahmenbedingungen
- **Verfahrensentwicklung und Simulation**
  mit Hilfe von Simualtions- und VHDL-Tools und anhand von generierten und / oder erfassten Signalen
- **Smart Comparator**
  Implementation der Nulldurchgangserkennung mit wechslenden Komparator-Schwellen als Sensorgrundfunktion sowie zur Bildung der fundamentalen Harmonische als Bezugsgröße
- **Verzerrungsermittlung**
  Implementation des Erfassung und Errechnung des Verzerrungsmaßes als Verhältnis der Harmonischenanteile (Lookup-Table, Multiplikation und akkumulierdende Addition, sowie Betrags- und Verhältnis-Errechnung)
- **Implementation auf FPGA**
  Erstellung einer Automatensteuerung für die Funktionsschritte der Signalverarbeitung, Nutzung der Protokollausgabe aus Vorarbeiten, Schnittstelle zur Signalerfassung / Einspeisung und Debugging-Ausgabe
- **Funktionsdemonstration**
  auf dem FPGA-System mit Hilfe errechneter und eingespeister Eingangssignale

- **Test, Variantenvergleich und Bewertung**
  insbesondere Untersuchung und Diskussion zur Rechengenauigkeit und zum Aufwand, sowie zur Gestaltung der Abtastung (vollständige Abtastung oder Unterabtastung)

1.1.3 Einordnung der Diplomarbeit in das Projekt

Am Ende der Entwicklungen soll der Prototyp eines eigenen ICs stehen. Bild 1.6 Seite 12 veranschaulicht, in welchen Schritten die Entwicklung eines Integrierten Schaltkreises durchgeführt wird. Aktuell wird an den ersten fünf Schritten gleichzeitig gearbeitet. Die Algorithmen werden in Matlab weiter optimiert und stabilisiert. Es muss auf der zweiten Ebene eine neue Umsetzung des Algorithmuses zur Regelung der Verstärkung gefunden werden. Der bisher verwendete funktioniert nicht mit allen Encoderrädern. Parallel zu dieser Arbeit wird ein Analog-Interface in VHDL Entwickelt [1] mit der Möglichkeit Einstellungen auf dem FPGA über RS232 zur Laufzeit zu ändern.

Diese Arbeit ist also der erste Schritt auf die Ebene „Logikdesign“. Sie dient als Plattform für die nachfolgenden Entwicklungsschritte.

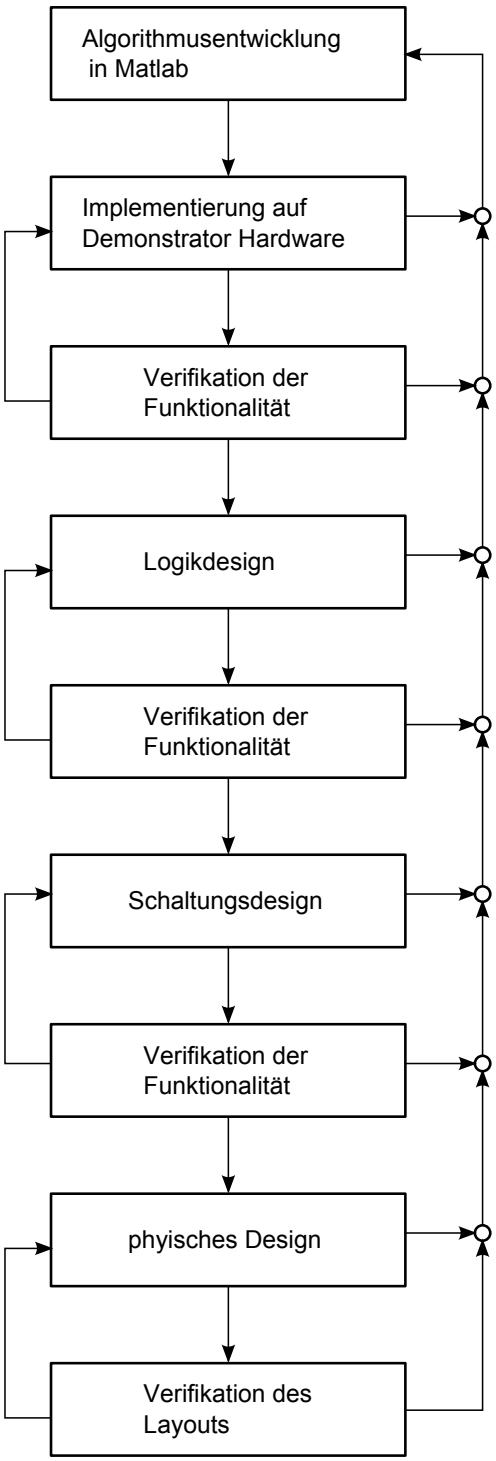


Bild 1.6: *Schritte bei der Entwicklung eines VLSI Designs [24]*

1.2 Der AMR Effekt

Der AMR Effekt wurde 1857 durch William Thomson, 1. Baron Kelvin entdeckt. Eine praktische Anwendung fand man aber erst über 100 Jahre später mit der Entwicklung des Magnetblasenspeichers (Bubblespeichern), welcher aber bald durch die Erfindung der Festplatte verdrängt wurde. Erst ab 1980 wurden AMR-Sensoren entwickelt. [23]

1.2.1 Grundlagen des Anisotropen magnetoresistiven Effektes

Der AMR-Effekt nutzt den Einfluss von Stromflussrichtung und Magnetfeldrichtung auf den Widerstand von Legierungen wie z.B. NiFeCo, wobei der Widerstand am größten ist, wenn Stromfluss und Magnetfeld parallel sind [25].

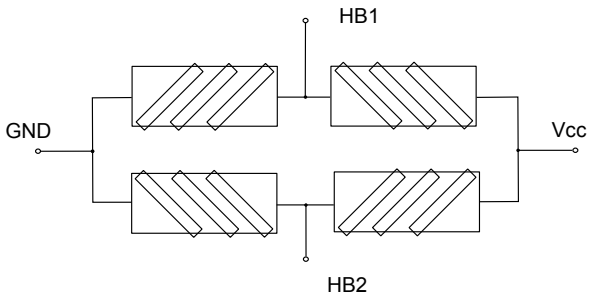


Bild 1.7: *Prinzipeller aufbau eines AMR-Sensors*

Die Magnetisierung zeigt in Längsrichtung der Widerstandsstreifen und die Stromflussrichtung wird durch die um 45° gedrehten Leiterbahnstrukturen um etwa 45° gegenüber der Streifenlängsachse ausgelenkt [21]. Durch Anlegen eines Magnetfeldes wird das Feld der magnetischen Schichten abgelenkt. Je nach Stärke und Richtung des Magnetfeldes verändert sich der Widerstand. Der Brückenaufbau des Sensors dient der Reduktion von Temperatureinflüssen auf die Widerstände (Bild 1.7).
Zum Messen der Beschleunigung verwendet man das Differenzsignal, die Spannung zwischen HB1 und HB2. Die Drehrichtung wird aus den Halbbrückensignalen zur Erde ermittelt.

1.2.2 Aussteuerung des Arbeitpunktes

In Bild 1.8 ist die theoritische Kennlinie des AMR-Sensors abgebildet. Es fällt auf, dass die Funktion im Arbeitspunkt annährend linear ist. Das bedeutet, dass ein sinusförmig angelegtes Magnetfeld eine Sinusförmige Spannung an den Ausgängen erzeugt.

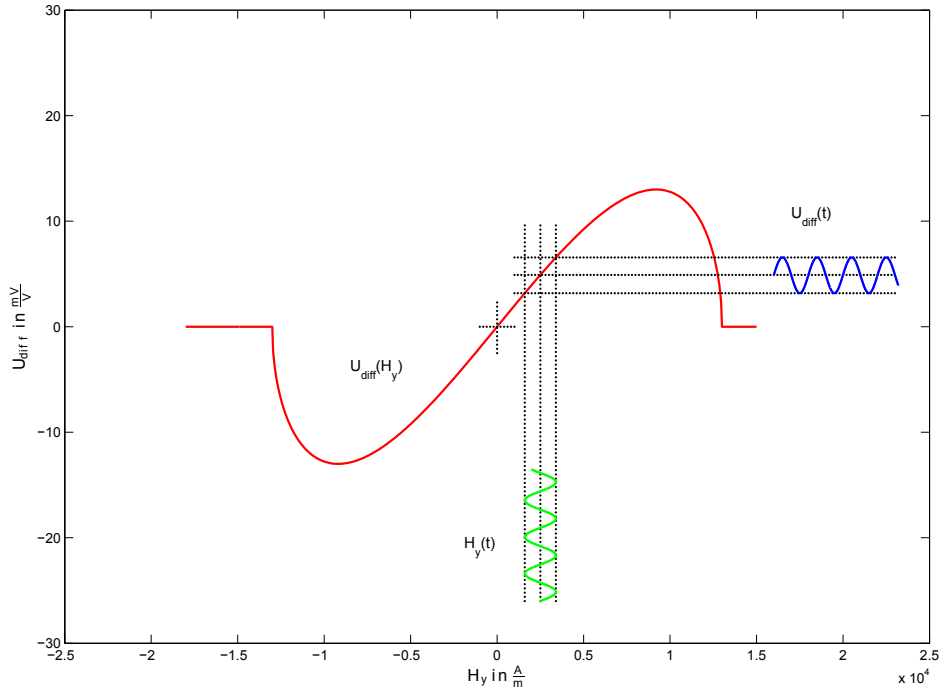


Bild 1.8: *theoritische Kennlinie eines AMR Sensors mit Aussteuerung im quasi-linearen Bereich [2]*

Temperatur und Entfernung zum Encoderrad beeinflussen das Ausgangssignal des Sensors. Eine Änderung der Temperatur verschiebt den Arbeitpunkt in H_y Richtung (Bild 1.9). Eine Änderung des Abstandes zwischen Sensor und Encoderrad verändert die Amplitude von $H_y(t)$ bis sie entweder so groß wird, dass das Ausgangssignal in Maximum und Minimum lokale Minima und Maxima hat (Bild 1.10), oder so klein, dass das Sensorsignal vom Rauschen verdeckt wird.
Weitere Einflüsse auf die Qualität des Sensorsignals sind Wirbelströme und Gegeninduktion, sowie der Winkel des Sensors vor dem Encoderrad.

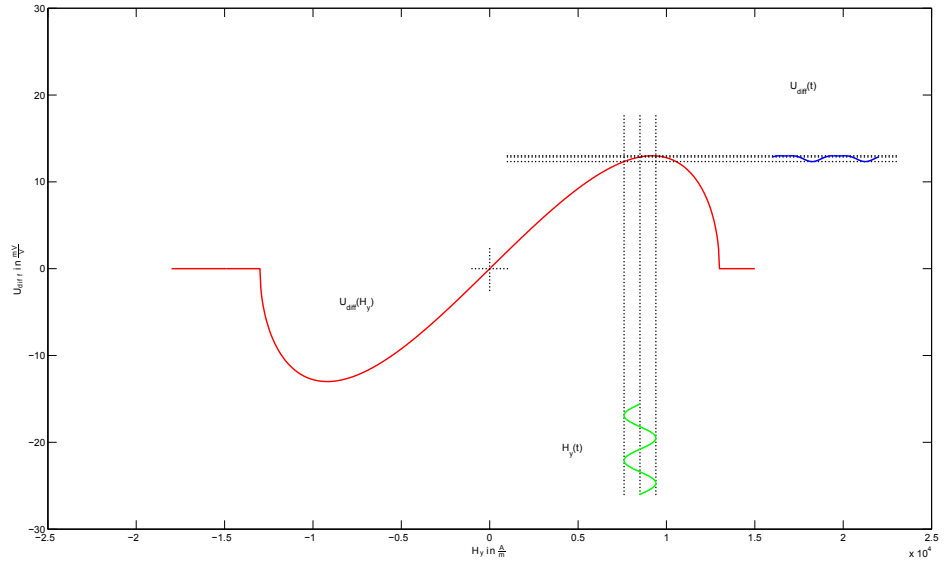


Bild 1.9: *theoritische Kennlinie eines AMR Sensors mit Aussteuerung im nichtlinearen Bereich [2]*

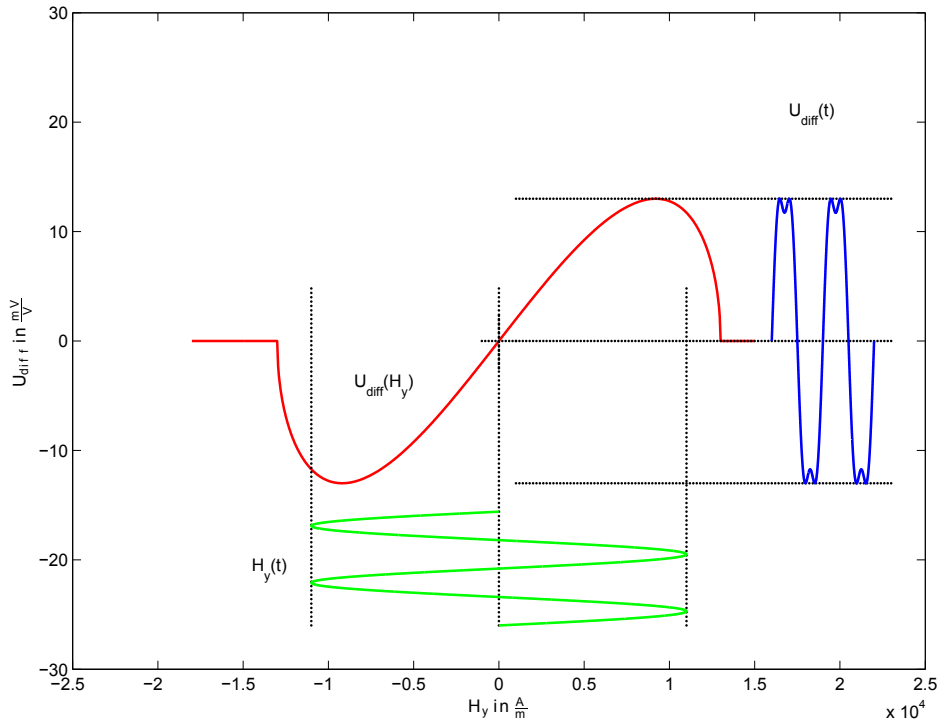


Bild 1.10: *theoritische Kennlinie eines AMR Sensors mit Übersteuereung durch geringe Entfernung zwischen Sensor und Encoderrad [9]*

2 Analyse

Dieses Kapitel veranschaulicht das Konzept der vorliegenden Arbeit. Zunächst werden die Grundfunktionen der aktuellen Sensoren besprochen. Dabei handelt es sich um die Funktionen, welche unabhängig von der Diagnosefunktion funktionieren müssen. Die Diagnosefunktion kann nur eine Aussage über die Zuverlässigkeit des Sensors machen. Danach wird ein kurzer Blick auf die Diagnosefunktion von aktuellen ABS Sensoren geworfen. Es wird deutlich werden, dass die Problematik, die aufgezeigt wird , bekannt ist, aber nur unzureichend in der aktuellen Generation erfasst werden kann. Anschließend wird der Lösungsansatz der Projektgruppe ESZ-ABS vorgestellt und diskutiert. Zu dieser Diskussion gehören die mathematischen Hintergründe des Lösungsansatzes, sowie das Ausarbeiten eines Konzeptes, wie diese Lösung in VHDL umsetzbar ist.

2.1 Funktionen des ABS-Sensors

In diesem Abschnitt werden erst die Grundfunktionen eines ABS Sensors erklärt. Dann wird auf die Analysefunktionalität der aktuellen Sensorgeneration eingegangen, um anschließend eine neuartige Analysefunktion vorzustellen.

2.1.1 Grundfunktionen des ABS-Sensors

Die Grundfunktion des ABS Sensors ist das Übertragen der Drehgeschwindigkeit des Rades. Dieses Signal ist pulsweiten-moduliert; oder anders ausgedrückt: die Nulldurchgänge des sinusförmigen Eingangssignals wurden erkannt und als Pegelwechsel ausgegeben (siehe Bild 2.1).

Dieses Signal ist ausreichend genau, um Änderungen der Geschwindigkeit an jedem Rad erkennen zu können.

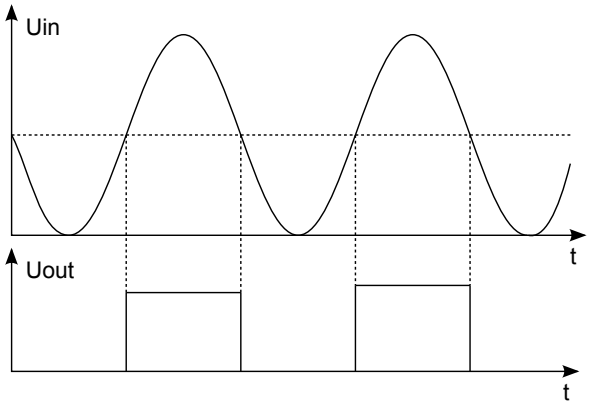


Bild 2.1: *Erkennung der Nulldurchgänge als Sensorausgangssignal*

2.1.2 Funktionszuwachs durch Einführung des „VDA 4.0 Protokolls“

Das VDA 4.0 Protokoll in der Umsetzung der *Daimler AG* [4] erweitert den Funktionsumfang des Sensors. Bild 2.2 zeigt das Verhältnis von Eingangssignal zu Ausgangssignal. Der Sensor überträgt geschwindigkeitsabhängig zwischen 2 und 8 Datenbits und einen Speedpuls, welcher den Nulldurchgang des Sensorsignals markiert.
Da die Bitlänge unabhängig von der Drehgeschwindigkeit des Rades ist, können nur bis zu einer Zahnfrequenz von $900Hz$ alle Bits des Protokolls übertragen werden. Aus Tabelle 2.1 kann entnommen werden, bis zu welcher Zahnfrequenz welche Bits übertragen werden, und welche Informationen ihnen zu entnehmen sind. Bei Betrachten der Tabelle 2.1 fällt auf, dass es in dem Protokoll ungenutzte Bits gibt.
Verwendet wird ein Protokoll, das auf Stromfluss, anstelle von Spannung, basiert. Dieses Protokoll kennt drei Stromstärken. $7mA$ ist der Grundfluss, $14mA$ das High für die Datenbits, $28mA$ für Speedpulse. Bild 2.3 zeigt das Ausgabeprotokoll im Detail. Hier erkennt man, dass das Signal manchestercodiert ist, d.h. dass nicht der Pegel der Signalträger ist, sondern die Richtung der Flanke. Die Zeiten t_p, t_s und t_d sind fest definiert, t_2 ist abhängig von der Drehgeschwindigkeit des Rades.

2.1.3 Bisheriger Ansatz zur Diagnose am Beispiel des KMI22/1 von NXP

Die Gegenüberstellung in Tabelle 2.1 zeigt, dass man sich entschlossen hat, die von der im VDA 4.0 Protokoll nicht definierten Bits für eine Zustandserkennung zu verwenden. Auf Bit 5 bis 7 wird eine Aussage über die Einbaulage des Sensors ausgegeben. Bit 1 ist im Initial Modus gesetzt, Bit 2 bei einem `low` auf dem digitalen

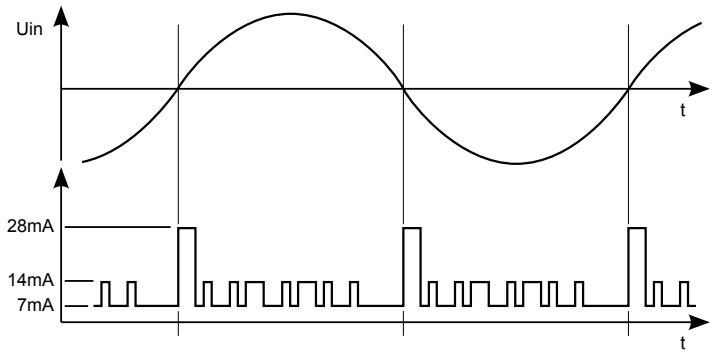


Bild 2.2: *Beispiel für Ein- und Ausgang eines ABS-Sensors mit VDA 4.0 Protokoll*

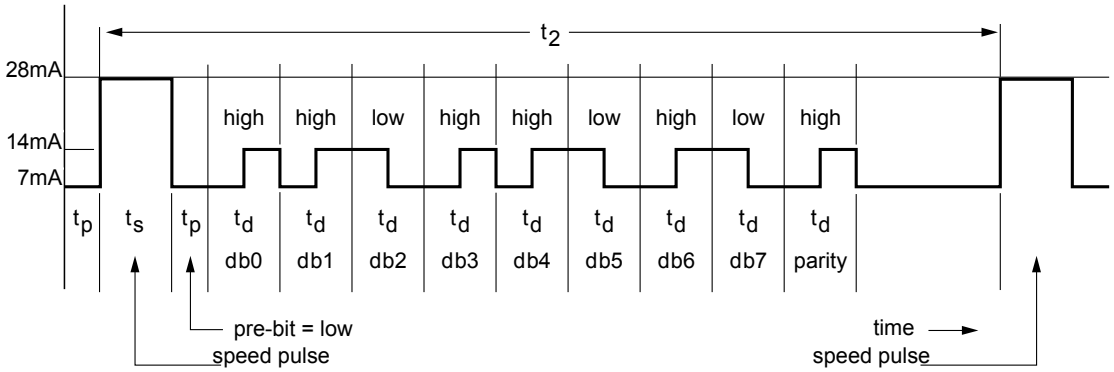


Bild 2.3: *Beispiel des Ausgabeprotokolls [4]*

Bit	**Gesendet bis**	**Wert nach Daimler**[4]	**Wert nach NXP**[17]
0	$2,5kHz$	Luftspaltreserve	Luftspaltreserve
1	$2,5kHz$	Frei	Mode State
2	$2,0kHz$	Frei	Digital input State
3	$1,6kHz$	Drehrichtung Gültigkeit	Drehrichtung Gültigkeit
4	$1,4kHz$	Drehrichtung	Drehrichtung
5	$1,2kHz$	Frei	Distance Bit 0
6	$1,1kHz$	Frei	Distance Bit 1
7	$1,0kHz$	Frei	Distance Bit 2
8	$900Hz$	Parität	Parität

Tabelle 2.1: *Definition der Datenbits des VDA 4.0 Protokolls*

Input Pin.
Die Aussage auf Bit 5 bis 7 beruht auf der Amplitude des Sensorsignals und ist wie auf Bild 2.4 dargestellt codiert. Das Signal wird im Sensor verstärkt. Mit der Verstärkung nimmt auch der Anteil des Rauschens auf dem Signal zu, so dass Signale mit geringer Amplitude schwerer zu interpretieren sind.

Dieser Ansatz ist gut, berücksichtigt aber nicht alle Fehlerquellen, die auftreten können, wie z.B. eine Verkippung des Sensors oder auftretende Nichtlinearitäten bei Unterschreiten eines Mindestabstandes (siehe Abschnitt 4.2, Bild 1.10). Es wird also nur das Vorhandensein des Sensorsignals quantisiert, unabhängig von seiner eigentlichen Qualität. Oder um es anders auszudrücken: ein mehrdimensionales Problem wird eindimensional betrachtet.
So kann es, wie gezeigt werden wird, auch bei voll ausgesteuertem Eingangssignal Fehldeutungen geben, wenn das Sensorsystem nicht auf das Erkennen dieser Fehler vorbereitet ist.

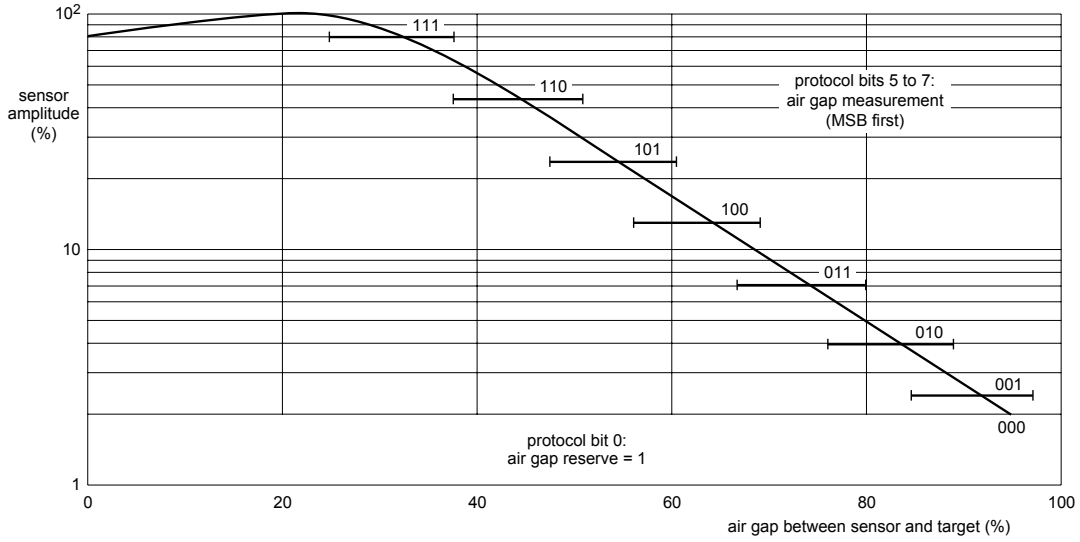


Bild 2.4: *Sensoramplitude über Luftspalt [17]*

2.1.4 Ansatz zur Diagnose durch Analyse der Harmonischen

Der erste Ansatz der Projektgruppe ESZ-ABS war es, der Problemlösung eine Dimension hinzuzufügen. Dazu wurde die Kennlinie des Sensors betrachtet und beschlossen, Nichtlinearität bei Übersteuerung durch Berechnung des THDs zu erkennen. Betrachtet man den THD, abhängig von der Entfernung des Sensors zum Encoderrad, bekommt man eine Graphik, die der auf Bild 2.5 ähnelt.

Ist der Sensor zu nahe am Encoderrad platziert, ist die Amplitude des Sensorsignals so groß, dass es bedingt durch die Kennlinie zu Oberwellen auf dem Sinussignal kommt (siehe Bild 1.10). Ist der Sensor zu weit vom Encoderrad entfernt, muss das Signal verstärkt werden. Mit dem Signal wird auch das Rauschen verstärkt, bis das Signal letztendlich nicht mehr interpretierbar ist. Sowohl die Nichtlinearität bei großen Amplituden als auch Rauschen verformen ein eigentlich ideal sinusförmiges Eingangssignal. Oberwellen scheinen also ein zur Analyse geeigneter Indikator

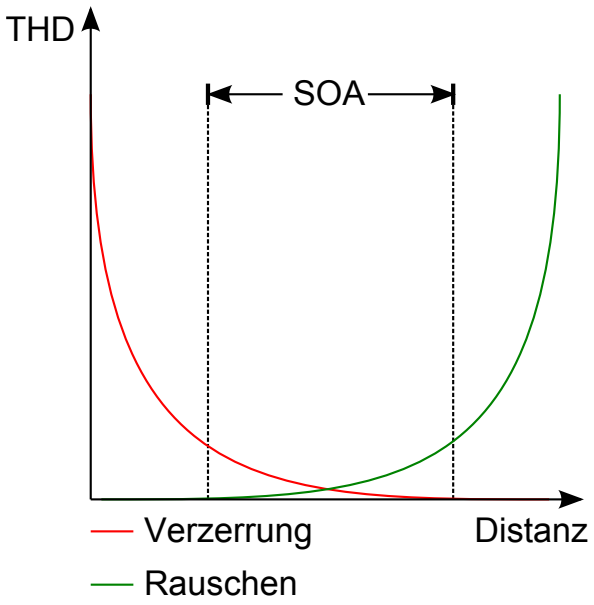


Bild 2.5: *Prinzipielle Darstellung des THD, aufgetragen über Abstand von Sensor zum Encoderrad*

zu sein, um bei einem nicht (oder nur leicht) verkippten Sensor unzuverlässige Eingangsgrößen zu erkennen.

2.2 Bestimmung der harmonischen Störanteile

Bei Harmonischen handelt es sich um die Oberwellen eines Signals, also jene Anteile, die mit ganzzahlig vielfacher Frequenz eines Grundsignals auftreten. Um die Harmonischen des Sensorsignals zu bestimmen, werden die Messwerte Fourier transformiert. Da es sich bei dieser Diplomarbeit um einen ersten VHDL Prototypen handelt, wird hier eine DFT (Diskrete Fourier Transformation) ohne Optimierungen durchgeführt. Optimierte Algorithmen wurden entwickelt [10], standen aber bei der Erstellung des Konzepts dieser Arbeit noch nicht zur Verfügung.

2.2.1 Einführung zur DFT

Um die harmonischen Anteile an dem Sensorsignal zu bestimmen, muss man dieses Signal in die Frequenzanteile zerlegen. Dafür wird hier die DFT verwendet. Diese basiert auf der Fouriertransformation, ist im Gegensatz zu ihr aber auf endliche Abtastfolgen anwendbar. Die Fouriertransformation betrachtet ein periodisches Signal als Summe aus Gleichanteil, Grundschwingung und Oberschwingung [13]. Die allgemeine Form der Fourierreihe ist:

$$x(t) = \frac{a_0}{2} + \sum_{k=1}^{\infty} (a_k cos(k\omega t) + b_k sin(k\omega t)) \text{ mit } \omega = \frac{2\pi}{T} \tag{2.1}$$

wodurch sich die Fourierkoeffizienten zu

$$a_k = \frac{1}{\pi} \int_0^{2\pi} x(t) cos(k\omega t) \mathrm{d}t, \text{mit } k = 0, 1, 2, ... \tag{2.2}$$

und

$$b_k = \frac{1}{\pi} \int_0^{2\pi} x(t) sin(k\omega t) \mathrm{d}t, \text{mit } k = 0, 1, 2, ... \tag{2.3}$$

ergeben.

Die allgemeine Form ist aber nicht dienlich, da die Signale abgetastet werden. Das bedeutet, dass die diskrete Fourier Transformation verwendet werden muss. Sie erlaubt es, das „diskrete Frequnzspektrum eines zeitdiskreten Signals endlicher Länge“ [8] zu betrachten.

Es wurden N Abtastwerte eines unbekannten Signals genommen und als Abtastfolge $x[k]$ abgelegt. Es gilt:

$$X[n] = \sum_{k=0}^{N-1} x[k] \cdot e^{-j\frac{2\pi nk}{N}} \text{ mit } 0 \leq n \leq N-1 \tag{2.4}$$

und

$$e^{-j\varphi} = cos\varphi - jsin\varphi \tag{2.5}$$

Setzt man in 2.4 die Eulersche Formel 2.5 ein, erhält man

$$X[n] = \sum_{k=0}^{N-1} x[k] \left(cos\left(\frac{2\pi}{N} nk\right) - jsin\left(\frac{2\pi}{N} nk\right)\right) \tag{2.6}$$

Die Gleichung 2.6 analysiert den gesamten Vektor x nach seinen Frequenzanteilen. Dies geschieht, indem ein Sinus und ein Cosinus, aus N Werten pro Periode bestehend, über den Vektor x gelegt wird. Wenn der Vektor x Anteile hat, die nach N Abtastwerten periodisch sind, dann werden diese durch die Multiplikation betont, erhöhen also $X[n]$. Falls dem nicht so ist, heben sich die gewichteten Werte aufgrund der Sinusform in der zweiten Halbwelle auf. Dies wird, mit höheren Sinus- und Cosinusfrequenzen (durch Inkrement von n), bis $n = N-1$, wiederholt. Bei $n = N-1$ bestehen die Gewichtungssignale Sinus und Cosinus nur noch aus Extrempunkten.

Dann entspricht ihre Frequenz der, die durch N Abtastwerte maximal darstellbar ist.

2.2.2 Bestimmung der Harmonischen

Der Aufwand zur Bestimmung der Frequenzanteile kann verringert werden. Da das Eingangssignal des Sensors idealerweise sinusförmig ist, kann man durch Erkennung der Nulldurchgänge (siehe 2.1.1) die Dauer einer Periode messen. So erhält man die Grundfrequenz des Sensorsignals. Diese und ihre ganzzahligen Vielfache nennt man „harmonische Frequenzen“, kurz: Harmonische.
Wenn die Grundfrequenz bekannt ist, muss man eine Annahme treffen, auf der die eingangs erwähnte Aufwandsveringerung beruht. Diese Annahme lautet: Die Zahnfrequenz ist annähernd konstant. Daraus folgt, dass die Grundfrequenz des Sensorsignals annähernd konstant ist. Das bedeutet, dass die Zeit, die zwischen zwei Nulldurchgängen des Sensorsignals vergeht, beinahe konstant ist, wenn man zwei benachbarte Perioden betrachtet. Es ist dann möglich, die Periodendauer zur Abtastung in S äquivalente Intervalle zu teilen. Oder anders ausgedrückt: man passt die Abtastfrequenz an die Grundfrequenz des Eingangssignals an und erhält immer S Abtastwerte pro Periode.
Wenn man also das Signal aus S Abtastwerten auf die ersten fünf Harmonischen h untersuchen will, berechnet man

$$X[h] = \sum_{k=0}^{S-1} x[k] \left(cos\left(\frac{2\pi}{S} hk\right) - jsin\left(\frac{2\pi}{S} hk\right)\right) \text{ mit } 1 \leq h \leq 5 \tag{2.7}$$

$X[h]$ aus Gleichung 2.7 enthält jetzt Amplitude und Phase der Harmonischen h des Sensorsignals.

2.2.3 Berechnung des THD

Der THD (Gesamte harmonische Verzerrung, engl. total harmonic distortion) Verhältniswert ist ein Maß der Verzerrung und kann in dB oder Prozent angegeben werden. In diesem Projekt wurde die Darstellung in Prozent gewählt.
Der THD besteht aus dem Anteil der Störleistung an der Gesamtleistung eines Signals [13]:

$$THD = \sqrt{\frac{P_{ges} - P_1}{P_{ges}}} \tag{2.8}$$

Da es sich bei den in 2.2.2 bestimmten Werten um ein Spannungssignal handelt, ist das Verhältnis der Effektivwerte zueinander gleich dem der Leistungswerte, so dass man Ergebnisse aus 2.6 einsetzen kann:

$$THD = \sqrt{\frac{\sum_{h=2}^{\infty} |X[h]|^2}{\sum_{h=1}^{\infty} |X[h]|^2}} \tag{2.9}$$

Es wird also mit den quadrierten Beträgen der harmonischen Anteile gerechnet. Dadurch wird die Berechnung der Wurzel an dieser Stelle unnötig, es reicht die Vorzeichen zu vernachlässigen. Berechnet wird:

$$THD = \sqrt{\frac{\sum_{h=2}^{\infty} |Re_h|^2 + |Im_h|^2}{\sum_{h=1}^{\infty} |Re_h|^2 + |Im_h|^2}} \tag{2.10}$$

2.2.4 Aufwand der Berechnung

In früheren Arbeiten wurde Entschieden, dass zur Diagnose die ersten fünf Harmonischen aus 64 Abtastwerten pro Periode bestimmt werden. Es stellt sich also die Frage, warum hier mit der DFT gearbeitet wird, anstelle der FFT (Fast-Fourier-Transformation). Die FFT würde sich anbieten auf Grund der 64 Abtastwerte, aber man bedenke, dass nur die ersten fünf Harmonischen in die Diagnose einfließen. Es ergibt sich die Aufwandsbetrachtung in Tabelle 2.2.

	FFT	DFT (alg.)	DFT (spez.)
Aufwand, allg.	$S \cdot ld(S)$	S^2	$S \cdot N$
Aufwand, hier	384	4096	320

Tabelle 2.2: *Aufwand der Berechnung*

Die Tabelle verdeutlicht, dass die Berechnung des Vollständigen Spektrums, wie es bei der FFT immer ausgeführt wird, bei der DFT aufwändiger ist. Da die Diagnose aber nur anhand der ersten fünf Harmoischen berechnet wird, ist hier der Aufwand bei der DFT geringer als bei der FFT.

2.3 Konzeptionierung der Umsetzung in VHDL

Bei der Ausarbeitung des Konzeptes gilt es zuerst eine geeignete Entwicklungsplattform auszusuchen. Anschließend müssen Entscheidungen zur Gestaltung der Systemarchitektur getroffen werden. Diskutiert wird in diesem Abschnitt:

- Grad der Parallelisierung
- Datenfluss
- Genauigkeit
- Abtastungsverfahren

Anschließend werden die Entscheidungen dargestellt und begründet.

2.3.1 Verwendete Hardware

Als Entwicklungsplattform dient das Nexys2 Board von Digilent, erweitert durch eine Platine mit ADC/DAC [1]. Verbunden werden diese beiden Boards durch ein Adapterboard.
Verwendet wird das Nexys2 Entwicklungsboard aufgrund verschiedener Charakteristika:

- XILINX Spartan 3e: ein ausreichend großer und schneller FPGA
- acht Schalter
- vier Taster
- vier Siebensegmentanzeigen
- RS232 Anschluß

Maßgeblich für die Entscheidung für das Nexsys 2 Board war aber dessen Verfügbarkeit. Es wird an der HAW-Hamburg, zusammen mit einem Adapterboard (ModSys) an welches man verschiedene Hardwarekomponenten anschließen kann, bei Labor versuchen verwendet.

Da am Ende des Projekts die Entwicklung eines integrierten Schaltkreises steht, muss hier schon bei der Entwicklung darauf geachtet werden, dass sich sowohl die Leistungsaufnahme als auch die Größe (auf Silizium) nicht zu sehr von den aktuellen Sensoren unterscheidet.
Richtwerte, die veröffentlicht werden dürfen, sind schwer zu finden zu diesem Thema. Deshalb wird sich hier zum Vergleich auf einen Speed Sensor von Delphi [5]

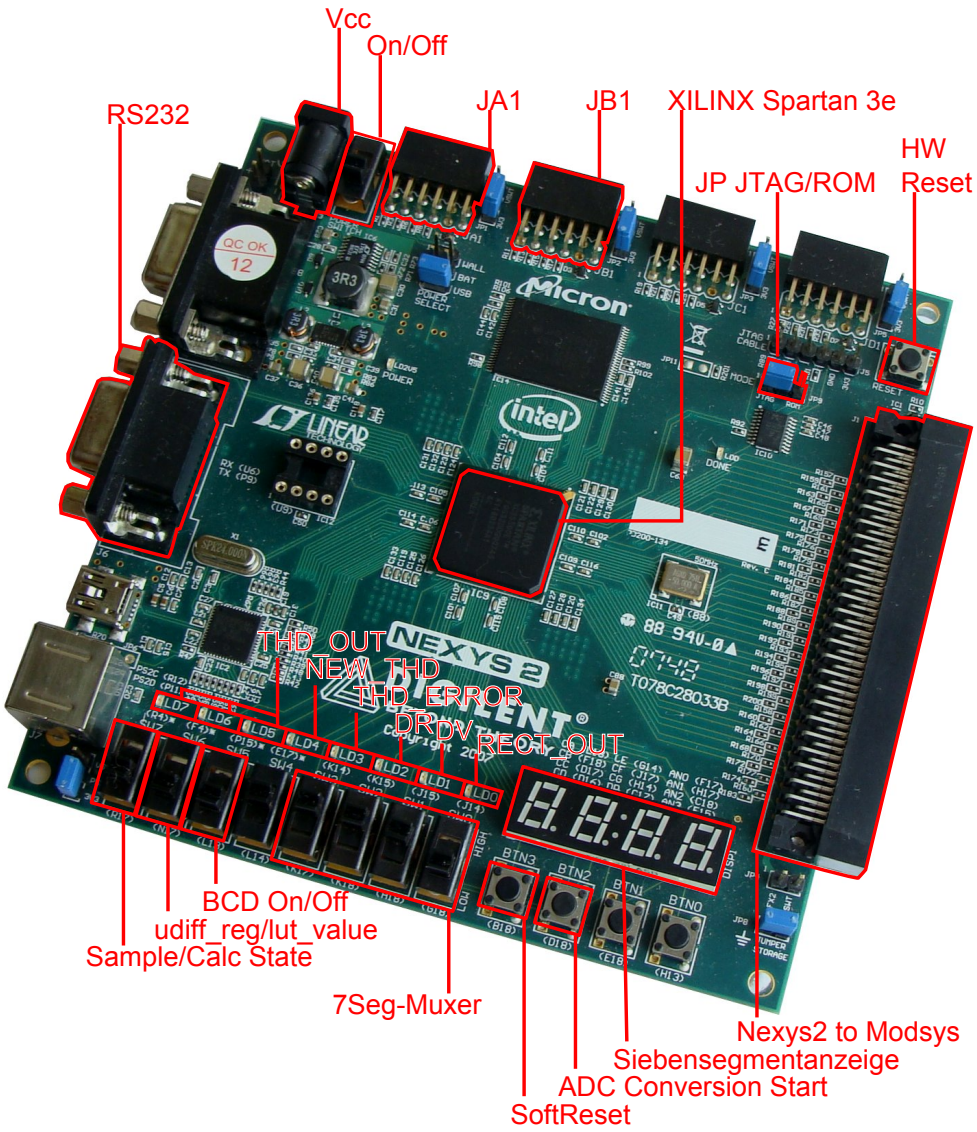


Bild 2.6: *Nexys2 Board, wichtige Elemente markiert (Bezeichnungen, siehe 3.6.2)*

gestützt, sowie den in dem Projekt schon verwendeten MSP430f169 von Texas Instrument [9]. Der Delphi Speed Sensor benötigt laut Datenblatt bei $4,5V - 16V$ weniger als $16mA$, was bedeutet, dass die Leistungsaufnahme kleiner $256mW$ ist. Der MSP430 benötigt $500\mu A$ pro MHz Takt, so dass man bei 8Mhz $4mA$ zusätzlich zur Protokollausgabe (welche bis zu $28mA$ benötigt) aufwenden muss. Man sieht, wie eine steigende Taktfrequenz Einfluss auf die Leistungsaufnahme hat. Deshalb wurde entschieden, den FPGA nur auf $12,5MHz$ zu takten. Dies scheint ein Wert zu sein, welcher gering genug ist, um praxisnah zu sein. Über die aktuelle Sensorgeneration ist folgendes bekannt:

- Die Stromaufnahme darf $7mA$ nicht überschreiten. Dies ist der Basisstrom des digitalen Ausgabeprotokolls. Als Spannung kann man von ca. $3,3V$ ausgehen.
- Der Brückenstrom beträgt ca. $1mA$
- Der Ausgangstreiber benötigt ca. $1mA$
- Die Verstärkungsregelung benötigt ca. $2mA$

Es bleiben also nur etwa $2-3mA$ übrig, um die Diagnosefunktion um zu setzten.

2.3.2 Grad der Parallelisierung

Die beiden wertvollsten Güter, neben der schon besprochenen Leistungsaufnahme, bei einem solchen Entwurf sind Platz (auf einem IC) und Zeit. Diese schließen sich leider gegenseitig aus: ein Design kann nicht schneller werden und gleichzeitig weniger Hardware benötigen (es sei denn die Taktfrequenz wird erhöht, was aber die Leistungsaufnahme steigert). Dies soll an einem Beispiel verdeutlicht werden. Angenommen sei Gleichung 2.11:

$$(a+b)\cdot(c+d+e)=f \tag{2.11}$$

Es gibt die Möglichkeit erst die Summe aus a und b zu berechnen und diese zu Speichern. Dann wird $c+d$ berechnet, gespeichert und anschließend wird e dazu addiert. Am Ende wird die Multiplikation als eine Reihe von Additionen und Schiebeoperationen durchgeführt. Da in jedem Schritt nur eine Operation vorkommt, währe hier ein Addierer ausreichend, um f zu errechen. Am schnellsten ist die Berechnung jedoch, wenn alle Summen gleichzeitig gebildet und die beiden Ergebnisse anschließend multipliziert werden. Das Ergebnis ist in nur zwei Schritten berechnet, aber auf Kosten von mehr Hardware: es werden drei Additionen gleichzeitig vollzogen und so werden dann auch drei Addierer benötigt. Man sieht also hier beide Enden eines Spektrums, in welchem man den richtigen Weg zwischen vertretbarem Hardwareaufwand und hinnehmbarer Geschwindigkeit wählen muss.

2.3.3 Datenfluss

Datenfluss und der Grad der Parallelisierung hängen direkt voneinander ab. Hier besteht das Spektrum aus einer Pipeline an einem Ende, und **einem** Zustandsautomaten am anderen.

Bild 2.7 soll dies schematisch an einem Beispiel verdeutlichen. Bekannt ist nur, dass es kontinuierlich Eingangsdaten gibt und jedes Datum vier Verarbeitungsprozesse

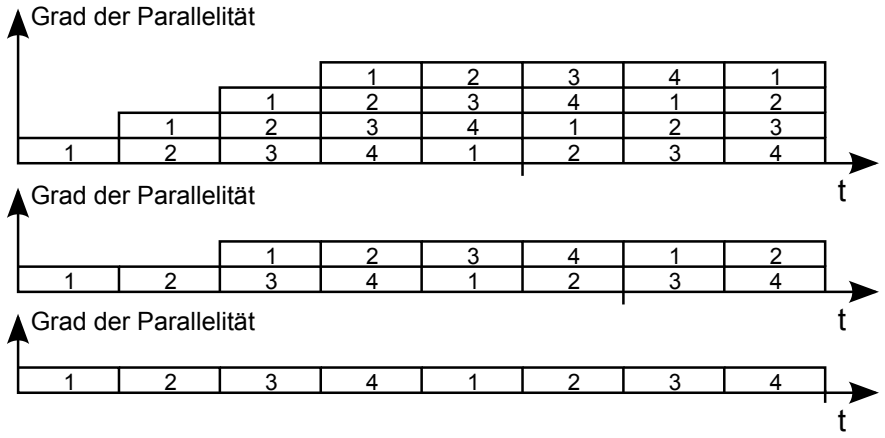


Bild 2.7: *Beispiele für Datenfluss*

durchläuft. Über die Prozesse eins bis vier ist nichts bekannt. Es könnten gleiche dabei sein, es können aber auch alle Verschieden sein. Sind die Prozesse alle verschieden, dann kann, ohne zusätzlichen Platz zu benötigen, die oberste Implementation gewählt werden. Die Prozesse greifen nahtlos ineinander und sind immer ausgelastet. Nach der vierten Zeiteinheit liegt am Ausgang das erste Ergebnis vor. Jetzt sind alle Komponenten mit Daten gefüllt; es folgt nach jeder Zeiteinheit ein neuer Wert am Ende der Pipeline.
Das Beispiel Nummer zwei arbeitet erst nach zwei Zeiteinheiten parallel, was daran liegen kann, dass entweder Komponente eins und zwei gleich sind, oder Komponente drei und vier, oder sowohl eins und zwei, drei und vier. Folge dessen ist, dass nach vier Zeiteinheiten das erste Ergebnis vorliegt und ab dann alle zwei Zeiteinheiten eines folgt.
Beispiel drei ist das langsamste, in welchem aber die Komponenten frei belegbar sind. Es können also alle Komponenten gleich sein, alle verschieden oder eine Kombination dazwischen.

2.3.4 Genauigkeit

Wenn es darum geht, Platz auf dem IC zu sparen, muss man auch über die Genauigkeit sprechen, mit der gerechnet wird. Die obere Schranke zu verwenden bedeutet, dass in keinem Fall ein Bit verloren geht. Eingangsannahmen sind dafür ein 12 Bit ADC und eine 12 Bit Sinus/Cosinus Wertetabelle.

- 12 Bit ADC mal 12 Bit Sinus/Cosinus = 24 Bit
- 24 Bit über 64 Abtastwerte addiert: 24 Bit + 63 Bit = 87 Bit
- 87 Bit Absolutwert gebildet 86 Bit

- 86 Bit quadriert: 86 * 2 = 172 Bit
- 172 Bit zehn mal addiert: 172 Bit + 9 Bit = 181 Bit.

