Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Ausarbeitung AW 1 - WS10 11

Erik Andresen

Modellierung und Codegenerierung von SOC-Beschleunigermodulen am Beispiel eines Kalman-Filters

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Inhaltsverzeichnis

1. Einleitung

1.1. Eingebettete Systeme

Eingesetzt zur Kommunikation in jedem Telefon, in Fortbewegungsmitteln wie Autos und Flugzeugen, zu Hause in Fernseher, Waschmaschine und Mikrowelle und in lebenserhaltenden Systemen in der Medizintechnik sind Eingebettete Systeme aus unserer Umgebung nicht wegzudenken. Mit einem jährlichem Wachstum von 9% gewinnt diese Branche immer weiter an Bedeutung wobei bisher vor allem ASICs, die nur für einen bestimmten Zweck gefertigt werden, zum Einsatz kamen [4] [14]. ASICs müssen in der Entwicklungsphase mehrmals bei Fehlern oder geänderten Anforderungen erneut produziert werden, FPGAs mit Preisen zwischen 5$ und 20000$ sind rekonfigurierbar: die Schaltung kann geändert werden ohne den Hardware-Baustein zu tauschen weshalb FPGAs zunehmend bei Produkten mit geringen Stückzahlen, Systemen die rekonfigurierbar sein sollen, sowie im Forschungsbereich Verwendung finden [3]. In den Eingebetteten Systemen verwendeten System-On-Chips werden vermehrt Multiprozessor-Systeme eingesetzt, da diese durch die geringere Taktrate bei zusätzlicher Rechenleistung weniger Strom verbrauchen [15].

1.2. High Performance Embedded Computing

Im Rahmen des FAUST-Projektes des Department Informatik der HAW-Hamburg werden zur Fahrbahnerkennung CCD-Kameras eingesetzt [7]. Zum Einsatz kam zunächst die Fahrspurerkennung durch Polynomapproximation auf einem Embedded-PC mit VIA-Chipsatz. Mit dem Ziel einer Energie- und Platzsparenden Umsetzung wurde der PC später durch einen Xilinx FPGA als System-On-Chip mit Microblaze Processor Soft-IP-Core [12] ersetzt (Vergleiche mit Abbildung 1).

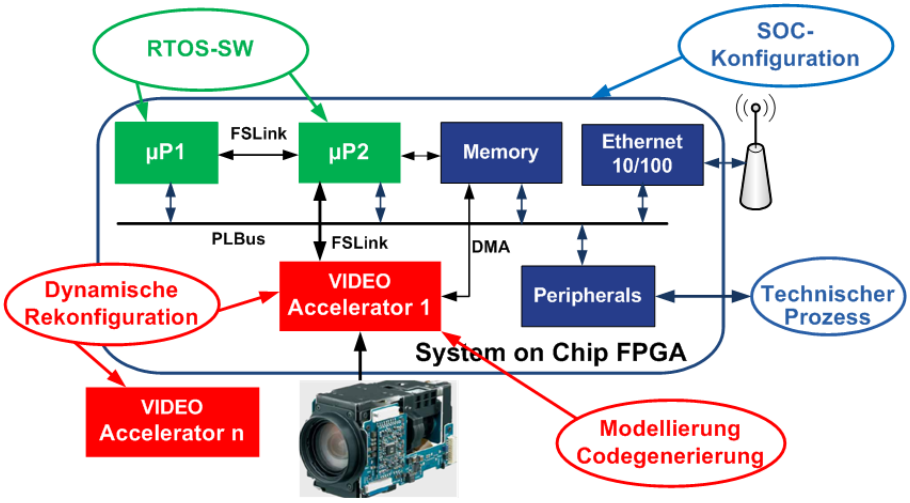

Abbildung 1: System-On-Chip FPGA: Dual-Core CPU mit austauschbaren Hardware-Beschleuniger-Module für Bildverarbeitung und zusätzlicher Peripherie.

Ein Soft-IP-Core ist ein wiederverwendbarer Schaltplan der entweder als Quellcode oder als bereits für die Hardware synthetisierte Netzliste existiert. Der Microblaze als Soft-IP-Core existiert also ausschließlich in den Logikblöcken des FPGA und nicht als physische Hardware. Die Polynomapproximation wurde zusammen mit der Bildverarbeitung in Hardware-Beschleuniger-Module ausgelagert. Die Beschleunigung unter dem Titel „High Performance Embedded Computing“ (HPEC) erfolgt durch Parallelverarbeitung. Zur „robusteren Erkennung“ der Fahrspur wurden gleichzeitig die Gleichungen für einen Kalman-Filter aufgestellt [12]. Bei dem Kalman-Filter handelt es sich um einen rekursiven

Schätzfilter der einen „optimalen Schätzwert für den Zustand x eines Systems“ liefert [10] und in Kapitel 3 behandelt wird. Durch die rekursive Eigenschaft wird nur der aktuelle Zustand gespeichert, was den Speicherbedarf sowie den Rechenaufwand gegenüber anderen Filtern reduziert. Damit eignet sich das Kalman-Filter vor allem zum Einsatz in Echtzeitsystemen [12].

Ein FPGA als System-On-Chip mit zwei Prozessoren ist in Abbildung 1 zu sehen. Durch die Verwendung der Video-Hardware-Beschleuniger-Module werden die Prozessoren entlastet. Da die einzelnen Stufen der Fahrspurerkennung nacheinander durchgeführt werden, können bei der dynamischen partiellen Rekonfiguration die einzelnen Module im laufenden Betrieb getauscht werden. Modellgetriebene Entwicklungswerkzeuge wie das in Kapitel 2 vorgestellte AccelDSP vereinfachen die Entwicklung der Hardware-Beschleuniger Module.

