Domänenspezifische Sprache für Videokodierung und dekodierung

Denis Fleischhauer

Hochschule für angewandte
Wissenschaften Hamburg
Email: denis.fleischhauer@haw-hamburg.de

I. Einleitung und Zielsetzung

Die Entwicklung neuer Verfahren für die Videokompression ist zugleich mit der technischen Umsetzung verbunden. Das macht es für die Forscher dieser Verfahren umständlicher diese einerseits zu implementieren, da dafür Kenntnisse in der jeweiligen Programmiersprache notwendig sind und die Besonderheiten dieser Sprache beachtet werden müssen. Andererseits ist die Umsetzung dieser Verfahren, wenn für die Implementierung andere Sprachexperten eingesetzt werden, mit Kommunikationsschwierigkeiten verbunden, da die gewünschte Funktionalität erst auf einer abstrakten Ebene, jenseits technischer Details, erläutert werden muss. Die Referenzimplementierung des High Efficiency Video Coding (HEVC) an dem sich diese Arbeit orientiert ist in der Programmiersprache C++ umgesetzt. Sie ist von der Forschungsgruppe des Fraunhofer-Instituts entwickelt worden. Der endgültiger Einsatz dieser Implementierung kann Beispielsweise auf den Field-Programmable Gate Array (FPGAs) z.B. der Kameras erfolgen. Die Implementierung dieser Verfahren sowie die Programmierung der FPGAs erfordern technisches Vorwissen in diesen Bereichen. Um die Experten der in HEVC eingesetzten Verfahren an der Entwicklung der Software direkt teilhaben zu lassen, sollen diese Verfahren und deren Einsatz auf FPGAs durch eine Domain-Specific Language (DSL) den Fachleuten zugänglich gemacht werden. HEVC wird hier als Gegenstand betrachtet, welcher mit Mitteln einer DSL auf eingebetteten Systemen eingesetzt werden kann.

Das Ziel dieser Arbeit ist es das Verfahren, welches Discrete Cosine Transform (DCT) beschreibt und aus dem HEVC stammt, mittels einer DSL auf die FPGAs zu bringen. Dabei sind die fachlichen Aspekte des DCTs besonders wichtig, da sie den Fachexperten solcher Algorithmen die Möglichkeit bieten sich am Entwicklungsprozess direkt zu beteiligen. Weiterhin soll im Rahmen dieser Arbeit Konzept einer DSL entwickelt werden. Mit Hilfe dieses Konzepts soll es möglich werden eine DSL zu entwickeln um DCT in Form einer mathematischen Formel beschreiben zu können.

Das Konzept für die Entwicklung dieser DSL soll nachfolgende Abbildung 1 zugrunde liegen und das eigentliche Vorhaben nochmal hervorheben. Die Abbildung zeigt einen Actor, der sich vorstellt verschiedene Verfahren direkt auf einem eingebetteten System einzusetzen. Der Actor repräsentiert Fachexperten des jeweiligen Verfahrens aus dem HEVC. Dies möchte er mittels einer DSL schaffen, die C++ (denkbar wären auch andere Sprachen) Quelldateien erzeugt. Diese Quelldateien sollen nach dem Erzeugen mit Hilfe von entsprechender Software, zum Beispiel von Xilinx, kompiliert und auf FPGA des eingebetteten Systems transportiert werden. Der übertragene Code soll auf FPGA laufen und seine Aufgabe erfüllen, die darin besteht, definierte Operationen auszuführen.

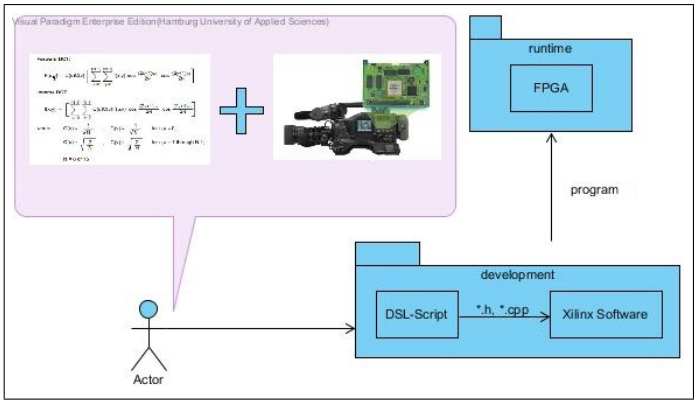


Figure 1. Zielsetzung

II. Grundbegriffe

Nachfolgend soll auf einige wichtige Grundbegriffe eingegangen werden, die bereits gefallen sind aber noch nicht näher erläutert wurden. Diese Begriffe sind wichtig, da ohne sie das Verständnis der nachfolgenden Inhalte und ohne entsprechende Vorkenntnisse bedingt erfolgreich sein wird. Der Aufbau dieses Abschnittes gliedert sich in vier Blöcke HEVC, DCT, FPGA und schließlich DSL in der gegebenen Reihenfolge.