Dieses Vorgehen nimmt keine Rücksicht auf die Plausibilität der Werte. 181 Bit ist die theorietische maximale Breite des Nenners, 179 Bit die des Zählers. Diese theoretische Obergrenze betrachtet nur, wie sich Bitlängen bei mathematischen Operationen verhalten. Sie nimmt keine Rücksicht auf die Signalform und das Gewichten des Signals mit Sinus und Cosinus. Es ist also möglich, ohne Datenverlust mit kürzeren Vektoren zu arbeiten, da diese Maximalwerte nie erreicht werden können.
Weitere Möglichkeiten zum kürzen der Vektoren würden auf Kosten der Genauigkeit des Ergebnisses gehen. Die beiden einfachsten Stellschrauben sind dabei die Bitbreite des ADCs und die Abtastrate. Dabei ist natürlich auf die Einhaltung des Nyquist Theorems zu achten. Dieses schreibt vor, dass nicht weniger als 10 Sample pro Periode genommen werden dürfen, da das Signal bis zur fünften Harmonischen analysiert werden soll.
Eine Halbierung der Abtastwerte auf 32 würde bei ansonsten gleich bleibenden Annahmen einen Ausgangsvektor von 118 Bit bedeuten, also einer Ersparnis von ca. 35% am Ergebnis, was natürlich zu kleineren Addierern führt und somit zu geringerem Ressourcenverbrauch.
Bei einer Reduzierung auf 16 Samples, wäre das Ergebnis nur noch 86 Bit breit.

$$f(x) = 2967 + 800 \cdot sin\left(\frac{2\pi x}{num\_samples}\right) + 200 \cdot sin\left(\frac{10\pi x}{num\_samples} + 7\right) \tag{2.12}$$

Ein Test mit der Funktion (siehe Bild 2.8, S. 29) ergab die in Tabelle 2.3 gelisteten Werte. Diese deuten darauf hin, dass eine Reduzierung der Abtastwerte bis runter auf 16 zu keiner deutlichen Verschlechterung des Ergebnisses führt.

num_samples	128	64	32	16	8
THD in %	24,247	24,240	24,249	24,269	33,348

Tabelle 2.3: *THD der Funktion 2.12, abhängig von der Anzahl der Abtastwerte*

Diese Aussage gilt natürlich nur für diese Funktion. Um sie allgemein treffen zu können, müsste diese Thematik genauer untersucht werden.

Die andere Möglichkeit, um die Bitbreite des Ergebnisses zu reduzieren, ist ein ADC mit geringerer Auflösung. Bei 64 Abtastwerten wird betrachtet, wie sich die Funktion 2.12 verhält, wenn die Bitbreite der Sinus/Cosinus Wertetabelle und des ADCs reduziert wird. Die Ergebnisse sind in Tabelle 2.4 abgebildet

Auch hier fällt auf, dass der THD sich (für diese Eingangsfunktion) nicht besonders stark verändert Dies liegt aber auch an der Ausstuerung des Signals. Sollte der

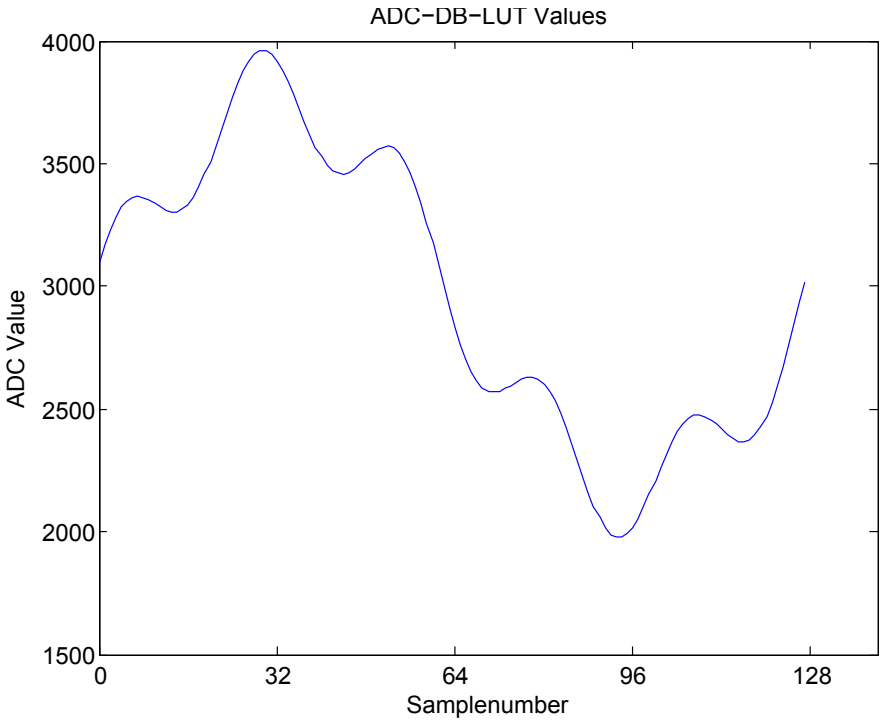


Bild 2.8: *Funktion 2.12 mit* `num_samples` *= 128*

`bit_length`	12	10	8	4
THD in %	24,240	24,251	24,171	23,538

Tabelle 2.4: *THD der Funktion 2.12, abhängig von der Bitbreite von ADC und Sinus-/Cosinus Wertetabelle*

ADC wirklich nur vier Bit haben, würde das Eingangssignal im Idealfall 16 Schritte haben. Man kann aber nicht von einem optimal ausgesteuerten Signal ausgehen, deshalb sollte an dieser Stelle nicht zu sehr gespart werden. Es würde sich eher anbiete, die Bitbreite der Sinus/Cosinus Wertetabelle zu kürzen, da diese immer perfekt „ausgesteuert“ ist.

2.3.5 Abtastung

Die Abtastung wurde in 2.3.4 schon kurz angesprochen, dort wurde aber nur die Möglichkeit diskutiert, sie pro berechnetem THD Wert zu verringern.
So würde man nicht nur Ressourcen sparen, sondern auch zwischen den Abtastungen mehr Zeit für die Verarbeitung der abgetasteten Daten haben. Eine weitere Möglichkeit, um Zeit zu gewinnen, besteht in dem Prinzip der Unterabtastung [9].

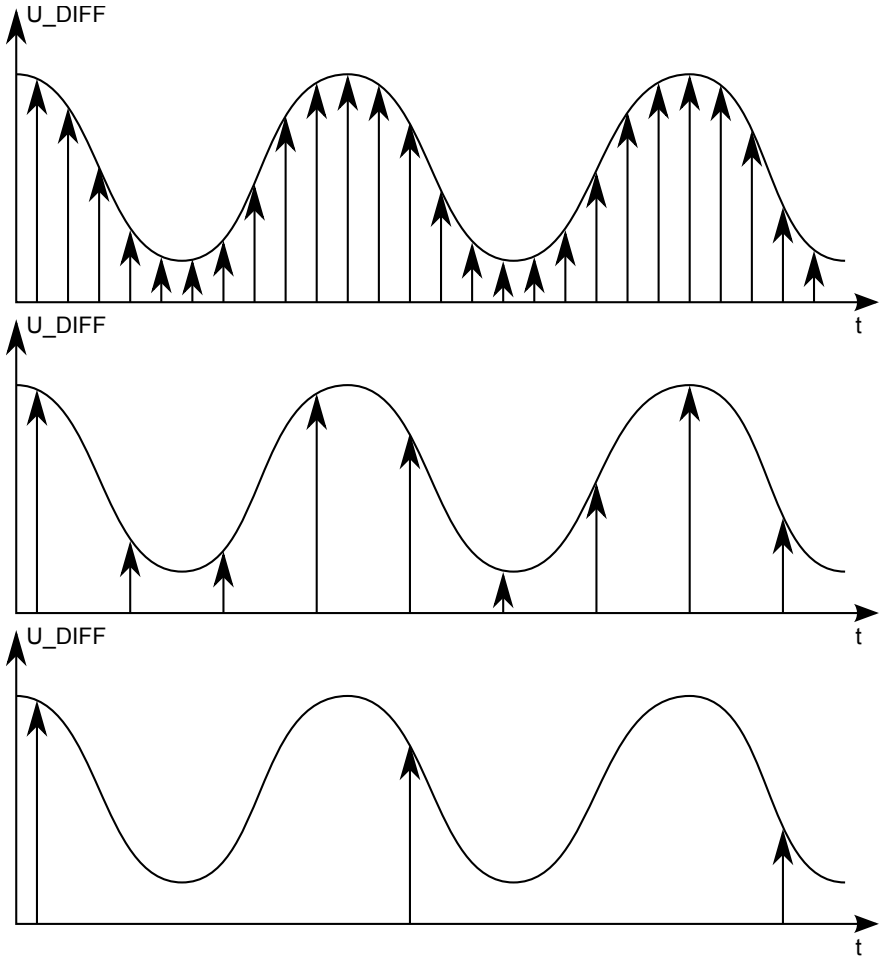


Bild 2.9: *Abtastung von Eingangssignalen*
oben: normale Abtastung
mitte: teilerfremde Periodenabtastung
unten: Unterabtastung

Normale Abtastung (Bild 2.9, oben) hält das Nyquisttheorem ein, indem ein Signal entsprechend schnell abgetastet wird. Unterabtastung setzt voraus, dass das Signal über mehrere Perioden nahezu identisch ist, also in Periodendauer und Amplitude sich nicht verändert. Ist dem so, kann man wie auf Bild 2.9 unten dargestellt, das Abtasten einer Periode über mehrere Perioden verteilen. Man benötigt dabei so viele Perioden wie Abtastwerte, da die Abtastwerte weiterhin in der Reihe nach genommen werden.
Es gibt aber noch (mindestens) eine weitere Möglichkeit, Abtastung zu implementieren: die teilerfremde Periodenabtastung. Sie ist möglich bei periodischen Signalen mit einer konstanten Menge an Abtastwerten S pro Periode. Sollte es nicht möglich sein, diese Menge S schnell genug zu verarbeiten, kann man alternativ jeden

u-ten Abtastwert nehmen, solange u und S teilerfremd sind. Nimmt man z.B. von $S = 64$ Abtastwerten pro Periode nur jeden dritten ($u = 3$), benötigt man zum Abtasten einer Pseudoperiode (also zum Erhalten von 64 Werten) drei Perioden. Man würde in der ersten Periode mit Abtastwert 0 anfangen und die Samples würden in folgender Reihenfolge genommen werden:

0,3,6,9,...,57,60,63,2,5,8,...,58,61,1,4,7,...,56,59,62

Wichtig ist, dass man die Abtastwerte aber trotzdem so behandelt, als wären sie in der korrekten Reihenfolge aufgenommen worden.
Man muss also darauf achten, dass jedes Sample mit dem richtigen Wert der Sinus/Cosinus Wertetabelle multipliziert wird. Dies kann man erreichen, indem man die Überlauffunktion der Addition unter VHDL ausnutzt. Soll heißen, man nimmt 2^n Abtastwerte, hat also einen Zählvektor, der 2^{n-1} Bit lang ist. Addiert man zu dem Vektor immer 3 hinzu, so läuft er über und man ist automatisch bei dem richtigen ersten Sample der nächsten Periode.

2.3.6 Bedeutung der Entscheidungen für das System

Ziel der Arbeit ist der Entwurf eines ersten Prototypen für die Signalverarbeitung von ABS Sensordaten in VHDL. Dem entsprechend ist es nicht das Ziel, ein fertiges, optimiertes System als Ergebnis zu haben. Entworfen wird eine Entwicklungsplattform für weiters Testen auf einem FPGA. Für diese wurden in Abschnitt 2.3.2 bis 2.3.5 die jetzt folgenden Entscheidungen getroffen.

Zu 2.3.2 Grad der Parallelisierung:
Um Platz zu sparen, werden Zugriffe möglichst sequentiell sein. Das bedeutet: sequentielle Multiplizierer und nur eine Wertetabelle für Sinus und Cosinus. In 2.3.4 wurde erwähnt, dass jeder Abtastwert mit einem Sinus- und einem Cosinuswert multipliziert wird. Daraus ergibt sich folgender Zeitrahmen:

- Das Nachschlagen des Sinus-/Cosinuswerte dauert zwei Takte.
- Die Multiplikation benötigt einen Takt pro Länge des kürzeren Vektors.
- Die Addition der Werte dauert einen Takt.

So kann es zwar sein, dass die Geschwindigkeit der Berechnung nicht hoch genug ist, aber Unterabtastung ist einem höheren Ressourcenverbrauch vorzuziehen und kann (wie in 2.3.5 besprochen) mit geringem Aufwand nachträglich implementiert werden.

Zu 2.3.3 Datenfluss:
Das Vorbereiten der THD Berechnung ist auf Grund der vielen Wertetabellenzugriffe und Multiplikationen zeitaufwändig, so dass nicht davon auszugehen ist, dass nach Ende der Datenaufnahme genügend Takte in einer Periode übrig sind, um den THD zu berechnen. Deshalb wird diese Berechnung von der Multiplikation/Addition abgetrennt und nach ihr durchgeführt. Die Verarbeitung der Daten selber geschieht auf Grund der variablen Abtastrate nicht unbedingt ohne Leerläufe. Deshalb bietet sich ein Zustandsautomat zum Steuern der Komponenten an. So werden die Komponenten zentral gesetzt und auch zurückgesetzt. Der gewünschte modulare Aufbau wird dadurch erleichtert, da alle Steuersignale zentral zusammenlaufen und die Komponenten so weniger Abhängigkeiten zu einander haben.
Benötigt werden zwei Zustandsautomaten: einen der die Werte abtastet und in Real- und Imaginärteil einteilt, und einen, der dann daraus den THD berechnet. Dieses geschieht dann parallel.

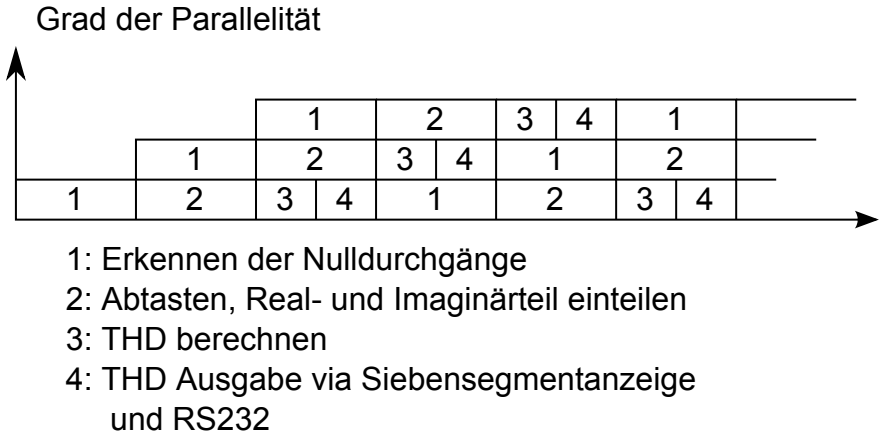


Bild 2.10: *Aufbau der Pipeline*

Bild 2.10 verdeutlicht den geplanten Ablauf. Bevor mit Abtastwerten gearbeitet werden kann, muss die Periodendauer bekannt sein. Nach Erkennen der Periodendauer wird diese in gleiche Abstände geteilt. Nach Ablauf eines jeden Zeitintervalls wird ein Wert genommen und dann für fünf Harmonische mit Sinus und Cosinus gewichtet. Die Verwendung von fünf Harmonischen hat sich in der Vergangenheit als brauchbar herausgestellt [14]. Ist diese Periode vorbei, wird aus Real und Imaginärteil in der nächsten Periode der THD berechnet. Dies geschieht relativ schnell, so dass jetzt noch die Siebensegmentanzeige aktualisiert werden kann und die Daten an einen PC via RS232 gesendet werden können.

Zu 2.3.4 Genauigkeit:
Es handelt sich bei dieser Implementation um den ersten Prototypen auf einer neuen Hardwareplattform. Deshalb werden Optimierungen hier noch nicht vorgenommen. Erst wenn der Funktionsnachweis erbracht worden ist, können Optimierun-

gen wie z.B. Erkenntnisse aus der Diplomarbeit von Koch [10] einfließen. Für die Eingangsvektoren (ADC Wert und Breite der Sinus-/Cosinus Wertetabelle) wurde identische Breite angenommen. Aus der Diplomarbeit von Jegenhorst [9] wurde die Vorauswahl von 64 Samples pro Periode übernommen. Deshalb wird hier wie folgt vorgegangen:

- 12 Bit ADC Wert mal 12 Bit Tabellenwert: 24 Bit
- 24 Bit über 64 Perioden Addiert: 24 Bit + 63 Bit = 87 Bit
  Dieser Wert ist zu hoch gegriffen, da der maximale, mögliche Wert durch zwei voll ausgesteuerte Signal in Matlab zu < 32 Bit ermittelt wurde. Mit Vorzeichen also 33 Bit.
- Bildung des Betrages: 32 Bit
- 32 Bit quadriert: 64 Bit
- Zehn 64 Bit Werte addiert: 73 Bit
  Auch dieser Wert ist zu hoch. Harmonische, die so stark vertreten sind, würden zu zusätzlichen Nulldurchgängen führen. Durch diese käme es schon vorher zu einem Fehler, so dass die THD Berechnung nicht beendet werden würde. Es genügen hier vier zusätzliche Bits Reserve: 68 Bit. Zehn Additionen ergeben sich aus Real- und Imaginärteil für fünf Harmonische.

Hat man diesen Punkt erreicht, kann man die Bitbreite abhängig von der gewünschten Bitbreite des Ergebnisses wählen. Bei einer 16 Bit Zahl zwischen 0 und 2 ist das erste Bit = 1, jedes weitere hat den halben Wert des ihm vorausgehenden (Q16 Darstellung). Das LSB hat also einen Wert von $2^{-15} = 3 \cdot 10^{-5}$. Dies ist ausreichend. Um ein 16 Bit Ergebnis zu bekommen, muss man aus einer 32 Bit Zahl die Wurzel ziehen. Also kann man sagen:

- 68 Bit durch 68 Bit geteilt: 32 Bit ausreichend genau.
- Wurzel aus 32 Bit: 16 Bit.
- 16 Bit auf $\frac{1}{\sqrt{2^{32}}}$ skaliert: 16 Bit

Zu 2.3.5 Abtastung:
Die Voraussetzung für die Berechnung der DFT eine konstante Menge Abtastwerde S pro Periode. Momentan wird mit 64 Abtastwerten pro Periode gearbeitet. Daraus ergibt sich, dass die Abtastfrequenz nicht konstant ist, sondern der Signalfrequenz angepasst wird (siehe Bild 2.11). Es wird eine zeitliche Abweichung von $\frac{1}{64}$ der Signalperiode zugelassen. Innerhalb dieser Toleranz ist die Eingangsperiode der prognostizierten ähnlich genug, um einen akzeptablen THD zu ergeben.

Ein weiterer wichtiger Faktor für die Wahl des Abtastverfahrens ist die Taktfrequenz des Systems. Der XILINX Spartan 3e kann mit bis zu 50 MHz arbeiten, aber

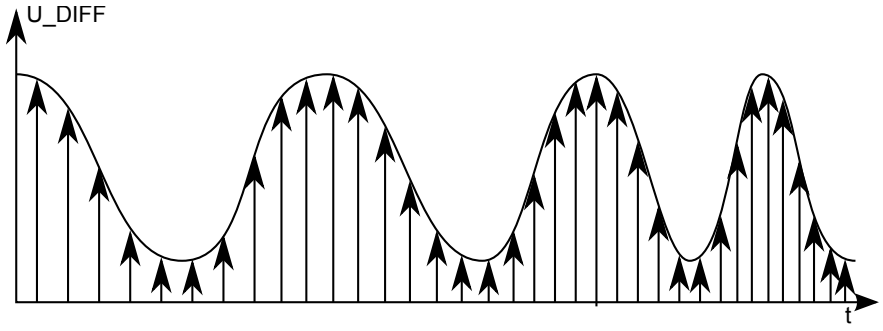


Bild 2.11: *Die Abtastfrequent passt sich der Signalfrequez an*

um es wurde festgelegt, den Takt auf $12,5MHz$ zu begrenzen. Dies ist der Praxis näher. $12,5MHz$ bedeutet, dass die Taktperiode $80ns$ lang ist. Die bekannten Rahmenbedingungen sind:

- 2 Takte Wertetabellenzugriff
- 12 Takte Multiplikation
- 1 Takt Addition
- 10-15 Takte um Steuersignale zu setzen und zurücknehmen, sowie Zustandswechsel
- 10 Multiplikationen, Wertetabellenzugriffe und Additionen pro Sample
- 64 Sample.

Daraus ergibt sich folgende Formel:

$$\frac{CLK}{(Takte \cdot 10 Bearbeitungen \cdot 64 Sample)} = F_{max} \tag{2.13}$$

Bei einer Taktfrequenz von $12,5MHz$ bedeutet das, dass die zu erwartende maximale Zahnfrequenz zwischen $651Hz$ und $781Hz$ liegt.

Es ist im Sinne der Prototypenentwicklung beschlossen worden, dies erst einmal hin zu nehmen. Die Entscheidung, ob das Abtastverfahren geändert, die Anzahl der Abtastwerte reduziert wird oder sogar der Systemtakt angehoben wird, ist damit für eine spätere Arbeit aufgehoben. In diese zukünftige Arbeit können dann die Erkenntnisse zur Aufwandsminimierung [10], welche Parallel zu dieser Arbeit gesammelt worden sind, einfließen.

3 Design

Bild 3.2 zeigt den Aufbau des Systems. Zuerst wird in 3.2 die Komponente `smart_comparator` besprochen, welche die Nulldurchgänge erkennt. Anschließend in 3.3 wird durch die Komponenten `dr_detection` die Drehrichtung erkannt. In den Abschnitten 3.4 und 3.5 werden die Komponenten `tl_sample` und `tl_calc` besprochen. Diese dienen zum Bestimmen der Harmonischen, respektive des THDs. Danach werden die Debug- und Analysemöglichkeiten durch die Komponenten `seven_seg_mux` und `seven_seg` in Abschnitt 3.6.1 beschrieben. Die Elemente `tl_miniuart` und `tl_adc` sind nicht vom Verfasser dieser Arbeit geschrieben worden, und werden deshalb nur kurz angesprochen.

3.1 Erläuterung zur Darstellung

Da VHDL sehr parallel arbeiten kann, musste das Format Flussdiagramm um die Darstellung der parallelen Ereignisse erweitert werden. In Bild 3.1 ist ein Beispiel wie dieses aussehen kann.
Der Anfang der parallelen Abläufe ist der breite Balken. Er ist der Transition aus dem Petrinetz nachempfunden. Zweig 1 und 2 sind dann Platzhalter für Code, der parallel ausgeführt wird. Ende der parallelen Anweisungen ist wieder die Transition.

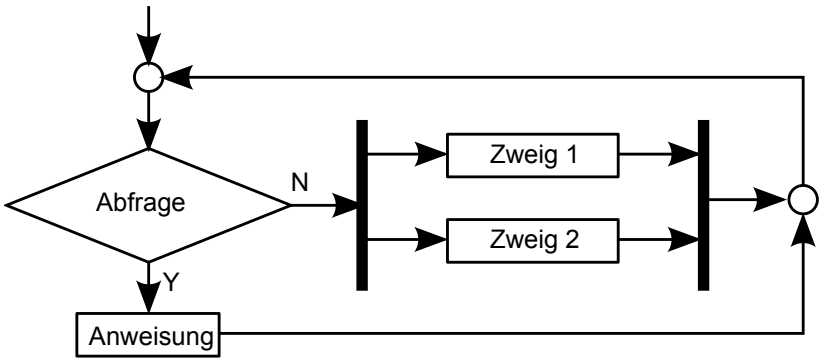


Bild 3.1: *Beispiel eines Flussdiagrams*

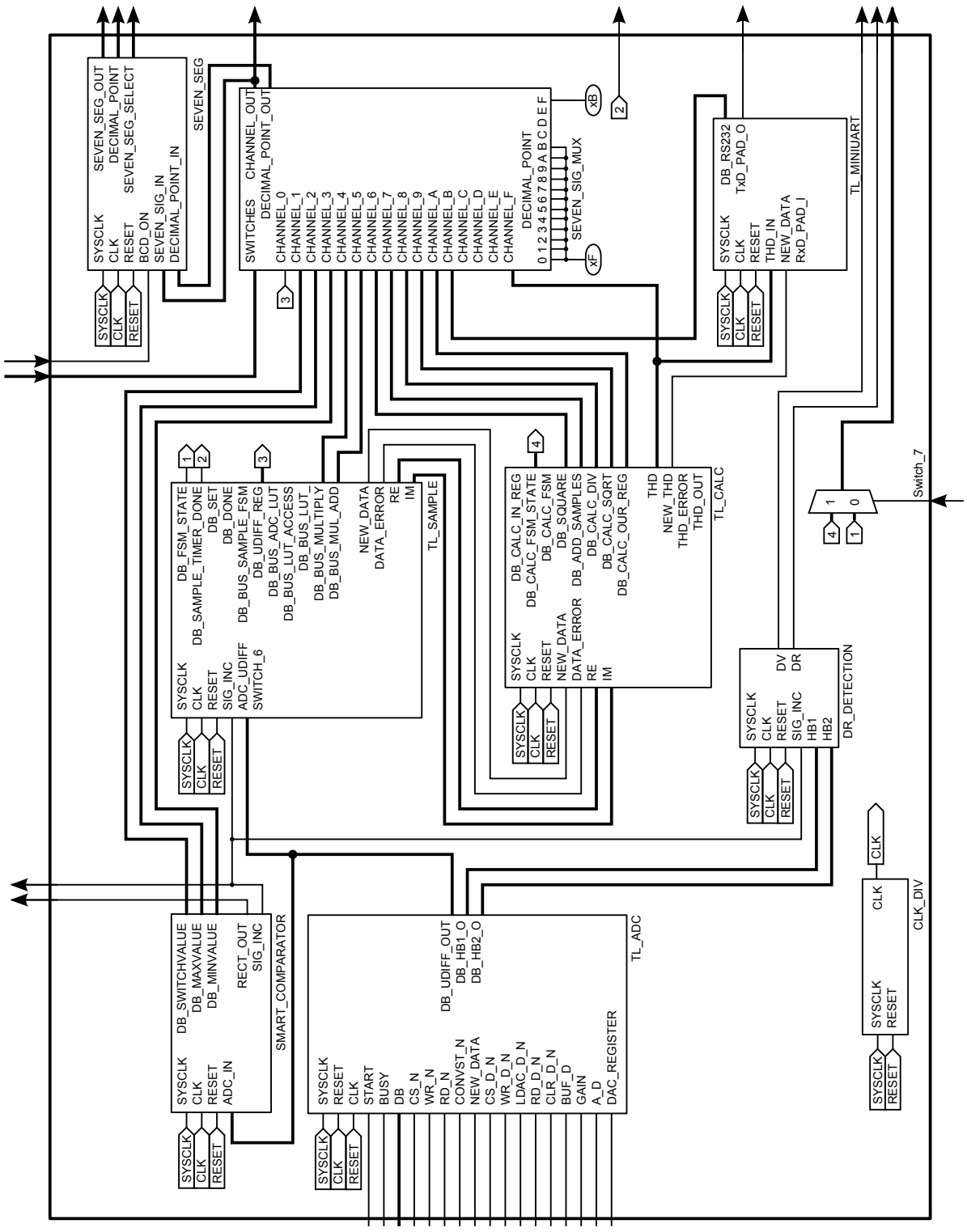


Bild 3.2: *Toplevelansicht des Systems*

3.2 Erkennung der Nulldurchgänge

Sensoren, welche noch nicht das von der Daimler AG spezifizierte VDA 4.0 Protokoll [4] umsetzen, nutzen die Nulldurchgangserkennung, um ein Rechtecksignal an die Bremssteuerung zu senden. Somit kann man die Nulldurchgangserkennung als Grundfunktion früher ABS Sensoren bezeichnen. Bei der Analyse der Harmonischen werden die Nulldurchgänge außerdem benötigt, um die Periode der Grundwelle des Sensorsignals zu messen. Dazu wird die steigende Flanke des Rechtecksignals durch einen Signalpuls markiert. Dieses Signal wird „Signal Increment“ (`SIG_INC`) genannt.

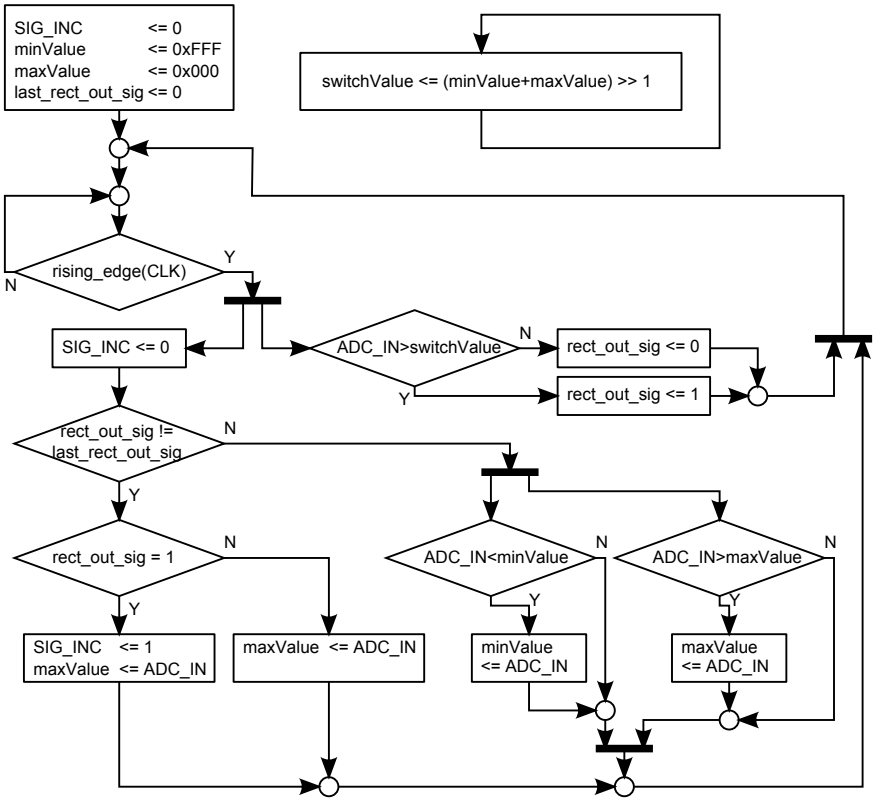


Bild 3.3: *Funktionsdiagram des Smartcomparators*

Um den Nulldurchgang zu erkennen wird das Prinzip eines Smartcomparators digital umgesetzt. Die in dieser Diplomarbeit gewählte Umsetzung unterscheidet sich aber von der bisher verwendeten Implementation. Bisher wird in einer Initialisierungsphase Minimum und Maximum des Eingangssignals bestimmt und anhand dieser Werte werden Schaltschwellen für eine Hysterese festgelegt. Diese Schaltschwellen werden mit einem DAC an einen Komparator angelegt und bei Schalten des Komparators gewechselt, so dass ein Schwingen verhindert wird [9].

Diese Implementation funktioniert auch mit unter abgetasteten Signalen, da das Schalten des Komparators einen Interrupt auslösen kann, welcher dann die Schaltschwelle wechselt. Diese Lösung wurde hier nicht gewählt, da die Schaltschwellen durch den möglichen hohen Grad der Parallelisierung permanent angepasst werden können. Wie auf Bild 3.4 gezeigt, wechselt die Schaltschwelle nicht zwischen zwei festen Werten. Sie wird wie folgt ermittelt:

$$switchValue = \frac{minValue + maxValue}{2} \tag{3.1}$$

Dies geschieht ungetaktet, der Vorgang ist deshalb in Flussdiagram 3.4 seperat dargestellt.

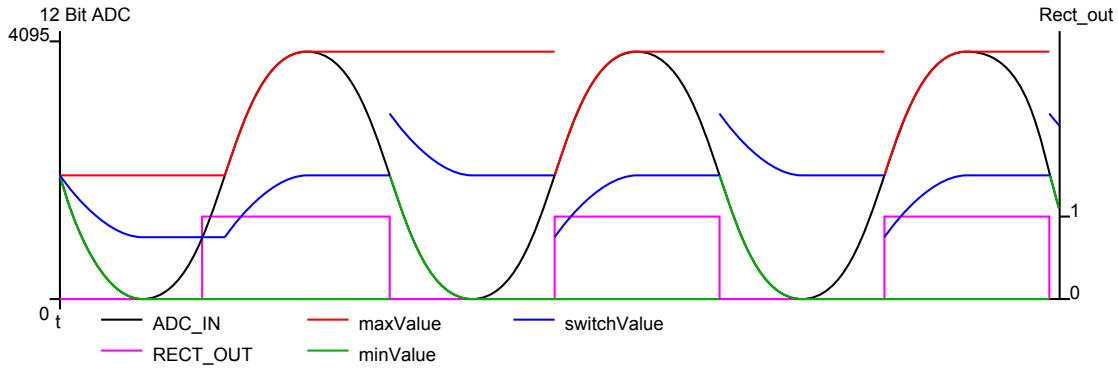


Bild 3.4: *Signale des Smartcomparators*

Es tritt ein Hysterese ähnlicher Effekt auf, aber mit dem Vorteil, dass der Schaltwert tatsächlich im Offset des Signals liegt. Da `maxValue` bei `rect_out_sig = 1` den Wert von `ADC_IN` erhält (`minValue` bei `rect_out_sig = 0`), springt nach dem Schalten die Schaltschwelle. Durch Annäherung des Signals an den Minimal- bzw. Maximalwerte verschiebt sich der Schaltwert wieder Richtung Offset, bis er am Extrempunkt erreicht wird.
Nach einem Reset hat `minValue` den Wert `0xFFF` und `maxValue` ist `0x000`, so dass `ADC_IN` bei Neustart offsetunabhängig erkannt wird. Ein Sonderfall, den es zu beachten gilt, ist die Anfangsphase, falls das Eingangssignal keine oder kaum Änderungen hat. Hier kann der Ausgang schwingen, da Minimal- und Maximalwert so nahe beieinander liegen, dass schon kleinste Änderungen die Schaltschwelle überschreiten. Dieses Problem wird durch eine Änderung in `ADC_IN` behoben, da mit dieser Änderung eine Differenz zwischen `maxValue` und `minValue` entsteht. Somit wird die Schaltschwelle aus dem instabilen Bereich geschoben (siehe Gl. 3.1). Der verwendete Schaltwert ist in der ersten Periode nicht korrekt, da das Maximum des Eingangs nie erreicht wurde. Deshalb schaltet die Komponente in diesem Fall zu früh.
Getestet wurde diese Implementierung mit einem Funktionsgenerator. Das Aus-

gangssignal war bei Frequenzen bis $110kHz$ sehr zuverlässig. Bei der maximal spezifizierten Zahnfrequenz von $2,5kHz$ konnten Signale mit einer Amplitude größer $45mV$ verarbeitet werden, nach einem Reset werden mindestens $320mV$ Amplitude benötigt.

3.3 Erkennung der Drehrichtung

Zur Erkennung der Drehrichtung wird das Summe-Differenzverfahren verwendet [16]. In dieser Komponente werden zwei Bits des digitalen Ausgabeprotokolls erzeugt, `DR` welches die Drehrichtung anzeigt, und `DV`, welches die Gültigkeit des Bits `DR` (Validität) anzeigt. Bei gültiger Drehrichtung ist `DV` gesetzt.

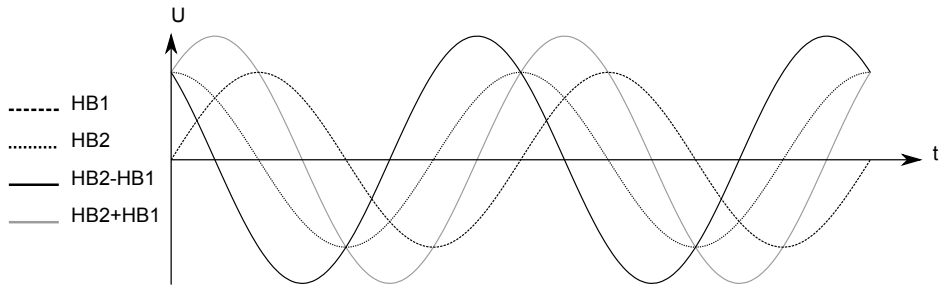


Bild 3.5: *Spannungen an den Ausgängen des Sensors, sowie deren Summe und Differenz*

In Bild 3.5 sind die Spannungen der beiden Halbbrücken des Sensors abgebildet, sowie die Summe der Halbbrückensignale (grau) und deren Differenz (schwarz). Die Drehrichtung wird, wie in Listing 3.1 als Pseudocode dargestellt, berechnet. Als Eingangsgrößen liegen an der Komponente die Halbbrückensignale des Sensors (`HB1` und `HB2`) sowie `SIG_INC` und `U_DIFF` an. Wenn Signal Increment gesetzt ist, also bei Nulldurchgang von `U_DIFF` wird die Summe der Halbbrücken mit dem Differenzsignal verglichen. Ist die Summe größer als `U_DIFF`, wird `DR` gesetzt [17].

```
1  int direction(int HB1, int HB2)
2  {
3    int sum, diff;
4
5    summe     = HB1 + HB2;
6    differenz = HB1 - HB2;
7
8    if differenz == 0 // die summe wird am Nulldurchgang ausgewertet
9    {                 // werte -1 und 1 repräsentieren vor und rück
10     if summe > 0
11       return 1;
```

```
12      else
13        return -1;
14    }
15    else
16      return 0;    //keine Aussage über Drehrichtung
17  }
```

Programmausdruck 3.1: *Summe-Differenzverfahren in Pseudocode*

3.4 Abtastung einer Periode

Das Differenzsignal wird zur Bestimmung der harmonischen Störung verwendet. An ihm wird auch die Periode gemessen und der Nulldurchgang erkannt.

Das Abtasten der Perioden geschieht in dem Modul Sample (siehe Bild 3.6). Das Modul besteht aus einem Zustandsautomaten, der die Abtastwerte mit Sinus und Cosinus bis zur fünften Harmonischen gewichtet und aufsummiert.

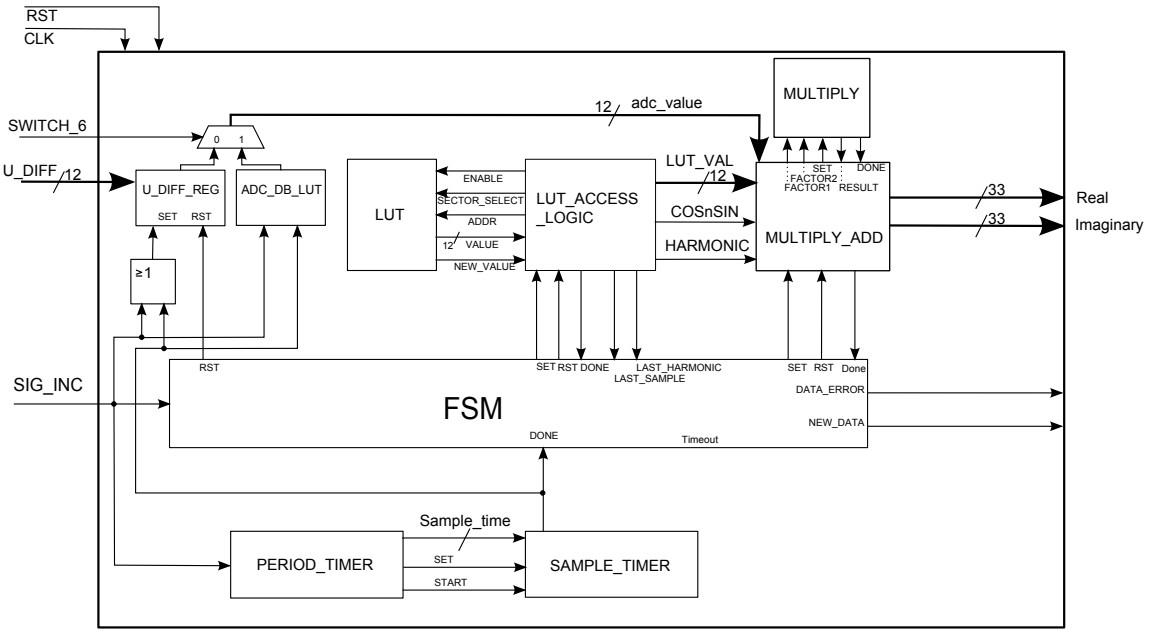


Bild 3.6: *Toplevel Ansicht des Moduls Sample*

3.4.1 Zeitmessung

Um 64 Abtastwerte pro Periode zu erhalten, werden zwei Timer benötigt. Der Periodentimer misst die Zeit zwischen zwei Nulldurchgängen und steuert den Sampletimer. Wenn der Periodentimer ein `SIG_INC` empfängt, dann stoppt er die Zeitmessung und setzt `SET_SAMPLE_TIMER`, so dass der Sampletimer die Zeit nehmen kann. Dann wird der Sampletimer gestartet, und der Periodentimer zurückgesetzt. Anschließend, wird der Periodentimer wieder gestartet.

Der Sampletimer bekommt vom Periodentimer einen 24 Bit Wert. Durch rechts shiften wird die Zeit der letzten Periode durch 64 geteilt. Dieser Wert wird als Startwert des Sampletimers abgelegt. Der Sampletimer zählt vom Startwert aus Richtung Null. Bei Erreichen setzt er `SAMPLE_TIMER_DONE`. Das Zählregister erhält wieder den Startwert.

3.4.2 Der Zustandsautomat

Der hier behandelte Zustandsautomat ist auf Bild 3.7 vereinfacht dargestellt. Der vollständige Automat ist in Anhang A.2, S. 77.
Der Zustandsautomat hat zwölf Zustände:
Nach POR werden zwei Zustände durchlaufen:

- FIRST_PERIOD
- WAIT_FOR_SIG_INC_LOW

Im regulärer Betrieb gibt es folgende Zustände:

- WAIT_FOR_SIG_INC_HIGH
- LUT_ENABLE_STATE
- LUT_DONE_STATE
- MUL_ADD_SET_STATE
- MUL_ADD_DONE_STATE
- WAIT_FOR_SAMPLE_TIMER
- NEW_DATA_STATE
- WAIT_FOR_SAMPLE_TIMER_AFTER_LAST_SAMPLE

Fehler werden in zwei Zuständen aufgefangen:

- ACCELERATION_ERROR

- DECELERATION_ERROR

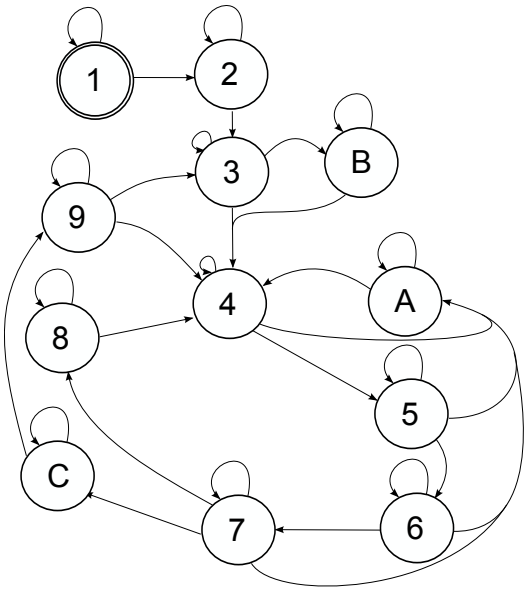


Bild 3.7: *Vereinfachter Aufbau der Sample FSM. Detaillierte Abbildung: Anhang A.2, S. 77*

Jetzt weredenwir die Zustände im einzelnen genauer betrachtet und auf ihre Übergangsbedingungen eingegangen. Sollte keine genannt werden, ist der Übergang bedingunslos nach einem Takt. Die Zustände werden hexadezimal gezählt, da diese Zahlen auf vier Bit codiert an einen Logicanalyzer übertragen werden können und man so Einblicke in den Ablauf des Automaten hat.
Anmerkung: Die Zustände sind nicht in aufsteigender Reihenfolge sortiert. Der hexadezimale Wert ist derselbe wie im Quellcode.

- 1 FIRST_PERIOD
  Der Startzustand. In ihm verweilt der Automat, bis ein Signal Increment vom Smart Komparator gesendet wird.
- 2 WAIT_FOR_SIG_INC_LOW
  Da der Zustandsautomat kombinatorisch ist, wird dieser Zustand benötigt, um zu garantieren, dass eine Periode abgewartet wird, bevor gesampelt wird. Der Zustand wird bei `SIG_INC` = 0 verlassen.
- 3 WAIT_FOR_SIG_INC_HIGH
  Erster Zustand eines Abtastzyklusses. In ihm wird der Look-Up-Table über ein Reset Signal zurückgesetzt. Regulär verlassen wird der Zustand durch ein Signal Increment, welches zum Übergang zu LUT_ENABLE_STATE führt.

- 4 LUT_ENABLE_STATE
  Die Komponente `look_up_access_logic` bekommt ein `Enable` Signal. Wenn die Komponente ein `DONE` meldet, wird der Zustand verlassen.
- 5 LUT_DONE_STATE
  Hier wird auf die Rücknahme des `DONE` Signals der Komponente Mul_Add gewartet.
- 6 MUL_ADD_SET_STATE
  Die Komponenten zum Multiplizieren des Samples mit dem Wert des LUT sowie zum Aufaddieren der Produkte wird angesprochen.
- 7 MUL_ADD_DONE_STATE
  Hier wird auf die Rücknahme des `DONE` Signals gewartet.
- 8 WAIT_FOR_SAMPLE_TIMER
  Nach Ende der Berechnungen wird auf das Ablaufen des Sample Timers gewartet, damit die Berechnung mit dem nächsten Abtastwert wiederholt werden kann.
- C NEW_DATA_STATE
  Dieser Zustand dauert nur einen Takt. Er wird eingenommen, um ein `NEW_DATA` Signal an die CALC FSM zu senden, bevor in seinem Folgezustand die Register zurückgesetzt werden.
- 9 WAIT_FOR_SAMPLE_TIMER_AFTER_LAST_SAMPLE
  Alle Register werden zurückgesetzt. Die Statemachine hat eine zeitliche Toleranz von $\frac{1}{64}$ Periode. Diese Zeit wird hier abgewartet, dann wird in den Zustand WAIT_FOR_SIG_INC_HIGH gewechselt. Nötig ist diese Toleranz, da es erstens zu einer Geschwindigkeitsänderung kommen kann, welche nicht jede THD-Messung mit einem Fehler abbrechen soll, und zweitens, da der Sampletimer die Periode in 64 gleiche Abschnitte einteilt. Dies geschieht ohne Runden, und es wird nur mit ganzen Zahlen gearbeitet, weil diese Takten entsprechen. So gibt es in der Summe der Samplezeiten eine Differenz zur tatsächlichen Periodenzeit, welche durch zeitliche Toleranz kompensiert werden muss.
- A ACCELERATION_ERROR
  Dieser Zustand wird nur einen Takt lang eingenommen, um den Übergang zur THD Berechnung so kurz wie möglich zu gestalten. Alle Register werden gelöscht und es wird ohne Bedingung in LUT_ENABLE_STATE übergegangen.
- B DECELERATION_ERROR
  Dieser Zustand ist nur aus WAIT_FOR_SIG_INC_HIGH erreichbar. Er wird

eingenommen, wenn in WAIT_FOR_SIG_INC_HIGH der Sample Timer abläuft. Dies würde eine Geschwindigkeitsänderung um $\frac{1}{64}$ bedeuten, wodurch die berechneten Werte als nicht aussagekräftig eingestuft und deshalb gelöscht werden.