1.3. Ziele des HPEC-Projektes und Ausblick auf weiterführende Arbeiten

Das aktuelle FAUST-System benutzt einen Microblaze Prozessor Soft-IP-Core. Da in der Industrie vermehrt Multiprozessor-Systeme zum Einsatz kommen wird im HPEC-Projekt an Konfigurationen mit mehreren Soft-IP-Core Prozessoren geforscht [1]. Der Schwerpunkt liegt auf Symmetrischen Multiprozessorsystem (SMP): Untereinander gleiche Prozessoren teilen sich einen gemeinsamen Daten- und Speicherbus [6], die Tasks werden dynamisch auf die Prozessoren verteilt.

Um die eigenen Produkte für Software-Entwickler attraktiver zu machen hat Xilinx für 2011 FPGAs mit physischen Dual-Core ARM Cortex-A9 Prozessoren angekündigt [13]. Die Vorteile sind:

- Gegenüber Prozessor Soft-IP-Cores werden keine FPGA-Ressourcen benötigt.
- Bei Prozessoren in Eingebetteten Systemen ist ARM der Marktführer und wird deswegen von vielen Software-Plattformen unterstützt [13].
- Der ARM-Prozessor ist von der restlichen FPGA-Hardware entkoppelt, weshalb Hardware- und Software Entwickler unabhängig voneinander parallel arbeiten können.

Xilinx hat den mit 800 MHz getakteten, am wenigsten Strom verbrauchenden, ARM Cortex-A9 genommen da in einem stromsparendem System komplexe Operationen durch Hardware-Beschleuniger-Module erledigt werden sollen [13]. Da Xilinx langfristig auf diese Architektur setzt, wird die HPEC-Arbeitsgruppe ihre Roadmap (Vergleiche mit Abbildung 2) auf diesen umsetzen. Es gibt drei Forschungspfade:

1. Die Aufteilung welche Algorithmen in Hardware, welche in Software durchgeführt werden [8].
2. Automatische Codegenerierung von Hardware-Beschleuniger-Modulen mit Modell Driven Development (vgl. Kap. 2).
3. Parallelisierung der Software. Dazu gehören [9]
   - Aufteilen der Software in mehrere, gleich gewichtete Threads.
   - Synchronisierung der einzelnen Threads unter Vermeidung von Race-Conditions und Deadlocks.
   - Zugriff auf Peripherie: Hardware, wie RS232-Schnittstellen können nur von einem Thread zur Zeit genutzt werden.
   - Auswahl der Befehls- und Datenströme (Flynnsche Klassifikation) [5].
   - Cache-Kohärenz und Speicher-Konsistenz, Daten sollen keine Inkonsistenzen aufweisen.

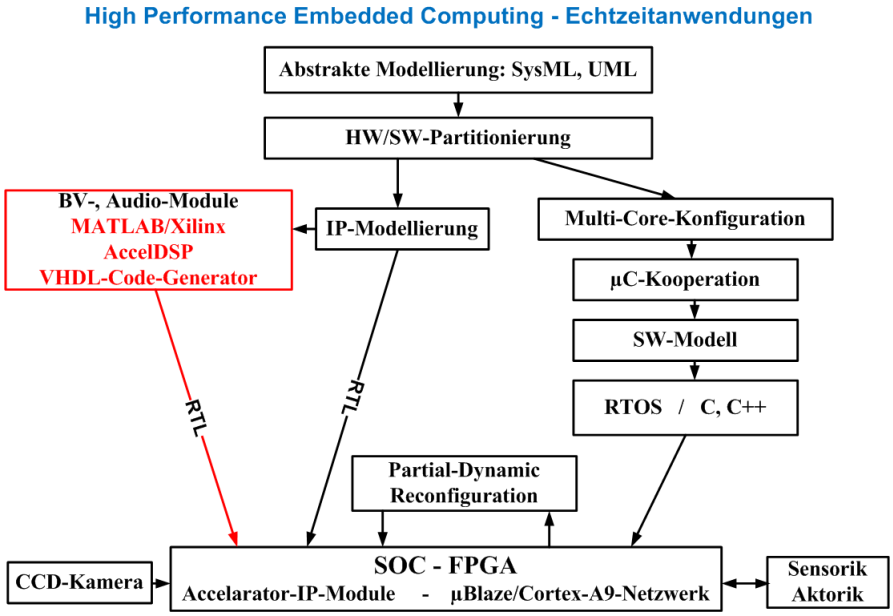

Abbildung 2: Roadmap HPEC: Der in Kapitel 2 und 3 betrachte Pfad ist rot hervorgehoben.

- Beim Debuggen von Programmen müssen u.u. die laufenden Befehle auf allen Prozessoren gleichzeitig betrachtet werden.

Wobei Punkt 3 für Studenten der Informatik die meiste Relevanz besitzt. Ziel für das Fach Anwendung 1 war die synthesegerechte Modellierung von DSP-Blöcken mit Codegenerierung, als Beispiel der Kalman-Filter (Vergleiche mit Kapitel 2 und 3). In Anwendung 2 wird dies in der Praxis erprobt, dazu können die aufgestellten Gleichungen des Kalman-Filters aus [12] in AccelDSP synthetisiert im Fahrzeug eingesetzt werden. Zusätzlich bietet es sich an nach weiteren Algorithmen für die Fahrspurerkennung zu suchen und zu testen.