Es sollen nur diejenigen Aspekte dieser Blöcke betrachtet werden mit deren Hilfe es möglich wird sie überein zu bringen. Dabei sollen deren technische Details nur in einem Umfang betrachtet werden in dem es für die Arbeit relevant ist.

A. HEVC

High Efficiency Video Coding (HEVC) ist ein in den letzten Jahren entwickelter Standard zur Videokomprimierung. Dieser Standard wurde von Joint Collaborative Team on Video Coding (JCT-VC) entwickelt. JCT-VC ist ein Team das aus der Zusammenarbeit von ITU-T Video Coding Experts Group(VCEG) und ISO/IEC Moving Picture Experts Group(MPEG) entstand. Erste Version des HEVC erschien 2013 durch die Veröffentlichung von JCT-VC[1]. Das Hauptziel der HEVC Standardisierung ist es gewesen eine deutlich bessere Kompressionsleistung im Vergleich zu seinem Vorgänger zu erhalten. Die Kompression soll die Bitrate um 50% bei gleichbleibender wahrnehmbaren Qualität verringern. H.265, wie HEVC auch bezeichnet wird, ist in Hinblick auf Ultra High Definition Television(UHDTV) bzw. Frame-Formate in 4k und 8k entwickelt worden und gilt als Nachfolger des

H.264[2].

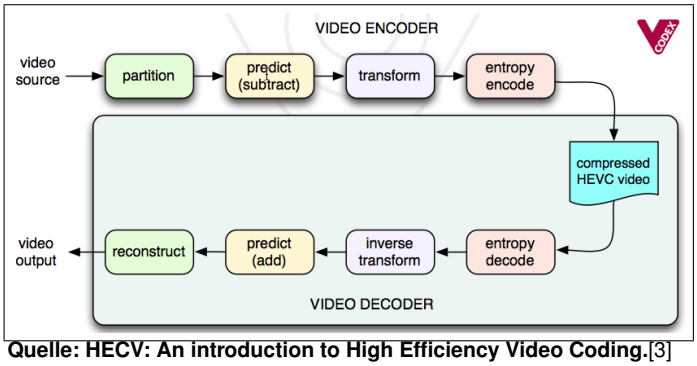
HEVC verfolgt in seiner Technologie den gleichen hybriden Ansatz wie seine Vorgänger bis hin zu H.261. Dieser Ansatz zeichnet sich durch seine Inter- und Intrabildvorhersage und zweidimensionalen Transformationskodierung. Die Vorhersage kommt aus den vorher dekodierten Informationen. Zum einen können diese Informationen auf den Bewegungen in bereits vorhandenen Bildern beruhen (Interbildvorhersage). Zum Anderen auf den bereits dekodierten Vorlagen (Bildregionen) des gleichen Bildes (Intrabildvorhersage)[1].

Die Funktionsweise des HEVC basiert auf der selben Struktur wie auch sein direkter Vorgänger H.264. Die Quellvideos, die aus einer Abfolge von Video-Frames bestehen werden vom Encoder komprimiert und in Bitstrom umgewandelt. Dieser Bitstrom kann entweder übertragen oder gespeichert werden. Ein Decoder hingegen empfängt oder ruft den Bitstrom auf und dekomprimiert ihn. So entsteht wieder die Videosequenz.

Um eine Videosequenz bestehend aus einer Reihe von Bildern zu komprimieren partitioniert der Encoder jedes Bild in mehrere Einheiten. Für jedes der Einheiten wird eine Vorhersage getroffen. Diese Vorhersagen können Intra- oder Inter-Verfahren benutzen. Anschließend wird die Vorhersage von der Einheit abgezogen. Als nächstes erfolgt die Transformation und Quantifizierung des Unterschieds zwischen dem Originalbild und der Vorhersage. Nachfolgend kommt Entropie-Kodierung des transformierten Outputs und Informationen zur Vorhersage sowie zum Kodierungsmodus und des Headers.

Der Dekoder kehrt die vom Encoder gemachte Schritte um. Zunächst wird Entropie-Dekodierung durchgeführt und alle Elemente der kodierten Sequenz extrahiert. Skalierung und die Umkehrung der Transformation als nächstes. Weiter folgen die Vorhersage der Einheiten und das Hinzufügen der Vorhersage zum Output der umgekehrten Transformation. Und schließlich Rekonstruktion des nunmehr dekodierten Videobildes.