3.4.3 Der Sinus/Cosinus Look-Up-Table

Der LUT besteht aus zwei Komponenten: der Tabelle selber und der Zugriffslogik, die die Eingänge der Tabelle so manipuliert, dass der benötigte Speicher minimal ist.

3.4.3.1 Zugriffslogik des LUT

Die Tabelle soll 64 Werte für jeweils Sinus und Cosinus enthalten, so dass es für jeden der 64 Abtastwerte pro Periode einen Wert in der Tabelle gibt. Um die Wertetabelle klein zu halten, werden Symmetrien der Funktion genutzt. Das erste Viertel der Sinusperiode, der Bereich zwischen 0° und 90°, genügt um Sinus und Cosinus zu definieren. Der Wertebereich zwischen 90° und 180° ist spiegelsymmetrisch zum ersten Abschnitt, und kann durch Manipulation der Eingangsadresse erzeugt werden, ohne mehr Speicher zu belegen. Der Bereich zwischen 180° und 360° entspricht dem Abschnitt zwischen 0° und 180° mit negativen Vorzeichen. Man sieht also, dass nur ein Viertel der Sinusperiode in der Wertetabelle abgelegt werden muss, um die Funktion nachzubilden. Da der Cosinus dem Sinus mit einer Phasenverschiebung von 90° entspricht, benötigt man nur eine Tabelle, deren Zugriffsadresse (die Eingangswerte) manipuliert werden, um auch den Cosinus ausgeben zu können. Bild 3.8 zeigt den schematischen Aufbau der Zugriffslogik.

Die Werte der Tabelle werden mit vier Addressbits abgerufen. Diese Bits liegen direkt am LUT an. Das zusätzliche Bit am Eingang des LUT wählt aus, ob ein Ergebnis aus dem Bereich $0° \leq \varphi < 90°$ oder $90° \leq \varphi < 180°$ erwartet wird. Es wird durch das Bilden einer Summe aus `ADDR(4,5)` und `COS` entschieden, in welchem Sektor der Funktion der gewünschte Wert ist. `ADDR(4) XOR COS` wählt das erste oder zweite Viertel aus. `ADDR(5) XOR (ADDR(4) AND COS)` legt fest, ob der Ausgang des LUT negiert wird. Das Ergebnis dieser logischen Verknüpfung muss einen Takt verzögert werden, da die Komponente auf das Ergebnis der Wertetabelle warten muss, welche einen Takt benötigt. Jeder Sinus- oder Cosinusabruf benötigt also zwei Takte.

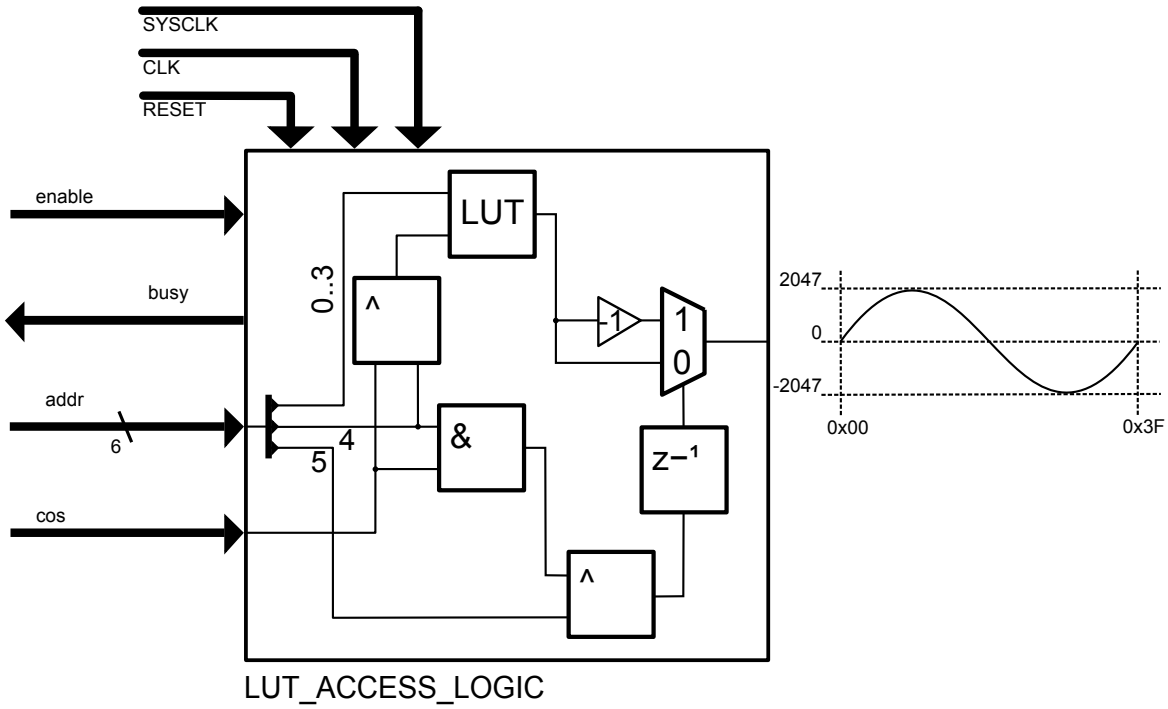


Bild 3.8: *Aufbau der Komponente LUT_ACCESS_LOGIC*

3.4.3.2 Aufbau des LUT

Die Komponente, die die Wertetabelle beinhaltet, hat zwei Dateneingänge: vier Bit Zugriffsadresse und ein Bit „Special Case“ (`SC`, siehe Bild 3.9).

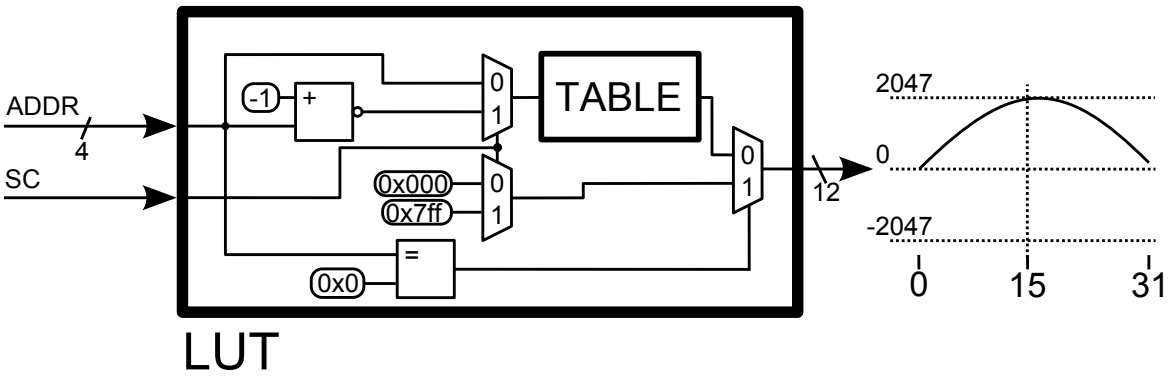


Bild 3.9: *Vereinfachter Aufbau der Komponente LUT*

In der Tabelle sind 15 Werte abgelegt. Durch

$$f(x) = sin(x \cdot \frac{2\pi}{64}) \cdot (2^{11} - 1), 1 \leq x \leq 15 \tag{3.2}$$

wird die Funktion in 64 Schritte unterteilt, von denen 15 in der Tabelle (siehe 3.1)gespeichert werden.

ADDR	Wert
0x1	201
0x2	399
0x3	594
0x4	783
0x5	965
0x6	1137
0x7	1299
0x8	1447
0x9	1582
0xA	1702
0xB	1805
0xC	1891
0xD	1959
0xE	2008
0xF	2037

Tabelle 3.1: *Inhalt der Wertetabelle*

In der Wertetabelle fehlen Minimal- und Maximalwert der Funktion, was daran liegt, dass der Adresswert `0x0` doppelt belegt ist. Die Ausgabe in diesem Fall ist abhängig von dem Eingangsbit `SC`, welches dann zwischen 0 und 2047 schaltet.

3.4.4 Sequentielle Multiplikation

Wie auf Bild 3.10 gezeigt, ist die sequentielle Multiplikation eine Reihe von Additionen mit Schiebeoperationen. Benötigt werden in diesem Fall zwölf Takte, um die Multiplikation abzuschließen, da die Faktoren zwölf Bit haben. Ein Index läuft dabei von null bis zwölf und dient einem Faktor als Bitindex und dem anderen als Schiebeindex. Ist das vom Index maskierte Bit im ersten Faktor gesetzt, wird der zweite Faktor um „Index“ Werte nach links geschoben. Diese geschobenen Werte werden addiert. Nach zwölf Takten ist die Summe das Produkt der Faktoren.

3.4.5 Aufsummieren der Produkte der sequentiellen Multiplikation

Die Komponente besteht aus zwei Prozessen. Einer ist getaktet und addiert die Produkte, und einer ist kombinatorisch und behandelt das Vorzeichen vor und nach

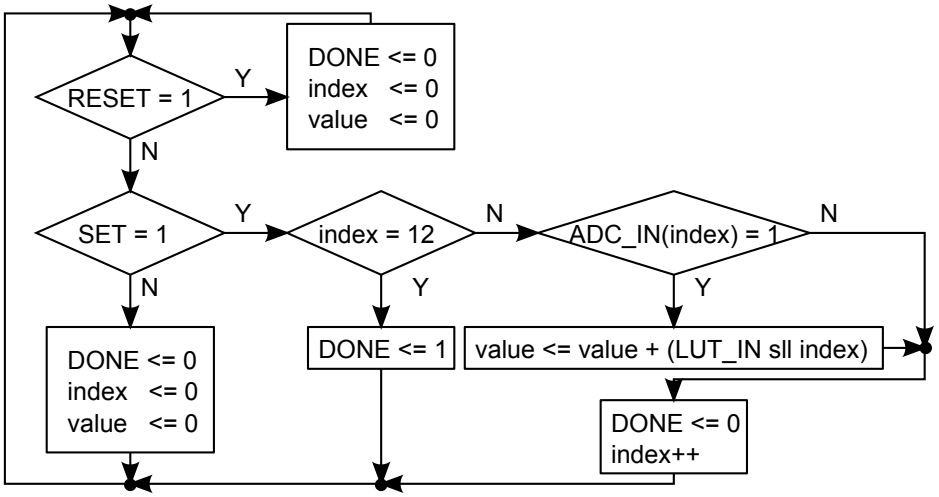


Bild 3.10: *Datenfluss des sequentiellen Multiplizierers*

der Multiplikation. Der ADC Wert ist immer positiv, der LUT Wert kann auch negativ sein. Deshalb achtet der Prozess `signed_unsigned_signed` auf das MSB des LUT Wertes und negiert, wenn es gesetzt ist, einen Eingang und den Ausgang des Multiplizierers (siehe Bild 3.11). Nötig ist dies eigentlich nicht, sofern das Schieben in der Multiplikation vorzeichenrichtig durchgeführt wird. Die dann verbleibende Operation (die Addition) verhält sich vorzeichenrichtig. Dafür hätte dann der Additionsoperator überladen werden müssen, um *signed* und *unsigned* Daten verarbeiten zu können. Der Aufwand wäre dem hiesigen ähnlich.

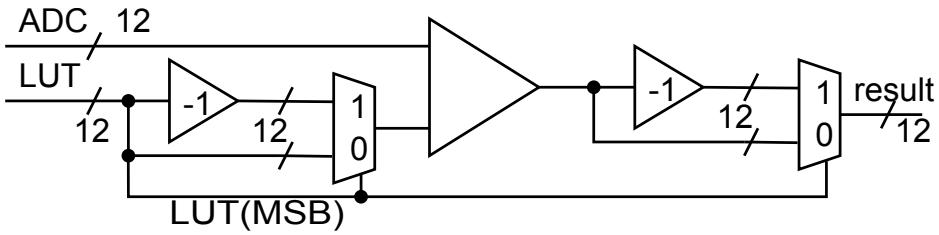


Bild 3.11: *Multiplizieren mit Vorzeichen*

Der Addierer reagiert auf das `DONE` Signal des Multiplizierers und addiert das berechnete Produkt zu der Summe hinzu. Am Ende der Periode sind die Sinus und Cosinus Anteile der ersten fünf Harmonischen im Ausgangsregister der Komponente. Sollte auf Grund einer falschen Annhme der Fall eintreten, dass die Summe größer 32 Bit wird, ist vorgesort. Der Summenvektor ist 33 Bit breit. Wenn das MSB (nach dem Vorzeichen) sich vom Vorzeichen unterscheidet, ist gilt es als gesetzt. Dann werden die Summen aller Harmonischen vorzeichenrichtig um ein Bit nach rechts geschoben.

3.5 Abschätzung des THD

Das Aufteilen der Periode in Real- und Imaginäranteil ist abgeschlossen. Bevor dieser Prozess erneut mit der nächsten Periode durchgeführt wird, wird das Ergebnis an die Komponente `CALC_THD` (Siehe Bild 3.12) weitergegeben. Hier wird der THD berechnet. In dieser Implementation geht die Berechnung weiter als eigentlich nötig; es kann nach Abschluss der Berechnung des Quotienten aus Oberwellen und Gesamtsignal schon eine Aussage über die Zuverlässigkeit des Eingangssignals gemacht werden. Die weitere Signalverarbeitung ist notwendig, um die Darstellung auf der Siebensegmentanzeige zu ermöglichen. Dort sollen von Menschen problemlos interpretierbare Werte abgebildet werden. Deshalb muss das Ergebnis der Division noch radiziert und skaliert werden.

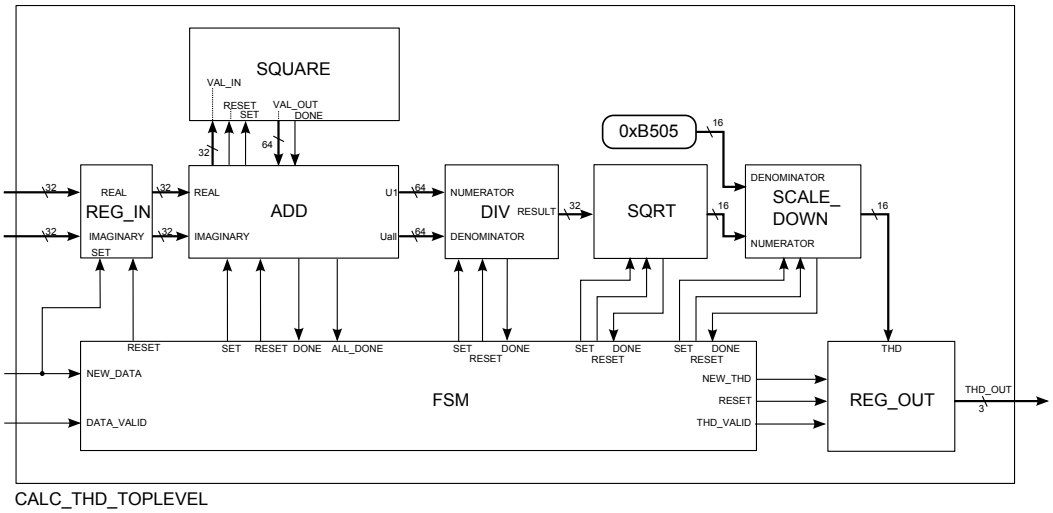


Bild 3.12: *Toplevel Ansicht der Komponente* `CALC_THD`

3.5.1 Der Datenpfad

Der Datenpfad ist die Umsetzung der Gleichung

$$THD = \sqrt{\frac{H_2^2 + H_3^2 + H_4^2 + H_5^2}{H_1^2 + H_2^2 + H_3^2 + H_4^2 + H_5^2}} \tag{3.3}$$

Man sieht also, dass folgende Komponenten benötigt werden:

- Addierer
- Quadrierer

- Dividierer
- Radizierer

3.5.1.1 Sequentielles Quadrieren mit Addition der Werte

Die Addition und die Quadratur sind, da die Quadratur eine Multiplikation mit identischen Faktoren ist, Elemente, deren Algorithmen hier nicht näher beschrieben werden müssen, da sie in 3.4.5 und 3.4.4 schon diskutiert wurden.

3.5.1.2 Sequentielle Division

Der detaillierte Aufbau der Komponente ist in Anhang A, Seite 76, einzusehen. Die wichtigsten Elemente sind in Bild 3.13

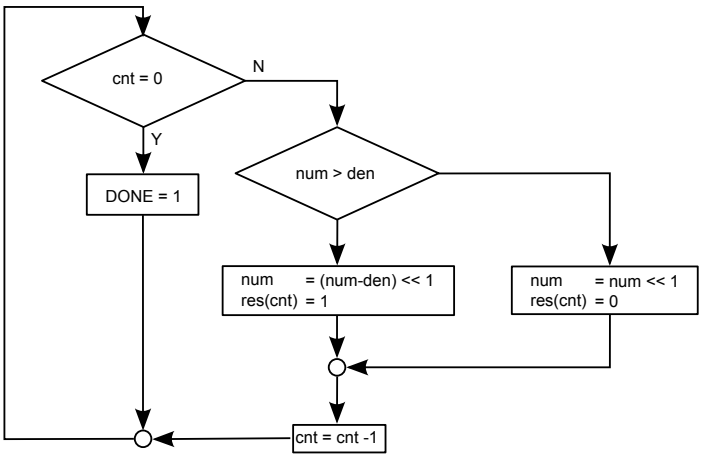


Bild 3.13: *Skizze der Komponente calc_div*

Die Division zweier gleichlanger Zahlen dauert immer einen Takt pro Bit Eingangsvektor. Deshalb wird ein Durchlaufzähler `cnt` verwendet, der das Ergebnis von MSB nach LSB einträgt, und mit Erreichen des Wertes Null den Abschluss der Berechnung indiziert. Es wird eingangs geprüft, ob der Zähler (Nummerator `num`) größer ist als der Nenner (Denominator `den`). Wenn ja, wird vom Zähler der Nenner subtrahiert und das Ergebnis mit zwei multipliziert.

Hier ist die Berechnung des THDs abgeschlossen. Der Ausgang der Division genügt zum Codieren auf drei Bit im Ausgabeprotokoll. Jede weiterführende Berechnung dient nur zur Visualisierung der Ergebnisse auf der Siebensegmentanzeige und zur Verkürzung des Ergebnisvektors von 32 auf 16 Bit. Der kürzere Vektor ermöglicht es, mehr Werte via RS232 zu senden.

3.5.1.3 Ziehen der Quadratwurzel

Die Quadratwurzel wird nicht durch eine Wertetabelle gezogen, sondern ausgerechnet. Verwendet wird *„A New Non-Restoring Square Root Algorithm“* [11], welcher es ermöglicht, aus einer $2N$-Bit Zahl innerhalb von N Takten die Wurzel zu ziehen. Bild 3.14 zeigt den schematischen Aufbau der Komponente.

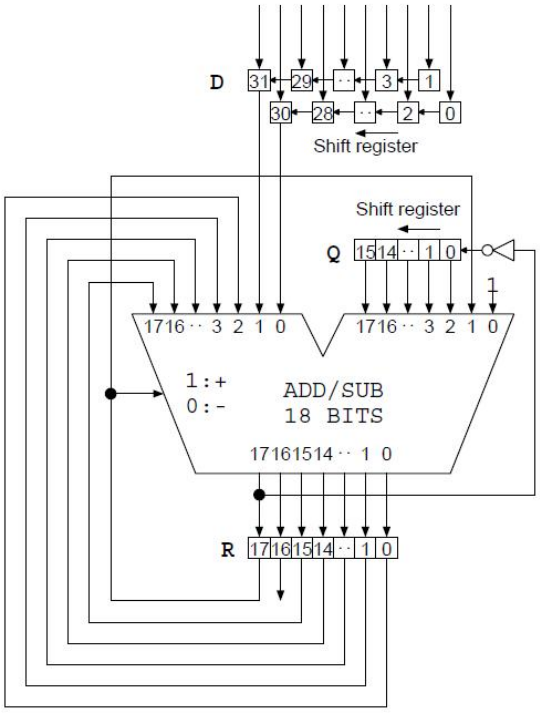


Bild 3.14: *Skizze der Komponente SQRT [26]*

Es wird Anfangs die Annahme getroffen, dass es für zwei Eingangsbits ein Ergebnisbit gibt, weshalb der Ergebnisvektor halb so lang ist wie der Eingangsvektor. Somit ist die Wurzel aus 32-Bit D: 16-Bit Q.
Es gibt also den Vektor $D = D_{31}D_{30}\cdots D_1D_0$ sowie $Q = Q_{15}Q_{14}\cdots Q_1Q_0$. Zusätzlich benötigt man einen Vektor R, in dem der Rest gespeichert ist, $R = D - Q^2$. Die nötige Anzahl an Bits für R läßt sich wie folgt ermitteln:

$$D = R + Q^2 < (Q+1)^2 \Rightarrow R < 2Q+1 \Rightarrow R \leq 2Q \tag{3.4}$$

Der Rest R hat also maximal ein Bit mehr als das Ergebnis Q.

Nachdem die Vektorlängen bestimmt und ihre Funktionen definiert sind, wird jetzt ein Blick auf die Berechnung der Wurzel geworfen. Anfangs gilt $R = 0$, $Q = 0$ und $i = 15$.

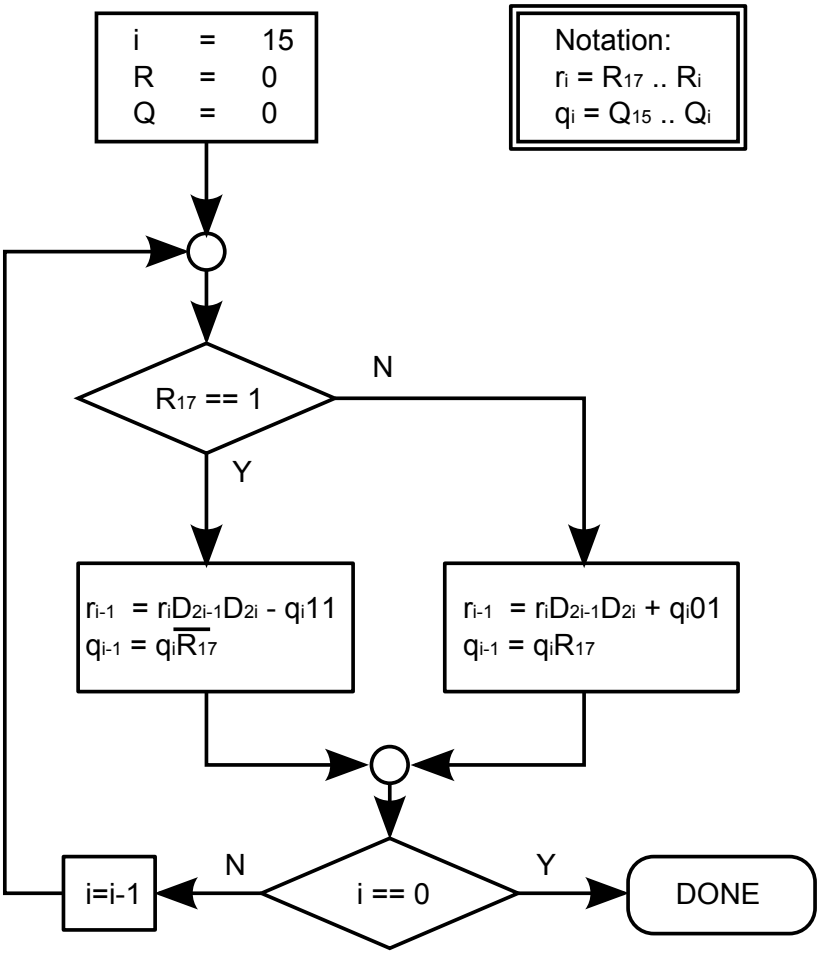


Bild 3.15: *Flussdiagramm des „Non-Restoring Square Root Algorithm“*

Bild 3.15 zeigt das Flussdiagramm des Algorithmusses. Für Teilvektoren gilt die in diesem Bild gezeigte Notation.
An den Vektor R werden immer die beiden aktuellen MSBs von D angehängt. Die dann folgende Berechnung ist Abhängig von r_i.

$$r_{i-1} = \begin{cases} r_i D_{2i+1} D_{2i} - q_i 01, & \text{wenn } r_i \geq 0 \\ r_i D_{2i+1} D_{2i} + q_i 11, & \text{wenn } r_i < 0 \end{cases} \tag{3.5}$$

Das aus Gleichung 3.5 hervorgegangenen Ergebnis bedingt den Wert von q_{i-1}.

$$q_{i-1} = \begin{cases} q_i 1, & \text{wenn } r_{i-1} \geq 0 \\ q_i 1, & \text{wenn } r_{i-1} < 0 \end{cases} \tag{3.6}$$

Vereinfacht beschrieben, handelt es sich hier um eine Division bei der Ergebnis und Divisor identisch sind und Ermittelt werden. Dabei wird eine Stelle (ein Bit) des Ergebnisses als gesetzt angenommen. Ist diese Annahme richtig, ist der Rest positiv, das Ergbnisbit wird gesetzt. Wenn sie Falsch ist, wird das Ergebnisbit nicht gesetzt. Dann wird sich durch Gleichung 3.5 dem Wert aus anderer Richtung genähert.

3.5.1.4 Skalierung des Ergebnisses

Für das Ergebnis der Division, r, gilt immer $0 < r \leq 1$. Der Grund für die Skaliergung soll am Beispiel im Dezimalsystem verdeutlicht werden:

$$\sqrt{0,4} = 0,6325 \tag{3.7}$$

oder auch

$$\sqrt{\frac{4}{10}} = \frac{2}{\sqrt{10}} = 0,6325 \tag{3.8}$$

Es wird deutlich, dass das Ergebnis wie eine ganze Zahl behandelt werden kann, wenn man es danach entsprechend skaliert. Da der Eingangsvektor der Wurzel 32-Bit hat, muss das Ergebnis auf $\sqrt{2^{31}}$ skaliert werden. Dazu wird wieder eine Division verwendet, also dieselbe Komponente wie in 3.5.1.2, mit einem konstanten Nenner mit dem Wert `x"B505"`. Dieser Wert entspricht der hexadezimalen Darstellung von $\sqrt{2^{31}}$

3.5.1.5 Quantisierung auf drei Bits

Hier wird das Ergebnis auf drei LEDs ausgegeben. Sie repräsentieren die drei Bits im digitalen Ausgabeprotokoll, auf denen der THD codiert werden soll. Ein- und Ausgaben sind in Tabelle 3.2 zu finden. Die Werte in dieser Tabelle sind nach Gefühl gesetzt, da es in dem Projekt ESZ-ABS noch keine Standardisierung dazu gibt.

Es wurde versucht, die THDs nahe dem optimalen höher aufzulösen. Dadurch sollen Abweichungen schneller auffallen.
Diese Implementation erzeugt die LED Kombination aus dem skalierten und radizierten Ergebnis. Dies ist nicht nötig. Die LED Ansteuerung wurde in das Ausgaberegister der Einfachheit halber eingebunden. Das Ausgaberegister wird benötigt, um den THD für die Siebensegmentanzeige und den Wishbone Bus zur RS232 Ausgabe bereit zu halten.

LED 5..7	THD[%]	THD[hex]
111	>29	>251E
110	>21	>1AE1
101	>14	>11EB
100	>8	>0A3D
011	>4	>051D
010	>2	>028F
001	>1	>0147
000	<1	≤0147

Tabelle 3.2: *Ein- und Ausgabe von* `CALC_THD_OUT_REG`

3.5.2 Der Steuerpfad

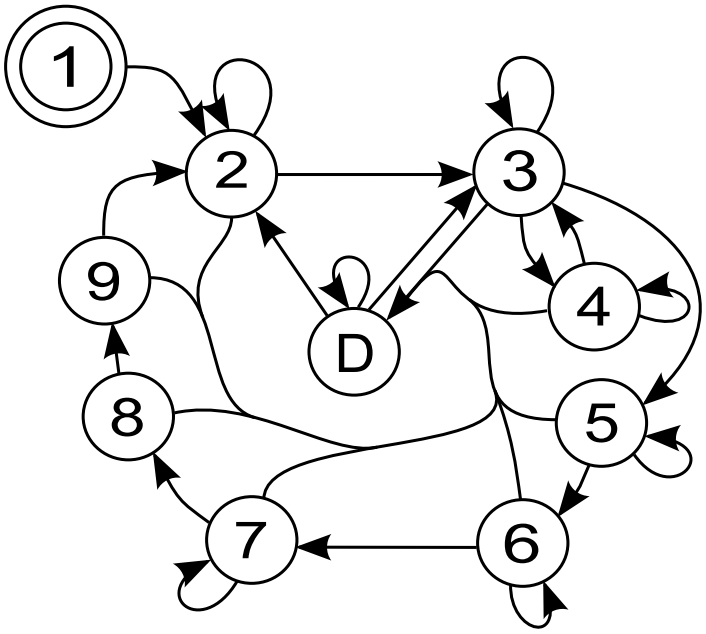


Bild 3.16: *Undetaillierter Aufbau der CALC_FSM*

Der Zustandsautomat (3.16) hat zehn Zustände:

- 1 `POR`
- 2 `IDLE`
- 3 `SET_ADD_STATE`
- 4 `ADD_DONE_STATE`
- 5 `ADD_ALL_DONE_STATE`

- 6 SET_CALC_DIV_STATE
- 7 SET_CALC_SQRT_STATE
- 8 SET_SCALE_DOWN_STATE
- 9 SUCCESS
- D FAIL

Die Namen der Zustände zusammen mit der Abbildung des Automaten geben einen ersten Eindruck über den Ablauf. Eine genaue Betrachtung der Vorgänge in den einzelnen Zuständen ist trotzdem nötig, detaillierte Abbildung des Zustandsautomaten im Anhang A.3.

- **1 POR**
  Power On Reset sendet ein Resetsignal an alle Komponenten in TL_CALC. Der Zustand wird ohne Bedingung verlassen.
- **2 IDLE**
  Setzt dieselben Signale wie POR, nur ohne Reset auf das Ein- und Ausgaberegister. Kann durch zwei Signale verlassen werden: NEW_DATA Puls geht in 3 über, ERROR_IN in D. Ein gesetztes Errorsignal führt in jedem Zustand zu einen Übergang zu D, es wird deshalb nur hier einmal aufgeführt.
- **3 SET_ADD_STATE**
  Setzt SET_ADD Signal, wodurch die Eingangssignale quadriert und aufsummiert werden. Geht bei gesetztem ADD_DONE in ADD_DONE_STATE (4) über. Ist zusätzlich noch ADD_ALL_DONE gesetzt, ist der folgende Zustand ADD_ALL_DONE_STATE (5).
- **4 ADD_DONE_STATE**
  Wartet auf das Zurücknehmen von ADD_DONE. Folgezustand ist SET_ADD_STATE (3).
- **5 ADD_ALL_DONE_STATE**
  Nach dem Zähler und Nenner für die Division gebildet wurden, wartet dieser Zustand auf das Zurücknehmen von den DONE Signalen. Ist das geschehen, wird in SET_CALC_DIV_STATE (6) übergegangen.
- **6 SET_CALC_DIV_STATE**
  Mit dem Setzen von SET_CALC_DIV werden Zähler und Nenner in die Eingangsregister der Komponente CALC_DIV geladen. Das Signal CALC_DIV_DONE führt zum Übergang in den Zustand SET_CALC_SQRT_STATE (7). Der Zustand wird durch einen SET_CALC_DIV_DONE Puls verlassen; es folgt SET_CALC_SQRT (8). Hier könnte die Berechnung des THD abgebrochen werden, da das Ergebnis der Division ausreicht, um

den THD zu quantisieren. Zur Ausgabe auf der Siebensegmentanzeige ist der Wert aber nicht geeignet, da man kein instinktives Gefühl für seine Aussagekraft hat. Dies ist bei einem Wert in Prozent anders, die Berechnungen werden also fortgesetzt.

- **7 SET_CALC_SQRT_STATE**
  SET_CALC_SQRT wird gesetzt, wird CALC_SQRT_DONE von der angesprochenen Komponente gesetzt, wechselt der Zustand zu SET_SCALE_DOWN_STATE (8).
- **8 SET_SCALE_DOWN_STATE**
  SET_SCALE_SOWN wird gesetzt, wenn SCALE_DOWN_DONE geantwortet wird, Übergang zu SUCCESS (9).
- **9 SUCCESS**
  NEW_THD wird gesetzt, es wird bedingungslos nach IDLE übergegangen.
- **D FAIL**
  Dieser Zustand ist die Alternative zu IDLE, mit dem Unterschied, dass nicht nur die Komponenten zurückgesetzt werden, sondern auch noch das Signal THD_ERROR gesetzt ist.

3.6 Schnittstellen für Statusanzeigen und Auswertung mit Matlab

Es gibt drei Möglichkeiten für den Entwickler, Daten aus der Laufzeit zu sichten:

- die Siebensegmentanzeige,
- den Logic Analyzer und
- Matlab via der RS232 Schnittstelle.

Zwei dieser Möglichkeiten wurden vom Autor implementiert, sie werden ausführlich besprochen. Die RS232 Schnittstelle wurde der Arbeit von Herrn Arvidsson entnommen [1] und deshalb nur kurz angesprochen.

3.6.1 Ausgaben auf der Siebensegmentanzeige

Die Dipschalter 0 bis 3 sind die Steuereingänge eines Multiplexers, welcher 15 Bit auf der Siebensegmentanzeige darstellt, und davon die mittleren 8 Bit an den Anschluss `JB1` gibt. Wie auf Bild 3.17 zu sehen ist, hat der Multiplexer 32 Eingänge, obwohl nur 16 Möglichkeiten einstellbar sind. Es werden nicht nur die Signale auf einen Ausgang geschaltet, sondern auch die dazugörige Ansteuerung für den Dezimalpunkt. Diese Werte kommen nicht aus einem variablen Signal, sondern werden i.d.R. als Konstante auf den entsprechenden Port gegeben.

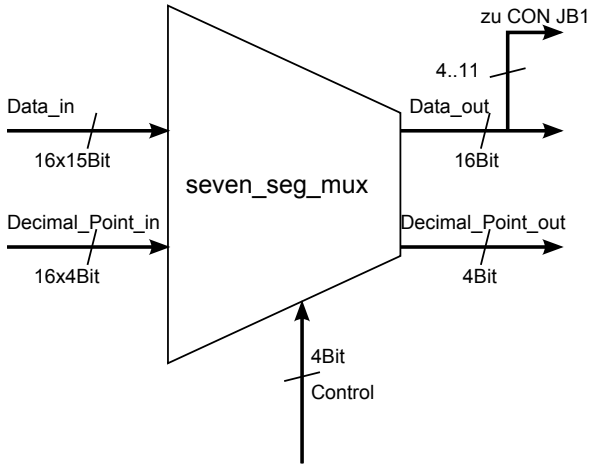


Bild 3.17: *Schematische Darstellung des Siebensegment Multiplexers*

Dieser Ausgang des Multiplexers liegt dann an der Siebensegmentanzeige an. Diese bietet einem die Möglichkeit, die Werte sowohl als Hexadezimalewert, als auch in Prozent von `FFFF` einzusehen. Der entsprechende Schalter ist `SWITCH_5`. Die Umwandlung ist relativ grob, da die Displayausgabe nur ein Gefühl für die Größenordnung des THD geben soll. Es gibt vier verlgeichsgrößen (siehe Tabelle 3.4), eine pro Anzeige, um die der Eingansvektor dekrementiert werden kann. Es werden von der Eingangsgröße sukzessiv Werte abgezogen, bis der Minuend kleiner ist als der aktuelle Subtrahend. Bei jeder Subtraktion wird ein Indexzähler inkrementiert. Es gibt einen Indexzähler pro Siebensegmentanzeige (also vier), so dass jeder Indexzähler einen Wert hat, der auf der Siebensegmentanzeige dargestellt werden kann. Es handelt sich hierbei, wie bereits erwähnt, um eine grobe Quantisierung, die keine Rücksicht auf Nachkommastellen der Minuenden nimmt. Es kann deshalb passieren, dass der Wert `A` angezeigt wird.

SWITCHES_0_3	Ausgang MUX
0000	Udiff_reg
0001	Switchvalue (Smart_comp)
0010	MaxValue (Smart_comp)
0011	MinValue (Smart_comp)
0100	DB_BUS_MULTIPLY
0101	DB_BUS_MUL_ADD
0110	DB_SQR
0111	DB_ADD_SAMPLES
1000	DB_CALC_DIV
1001	DB_SQRT
1010	DB_SCALE_DOWN
1011	DB_RS232
1100	
1101	
1110	
1111	THD

Tabelle 3.3: *Ausgang des Multiplexers bei Schalterstellung 0..3*

% THD	Minuend
10	0CCC
1	0147
0,1	0020
0,01	0003

Tabelle 3.4: *Quantisierung für die Siebensegmentanzeige*

3.6.2 Debuggen zur Laufzeit mit dem Logic Analyzer

Wie in 3.6.1 erwähnt, werden die mittleren acht Bits auf `CON JB1` gegeben. Zusätzlich belegte Schalter sind:

`SWITCH_6`: Dieser schaltet zwischen adc_udiff_reg und adc_lut_value um. Bei der Einstellung adc_udiff_reg arbeitet der Prototyp mit Abtastwerten, es kann also der THD der aktuellen Eingangsfunktion ermittelt werden. Die Wahl von adc_lut_value bedeutet, dass die Abtastwerte aus einer Wertetabelle stammen, also keine echten Samples sind. Diese Option wurde zum Debuggen eingebaut, da man so rekonstruierbare Werte erhält, die vorausberechnet werden können, und anschließend mit den tatsächlichen Werten Bit für Bit vergleichen.

`SWITCH_7` schaltet einen 4 Bit Bus um. Auf dem Anschluss `CON JA1` können dann entweder die Zustände der `SAMPLE_FSM` oder `CALC_FSM` eingesehen werden(siehe 3.5).

	`SWITCH_7` = 0	`SWITCH_7` = 1
OUT	`SAMPLE_FSM`	`CALC_FSM`
1	wait_for_first_period	POR
2	wait_for_sig_inc_low	IDLE
3	wait_for_sig_inc_high	set_add_state
4	LUT_enable_state	add_done_state
5	LUT_done_state	set_calc_div_state
6	MulAdd_set_state	set_calc_sqrt_state
7	MulAdd_done_state	success
8	wait_for_sample_timer	add_all_done_state
9	wait_for_sample_timer_after_last_sample	set_scale_down_state
A	Acceleration_error	
B	Deceleration_error	
C	New_data_state	
D		Fail
E	others	others
F	default	default

Tabelle 3.5: *Ausgänge an State out, abhängig von Schalter 7*

Byte	Bedeutung
'#'	Präambel
N	Anzahl an an Stellen in D, ASCII Darstellung
D	Anzahl an an Bytes in A, ASCII Darstellung
A	Daten

Tabelle 3.6: *Bytewerte des RS232 Ausgabeprotokolls.*

3.6.3 Ausgabe von Ergebnissen via RS232 an Matlab

Zur Ausgabe der berechneten Werte wird der Opencore Wishbone Bus [15] verwendet. Dieser kommuniziert dann mit der RS232 Schnittstelle. Der Miniuart (bestehend aus Wishbone Bus und TxD-RxD Komponenten) wurde von Philippe Carton [3] geschrieben, die Schnittstelle zu der hier verwendeten Hardware wurde von Andreas Arvidsson [1] entwickelt. Diese Schnittstelle wurde nur um ein Triggersignal erweitert, welches das Senden der Daten auslöst. In der momentanen Implementation wird bei 19200 Baud übertragen, was für eine Übertragung von Daten bis zu einer Zahnfrequenz von 200 Hz ausreicht. Das verwendete Protokoll entspricht jenem, welches von Matlab durch `binblockread` gelesen werden kann. In Tabelle 3.6 ist das Protokoll abgebildet.

3.7 Ausgabe des Protokolls

In einer Studienarbeit [6] wurde der Protokollgenerator für das digitale Augabeprotokoll geschrieben. Der Generator wurde so entwickelt, dass er an die Signalverarbeitung angeschlossen werden kann (siehe Bild 3.18).

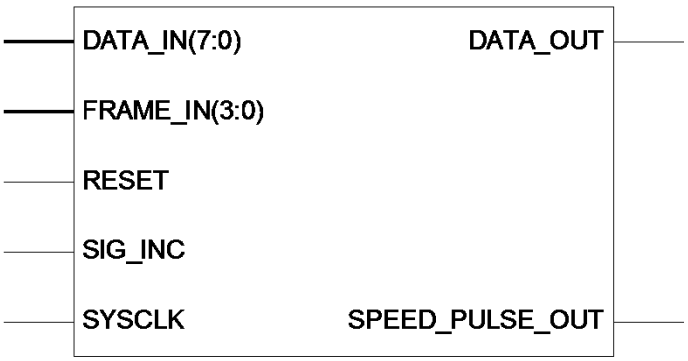


Bild 3.18: *Toplevelansicht des Protokollgenerators*

FRAME_IN ist dabei ein aus der Testphase stammender Eingangsbus, welcher hart auf x"9"gesetzt werden muss, da die Framegröße des zu sendenden Signals neun Bit ist (acht Datenbits und ein Paritätbit). Die Datenbits sind wie in Tabelle 3.7 belegt. Zu beachten ist, dass der Taktteiler aus der Komponente entfernt werden muss, zu Gunsten des hier verwendeten Taktteilers.
Tabelle 3.7 zeigt die Belegung von DATA_IN. DATA_IN(0) ist offen, weil die Luftspaltreseserve aus der Versärkungregelung kommt. Die Verstärkungsregelung wurde in dieser Arbeit nicht realisiert, das Bit kann also noch nicht gesetzt werden. DATA_IN(2) ist nicht offen, sondern frei. Ihm ist noch keine Funktion zugeteilt worden, es wird also auf 0 gesetzt.

DATA_IN	Signal	
0	offen	Luftspaltreserve
1	THD_ERROR	THD Fehlerbit
2	frei	frei
3	DV	Drehrichtung Fehlerbit
4	DR	Drehrichtung
5	THD_OUT(3)	THD Bit 0
6	THD_OUT(3)	THD Bit 1
7	THD_OUT(3)	THD Bit 2

Tabelle 3.7: *Zuweisung der Ausgänge der Signalverarbeitung auf die Eingänge des Protokollgenerators.*

3.8 Benötigte Ressourcen

Die hier beschriebene Implementation benötigte die in Tabelle 3.8 gelisteten Ressourcen des FPGAs. Der kritische Pfad begrenzt die Taktgeschwindigkeit auf $67MHz$.

	genutzt	verfügbar	in Prozent
Anz. Slice Flip Flops	1669	17344	9%
Anz. 4 input LUTs	4490	17344	25%
Anz. genutzer Slices	2506	8672	28%
Anz. genutzte IOBs	89	250	35%

Tabelle 3.8: *Auflistung der benötigten Ressourcen*

4 Messungen

Das Kapitel Messungen ist in drei Abschnitte eingeteilt:

1. Verifikation der Komponenten.
   Es wird gezeigt, dass die Messergebnisse zuverlässig sind.
2. Messung via RS232 und Analyse mit Matlab
   Messdaten werden Automatisch aufgenommen und anschließend Analysiert.
3. Vergleich mit vorherigen Arbeiten
   Das Ergebnis der Messung wird mit einer ähnlichen Messung verglichen.

4.1 Verifikation der Komponenten

Um die einzelnen Komponenten prüfen zu können, müssen wiederholbare Ergebnisse erzeugt werden können. Deshalb wurde in `TL_SAMPLE` eine Wertetabelle mit dem Namen `ADC_DB_LUT` integriert (siehe Seite 40, Bild 3.6). Diese Komponente liefert jede Periode die selben Werte, so dass Register wie folgt ausgelesen werden können: Der Debug Bus der zu untersuchenden Komponente wird an den Siebensegmenmultiplexer angeschlossen. Die Register können dann schrittweise ausgelesen werden.

4.1.1 Aufstellen einer Testfunktion mit Matlab

Anforderung an die Testfunktion ist, dass sie ein starkes sinusförmiges Trägersignal und ein Störsignal hat. Letzteres sollte Sinus und Cosinusanteile enthalten, damit jedes Register genutzt wird. Die verwendete Funktion ist (siehe Bild 4.1)

$$f(s) = int(2967 + 800sin(\frac{2\pi s}{64}) + 200sin(\frac{4\pi s}{64} + 7)) \tag{4.1}$$

Die zu prüfenden Daten sind:

- der Ausgang von `TL_SAMPLE`

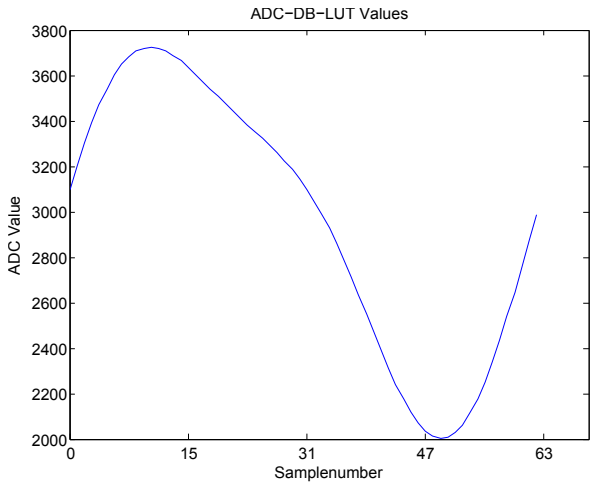


Bild 4.1: *Werte aus Gleichung 4.1*

- das Ergebnis der Quadrierung
- das Ergebnis der Summierung des Gesamtsignals
- das Ergebnis der Oberwellen
- das Ergebnis der Division
- das Ergebnis der Radizierung

4.1.2 Verifikation der Komponenten

In Tabelle 4.1 sind die von Matlab berechneten, in Tabelle 4.2 die tatsächlichen Werte dargestellt. Die tatsächlichen Werte sind ohne Vorzeichen, da beim Übergang von `TL_SAMPLE` zu `TL_CALC` der Betrag aller Werte gebildet wird. Das Vorzeichen ist ab hier auch nicht mehr von Bedeutung, weil die Werte als nächstes quadriert werden.

index	1	2	3	4	5
real	-1078	8600429	5109	3338	-8056
imaginär	52403705	9874000	-10001	-4446	2129

Tabelle 4.1: *Sollwerte am Ausgang von `TL_SAMPLE`*

Die Werte aus Tabelle 4.1 werden quadriert (siehe Tabelle 4.3) und mit den gemessenen Ergebnissen in Tabelle 4.4 verglichen. Da die Ergebnisse identisch sind, ist davon auszugehen, dass die Komponente funktioniert.

index	RE/IM	Hex	Dec
1	Re	436	1078
1	Im	31F9DF9	52403705
2	Re	833B6D	8600429
2	Im	96AA50	9874000
3	Re	13F5	5109
3	Im	2711	10001
4	Re	D0A	3338
4	Im	115E	4446
5	Re	1F78	8056
5	Im	851	2129

Tabelle 4.2: *Registerwerte am Eingang von* `TL_CALC`

$SQRT\_IN^2$		$SQRT\_OUT$	Σ
851^2	=	4529A1	4529A1
$1F5E^2$	=	3DE4840	42371E1
$115E^2$	=	12D9E84	5511064
$D0A^2$	=	AA0464	5FB14C9
2711^2	=	5F62F21	BF143EA
$13F5^2$	=	18E4879	D7F8C63
$96AA50^2$	=	58AC70E5900	58AC148DE563
$833B6D^2$	=	4345DF596C69	9BF1F3E751CC
$31F9DF9^2$	=	9C19B79C95C31	A5D8D6DB0ADFD
436^2	=	11BB64	A5D8D6DC26961

Tabelle 4.3: *Prognose für Ein- und Ausgang von* `square.vhd`*, sowie der Summe der Produkte*

index	RE/IM	result	Signal
5	Im	4529A1	
5	Re	42371E1	
4	Im	5511064	
4	Re	5FB14C9	
3	Im	BF143EA	
3	Re	D7F8C63	
2	Im	58AC148DE563	
2	Re	9BF1F3E751CC	u25_sig
1	Im	A5D8D6DB0ADFD	
1	Re	A5D8D6DC26961	uall_sig

Tabelle 4.4: *Gemessene Ergebnisse der Summe validieren die Prognose*

Die Datenaufnahme war für diese beiden Komponenten zeitaufwändig. Wie in 3.6.1 beschrieben, ist der Debugvektor nur acht Bit breit, so dass fünf Synthesen benötigt werden, um alle Bits eines Registers auszulesen. Der Aufwand ist bei den verbleibenden Komponenten, `calc_div` und `calc_sqrt`, geringer. Bei ihnen werden nur die Ausgangsvektoren untersucht. Jeder Ausgangsvektor ist der Eingangsvektor der nächsten Komponente und nur halb so lang wie der vorherige.