Im Fach Projekt 1 wird die Forschung an Multiprozessor-Systemen durchgeführt. Solange die FPGAs mit integrierten ARM-Prozessoren noch nicht erhältlich sind werden die Untersuchungen an Microblaze Multiprozessor-Systemen durchgeführt. Es gibt erste Forschungsergebnisse die das für den Microblaze (Version 4.0a und 5.0a, aktuell ist 8.0) vorhandene Echtzeitbetriebssystem Xilkernel auf mehrere Prozessoren anpasst [6]. Xilkernel bietet für das verteile Rechnen u.a. POSIX-Threads und Synchronisations-Vorrichtungen wie Semaphoren und Mutexe.

Für die Einarbeitung in die Cortex-A9 ARM-Prozessoren und Entwicklungswerkzeuge bietet sich z.B. die „Mobile Software Entwicklungs Plattform“ PandaBoard [11] an, die über umfangreiche Dokumentation verfügt. Auf Basis eines Texas Instruments OMAP430 (Open Multimedia Application Platform) System-On-Chips verfügt das Board über Peripherie wie USB, Ethernet, WLAN, HDMI, DVI, Audio ein- und Ausgänge sowie einen OpenGL fähigen Grafikchip.

Folgende Risiken sehe ich:

- Nur begrenzt zur Verfügung stehende Zeit um mehrere Algorithmen zu verstehen und auszuprobieren.
- SMP ist an der HAW ein neues Feld. Es ist deswegen im Haus kaum Know How zum darauf Aufbauen vorhanden.
- Mathematische Grundlagen zu den Filtern wurden im Studium nicht behandelt und sind deswegen nicht vorhanden.

2. Synthesegerechte Modellierung von DSP-Algorithmen in AccelDSP

Viele Algorithmen werden erst in der Form von mathematischen Gleichungen definiert. Es müssen dann mathematische Operationen auf Vektoren und Matrizen durchgeführt werden. Üblicherweise implementiert man diese Algorithmen nicht sofort im Zielsystem sondern verifiziert erst einmal die Gleichungen mit Programmen wie Matlab von „The MathWorks“, die Operationen auf mathematische Datenstrukturen wie Vektoren und Matrizen ohne zusätzlich notwendige Erweiterungen durchführen können.

Die AccelDSP Synthese-Werkzeuge haben das Ziel Matlab-Code in eine Hardware-Beschreibungssprache wie VHDL auf Register Transfer Level (RTL) umzusetzen. Durch Automatisierung dieser Prozesse soll die Entwicklungszeit verkürzt werden.

Der Matlab-Code muss dafür einem Synthese fähigem Stil [17] entsprechen. Um den bei Fließkomma-Arithmetik entstehenden Mehraufwand [16] und die festen Wortbreiten gegenüber Festkomma-Arithmetik zu vermeiden wird eine Quantisierung in Festkommazahlen durchgeführt. Das Ziel ist eine minimale Wortbreite ohne ungewollte arithmetische Über- oder Unterläufe.

Die Entwicklung erfolgt in vier Schritten:

1. Entwicklung des Algorithmus in Matlab.
2. Simulation und Verifizierung des fertigen Algorithmus in AccelDSP mit Fließkommazahlen. Hierzu ist eine Testbench notwendig. Das Ergebnis dient als Vergleich für spätere Simulationen.
3. Konvertierung des Algorithmus in Festkommazahlen.
4. Simulation und Verifizierung der Festkomma-Variante, Vergleich mit der Simulation aus 2. Gegebenenfalls anpassen der Wortbreiten und wiederholen der Simulation bis das Ergebnis alle gegebenen Anforderungen erfüllt.

Anschließend können zusätzliche Einstellungen wie Pipeline-Stufen oder RAM/ROM-Aufteilungen gesetzt und die RTL-Dateien des Algorithmus erzeugt und in das FPGA-Projekt eingebunden werden.

3. Beispiel Kalman-Filter zur Zustandsschätzung verrauschter Sinus-Signale

Ein Kalman-Filter soll als Einstieg in AccelDSP Modelliert werden. Das Kalman-Filter ist ein rekursiver Schätzalgorithmus das häufig in der Bildverfolgung eingesetzt wird [10], z.B. ein von einer Kamera beobachteter geworfener Ball. Das Kalman-Filter hat das Ziel „einen optimalen Schätzwert für den Zustand x eines Systems“, beruhend auf einer Messung und vorhergehenden Schätzung zu liefern. Dazu müssen der Messfehler sowie die Grundlagen des beobachteten Systems (bei dem Beispiel mit dem Ball u.a. die Gravitationskraft) bekannt sein. Der Messfehler wird genutzt um zum Zeitpunkt $\mathbf{k}$ die Messung $\mathbf{y}_k$ des Zustandes $\mathbf{x}_k$ für die Kalman-Verstärkung $\mathbf{K}_k$ zu gewichten, genaue Messungen sollen stärker berücksichtigt werden. Aufgrund der rekursiven Eigenschaft fließen alle vorhergehenden Schätzungen in den aktuellen Schätzwert $\hat{\mathbf{x}}_k$ mit ein. Es wird der prädizierte „a priori“ Schätzwert $\hat{\mathbf{x}}_k^-$ von dem mit der Messung korrigierten „a posteriori“ Schätzwert $\hat{\mathbf{x}}_k^+$ unterschieden. Es wird zuerst auf die mathematischen Grundlagen eingegangen.