Die Abbildung 2 bietet einen Überblick über den HEVC Standard. Dargestellte Blöcke stehen für Hauptfunktionen des HEVC und haben jeweils einen Gegenpart im De- bzw. Encoder.



Quelle: HECV: An introduction to High Efficiency Video Coding.[3]

Figure 2. Überblick über HEVC

Partitionierung: HEVC unterstützt hochflexible Partitionierung von Videosequenzen. Jedes kodierte Videoframe oder Bild wird in Tiles und Slices partitioniert, die später in Coding Tree Units (CTUs) partitioniert werden. CTUs sind Basiseinheiten dieser Kodierung und können bis zu 64x64 Bildpunkte groß sein. Außerdem können sie in eine Anzahl von quadratischen Regionen genannt Coding Units (CUs) unterteilt werden.

Vorhersage: Während der Vorhersage werden CUs in ein oder mehrere Prediction Units (PUs) unterteilt, die durch Intra- oder Interverfahren zur Vorhersage bestimmt werden. Inter steht dabei für eine Methode in der die PU aus Bilddaten ein oder mehreren Referenzbildern vorhergesagt wird. Intra nimmt zur Vorhersage Daten von benachbarten Regionen des selben Bildes.

Transformation und Quantifikation: Nach der Vorhersage wird das erzeugte Datum durch ein Verfahren transformiert das dazu Discrete Cosine Transform (DCT) oder Discrete Sine Transform (DST) benutzt. Eine oder mehrere block-Transformationen der Größe 32x32, 16x16, 8x8, 4x4 werden zu den restlichen Daten in die CU hinzugefügt.

Entropie-Kodierung: Der Kodierte HEVC-Bitstream, dass aus quantifizierten Transformationskoeffizienten, Vorhersageinformationen sowie Vorhersagemodus und Bewegungsvektor, Informationen zur Partitionierung und anderen Headerdaten besteht wird durch Context Adaptive Binary Arithmetic Coding (CABAC) Verfahren kodiert[2] [3].

B. DCT

Discrete Cosine Transform (DCT) spielt eine große Rolle in der Video- bzw. Bildkompression wegen ihrer hohen Effizienz. DCT findet unter anderem den Einsatz in dem aktuellen Standard zur Videokodierung (HEVC). DCT wird dazu eingesetzt um, wie im Falle des JPEG, MPEG-2/4 oder H.263, Videosequenzen bzw. Bilder zu komprimieren. Auch HEVC unterstützt dieses Verfahren. Das Verfahren, das die Komprimierung rückgängig macht heißt Inverse DCT (IDCT). Es gib eine Reihe von unterschiedlichen DCT-Verfahren zum Komprimieren von Bildern. Diese müssen jedoch immer den richtigen Gegenpartner im Decoder im Falle der Dekomprimierung haben. DCT wurde durch die Fähigkeit zeitbezogene Daten in frequenzbezogene zu überführen bekannt. Das macht das Resultat der Kodierung kompakter und macht es möglich Redundanzen zu entfernen. Die Elemente der Matrix in DCT sind reelle Zahlen, die durch eine endliche folge von Bits repräsentiert werden kann. Die Repräsentation durch Bits führt jedoch unweigerlich zu einem Drift, dass eine Abweichung zwischen den dekodierten Daten der Decoder und Encoder bedeutet. Das hat die Forscher auf den Plan gerufen um dieses Verhalten zu untersuchen, dass sich in einer Menge von Papern zu diesem Thema äußert[4].

Eine Variante des DCT-Verfahrens bietet das Paper [5]. Es stellt in der Abbildung 3 dargestellte Formel als eine Vorwärts-Transformation vor. In dieser Formel stellt das 'F' eine Matrix dar, die nach der Transformation als Ergebnis vorliegt. Sie entspricht von ihrer Größe her der Eingangsmatrix 'f'. Während der Iteration über die Matrix, welche mit Daten gefüllt werden soll, werden bedingt von den aktuellen Werten zu 'u' und 'v' jeweils bestimmter Wert für die Berechnung übernommen. Anschließend erfolgt eine Aufsummierung der Werte aus der Quellmatrix, welche mit Hilfe von Kosinusfunktionen und weiteren Berechnungen zuvor aufbereitet werden.