CALC_DIV hat einen Ergebnisvektor mit einer Länge von 32 Bit. Dieser wurde aus `u25_sig` und `uall_sig` als Nenner und Zähler gebildet.
Das gemessene Ergebnis lautet `x"0785B9"`
Diese Form ist wenig aussagekräftig. Die einzelnen Bits müssen gewichtet, mit dieser Gewichtung multipliziert werden und die Produkte zu einer Summe addiert werden. Diese Summe kann dann mit dem berechneten Ergebnis aus Matlab verglichen werden.
Der erwartete Wert wurde zu 0,05876 bestimmt. Skaliert man das Ergebnis auf 2^{31}, d.h. teilt es durch diese Zahl, erhält man:

$$\frac{0785B9_{hex}}{2^{31}_{dec}} = 0,05876_{dec} \tag{4.2}$$

Dies ist der erwartete Wert.

CALC_SQRT hat als Ausgang ein 16 Bit breites Ergebnis. Dieses wird zu $\sqrt{0,05879} = 0,2424$ vorausberechnet. Gemessen wird am Ausgang von `CALC_SQRT`, und skaliert auf Wurzel aus 32 Bit.

$$\frac{2BE2_{hex}}{2^{15,5}_{dec}} = 0,2424 \tag{4.3}$$

Auch hier entspach die Messung dem erwarteten Wert. Man kann also annehmen, dass die Komponenten funktionieren. Ab jetzt werden die Komponenten als Ganzes getestet, und nur bei nicht plausiblen Ergebnissen der Rückschritt auf das Durchleuchten einzelner Komponenten zur Laufzeit vollzogen.

4.2 Messung via RS232 und Analyse mit Matlab

Gemessen wurde an Radmessplatz 2 (siehe Bild 4.2). Abgebildet ist dort die Verfahreinheit des Messplatzes. Unter dem passiven Encoderrad ist ein Referenzsensor fixiert. Hinter dem Encoderrad ist der Messsensor. Dieser wird während der Messung linear vom Encoderrad angefahren. Die Eckdaten der Messung sind:

- Entfernung: 3 mm bis 0,1 mm
- Schrittgröße: 0,1 mm

- Zahnfrequenz: 120 Hz
- Zeit: 4s pro Schritt

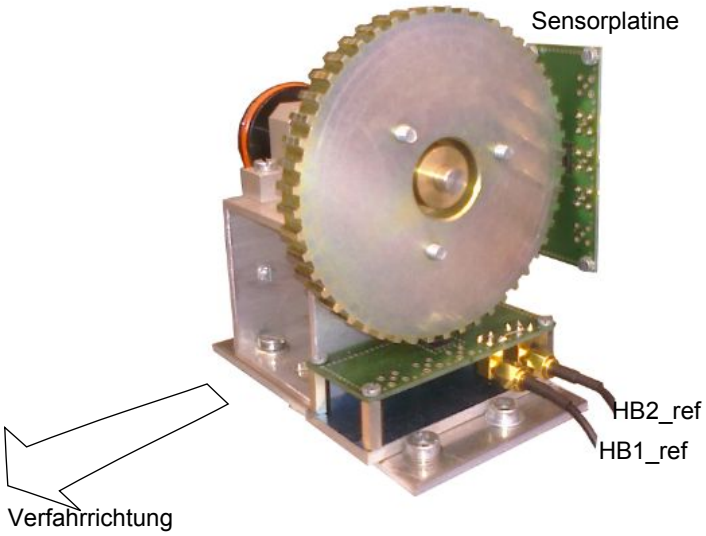


Bild 4.2: *Auschnitt von Radmessplatz 2*

Die Anschlüsse des Messsensors sind auf Bild 4.2 nicht zu sehen. Auch von ihm werden, wie vom Referenzsensor, die Halbbrückensignale `HB1` und `HB2` genommen. In der in 1.1.1 beschriebenen Verstärkerplatine mit Verstärkungsregelung wird aus den beiden Halbbrückensignalen das Signal `U_diff` gebildet und verstärkt. Leider ist die Verstärkungsregelung momentan nicht fehlerfrei, so dass sie etwa bei einer Entfernung von 3 mm den Regelungsbetrieb einstellt. Es sind deshalb also momentan leider keine Messungen möglich, die weiter als 3 mm vom Sensor entfernt sind.

Zur Messung:
Jedes radizierte und skalierte Ergebnis der THD Berechnung wird über RS232 an Matlab gesendet. Dort werden die Daten in Prozent umgewandelt und gespeichert. Jeder der THD Werte aus der Ergebnisgraphik 4.3 ist ein Mittelwert, gebildet aus 50 THD Werten. Ohne den Mittelwert würden die Ergebnisse um einige Prozent schwanken, da das Encoderrad nicht ganz rund läuft. Die Graphik 4.3 verdeutlicht, wie in besprochen, das Verlassen der linearen Übertragung des Sensors bei starkem Magnetfeld.

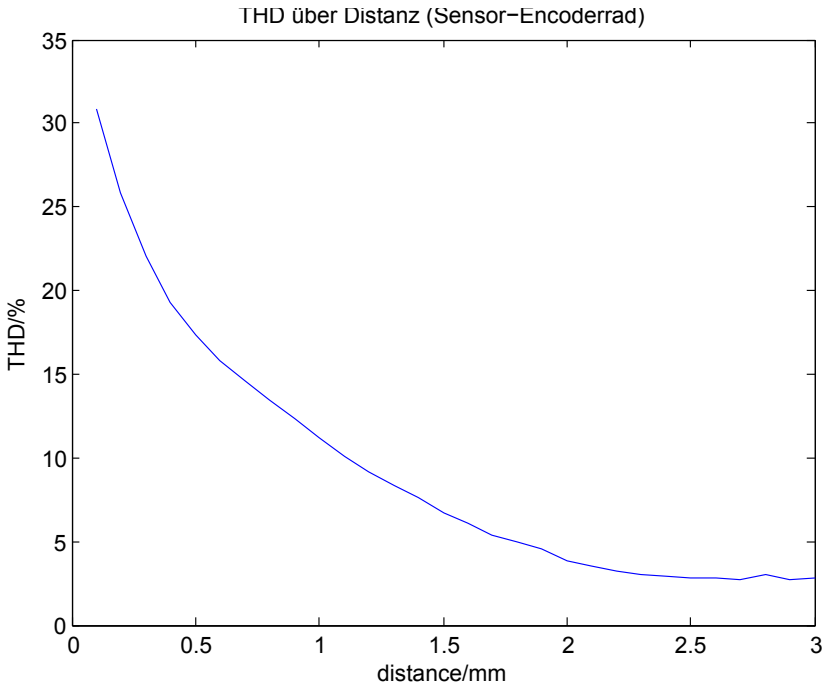


Bild 4.3: *Messung THD über Anbstand zwischen Sensor und Encoderrad*

4.3 Vergleich mit vorherigen Arbeiten

Die Ergebnisse sind nur von der Aussage her zu vergleichen, nicht auf Prozente genau, da sowohl der Radmessplatz als auch die Datenauswertung in Matlab unteschiedlich sind. Die zur Datenanalyse verwendete Demonstratorplatine kann nicht ausreichend schnell abtasten, deshalb wird eine Unterabtastung verwendet (siehe 2.3.5). Hier kommt die Form der Unterabtastung zum tragen, welche maximal einen Abtastwert pro Encoderzahn nimmt. Deshalb wird nicht 4 s pro Verfahrschritt benötigt, sondern 30 s. Als die Messung aufgenommen wurde, gab es die Verstärkungssteuerung noch nicht, es wurde mit der Hand die Verstärkung eingestellt.

Die Aussage der Messung auf Bild 4.4 ist ähnlich der auf Bild 4.3. Beide haben nahe dem Encoder einen THD um 30%, welcher dann exponetiell abnimmt, bis er zwischen 1,5 mm und 2 mm Abstand auf unter 5 % fällt. Dort ist er dann relativ stabil zwischen 2 % und 3 % was, durch das zunehmende Rauschen zu erklären ist. Wenn man nur von der Übertragungskennlinie des Sensor ausgeht, müsste der THD mit zunehmender Entfernung gegen 0 % gehen. Die Verstärker verstärken aber nicht nur das Sensorsignal, sie verstärken auch das Rauschen auf dem Signal, so dass mit zunehmendem Abstand zwischen Sensor und Encoderrad die Ursache des ho-

hen THDs wechselt. Von Nichtlinearitäten in der Sensorübertragungsfunktion zu Rauschen auf dem Sensorsignal.

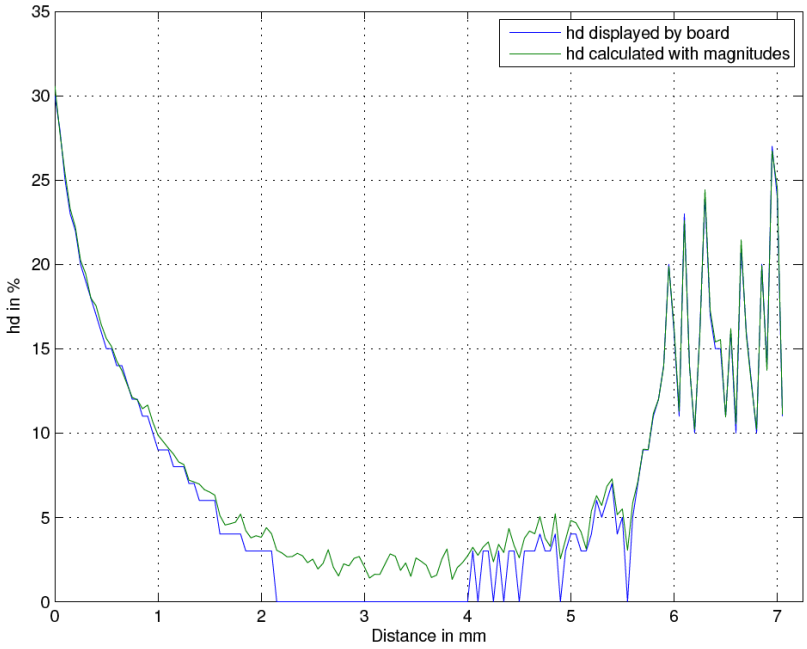


Bild 4.4: *Messung THD auf Demonstrator (siehe 1.1.1)Quelle: [9]*

5 Fazit, Bewertung & Ausblick

5.1 Fazit und Bewertung

Das Hauptaugenmerk dieser Arbeit, die Entwicklung eines FPGA Prototypen zur Signalverarbeitung für ABS-Sensoren mit Diagnosefunktion, ist erfolgreich durchgeführt worden. Es wurde ein Ansatz gefunden, der es auf Grund seiner Modularität erlaubt, ihn als Entwicklungsplattform für weiterführende Arbeiten zu nutzen. Dieser Ansatz setzt sowohl die Grundfunktionen des ABS-Sensors um, als auch die im Rahmen des Projektes „Experimentelle digitale Signalverarbeitung und Zustandserkennung für ABS-Sensoren“ entwickelte Diagnosefunktion. Zusätzlich verfügt der Prototyp über Schnittstellen, die es erlauben Prozesse zur Laufzeit zu Analysieren, sowie automatisierte Messungen auf zu zeichnen.

Am Anfang der Arbeit wurde ein Konzept für den Aufbau der Architektur entworfen. Dabei wurde die Arbeit in fünf Kernpunkte eingeteilt und priorisiert:

1. Grundfunktionen
2. Abtasten
3. Berechnung des THD
4. Ausgabe von Werten an den Logicanalyzer durch Debug Multiplexer
5. Senden des THDs via RS232

In ModelSim wurden die ersten drei Punkte umgesetzt. Es musste zu diesem Zeitpunkt mit einem groben Simulationsmodell des ADCs gearbeitet werden; die Ansteuerung war noch in einer anderen Arbeit in Entwicklung [1]. Eine Testbench übernahm übergangsweise die Funktion des ADCs, und versorgte die Komponenten mit den Signalen `U_DIFF`, `HB1` und `HB2`. Auf dieser Basis wurden die Grundfunktionen entwickelt und simuliert.

Das Konzept des Smart Comparators wurde dafür neu interpretiert: eine dynamische Schaltschwelle läßt den Comparator immer im tatsächlichen Offset von `U_DIFF` schalten. So tritt ein Hysterese ähnlicher Effekt auf und die Schaltschwelle ist trotzdem genau im Offset. Der Smart Comparator generiert so aus dem Eingangssignal ein Rechtecksignal und einen Signalpuls, wenn das Rechtecksignal

einen Polwechsel hat.
Die Zweite Komponente zur Sicherung der Grundfunktion erkennt die Drehrichtung. Diese Komponete konnte leider nur simuliert werden, da diese Arbeit mit einer noch nicht fertig entwickelten Ausgabe der ADC-Steuerung geschrieben wurde. Das heißt, dass diese ADC-Steuerung noch keine Halbbrückensignale ausgibt.

Die Diagnosefunktion wurde in zwei Komponenten umgesetzt. Die erste Komonente, `SAMPLE_TL`, steuert die Abtastung und teilt `U_DIFF` in Real- und Imaginärteil ein. Das Besondere bei der Abtastung ist, dass immer 64 Werte pro Periode genommen werden, unabhängig von der Signalfrequenz. Der begrenzende Faktor dieser Komponente, die Verarbeitungszeit, begrenzt das gesamte Design. Diese Verarbeitungszeit setzt sich zusammen aus der Zugriffszeit auf eine Wertetabelle, in der Sinus und Cosinus abgelegt sind, sowie der sequentiellen Multiplikation von Abtastwert `U_DIFF` mit einem Wert aus der Wertetabelle. Da dies zusammen sehr zeitaufwendig ist, können momentan nur Eingangssignale bis zu einer Zahnfrequenz von ca. 700 Hz verarbeitet werden. Dies wurde in Abschnitt 2.3.5 schon diskutiert.
Die zweite Komponente der Diagnosefunktion, `CALC_TL`, nimmt die zehn Ergebnisse von `SAMPLE_TL` und berechnet aus ihnen den THD. Dabei geht hier die Implementation tiefer als nötig: der THD wird vollständig berechnet. Das ziehen der Wurzel wäre in einer finalen Version nicht nötig, das Ergebnis wird nur auf drei Bit ausgegeben. Es ist hier aber hilfreich, diesen Schritt zu weit zu gehen, da man so ohne Hilfe eines Computers die Ergebnisse Interpretieren kann. Zusätzlich wird der THD auf drei LEDs ausgegeben. Diese LEDs symbolisieren die drei Bits, auf welche er im digitalen Ausgabeprotokoll quantisiert werden soll. Diese Quantisierung muss noch genau definiert werden.

An dieser Stelle waren die Möglichkeiten der Simulation erschöpft und die Entwicklung wurde auf dem NEXYS2 Board fortgesetzt. Der Wechsel der Entwicklungsplattform vollzog sich Schrittweise. Zuerst wurde unfertige ADC wurde eingebunden, ohne dass es für ihn ein Simulationsmodell gab. Dann mussten die simulierten Komponenten erneut angepasst und umgeschrieben werden. Dabei half der dann entwickelte Debug-Multiplexer. Nach seiner Fertigstellung war es möglich den Signalverlauf sowohl auf dem Logic Analyzer als auch auf der Siebensegmentanzeige zu verfolgen.

Nachdem volle Funktionalität sicher gestellt war, wurde das RS232 Interface eingebunden. Diese Komponente ist, wie auch die ADC-Steuerung, geborgt [1]. Mit ihr war dann das automatische Aufnehmen von Messdaten möglich.

Es wurden zwei Messreihen durchgeführt, eine davon wird in Abschnitt 4.2 diskutiert. Sie zeigt eine sehr glatte Kurve, was daran liegt, dass über 50 THD Werte gemittelt wurde. Die Zweite Messreihe nahm pro Messpunkt immer nur einen THD

Wert auf, so dass es deutliche Schwankungen von bis zu 2 % in den Messungen gab. Dies kann auf die Rundungsfehler beim Wählen der Abtastzeitpunkte zurückgeführt werden. Jeder Abtastzyklus wird auf den Takt genau in 64 gleich große Abschnitte eingeteilt. Dadurch gibt es Rundungsfehler, welche zu leicht schwankenden Ergebnissen führen können. Die Schwankungen sind aber im untersten Prozentbereich, fallen also nicht ins Gewicht.

Mit dem Funktionsumfang, den dieser Prototyp hat, ergänzt er das Projekt ESZ-ABS und kann einen guten Beitrag leisten. Seine Komponenten mit definierten Schnittstellen ermöglichen es, mit relativ geringen Aufwand neue Algorithmen aus zu probieren. Im Gegensatz zu einer Implementation auf einem Microcontroller muss man sich wenige Gedanken um das Scheduling machen. Es sind noch ausreichend Ressourcen auf dem FPGA frei.
Erfreulich ist außerdem, dass die Messgeschwindigkeit erhöht werden konnte. Eine Messreihe hat früher 30 Sekunden pro Schritt gedauert, die selbe Messreihe wird jetzt mit 4 Sekunden pro Schritt verarbeitet. Begrenzender Faktor ist dabei die Verstärkungsregelung und Offsetkompensation. Der Prototyp sammelt idealerweise 50 Messwerte in 51 Perioden.

5.2 Ausblick

Es ist nur der erste Schritt auf die neue Plattform vollbracht. Nun müssen nach und nach weitere folgen, um die Umsetzung in Hardware zu vervollständigen. Offene Punkte für die Entwicklung in VHDL sind:

- Lösen des in 2.3.6 angesprochenen Problems der Rechengeschwindigkeit. Dazu kann an mehreren Stellschrauben gedreht werden. Es kann die Taktfrequenz erhöht, das Abtastverfahren gewechselt, die Bitbreite der Multiplikation verringert, paralleler gerechnet oder eine Kombination aus diesen Ansätzen verwendet werden.
- Einbinden der finalen Komponenten zur Ansteuerung von RS232 und ADC. Diese sind momentan beide in der Entwicklung [1].
- Die Erkenntnisse aus der Arbeit zur Aufwandsminimierung bei der THD Berechnung [10] müssen implementiert werden. Dadurch sollte sowohl die Berechnungszeit als auch der Hardwareaufwand verringert werden.
- Die Verstärkungs- und Offsetregelung wird aktuell noch mit einem MSP430f1611 durchgeführt. Die vom Autor verwendete ADC Platine verfügt über eine Anschlussmöglichkeit für die bereits entwickelte Verstärkerplatine. Dies sollte genutzt werden. Um das umsetzen zu können, muss der aktuelle

Regelalgorithmus überarbeitet werden, da er bei mehr als 3*mm* Abstand vom Encoderrad die Funktion einstellt.

- Verschiedene Messungen.
  Mit der erhöhten Verarbeitungsgeschwindigkeit konnte das Aufnehmen von automatischen Messreihen signifikant beschleunigt werden. Diesen Zuwachs an Messgeschwindigkeit kann in umfassenden Messungen an dem präzisions Radmessplatz [20] genutzt werden.

Nachdem diese Schritte alle erfolgreich abgeschlossen sind, kann die Entwicklung eines VLSI Designs beginnen.

6 Danksagung

An dieser Stelle möchte ich mich bei Herrn Prof. Dr.-Ing. Karl-Ragmar Riemschneider für seine Unterstützung und sein Engagement bedanken. Sie waren eine große Hilfe.

Ebenso bedanke ich mich bei Herrn Prof. Dr. Franz Schubert für seine Arbeit als Zweitgutachter.

Des Weiteren bedanke ich mich besonders bei Herrn Dipl.-Ing. Martin Krey, für seinen fachlichen Rat und die tatkräftige Unterstützung bei meiner Diplomarbeit.

Natürlich danke ich ebenso meinen Kommilitonen für ihre Unterstützung. Insbesondere Andreas Arvidsson für die Entwicklung der ADC-Steuerung und für den Mini-Uart.

Literaturverzeichnis

[1] Andreas Arvidsson. Analog-interface-modul für einen fpga-prototyp der signalverarbeitung bei abs-sensoren. Master's thesis, HAW-Hamburg, 2010.

[2] R. Boll and K. J. Overshott. Magnetic sensors. In W. Göpel, J. Hesse, and J. N. Zemel, editors, *Sensors a Comprehensive Survey*, volume 5. VCH, Weinheim, 1989. insb. Kap. 9 von U. Dibbern.

[3] Philippe Carton. Miniuart ip core specification. 18.6.2010 http://www.ece.utk.edu/ ddarby/-My%20Documents/Projects/Hardware/DAQ/DAQ/doc/part_data/Cores/MiniUart.pdfı

[4] Daimler AG. Requirement specifications for standardized interface for wheel speed sensors with additional information 'ak-protokoll', Februar 2008.

[5] Delphi. Delphi digital speed and position sensors, 2010. 15.7.2010 http://delphi.com/manufacturers/auto/sensors/et/digspd/.

[6] Jan-Heiner Dreschhoff. Vhdl - implementierung des ausgabeprotokolls von abs - sensoren. Studienarbeit, 2010.

[7] Andreas Göbel. Ziehen einer wurzel. 7.5.2010 http://www.diaware.de/html/wurzel.html.

[8] Julius O. Smith III. *Mathematics of the Discrete Fourier Transform (DFT), with Audio Applications Second Edition*. W3K Publishing, 2007.

[9] Niels Jegenhorst. Entwicklung eines controllersystems zur zustandserkennung von abs-sensoren. Master's thesis, HAW-Hamburg, 2009.

[10] Lennart Koch. Aufwandsminimierte schätzung von harmonischen zur zustandsbestimmung von abs-sensoren. Master's thesis, HAW-Hamburg, 2010.

[11] Yamin Li and Wanming Chu. A new non-restoring square root algorithm and its vlsi implementations, 1996. The University of Aizu, Japan.

[12] Dr. David Lin. Drehzahlen einfach erfassen. 23.6.2010 http://imperia.mi-verlag.de/imperia/md/content/ai/ae/fachartikel/ei/2008/07/ei08_07_030.pdf.

[13] H. Lindner, H. Brauer, and C. Lehmann. *Taschenbuch der Elektrotechnik und Elektronik*. Linder u.a., 2005. insb. Kap. 5.

[14] Abdelkhalek Mahtouf. Messungen und signalanalyse an einem magnetischen sensor. Master's thesis, Hochschule für Angewandte Wissenschaften Hamburg, 2008.

[15] OpenCores.org. Soc interconnection: Wishbone. 12.6.201 http://opencores.org/opencores,wishbone.

[16] Noel Oriordan. An automotive multi-chip module for rotational speed and direction measurement. Printed and Published by IEE, Savory Place, London WC2R 0BL, UK, 2000. Mixed Signal Group, Silicon & Sofware Systems, Dublin, Ireland.

[17] Philips. Kmi22/1 rotational speed sensor for extended air gap application and direction detection, 2000.

[18] Reichardt and Schwarz. *VHDL-Synthese: Entwurf digitaler Schaltungen und Systeme*. Oldenbourg Wissenschaftsverlag GmbH, 5. auflage edition, 2009.

[19] Christian Schoermer. Amr-messbrücken für abs-sensoren. Studienarbeit, 2009.

[20] Christian Schoermer. Konstruktion und automatisierung eines radmessplatzes für abs sensoren mit encodern verschiedener automobil-hersteller. Master's thesis, HAW-Hamburg, 2010.

[21] Fritz Dettmann U. Loreit. Neue magnetoresistive sensorchips für die magnetfeldmessung, 1995.

[22] Thomas Tille und 83 Mitautoren. *Sensoren im Automobil*. Prof. Dr.-Ing Ulrich Brill, 2009. insb. Kap. 8 von E. Katzmaier, P. Slama, T. Werth, K. Kasper.

[23] R. Gross und A. Marx. *Der anisotrope Magnetwiderstand*, chapter 3. Walther-Meißner-Institut, 2005.

[24] Daniel Mlynek und Yusuf Leblebici. Design of vlsi systems. 20.7.2010 http://lsiwww.epfl.ch/LSI2001/teaching/webcourse/toc.html.

[25] Schelter van den Berg. Device for detecting the angular postion of an object using a magnetoresistive sensor:united states patent us5650721, 1994.

[26] vipin. Vhdl coding tips and tricks. insb.: A VHDL Function for finding SQUARE ROOT.

[27] XILINX. *XST User Guide*. www.xilinx.com, 2005 edition, 2005.

[28] XILINX. *Constraints Guide*. www.xilinx.com, ug625 (v 11.3) edition, 2009.

A Entwurf

In Anhang A sind die Komponenten detailliert aufgezeigt, welche im Text nur skizziert werden konnten.

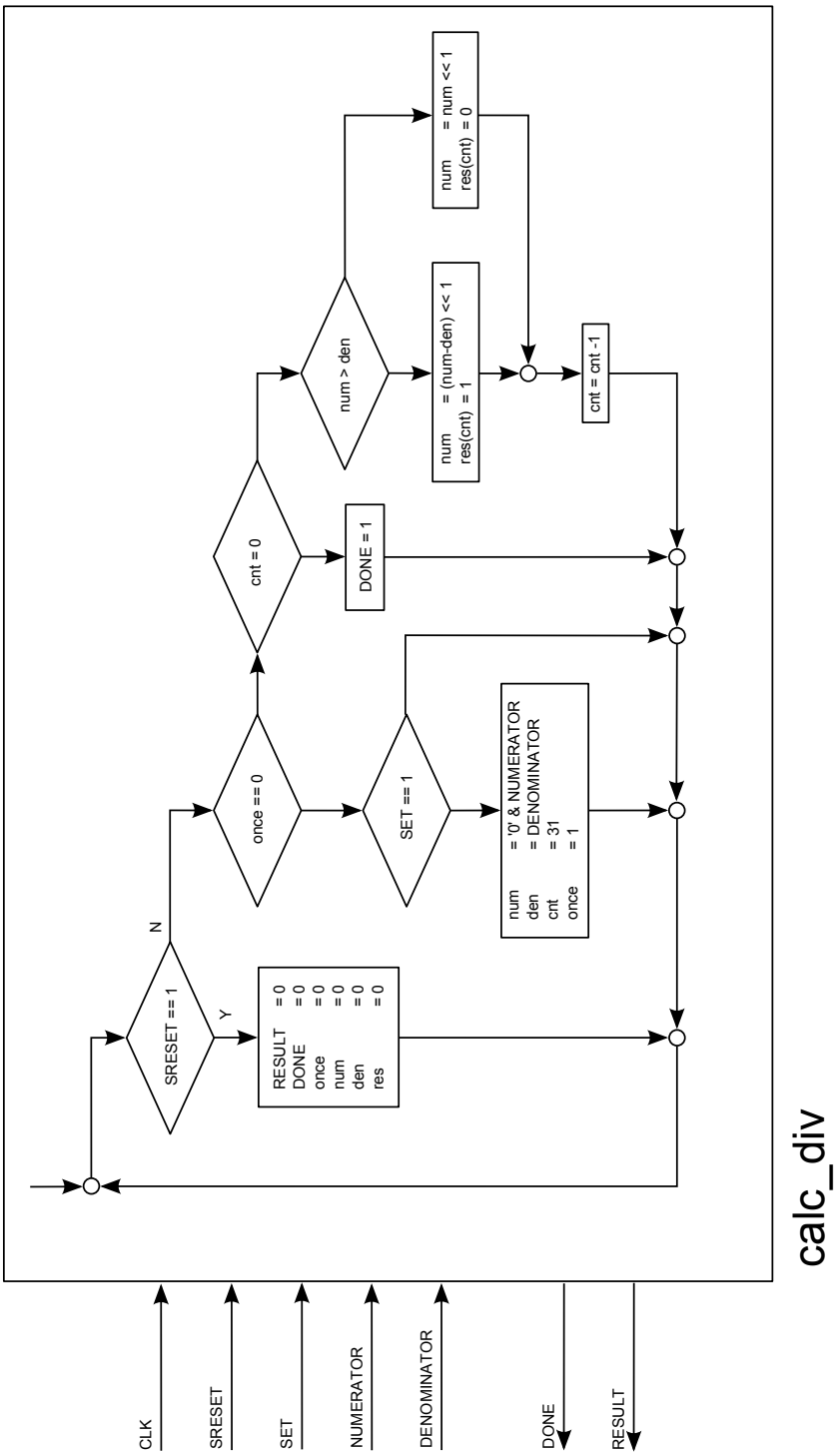


Bild A.1: *Aufbau der Divisionskomponente*

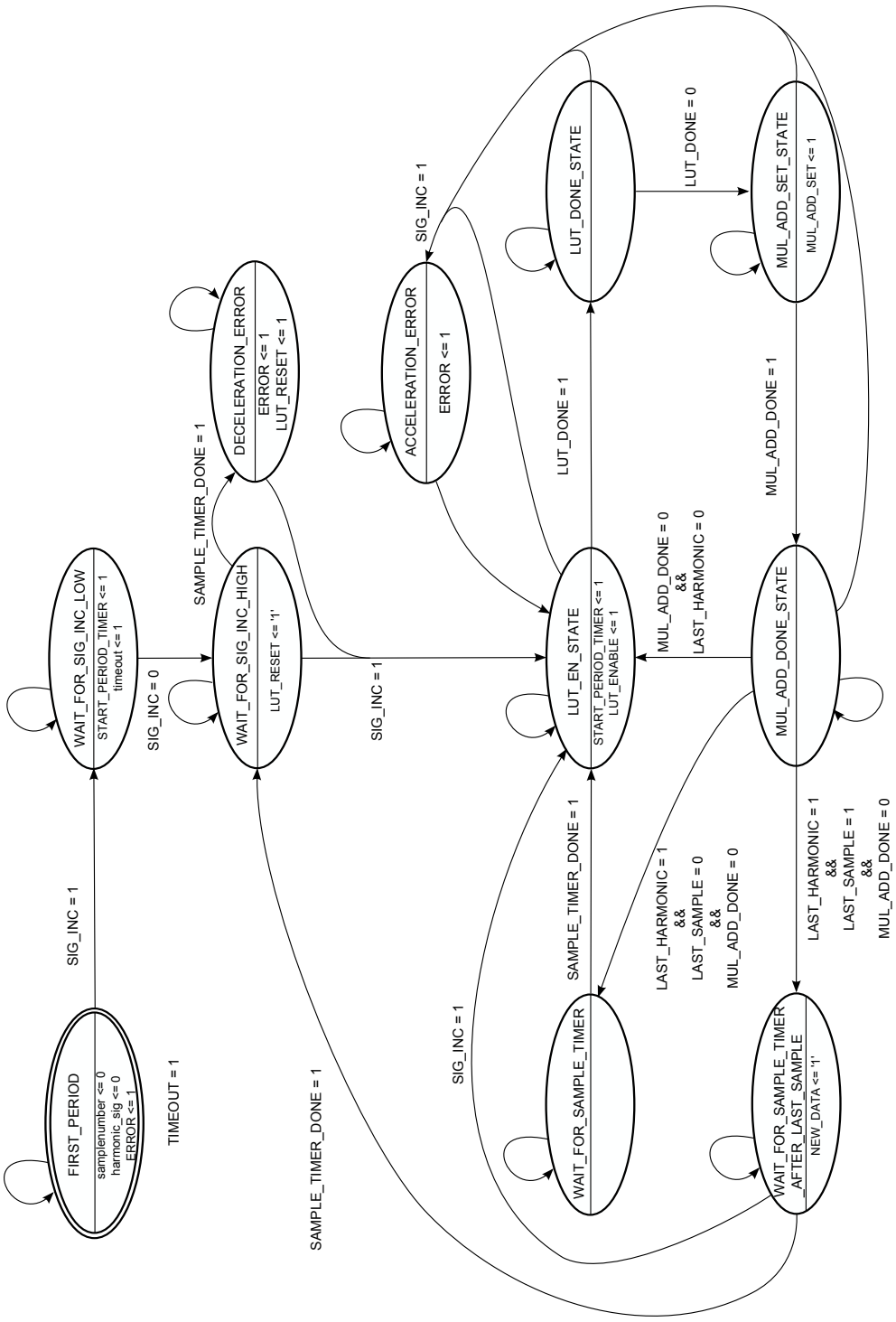


Bild A.2: *Aufbau des Zustandsautomaten* `SAMPLE_FSM`

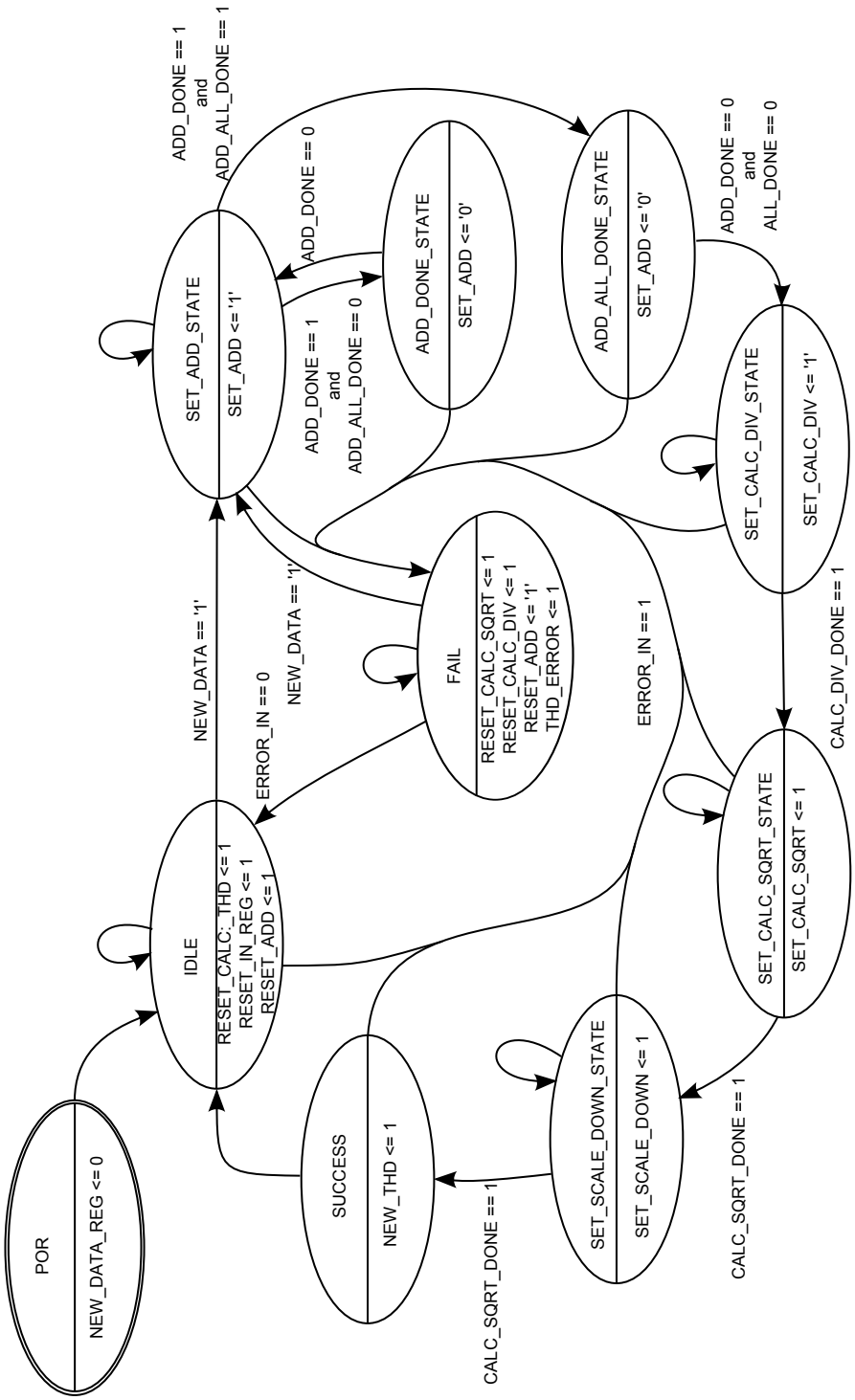
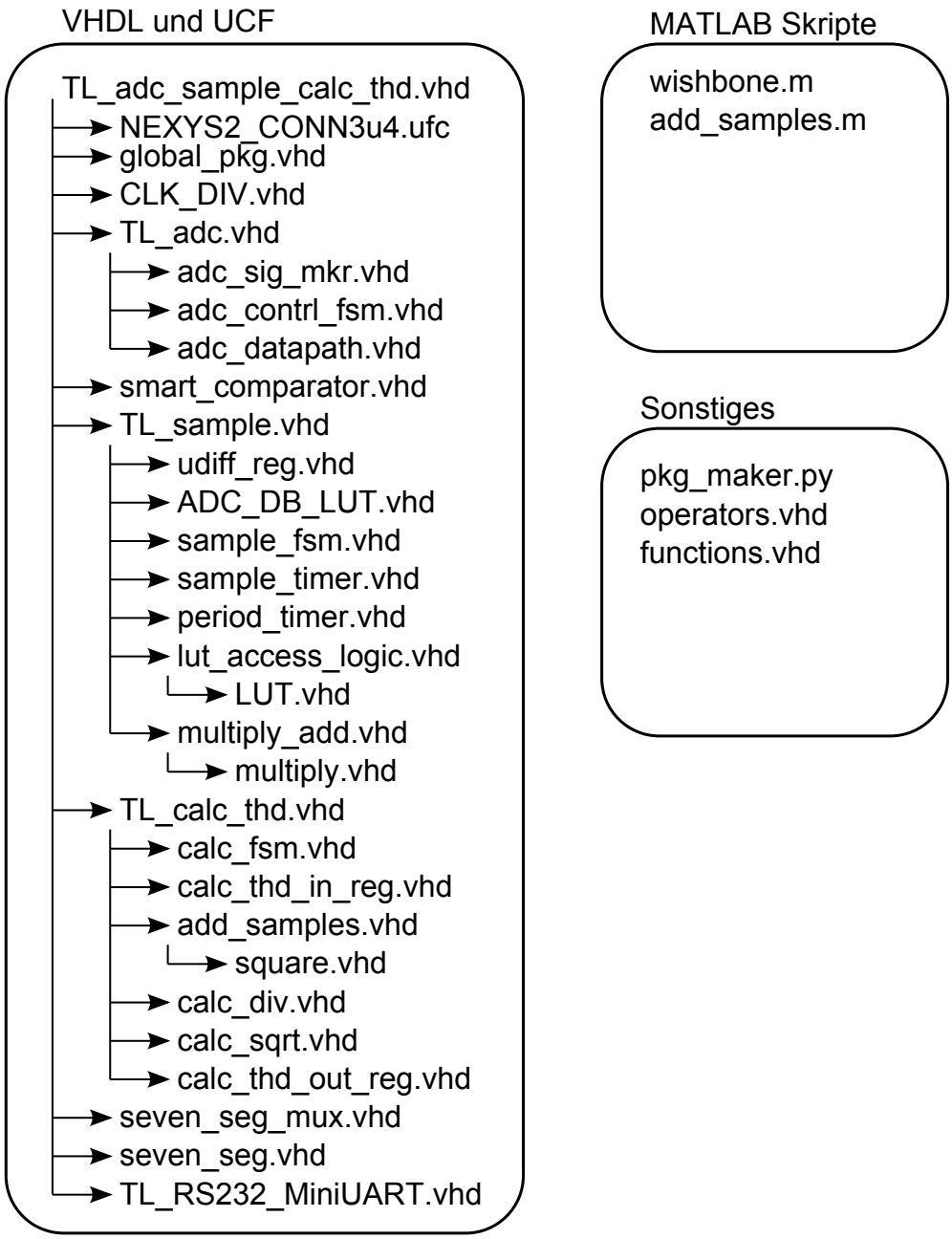


Bild A.3: *Aufbau des Zustandsautomaten* `CALC_FSM`

B Quellcode


VHDL und UCF
TL_adc_sample_calc_thd.vhd
NEXYS2_CONN3u4.ufc
global_pkg.vhd
CLK_DIV.vhd
TL_adc.vhd
adc_sig_mkr.vhd
adc_contrl_fsm.vhd
adc_datapath.vhd
smart_comparator.vhd
TL_sample.vhd
udiff_reg.vhd
ADC_DB_LUT.vhd
sample_fsm.vhd
sample_timer.vhd
period_timer.vhd
lut_access_logic.vhd
LUT.vhd
multiply_add.vhd
multiply.vhd
TL_calc_thd.vhd
calc_fsm.vhd
calc_thd_in_reg.vhd
add_samples.vhd
square.vhd
calc_div.vhd
calc_sqrt.vhd
calc_thd_out_reg.vhd
seven_seg_mux.vhd
seven_seg.vhd
TL_RS232_MiniUART.vhd
MATLAB Skripte
wishbone.m
add_samples.m
Sonstiges
pkg_maker.py
operators.vhd
functions.vhd

B.1 VHDL und UCF

Hinweise zur Notation:
Ports werden GROSS geschrieben.
Signale und Variablen werden klein geschrieben.
In Kommentaren sind XML-Tags zu finden. Diese sind sehr praktisch um z.B. den Anfang, und das Ende von Codesegmenten zu markieren, die nur zu Debugzwecken einkommentiert werden (siehe B.1.7, Zeile 123-165).
Am Anfang einer jeden *.vhd Datei sind Anweisungen in den Kommentaren. Auch diese sind in XML-Format. Geparst werden diese Zeilen von `pkg_maker.py` (siehe B.2, S. 181). Es handelt sich hierbei um ein kleines Pythonskript zur Pflege von Paketen. Mit der Anweisung `<makepackage>` wird ein Paket mit dem Namen der aktuellen Datei erweitert duch `_pkg` erzeugt. In diesem Paket ist dann eine Komponentendefinition, die mit `use` aus der library `work` eingebunden und verwendet werden kann. Es ging dabei um das Minimieren von Zeilen die verändert werden müssen, wenn man die Schnittstelle einer Komponente an neue Gegebenheiten anpasst. Zusätzlich können globale Konstanten, Typen und Subtypen definiert werden. Diese werden dann in eine Datei namens `global_pkg.vhd` geschrieben und mit einem Kommentar versehen, wo diese Werte ihren Ursprung haben (Siehe B.1.2, S. 90).