3.1. Zustandsgleichung und Messgleichung

Gegeben ist die Zustandsgleichung [10]

$$\mathbf{x}_{k+1} = \mathbf{A}_k * \mathbf{x}_k + \mathbf{B}_k * \mathbf{u}_k + \mathbf{w}_k \tag{1}$$

Hier

- bildet die Systemmatrix $\mathbf{A}_k$ den Zustand $\mathbf{x}_k$ auf den nachfolgenden Zustand $\mathbf{x}_{k+1}$ und
- die Eingabematrix $\mathbf{B}_k$ die externe Kraft $\mathbf{u}_k$ auf den Zustand $\mathbf{x}_k$ ab.
- Ist $\mathbf{w}_k$ ist das Systemrauschen.

In der Messgleichung

$$\mathbf{y}_k = \mathbf{H}_k * \mathbf{x}_k + \mathbf{v}_k \tag{2}$$

ist $\mathbf{H}_k$ die Messmatrix, die die Messung zum Zeitpunkt $\mathbf{k}$ ohne das Messrauschen $\mathbf{v}_k$ auf den Zustand $\mathbf{x}_k$ abbildet.

Die a priori und a posteriori Schätzfehler sind definiert als

$$\mathbf{e}_k^- = \mathbf{x}_k - \hat{\mathbf{x}}_k^- \tag{3}$$

$$\mathbf{e}_k^+ = \mathbf{x}_k - \hat{\mathbf{x}}_k^+ \tag{4}$$

und die „a priori“ „a posteriori“ Schätzfehlerkovarianzen als

$$\mathbf{P}_k^- = \mathbf{E}(\mathbf{e}_k^- * \mathbf{e}_k^{-\emptyset}) \tag{5}$$

$$\mathbf{P}_k^+ = \mathbf{E}(\mathbf{e}_k^+ * \mathbf{e}_k^{+\emptyset}) \tag{6}$$

Mit $\mathbf{E}$ als Erwartungswert der Schätzfehler.

3.2. Kalman-Verstärkung und Kalman-Algorithmus

Mit der Varianz des Messfehlers $\mathbf{R}_k$ lautet die Kalman-Verstärkung

$$\mathbf{K}_k = \frac{\mathbf{P}_k^- * \mathbf{H}_k^\top}{\mathbf{R}_k + \mathbf{H}_k \mathbf{P}_k^- \mathbf{H}_k^\top} \tag{7}$$

Es ergeben sich aus dem Bruch folgende Eigenschaften:

- Wenn die Varianz des Messfehlers $\mathbf{R}_k$ steigt, wird der Messung weniger „vertraut“, der Schätzung jedoch mehr.
- Wenn die Schätzfehlerkovarianz $\mathbf{P}_k^-$ zunimmt, wird der Messung mehr „vertraut“, der Schätzung jedoch weniger.

Der Algorithmus, in Abbildung 3 dargestellt, wird nach einer Startschätzung für Zustand $\hat{\mathbf{x}}_0^-$ und Schätzfehler $\mathbf{P}_0^-$ in zwei Gruppen aufgeteilt:

1. Berechnen der „a posteriori“ Schätzung $\hat{\mathbf{x}}_k^+$ aus der durch die Messung $\mathbf{y}_k$ korrigierten, prädizierten Schätzung $\hat{\mathbf{x}}_k^+$.
2. Prädiktion des nächsten Zustandes $\hat{\mathbf{x}}_{k+1}^-$.

Die für jeden Zeitpunkt $\mathbf{k}$ in einer Schleife abgearbeitet werden.

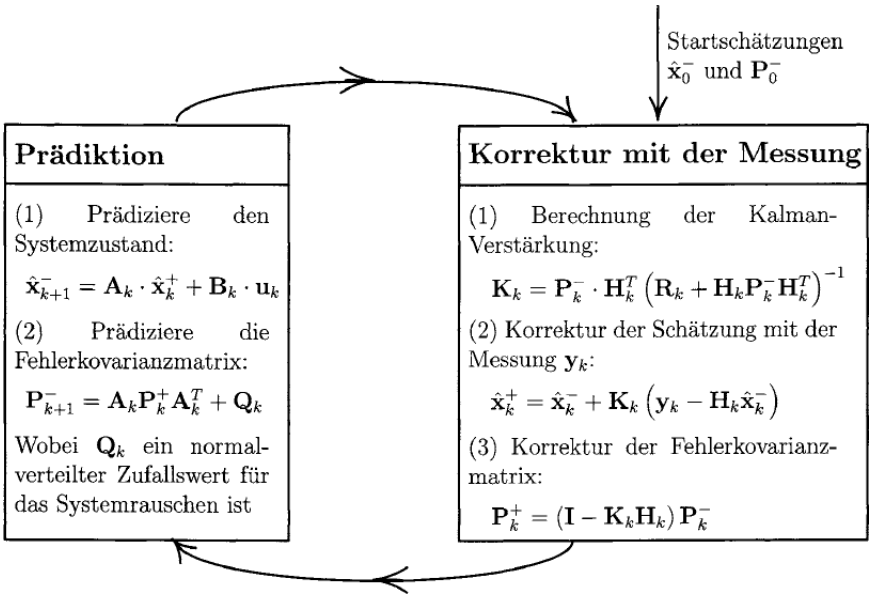

Abbildung 3: Algorithmus des Kalman-Filters. Ausgehend von einer Startschätzung für den Zustand und Schätzfehler werden in einer Schleife nacheinander die aktuelle Schätzung mit der Messung korrigiert und anschließend eine Vorhersage für den folgenden Schätzwert durchgeführt [10].