B.1.1 TL_adc_sample_calc_thd.vhd

```
1  --#########################################################
2  --##  Project name : adc_sample_calc_thd
3  --##  File name    : TL_adc_sample_calc_thd.vhd
4  --##  Author       : Jan-Heiner Dreschhoff
5
6
7  --<makepackage>
8  --  <global>
9  --    constant RESET_VAL : bit := '1';
10 --    subtype U32 is unsigned(31 downto 0);
11 --    subtype S33 is signed(32 downto 0);
12 --    type MulArrayS is array(1 to 5) of S33;
13 --    type MulArray is array(1 to 5) of U32;
14 --  </global>
15 --  <component = TL_adc_sample_calc_thd>
16 --    <name = dreschhoff>
17 --    <project = adc_sample_calc_thd>
18 --  </component>
19 --</makepackage>
20
21 library ieee;
```

```
22    use ieee.std_logic_1164.all;
23    use ieee.numeric_std.all;

25  -- Library UNISIM;   --für BUFG
26    -- use UNISIM.vcomponents.all;

28  library work;
29    use work.global_pkg.all;
30    use work.operators_pkg.all;
31    use work.functions_pkg.all;
32    use work.clk_div_pkg.all;
33    use work.TL_adc_pkg.all;
34    use work.dr_detection_pkg.all;
35    use work.smart_comparator_pkg.all;
36    use work.TL_sample_pkg.all;
37    use work.TL_calc_thd_pkg.all;
38    use work.seven_seg_mux_pkg.all;
39    use work.seven_seg_pkg.all;
40    use work.TL_RS232_MiniUART_pkg.all;

42  entity TL_adc_sample_calc_thd is
43    generic(
44      ADC_LOWER : unsigned((ADC_WIDTH-1) downto 0) := x"700";
45      ADC_MIDDLE: unsigned((ADC_WIDTH-1) downto 0) := x"7FF";
46      ADC_UPPER : unsigned((ADC_WIDTH-1) downto 0) := x"8FF"
47      );
48    port(
49      SYSCLK         : in bit;
50      RESET          : in bit;

52      --Manchester code
53      RECT_OUT       : out bit;
54      SIG_INC        : out bit;
55      DR             : out bit;
56      DV             : out bit;
57      THD_OUT        : out unsigned(2 downto 0);
58      THD_ERROR      : out bit;
59      NEW_THD        : out bit;

61      --ADC
62      CLK_OUT         : out bit;
63      ADC_BUSY        : in bit;
64      ADC_DB          : inout std_logic_vector(11 downto 0);
65      ADC_CS_N        : out bit;
66      ADC_WR_N        : out bit;
67      ADC_RD_N        : out bit;
68      ADC_CONVST_N    : out std_logic;
69      ADC_NEW_DATA_O : out bit_vector(1 downto 0);

71      --- DAC ---
```

```
72      DAC_CS_D_N        : out bit;
73      DAC_WR_D_N        : out bit;
74      DAC_LDAC_D_N      : out bit;
75      DAC_RD_D_N        : out bit;
76      DAC_CLR_D_N       : out bit;
77      DAC_BUF_D         : out bit;
78      DAC_GAIN          : out bit;
79      DAC_A_D           : out bit_vector(2 downto 0);
80      DAC_DAC_REGISTER  : out std_logic_vector(11 downto 0);

82      -- 7 seg
83      SEVEN_SEG_out     : out unsigned(6 downto 0);
84      DECIMAL_POINT     : out bit;
85      SEVEN_SEG_SELECT  : out bit_vector(3 downto 0);


88      P_BUTTON_2        : in bit;

90      --switches
91      SWITCH_7           : in bit;    --fsm_out
92      SWITCH_6           : in bit;    --adc_lut_db
93      SWITCH_5           : in bit;    --BCD ON
94      SWITCHES_0_3       : in bit_vector(3 downto 0);  --db_muxer

96      --LEDS
97      LED_0              : out bit;

99      --debug
100     S_TIMER_DONE   : out bit;
101     DB_DONE        : out bit;
102     DB_SET         : out bit;

104     BUS2       : out unsigned(7 downto 0);
105     STATE_OUT  : out std_logic_vector(3 downto 0);

107     --RS232
108     TxD_PAD_O : out std_logic;  -- Tx RS232 Line
109     RxD_PAD_I : in  std_logic   -- Rx RS232 Line
110   );
111 end TL_adc_sample_calc_thd;

113 architecture structure of TL_adc_sample_calc_thd is

115 --SIGNALS FOR CLK_DIV
116 signal CLK             : bit;
117 --SIGNALS FOR ADC
118 signal adc_udiff_sig   : unsigned((ADC_WIDTH-1) downto 0);
119 signal adc_hb1_sig     : unsigned((ADC_WIDTH-1) downto 0);
120 signal adc_hb2_sig     : unsigned((ADC_WIDTH-1) downto 0);
121 signal SLV_adc_udiff_sig  : std_logic_vector((ADC_WIDTH-1) downto 0);
```

```
122 signal SLV_adc_hb1_sig    : std_logic_vector((ADC_WIDTH-1) downto 0);
123 signal SLV_adc_hb2_sig    : std_logic_vector((ADC_WIDTH-1) downto 0);
124 --SIGNALS FOR CALC_THD
125 signal new_data_sig   : bit;
126 signal data_error_sig : bit;
127 signal re_S33         : MulArrayS;
128 signal im_S33         : MulArrayS;
129 signal re_U32         : MulArray;
130 signal im_U32         : MulArray;
131 signal new_thd_sig    : bit;
132 signal thd_error_sig  : bit;
133 signal thd_sig        : unsigned(15 downto 0);
134 signal clk_out_sig    : bit;

136 --SIGNALS FOR SMART_COMPARATOR
137 signal rect_out_sig : bit;
138 signal sig_inc_sig : bit;

140 --signals for rs232
141 signal txd_sig    : std_logic;
142 signal rxd_sig    : std_logic;

144 --signals for debug
145 signal channel_0_sig      : std_logic_vector(15 downto 0);
146 signal channel_1_sig      : std_logic_vector(15 downto 0);
147 signal channel_2_sig      : std_logic_vector(15 downto 0);
148 signal channel_3_sig      : std_logic_vector(15 downto 0);
149 signal channel_4_sig      : std_logic_vector(15 downto 0);
150 signal channel_5_sig      : std_logic_vector(15 downto 0);
151 signal channel_6_sig      : std_logic_vector(15 downto 0);
152 signal channel_7_sig      : std_logic_vector(15 downto 0);
153 signal channel_8_sig      : std_logic_vector(15 downto 0);
154 signal channel_9_sig      : std_logic_vector(15 downto 0);
155 signal channel_A_sig      : std_logic_vector(15 downto 0);
156 signal channel_B_sig      : std_logic_vector(15 downto 0);
157 signal channel_C_sig      : std_logic_vector(15 downto 0);
158 signal channel_D_sig      : std_logic_vector(15 downto 0);
159 signal channel_E_sig      : std_logic_vector(15 downto 0);
160 signal channel_F_sig      : std_logic_vector(15 downto 0);

162 signal db_udiff_reg       : std_logic_vector(15 downto 0);
163 signal db_bus_adc_lut     : std_logic_vector(15 downto 0);
164 signal db_rs232           : std_logic_vector(15 downto 0);

166 --<debug>
167   signal db_sample_fsm_state_sig  : std_logic_vector(3 downto 0);
168   signal db_calc_fsm_state_sig    : std_logic_vector(3 downto 0);
169   signal db_switchValue_sig       : std_logic_vector(11 downto 0);
170   signal db_minValue_sig          : std_logic_vector(11 downto 0);
171   signal db_maxValue_sig          : std_logic_vector(11 downto 0);
```

```
172   signal db_sample_timer_done_sig : bit;
173   signal db_sample_timer_set_sig  : bit;
174   signal db_8_bytes_out_sig       : std_logic_vector(63 downto 0);
175 --</debug>

177 signal seven_seg_sig     : std_logic_vector(15 downto 0);
178 signal decimal_point_sig : bit_vector(3 downto 0);


181 begin
182   SIG_INC  <= sig_inc_sig;
183   RECT_OUT <= rect_out_sig;
184   NEW_THD  <= new_thd_sig;

186 --<CONVERSION: adc_value to numeric.unsigned>
187   adc_udiff_sig(11)          <= not SLV_adc_udiff_sig(11);
188   adc_udiff_sig(10 downto 0) <= unsigned(SLV_adc_udiff_sig(10 downto 0));

190   adc_hb1_sig(11)          <= not SLV_adc_hb1_sig(11);
191   adc_hb1_sig(10 downto 0) <= unsigned(SLV_adc_hb1_sig(10 downto 0));

193   adc_hb2_sig(11)          <= not SLV_adc_hb2_sig(11);
194   adc_hb2_sig(10 downto 0) <= unsigned(SLV_adc_hb2_sig(10 downto 0));
195 --</CONVERSION>

197   CLK_OUT <= clk_out_sig;
198   rxd_sig   <= RxD_PAD_I;
199   TxD_PAD_O <= txd_sig;

201 --<debug>
202   LED_0        <= rect_out_sig;
203   S_TIMER_DONE <= db_sample_timer_done_sig;
204   BUS2         <= unsigned(seven_seg_sig(11 downto 4));
205 --</debug>

207   --<MUX: STATES of SAMPLE_FSM and CALC_FSM>
208     process(SWITCH_7, db_calc_fsm_state_sig, db_sample_fsm_state_sig)
209     begin
210       if SWITCH_7 = '0' then
211         STATE_OUT <= db_sample_fsm_state_sig;
212       else
213         STATE_OUT <= db_calc_fsm_state_sig;
214       end if;
215     end process;
216   --</MUX>
217   --<MUX: ADC_VALUES and predefined samples from LUT>
218     process(SWITCH_6, db_udiff_reg, db_bus_adc_lut)
219     begin
220       if SWITCH_6 = '1' then
221         channel_0_sig <= db_bus_adc_lut;
```

```
222       else
223         channel_0_sig <= db_udiff_reg;
224       end if;
225     end process;
226   --</MUX>
227 --</debug>

229 --<CONVERSION: Signed 33 bit to unsigned 32 bit or ABS(value)>
230   process(re_S33, IM_S33)
231     begin
232       for i in 1 to 5 loop
233         re_U32(i) <= S33toU32(re_S33(i)) after 9 ns;
234         im_U32(i) <= S33toU32(im_S33(i)) after 9 ns;
235       end loop;
236     end process;
237 --</CONVERSION>

239 --z_____________________________________________CLK_DIV
240   CLK_DIV_COMP : clk_div
241     port map (
242         SYSCLK    => SYSCLK,
243         RESET     => RESET,
244         CLK       => CLK,
245         CLK_50_50 => clk_out_sig
246      );
247 --z_____________________________________________TL_ADC
248   ADC : TL_adc
249     port map (
250     SYSCLK  => SYSCLK,
251     RESET   => RESET,
252     CLK     => CLK,
253     START   => P_BUTTON_2,

255     --- ADC ---
256     BUSY          => ADC_BUSY,
257     DB            => ADC_DB,
258     CS_N          => ADC_CS_N,
259     WR_N          => ADC_WR_N,
260     RD_N          => ADC_RD_N,
261     CONVST_N      => ADC_CONVST_N,
262     NEW_DATA_O    => ADC_NEW_DATA_O,

264     DB_UDIFF_OUT => SLV_adc_udiff_sig,
265     DB_HB1_O     => SLV_adc_hb1_sig,
266     DB_HB2_O     => SLV_adc_hb2_sig,

268     --- DAC ---
269     CS_D_N        => DAC_CS_D_N,
270     WR_D_N        => DAC_WR_D_N,
271     LDAC_D_N      => DAC_LDAC_D_N,
```

```
272     RD_D_N       => DAC_RD_D_N,
273     CLR_D_N      => DAC_CLR_D_N,
274     BUF_D        => DAC_BUF_D,
275     GAIN         => DAC_GAIN,
276     A_D          => DAC_A_D,
277     DAC_REGISTER => DAC_DAC_REGISTER
278   );

280 --z_____________________________________SMART_COMPARATOR
281   SMART_COMP : smart_comparator
282     generic map(
283       RESET_VAL => RESET_VAL
284       )
285     port map(
286       --<debug>
287         DB_SWITCHVALUE => db_switchValue_sig,
288         DB_MAXVALUE    => db_maxValue_sig,
289         DB_MINVALUE    => db_minValue_sig,
290       --</debug>
291         SYSCLK   => SYSCLK,
292         CLK      => CLK,
293         RESET    => RESET,
294         ADC_IN   => adc_udiff_sig,
295         RECT_OUT => rect_out_sig,
296         SIG_INC  => sig_inc_sig
297     );

299 --z_____________________________________________DR_DETECTION
300     -- DR_DETECTION_COMP : dr_detection
301     -- generic map(
302       -- RESET_VAL => RESET_VAL
303     -- )
304     -- port map(
305       -- SYSCLK   => SYSCLK,
306       -- CLK      => CLK,
307       -- RESET    => RESET,
308       -- SIG_INC  => sig_inc_sig,
309       -- HB1      => adc_hb1_sig,
310       -- HB2      => adc_hb2_sig,
311       -- DR       => DR,
312       -- DV       => DV
313     -- );

315 --z_____________________________________________TL_SAMPLE
316   SAMPLE : TL_sample
317   port map(
318     --<debug>
319       DB_FSM_STATE         => db_sample_fsm_state_sig,
320       DB_SAMPLE_TIMER_DONE => db_sample_timer_done_sig,
321       DB_SET               => DB_SET,
```

```
322       DB_DONE               => DB_DONE,
323       DB_BUS_SAMPLE_FSM     => open,
324       DB_UDIFF_REG          => db_udiff_reg,
325       DB_BUS_ADC_LUT        => db_bus_adc_lut,
326       DB_BUS_LUT_ACCESS     => open,
327       DB_BUS_LUT            => open,
328       DB_BUS_MUL_ADD        => channel_5_sig,
329       DB_BUS_MULTIPLY       => channel_4_sig,
330       SWITCH_6              => SWITCH_6,
331     --</debug>
332     --regular
333       SYSCLK         => SYSCLK,
334       CLK            => CLK,
335       RESET          => RESET,
336       SIG_INC        => sig_inc_sig,
337       ADC_UDIFF      => adc_udiff_sig,
338       NEW_DATA       => new_data_sig,
339       DATA_ERROR     => data_error_sig,
340       RE             => re_S33,
341       IM             => im_S33
342   );

344 --z_____________________________________________TL_CALC_THD
345   CALC : TL_calc_thd
346     port map(
347     --<debug>
348       DB_CALC_IN_REG      => open,
349       DB_CALC_FSM_STATE   => db_calc_fsm_state_sig,
350       DB_CALC_FSM         => open,
351       DB_ADD_SAMPLES      => channel_7_sig,
352       DB_SQUARE           => channel_6_sig,
353       DB_CALC_DIV         => channel_8_sig,
354       DB_CALC_SQRT        => channel_9_sig,
355       DB_SCALE_DOWN       => channel_A_sig,
356       DB_CALC_OUT_REG     => open,
357       DB_8_BYTES_OUT       => db_8_bytes_out_sig,
358     --</debug>
359       SYSCLK         => SYSCLK,
360       CLK            => CLK,
361       RESET          => RESET,

363       NEW_DATA       => new_data_sig,
364       DATA_ERROR     => data_error_sig,

366       RE             => re_U32,
367       IM             => im_U32,

369       NEW_THD        => new_thd_sig,
370       THD_ERROR      => THD_ERROR,
371       THD            => thd_sig,
```

```
373       THD_OUT        => THD_OUT
374      );

376 --<assignment: debug vectors to MUX inputs>
377   channel_1_sig(15 downto 12) <= "0000";
378   channel_1_sig(11 downto 0)  <= db_switchValue_sig;
379   channel_2_sig(15 downto 12) <= "0000";
380   channel_2_sig(11 downto 0)  <=  db_maxValue_sig;
381   channel_3_sig(15 downto 12) <= "0000";
382   channel_3_sig(11 downto 0)  <= db_minValue_sig;
383   channel_B_sig(15 downto 12)  <= (others => '0');
384   channel_B_sig(11)  <= to_stdulogic(new_thd_sig);
385   channel_B_sig(10)  <= txd_sig;
386   channel_B_sig( 9)  <= rxd_sig;
387   channel_B_sig( 8)  <= '0';
388   channel_B_sig( 7 downto 0)  <= db_rs232(7 downto 0);
389   -- channel_B_sig( 3 downto 0)  <= (others => '0');
390   channel_C_sig(15 downto 0)  <= (others => '0');
391   channel_D_sig(15 downto 0)  <= (others => '0');
392   channel_E_sig(15 downto 0)  <= (others => '0');
393   channel_F_sig(15 downto 0)  <= std_logic_vector(thd_sig);
394 --</assignment>

396 --z_____________________________________SEVEN_SEG_MUX
397     SEVEN_SEG_MUX_COMP : seven_seg_mux
398     port map(
399     SWITCHES          => SWITCHES_0_3,

401     CHANNEL_0         => channel_0_sig,    --UDIFF_REG
402     CHANNEL_1         => channel_1_sig,    --SWITCHVALUE
403     CHANNEL_2         => channel_2_sig,    --MAXVALUE
404     CHANNEL_3         => channel_3_sig,    --MINVALUE
405     CHANNEL_4         => channel_4_sig,    --multiply
406     CHANNEL_5         => channel_5_sig,    --mul_add
407     CHANNEL_6         => channel_6_sig,    --SQR
408     CHANNEL_7         => channel_7_sig,    --add_samples
409     CHANNEL_8         => channel_8_sig,    --DIV
410     CHANNEL_9         => channel_9_sig,    --SQRT
411     CHANNEL_A         => channel_A_sig,    --scale down
412     CHANNEL_B         => channel_B_sig,
413     -- CHANNEL_C         => channel_C_sig,
414     -- CHANNEL_D         => channel_D_sig,
415     -- CHANNEL_E         => channel_E_sig,
416     CHANNEL_F         => channel_F_sig,    --THD_SIG

418     DECIMAL_POINT_CH0 => "1111",
419     DECIMAL_POINT_CH1 => "1111",
420     DECIMAL_POINT_CH2 => "1111",
421     DECIMAL_POINT_CH3 => "1111",
```

```
422     DECIMAL_POINT_CH4 => "1111",
423     DECIMAL_POINT_CH5 => "1111",
424     DECIMAL_POINT_CH6 => "1111",
425     DECIMAL_POINT_CH7 => "1111",
426     DECIMAL_POINT_CH8 => "1111",
427     DECIMAL_POINT_CH9 => "1111",
428     DECIMAL_POINT_CHA => "1111",
429     DECIMAL_POINT_CHB => "1111",
430     -- DECIMAL_POINT_CHC => "1111",
431     -- DECIMAL_POINT_CHD => "1111",
432     -- DECIMAL_POINT_CHE => "1111",
433     DECIMAL_POINT_CHF => "1011",

435     CHANNEL_OUT       => seven_seg_sig,
436     DECIMAL_POINT_OUT => decimal_point_sig
437     );

439 --z_____________________________________________SEVEN_SEG
440     SEV_SEG : seven_seg
441     port map(
442     SYSCLK            => SYSCLK,
443     CLK               => CLK,
444     RESET             => RESET,

446     BCD_ON            => SWITCH_5,

448     SEVEN_SEG_IN      => seven_seg_sig,
449     DECIMAL_POINT_IN  => decimal_point_sig,
450     NEW_THD           => new_thd_sig,

452     SEVEN_SEG_out     => SEVEN_SEG_out,
453     DECIMAL_POINT     => DECIMAL_POINT,
454     SEVEN_SEG_SELECT  => SEVEN_SEG_SELECT
455     );

457 --z_____________________________________TL_RS232_MiniUART
458   RS232: TL_RS232_MiniUART
459   port map(
460   --<debug>
461   DB_RS232    => db_rs232,
462   --</debug>
463   SYS_CLK_S   => to_stdulogic(SYSCLK),
464   CLK         => to_stdulogic(CLK),
465   RESET       => to_stdulogic(RESET),

467   NEW_THD     => new_thd_sig,
468   THD_IN      => std_logic_vector(thd_sig),
469     ---- MiniUART ----
470   TxD_PAD_O   =>  txd_sig,     -- Tx RS232 Line
471   RxD_PAD_I   =>  rxd_sig      -- Rx RS232 Line
```

```
472     -------------------
473   );

475 end structure;
```

B.1.2 global_pkg.vhd

Alle Pakete wurden durch `pkg_maker.py` (Siehe B.2, S. 181 erzeugt, deshalb hier nur das globale Paket.

```
1  ----------------------------------------
2  ---------GLOBAL PACKAGE--------------------
3  ----------------------------------------
4  -------------------gererated by pkg_maker.py

6  library ieee;
7     use ieee.std_logic_1164.all;
8     use ieee.numeric_std.all;

10 package global_pkg is


13 --- CONSTANTS ---
14   constant DR_ADC_WIDTH : integer := 12;  --dr_detection.vhd
15   constant Q15_MSB : integer := 15; --functions.vhd
16   constant LUT_WIDTH_IN  : natural := 4;  --LUT.vhd
17   constant LUT_WIDTH_OUT : natural := 12; --LUT.vhd
18   constant PERIOD_TIMER_WIDTH : integer := 24;  --period_timer.vhd
19   constant PERIOD_TIMER_MSB   : natural := 23;  --period_timer.vhd
20   constant ADC_WIDTH : integer := 12; --smart_comparator.vhd
21   constant RESET_VAL : bit := '1';  --TL_adc_sample_calc_thd.vhd

23 --- TYPES ---
24   type db_lut_array is array(0 to 63) of unsigned(11 downto 0); --adc_db_lut.vhd
25   type CALC_STATE_TYPE is (POR, IDLE, SET_IN_REG_STATE, --calc_fsm.vhd
26         SET_ADD_STATE, ADD_DONE_STATE, ADD_ALL_DONE_STATE,
27         SET_CALC_DIV_STATE, SET_CALC_SQRT_STATE,
28         SET_SCALE_DOWN_STATE, SUCCESS, FAIL);
29   type Tarray is array(1 to 15) of signed(11 downto 0); --LUT.vhd
30   type SAMPLE_STATE_TYPE is (FIRST_PERIOD,  --sample_fsm.vhd
31         WAIT_FOR_SIG_INC_LOW, WAIT_FOR_SIG_INC_HIGH,
32         WAIT_FOR_SAMPLE_TIMER, LUT_ENABLE_STATE,
33         LUT_DONE_STATE, MUL_ADD_SET_STATE,
34         MUL_ADD_DONE_STATE,NEW_DATA_STATE,
35         WAIT_FOR_SAMPLE_TIMER_AFTER_LAST_SAMPLE,
36         ACCELERATION_ERROR, DECELERATION_ERROR);
37   subtype U32 is unsigned(31 downto 0); --TL_adc_sample_calc_thd.vhd
```

```
38   subtype S33 is signed(32 downto 0);   --TL_adc_sample_calc_thd.vhd
39   type MulArrayS is array(1 to 5) of S33; --TL_adc_sample_calc_thd.vhd
40   type MulArray is array(1 to 5) of U32;  --TL_adc_sample_calc_thd.vhd

42 --- MISC ---









52 end global_pkg;
```

B.1.3 NEXYS2_CONN3u4.ucf

```
2  NET "SYSCLK"    LOC = "B8";
3  #NET "RESET"     LOC = "B1";


6  # 7 segment display
7  NET "SEVEN_SEG_out<0>" LOC = "L18";
8  NET "SEVEN_SEG_out<1>" LOC = "F18";
9  NET "SEVEN_SEG_out<2>" LOC = "D17";
10 NET "SEVEN_SEG_out<3>" LOC = "D16";
11 NET "SEVEN_SEG_out<4>" LOC = "G14";
12 NET "SEVEN_SEG_out<5>" LOC = "J17";
13 NET "SEVEN_SEG_out<6>" LOC = "H14";
14 NET "DECIMAL_POINT"      LOC = "C17";
15 # 7 Segment MUX control
16 NET "SEVEN_SEG_SELECT<0>" LOC = "F17";
17 NET "SEVEN_SEG_SELECT<1>" LOC = "H17";
18 NET "SEVEN_SEG_SELECT<2>" LOC = "C18";
19 NET "SEVEN_SEG_SELECT<3>" LOC = "F15";

21 # Leds
22 NET "RECT_OUT"  LOC = "J14";  #LED0
23 # NET "DV"                    #LED1
24 # NET "DR"                    #LED2
25 NET "THD_ERROR"  LOC = "K14"; #LED3
26 NET "NEW_THD" LOC = "E16";    #LED4
27 NET "THD_OUT<0>" LOC = "P16"; #LED5
28 NET "THD_OUT<1>" LOC = "E4";  #LED6
29 NET "THD_OUT<2>" LOC = "P4";  #LED7
```

```
31 # Switches
32 NET "SWITCHES_0_3<0>" LOC = "G18"; #debug mux control 0
33 NET "SWITCHES_0_3<1>" LOC = "H18"; #debug mux control 1
34 NET "SWITCHES_0_3<2>" LOC = "K18"; #debug mux control 2
35 NET "SWITCHES_0_3<3>" LOC = "K17"; #debug mux control 3
36 # NET "SWITCH_4" LOC = "L14";
37 NET "SWITCH_5" LOC = "L13";   #BCD on/off
38 NET "SWITCH_6" LOC = "N17";   #adc_udiff_reg/adc_lut_value
39 NET "SWITCH_7" LOC = "R17";   #sample states/calc states

41 # Buttons

43 # NET "P_BUTTON_0" LOC = "B18";
44 # NET "P_BUTTON_1" LOC = "D18";
45 NET "P_BUTTON_2" LOC = "E18"; # Start ADC
46 NET "RESET" LOC = "H13";      # Reset



49 #CON: JB1
50 NET "BUS2<0>" LOC = "M13";
51 NET "BUS2<1>" LOC = "R18";
52 NET "BUS2<2>" LOC = "R15";
53 NET "BUS2<3>" LOC = "T17";
54 NET "BUS2<4>" LOC = "P17";
55 NET "BUS2<5>" LOC = "R16";
56 NET "BUS2<6>" LOC = "T18";
57 NET "BUS2<7>" LOC = "U18";
58 #
59 #CON: JA1
60 NET "SIG_INC" LOC = "K13";
61 NET "DB_SET" LOC = "L16";
62 NET "S_TIMER_DONE" LOC = "M14";
63 NET "DB_DONE" LOC = "M16";
64 NET "STATE_OUT<0>" LOC = "L15";
65 NET "STATE_OUT<1>" LOC = "K12";
66 NET "STATE_OUT<2>" LOC = "L17";
67 NET "STATE_OUT<3>" LOC = "M15";

69 #ADC SIGNALS
70 #Output
71 NET "DAC_WR_D_N" LOC = "E9";
72 NET "DAC_RD_D_N" LOC = "C9";

74 NET "DAC_CS_D_N" LOC = "A8";
75 NET DAC_GAIN LOC = "G9";
76 NET DAC_BUF_D LOC = "F9";
77 NET DAC_A_D<0> LOC = "D10";
78 #
79 #Input
80 NET DAC_A_D<1> LOC = "A10";
```

```
81  NET DAC_A_D<2> LOC = "B10";
82  NET ADC_BUSY LOC = "A11";      # PIN 18
83  NET ADC_WR_N LOC = "D11";
84  NET ADC_RD_N LOC = "E10";
85  NET ADC_CS_N LOC = "B11";
86  NET CLK_OUT LOC = "C11";  # 1/4 von SYS_CLK_S
87  NET ADC_CONVST_N LOC = "E11";    # PIN 13
88  NET ADC_DB<0> LOC = "F11";
89  NET ADC_DB<1> LOC = "E12";
90  NET ADC_DB<2> LOC = "F12";
91  NET ADC_DB<3> LOC = "A13";
92  NET ADC_DB<4> LOC = "B13";
93  NET ADC_DB<5> LOC = "E13";
94  NET ADC_DB<6> LOC = "A14";
95  NET ADC_DB<7> LOC = "C14";
96  NET ADC_DB<8> LOC = "D14";
97  NET ADC_DB<9> LOC = "B14";
98  NET ADC_DB<10> LOC = "A16";      # PIN 2
99  NET ADC_DB<11> LOC = "B16";      # PIN 1

101 # RS232 connector
102 NET "RxD_PAD_I" LOC = "U6";
103 NET "TxD_PAD_O" LOC = "P9";
```

B.1.4 CLK_DIV.vhd

```
1   --#########################################################
2   --##  Project name : adc_sample_calc_thd
3   --##  File name    : clk_div.vhd
4   --##  Author       : Jan-Heiner Dreschhoff

6   -- <makepackage>
7   --      <component = CLK_DIV>
8   --      <name = dreschhoff>
9   --      <project = ada_conv_modsys>
10  --       </component>
11  -- </makepackage>

13  library IEEE;
14    use IEEE.std_logic_1164.all;
15    use IEEE.numeric_std.all;
16  library work;
17    use work.global_pkg.all;

19  entity CLK_DIV is
20    port (
21      SYSCLK     : in bit;
22      RESET      : in bit;
```

```
23      CLK       : out bit;
24      CLK_50_50 : out bit
25      );
26  end CLK_DIV;

28  architecture behavior of CLK_DIV is

30  signal counter : unsigned(1 downto 0);

32  begin

34    process(counter)
35      variable bit_counter : bit_vector(1 downto 0);
36    begin
37      bit_counter := to_bitvector(std_logic_vector(counter));
38      CLK_50_50 <= bit_counter(1) after 9 ns;
39      CLK       <= bit_counter(1) nor bit_counter(0) after 9 ns;
40    end process;

42    process(SYSCLK,RESET)
43    begin
44      if RESET = RESET_VAL then
45        counter <= "11" after 9 ns;
46      elsif SYSCLK = '1' and SYSCLK'event then
47        counter <= counter + 1 after 9 ns;
48      end if;
49    end process;

51  end behavior;
```

B.1.5 TL_adc.vhd

```
1  ------------------------------------------------------------
2  --  Toplevel for testing ada_conv_modsys-board
3  --
4  --  Author: A.Arvidsson
5  ------------------------------------------------------------

7  -- <makepackage>
8  --      <component = TL_adc>
9  --      <name = arvidsson>
10 --      <project = adc_sample_calc_thd>
11 --       </component>
12 -- </makepackage>


15 library ieee;
16   use ieee.std_logic_1164.all;
```

```
17  use ieee.numeric_std.all;



21  library work;
22  use work.global_pkg.all;
23  use work.adc_sig_mkr_pkg.all;
24  use work.adc_contrl_fsm_pkg.all;
25  use work.adc_datapath_pkg.all;


28  entity TL_adc is
29  port (
30    SYSCLK : in bit; -- Systemtakt
31    RESET : in bit;  -- allgemeiner Resets
32    CLK   : in bit;
33    START : in bit;

35    --- ADC ---
36    --READ_ADC           : in bit;
37    BUSY               : in bit;
38    DB                 : inout std_logic_vector(11 downto 0);
39    CS_N, WR_N, RD_N   : out bit;
40    CONVST_N           : out std_logic;
41    NEW_DATA_O         : out bit_vector(1 downto 0);  -- new data is available
42    DB_UDIFF_OUT       : out std_logic_vector(11 downto 0); -- ADC-Conv_Ausgabe
43    DB_HB1_O, DB_HB2_O : out std_logic_vector(11 downto 0); -- ADC-Conv_Ausgabe
44    --CLK_OUT            : out std_logic; -- ADC-Takt


47    --- DAC ---
48    --WRITE_DAC    : in  bit;
49    CS_D_N       : out bit;
50    WR_D_N       : out bit;
51    LDAC_D_N     : out bit;
52    RD_D_N       : out bit;
53    CLR_D_N      : out bit;
54    BUF_D        : out bit;
55    GAIN         : out bit;
56    A_D          : out bit_vector(2 downto 0);
57    DAC_REGISTER : out std_logic_vector(11 downto 0)
58    -------
59  );
60  end TL_adc;

62  architecture structure of TL_adc is
63  signal CNT_RESET       : bit;
64  signal CNT             : unsigned(3 downto 0);
65  signal WR_N_EN         : bit;
66  signal DAC             : bit;
```

```
67  signal DAC_PROG        : bit;
68  signal PART            : bit;
69  signal CONV_CH         : bit;
70  --signal CLK, CLK_O      : std_logic;
71  signal DB_UDIFF        : std_logic_vector(11 downto 0);
72  signal DB_HB1          : std_logic_vector(11 downto 0);
73  signal DB_HB2          : std_logic_vector(11 downto 0);
74  -- signal SYS_CLK_S       : std_logic;
75  -- signal T              : std_logic;
76  -- signal T_O            : std_logic;
77  signal READ_ACCESS_ADC : bit;
78  signal READ_ACCESS_DAC : bit;
79  signal READ_ADC_DAC    : bit;
80  signal ReadNow         : bit;
81  signal NEW_DATA        : bit_vector(1 downto 0);

83  -- component BUFG
84     -- port (O : out std_logic;
85           -- I : in std_logic);
86  -- end component;


89  begin
90    -- inverting system-CLK
91    -- SYS_CLK_S   <= not NSYS_CLK_S;
92    -- outgoing
93    DB_UDIFF_OUT <= DB_UDIFF;
94    DB_HB1_O     <= DB_HB1;
95    DB_HB2_O     <= DB_HB2;
96    --CLK_OUT      <= CLK;
97    NEW_DATA_O   <= NEW_DATA;

99    I_SIG : adc_sig_mkr
100     port map (
101       SYSCLK    => SYSCLK,
102       CLK       => CLK, -- T_O,
103       RESET     => RESET,
104       CNT_RESET => CNT_RESET,
105       -- BUSY       => BUSY,
106       CNT       => CNT
107     );
108   I_FSM : adc_contrl_fsm
109     port map (
110       SYSCLK   => SYSCLK,
111       CLK      => CLK,
112       RESET    => RESET,
113       START    => START,
114       --READ_ADC => READ_ADC,
115       BUSY     => BUSY,
```

```
117       --- ADC ---
118       CONVST_N         => CONVST_N,
119       CNT              => CNT,
120       CNT_RESET        => CNT_RESET,
121       WR_N_EN          => WR_N_EN,
122       PART             => PART,
123       CONV_CH          => CONV_CH,
124       CS_N             => CS_N,
125       WR_N             => WR_N,
126       RD_N             => RD_N,
127       ReadNow          => ReadNow,
128       DAC              => DAC,
129       READ_ACCESS_ADC  => READ_ACCESS_ADC,
130       READ_ADC_DAC     => READ_ADC_DAC,

132       --- DAC ---
133       --WRITE_DAC       => WRITE_DAC,
134       CS_D_N           => CS_D_N,
135       WR_D_N           => WR_D_N,
136       LDAC_D_N         => LDAC_D_N,
137       RD_D_N           => RD_D_N,
138       CLR_D_N          => CLR_D_N,
139       BUF_D            => BUF_D,
140       GAIN             => GAIN,
141       DAC_PROG         => DAC_PROG,
142       READ_ACCESS_DAC  => READ_ACCESS_DAC
143     );
144   I_DATA : adc_datapath
145     port map (
146       SYSCLK           => SYSCLK,
147       CLK              => CLK,
148       RESET            => RESET,
149       WR_N_EN          => WR_N_EN,
150       PART             => PART,
151       CONV_CH          => CONV_CH,
152       DAC              => DAC,
153       READ_ACCESS_ADC  => READ_ACCESS_ADC,
154       --READ_ACCESS_DAC => READ_ACCESS_DAC,
155       READ_ADC_DAC     => READ_ADC_DAC,
156       ReadNow          => ReadNow,
157       DAC_PROG         => DAC_PROG,
158       DB               => DB,
159       NEW_DATA         => NEW_DATA,
160       DB_UDIFF         => DB_UDIFF,
161       DB_HB1           => DB_HB1,
162       DB_HB2           => DB_HB2,
163       DAC_REGISTER     => DAC_REGISTER,
164       A_D              => A_D
165     );
```

```
167 end structure;
```

B.1.5.1 adc_sig_mkr.vhd

```
1  ------------------------------------------------------------
2  --  Clock-creator for testing
3  --
4  --  Author: A.Arvidsson
5  ------------------------------------------------------------

7  -- <makepackage>
8  --      <component = adc_sig_mkr>
9  --      <name = arvidsson>
10 --      <project = adc_sample_calc_thd>
11 --       </component>
12 -- </makepackage>

14 library ieee;
15   use ieee.std_logic_1164.all;
16   use ieee.numeric_std.all;
17 library work;
18   use work.global_pkg.all;


21 entity adc_sig_mkr is
22   port(
23     SYSCLK    : in bit;
24     CLK       : in bit;
25     RESET     : in bit;
26     CNT_RESET : in bit;
27     CNT       : out unsigned(3 downto 0)
28     );
29 end adc_sig_mkr;

31 architecture behavior of adc_sig_mkr is
32   signal counter : unsigned(3 downto 0);
33 begin

35   COUNT : process(SYSCLK,RESET)
36   begin
37     if RESET = RESET_VAL then
38       counter <= x"0" after 9 ns; -- zur Synchronisierung
39     elsif SYSCLK = '1' and SYSCLK'event then
40       if CLK = '1' then
41         if CNT_RESET = '1' then -- normaler Zähler
42           counter <= x"0" after 9 ns;
43         else
44           counter <= counter + 1 after 9 ns;
```

```
45          end if;
46        end if;
47      end if;
48    end process COUNT;

50    CNT <= counter;

52  end behavior;
```

B.1.5.2 adc_cntrl_fsm.vhd

```
1   ------------------------------------------------------------
2   --  Control-FSM for signal-control
3   --
4   --  Author: A.Arvidsson
5   --  modified by dreschhoff
6   ------------------------------------------------------------

8   -- <makepackage>
9   --      <component = adc_contrl_fsm>
10  --      <name = arvidsson>
11  --      <project = adc_sample_calc_thd>
12  --       </component>
13  -- </makepackage>

15  library ieee;
16    use ieee.std_logic_1164.all;
17    use ieee.numeric_std.all;
18  library work;
19    use work.global_pkg.all;

21  entity adc_contrl_fsm is
22    port(
23      SYSCLK : in bit;
24      CLK    : in bit;
25      RESET : in bit;
26      START : in bit;

28      -- write_en : out bit;
29      --- ADC ---
30      --READ_ADC          : in bit;
31      BUSY              : in bit;
32      CONVST_N          : out std_logic; -- Start-Bit
33      CNT               : in unsigned(3 downto 0); -- actual count
34      CNT_RESET         : out bit; -- count reset
35      WR_N_EN           : out bit; -- Write-enable
36      PART              : out bit; -- Config-/Sequencer-Register-Auswahl
37      CONV_CH           : out bit; -- channel which is conversioned
```

```
38      CS_N, WR_N, RD_N : out bit;
39      ReadNow          : out bit;
40      DAC              : out bit;
41      READ_ACCESS_ADC  : out bit;
42      READ_ADC_DAC     : out bit;
43      -----------
44      --- DAC ---
45      --WRITE_DAC       : in bit;
46      CS_D_N           : out bit;
47      WR_D_N           : out bit;
48      LDAC_D_N         : out bit;
49      RD_D_N           : out bit;
50      CLR_D_N          : out bit;
51      BUF_D            : out bit;
52      GAIN             : out bit;
53      DAC_PROG         : out bit;
54      READ_ACCESS_DAC : out bit
55      -----------
56    );
57  end adc_contrl_fsm;

59  architecture FSM of adc_contrl_fsm is
60  type STATE_TYPE is (STANDBY,READY,
61                      IDLE,DAC_REG,DAC_PAUSE,DAC_WR,
62                      DAC_WAIT,S0,STATE_TEST,S1,S2,
63                      S2_BUSY,S3_BUSY,S3,S4,S5,S6);
64  signal STATE, NEXT_STATE : STATE_TYPE;
65  signal CONV_CNT : bit;

67  begin

69    REG : process(SYSCLK,RESET)
70    begin
71      if RESET = RESET_VAL then
72        STATE <= STANDBY after 9 ns;
73      elsif SYSCLK = '1' and SYSCLK'event then
74        if CLK = '1' then
75          STATE <= NEXT_STATE after 9 ns;
76        end if;
77      end if;
78    end process REG;

80     READ_CONV : process(CNT,STATE,RESET,START,BUSY)
81    variable CONV_CHANNEL : bit;
82    begin
83      ---------- ADC --------------------------
84      CS_N <= '1' after 9 ns; -- defaults
85      WR_N <= '1' after 9 ns;
86      WR_N_EN <= '0' after 9 ns;
87      RD_N <= '1' after 9 ns;
```

```
88      PART <= '0' after 9 ns;
89      CNT_RESET <= '0' after 9 ns;
90      CONV_CHANNEL := '0';
91      CONV_CNT <= '1' after 9 ns;
92      DAC <= '0' after 9 ns;
93      READ_ACCESS_ADC <= '0' after 9 ns;
94      READ_ADC_DAC <= '0' after 9 ns;
95      ReadNow <= '0' after 9 ns;
96      CONVST_N <= '1' after 9 ns;
97      ----------------------------------------

99      ------------ DAC ------------
100     CS_D_N <= '1' after 9 ns;
101     WR_D_N <= '1' after 9 ns;
102     RD_D_N <= '1' after 9 ns;
103     LDAC_D_N <= '1' after 9 ns;
104     CLR_D_N <= '1' after 9 ns;
105     BUF_D <= '0' after 9 ns;
106     GAIN <= '1' after 9 ns;
107     DAC_PROG <= '0' after 9 ns;
108     READ_ACCESS_DAC <= '0' after 9 ns;
109     ----------------------------

111     if RESET = '1' then
112       NEXT_STATE <= STANDBY after 9 ns;
113       CNT_RESET <= '1' after 9 ns;
114       CONV_CHANNEL := '0';
115       CONV_CNT <= '1' after 9 ns;
116     end if;

118     case STATE is
119     when STANDBY =>
120       if START = '1' then
121         NEXT_STATE <= READY after 9 ns;
122       else
123         NEXT_STATE <= STANDBY after 9 ns;
124       end if;
125     when READY =>
126         NEXT_STATE <= IDLE after 9 ns;
127     -------------------- ADC ------------------------------------------------
128     when IDLE =>  -- Initialisierung beim ersten Start
129       CNT_RESET <= '1' after 9 ns;
130       NEXT_STATE <= DAC_REG after 9 ns;
131     when DAC_REG => -- schreibe in die verschiedenen Register
132       if CNT = x"A" then
133         CS_N <= '0' after 9 ns;
134         WR_N <= '0' after 9 ns;
135         WR_N_EN <= '1' after 9 ns;
136         DAC <= '1' after 9 ns;
137         CONV_CHANNEL := '0';
```

```
138         NEXT_STATE <= DAC_PAUSE after 9 ns;
139       else
140         NEXT_STATE <= DAC_REG after 9 ns;
141       end if;
142     when DAC_PAUSE =>   -- SchreibePause muss eingehalten werden!!
143                 -- Zwei Schreibzyklen ohne WR_N = LOW nicht möglich!!
144       CONV_CHANNEL := '0';
145       NEXT_STATE <= DAC_WR after 9 ns;
146     when DAC_WR =>
147       if CNT = x"C" then
148         CS_N <= '0' after 9 ns;
149         WR_N <= '0' after 9 ns;
150         DAC <= '1' after 9 ns;
151         CONV_CHANNEL := '1';
152         WR_N_EN <= '1' after 9 ns;
153         NEXT_STATE <= DAC_WAIT after 9 ns;
154       else
155         NEXT_STATE <= DAC_WR after 9 ns;
156       end if;
157     when DAC_WAIT =>    -- SchreibePause muss eingehalten werden!!
158                 -- Zwei Schreibzyklen ohne WR_N = LOW nicht möglich!!
159       CONV_CHANNEL := '0';
160       NEXT_STATE <= S0 after 9 ns;  --S0
161     when S0 =>
162       if CNT = x"A" then
163         CS_N <= '0' after 9 ns;
164         WR_N <= '0' after 9 ns;
165         WR_N_EN <= '1' after 9 ns;
166         CONV_CHANNEL := '0';
167         NEXT_STATE <= STATE_TEST after 9 ns;
168       else
169         NEXT_STATE <= S0 after 9 ns;
170       end if;
171     when STATE_TEST =>    -- SchreibePause muss eingehalten werden!!
172                   -- Zwei Schreibzyklen ohne WR_N = LOW nicht möglich!!
173       CONV_CHANNEL := '0';
174       NEXT_STATE <= S1 after 9 ns;  --S1
175     when S1 =>
176       if CNT = x"C" then
177         CS_N <= '0' after 9 ns;
178         WR_N <= '0' after 9 ns;
179         WR_N_EN <= '1' after 9 ns;
180         CONV_CHANNEL := '1';  -- write 2. value into register
181         NEXT_STATE <= S2 after 9 ns;  --S2
182       else
183         NEXT_STATE <= S1 after 9 ns;
184       end if;
185     ------------- start conversion and reading ---------
186     when S2 =>
187         NEXT_STATE <= S2_BUSY after 9 ns;
```

```
188         CONV_CNT <= '0';
189         CONVST_N <= '0' after 9 ns; -- start conversion
190       when S2_BUSY =>
191         if BUSY = '0' then
192           NEXT_STATE <= S3_BUSY after 9 ns;
193         else
194           NEXT_STATE <= S2_BUSY after 9 ns;
195         end if;
196       ---------------------------------------- LESEN -----
197       when S3_BUSY =>
198         if BUSY = '1' then
199           CNT_RESET <= '1' after 9 ns;
200           NEXT_STATE <= S3 after 9 ns;
201         else
202           CONVST_N <= '0' after 9 ns;
203           NEXT_STATE <= S3_BUSY after 9 ns;
204         end if;
205       when S3 =>  -- Werte aus dem ADC auslesen
206         if CNT <= x"2" then
207           CS_N <= '0' after 9 ns;
208           RD_N <= '0' after 9 ns;
209           ReadNow <= '1' after 9 ns;
210           PART <= '0' after 9 ns;  -- choose right Register
211                                    -- to save data in (1.part)
212           if CONV_CNT = '0' then  --beim ersten Durchlauf wird getoggelt
213             CONV_CHANNEL := not CONV_CHANNEL; -- toggeln
214             CONV_CNT <= '1';
215           end if;
216           NEXT_STATE <= S4 after 9 ns;
217         else
218           NEXT_STATE <= S3 after 9 ns;
219         end if;
220       when S4 =>
221           NEXT_STATE <= S5 after 9 ns;
222       when S5 =>
223         if CNT = x"4" then
224           CS_N <= '0' after 9 ns;
225           RD_N <= '0' after 9 ns;
226           ReadNow <= '1' after 9 ns;
227           PART <= '1' after 9 ns;  -- choose right Register
228                                    -- to save data in(2.part)
229           NEXT_STATE <= S6 after 9 ns;
230         else
231           NEXT_STATE <= S5 after 9 ns;
232         end if;
233       when S6 =>
234         NEXT_STATE <= S2_BUSY after 9 ns; -- Sequence oder STANDBY
235         CONV_CNT <= '0';
236       end case;
```

```
238     CONV_CH <= CONV_CHANNEL;

240   end process READ_CONV;

242 end FSM;
```

B.1.5.3 adc_datapath.vhd

```
1  ------------------------------------------------------------
2  --  Datapath for reading and writing data
3  --
4  --  Author: A.Arvidsson
5  ------------------------------------------------------------

7  -- <makepackage>
8  --      <component = adc_datapath>
9  --      <name = arvidsson>
10 --      <project = adc_sample_calc_thd>
11 --      </component>
12 -- </makepackage>

14 library ieee;
15   use ieee.std_logic_1164.all;
16   use ieee.numeric_std.all;

18 library work;
19   use work.global_pkg.all;

21 entity adc_datapath is
22   port(
23     SYSCLK          : in bit;
24     CLK             : in bit;
25     RESET           : in bit;
26     WR_N_EN         : in bit;
27     PART            : in bit;
28     CONV_CH         : in bit;
29     DAC             : in bit;
30     READ_ACCESS_ADC : in bit;
31     --READ_ACCESS_DAC : in bit;
32     READ_ADC_DAC    : in bit;
33     ReadNow         : in bit;
34     DAC_PROG        : in bit; --- DAC ---
35     DB              : inout std_logic_vector(11 downto 0);
36     NEW_DATA        : out bit_vector(1 downto 0);
37     DB_UDIFF        : out std_logic_vector(11 downto 0);
38     DB_HB1          : out std_logic_vector(11 downto 0);
39     DB_HB2          : out std_logic_vector(11 downto 0);
40     --- DAC ---
```

```
41      DAC_REGISTER : out std_logic_vector(11 downto 0);
42      A_D          : out bit_vector(2 downto 0)
43      );
44  end adc_datapath;

46  architecture DATA of adc_datapath is
47  constant DB_CHx0    : std_logic_vector(11 downto 0) := "000100000000";  -- x"100"
48  constant DB_CHx1    : std_logic_vector(11 downto 0) := "110100000000";  -- x"D00"
49  constant DB_SEQ     : std_logic_vector(11 downto 0) := x"104";  -- ins Sequence-Regis
50  constant DB_SEQ_CTR : std_logic_vector(11 downto 0) := x"230";  -- 1xCONVST_N, jeder

52  signal X_NEW_DATA : bit_vector(1 downto 0); -- internes NEW_DATA-Signal
53  signal WriteTo  : std_logic_vector(11 downto 0);
54  signal ReadFrom : std_logic_vector(11 downto 0);
55  signal WriteNow : bit;

57  signal UDIFF : std_logic_vector(11 downto 0);
58  signal HB1 : std_logic_vector(11 downto 0);
59  signal HB2 : std_logic_vector(11 downto 0);


62  begin

64    REG_DATA : process(ReadNow,CONV_CH,PART,ReadFrom,WR_N_EN,DAC)
65    begin
66      -- if RESET = '1' then
67        UDIFF <= x"000" after 9 ns; -- default
68        HB1 <= x"000" after 9 ns;
69        HB2 <= x"000" after 9 ns;
70        X_NEW_DATA <= "10" after 9 ns; -- default (nicht vergeben)
71        DAC_REGISTER <= x"000" after 9 ns;
72        WriteTo <= (others => '0') after 9 ns; -- with '0' because it is the write proc
73        A_D <= "000" after 9 ns;
74        WriteNow <= '0' after 9 ns;
75        if ReadNow = '1' then
76          if CONV_CH = '0' then
77            if PART = '0' then  -- PART = '0' ist 1.Teil vom Konvertieren, '1' der 2.Te
78              UDIFF <= ReadFrom after 9 ns;
79            -- end if;
80              X_NEW_DATA <= "00" after 9 ns;
81            elsif PART = '1' then
82              HB1 <= ReadFrom after 9 ns;
83              X_NEW_DATA <= "01" after 9 ns;
84            end if;
85          elsif CONV_CH = '1' then
86            if PART = '1' then
87              HB2 <= ReadFrom after 9 ns;
88              X_NEW_DATA <= "11" after 9 ns;
89            end if;
90          end if;
```

```
91        end if;
92        -- set values to ADC and DAC
93        if WR_N_EN = '1' then
94          WriteNow <= '1' after 9 ns;
95          if DAC = '1' then -- write to DAC of ADC
96            if CONV_CH = '0' then
97              WriteTo <= x"101" after 9 ns; -- goto DAC-register of ADC
98            elsif CONV_CH = '1' then
99              WriteTo <= x"3FF" after 9 ns; -- write value
100           end if;
101         elsif DAC = '0' then  -- write to ADC
102           if CONV_CH = '0' then
103             WriteTo <= DB_SEQ after 9 ns; -- DB_SEQ
104           elsif CONV_CH = '1' then
105             WriteTo <= DB_SEQ_CTR after 9 ns; --DB_SEQ_CTR
106           end if;
107         end if;
108       end if;
109     -- end if;
110   end process REG_DATA;

112   NEW_DATA <= X_NEW_DATA; -- Flag/Signal für Heiner


115   OUTPUT : process (WriteNow, DB, WriteTo)    -- Behavioral representation
116   begin                                       -- of tri-states.
117     if WriteNow = '0' then
118       DB <= (others => 'Z') after 9 ns;
119       ReadFrom <= DB after 9 ns;
120     elsif WriteNow = '1' then
121       DB <= WriteTo after 9 ns;
122       ReadFrom <= DB after 9 ns;
123     end if;
124   end process OUTPUT;

126   FF_UDIFF : process (SYSCLK,RESET)
127   begin
128     if RESET = RESET_VAL then
129       DB_UDIFF <= x"000" after 9 ns;
130       DB_HB1 <= x"000" after 9 ns;
131       DB_HB2 <= x"000" after 9 ns;
132     elsif SYSCLK = '1' and SYSCLK'event then
133       if CLK = '1' then
134         if X_NEW_DATA = "00" then
135           DB_UDIFF <= UDIFF after 9 ns;
136         elsif X_NEW_DATA = "01" then
137           DB_HB1 <= HB1 after 9 ns;
138         elsif X_NEW_DATA = "11" then
139           DB_HB2 <= HB2 after 9 ns;
140         end if;   --X_NEW_DATA
```

```
141       end if;      --CLK
142     end if;        --RESET
143   end process FF_UDIFF;

145 end DATA;
```

B.1.6 smart_comparator.vhd

```
1  --#######################################################
2  --##  Project name : adc_sample_calc_thd
3  --##  File name    : smart_comparator.vhd
4  --##  Author       : Jan-Heiner Dreschhoff
5  --##
6  --##  description:
7  --##  this file contains a component, that gets a value from an ADC,
8  --##  and changes its output to control a DAC, so with the help of
9  --##  a comperator, a rectangle signal with the frequency of the
10 --##  input signal can be generated.



14 --<makepackage>
15 --  <global>
16 --    constant ADC_WIDTH : integer := 12;
17 --  </global>
18 --  <component = smart_comparator>
19 --    <name = dreschhoff>
20 --    <project = thd>
21 --  </component>
22 --</makepackage>

24 library ieee;
25   use ieee.std_logic_1164.all;
26   use ieee.numeric_std.all;
27 library work;
28    use work.global_pkg.all;

30 entity smart_comparator is
31   generic(
32       RESET_VAL : bit := '1'     -- reset is low active
33     );
34   port(
35     --<debug>
36     DB_SWITCHVALUE : out std_logic_vector(11 downto 0);
37     DB_MAXVALUE    : out std_logic_vector(11 downto 0);
38     DB_MINVALUE    : out std_logic_vector(11 downto 0);
39     --</debug>
40     SYSCLK    : in bit;
```

```
41      CLK       : in bit;
42      RESET     : in bit;
43        ADC_IN    : in unsigned((ADC_WIDTH-1) downto 0);
44      RECT_OUT : out bit;
45      SIG_INC   : out bit
46    );
47  end smart_comparator;

49  architecture behavior of smart_comparator is

51  signal switchValue       : unsigned((ADC_WIDTH-1) downto 0);
52  signal maxValue          : unsigned((ADC_WIDTH-1) downto 0);
53  signal minValue          : unsigned((ADC_WIDTH-1) downto 0);
54  signal rect_out_sig      : bit;
55  signal last_rect_out_sig : bit;

57  begin
58    --<debug>
59    DB_SWITCHVALUE <= std_logic_vector(switchValue);
60    DB_MAXVALUE    <= std_logic_vector(maxValue);
61    DB_MINVALUE    <= std_logic_vector(minValue);
62    --</debug>

64    RECT_OUT <= rect_out_sig;

66    generate_RECT_OUT : process(SYSCLK, RESET)
67    begin
68      if RESET = RESET_VAL then
69        rect_out_sig    <= '0'        after 9 ns;
70      elsif SYSCLK = '1' and SYSCLK'event then
71        if CLK = '1' then
72          if ADC_IN < switchValue then
73            rect_out_sig <= '0' after 9 ns;
74          else
75            rect_out_sig <= '1' after 9 ns;
76          end if;   --adc_in < switchValue
77        end if;     --CLK
78      end if;       --RESET
79    end process generate_RECT_OUT;

81    generate_switchValue : process(minValue, maxValue)
82      variable tempswitch: unsigned(switchValue'length downto 0);
83    begin
84      tempswitch := ('0'&minValue)+('0'&maxValue);
85      switchValue <=  tempswitch(switchValue'length  downto 1);
86    end process generate_switchValue;


89    trace_ADC_IN : process(SYSCLK, RESET)
90    begin
```

```
91      if RESET = RESET_VAL then
92        minValue   <= x"FFF" after 9 ns;
93        maxValue   <= x"000" after 9 ns;
94        last_rect_out_sig <= '0' after 9 ns;
95        SIG_INC   <= '0' after 9 ns;
96      elsif SYSCLK = '1' and SYSCLK'event then
97        if CLK = '1' then
98          SIG_INC <= '0' after 9 ns;
99          if (last_rect_out_sig = not rect_out_sig) then
100           if rect_out_sig = '1' then
101             SIG_INC     <= '1' after 9 ns;
102             maxValue <= ADC_IN after 9 ns;
103           else
104             minValue <= ADC_IN after  9 ns;
105           end if;    -- rect_out_sig = 1
106           last_rect_out_sig <= rect_out_sig after 9 ns;
107         else
108           if ADC_IN < minValue  then
109             minValue   <= ADC_IN after 9 ns;
110           end if;     --adc_in < lower_reference
111           if ADC_IN > maxValue then
112             maxValue <= ADC_IN after 9 ns;
113           end if;    -- ADC_IN > maxValue
114         end if;      -- last_rect_out_sig != rect_out_sig
115       end if;       --SYSCLK
116     end if;        --RESET
117   end process trace_ADC_IN;

119 end behavior;
```

B.1.7 TL_sample.vhd

```
1  --#######################################################
2  --##  Project name : adc_sample_calc_thd
3  --##  File name    : TL_sample.vhd
4  --##  Author       : Jan-Heiner Dreschhoff
5  --##

7  --<makepackage>
8  --  <component = TL_sample>
9  --    <name = dreschhoff>
10 --    <project = thd>
11 --  </component>
12 --</makepackage>

14 library ieee;
15   use ieee.std_logic_1164.all;
16   use ieee.numeric_std.all;
```

```
17                --<debug>
18  -- use std.textio.all;
19  -- use ieee.std_logic_textio.all;
20                --</debug>
21  library work;
22    use work.udiff_reg_pkg.all;
23    use work.functions_pkg.all;
24    use work.global_pkg.all;
25    use work.lut_access_logic_pkg.all;
26    use work.multiply_add_pkg.all;
27    use work.operators_pkg.all;
28    use work.period_timer_pkg.all;
29    use work.sample_timer_pkg.all;
30    use work.sample_fsm_pkg.all;
31    use work.adc_db_lut_pkg.all;


34  entity TL_sample is

36    port(
37    --<debug>
38      DB_FSM_STATE          : out std_logic_vector(3 downto 0);
39      DB_SAMPLE_TIMER_DONE  : out bit;
40      DB_SET                : out bit;
41      DB_DONE               : out bit;
42      DB_BUS_SAMPLE_FSM     : out std_logic_vector(15 downto 0);
43      DB_UDIFF_REG          : out std_logic_vector(15 downto 0);
44      DB_BUS_ADC_LUT        : out std_logic_vector(15 downto 0);
45      DB_BUS_LUT_ACCESS     : out std_logic_vector(15 downto 0);
46      DB_BUS_LUT            : out std_logic_vector(15 downto 0);
47      DB_BUS_MUL_ADD        : out std_logic_vector(15 downto 0);
48      DB_BUS_MULTIPLY       : out std_logic_vector(15 downto 0);
49      SWITCH_6              : in bit;
50    --</debug>
51      SYSCLK         : in bit;
52      CLK            : in bit;
53      RESET          : in bit;
54      SIG_INC        : in bit;
55      ADC_UDIFF      : in unsigned((ADC_WIDTH-1) downto 0);
56      NEW_DATA       : out bit;
57      DATA_ERROR     : out bit;
58      RE             : out MulArrayS;
59      IM             : out MulArrayS
60    );
61  end TL_sample;

63  architecture structure of TL_sample is

65  --<debug>
66      signal db_state_sig : std_logic_vector(3 downto 0);
```

```
67      signal db_lut_value : unsigned(11 downto 0);
68  --</debug>

70  --SIGNALS FOR ADC_INTERFACE
71  signal adc_udiff_reg : unsigned ((ADC_WIDTH-1) downto 0);
72  signal set_udiff_reg       : bit;
73  signal fsm_adc_reset       : bit;

75  --SIGNALS FOR LUT
76  signal fsm_lut_enable         : bit;
77  signal lut_fsm_done           : bit;
78  signal fsm_lut_reset          : bit;
79  signal lut_fsm_last_harmonic  : bit;
80  signal lut_fsm_last_sample    : bit;
81  signal lut_multiply_add_index : unsigned(2 downto 0);
82  signal lut_multiply_add_value : signed((LUT_WIDTH_OUT -1) downto 0);
83  signal lut_multiply_add_cos   : std_logic;

85  --SIGNALS FOR SAMPLE_TIMER
86  signal period_time        : unsigned(PERIOD_TIMER_MSB downto 0);
87  signal sample_timer_start : bit;
88  signal sample_timer_set   : bit;
89  signal sample_timer_done  : bit;

91  --SIGNALS FOR PERIOD_TIMER
92  signal fsm_period_timer_start : bit;
93  signal fsm_period_timer_reset : bit;
94  signal fsm_period_timer_hold  : bit;
95  signal period_timer_timeout   : bit;

97  --SIGNALS FOR MULTIPLY_ADD
98  signal adc_value              : unsigned(11 downto 0);
99  signal fsm_multiply_add_set   : bit;
100 signal fsm_multiply_add_reset : bit;
101 signal multiply_add_fsm_done  : bit;

103 begin

105   set_udiff_reg  <= sample_timer_done or SIG_INC after 9 ns;

107 --<debug>
108   DB_FSM_STATE          <= db_state_sig;
109   DB_SAMPLE_TIMER_DONE  <= sample_timer_done;
110   DB_SET  <=  lut_fsm_last_sample;
111   DB_DONE <= lut_fsm_last_harmonic;
112 --</debug>

114   process(SWITCH_6, adc_udiff_reg, db_lut_value)
115   begin
116     if SWITCH_6 = '0' then
```

```
117       adc_value <= adc_udiff_reg;
118     else
119       adc_value <= db_lut_value;
120     end if;
121   end process;

123 --<debug = no_synthesis>
124 -- debug_mul_add : process(RESET, SYSCLK)
125    -- file myfile     : text open write_mode is "mul_add_data.txt";
126    -- variable myline : line;
127    -- variable fstatus : FILE_OPEN_STATUS;
128    -- variable sig_int_count : natural ;
129    -- variable old : bit;
130  -- begin
131    -- if RESET = RESET_VAL then
132      -- sig_int_count := 0;
133    -- elsif SYSCLK = '1' and SYSCLK'event then
134      -- if CLK = '1' then
135        -- if  sig_inc_sig = '1' then
136          -- sig_int_count := sig_int_count +1;
137          -- old:= '0';
138        -- end if;
139        -- if sig_int_count = 2 and fsm_multiply_add_set = '1' then
140          -- if old = '0' then
141            -- old := '1';
142            -- write(myline, string'("index"));
143            -- writeline(myfile, myline);
144            -- write(myline, std_logic_vector(lut_multiply_add_index));
145            -- writeline(myfile, myline);
146            -- write(myline, string'("COS"));
147            -- writeline(myfile, myline);
148            -- write(myline, std_logic(lut_multiply_add_cos));
149            -- writeline(myfile, myline);
150            -- write(myline, string'("LUT_VALUE"));
151            -- writeline(myfile, myline);
152            -- write(myline, std_logic_vector(lut_multiply_add_value));
153            -- writeline(myfile, myline);
154            -- write(myline, string'("ADC_VALUE"));
155            -- writeline(myfile, myline);
156            -- write(myline, std_logic_vector(adc_udiff_reg));
157            -- writeline(myfile, myline);
158          -- end if; --old
159        -- elsif fsm_multiply_add_set = '0' then
160          -- old := '0';
161        -- end if; --siginc
162      -- end if; --clk
163    -- end if;   --reset
164  -- end process debug_mul_add;
165  --</debug>
```

```
167 --z_____________________________________________________UDIFF_REG
168   UDIFF_REG_COMP : udiff_reg
169   port map(   -- std. ports
170   --<debug>
171     DB_UDIFF_REG => DB_UDIFF_REG,
172   --</debug>
173     SYSCLK => SYSCLK,
174     RESET  => RESET,
175     CLK    => CLK,
176           -- controlpaths to fsm
177     SET_REG    => set_udiff_reg,--sample_timer_done or SIG_INC,
178     SRESET     => fsm_adc_reset,
179     U_DIFF_IN  => ADC_UDIFF,
180     U_DIFF_REG => adc_udiff_reg
181     );

183 --z_____________________________________________________ADC_DB_LUT
184   DB_ADC : ADC_DB_LUT
185   port map(
186   --<debug>
187     DB_BUS_ADC_LUT => DB_BUS_ADC_LUT,
188   --</debug>
189     SYSCLK  => SYSCLK,
190     RESET   => RESET,
191     CLK     => CLK,
192     SIG_INC => SIG_INC,
193     SAMPLE_TIMER_DONE => sample_timer_done,
194     DB_LUT_OUT        => db_lut_value
195   );
196 --z_____________________________________________________FSM
197   FSM : sample_fsm
198   port map(
199     --<debug>
200       BD_FSM_STATE => db_state_sig,
201       DB_BUS_SAMPLE_FSM => DB_BUS_SAMPLE_FSM,
202     --</debug>
203     SYSCLK => SYSCLK,
204     RESET  => RESET,
205     CLK    => CLK,

207     SIG_INC  => SIG_INC,

209     ERROR    => DATA_ERROR,
210     NEW_DATA => NEW_DATA,

212     LUT_ENABLE        => fsm_lut_enable,
213     LUT_DONE          => lut_fsm_done,
214     LUT_RESET         => fsm_lut_reset,
215     LUT_LAST_SAMPLE   => lut_fsm_last_sample,
216     LUT_LAST_HARMONIC => lut_fsm_last_harmonic,
```

```
218     MUL_ADD_SET   => fsm_multiply_add_set,
219     MUL_ADD_RESET => fsm_multiply_add_reset,
220     MUL_ADD_DONE  => multiply_add_fsm_done,

222     PERIOD_TIMEOUT     => period_timer_timeout,

224     SAMPLE_TIMER_DONE  => sample_timer_done
225     );
226 --z_____________________________________________SAMPLE_TIMER
227   SAMPLE_TIMER_COMP : sample_timer
228     port map(
229       SYSCLK      => SYSCLK,
230       RESET       => RESET,
231       CLK         => CLK,
232       VALUE       => period_time,
233       START       => sample_timer_start,
234       SET         => sample_timer_set,
235       DONE        => sample_timer_done
236     );
237 --z_____________________________________________PERIOD_TIMER
238   PERIOD_TIMER_COMP : period_timer
239     port map(
240       SYSCLK      => SYSCLK,
241       RESET       => RESET,
242       CLK         => CLK,
243       VALUE       => period_time,
244       SIG_INC_IN  => SIG_INC,
245       SET_SAMPLE_TIMER   => sample_timer_set,
246       START_SAMPLE_TIMER => sample_timer_start,
247       TIMEOUT            => period_timer_timeout
248     );
249 --z_____________________________________________________LUT
250   LUT_COMP: lut_access_logic
251     port map(
252     --<debug>
253       DB_BUS_LUT_ACCESS => DB_BUS_LUT_ACCESS,
254       DB_BUS_LUT        => DB_BUS_LUT,
255     --</debug>
256       SYSCLK        => SYSCLK,
257       RESET         => RESET,
258       CLK           => CLK,
259       ENABLE        => fsm_lut_enable,
260       DONE          => lut_fsm_done,
261       SRESET        => fsm_lut_reset,
262       UNDERSAMPLE   => '1',
263       LAST_SAMPLE   => lut_fsm_last_sample,
264       LAST_HARMONIC => lut_fsm_last_harmonic,
265       INDEX_OUT     => lut_multiply_add_index,
266       VALUE         => lut_multiply_add_value,
```

```
267       COS_OUT       => lut_multiply_add_cos
268     );
269 --_____________________________________________________MULTIPLY
270     MULTIPLY_ADD_COMP : multiply_add
271     port map(
272     --<debug>
273       DB_BUS_MUL_ADD  => DB_BUS_MUL_ADD,
274       DB_BUS_MULTIPLY => DB_BUS_MULTIPLY,
275     --</debug>
276       SYSCLK     => SYSCLK,
277       RESET      => RESET,
278       CLK        => CLK,
279       SET        => fsm_multiply_add_set,
280       SRESET     => fsm_multiply_add_reset,
281       DONE       => multiply_add_fsm_done,
282       INDEX_IN   => lut_multiply_add_index,
283       COS        => lut_multiply_add_cos,
284       FACTOR_IN  => lut_multiply_add_value,
285       ADC_IN     => adc_value,
286       RE_REG_OUT => RE,
287       IM_REG_OUT => IM
288     );
289 end structure;
```

B.1.7.1 udiff_reg.vhd

```
1  --#######################################################
2  --##  Project name : adc_sample_calc_thd
3  --##  File name    : udiff_reg.vhd
4  --##  Author       : Jan-Heiner Dreschhoff
5
6
7  --<makepackage>
8  --  <component = udiff_reg>
9  --    <name = dreschhoff>
10 --    <project = adc_sample_calc_thd>
11 --  </component>
12 --</makepackage>
13
14 library ieee;
15   use ieee.std_logic_1164.all;
16   use ieee.numeric_std.all;
17 library work;
18   use work.global_pkg.all;
19   use work.operators_pkg.all;
20   use work.functions_pkg.all;
21
22 entity udiff_reg is
```

```
23    port(
24    --<debug>
25      DB_UDIFF_REG : out std_logic_vector(15 downto 0);
26    --</debug>
27      SYSCLK : in bit;
28      RESET  : in bit;
29      CLK    : in bit;
30            -- controlpaths to fsm
31      SET_REG    : in bit;
32      SRESET     : in bit;
33            -- datapath
34      U_DIFF_IN  : in  unsigned((ADC_WIDTH-1) downto 0);
35      U_DIFF_REG : out unsigned((ADC_WIDTH-1) downto 0)
36      );
37  end udiff_reg;

39  architecture behavior of udiff_reg is

41  signal udiff : unsigned((ADC_WIDTH-1) downto 0) := (others => '0');

43  begin

45    U_DIFF_REG <= udiff after 9 ns;
46    DB_UDIFF_REG(15 downto 12) <= "0000";
47    DB_UDIFF_REG(11 downto 0)  <= std_logic_vector(udiff(11 downto 0));
48    --DB_UDIFF_REG(3 downto 0)   <= "0000";

50    process(SYSCLK, RESET)
51    begin
52      if RESET = RESET_VAL then
53        udiff <= (others => '0') after 9 ns;
54      elsif SYSCLK = '1' and SYSCLK'event then
55        if CLK = '1' then
56          if SRESET = '1' then
57            udiff <= (others => '0') after 9 ns;
58          elsif SET_REG = '1' then
59            udiff <= U_DIFF_IN after 9 ns;
60          end if;    -- SRESET
61        end if;      --CLK
62      end if;    --RESET
63    end process;

65  end behavior;
```

B.1.7.2 ADC_DB_LUT.vhd

```
1   --#########################################################
2   --##  Project name : adc_sample_calc_thd
```

```
3  --##  File name    : LUT.vhd
4  --##  Author       : Jan-Heiner Dreschhoff
5  --##
6  --## contains 64 samples with a thd of 24,24%.
7  --##  index resetted by SIG_INC, incremented by
8  --##  sample_timer_done

10 --<makepackage>
11 --  <global>
12 --    type db_lut_array is array(0 to 63) of unsigned(11 downto 0);
13 --  </global>
14 --  <component = ADC_DB_LUT>
15 --    <name = dreschhoff>
16 --    <project = thd>
17 --  </component>
18 --</makepackage>

20 library ieee;
21   use ieee.std_logic_1164.all;
22   use ieee.numeric_std.all;
23 library work;
24   use work.global_pkg.all;
25   use work.functions_pkg.all;


28 entity ADC_DB_LUT is
29   port
30   (
31     --<debug>
32       DB_BUS_ADC_LUT : out std_logic_vector(15 downto 0);
33     --</debug>
34     SYSCLK : in bit;
35     RESET  : in bit;
36     CLK    : in bit;
37     SIG_INC : in bit;
38     SAMPLE_TIMER_DONE : in bit;
39     DB_LUT_OUT: out unsigned(11 downto 0)
40   );
41 end entity ADC_DB_LUT;

43 architecture behavior of ADC_DB_LUT is

45   constant LUT_LENGTH   : natural     := 64;
46   constant omega  : real := 3.14159*2/real(LUT_LENGTH);
47   constant omega2  : real := 3.14159*4/real(LUT_LENGTH);
48   constant offset : real := 2967.0;
49   -- constant amp1    : real :=  823.0;
50   constant amp1   : real := 800.0;
51   constant amp2   : real := 23.0;
52   constant amp3   : real := 200.0;
```

```
54    function db_lut_func_sin return db_lut_array is
55      variable lut : db_lut_array;
56      variable temp : signed(12 downto 0);
57    begin
58      for i in 0 to 63 loop
59        temp :=to_signed(offset
60        + amp1 * ieee.math_real.sin(real(i)*omega)
61        -- + amp2 * ieee.math_real.cos(real(i)*omega), 12) ;
62        + amp3 * ieee.math_real.sin((real(i)*omega2)+7.0), 12);
63        -- temp   := to_signed(amp1 * ieee.math_real.sin(real(i)*omega)
64                  -- + offset, 12);
65        lut(i) := unsigned(std_logic_vector(temp(11 downto 0)));
66      end loop;
67      return lut;
68    end db_lut_func_sin;

70    constant db_lut : db_lut_array := db_lut_func_sin;

72    signal index : integer range 0 to 63;

74  begin

76    DB_LUT_OUT <= db_lut(index);
77    DB_BUS_ADC_LUT(15 downto 12) <= (others => '0');
78    DB_BUS_ADC_LUT(11 downto  0) <= std_logic_vector(db_lut(index));

80    process(SYSCLK, RESET)
81    begin
82      if RESET = RESET_VAL then
83        index <= 0;
84      elsif SYSCLK = '1' and SYSCLK'event then
85        if CLK = '1' then
86          if index = 63 then
87            if SIG_INC = '1' then
88              index <= 0;
89            end if;
90          else
91            if SIG_INC = '1' then
92              index <= 0;
93            elsif SAMPLE_TIMER_DONE = '1' then
94              index <= index + 1;
95            end if;    -- SIG_INC
96          end if;    -- index
97        end if;    --SYSCLK
98      end if;    -- RESET
99    end process;

101 end behavior;
```

B.1.7.3 sample_fsm.vhd

```
--#######################################################
--##  Project name : adc_sample_calc_thd
--##  File name    : sample_fsm.vhd
--##  Author       : Jan-Heiner Dreschhoff




--<makepackage>
--  <global>
--      type SAMPLE_STATE_TYPE is (FIRST_PERIOD,
--          WAIT_FOR_SIG_INC_LOW, WAIT_FOR_SIG_INC_HIGH,
--          WAIT_FOR_SAMPLE_TIMER, LUT_ENABLE_STATE,
--          LUT_DONE_STATE, MUL_ADD_SET_STATE,
--          MUL_ADD_DONE_STATE,NEW_DATA_STATE,
--          WAIT_FOR_SAMPLE_TIMER_AFTER_LAST_SAMPLE,
--          ACCELERATION_ERROR, DECELERATION_ERROR);
--  </global>
--  <component = sample_fsm>
--    <name = dreschhoff>
--    <project = thd>
--  </component>
--</makepackage>

library ieee;
  use ieee.std_logic_1164.all;
  use ieee.numeric_std.all;
library work;
  use work.global_pkg.all;
  use work.operators_pkg.all;
  use work.functions_pkg.all;

  entity sample_fsm is
    port(
    --<debug>
      BD_FSM_STATE      : out std_logic_vector(3 downto 0);
      DB_BUS_SAMPLE_FSM : out std_logic_vector(15 downto 0);
    --</debug>

      SYSCLK  : in bit;
      RESET   : in bit;
      CLK     : in bit;

      SIG_INC : in bit;

      ERROR    : out bit;
      NEW_DATA : out bit;
```

```
49        RESET_SAMPLE_DATA : out bit;

51        LUT_ENABLE        : out bit;
52        LUT_DONE          : in bit;
53        LUT_RESET         : out bit;
54        LUT_LAST_SAMPLE   : in bit;
55        LUT_LAST_HARMONIC : in bit;

57        MUL_ADD_SET : out bit;
58        MUL_ADD_RESET : out bit;
59        MUL_ADD_DONE : in bit;

61        PERIOD_TIMEOUT     : in bit;

63        SAMPLE_TIMER_DONE  : in bit
64        );

66    end sample_fsm;

68    architecture behaviour of sample_fsm is

70    signal STATE, NEXT_STATE : SAMPLE_STATE_TYPE;

72    --<debug = LUT_DUMMY>
73    -- signal LUT_ENABLE : bit;
74    -- signal LUT_RESET : bit;
75    -- signal LUT_DONE : bit;
76    -- signal LUT_LAST_SAMPLE : bit;
77    -- signal LUT_LAST_HARMONIC : bit;
78    --</debug>
79    --<debug = MUL_ADD_DUMMY>
80    -- signal MUL_ADD_SET   : bit;
81    -- signal MUL_ADD_DONE  : bit;
82    -- signal MUL_ADD_RESET : bit;
83    --</debug>

85    begin

87      STATE_REG : process(SYSCLK, RESET)
88      begin
89        if RESET = RESET_VAL then
90          STATE <= FIRST_PERIOD after 9 ns;
91        elsif SYSCLK = '1' and SYSCLK'event then
92          if CLK = '1' then
93            STATE <= NEXT_STATE after 9 ns;
94          end if;
95        end if;
96      end process STATE_REG;
```

```
99      FSM : process(STATE, SAMPLE_TIMER_DONE, SIG_INC,PERIOD_TIMEOUT, LUT_DONE, MUL_ADD
100     begin
101     --<debug>
102       BD_FSM_STATE <= x"F" after 9 ns;
103     --</debug>
104       ERROR  <= '0' after 9 ns;
105       NEW_DATA <= '0' after 9 ns;
107       RESET_SAMPLE_DATA <= '0' after 9 ns;
109       MUL_ADD_RESET <= '0' after 9 ns;
110       MUL_ADD_SET <= '0' after 9 ns;
112       LUT_ENABLE <= '0' after 9 ns;
113       LUT_RESET  <= '0' after 9 ns;
115       case STATE is
116         when FIRST_PERIOD =>
117         --<debug>
118           BD_FSM_STATE <= x"1" after 9 ns;
119         --</debug>
120           ERROR <= '1' after 9 ns;
121           if SIG_INC = '1' then
122             NEXT_STATE <= WAIT_FOR_SIG_INC_LOW after 9 ns;
123           else
124             NEXT_STATE <= FIRST_PERIOD after 9 ns;
125           end if;
126         when WAIT_FOR_SIG_INC_LOW=>
127         --<debug>
128           BD_FSM_STATE <= x"2" after 9 ns;
129         --</debug>
130           if SIG_INC = '0' then
131             NEXT_STATE <= WAIT_FOR_SIG_INC_HIGH after 9 ns;
132           else
133             NEXT_STATE <= WAIT_FOR_SIG_INC_LOW after 9 ns;
134           end if;
135         when WAIT_FOR_SIG_INC_HIGH =>
136         --<debug>
137           BD_FSM_STATE <= x"3" after 9 ns;
138         --</debug>
139           LUT_RESET <= '1' after 9 ns;
140           if SIG_INC = '1' then
141             NEXT_STATE <= LUT_ENABLE_STATE after 9 ns;
142           elsif PERIOD_TIMEOUT = '1' then
143             NEXT_STATE <= FIRST_PERIOD after 9 ns;
144           elsif SAMPLE_TIMER_DONE = '1' then
145             NEXT_STATE <= DECELERATION_ERROR after 9 ns;
146           else
147             NEXT_STATE <= WAIT_FOR_SIG_INC_HIGH after 9 ns;
148           end if;
```

```
149         when LUT_ENABLE_STATE =>
150         --<debug>
151           BD_FSM_STATE <= x"4" after 9 ns;
152         --</debug>
153           LUT_ENABLE <= '1';
154           if SIG_INC = '1' then
155             NEXT_STATE <= ACCELERATION_ERROR after 9 ns;
156           elsif LUT_DONE = '1' then
157             NEXT_STATE <= LUT_DONE_STATE after 9 ns;
158           else
159             NEXT_STATE <= LUT_ENABLE_STATE after 9 ns;
160           end if;
161         when LUT_DONE_STATE =>
162         --<debug>
163           BD_FSM_STATE <= x"5" after 9 ns;
164         --</debug>
165           if SIG_INC = '1' then
166             NEXT_STATE <= ACCELERATION_ERROR after 9 ns;
167           elsif LUT_DONE <= '0' then
168             NEXT_STATE <= MUL_ADD_SET_STATE after 9 ns;
169           else
170             NEXT_STATE <= LUT_DONE_STATE after 9 ns;
171           end if;
172         when MUL_ADD_SET_STATE =>
173         --<debug>
174           BD_FSM_STATE <= x"6" after 9 ns;
175         --</debug>
176           MUL_ADD_SET <= '1';
177           if SIG_INC = '1' then
178             NEXT_STATE <= ACCELERATION_ERROR after 9 ns;
179           elsif MUL_ADD_DONE = '1' then
180             NEXT_STATE <= MUL_ADD_DONE_STATE after 9 ns;
181           else
182             NEXT_STATE <= MUL_ADD_SET_STATE after 9 ns;
183           end if;
184         when MUL_ADD_DONE_STATE =>
185         --<debug>
186           BD_FSM_STATE <= x"7" after 9 ns;
187         --</debug>
188           if SIG_INC = '1' then
189             NEXT_STATE <= ACCELERATION_ERROR after 9 ns;
190           elsif MUL_ADD_DONE <= '0' then
191             if LUT_LAST_HARMONIC = '1' then
192               if LUT_LAST_SAMPLE = '1' then
193                 NEXT_STATE <= NEW_DATA_STATE after 9 ns;
194                 -- NEXT_STATE <= WAIT_FOR_SAMPLE_TIMER_AFTER_LAST_SAMPLE after 9 ns;
195               else
196                 NEXT_STATE <= WAIT_FOR_SAMPLE_TIMER;
197               end if;
198             else
```

```
199                 NEXT_STATE <= LUT_ENABLE_STATE;
200               end if;
201             else
202               NEXT_STATE <= MUL_ADD_DONE_STATE after 9 ns;
203             end if;
204           when WAIT_FOR_SAMPLE_TIMER =>
205           --<debug>
206             BD_FSM_STATE <= x"8" after 9 ns;
207           --</debug>
208             if SAMPLE_TIMER_DONE = '1' then
209               NEXT_STATE <= LUT_ENABLE_STATE;
210             elsif SIG_INC = '1' then
211               NEXT_STATE <= ACCELERATION_ERROR after 9 ns;
212             else
213               NEXT_STATE <= WAIT_FOR_SAMPLE_TIMER;
214             end if;
215           when NEW_DATA_STATE =>
216           --<debug>
217             BD_FSM_STATE <= x"C" after 9 ns;
218           --</debug>
219             NEW_DATA <= '1' after 9 ns;
220             NEXT_STATE <= WAIT_FOR_SAMPLE_TIMER_AFTER_LAST_SAMPLE after 9 ns;
221             if SIG_INC = '1' then
222               NEXT_STATE <= ACCELERATION_ERROR after 9 ns;
223             end if;
224           when WAIT_FOR_SAMPLE_TIMER_AFTER_LAST_SAMPLE =>
225           --<debug>
226             BD_FSM_STATE <= x"9" after 9 ns;
227           --</debug>
228             MUL_ADD_RESET <= '1' after 9 ns;
229             LUT_RESET     <= '1' after 9 ns;
230             if SIG_INC = '1' then
231               NEXT_STATE <= LUT_ENABLE_STATE after 9 ns;
232             elsif SAMPLE_TIMER_DONE = '1' then
233               NEXT_STATE <= WAIT_FOR_SIG_INC_HIGH after 9 ns;
234             elsif PERIOD_TIMEOUT = '1' then
235               NEXT_STATE <= DECELERATION_ERROR after 9 ns;
236             else
237               NEXT_STATE <= WAIT_FOR_SAMPLE_TIMER_AFTER_LAST_SAMPLE after 9 ns;
238             end if;
239           when ACCELERATION_ERROR =>
240           --<debug>
241             BD_FSM_STATE <= x"A";
242           --</debug>
243             ERROR <= '1';
244             LUT_RESET <= '1' after 9 ns;
245             MUL_ADD_RESET <= '1' after 9 ns;
246             NEXT_STATE <= LUT_ENABLE_STATE after 9 ns;
247           when DECELERATION_ERROR =>
248           --<debug>
```

```
249           BD_FSM_STATE <= x"B";
250         --</debug>
251           ERROR <= '1' after 9 ns;
252           LUT_RESET <= '1' after 9 ns;
253           MUL_ADD_RESET <= '1' after 9 ns;
254           if SIG_INC = '1' then
255             NEXT_STATE <= LUT_ENABLE_STATE after 9 ns;
256           elsif PERIOD_TIMEOUT = '1' then
257             NEXT_STATE <= FIRST_PERIOD;
258           else
259             NEXT_STATE <= DECELERATION_ERROR after 9 ns;
260           end if;
261         when others =>
262             NEXT_STATE  <= FIRST_PERIOD after 9 ns;
263           --<debug>
264             BD_FSM_STATE <= x"E" after 9 ns;
265           --</debug>
266       end case;
267     end process FSM;

269 --<debug = LUT_DUMMY>
270   -- LUT_DUMMY : process(SYSCLK, RESET)
271     -- variable sample_cnt : unsigned(5 downto 0);
272     -- variable harmonic_cos : unsigned(3 downto 0);
273     -- alias harmonic : unsigned(2 downto 0) is harmonic_cos(3 downto 1);
274     -- alias cos      : std_logic            is harmonic_cos(0);
275     -- variable LUT_wait_one_clk : bit;
276   -- begin
277     -- DB_SET    <= LUT_LAST_SAMPLE;
278     -- DB_DONE   <= LUT_LAST_HARMONIC;
279     -- if RESET = RESET_VAL then
280       -- LUT_DONE <= '1';
281       -- LUT_wait_one_clk := '0';
282       -- sample_cnt := (others => '0');
283       -- LUT_LAST_SAMPLE <= '0';
284       -- harmonic := "000";
285       -- cos := '1';
286     -- elsif SYSCLK = '1' and SYSCLK'event then
287       -- if CLK = '1' then
288     -- ----------------------------------------LUT DUMMY
289         -- LUT_LAST_SAMPLE <= '0' after 9 ns;
290         -- LUT_LAST_HARMONIC <= '0' after 9 ns;
291         -- if LUT_RESET = '1' then
292           -- LUT_LAST_SAMPLE <= '0';
293           -- LUT_wait_one_clk := '0';
294           -- LUT_DONE <= '0';
295           -- sample_cnt := (others => '0');
296           -- harmonic := "000";
297           -- cos := '1';
298         -- elsif LUT_ENABLE = '1' then
```

```
299           -- if LUT_wait_one_clk = '1' then
300             -- LUT_DONE <= '1';
301           -- else
302             -- if harmonic = "101" and cos = '1' then
303               -- sample_cnt := sample_cnt + x"1";
304               -- harmonic := "000";
305               -- cos      := '1';
306             -- end if;
307             -- harmonic_cos := harmonic_cos +1;
308             -- LUT_wait_one_clk := '1';
309           -- end if;
310         -- else
311           -- LUT_DONE <= '0';
312           -- LUT_wait_one_clk := '0';
313         -- end if;
314         -- LUT_LAST_SAMPLE <= to_bit(sample_cnt(5) and sample_cnt(4)
315                               -- and sample_cnt(3) and sample_cnt(2)
316                               -- and sample_cnt(1) and sample_cnt(0));
317         -- if harmonic = "101" and cos = '1' then
318           -- LUT_LAST_HARMONIC <= '1' after 9 ns;
319         -- end if;
320       -- end if;
321     -- end if;
322   -- end process LUT_DUMMY;
323 --</debug>

325 --<debug = MUL_ADD_DUMMY>
326   -- MUL_ADD_DUMMY : process(SYSCLK, RESET)
327     -- variable MUL_ADD_one_clk : bit;
328   -- begin
329     -- if RESET = RESET_VAL then
330       -- MUL_ADD_DONE <= '0';
331       -- MUL_ADD_one_clk := '0';
332     -- elsif SYSCLK = '1' and SYSCLK'event then
333       -- if CLK = '1' then
334       -- ------------------------------------MUL ADD DUMMY
335         -- if MUL_ADD_RESET = '1' then
336           -- MUL_ADD_DONE <= '0';
337           -- MUL_ADD_one_clk := '0';
338         -- elsif MUL_ADD_SET = '1' then
339           -- if MUL_ADD_one_clk = '1' then
340             -- MUL_ADD_DONE <= '1';
341           -- else
342             -- MUL_ADD_one_clk := '1';
343           -- end if;
344         -- else
345           -- MUL_ADD_DONE <= '0';
346           -- MUL_ADD_one_clk := '0';
347         -- end if;
348       -- end if;
```

```
349     -- end if;
350   -- end process MUL_ADD_DUMMY;
351  --</debug>

353   end behaviour;
```

B.1.7.4 sample_timer.vhd

```
1  --#######################################################
2  --##  Project name : adc_sample_calc_thd
3  --##  File name    : sample_timer.vhd
4  --##  Author       : Jan-Heiner Dreschhoff



8  --<makepackage>
9  --  <component = sample_timer>
10 --    <name = dreschhoff>
11 --    <project = thd>
12 --  </component>
13 --</makepackage>


16 library ieee;
17   use ieee.std_logic_1164.all;
18   use ieee.numeric_std.all;
19 library work;
20   use work.global_pkg.all;
21   use work.functions_pkg.all;


24 entity sample_timer is
25   port(
26     SYSCLK : in bit;     --TOPLEVEL
27     RESET : in bit;      --TOPLEVEL
28     CLK    : in bit;     --TOPLEVEL
29     VALUE : in unsigned(PERIOD_TIMER_MSB downto 0);    --PERIOD_TIMER
30     START : in bit;      --PERIOD_TIMER
31     SET    : in bit;     --PERIOD_TIMER
32     DONE   : out bit     --FSM
33     );
34 end sample_timer;

36 architecture behaviour of sample_timer is

38 signal counter     :  unsigned((PERIOD_TIMER_MSB-6) downto 0);
39 signal start_value :  unsigned((PERIOD_TIMER_MSB-6) downto 0);
```

```
41 begin

43   process(SYSCLK, RESET)
44     variable period_timer_var : unsigned((PERIOD_TIMER_MSB-6) downto 0);
45   begin
46     if RESET = RESET_VAL then
47       counter     <= (others =>'0') after 9 ns;
48       start_value <= (others =>'0') after 9 ns;
49     elsif SYSCLK = '1' and SYSCLK'event then
50       if CLK = '1' then
51         DONE <= '0' after 9 ns;
52         if SET = '1' then
53           period_timer_var := VALUE(PERIOD_TIMER_MSB downto 6);
54           start_value <= period_timer_var after 9 ns;
55           counter     <= period_timer_var after 9 ns;
56         elsif START = '1' then
57           counter <= counter - 1 after 9 ns;
58         elsif counter = 0 then
59           DONE    <= '1' after 9 ns;
60           counter <= start_value after 9 ns;
61         else
62           counter <= counter - 1 after 9 ns;
63         end if; --SET
64       end if;   -- CLK
65     end if;     --RESET
66   end process;

68 end behaviour;
```

B.1.7.5 period_timer.vhd

```
1  --#########################################################
2  --##  Project name : adc_sample_calc_thd
3  --##  File name    : period_timer.vhd
4  --##  Author       : Jan-Heiner Dreschhoff




9  --<makepackage>
10 --  <global>
11 --    constant PERIOD_TIMER_WIDTH : integer := 24;
12 --    constant PERIOD_TIMER_MSB   : natural := 23;
13 --  </global>
14 --  <component = period_timer>
15 --    <name = dreschhoff>
16 --    <project = thd>
17 --  </component>
```

```
18 --</makepackage>

20 library ieee;
21   use ieee.std_logic_1164.all;
22   use ieee.numeric_std.all;
23 library work;
24   use work.global_pkg.all;
25   use work.functions_pkg.all;

27 entity period_timer is
28   port(
29     SYSCLK     : in bit; --TOPLEVEL
30     RESET      : in bit; --TOPLEVEL
31     CLK        : in bit; --TOPLEVEL
32     VALUE      : out  unsigned(PERIOD_TIMER_MSB downto 0); --SAMPLE_TIMER
33     SIG_INC_IN : in bit; --TOPLEVEL
34     SET_SAMPLE_TIMER   : out bit;     --SAMPLE_TIMER
35     START_SAMPLE_TIMER : out bit;     --SAMPLE_TIMER
36     TIMEOUT    : out bit              --FSM
37     );
38 end period_timer;

40 architecture behaviour of period_timer is

42 signal counter :  unsigned(PERIOD_TIMER_MSB downto 0);
43 signal run : bit;

45 begin

47   VALUE <= counter after 9 ns;
48   process(SYSCLK, RESET)
49   begin
50     if RESET = RESET_VAL then
51       counter <= (others =>'0') after 9 ns;
52       run <= '0' after 9 ns;
53       SET_SAMPLE_TIMER   <= '0' after 9 ns;
54       START_SAMPLE_TIMER <= '0' after 9 ns;
55       TIMEOUT <= '0' after 9 ns;
56     elsif SYSCLK = '1' and SYSCLK'event then
57       if CLK = '1' then
58         TIMEOUT <= '0' after 9 ns;
59         SET_SAMPLE_TIMER   <= '0' after 9 ns;
60         START_SAMPLE_TIMER <= '0' after 9 ns;
61         if SIG_INC_IN = '1' then
62           SET_SAMPLE_TIMER <= '1' after 9 ns;
63           run <= '0';
64         elsif run = '0' then
65           START_SAMPLE_TIMER <= '1' after 9 ns;
66           counter <= (others =>'0') after 9 ns;
67           run <= '1' after 9 ns;
```

```
68          elsif counter(PERIOD_TIMER_MSB) = '1' then
69            TIMEOUT <= '1' after 9 ns;
70          else
71            counter <= counter + 1 after 9 ns;
72          end if;
73        end if;
74      end if;
75    end process;


78  end behaviour;
```

B.1.7.6 lut_access_logic.vhd

```
1   --#######################################################
2   --##  Project name : adc_sample_calc_thd
3   --##  File name    : lut_access_logic.vhd
4   --##  Author       : Jan-Heiner Dreschhoff
5   --##
6   --##  description:
7   --##  LUT for sine and cosine for DFT.
8   --##    input is sample
9   --##
10  --##
11  --##    discription:
12  --##  accesses the look up table and manipulates the outputs
13  --##      means, multiplies by -1, if output is in second half
14  --##      of the period



18  --<makepackage>
19  --  <component = lut_access_logic>
20  --    <name = dreschhoff>
21  --    <project = thd>
22  --  </component>
23  --</makepackage>

25  library ieee;
26    use ieee.std_logic_1164.all;
27     use ieee.numeric_std.all;
28  library work;
29    use work.global_pkg.all;
30    use work.lut_pkg.all;

32  entity lut_access_logic is
33    port(
34    --<debug>
```

```
35      DB_BUS_LUT_ACCESS : out std_logic_vector(15 downto 0);
36      DB_BUS_LUT        : out std_logic_vector(15 downto 0);
37    --</debug>
38      SYSCLK        : in bit;
39      RESET         : in bit;
40      CLK           : in bit;
41      ENABLE        : in bit;
42      DONE          : out bit;
43      SRESET        : in bit;
44      UNDERSAMPLE   : in bit;
45      LAST_SAMPLE   : out bit;
46      LAST_HARMONIC : out bit;
47      INDEX_OUT     : out unsigned(2 downto 0);
48      VALUE         : out signed((LUT_WIDTH_OUT-1) downto 0);  -- manipulated lut value
49      COS_OUT       : out std_logic
50    );
51  end lut_access_logic;

53  architecture behavior of lut_access_logic is

55    signal lut_enable        : bit;
56    signal new_value_sig     : bit;
57    signal lut_value         : signed((LUT_WIDTH_OUT-1) downto 0);
58    signal lut_value_negativ : signed((LUT_WIDTH_OUT-1) downto 0);

60    signal quater_select : std_logic;
61    signal half_select   : std_logic;
62    signal addr          : unsigned((LUT_WIDTH_IN+1) downto 0);
63    signal cos_sig       : std_logic;

65  begin
66    --<debug>
67      DB_BUS_LUT_ACCESS(11 downto 8) <= std_logic_vector(addr(3 downto 0));
68      DB_BUS_LUT_ACCESS(7) <= to_stdulogic(ENABLE);--addr(5);
69      DB_BUS_LUT_ACCESS(6) <= to_stdulogic(lut_enable);--addr(4);
70      DB_BUS_LUT_ACCESS(5) <= quater_select;
71      DB_BUS_LUT_ACCESS(4) <= half_select;
72      DB_BUS_LUT_ACCESS(3 downto 0) <= (others => '0');
73    --</debug>

75    lut_value_negativ <= -lut_value;
76    quater_select     <= addr(4) xor cos_sig;
77    half_select       <= addr(5) xor(addr(4) and cos_sig);

79    sample_and_harmonics_counter : process(SYSCLK, RESET)
80      variable sample_cnt : unsigned(5 downto 0);
81      variable harmonic_cos : unsigned(3 downto 0);
82      alias harmonic : unsigned(2 downto 0) is harmonic_cos(3 downto 1);
83      alias cos      : std_logic            is harmonic_cos(0);
84      variable once : bit;
```

```
85      variable wait_one_clk : bit;
86    begin
87      COS_OUT   <= cos;
88      cos_sig   <= cos;
89      INDEX_OUT <= harmonic;
90      if RESET = RESET_VAL then
91        -- DONE          <= '0';
92        once  := '0';
93        sample_cnt    := (others => '0');
94        LAST_SAMPLE   <= '0';
95        LAST_HARMONIC <= '0';
96        harmonic      := "000";
97        cos           := '1';
98        wait_one_clk  := '0';
99        lut_enable    <= '0';
100       addr          <= (others => '0');
101     elsif SYSCLK = '1' and SYSCLK'event then
102       if CLK = '1' then
103         lut_enable    <= '0';
104         LAST_SAMPLE   <= '0' after 9 ns;
105         LAST_HARMONIC <= '0' after 9 ns;
106         if SRESET = '1' then
107           once  := '0';
108           LAST_SAMPLE   <= '0';
109           LAST_HARMONIC <= '0';
110           sample_cnt    := (others => '0');
111           harmonic      := "000";
112           cos           := '1';
113           wait_one_clk  := '0';
114           lut_enable    <= '0';
115           addr          <= (others => '0');
116         elsif ENABLE = '1' then
117           if wait_one_clk = '0' then
118             wait_one_clk := '1';
119           else
120             wait_one_clk := '1';
121             lut_enable <= '1' after 9 ns;
122             if once = '0' then
123               if harmonic = "101" and cos = '1' then
124                 if UNDERSAMPLE = '1' then
125                   sample_cnt := sample_cnt + x"9";
126                 else
127                   sample_cnt := sample_cnt + x"1";
128                 end if;
129                 harmonic   := "000";
130                 cos        := '1';
131                 addr       <= sample_cnt;
132               elsif cos = '1' then
133                 addr <= addr + sample_cnt;
134               end if;
```

```
135               harmonic_cos := harmonic_cos +1;
136               once := '1';
137             end if;
138           end if;
139         else
140           wait_one_clk := '0';
141           once := '0';
142         end if;
143         if UNDERSAMPLE = '1' then
144           LAST_SAMPLE <= to_bit(sample_cnt(5) and sample_cnt(4)
145                                 and (not sample_cnt(3)) and sample_cnt(2)
146                                 and sample_cnt(1) and sample_cnt(0));
147         else
148           LAST_SAMPLE <= to_bit(sample_cnt(5) and sample_cnt(4)
149                                 and sample_cnt(3) and sample_cnt(2)
150                                 and sample_cnt(1) and sample_cnt(0));
151         end if;
152         if cos = '1' and harmonic = "101" then
153           LAST_HARMONIC <= '1' after 9 ns;
154         end if;
155       end if;
156     end if;
157   end process sample_and_harmonics_counter;


160   --interfaces w/ LUT, determines weather lut_value is
161   -- in first or second half of period.
162   --  manipulates output accordingly
163   process(SYSCLK, RESET)
164     variable wait_one_clk : bit;
165     variable positiv_value : signed(11 downto 0) := x"000";
166     variable negativ_value : signed(11 downto 0) := x"000";
167   begin
168     if RESET = RESET_VAL then
169       VALUE <= x"000";
170       wait_one_clk := '0';
171       DONE <= '0';
172       positiv_value := x"000";
173       negativ_value := x"000";
174     elsif SYSCLK = '1' and SYSCLK'event then
175       if CLK = '1' then
176         positiv_value := lut_value;
177         negativ_value(11) := '1';
178         negativ_value(10 downto 0) := x"3ff"-lut_value(10 downto 0);
179         if ENABLE = '1' then
180           if new_value_sig = '1' then
181             if wait_one_clk = '0' then
182               wait_one_clk := '1';
183             else
184               DONE <= '1';
```

```
185                 if half_select = '1' then --addr(4,5) + cos
186                   VALUE <= lut_value_negativ after 9 ns;--second half of period: negati
187                 else   --first half of period: positive
188                   VALUE <= lut_value after 9 ns;
189                 end if;     -- addr45+cos
190               end if;
191             end if;
192           elsif ENABLE = '0' then
193             DONE <= '0';
194             wait_one_clk := '0';
195           end if;
196         end if;
197       end if;
198     end process;


201     sinLut : LUT
202     port map
203     (
204     --<debug>
205       DB_BUS_LUT => DB_BUS_LUT,
206     --</debug>
207       SYSCLK => SYSCLK,
208       RESET  => RESET,
209       CLK    => CLK,
210       ENABLE        => lut_enable,
211       ADDR          => addr((LUT_WIDTH_IN-1) downto 0),
212       SECTOR_SELECT => quater_select,
213       LUT_OUT       => lut_value,
214       NEW_VALUE     => new_value_sig
215     );
216 end behavior;
```