3.3. Durchführung, Modellierung und Synthese des Kalman-Filters

In diesem Beispiel von Xilinx [2] werden drei Sinus-Schwingungen erzeugt und für die Messung mit Zufallszahlen verrauscht. Das Kalman-Filter soll aus den verrauschten Signalen die Sinus-Schwingungen zurückrechnen. Das Algorithmus wurde von Xilinx wie folgt vereinfacht:

- Systemmatrix $A_k = 1$, da der Zustand zeitlich konstant ist.
- Externe Kraft nicht vorhanden, also $u_k = 0$.
- System ist direkt messbar und damit die Messmatrix $H_k = 1$.

Damit vereinfacht sich die Zustandsgleichung (1) zu

$$\mathbf{x}_{k+1} = \mathbf{x}_k \tag{8}$$

und die Kalman-Verstärkung (7) zu

$$\mathbf{K}_k = \frac{\mathbf{P}_k^-}{\mathbf{R}_k + \mathbf{P}_k^-} \tag{9}$$

In AccelDSP wird zuerst ein neues Projekt erstellt und der fertig geschriebene Kalman-Algorithmus (Anhang A.2) zusammen mit einer Testbench (Anhang A.1) importiert.

Die Vorlage von Xilinx (Anhang A.3) wurde umgeschrieben um den Formeln aus Abbildung 3 zu entsprechen. Dabei entspricht „P_est“ der a posteriori P_k^+ und „P“ der a priori Schätzfehlerkovarianz P_k^-, sowie „x_est“ der a posteriori $\hat{x}_k^+$ und „x“ der a priori Schätzung $\hat{x}_k^-$. Die folgenden Änderungen wurden unternommen:

1. Funktionsargument „S“ umbenannt in „y“ da es sich um die Messung handelt.
2. Bei der persistenten Variable „p“ handelt es sich um die a posteriori Schätzung für den Zustand. Entsprechend wird die variable in „x_est“ umbenannt.

3. Aufgrund der Vereinfachung der Systemmatrix zu $A_k = 1$, bzw Wegfall der externen Kraft u_k unterscheidet Xilinx nicht zwischen den a priori und a posteriori Schätzungen. Die Unterscheidung wird zur Übersichtlichkeit wieder eingefügt. (Zeilen 23 und angepasste Korrektur der Schätzung in Zeile 17, Anhang A.2).

4. Für die Prädiktion der Fehlerkovarianz wird die hinzugefügte Variable für das Systemrauschen „Q“ anstatt der Einheitsmatrix „I“ verwendet.

5. Die Prädiktion wird nach der Korrektur ausgeführt. Entsprechend muss der Initiale Wert für den Schätzfehler um 1 erhöht werden (Zeile 5 und 25 in A.2).

6. Anstatt „inv()“ wird die Synthesefähige Funktion „accel_qr_inverse()“ zur Berechnung der Inversen Matrix genutzt.

Das Ergebnis des ersten Schritt „Verify Floating Point“ zeigt Abbildung 4. Aus dem verrauschten „Input Signal with Random Noise“ wurde das Eingangssignal „Input Signal without Noise“ im Ergebnis „Filtered Output Signal“ annähernd wiederhergestellt.

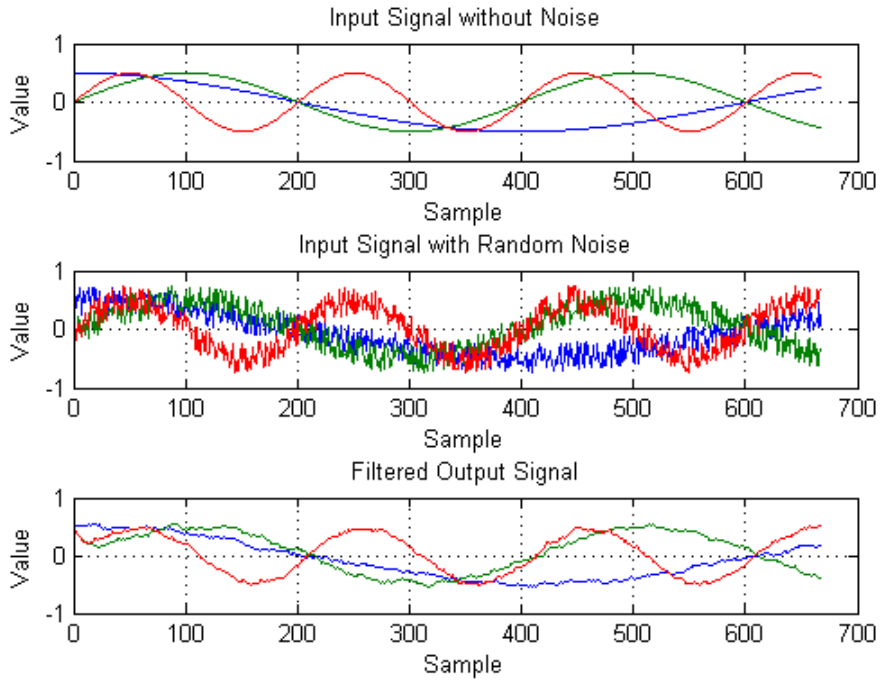

Abbildung 4: Ausgabe der Testbench unter Nutzung von Fließkommazahlen: Die drei Sinusschwingungen mit den Frequenzen 1/400 (grün, Phasenverschoben um 90°), 1/200 (rot) und 1/1800 (blau) und Amplitude 0.5 aus der obersten Grafik werden in der mittleren durch Zufallszahlen zwischen -0.25 und 0.25 verrauscht. In der untersten Grafik ist das Ergebnis des Kalman-Filters mit einer Startschätzung für den Zustand von $\hat{x}_0^- = 0.5$ für alle drei Signale zu sehen.