LUT.vhd

```
1  --#######################################################
2  --##  Project name : adc_sample_calc_thd
3  --##  File name    : LUT.vhd
4  --##  Author       : Jan-Heiner Dreschhoff
5  --##
6  --##  description:
7  --##  LUT for sine and cosine for DFT.
8  --##    input is sample
9  --##    sine LUT => sin(addr_in*omega)

11 ----------
```

```
15 --<makepackage>
16 --  <global>
17 --    type Tarray is array(1 to 15) of signed(11 downto 0);
18 --    constant LUT_WIDTH_IN  : natural := 4;
19 --    constant LUT_WIDTH_OUT : natural := 12;
20 --  </global>
21 --  <component = LUT>
22 --    <name = dreschhoff>
23 --    <project = thd>
24 --  </component>
25 --</makepackage>

27 library ieee;
28   use ieee.std_logic_1164.all;
29   use ieee.numeric_std.all;
30 library work;
31   use work.global_pkg.all;
32   use work.functions_pkg.all;


35 entity LUT is
36   port
37   (
38     --<debug>
39       DB_BUS_LUT : out std_logic_vector(15 downto 0);
40     --</debug>
41     SYSCLK : in bit;
42     RESET  : in bit;
43     CLK    : in bit;
44     ENABLE : in bit;
45     ADDR   : in unsigned((LUT_WIDTH_IN-1) downto 0);
46     SECTOR_SELECT     : in std_logic;
47     LUT_OUT: out signed((LUT_WIDTH_OUT-1) downto 0);
48     NEW_VALUE : out bit
49   );
50 end entity LUT;

52 architecture behavior of LUT is

54   constant LUT_LENGTH    : natural     := (2**LUT_WIDTH_IN)-1;
55   constant omega : real := 3.14159*2/real((LUT_LENGTH*4)+4);




60   --create LUT. calling this function creates an array of 15
61   --            sin Values from >0° to <90°.
62   function sin_lut_func return Tarray is
63     variable luts : Tarray;
```

```
64    begin
65      for i in 1 to LUT_LENGTH loop
66        luts(i) := to_signed(ieee.math_real.sin(real(i)*omega)*
67                             real(2**(LUT_WIDTH_OUT-1)-1),
68                             luts(1)'length);
69      end loop;
70      return luts;
71    end sin_lut_func;

73    constant cSin : Tarray := sin_lut_func;

75    signal index1_sig : integer range 0 to 15;
76    signal index1_sig_range_check : integer range 0 to 2**4;
77    signal index2_sig : integer range 0 to 15;
78    signal index2_sig_range_check : integer range 0 to 2**4;

80  begin

82      index1_sig_range_check <= to_integer(ADDR);
83      index1_sig <= 0 when index1_sig_range_check < 0 else
84                    15 when index1_sig_range_check > 15 else
85                    index1_sig_range_check;
86      index2_sig_range_check <= to_integer(not(ADDR-1));
87      index2_sig <= 0 when index2_sig_range_check < 0 else
88                    15 when index2_sig_range_check > 15 else
89                    index2_sig_range_check;

91    process(SYSCLK, RESET)
92      variable wait_one_clk : bit;
93    begin
94      if RESET = RESET_VAL then
95        LUT_OUT <= x"FFF";
96        wait_one_clk := '0';
97      elsif SYSCLK = '1' and SYSCLK'event then
98        if CLK = '1' then
99          if ENABLE = '1' then
100           if wait_one_clk = '0' then
101             wait_one_clk := '1'; --stabilizes signals. necessary!
102           else
103             NEW_VALUE <= '1';
104             if SECTOR_SELECT = '1' then  -- 90° <= addr < 180°
105               if ADDR = x"0" then -- 90°
106                 LUT_OUT <= x"7ff" after 9 ns;
107               else                -- 90° < addr < 180°
108                 LUT_OUT <= cSin(index2_sig) after 9 ns;
109               end if;
110             else   -- 0° <= addr < 90°
111               if ADDR = x"0" then -- 0°
112                 LUT_OUT <= x"000" after 9 ns;
113               else    -- 0° < addr < 90°
```

```
114                 LUT_OUT <= cSin(index1_sig) after 9 ns;
115               end if;    -- ADDR = 0
116             end if;    --sector_select
117           end if;
118         else
119           NEW_VALUE <= '0';
120           wait_one_clk := '0';
121         end if;
122       end if;
123     end if;
124   end process;

126 end behavior;
```

B.1.7.7 multiply_add.vhd

```
1  --#######################################################
2  --##  Project name : adc_sample_calc_thd
3  --##  File name    : multiply_add.vhd
4  --##  Author       : Jan-Heiner Dreschhoff

6  --<makepackage>
7  --  <component = multiply_add>
8  --    <name = dreschhoff>
9  --    <project = thd>
10 --  </component>
11 --</makepackage>


14 library ieee;
15   use ieee.std_logic_1164.all;
16   use ieee.numeric_std.all;
17 library work;
18   use work.global_pkg.all;
19   use work.operators_pkg.all;
20   use work.functions_pkg.all;
21   use work.multiply_pkg.all;

23   entity multiply_add is
24     port(
25       --<debug>
26         DB_BUS_MUL_ADD  : out std_logic_vector(15 downto 0);
27         DB_BUS_MULTIPLY : out std_logic_vector(15 downto 0);
28       --</debug>
29         SYSCLK     : in bit;
30         CLK        : in bit;
31         RESET      : in bit;
32         SET        : in bit;
```

```
33          SRESET      : in bit;
34          DONE        : out bit;
35          INDEX_IN    : in unsigned(2 downto 0);
36          COS         : in std_logic;
37          FACTOR_IN   : in signed((LUT_WIDTH_OUT-1) downto 0);
38          ADC_IN      : in unsigned((ADC_WIDTH -1) downto 0);
39          RE_REG_OUT  : out MulArrayS;
40          IM_REG_OUT  : out MulArrayS
41    );
42  end multiply_add;

44    architecture behaviour of multiply_add is


47      signal index : integer range 1 to 5;
48      signal index_range_check : integer range 0 to 7;
49      signal im_reg    : MulArrayS;
50      signal re_reg    : MulArrayS;

52      signal wait_one_clk   : bit;
53      -- signal shift          : integer range 0 to 5;
54      signal factor_in_u    : signed (11 downto 0);
55      signal multiply_done  : bit;
56      signal multiply_set   : bit;
57      signal multiply_reset : bit;
58      signal multiply_value_out_u  : signed(23 downto 0);
59      signal multiply_value_out    : signed(23 downto 0);
60      signal shift_add_result      : signed(32 downto 0);
61      signal im_result             : MulArrayS;
62      signal re_result             : MulArrayS;

64    begin

66    --<debug>
67      DB_BUS_MUL_ADD(15 downto 12)   <= "0000";
68      DB_BUS_MUL_ADD(11 downto 4)   <= std_logic_vector(re_reg(index)(30 downto 23));
69      DB_BUS_MUL_ADD(3 downto 0)   <= "0000";
70    --</debug>
71    multiply_set <= SET;

73    index_range_check <= to_integer(INDEX_IN);
74    index <= 1 when index_range_check < 1 else
75             5 when index_range_check > 5 else
76             index_range_check;

78    IM_REG_OUT <= im_reg;
79    RE_REG_OUT <= re_reg;

81    signed_unsigned_signed : process(FACTOR_IN, multiply_value_out_u)
82    begin
```

```
83      if FACTOR_IN(11) = '1' then
84        factor_in_u        <= -FACTOR_IN;
85        --multiply_value_out <= -(multiply_value_out_u srl shift);
    --NEU
86        multiply_value_out <= -multiply_value_out_u;
87      else
88        factor_in_u        <=  FACTOR_IN;
89        -- multiply_value_out <=  multiply_value_out_u srl shift;
    --NEU
90        multiply_value_out <=  multiply_value_out_u;
91      end if;
92    end process signed_unsigned_signed;

94    process(SYSCLK, RESET) is
95      variable tempvar : signed (32 downto 0);
96      variable wait_one_clk : bit;
97    begin
98      if RESET = RESET_VAL then
99        for i in 1 to 5 loop
100         re_reg(i) <= (others => '0') after 9 ns;
101         im_reg(i) <= (others => '0') after 9 ns;
102       end loop;
103       wait_one_clk := '0';
104       DONE         <= '0' after 9 ns;
105       -- shift        <=  0;           ---NEU
106     elsif SYSCLK = '1' and SYSCLK'event then
107       if CLK = '1' then
108         if SRESET = '1' then
109           -- shift        <=  0;           ---NEU
110           for i in 1 to 5 loop
111             re_reg(i) <= (others => '0') after 9 ns;
112             im_reg(i) <= (others => '0') after 9 ns;
113           end loop;
114           DONE <= '0' after 9 ns;
115         elsif multiply_done  = '1' then
116           if wait_one_clk = '0' then
117             wait_one_clk := '1';
118             DONE         <= '0' after 9 ns;
119             if COS = '1' then
120               tempvar := multiply_value_out + re_reg(index);
121               if tempvar(32) = not tempvar(31) then
122                 for i in 1 to 5 loop  --shift right /w sign extension
123                   re_reg(i) <= re_reg(i)(32) & re_reg(i)(32 downto 1) after 9 ns;
124                   im_reg(i) <= im_reg(i)(32) & im_reg(i)(32 downto 1) after 9 ns;
125                 end loop;
126                   re_reg(index) <= tempvar srl 1 after 9 ns;
127                   -- shift <= shift + 1;           ---NEU
128               else
129                 re_reg(index) <= tempvar after 9 ns;
130               end if;
```

```
131             else
132               tempvar:=im_reg(index) + multiply_value_out;
133               if tempvar(32) = not tempvar(31) then
134                 for i in 1 to 5 loop   --shift right /w sign extension
135                   re_reg(i) <= re_reg(i)(32) & re_reg(i)(32 downto 1) after 9 ns;
136                   im_reg(i) <= im_reg(i)(32) & im_reg(i)(32 downto 1) after 9 ns;
137                 end loop;
138                 im_reg(index) <= tempvar srl 1 after 9 ns;
139                 -- shift <= shift + 1;                  --NEU
140               else
141                 im_reg(index) <= tempvar after 9 ns;
142               end if;
143             end if; --COS
144           else
145             DONE <= '1';
146           end if; --wait_one_clk
147         else
148           wait_one_clk := '0';
149           DONE <= '0' after 9 ns;
150         end if;       -- SRESET
151       end if;         -- CLK
152     end if;           -- RESET
153   end process;

155  MULTIPLY_COMPONENT : multiply
156   port map
157   (
158   --<debug>
159     DB_BUS_MULTIPLY => DB_BUS_MULTIPLY,
160   --</debug>
161     SYSCLK => SYSCLK,
162     RESET  => RESET,
163     CLK    => CLK,

165     SRESET => multiply_reset,
166     SET    => multiply_set,
167     DONE   => multiply_done,

169     LUT_IN    => factor_in_u,
170     ADC_IN    => ADC_IN,
171     VALUE_OUT => multiply_value_out_u
172   );
173   end behaviour;
```

multiply.vhd

```
1  --#######################################################
2  --##  Project name : adc_sample_calc_thd
```

```
3  --##  File name    : multiply_add.vhd
4  --##  Author       : Jan-Heiner Dreschhoff

6  --<makepackage>
7  --  <component = multiply>
8  --    <name = dreschhoff>
9  --    <project = thd>
10 --  </component>
11 --</makepackage>

13 library ieee;
14   use ieee.std_logic_1164.all;
15   use ieee.numeric_std.all;
16 library work;
17   use work.global_pkg.all;
18   use work.operators_pkg.all;
19   use work.functions_pkg.all;


22 entity multiply is
23   port
24   (
25   --<debug>
26     DB_BUS_MULTIPLY :  out std_logic_vector(15 downto 0);
27   --</debug>
28     SYSCLK  : in bit;
29     RESET   : in bit;
30     CLK     : in bit;

32     SRESET   : in bit;
33     SET     : in bit;
34     DONE    : out bit;

36     LUT_IN    : in signed(11 downto 0);
37     ADC_IN    : in unsigned(11 downto 0);
38     VALUE_OUT : out signed(23 downto 0)
39   );

41 end multiply;

43  architecture sequential_multiplication of multiply is
44     --constant testvalue     : signed(11 downto 0)  := x"003";
45     signal value           : signed(23 downto 0);
46     signal lut_in_next_val : signed(23 downto 0);
47     signal next_value      : signed(23 downto 0);
48     signal index           : integer range 0 to 12 := 0;

50   --<debug>
51     signal db_index : unsigned(3 downto 0);
52   --</debug>
```

```
55  begin

57   VALUE_OUT  <= value;
58   --<debug>
59   DB_BUS_MULTIPLY(15 downto 12) <= "0000";
60   DB_BUS_MULTIPLY(11 downto  4) <= std_logic_vector(value(7 downto 0));
61   DB_BUS_MULTIPLY( 3 downto  0) <= "0000";
62   --</debug>

64   process(SYSCLK, RESET)
65     variable lut_var : signed (23 downto 0):=x"000000";
66   begin
67     if RESET = RESET_VAL then
68       index <= 0;
69       db_index <= (others => '0');
70       DONE <= '0';
71     elsif SYSCLK = '1' and SYSCLK'event then
72       if CLK = '1' then
73         lut_var(23 downto 12) := (others => '0');
74         lut_var(11 downto  0) := LUT_IN;
75         if SRESET = '1' then
76           DONE <= '0';
77           index <= 0;
78           value <= (others => '0');
79           db_index <= (others => '0');
80         elsif SET = '1' then
81           if index = 12 then
82             DONE <= '1';
83           else
84             DONE <= '0';
85             if ADC_IN(index) = '1' then
86               value <= value + (lut_var sll index);--next_value;
87             end if;
88             index <= index + 1;
89             db_index <= db_index + 1;
90           end if;
91         else
92           DONE <= '0';
93           value <= (others => '0');
94           index <= 0;
95           db_index <= (others => '0');
96         end if;
97       end if;
98     end if;
99   end process ;

101 end sequential_multiplication;
```

B.1.8 TL_calc_thd.vhd

```
--#########################################################
--##  Project name : adc_sample_calc_thd
--##  File name    : TL_calc_thd.vhd
--##  Author       : Jan-Heiner Dreschhoff

--<makepackage>
--  <component = TL_calc_thd>
--    <name = dreschhoff>
--    <project = sample_calc_thd>
--  </component>
--</makepackage>

library ieee;
  use ieee.std_logic_1164.all;
  use ieee.numeric_std.all;

library work;
  use work.global_pkg.all;
  use work.operators_pkg.all;
  use work.functions_pkg.all;
  use work.add_samples_pkg.all;
  use work.calc_thd_in_reg_pkg.all;
  use work.calc_fsm_pkg.all;
  use work.calc_thd_out_reg_pkg.all;
  use work.calc_div_pkg.all;
  use work.calc_sqrt_pkg.all;

entity TL_calc_thd is
    port(
    --<debug>
      DB_CALC_IN_REG     : out std_logic_vector(15 downto 0);
      DB_CALC_FSM_STATE  : out std_logic_vector( 3 downto 0);
      DB_CALC_FSM        : out std_logic_vector(15 downto 0);
      DB_ADD_SAMPLES     : out std_logic_vector(15 downto 0);
      DB_SQUARE          : out std_logic_vector(15 downto 0);
      DB_CALC_DIV        : out std_logic_vector(15 downto 0);
      DB_CALC_SQRT       : out std_logic_vector(15 downto 0);
      DB_SCALE_DOWN      : out std_logic_vector(15 downto 0);
      DB_CALC_OUT_REG    : out std_logic_vector(15 downto 0);
      DB_8_BYTES_OUT     : out std_logic_vector(63 downto 0);
    --</debug>
      SYSCLK          : in bit;
      CLK             : in bit;
      RESET           : in bit;

      NEW_DATA        : in bit;
      DATA_ERROR      : in bit;
```

```
49        RE                : in MulArray;
50        IM                : in MulArray;

52        NEW_THD           : out bit;
53        THD_ERROR         : out bit;
54        THD               : out unsigned(15 downto 0);

56        THD_OUT           : out unsigned(2 downto 0)
57          );
58  end TL_calc_thd;

60  architecture structure of TL_calc_thd is

62  signal fsm_out_reg_new_thd :  bit;

64  signal fsm_reg_in_reset  : bit;
65  signal reg_in_add_re     : MulArray;
66  signal reg_in_add_im     : MulArray;

68  signal fsm_add_set       : bit;
69  signal fsm_add_reset     : bit;
70  signal add_fsm_done      : bit;
71  signal add_fsm_all_done  : bit;
72  signal fsm_add_index_U   : unsigned(2 downto 0);
73  signal fsm_add_index_STD : std_logic_vector(2 downto 0);
74  signal fsm_add_re_not_im : std_logic;
75  signal numerator         : unsigned(68 downto 0);
76  signal denominator       : unsigned(68 downto 0);

78  signal fsm_calc_div_set    : bit;
79  signal fsm_calc_div_reset  : bit;
80  signal calc_div_fsm_done   : bit;
81  signal calc_div_result     : U32;

83  signal fsm_calc_sqrt_set    : bit;
84  signal fsm_calc_sqrt_reset  : bit;
85  signal calc_sqrt_fsm_done   : bit;
86  signal calc_sqrt_result     : unsigned(15 downto 0);

88  signal fsm_scale_down_set     : bit;
89  signal fsm_scale_down_reset   : bit;
90  signal scale_down_fsm_done    : bit;
91  signal scale_down_result      : unsigned(31 downto 0);
92  signal scale_down_numerator   : unsigned(68 downto 0);
93  signal scale_down_denominator : unsigned(68 downto 0);

95  signal fsm_new_thd       : bit;
96  signal thd_error_sig     : bit;
97  signal thd_valid         : bit;
98  signal fsm_out_reg_reset : bit;
```

```
101 begin
102
103     fsm_add_index_STD <= std_logic_vector(fsm_add_index_U);
104     fsm_new_thd       <= fsm_out_reg_new_thd after 9 ns;
105     NEW_THD           <= fsm_out_reg_new_thd after 9 ns;
106     THD               <= scale_down_result(31 downto 16) after 9 ns;
107     thd_valid         <= not thd_error_sig after 9 ns;
108     THD_ERROR         <= thd_error_sig;
109
110     DB_8_BYTES_OUT <= std_logic_vector(numerator(63 downto 0));    -- to RS232
111
112 --z_____________________________________________________________FSM
113   FSM : calc_fsm
114     port map(
115       --<debug>
116         DB_CALC_FSM_STATE => DB_CALC_FSM_STATE,
117         DB_CALC_FSM       => DB_CALC_FSM,
118       --</debug>
119         SYSCLK   => SYSCLK,
120         RESET    => RESET,
121         CLK      => CLK,
122
123         NEW_DATA => NEW_DATA,
124         ERROR_IN => DATA_ERROR,
125
126         RESET_IN_REG => fsm_reg_in_reset,
127
128         SET_ADD      => fsm_add_set,
129         RESET_ADD    => fsm_add_reset,
130         ADD_DONE     => add_fsm_done,
131         ADD_ALL_DONE => add_fsm_all_done,
132
133         SET_CALC_DIV   => fsm_calc_div_set,
134         RESET_CALC_DIV => fsm_calc_div_reset,
135         CALC_DIV_DONE  => calc_div_fsm_done,
136
137         SET_CALC_SQRT   => fsm_calc_sqrt_set,
138         RESET_CALC_SQRT => fsm_calc_sqrt_reset,
139         CALC_SQRT_DONE  => calc_sqrt_fsm_done,
140
141         SET_SCALE_DOWN   => fsm_scale_down_set,
142         RESET_SCALE_DOWN => fsm_scale_down_reset,
143         SCALE_DOWN_DONE  => scale_down_fsm_done,
144
145         NEW_THD       => fsm_out_reg_new_thd,
146         THD_ERROR     => thd_error_sig,
147         RESET_OUT_REG => fsm_out_reg_reset
148     );
```

```
149 --z_____________________________________________________IN_REG
150     IN_REG : calc_thd_in_reg
151     port map(
152       --<debug>
153         DB_CALC_IN_REG => DB_CALC_IN_REG,
154       --</debug
155         SYSCLK   => SYSCLK,
156         RESET    => RESET,
157         CLK      => CLK,
158
159         SET     => NEW_DATA,
160         SRESET  => fsm_reg_in_reset,
161
162         RE_IN_REG_i => RE ,
163         IM_IN_REG_i => IM,
164
165         RE_IN_REG_o => reg_in_add_re,
166         IM_IN_REG_o => reg_in_add_im
167         );
168 --z_____________________________________________ADD_SAMPLES
169     ADD_SAMPLES_COMP : add_samples
170       port map(
171       --<debug>
172         DB_ADD_SAMPLES => DB_ADD_SAMPLES,
173         DB_SQUARE      => DB_SQUARE,
174       --</debug>>
175         SYSCLK   => SYSCLK,
176         RESET    => RESET,
177         CLK      => CLK,
178
179         SET      => fsm_add_set,
180         SRESET   => fsm_add_reset,
181         DONE     => add_fsm_done,
182         ALL_DONE => add_fsm_all_done,
183
184         RE       => reg_in_add_re,
185         IM       => reg_in_add_im,
186
187         NUMERATOR      => numerator,
188         DENOMINATOR    => denominator
189         );
190
191 --z_____________________________________________ADD_SAMPLES
192     CALC_DIV_COMP : calc_div
193       port map(
194       --<debug>
195         DB_CALC_DIV => DB_CALC_DIV,
196       --</debug>
197         SYSCLK   => SYSCLK,
198         RESET    => RESET,
```

```
199         CLK      => CLK,

201         SET     => fsm_calc_div_set,
202         SRESET  => fsm_calc_div_reset,
203         DONE    => calc_div_fsm_done,

205         NUMERATOR   => numerator,--x"0400000000000000",--
206         DENOMINATOR => denominator,--x"0800000000000000",--

208         RESULT => calc_div_result
209         );
210 --z_____________________________________________CALC_SQRT
211     CALC_SQRT_COMP : calc_sqrt
212     port map(
213     --<debug>
214       DB_CALC_SQRT => DB_CALC_SQRT,
215     --</debug>
216       SYSCLK   => SYSCLK,
217       RESET    => RESET,
218       CLK      => CLK,

220       SET     => fsm_calc_sqrt_set,
221       SRESET  => fsm_calc_sqrt_reset,
222       DONE    => calc_sqrt_fsm_done,

224       VAL_IN  => calc_div_result,--x"59000000",--
225       VAL_OUT => calc_sqrt_result

227     );

229   scale_down_denominator(68)          <= '0';
230   scale_down_denominator(67 downto 0) <= x"0000000000000B505";
231   scale_down_numerator(68 downto 16)  <= (others => '0');
232   scale_down_numerator(15 downto  0)  <= calc_sqrt_result;
233 -- --z_____________________________________________SCALE_DOWN
234     SCALE_DOWN : calc_div
235       port map(
236       --<debug>
237         DB_CALC_DIV => open,
238       --</debug>
239         SYSCLK   => SYSCLK,
240         RESET    => RESET,
241         CLK      => CLK,

243         SET     => fsm_scale_down_set,
244         SRESET  => fsm_scale_down_reset,
245         DONE    => scale_down_fsm_done,

247         NUMERATOR   => scale_down_numerator,
248         DENOMINATOR => scale_down_denominator,
```

```
250         RESULT => scale_down_result
251         );
252   DB_SCALE_DOWN <= std_logic_vector(scale_down_result(31 downto 16));

254 --z_____________________________________________________ADD_SAMPLES
255     OUT_REG : calc_thd_out_reg
256       port map(
257       --<debug>
258         DB_CALC_OUT_REG => DB_CALC_OUT_REG,
259       --</debug>
260         SYSCLK   => SYSCLK,
261         RESET    => RESET,
262         CLK      => CLK,

264         NEW_THD   => fsm_new_thd,
265         THD_VALID => thd_valid,

267         SRESET    => fsm_out_reg_reset,

269         THD_IN    => scale_down_result(31 downto 16),
270         THD_OUT   => THD_OUT
271         );

273 end structure;
```

B.1.8.1 calc_fsm.vhd

```
1  --#######################################################
2  --##  Project name : adc_sample_calc_thd
3  --##  File name    : calc_fsm.vhd
4  --##  Author       : Jan-Heiner Dreschhoff

6  --<makepackage>
7  --  <global>
8  --      type CALC_STATE_TYPE is (POR, IDLE, SET_IN_REG_STATE,
9  --            SET_ADD_STATE, ADD_DONE_STATE, ADD_ALL_DONE_STATE,
10 --            SET_CALC_DIV_STATE, SET_CALC_SQRT_STATE,
11 --            SET_SCALE_DOWN_STATE, SUCCESS, FAIL);
12 --  </global>
13 --  <component = calc_fsm>
14 --    <name = dreschhoff>
15 --    <project = calc_thd>
16 --  </component>
17 --</makepackage>

19 library ieee;
20   use ieee.std_logic_1164.all;
```

```
21   use ieee.numeric_std.all;
22 library work;
23   use work.global_pkg.all;
24   use work.operators_pkg.all;
25   use work.functions_pkg.all;

27 entity calc_fsm is
28       port(
29       --<debug>
30         DB_CALC_FSM_STATE  : out std_logic_vector(3 downto 0);
31         DB_CALC_FSM        : out std_logic_vector(15 downto 0);
32       --</debug>
33         SYSCLK    : in bit;
34         RESET     : in bit;
35         CLK       : in bit;

37         NEW_DATA  : in bit;
38         ERROR_IN  : in bit;

40         RESET_IN_REG : out bit;

42         SET_ADD   : out bit;
43         RESET_ADD : out bit;
44         ADD_DONE  : in bit;
45         ADD_ALL_DONE  : in bit;

47         SET_CALC_DIV   : out bit;
48         RESET_CALC_DIV : out bit;
49         CALC_DIV_DONE  : in bit;

51         SET_CALC_SQRT   : out bit;
52         RESET_CALC_SQRT : out bit;
53         CALC_SQRT_DONE  : in bit;

55         SET_SCALE_DOWN   : out bit;
56         RESET_SCALE_DOWN : out bit;
57         SCALE_DOWN_DONE  : in bit;

59         NEW_THD   : out bit;
60         THD_ERROR : out bit;
61         RESET_OUT_REG : out bit
62         );
63 end calc_fsm;

65 architecture behavior of calc_fsm is
66     attribute safe_implementation: string;
67     signal STATE      : CALC_STATE_TYPE := POR;
68     signal NEXT_STATE : CALC_STATE_TYPE;
69     attribute safe_implementation of STATE: signal is "yes";
```

```
71      --<debug>
72      -- signal SET_ADD   :  bit;
73      -- signal RESET_ADD :  bit;
74      -- signal ADD_DONE  :  bit;
75      -- signal ADD_ALL_DONE :  bit;
76      -- signal INDEX     :  unsigned(2 downto 0);
77      -- signal REnotIM   :  std_logic;

79      -- signal SET_SCALE_DOWN   :  bit;
80      -- signal RESET_SCALE_DOWN :  bit;
81      -- signal SCALE_DOWN_DONE  :  bit;

83      -- signal SET_CALC_SQRT   :  bit;
84      -- signal RESET_CALC_SQRT :  bit;
85      -- signal CALC_SQRT_DONE  :  bit;

87      -- signal NEW_THD   :  bit;
88      -- signal THD_ERROR :  bit;
89      -- signal RESET_OUT_REG :  bit;
90      --</debug>

92  begin

94      --NEW_THD <= CALC_THD_DONE after 9 ns;
95      STATE_REG : process(SYSCLK, RESET)
96      begin
97        if RESET = RESET_VAL then
98          STATE <= POR after 9 ns;
99        elsif SYSCLK = '1' and SYSCLK'event then
100         if CLK = '1' then
101           STATE <= NEXT_STATE;
102       end if;    --CLK
103     end if;      --RESET
104   end process;

106   FSM : process(STATE, NEW_DATA, ERROR_IN, ADD_DONE, CALC_DIV_DONE, CALC_SQRT_DONE,
107   variable once : bit;
108   begin
109       --## DEFAULT WERTE
110     --<debug>
111       DB_CALC_FSM_STATE <= x"F";
112     --</debug>
113       RESET_IN_REG    <= '0' after 9 ns;
114       SET_ADD         <= '0' after 9 ns;
115       RESET_ADD       <= '0' after 9 ns;
116       SET_CALC_DIV    <= '0' after 9 ns;
117       RESET_CALC_DIV  <= '0' after 9 ns;
118       SET_CALC_SQRT   <= '0' after 9 ns;
119       RESET_CALC_SQRT <= '0' after 9 ns;
120       SET_SCALE_DOWN  <= '0' after 9 ns;
```

```
121         RESET_SCALE_DOWN<= '0' after 9 ns;
122         NEW_THD         <= '0' after 9 ns;
123         THD_ERROR       <= '0' after 9 ns;
124         RESET_OUT_REG   <= '0' after 9 ns;

126         case STATE is
127             when POR =>
128             --<debug>
129               DB_CALC_FSM_STATE <= x"1";
130             --</debug>
131               RESET_IN_REG     <= '1'  after 9 ns;
132               RESET_CALC_SQRT  <= '1' after 9 ns;
133               RESET_CALC_DIV   <= '1' after 9 ns;
134               NEXT_STATE       <= IDLE after 9 ns;
135               RESET_OUT_REG    <= '1' after 9 ns;
136               RESET_SCALE_DOWN <= '1' after 9 ns;
137             when IDLE =>
138             --<debug>
139               DB_CALC_FSM_STATE <= x"2";
140             --</debug>
141               --counter         <= "1011" after 9 ns;
142               RESET_CALC_DIV  <= '1' after 9 ns;
143               RESET_CALC_SQRT <= '1' after 9 ns;
144               RESET_ADD       <= '1' after 9 ns;
145               if NEW_DATA = '1' then
146                 NEXT_STATE <= SET_ADD_STATE after 9 ns;
147               elsif ERROR_IN = '1' then
148                 NEXT_STATE <= FAIL after 9 ns;
149               else
150                 NEXT_STATE <= IDLE after 9 ns;
151               end if;
152             when SET_ADD_STATE =>
153             --<debug>
154               DB_CALC_FSM_STATE <= x"3";
155             --</debug>
156               SET_ADD <= '1' after 9 ns;
157               once := '0' ;
158               if ERROR_IN = '1' then -- or NEW_DATA = '1' then
159                 NEXT_STATE <= FAIL after 9 ns;
160               elsif ADD_DONE = '1' then
161                 if ADD_ALL_DONE = '1' then
162                   NEXT_STATE <= ADD_ALL_DONE_STATE after 9 ns;
163                 else
164                   NEXT_STATE <= ADD_DONE_STATE after 9 ns;
165                 end if;
166               else
167                 NEXT_STATE <= SET_ADD_STATE after 9 ns;
168               end if;
169             when ADD_DONE_STATE =>
170             --<debug>
```

```
171               DB_CALC_FSM_STATE <= x"4";
172             --</debug>
173               if ERROR_IN = '1' then
174                 NEXT_STATE <= FAIL after 9 ns;
175               elsif ADD_DONE = '0' then
176                 NEXT_STATE <= SET_ADD_STATE after 9 ns;
177               else
178                 NEXT_STATE <= ADD_DONE_STATE after 9 ns;
179               end if;
180             when ADD_ALL_DONE_STATE =>
181             --<debug>
182               DB_CALC_FSM_STATE <= x"8";
183             --</debug>
184               if ERROR_IN = '1' then
185                 NEXT_STATE <= FAIL after 9 ns;
186               elsif ADD_ALL_DONE = '0' then
187                 NEXT_STATE <= SET_CALC_DIV_STATE after 9 ns;
188               else
189                 NEXT_STATE <= ADD_ALL_DONE_STATE after 9 ns;
190               end if;
191             when SET_CALC_DIV_STATE =>
192             --<debug>
193               DB_CALC_FSM_STATE <= x"5";
194             --</debug>
195               SET_CALC_DIV <= '1' after 9 ns;
196               RESET_SCALE_DOWN<= '1' after 9 ns;
197               if ERROR_IN = '1' then
198                 NEXT_STATE <= FAIL after 9 ns;
199               elsif CALC_DIV_DONE = '1' then
200                 NEXT_STATE <= SET_CALC_SQRT_STATE after 9 ns;
201               else
202                 NEXT_STATE <= SET_CALC_DIV_STATE after 9 ns;
203               end if;
204             when SET_CALC_SQRT_STATE =>
205             --<debug>
206               DB_CALC_FSM_STATE <= x"6";
207             --</debug>
208               SET_CALC_SQRT <= '1' after 9 ns;
209               if ERROR_IN = '1' then -- or NEW_DATA = '1' then
210                 NEXT_STATE <= FAIL after 9 ns;
211               elsif CALC_SQRT_DONE = '1' then
212                 NEXT_STATE <= SET_SCALE_DOWN_STATE after 9 ns;
213                 -- NEXT_STATE <= IDLE after 9 ns;
214               else
215                 NEXT_STATE <= SET_CALC_SQRT_STATE after 9 ns;
216               end if;
217             when SET_SCALE_DOWN_STATE =>
218             --<debug>
219               DB_CALC_FSM_STATE <= x"9";
220             --</debug>
```

```
221               SET_SCALE_DOWN <= '1' after 9 ns;
222               if ERROR_IN = '1' then
223                 NEXT_STATE <= FAIL after 9 ns;
224               elsif SCALE_DOWN_DONE = '1' then
225                 NEXT_STATE <= SUCCESS after 9 ns;
226                 NEW_THD <= '1' after 9 ns;
227               else
228                 NEXT_STATE <= SET_SCALE_DOWN_STATE after 9 ns;
229               end if;
230             when SUCCESS =>
231             --<debug>
232               DB_CALC_FSM_STATE <= x"7";
233             --</debug>
234               NEW_THD <= '1' after 9 ns;
235               if ERROR_IN = '1' then
236                 NEXT_STATE <= FAIL after 9 ns;
237               else
238                 NEXT_STATE <= IDLE after 9 ns;
239               end if;
240             when FAIL =>
241             --<debug>
242               DB_CALC_FSM_STATE <= x"D";
243             --</debug>
244               THD_ERROR <= '1' after 9 ns;
245               RESET_CALC_DIV  <= '1' after 9 ns;
246               RESET_CALC_SQRT <= '1' after 9 ns;
247               RESET_ADD       <= '1' after 9 ns;
248               if NEW_DATA = '1' then
249                 NEXT_STATE <= SET_ADD_STATE after 9 ns;
250               else --ERROR_IN = '1' then
251                 NEXT_STATE <= FAIL after 9 ns;
252               -- else
253                 -- NEXT_STATE <= IDLE after 9 ns;
254               end if;
255             when others =>
256             --<debug>
257               DB_CALC_FSM_STATE <= x"E";
258             --</debug>
259               NEXT_STATE <= FAIL after 9 ns;
260         end case;
261     end process;

263     --<debug = ADD_SQUARE_DUMMY>
264       -- ADD_DUMMY : process(SYSCLK, RESET)
265       -- variable counter : integer range 0 to 10 := 0;
266       -- variable clk_counter : integer range 0 to 16 := 0;
267       -- begin
268         -- if RESET = RESET_VAL then
269           -- ADD_DONE     <= '1';
270           -- ADD_ALL_DONE <= '0';
```

```
271           -- counter := 0;
272           -- clk_counter := 0;
273         -- elsif SYSCLK = '1' and SYSCLK'event then
274           -- if CLK = '1' then
275             -- if RESET_ADD = '1' then
276               -- counter := 0;
277               -- clk_counter := 0;
278               -- ADD_DONE     <= '0';
279               -- ADD_ALL_DONE <= '0';
280             -- elsif SET_ADD = '1' then
281               -- if clk_counter = 16 then
282                 -- ADD_DONE <= '1';
283                 -- if counter = 10 then
284                   -- ADD_ALL_DONE <= '1';
285                 -- else
286                   -- ADD_ALL_DONE <= '0';
287                   -- counter      := counter + 1;
288                 -- end if;
289               -- else
290                 -- clk_counter := clk_counter + 1;
291               -- end if;
292             -- else
293               -- clk_counter  := 0;
294               -- ADD_DONE     <= '0';
295               -- ADD_ALL_DONE <= '0';
296             -- end if;
297           -- end if;
298         -- end if;
299       -- end process ADD_DUMMY;
300     --</debug>

302     --<debug = DIV_DUMMY>
303       -- DIV_DUMMY : process(SYSCLK, RESET)
304         -- variable clk_counter : integer range 0 to 32 := 0;
305       -- begin
306         -- if RESET = RESET_VAL then
307           -- SCALE_DOWN_DONE <= '0';
308           -- clk_counter := 0;
309         -- elsif SYSCLK = '1' and SYSCLK'event then
310           -- if CLK = '1' then
311             -- if RESET_SCALE_DOWN = '1' then
312               -- SCALE_DOWN_DONE <= '0';
313               -- clk_counter := 0;
314             -- elsif SET_SCALE_DOWN = '1' then
315               -- if clk_counter = 32 then
316                 -- SCALE_DOWN_DONE <= '1';
317               -- else
318                 -- SCALE_DOWN_DONE <= '0';
319                 -- clk_counter := clk_counter + 1;
320               -- end if;
```

```
321             -- else
322               -- clk_counter := 0;
323               -- SCALE_DOWN_DONE <= '0';
324             -- end if;
325           -- end if;
326         -- end if;
327       -- end process DIV_DUMMY;
328     --</debug>

330     --<debug = CALC_SQRT_DUMMY>
331       -- CALC_SQRT_DUMMY : process(SYSCLK, RESET)
332         -- variable clk_counter : integer range 0 to 16 := 0;
333       -- begin
334         -- if RESET = RESET_VAL then
335           -- CALC_SQRT_DONE <= '0';
336           -- clk_counter := 0;
337         -- elsif SYSCLK = '1' and SYSCLK'event then
338           -- if CLK = '1' then
339             -- if RESET_CALC_SQRT = '1' then
340               -- CALC_SQRT_DONE <= '0';
341               -- clk_counter := 0;
342             -- elsif SET_CALC_SQRT = '1' then
343               -- if clk_counter = 16 then
344                 -- CALC_SQRT_DONE <= '1';
345               -- else
346                 -- CALC_SQRT_DONE <= '0';
347                 -- clk_counter := clk_counter + 1;
348               -- end if;
349             -- else
350               -- clk_counter := 0;
351               -- CALC_SQRT_DONE <= '0';
352             -- end if;
353           -- end if;
354         -- end if;
355       -- end process CALC_SQRT_DUMMY;
356     --</debug>

358 end behavior;
```

B.1.8.2 calc_thd_in_reg.vhd

```
1  --#######################################################
2  --##  Project name : adc_sample_calc_thd
3  --##  File name    : calc_thd_in_reg.vhd
4  --##  Author       : Jan-Heiner Dreschhoff

6  --<makepackage>
7  --  <component = calc_thd_in_reg>
```

```
8  --    <name = dreschhoff>
9  --    <project = calc_thd_in_reg>
10 --  </component>
11 --</makepackage>


14 library ieee;
15   use ieee.std_logic_1164.all;
16   use ieee.numeric_std.all;
17 library work;
18   use work.global_pkg.all;
19   use work.operators_pkg.all;
20   use work.functions_pkg.all;



24 entity calc_thd_in_reg is
25   port(
26   --<debug>
27     DB_CALC_IN_REG : out std_logic_vector(15 downto 0);
28   --</debug>
29     SYSCLK     : in bit;
30     RESET      : in bit;
31     CLK        : in bit;

33     SET        : in bit;
34     SRESET     : in bit;

36     RE_IN_REG_i : in MulArray;
37     IM_IN_REG_i : in MulArray;

39     RE_IN_REG_o : out MulArray;
40     IM_IN_REG_o : out MulArray
41         );
42 end calc_thd_in_reg;

44 architecture behavior of calc_thd_in_reg is

46 signal re_sig : MulArray;
47 signal im_sig : MulArray;

49 begin
50     RE_IN_REG_o <= re_sig after 9 ns;
51     IM_IN_REG_o <= im_sig after 9 ns;

53     process(SYSCLK, RESET)
54     begin
55       if RESET = RESET_VAL then
56         for i in 1 to 5 loop
57             re_sig(i) <= (others => '0') after 9 ns;
```

```
58              im_sig(i) <= (others => '0') after 9 ns;
59          end loop;
60        elsif SYSCLK = '1' and SYSCLK'event then
61          if CLK = '1' then
62            if SET = '1' then
63             for i in 1 to 5 loop
64                re_sig(i) <= RE_IN_REG_i(i) after 9 ns;
65                im_sig(i) <= IM_IN_REG_i(i) after 9 ns;
66              end loop;
67            elsif SRESET = '1' then
68              for i in 1 to 5 loop
69                re_sig(i) <= (others => '0') after 9 ns;
70                im_sig(i) <= (others => '0') after 9 ns;
71              end loop;
72            end if; --set
73          end if;   --CLK
74        end if;     --RESET
75      end process;

77  end behavior;
```

B.1.8.3 add_samples.vhd

```
1   --#########################################################
2   --##  Project name : adc_sample_calc_thd
3   --##  File name    : add_samples.vhd
4   --##  Author       : Jan-Heiner Dreschhoff

6   -- <makepackage>
7   --  <component = add_samples>
8   --    <name = dreschhoff>
9   --    <project = calc_thd>
10  --  </component>
11  -- </makepackage>

13  library ieee;
14    use ieee.std_logic_1164.all;
15    use ieee.numeric_std.all;

17  library work;
18    use work.global_pkg.all;
19    use work.functions_pkg.all;
20    use work.operators_pkg.all;
21    use work.square_pkg.all;

23  entity add_samples is
24      port(
25      --<debug>
```

```
26        DB_ADD_SAMPLES   : out std_logic_vector(15 downto 0);
27        DB_SQUARE        : out std_logic_vector(15 downto 0);
28      --</debug>
29        SYSCLK           : in bit;
30        RESET            : in bit;
31        CLK              : in bit;

33        SET              : in bit;
34        SRESET           : in bit;
35        DONE             : out bit;
36        ALL_DONE         : out bit;

38        RE               : in MulArray;
39        IM               : in MulArray;

41        NUMERATOR              : out unsigned(68 downto 0);
42        DENOMINATOR            : out unsigned(68 downto 0)
43      );
44  end add_samples;

46  architecture behavior of add_samples is

48  signal index     : integer range 0 to 5;
49  signal REnotIM   : bit;


52  signal result       : unsigned(63 downto 0);

54  signal u25_sig      : unsigned(68 downto 0);
55  signal next_value   : unsigned(68 downto 0);
56  signal last_value   : unsigned(68 downto 0);
57  signal uall_sig     : unsigned(68 downto 0);

59  signal square_done  : bit;
60  signal set_square   : bit;
61  signal done_sig     : bit;
62  signal all_done_sig : bit;
63  signal once         : bit;

65  signal val_in       : U32;
66  signal setting_u25  : std_logic;
67  signal setting_uall : std_logic;

69  begin
70    --<debug>
71      DB_ADD_SAMPLES(15 downto 12) <= (others => '0');
72      -- DB_ADD_SAMPLES(11)     <= '0';
73     DB_ADD_SAMPLES(11 downto 4) <= std_logic_vector(u25_sig(7 downto 0));
74      DB_ADD_SAMPLES( 3 downto  0) <= (others => '0');
75    --</debug>
```

```
78   NUMERATOR <= u25_sig after 9 ns;
79   DENOMINATOR <= uall_sig after 9 ns;

81   next_value <= u25_sig  + ("0000" & result);
82   last_value <= uall_sig + ("0000" & result);



86   ADD_DUMMY : process(SYSCLK, RESET)
87     variable counter : integer range 0 to 15 := 0;
88   begin
89     if RESET = RESET_VAL then
90       DONE     <= '1';
91       ALL_DONE <= '0';
92       index    <=  5 ;
93       REnotIM  <= '0';
94       once     <= '0';
95       uall_sig <= (others => '0');
96       u25_sig  <= (others => '0');
97     elsif SYSCLK = '1' and SYSCLK'event then
98       if CLK = '1' then
99         setting_uall <= '0';
100        setting_u25 <= '0';
101        if SRESET = '1' then
102          DONE     <= '0';
103          ALL_DONE <= '0';
104          once     <= '0';
105          index   <= 5;
106          REnotIM <= '0';
107          uall_sig <= (others => '0');
108          u25_sig  <= (others => '0');
109        elsif square_done = '1' then
110          if once = '0' then
111            REnotIM <= not REnotIM;
112            if REnotIM = '1' then
113              index <= index - 1;
114            end if;
115            DONE <= '1';
116            ALL_DONE <= '0';
117            if index = 1 then
118              uall_sig <= uall_sig + ("0000" & result);
119              if REnotIM = '1' then
120                -- uall_sig <= last_value;
121                ALL_DONE <= '1';
122                setting_uall <= '1';
123              -- else
124                -- uall_sig <= next_value;
125                -- setting_uall <= '1';
```

```
126                 -- ALL_DONE <= '0';
127               end if;
128             else
129               -- ALL_DONE <= '0';
130               setting_u25 <= '1';
131               -- u25_sig  <= next_value;
132               u25_sig <= u25_sig + ("0000" & result);
133               uall_sig <= uall_sig + ("0000" & result);
134             end if;
135             once <= '1';
136           end if;
137         else
138           once <= '0';
139           DONE     <= '0';
140           ALL_DONE <= '0';
141         end if;
142       end if;
143     end if;
144   end process ADD_DUMMY;


147   process(RE, IM, index, REnotIM)
148   begin
149     if REnotIM = '1' then
150       val_in <= RE(index) after 9 ns;
151     else
152       val_in <= IM(index) after 9 ns;
153     end if;
154   end process;


157   SQR : square
158   port map(
159   --<debug>
160     DB_SQUARE => DB_SQUARE,
161   --</debug>
162     SYSCLK => SYSCLK,
163     RESET   => RESET,
164     CLK     => CLK,

166     SET     => SET,
167     SRESET  => SRESET,
168     DONE    => square_done,

170     VAL_IN    => val_in,
171     VAL_OUT   => result
172   );

174 end behavior;
```

square.vhd

```
--#########################################################
--##  Project name : adc_sample_calc_thd
--##  File name    : square.vhd
--##  Author       : Jan-Heiner Dreschhoff


-- <makepackage>
--  <component = square>
--    <name = dreschhoff>
--    <project = calc_thd>
--  </component>
-- </makepackage>

library ieee;
  use ieee.std_logic_1164.all;
  use ieee.numeric_std.all;

library work;
  use work.global_pkg.all;
  use work.functions_pkg.all;
  use work.operators_pkg.all;

entity square is
    port(
    --<debug>
      DB_SQUARE       : out std_logic_vector(15 downto 0);
    --</debug>
      SYSCLK          : in bit;
      RESET           : in bit;
      CLK             : in bit;

      SET             : in bit;
      SRESET          : in bit;
      DONE            : out bit;

      VAL_IN          : in U32;
      VAL_OUT         : out unsigned(63 downto 0)
      );
end square;

architecture behavior of square is

    signal result : unsigned(63 downto 0);
    signal index : integer range 0 to 32;  --64
    signal done_sig : std_logic;

begin
  --<debug>
```

```
49      DB_SQUARE(15 downto 12) <= (others => '0');
50      -- DB_SQUARE(11) <= done_sig;
51      DB_SQUARE( 11 downto  4) <= std_logic_vector(result(39 downto 32));
52      DB_SQUARE( 3 downto  0) <= (others => '0');
53    --</debug>

55      VAL_OUT <= result;
56      --DONE <= to_bit(done_sig);
57    process(SYSCLK, RESET)
58    variable val_in_var : unsigned (63 downto 0):= (others => '0');
59    begin
60      if RESET = RESET_VAL then
61        index <= 0;
62        DONE <= '0';
63        done_sig <= '0';
64      elsif SYSCLK = '1' and SYSCLK'event then
65        if CLK = '1' then
66          val_in_var(63 downto 32) := (others => '0');
67          val_in_var(31 downto  0) := VAL_IN;
68          if SRESET = '1' then
69            DONE <= '0';
70            done_sig <= '0';
71            index <= 0;
72            result <= (others => '0');
73          elsif SET = '1' then
74            if index = 32 then
75              DONE <= '1';
76              done_sig <= '1';
77            else
78              DONE <= '0';
79              done_sig <= '0';
80              if VAL_IN(index) = '1' then
81                result <= result + (val_in_var sll index);
82              end if;
83              index <= index + 1;
84            end if;
85          else
86            DONE <= '0';
87            done_sig <= '0';
88            result <= (others => '0');
89            index <= 0;
90          end if;
91        end if;
92      end if;
93    end process ;

95  end behavior;
```

B.1.8.4 calc_div.vhd

```
1  --#########################################################
2  --##  Project name : adc_sample_calc_thd
3  --##  File name    : calc_div.vhd
4  --##  Author       : Jan-Heiner Dreschhoff
5
6  --<makepackage>
7  --  <component = calc_div>
8  --    <name = dreschhoff>
9  --    <project = calc_div>
10 --  </component>
11 --</makepackage>
12
13
14 library ieee;
15   use ieee.std_logic_1164.all;
16   use ieee.numeric_std.all;
17 library work;
18   use work.global_pkg.all;
19   use work.operators_pkg.all;
20   use work.functions_pkg.all;
21
22 entity calc_div is
23   port(
24   --<debug>
25     DB_CALC_DIV : out std_logic_vector(15 downto 0);
26   --</debug>
27     SYSCLK    : in bit;
28     RESET     : in bit;
29     CLK       : in bit;
30
31     SET       : in bit;
32     SRESET    : in bit;
33     DONE      : out bit;
34
35     NUMERATOR   : in unsigned(68 downto 0);
36     DENOMINATOR : in unsigned(68 downto 0);
37
38     RESULT      : out U32
39     );
40 end calc_div;
41
42 architecture behavior of calc_div is
43
44 signal once : bit;
45 signal num  : unsigned(69 downto 0);
46 signal den  : unsigned(68 downto 0);
47 signal res  : U32;
48 signal cnt  : integer range 0 to 32;
```

```
49  begin

51    --<debug>
52      DB_CALC_DIV(15 downto 12) <= (others => '0');
53      DB_CALC_DIV(11 downto  4) <= std_logic_vector(res(7 downto 0));
54      DB_CALC_DIV( 3 downto  0) <= (others => '0');
55    --</debug>

57      process(SYSCLK, RESET)
58        variable temp_num : unsigned(32 downto 0);
59      begin
60        if RESET = RESET_VAL then
61          cnt <= 31 after 9 ns;
62          DONE <= '0' after 9 ns;
63          once <= '0' after 9 ns;
64          res <= (others => '0') after 9 ns;
65          num <= (others => '0') after 9 ns;
66          den <= (others => '0') after 9 ns;
67          RESULT <= (others => '0') after 9 ns;
68        elsif SYSCLK = '1' and SYSCLK'event then
69          if CLK = '1' then
70            if SRESET = '1' then
71              once <= '0' after 9 ns;
72              res <= (others => '0') after 9 ns;
73              num <= (others => '0') after 9 ns;
74              den <= (others => '0') after 9 ns;
75              DONE <= '0' after 9 ns;
76              RESULT <= (others => '0') after 9 ns;
77            elsif once = '0' then
78              if SET = '1' then
79                num <= '0' & NUMERATOR after 9 ns;
80                den <= DENOMINATOR after 9 ns;
81                cnt <= 31 after 9 ns;
82                once <= '1' after 9 ns;
83              end if;      --SET
84            else
85              if cnt = 0 then
86                DONE <= '1' after 9 ns;
87                RESULT <= res after 9 ns;
88              else
89                RESULT <= (others => '0') after 9 ns;
90                if num >= den then
91                  temp_num := (num - den);------------test
92                  num <= temp_num sll 1 after 9 ns;
93                  res(cnt) <= '1' after 9 ns;
94                else
95                  num <= num sll 1 after 9 ns;
96                  res(cnt-1) <= '0' after 9 ns;
97                end if;
98                cnt <= cnt - 1 after 9 ns;
```

```
99              end if; --counter
100           end if;   --SRESET
101         end if;   --CLK
102       end if;     --RESET
103     end process;
104 end behavior;
```

B.1.8.5 calc_sqrt.vhd

```
1  --#########################################################
2  --##  Project name : adc_sample_calc_thd
3  --##  File name    : calc_sqrt.vhd
4  --##  Author       : Jan-Heiner Dreschhoff

6  --<makepackage>
7  --  <component = calc_sqrt>
8  --    <name = dreschhoff>
9  --    <project = thd>
10 --  </component>
11 --</makepackage>


14 library IEEE;
15   use IEEE.std_logic_1164.all;
16   use ieee.numeric_std.all;     -- for UNSIGNED
17 library work;
18   use work.global_pkg.all;
19   use work.operators_pkg.all;
20   use work.functions_pkg.all;

22 entity calc_sqrt is
23   port(
24   --<debug>
25     DB_CALC_SQRT : out std_logic_vector(15 downto 0);
26   --</debug>
27     SYSCLK : in bit;
28     CLK    : in bit;
29     RESET  : in bit;

31     SET    : in bit;
32     SRESET  : in bit;
33     DONE   : out bit;

35     VAL_IN : in U32;
36     VAL_OUT: out unsigned(15 downto 0)
37   );
38 end calc_sqrt;
```

```
40 architecture behaviour of calc_sqrt is

42   signal q : unsigned(15 downto 0);  --result.
43   signal a : unsigned(31 downto 0);  --original input.
44   signal i : integer range 0 to 16;

46   -- signal work_input_vector : unsigned(63 downto 0);
47   -- alias work_vec  : unsigned(31 downto 0) is work_input_vector(63 downto 32);
48   -- alias input_vec : unsigned(31 downto 0) is work_input_vector(31 downto
     0);
49   -- signal result   : unsigned(15 downto 0);
50   -- signal index    : integer range 0 to 16;
51   -- signal once     : bit;

53 begin

55   --<debug>
56     DB_CALC_SQRT(15 downto 12) <= (others => '0');
57     DB_CALC_SQRT(11 downto  4) <= std_logic_vector(q(7 downto 0));
58     DB_CALC_SQRT( 3 downto  0) <= (others => '0');
59   --</debug>

61   -- process(SYSCLK, RESET)
62    -- variable result_sll1_plus1 : unsigned(15 downto 0);
63   -- begin
64     -- if RESET = RESET_VAL then
65       -- index             <=  0;
66       -- once              <= '0';
67       -- work_input_vector <= (others => '0');
68       -- DONE <= '0';
69       -- VAL_OUT <= (others => '0');
70     -- elsif SYSCLK = '1' and SYSCLK'event then
71       -- if CLK = '1' then
72         -- result_sll1_plus1(15 downto 1) := result(14 downto 0);
73         -- result_sll1_plus1(0)           := '1';
74         -- if SRESET = '1' then
75           -- index             <=  0;
76           -- once              <= '0';
77           -- work_input_vector <= (others => '0');
78           -- DONE <= '0';
79           -- VAL_OUT <= (others => '0');
80         -- elsif SET = '1' then
81           -- if index = 16 then
82             -- DONE <= '1';
83             -- VAL_OUT <= result;
84           -- else
85             -- DONE <= '0';
86             -- if once = '0' then
87               -- once <= '1';
88               -- input_vec <= VAL_IN;
```

```
89              -- work_vec  <= (others => '0');
90              -- result    <= (others => '0');
91              -- index <= 0;
92            -- else
93              -- index <= index + 1;
94              -- if work_vec < result_sll1_plus1 then
95                -- result(15 downto 1) <= result(14 downto 0);
96                -- result(0)           <= '0';
97                -- work_input_vector <= work_input_vector sll 2;
98              -- else
99                -- result <= result_sll1_plus1;
100               -- work_vec
101               -- work_vec
102               -- input_vec(15 downto 2) <= input_vec(13 downto 0);
103               -- input_vec(
104             -- end if;
105           -- end if;
106         -- end if;
107       -- else
108         -- once <= '0';
109         -- DONE <= '0';
110       -- end if;
111     -- end if;
112   -- end if;
113 -- end process;

115 process(SYSCLK, RESET)
116   variable left,right, r : unsigned(17 downto 0);  --input to adder/sub,r-remainder
117   variable once : bit;
118 begin
119   if RESET = RESET_VAL then
120     q     <= (others => '0') after 9 ns;
121     a     <= (others => '0') after 9 ns;
122     i     <= 0 after 9 ns;
123     r     := (others => '0');
124     right := (others => '0');
125     left  := (others => '0');
126     once  := '0';
127     VAL_OUT <= (others => '0') after 9 ns;
128     DONE   <= '0' after 9 ns;
129   elsif SYSCLK = '1' and SYSCLK'event then
130     if CLK = '1' then
131         if SRESET = '1' then
132           q     <= (others => '0') after 9 ns;
133           a     <= (others => '0') after 9 ns;
134           i     <= 0 after 9 ns;
135           r     := (others => '0');
136           right := (others => '0');
137           left  := (others => '0');
138           VAL_OUT <= (others => '0') after 9 ns;
```

```
139             DONE   <= '0' after 9 ns;
140             once  := '0';
141           elsif SET = '1' then
142             if once = '0' then
143               a <= VAL_IN after 9 ns;
144               once := '1';
145             elsif i = 16 then
146               DONE <= '1' after 9 ns;
147               VAL_OUT <= q after 9 ns;
148             else
149               right(0):='1';
150               right(1):=r(17);
151               right(17 downto 2):=q;
152               left(1 downto 0):=a(31 downto 30);
153               left(17 downto 2):=r(15 downto 0);
154               a(31 downto 2)<=a(29 downto 0);  --shifting by 2 bit.
155               if ( r(17) = '1') then
156                 r := left + right;
157               else
158                 r := left - right;
159               end if;
160               q(15 downto 1) <= q(14 downto 0);
161               q(0) <= not r(17);
162               i <= i + 1 after 9 ns;
163             end if;   -- once
164           else
165             DONE   <= '0' after 9 ns;
166           end if;   --SRESET
167       end if;  --CLK
168     end if;    --RESET
169   end process;

171 end behaviour;
```

B.1.8.6 calc_thd_out_reg.vhd

```
1  --#######################################################
2  --##  Project name : adc_sample_calc_thd
3  --##  File name    : calc_thd_out_reg.vhd
4  --##  Author       : Jan-Heiner Dreschhoff


7  --<makepackage>
8  --  <component = calc_thd_out_reg>
9  --    <name = dreschhoff>
10 --    <project = calc_thd_out_reg>
11 --  </component>
12 --</makepackage>
```

```
15 library ieee;
16   use ieee.std_logic_1164.all;
17   use ieee.numeric_std.all;
18 library work;
19   use work.global_pkg.all;
20   use work.operators_pkg.all;
21   use work.functions_pkg.all;



25 entity calc_thd_out_reg is
26   port(
27   --<debug>
28     DB_CALC_OUT_REG : out std_logic_vector(15 downto 0);
29   --</debug>
30     SYSCLK    : in bit;
31     RESET     : in bit;
32     CLK       : in bit;

34     NEW_THD   : in bit;
35     THD_VALID : in bit;

37     SRESET    : in bit;

39     THD_IN    : in unsigned(15 downto 0);
40     THD_OUT   : out unsigned(2 downto 0)
41   );
42 end calc_thd_out_reg;

44 architecture behavior of calc_thd_out_reg is

46 begin

48     process(SYSCLK, RESET)
49         variable bit1 : std_logic;
50         variable bit2 : std_logic;
51         variable bit3 : std_logic;
52     begin
53       if RESET = RESET_VAL then
54         THD_OUT <= "000" after 9 ns;
55         bit1:= '0';
56         bit2:= '0';
57         bit3:= '0';
58       elsif SYSCLK = '1' and SYSCLK'event then
59         if CLK = '1' then
60           if SRESET = '1' then
61             bit1:= '0';
62             bit2:= '0';
```

```
63              bit3:= '0';
64              THD_OUT <= "000" after 9 ns;
65            elsif THD_VALID = '0' then
66              THD_OUT <= "111" after 9 ns;
67            elsif NEW_THD = '1' then
68              if    THD_IN > x"251E" then  -- 29%
69                THD_OUT <= "111";
70              elsif THD_IN > x"1AE1" then  -- 21%
71                THD_OUT <= "110";
72              elsif THD_IN > x"11EB" then  -- 14%
73                THD_OUT <= "101";
74              elsif THD_IN > x"0A3D" then  --  8%
75                THD_OUT <= "100";
76              elsif THD_IN > x"051B" then  --  4%
77                THD_OUT <= "011";
78              elsif THD_IN > x"028F" then  --  2%
79                THD_OUT <= "010";
80              elsif THD_IN > x"0147" then  --  1%
81                THD_OUT <= "001";
82              else                         -- <1%
83                THD_OUT <= "000";
84              end if;
85            end if; --sreset thd_valid
86          end if;   --CLK
87        end if;     --RESET
88      end process;

90  end behavior;
```

B.1.9 seven_seg_mux.vhd

```
1   --#########################################################
2   --##  Project name : adc_sample_calc_thd
3   --##  File name    : seven_seg_mux.vhd
4   --##  Author       : Jan-Heiner Dreschhoff

6   --<makepackage>
7   --  <component = seven_seg_mux>
8   --    <name = dreschhoff>
9   --    <project = adc_sample_calc_thd>
10  --  </component>
11  --</makepackage>

13  library ieee;
14    use ieee.std_logic_1164.all;
15    use ieee.numeric_std.all;
16  library work;
17    use work.global_pkg.all;
```

```
18  use work.operators_pkg.all;
19  use work.functions_pkg.all;

21  entity seven_seg_mux is
22    port (
23      SWITCHES         : in bit_vector(3 downto 0);

25      CHANNEL_0        : in std_logic_vector(15 downto 0);
26      CHANNEL_1        : in std_logic_vector(15 downto 0);
27      CHANNEL_2        : in std_logic_vector(15 downto 0);
28      CHANNEL_3        : in std_logic_vector(15 downto 0);
29      CHANNEL_4        : in std_logic_vector(15 downto 0);
30      CHANNEL_5        : in std_logic_vector(15 downto 0);
31      CHANNEL_6        : in std_logic_vector(15 downto 0);
32      CHANNEL_7        : in std_logic_vector(15 downto 0);
33      CHANNEL_8        : in std_logic_vector(15 downto 0);
34      CHANNEL_9        : in std_logic_vector(15 downto 0);
35      CHANNEL_A        : in std_logic_vector(15 downto 0);
36      CHANNEL_B        : in std_logic_vector(15 downto 0);
37      -- CHANNEL_C        : in std_logic_vector(15 downto 0);
38      -- CHANNEL_D        : in std_logic_vector(15 downto 0);
39      -- CHANNEL_E        : in std_logic_vector(15 downto 0);
40      CHANNEL_F        : in std_logic_vector(15 downto 0);

42      DECIMAL_POINT_CH0 : in bit_vector(3 downto 0);
43      DECIMAL_POINT_CH1 : in bit_vector(3 downto 0);
44      DECIMAL_POINT_CH2 : in bit_vector(3 downto 0);
45      DECIMAL_POINT_CH3 : in bit_vector(3 downto 0);
46      DECIMAL_POINT_CH4 : in bit_vector(3 downto 0);
47      DECIMAL_POINT_CH5 : in bit_vector(3 downto 0);
48      DECIMAL_POINT_CH6 : in bit_vector(3 downto 0);
49      DECIMAL_POINT_CH7 : in bit_vector(3 downto 0);
50      DECIMAL_POINT_CH8 : in bit_vector(3 downto 0);
51      DECIMAL_POINT_CH9 : in bit_vector(3 downto 0);
52      DECIMAL_POINT_CHA : in bit_vector(3 downto 0);
53      DECIMAL_POINT_CHB : in bit_vector(3 downto 0);
54      -- DECIMAL_POINT_CHC : in bit_vector(3 downto 0);
55      -- DECIMAL_POINT_CHD : in bit_vector(3 downto 0);
56      -- DECIMAL_POINT_CHE : in bit_vector(3 downto 0);
57      DECIMAL_POINT_CHF : in bit_vector(3 downto 0);

59      CHANNEL_OUT       : out std_logic_vector(15 downto 0);
60      DECIMAL_POINT_OUT : out bit_vector(3 downto 0)
61    );
62    end entity;

64    architecture mux of seven_seg_mux is

66    begin
```

```
69  process(SWITCHES
70          , CHANNEL_0, DECIMAL_POINT_CH0
71          , CHANNEL_1, DECIMAL_POINT_CH1
72          , CHANNEL_2, DECIMAL_POINT_CH2
73          , CHANNEL_3, DECIMAL_POINT_CH3
74          , CHANNEL_4, DECIMAL_POINT_CH4
75          , CHANNEL_5, DECIMAL_POINT_CH5
76          , CHANNEL_6, DECIMAL_POINT_CH6
77          , CHANNEL_7, DECIMAL_POINT_CH7
78          , CHANNEL_8, DECIMAL_POINT_CH8
79          , CHANNEL_9, DECIMAL_POINT_CH9
80          , CHANNEL_A, DECIMAL_POINT_CHA
81          , CHANNEL_B, DECIMAL_POINT_CHB
82          -- , CHANNEL_C, DECIMAL_POINT_CHC
83          -- , CHANNEL_D, DECIMAL_POINT_CHD
84          -- , CHANNEL_E, DECIMAL_POINT_CHE
85          , CHANNEL_F, DECIMAL_POINT_CHF)
86  begin
87    case SWITCHES is
88      when "0000"  => CHANNEL_OUT <= CHANNEL_0;
89                      DECIMAL_POINT_OUT <= DECIMAL_POINT_CH0;
90      when "0001"  => CHANNEL_OUT <= CHANNEL_1;
91                      DECIMAL_POINT_OUT <= DECIMAL_POINT_CH1;
92      when "0010"  => CHANNEL_OUT <= CHANNEL_2;
93                      DECIMAL_POINT_OUT <= DECIMAL_POINT_CH2;
94      when "0011"  => CHANNEL_OUT <= CHANNEL_3;
95                      DECIMAL_POINT_OUT <= DECIMAL_POINT_CH3;
96      when "0100"  => CHANNEL_OUT <= CHANNEL_4;
97                      DECIMAL_POINT_OUT <= DECIMAL_POINT_CH4;
98      when "0101"  => CHANNEL_OUT <= CHANNEL_5;
99                      DECIMAL_POINT_OUT <= DECIMAL_POINT_CH5;
100     when "0110"  => CHANNEL_OUT <= CHANNEL_6;
101                     DECIMAL_POINT_OUT <= DECIMAL_POINT_CH6;
102     when "0111"  => CHANNEL_OUT <= CHANNEL_7;
103                     DECIMAL_POINT_OUT <= DECIMAL_POINT_CH7;
104     when "1000"  => CHANNEL_OUT <= CHANNEL_8;
105                     DECIMAL_POINT_OUT <= DECIMAL_POINT_CH8;
106     when "1001"  => CHANNEL_OUT <= CHANNEL_9;
107                     DECIMAL_POINT_OUT <= DECIMAL_POINT_CH9;
108     when "1010"  => CHANNEL_OUT <= CHANNEL_A;
109                     DECIMAL_POINT_OUT <= DECIMAL_POINT_CHA;
110     when "1011"  => CHANNEL_OUT <= CHANNEL_B;
111                     DECIMAL_POINT_OUT <= DECIMAL_POINT_CHB;
112     -- when "1100"  => CHANNEL_OUT <= CHANNEL_C;
113                     -- DECIMAL_POINT_OUT <= DECIMAL_POINT_CHC;
114     -- when "1101"  => CHANNEL_OUT <= CHANNEL_D;
115                     -- DECIMAL_POINT_OUT <= DECIMAL_POINT_CHD;
116     -- when "1110"  => CHANNEL_OUT <= CHANNEL_E;
117                     -- DECIMAL_POINT_OUT <= DECIMAL_POINT_CHE;
```

```
118       when "1111"  => CHANNEL_OUT <= CHANNEL_F;
119                       DECIMAL_POINT_OUT <= DECIMAL_POINT_CHF;
120       when others => CHANNEL_OUT <= CHANNEL_0;
121                       DECIMAL_POINT_OUT <= (others => '1');
122     end case;
123   end process;

125   end mux;
```

B.1.10 seven_seg.vhd

```
1  --#######################################################
2  --##  Project name : adc_sample_calc_thd
3  --##  File name    : seven_seg.vhd
4  --##  Author       : Jan-Heiner Dreschhoff

6  --<makepackage>
7  --  <component = seven_seg>
8  --    <name = dreschhoff>
9  --    <project = calc_thd>
10 --  </component>
11 --</makepackage>

13 -- the bcd algorithm: double dabble
14 -- source: http://en.wikipedia.org/wiki/Double_dabble

16 library ieee;
17   use ieee.std_logic_1164.all;
18   use ieee.numeric_std.all;
19 library work;
20   use work.global_pkg.all;
21   use work.operators_pkg.all;
22   use work.functions_pkg.all;

24 entity seven_seg is
25   port(
26     SYSCLK          : in bit;
27     CLK             : in bit;
28     RESET           : in bit;

30     BCD_ON          : in bit;
31     SEVEN_SEG_IN    : in std_logic_vector(15 downto 0);
32     DECIMAL_POINT_IN : in bit_vector(3 downto 0);
33     NEW_THD         : in bit;

35     SEVEN_SEG_out   : out unsigned(6 downto 0);
36     DECIMAL_POINT   : out bit;
37     SEVEN_SEG_SELECT : out bit_vector(3 downto 0)
```

```
38      );
39  end seven_seg;

41  architecture behavior of seven_seg is

43    signal binary        : unsigned(9 downto 0);
44    signal count_to_10   : integer range 0 to 10;

46    signal seven_seg_in_U   : unsigned(15 downto 0);
47    signal start_conversion : bit;
48    signal SEVEN_SEG_1      : unsigned(3 downto 0);
49    signal SEVEN_SEG_10     : unsigned(3 downto 0);
50    signal SEVEN_SEG_100    : unsigned(3 downto 0);
51    signal SEVEN_SEG_1000   : unsigned(3 downto 0);

53    signal seven_seg_sig    : unsigned(3 downto 0);

55    signal bcd_reg_1000     : unsigned( 3 downto 0);
56    signal bcd_reg_100      : unsigned( 3 downto 0);
57    signal bcd_reg_10       : unsigned( 3 downto 0);
58    signal bcd_reg_1        : unsigned( 3 downto 0);
59    signal seven_seg_in_reg : unsigned(15 downto 0);
60    signal counter2         : unsigned(22 downto 0);
61     alias select_segment   : unsigned( 1 downto 0) is counter2(15 downto 14);
62     alias delay            : unsigned(13 downto 0) is counter2(13 downto
    0);
63    constant delay0         : unsigned(15 downto 0) := (others => '0');
64    constant counter2_zero  : unsigned(22 downto 0) := (others => '0');


67    begin

69    seven_seg_in_U <= unsigned(SEVEN_SEG_IN);

71    process(seven_seg_sig)
72    begin
73      case seven_seg_sig is
74        when x"0"   =>  SEVEN_SEG_out <= "1000000";
75        when x"1"   =>  SEVEN_SEG_out <= "1111001";
76        when x"2"   =>  SEVEN_SEG_out <= "0100100";
77        when x"3"   =>  SEVEN_SEG_out <= "0110000";
78        when x"4"   =>  SEVEN_SEG_out <= "0011001";
79        when x"5"   =>  SEVEN_SEG_out <= "0010010";
80        when x"6"   =>  SEVEN_SEG_out <= "0000010";
81        when x"7"   =>  SEVEN_SEG_out <= "1111000";
82        when x"8"   =>  SEVEN_SEG_out <= "0000000";
83        when x"9"   =>  SEVEN_SEG_out <= "0010000";
84        when x"A"   =>  SEVEN_SEG_out <= "0001000";
85        when x"B"   =>  SEVEN_SEG_out <= "0000011";
86        when x"C"   =>  SEVEN_SEG_out <= "1000110";
```

```
87        when x"D"    =>  SEVEN_SEG_out <= "0100001";
88        when x"E"    =>  SEVEN_SEG_out <= "0000110";
89        when x"F"    =>  SEVEN_SEG_out <= "0001110";
90        when others  =>  SEVEN_SEG_out <= "1111111";
91      end case;
92    end process;

94    seg_select : process(SYSCLK)
95    begin
96      if SYSCLK = '1' and SYSCLK'event then
97        if CLK = '1' then
98          counter2 <= counter2 + 1 after 9 ns;
99          if delay = delay0 then
100           case select_segment is
101             when "00" =>
102               seven_seg_sig <= SEVEN_SEG_1000 after 9 ns;
103               SEVEN_SEG_SELECT <= "0111" after 9 ns;
104               DECIMAL_POINT <= DECIMAL_POINT_IN(3) after 9 ns;
105             when "01" =>
106               seven_seg_sig <= SEVEN_SEG_100 after 9 ns;
107               SEVEN_SEG_SELECT <= "1011" after 9 ns;
108               DECIMAL_POINT <= DECIMAL_POINT_IN(2) after 9 ns;
109             when "10" =>
110               seven_seg_sig <= SEVEN_SEG_10 after 9 ns;
111               SEVEN_SEG_SELECT <= "1101" after 9 ns;
112               DECIMAL_POINT <= DECIMAL_POINT_IN(1) after 9 ns;
113             when "11" =>
114               seven_seg_sig <= SEVEN_SEG_1 after 9 ns;
115               SEVEN_SEG_SELECT <= "1110" after 9 ns;
116               DECIMAL_POINT <= DECIMAL_POINT_IN(0) after 9 ns;
117             when others =>
118               seven_seg_sig <= SEVEN_SEG_1 after 9 ns;
119               SEVEN_SEG_SELECT <= "1110" after 9 ns;
120               DECIMAL_POINT <= DECIMAL_POINT_IN(0) after 9 ns;
121         end case;
122       end if;  --delay
123     end if;    -- clk
124   end if;      --reset
125   end process seg_select;

127   unsigned_2_bcd : process(SYSCLK, RESET)
128   begin
129     if RESET = RESET_VAL then
130       SEVEN_SEG_1000 <= x"9" after 9 ns;
131       SEVEN_SEG_100  <= x"9" after 9 ns;
132       SEVEN_SEG_10   <= x"9" after 9 ns;
133       SEVEN_SEG_1    <= x"9" after 9 ns;
134       binary         <= (others => '0') after 9 ns;
135       count_to_10       <= 0               after 9 ns;
136     elsif SYSCLK = '1' and SYSCLK'event then
```

```
137       if CLK = '1' then
138         if BCD_ON = '1' then
139           if NEW_THD = '1' then
140             seven_seg_in_reg <= seven_seg_in_U;
141             bcd_reg_1    <= (others =>'0');
142             bcd_reg_10   <= (others =>'0');
143             bcd_reg_100  <= (others =>'0');
144             bcd_reg_1000 <= (others =>'0');
145           else    -- BCD conversion
146             if seven_seg_in_reg > x"0CCC" then    -- 10 %
147               seven_seg_in_reg <= seven_seg_in_reg - x"0CCC";
148               bcd_reg_1000 <= bcd_reg_1000 + 1;
149             elsif seven_seg_in_reg > x"0147" then -- 1 %
150               seven_seg_in_reg <= seven_seg_in_reg - x"0147";
151               bcd_reg_100 <= bcd_reg_100 + 1;
152             elsif seven_seg_in_reg > x"0020" then -- 0.1 %
153               seven_seg_in_reg <= seven_seg_in_reg - x"0020";
154               bcd_reg_10 <= bcd_reg_10 + 1;
155             elsif seven_seg_in_reg > x"0003" then -- 0.01 %
156               seven_seg_in_reg <= seven_seg_in_reg - x"0003";
157               bcd_reg_1 <= bcd_reg_1 + 1;
158             else
159               if counter2 = counter2_zero then
160                 SEVEN_SEG_1000 <= bcd_reg_1000 after 9 ns;
161                 SEVEN_SEG_100  <= bcd_reg_100  after 9 ns;
162                 SEVEN_SEG_10   <= bcd_reg_10   after 9 ns;
163                 SEVEN_SEG_1    <= bcd_reg_1    after 9 ns;
164               end if;
165             end if;
166           end if;
167         else
168           if counter2 = counter2_zero then
169             SEVEN_SEG_1000 <= seven_seg_in_U(15 downto 12) after 9 ns;
170             SEVEN_SEG_100  <= seven_seg_in_U(11 downto  8) after 9 ns;
171             SEVEN_SEG_10   <= seven_seg_in_U( 7 downto  4) after 9 ns;
172             SEVEN_SEG_1    <= seven_seg_in_U( 3 downto  0) after 9 ns;
173           end if;
174         end if;      --BCD_ON
175       end if;      --CLK
176     end if;      --RESET
177   end process unsigned_2_bcd;


180   end behavior;
```

B.1.11 operators.vhd

```
1  --#######################################################
2  --##  Project name : adc_sample_calc_thd
3  --##  File name    : operators.vhd
4  --##  Author       : Jan-Heiner Dreschhoff
5
6
7
8  --<makepackage>
9  --  <component = operators>
10 --    <name = dreschhoff>
11 --    <project = thd>
12 --    <prototypes>
13 --      ## overloaded operators
14 --       function "and"(L: std_logic; R: unsigned) return unsigned;
15 --       function "and"(L: unsigned; R: std_logic) return unsigned;
16 --       function "and"(L: std_logic; R: integer) return integer;
17 --       function "and"(L: integer; R: std_logic) return integer;
18 --       function "and"(L: std_logic; R: signed) return signed;
19 --       function "and"(L: signed; R: std_logic) return signed;
20 --       function "sra"(L: signed; R: natural) return signed;
21 --    </prototypes>
22 --  </component>
23 --</makepackage>
24
25 library ieee;
26   use ieee.std_logic_1164.all;
27   use ieee.numeric_std.all;
28
29 library work;
30    use work.global_pkg.all;
31
32 package body operators_pkg is
33
34 --####
35 --overloaded operators
36 --####
37
38 -- FUNCTION AND--------------------
39 --  inputs: unsigned and std_logic
40 --  returns: unsigned
41 --    the std_logic enables the unsigned output.
42 --    std_logic == '1', output is unsigned, else naught.
43   function "and"(L: std_logic; R: unsigned) return unsigned is
44   variable temp : unsigned ((R'length-1) downto 0);
45   begin
46     temp := (others => '0');
47     if L = '0' then
48       return temp;
49     else
50       return R;
```

```
51      end if;
52    end "and";
53  -- like the function above, but with exchanged operants.
54    function "and"(L: unsigned; R: std_logic) return unsigned is
55    begin
56      return R and L;
57    end "and";

59    function "and"(L: std_logic; R: signed) return signed is
60    variable temp : signed ((R'length-1) downto 0);
61    begin
62      temp := (others => '0');
63      if L = '0' then
64        return temp;
65      else
66        return R;
67      end if;
68    end "and";
69  -- like the function above, but with exchanged operants.
70    function "and"(L: signed; R: std_logic) return signed is
71    begin
72      return R and L;
73    end "and";

75    function "and"(L: std_logic; R: integer) return integer is
76    begin
77      if L = '1' then
78        return R;
79      else
80        return 0;
81      end if;
82    end "and";

84    function "and"(L: integer; R: std_logic ) return integer is
85    begin
86      return R and L;
87    end "and";

89    function "sra"(L: signed; R: natural) return signed is
90      constant L_length: integer := L'length-1;
91      variable result : signed(L_length downto 0);
92      variable xR : natural := R;
93    begin
94      if (L'length <= 1) or (R=0) then
95        return L;
96      else
97        if xR > L_length then
98          xR := L_length;
99        end if;
100       result((L_length-xR) downto 0) := L(L_length downto xR);
```

```
101       result(L_length downto (L_length-xR+1)) := (others => L(L_length));
102       return result;
103     end if;
104   end "sra";
105 end package body operators_pkg;
```

B.1.12 functions.vhd

```
1  --#######################################################
2  --##  Project name : adc_sample_calc_thd
3  --##  File name    : functions.vhd
4  --##  Author       : Jan-Heiner Dreschhoff
5
6  --<makepackage>
7  --  <global>
8  --    constant Q15_MSB : integer := 15;
9  --  </global>
10 --  <component = functions>
11 --    <name = dreschhoff>
12 --    <project = thd>
13 --    <prototypes>
14 --      ## functions
15 --       function mul(L: unsigned ;R : unsigned) return unsigned;
16 --       function mul(L: signed ;R : unsigned) return signed;
17 --       function mul(L: unsigned ;R : signed) return signed;
18 --       function S33toU32(VAL : S33) return U32;
19 --        ## yet to be written
20 --       ##function mul(L: signed ;R : signed) return signed;
21 --      ## overloaded functions
22 --       function to_signed(L: real; R:natural) return signed;
23 --       function to_signed(VAL: unsigned) return signed;
24 --       function to_unsigned(VAL: signed) return unsigned;
25 --       function to_integer(val: std_logic) return integer;
26 --       function to_integer(val: std_logic_vector) return integer;
27 --       function to_integer(val: unsigned; lower: unsigned; upper: unsigned) return
28 --    </prototypes>
29 --  </component>
30 --</makepackage>
31
32 library ieee;
33   use ieee.std_logic_1164.all;
34   use ieee.numeric_std.all;
35
36 library work;
37    use work.global_pkg.all;
38    use work.operators_pkg.all;
39
40 package body functions_pkg is
```

```
42 -- FUNCTION MUL
43 --   multiplies using shift operations and addition.
44 --   the "and" in the loop is overloaded to use a std_logic
45 --   bit as an enable. compare: value*1, value*0
46   function mul(L: unsigned ;R : unsigned) return unsigned is
47     variable result : unsigned((L'length+R'length-1) downto 0);
48     variable temp : unsigned((L'length+R'length-1) downto 0);
49     variable i : integer range (R'length-1) downto 0;
50   begin
51     result := (others =>'0');
52     temp((L'length+R'length-1) downto L'length) := (others =>'0');
53     temp((L'length-1) downto 0) := L; -- L can not be shifted by R'length
54     for i in 0 to (R'length-1) loop
55       result := result + (R(i) and (temp sll i));
56     end loop;
57     return result;
58   end mul;

60   function mul(L: signed ;R : unsigned) return signed is
61     variable result : signed((L'length+R'length-1) downto 0);
62     variable temp : signed((L'length+R'length-1) downto 0);
63     variable i : integer range (R'length-1) downto 0;
64     variable minus : std_logic;
65   begin
66     result := (others =>'0');
67     minus := L(L'length-1);
68     temp((L'length+R'length-1) downto L'length) := (others => minus);
69     temp((L'length-1) downto 0) := L; -- L can not be shifted by R'length
70     if minus = '1' then
71       temp := -temp;
72     end if;
73     for i in 0 to (R'length-1) loop
74       result := result + (R(i) and (temp sll i));
75     end loop;
76     if minus = '1' then
77       return -result;
78     else
79       return result;
80     end if;
81   end mul;

83   function mul(L: unsigned ;R : signed) return signed is
84   begin
85     return mul(R,L);
86   end mul;

88   function S33toU32(VAL : S33) return U32 is
89     variable VAR : std_logic_vector(32 downto 0);
90   begin
```

```
91      if VAL(32) = '1' then
92        VAR := std_logic_vector(-VAL);
93      else
94        VAR := std_logic_vector(VAL);
95      end if;
96      return unsigned(VAR(31 downto 0));
97    end S33toU32;



101     --overloaded functions
102   function to_signed(L: real; R: natural) return signed is
103   begin
104     return to_signed(integer(L),R);
105   end to_signed;

107   function to_signed(VAL: unsigned) return signed is
108     variable result : signed (VAL'length downto 0);
109   begin
110     result := '0' & signed(std_logic_vector(VAL));
111     return result;
112   end to_signed;

114   function to_unsigned(VAL: signed) return unsigned is
115     constant VAL_L  : natural := VAL'length-1;
116     variable XVAL   : signed(VAL_L downto 0) := VAL;
117     variable result : unsigned((VAL_L-1) downto 0);
118   begin
119     if XVAL(VAL_L) = '1' then
120         XVAL := -XVAL;
121         result := unsigned(std_logic_vector(XVAL(VAL_L-1 downto 0)));
122     else
123         result := unsigned(std_logic_vector(XVAL(VAL_L-1 downto 0)));
124     end if;
125     return result;
126   end to_unsigned;

128   function to_integer(val: std_logic) return integer is
129   begin
130     if val = '1' then
131         return 1;
132     else
133         return 0;
134     end if;
135   end to_integer;

137   function to_integer(val: std_logic_vector) return integer is
138     constant VAL_L : natural := (val'length-1);
139     constant value : unsigned := unsigned(val);
140     variable result : integer range 0 to (2**VAL_L+1);
```

```
141   begin
142     result := 0;
143     for i in 0 to VAL_L loop
144       result := result + (value(i) and 2**i);
145     end loop;
146     return result;
147   end to_integer;

149   function to_integer(val: unsigned; lower: unsigned; upper: unsigned) return integer
150     variable result : integer range to_integer(lower) to to_integer(upper);
151     constant VAL_L : natural := (val'length-1);
152   begin
153     if (val >= (lower)) and (val <= (upper-1)) then
154       for i in 0 to VAL_L loop
155         result := result + (val(i) and 2**i);
156       end loop;
157     elsif val< lower then
158       result := to_integer(lower);
159     else
160       result := to_integer(upper);
161     end if;
162     return result;

164   end to_integer;

166 end package body functions_pkg;
```

B.2 pkg_maker.py

```
1  ##########################################################
2  ##    PACKAGE MAKER for *.vhdl files.
3  ##    it interpretes code in the comments of *.vhdl files
4  ##    and creates packages accordingly.
5  ##    FORMAT:
6  ##  <makepackage>
7  ##    <global>
8  ##        constant ADC_Width : integer := 12;
9  ##    </global>
10 ##    <component = adc_control>
11 ##      ##comments
12 ##    <name = dreschhoff>
13 ##    <project = thd detection>
14 ##      <prototypes>
15 ##         function "*"(L: unsigned; R: unsigned) return unsigned;
16 ##    </prototypes>
17 ##    </component>
18 ##  </makepackage>
```

```
19 ##
20 ##    author: Jan-Heiner Dreschhoff
21 ##    date  : 20.1.2010

23 import os, time

25 vhd_files = []
26 entityIO = []
27 prototypes = []
28 globals = []
29 gConstants = []
30 gTypes = []
31 name = ""
32 project = ""
33 num_pkg = 0


36 currpath = os.getcwd()    # get current working directory
37 all_files = os.listdir(currpath)
38 for fname in all_files:
39   if ".vhd" in fname[-5:] :
40     vhd_files.append(fname)

42 for ifile in vhd_files:
43   file = open(ifile, "r")
44   for line in file:
45     if "<makepackage>" in line:
46         while "</makepackage>" not in line:
47             line = file.next()
48             if "<global>" in line:
49                  line = file.next()
50                  while "</global>" not in line:
51                      line = line.lstrip('- \t')
52                      line = line.rstrip('- \n')
53                      line = line + "\t--" + file.name + "\n"
54                      if "type" in line:
55                        gTypes.append(line)
56                        while ";" not in line:
57                          line = file.next()
58                          line = line.lstrip('- \t')
59                          line = line.rstrip('- \n')
60                          line = "     " + line +  "\n"
61                          gTypes.append(line)
62                      elif "constant" in line:
63                        gConstants.append(line)
64                      elif line:
65                        globals.append(line)
66                      line = file.next()
67                  globals.append("\n")
68             if "<component" in line:
```

```
                line = line.lstrip('- \t')
                entity = line[10:]
                entity = entity.lstrip(' =')
                entity = entity.rstrip('> \n')
                line = file.next()
                while "</component>" not in line:
                    if "<prototypes" in line:
                        line = file.next()
                        while "</prototypes" not in line:
                            line = line.lstrip('- \t')
                            line = line.rstrip('\n')
                            if "##" in line[:2]:
                                line = "--"+line[2:]
                            prototypes.append(line)
                            line = file.next()
                    elif "<name" in line:
                        line = line.lstrip('- \t')
                        name = line[5:]
                        name = name.lstrip(' =')
                        name = name.rstrip('> \n')
                    elif "<project" in line:
                        line = line.lstrip('- \t')
                        project = line[8:]
                        project = project.lstrip(' =')
                        project = project.rstrip('> \n')
                    line = file.next()
        else:
            break
file.seek(0)
for line in file:
  if "entity" in line and entity:
    for line in file:
      if "end" in line:
        break
      entityIO.append(line)
    if entityIO:
      break
if entityIO or prototypes:
  num_pkg += 1
  dot = file.name.rfind(".")
  wfile = open(file.name[0:dot] + "_pkg" + \
                file.name[dot:], "w")
  wfile.write("----------------------------------------\n")
  wfile.write("----- Componenet declaration-------------\n")
  if project:
     wfile.write("-- Project     : " + project + "\n")
  if name:
     wfile.write("-- Author      : " + name + "\n")
  wfile.write("-- package for : " + file.name + "\n")
  wfile.write("----------------------------------------\n")
```

```
119     wfile.write("----------------gererated by pkg_maker.py\n\n")
120     wfile.write("library ieee;\n")
121     wfile.write("   use ieee.std_logic_1164.all;\n")
122     wfile.write("   use ieee.numeric_std.all;\n\n")
123     wfile.write("library work;\n")
124     wfile.write("   use work.global_pkg.all;\n\n")
125     wfile.write("package " + entity + "_pkg is\n")
126     if entityIO:
127         wfile.write("  component " + entity + " is\n")
128         for component in entityIO:
129           wfile.write("  " + component)
130         wfile.write("  end component;\n")
131     if prototypes:
132         wfile.write("\n")
133         for line in prototypes:
134             wfile.write("  " + line + "\n")
135         wfile.write("\n")
136     wfile.write("end " + entity + "_pkg;")
137     wfile.close()
138     del entityIO
139     entityIO = []
140     del prototypes
141     prototypes = []
142     entity = 0
143   file.close()

145 if globals or gTypes or gConstants:
146   num_pkg += 1
147   dot = vhd_files[0].rfind(".")
148   gfile = open("global_pkg" + vhd_files[0][dot:], "w")
149   gfile.write("-----------------------------------------\n")
150   gfile.write("---------GLOBAL PACKAGE-------------------\n")
151   gfile.write("-----------------------------------------\n")
152   gfile.write("----------------gererated by pkg_maker.py\n\n")
153   gfile.write("library ieee;\n")
154   gfile.write("   use ieee.std_logic_1164.all;\n")
155   gfile.write("   use ieee.numeric_std.all;\n\n")
156   gfile.write("package global_pkg is\n\n")
157   if gConstants:
158     gfile.write("\n--- CONSTANTS ---\n")
159     for line in gConstants:
160       gfile.write("  " + line)
161   if gTypes:
162     gfile.write("\n--- TYPES ---\n")
163     for line in gTypes:
164       gfile.write("  " + line)
165   if globals:
166     gfile.write("\n--- MISC ---\n")
167     for line in globals:
168       gfile.write("  " + line)
```

```
169   gfile.write("end global_pkg;")
170   gfile.close()

172 now = time.localtime()
173 print("\n" + str(now.tm_mday)+ "."+str(now.tm_mon)+"."+str(now.tm_year) \
174         + " " + str(now.tm_hour)+":"+str(now.tm_min)+":"+str(now.tm_sec)\
175         + "  ->  "+ str(num_pkg) + " Pakete geschrieben \n")
```

Abkürzungsverzeichnis

ABS Anti Blockier System

AMR Effekt anisotroper magnetoresistiver Effekt

ASR Antriebsschlupfregelung

BPSK Binary Phase shift keying

ESC Electronic Stability Control (Fahrdynamikregelung)

ESZ-ABS Experimentelle digitale Signalverarbeitung und Zustandserkennung für ABS-Sensoren

NRZ Non-Return-to-Zero

POR Power On Reset

SOA Safe Operating Area

Tabellenverzeichnis

Bildverzeichnis

Versicherung über Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §25(4) ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 28.07.2010
Ort, Datum

Unterschrift