Im zweiten Schritt wird „Analyze“ und „Generate Fixed Point“ ausgeführt in denen AccelDSP den Algorithmus sowie den Kodierstil analysiert und die Wortbreiten für die Festkommazahlen berechnet. Das Ergebnis zeigt Abbildung 5.

Das „Filtered Output Signal“ in der Fixkomma-Variante lässt sich noch verbessern: Die Sinus-Schwingungen sind hier noch nicht erkennbar. Die automatisch gewählten Wortbreiten zeigen die ersten beiden Spalten der Tabelle 1. Die 53-Bit, bzw 54-Bit für signed, Wortbreiten stellen die maximale Breite dar. Dies ist unter den AccelDSP-Einstellungen definierbar.

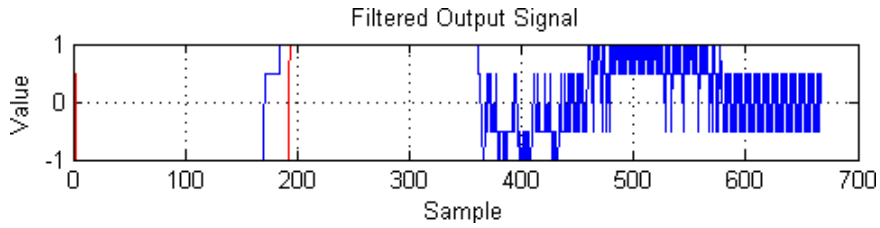

Abbildung 5: Nach Tabelle 1 von AccelDSP automatisch gewählte Fix-Komma Breiten. Die Sinus-Schwingungen aus Abbildung 4 sind hier noch nicht erkennbar.

	Automatische Wahl		Manuelle Wahl	
Name	**Gesamte Wortbreite**	**Nachkomma Wortbreite**	**Gesamte Wortbreite**	**Nachkomma Wortbreite**
Messung y (Eingangsvektor)	14	12	10	8
Schätzwert S (Ausgangsvektor)	53	16	10	8
Einheitsmatrix I	1	0	1	0
Kalman-Verstärkung K	54	12	9	8
a posteriori Schätzfehlerkovarianz P	54	16	13	8
a priori Schätzfehlerkovarianz P_est	54	15	13	8
Systemrauschen Q	1	0	1	0
Varianz des Messfehlers R	8	0	8	0
a posteriori Schätzwert x	54	17	10	8
a priori Schätzwert x_est	54	17	10	8

Tabelle 1: Von AccelDSP automatische und von mir manuell gewählte Wortbreiten im Vergleich.

Die Folgenden Änderungen werden unternommen:

- Die Schätzwerte sollen die gleichen Wortbreiten wie die Messung aufweisen. 1-Bit Vorkomma zuzüglich 1-Bit Vorzeichen und Nachkommastellen genügen.
- Die Kalman-Verstärkung liegt zwischen Null und Eins. Entsprechend wird die Vorkomma-Wortbreite angepasst.
- Die Schätzfehlerkovarianzen betragen höchstens zwölf. 4-Bit Wert mit Vorzeichen und Nachkommastellen genügen.
- 8-Bit Nachkomma Wortbreite werden gewählt.

Die manuell eingestellten Werte sind in den letzten zwei Spalten der Tabelle 1 zu sehen, die Simulation mit den manuell eingestellten Werten zeigt Bild 6: Die Schwingungen sind klar erkennbar. Damit ist die Umsetzung in Fixkommazahlen erfolgreich und es kann nun der RTL-Code erzeugt werden, es gab keine Warnungen zu arithmetischen Über- oder Unterläufen.

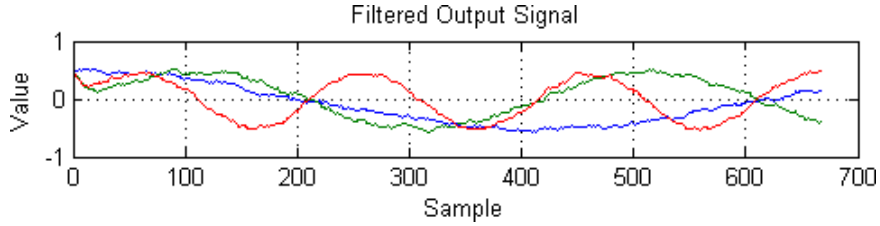

Abbildung 6: Nach Tabelle 1 manuell gewählte Fix-Komma Breiten. Das Ergebnis ist nun vergleichbar mit Abbildung 4.

Instanziiert in der Hardware werden mit 42 Multiplizierer, 74 Addierer und 67 Subtrahierer 74% von 5472 Slices auf einem Virtex 4 FX12 FPGA.

4. Zusammenfassung

Eingebettete Systeme mit Multiprozessorsystemen sind in der Industrie die Zukunft [9], weshalb auch Xilinx beginnt FPGAs mit Dual-Core Prozessor zu produzieren [13]. Deswegen wird im Themenbereich „High Performance Embedded Computing“ (HPEC) der HAW-Hamburg an Multiprozessorsystem in einer SMP-Konfiguration geforscht. Herausforderung sind hierbei u.a. die Aufgabenteilung zwischen den einzelnen Prozessoren, die Synchronisation zwischen den einzelnen Tasks, das effektive Ausnutzen der vorhanden Ressourcen, Cache-Kohärenz, Speicher-Konsistenz sowie das Debuggen der verteilten Anwendung [5] [9]. Als Grundlage dienen FPGAs als System-On-Chip auf denen sich mehrere Prozessoren als Soft-IP-Core, mit verschiedenen Hardware-Konfigurationen modellieren lassen. In Folgeprojekten lässt sich die Forschung auf den angekündigten FPGAs mit physischen Dual-Core ARM-Prozessoren durchzuführen. Die in Eingebetteten Systemen häufig anzutreffenden ARM-Prozessoren haben den Vorteil [13] von vielen Software-Plattformen unterstützt zu werden.

Als praktische Anwendung dient hierbei das Fahrzeug des FAUST-Projektes, daß autonom einer Fahrspur folgen soll. Die Fahrspur wird von einer Kamera aufgenommen, die Bildverarbeitung arbeitet in einem Eingebetteten FPGA System-On-Chip beschleunigt außerhalb des Prozessors. Zur Fahrspurerkennung wird eine Polynomapproximation verwendet, erste Berechnungen diese durch einen robusteren Kalman-Filter auszutauschen sind vorhanden [12]. Der in Kapitel 3 vorgestellte Kalman-Filter ist ein rekursiver Schätzfilter der zusätzlich zur Messung ein Mathematisches Modell benutzt um daraus einen optimalen Schätzwert für die aktuelle Position zu berechnen. Durch die rekursive Arbeitsweise hat das Kalman-Filter nur einen geringen Speicherbedarf was den Einsatz in FPGAs vereinfacht [12].

Zur Entlastung der Prozessoren werden aufwendige Aufgaben wie Bildverarbeitung in Hardware-Beschleuniger-Module ausgelagert, die Entwicklung der Module kann durch modellgetriebene Entwicklungswerkzeuge wie AccelDSP von der Firma Xilinx unterstützt werden: Der Algorithmus wird in Matlab entwickelt und verifiziert, anschließend zusammen mit einer Testbench in AccelDSP importiert. Es wird zuerst eine Simulation mit Fließkommazahlen durchgeführt, das Ergebnis der ersten Simulation ist die Vorlage für alle weiteren Simulationen. Vor der Codegenerierung in einer Hardware-Beschreibungssprache werden die Fließkommazahlen von Matlab in die Festkommazahlen des FPGA konvertiert, wo der Entwickler durch berechnete Voreinstellungen und Werkzeuge von AccelDSP unterstützt wird. Die Konvertierung ist abgeschlossen wenn die Simulation mit den aktuellen Festkomma-Parametern ein an die Fließkommazahlen angenähertes, für den Entwickler akzeptables, Ergebnis erreicht hat.

Literatur

[1] BORDASCH, Heiko: Multiprozessor System on Chip / HAW Hamburg. 2010. – Ausarbeitung

[2] CIGAN, Eric ; LALL, Narinder: Integrating MATLAB Algorithms into FPGA Designs. In: *DSP Magazine* 1 (2005). – URL http://www.xilinx.com/publications/

[3] DORTA, T. ; JIMENEZ, J. ; MARTIN, J.L. ; BIDARTE, U. ; ASTARLOA, A.: Overview of FPGA-Based Multiprocessor Systems. In: *Reconfigurable Computing and FPGAs, 2009. ReConFig '09. International Conference on*, Dezember 2009, S. 273 –278

[4] EBERT, Christof ; SALECKER, Jü a.: Guest Editors' Introduction: Embedded Software Technologies and Trends. In: *Software, IEEE* 26 (2009), Mai, Nr. 3, S. 14 –18. – ISSN 0740-7459

[5] FLYNN, Michael J.: Some Computer Organizations and Their Effectiveness. In: *Computers, IEEE Transactions on* C-21 (1972), September, Nr. 9, S. 948 –960. – ISSN 0018-9340

[6] HUERTA, P. ; CASTILLO, J. ; PEDRAZA, C. ; CANO, J. ; MARTINEZ, J.I.: Symmetric Multiprocessor Systems on FPGA. In: *Reconfigurable Computing and FPGAs, 2009. ReConFig '09. International Conference on*, Dezember 2009, S. 279 –283

[7] JENNING, Eike: *Systemidentifikation eines autonomen Fahrzeugs mit einer robusten, kamerabasierten Fahrspurerkennung in Echtzeit*, HAW Hamburg, Diplomarbeit, 2008

[8] JESTEL, Andre: Hardware-Software Partitionierung für SoC-Plattformen / HAW Hamburg. 2011. – Ausarbeitung

[9] MIGNOLET, J.-Y. ; WUYTS, R.: Embedded Multiprocessor Systems-on-Chip Programming. In: *Software, IEEE* 26 (2009), Mai, Nr. 3, S. 34 –41. – ISSN 0740-7459

[10] NISCHWITZ ; FISCHER ; HABERÄCKER: *Masterkurs Computergrafik und Bildverarbeitung*. second. Vieweg Friedr. + Sohn Ver, 2007. – ISBN 978-3-8348-0186-9

[11] PANDABOARD: *PandaBoard*. – URL http://pandaboard.org

[12] PETERS, Falko: *FPGA-basierte Bildverarbeitungspipeline zur Fahrspurerkennung eines autonomen Fahrzeugs*, HAW Hamburg, Diplomarbeit, 2009

[13] SANTARINI, Mike: Xilinx Architects ARM-Based Processor-First, Processor-Centric Device. In: *Xcell Journal* 71 (2010). – URL http://www.xilinx.com/publications/xcellonline/

[14] SCHMIDT, Andrew: *Embedded Systems Design with Platform FPGAs*. first. Morgan Kaufmann, 2010. – ISBN 978-0-12-374333-6

[15] WOLF, W. ; JERRAYA, A.A. ; MARTIN, G.: Multiprocessor System-on-Chip (MPSoC) Technology. In: *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on* 27 (2008), Oktober, Nr. 10, S. 1701 –1713. – ISSN 0278-0070

[16] Xilinx (Veranst.): *AccelDSP Synthesis Tool User Guide*. April 2008. – URL http://www.xilinx.com/support/documentation/sw_manuals/acceldsp_user.pdf

[17] Xilinx (Veranst.): *MATLAB for Synthesis Style Guide*. December 2009. – URL http://www.xilinx.com/support/documentation/sw_manuals/xilinx11/acceldsp_style.pdf

A. Matlab Quellcode

A.1. AccelDSP Testbench

```
0001 % Real Input Signal
0002 A = [
0003     cos(0:pi/400:(5/3)*pi)/2;
0004     sin(0:pi/200:(10/3)*pi)/2;
0005     sin(0:pi/100:(20/3)*pi)/2
0006     ]';
0007
0008 % Add some noise
0009 A_in = A + 0.5*(rand(size(A)) - 0.5);
0010
0011 % Run Kalman
0012 for i=1:size(A,1),
0013     [S(i,:)] = simple_kalman(A_in(i,:));
0014 end
0015
0016 % Plot
0017 subplot(3,1,1);
0018 plot(A);
0019 axis([0 700 -1 1])
0020 title('Input Signal without Noise')
0021
0022 subplot(3,1,2);
0023 plot(A_in);
0024 axis([0 700 -1 1])
0025 title('Input Signal with Random Noise')
0026
0027 subplot(3,1,3);
0028 plot(S);
0029 axis([0 700 -1 1])
0030 title('Filtered Output Signal')
```

A.2. Verwendeter Kalman Filter

```
0001 function [S] = simple_kalman(y)
0002     persistent x_est P_est
0003
0004     if isempty(P_est)
0005         P_est = eye(size(y,2))*9; % Schätzfehlerkovarianz = 9
0006         x_est = ones(size(y,2),1)/2; % Systemzustand = 0.5
0007     end;
0008
0009     I = eye(size(y,2)); % Einheitsmatrix
0010     Q = eye(size(y,2)); % Systemrauschen = 1
0011     R = eye(size(y,2))*128; % Varianz des Messfehlers = 128
0012
0013     % correction step:
0014     % (1) Berechnung der Kalman-verstärkung
0015     K = P_est * accel_qr_inverse(R + P_est);
0016     % (2) Korrektur der Schätzung mit der Messung
0017     x = x_est + K * (y' - x_est);
0018     % (3) Korrektur mit der Messung
0019     P = (I - K)*P_est;
0020
0021     % estimate step:
0022     % (1) Präzidiere den Systemzustand
0023     x_est = x;
0024     % (2) Präzidiere die Fehlerkovarianzmatrix
0025     P_est = P+Q;
0026
0027     % Return
0028     S = x';
```

A.3. Kalman Filter Vorlage von Xilinx

```
0001 function [S] = simple_kalman(A)
0002     DIM = size(A,2);
0003     persistent p P_cap
0004     if isempty(P_cap)
0005         P_cap = [8 0 0; 0 8 0; 0 0 8];
0006         p     = ones(DIM,1)/2;
0007     end;
0008
0009     I = eye(DIM);
0010     R = [128 0 0;0 128 0; 0 0 128];
0011
0012     % estimate step:
0013     %p_est     = p;
0014     P_cap_est = P_cap+I;
0015
0016     % correction step:
0017     K     = P_cap_est * inv(P_cap_est+R);
0018     p     = p + K * ( A' - p );
0019     P_cap = (I - K)*P_cap_est;
0020     S = p';
```

Glossar

ARM 32-Bit RISC Prozessor Architektur, weit verbreitet in Eingebetten Systemen.

ASIC Application specific integrated circuit

DSP Digitaler Signalprozessor

FAUST Fahrerassistenz- und autonome Systeme

FPGA Field Programmable Gate Array

IC Integrierter Schaltkreis

Matlab Mathematik Software, vorrangig für numerische Berechnung und Simulation verwendet.

POSIX Portable Operating System Interface

RTL Register transfer level, High-level Darstellung von Schaltkreisen.

SMP Symmetric multiprocessing

VHDL Very High Speed Integrated Circuit Hardware Description Language, Hardwarebeschreibungssprache

Xilinx Ein führender Hersteller von FPGAs.