Die DCT-Formel lässt sich in zwei Teile unterteilen. Zum einen in den Teil mit Zeilen und zum anderen in den Teil mit Spalten. Dabei ist der Teil mit Spalten transponierte Matrix

$$F(u,v) = C(u)C(v)\sum_{x=0}^{N-1}\sum_{y=0}^{N-1} f(x,y)\cos\left[\frac{(2x+1)u\pi}{2N}\right] \times \cos\left[\frac{(2y+1)v\pi}{2N}\right]$$

$$C(n) = \begin{cases} \sqrt{\frac{1}{N}}, & \text{for } n = 0 \\ \sqrt{\frac{2}{N}}, & \text{for } n > 0. \end{cases}$$

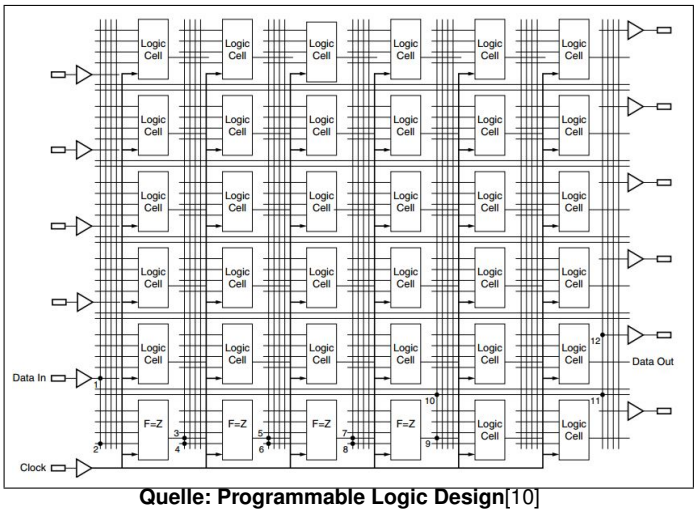
Quelle: Early Determination of Zero-Quantized 8x8 DCT Coefficients[5]

Figure 3. DCT und IDCT Formeln

aus dem ersten Teil. In Bericht von [6] wird unter anderem ein Verfahren vorgestellt mit dem es möglich wird aus der DCT-Formel, die in [5] vorgestellte Matrix zu generieren. Die Koeffizienten dieser Matrix sind konstant und können bei Bedarf mit Hilfe von MATLAB nachgerechnet werden[6].

C. FPGA

Field-Programmable Gate Array (FPGAs) gehören zu den programmierbaren Logikschaltkreisen und dienen zum Aufbau von digitalen, logischen Schaltungen. Sie sind wiederprogrammierbare Schaltkreise, die aus Funktionsblöcken bestehen, welche in einem Array angeordnet sind. Die Funktionsblöcke und die Verbindungen unter ihnen lassen sich vom Anwender programmieren. FPGAs werden in eingebettete Systeme eingesetzt. In dieser Umgebung bieten sie den Vorteil sich auf bestimmte Aufgaben programmieren zu lassen. Auf diese Weise steuern die zum Beispiel Kameras [7][8][9]. Die Abbildung 4 bietet einen Einblick in das Innere eines FPGAs. Sie zeigt eine gleichmäßig angeordnete Struktur von Logikzellen, die Dateneingänge sowie -ausgänge besitzen.



Quelle: Programmable Logic Design[10]

Figure 4. FPGA Architektur

Anwendungsgebiete: FPGAs werden in eingebettete Systeme eingesetzt. In dieser Umgebung bieten sie den Vorteil sich auf bestimmte Aufgaben programmieren zu lassen. Auf diese Weise steuern sie zum Beispiel Kameras[**?**]. Die Entwicklung von solchen Systemen kann in zwei Phasen: Hardwaredesign und Softwaredesign unterteilt werden.

- **Hardwaredesign:** In der ersten Phase erfolgt die Entwicklung der Hardware. Spezielle Softwarelösungen bieten eine Umgebung an in der die Hardware mit allen benötigten Bausteinen virtuell zusammengestellt wird. Komponente, die diese Hardware enthält orientiert sich dabei an der Spezifikation der konkreten Aufgabe. Hardwaredesign erfolgt mittels Hardwarebeschreibungssprachen wie VHDL.
- **Softwaredesign:** Nachfolgende Phase umfasst die Entwicklung einer Anwendung, welche mit Hilfe von Hardwareeigenschaften Aufgaben erledigt. In welcher Sprache die Entwicklung der Anwendungen für die Hardware erfolgt hängt von den eingesetzten Werkzeugen ab. Zum Beispiel bietet Xilinx C++ Unterstützung.

Im Allgemeinen finden die FPGAs Anwendung in einer großen Anzahl von unterschiedlichen Geräten. Sie bringen Wiederverwendbarkeit und Flexibilität mit. Außerdem sind sie kostengünstiger im Vergleich zu fest verdrahteten Lösungen. [11]

Bewertung: Eine Entscheidungshilfe, für einen Einsatz von FPGAs bzw. gegen sie, kann die nachfolgende Bewertung, in Form von einer Auflistung einiger Vorteile und Nachteile, bieten[10].

Vorteile:

- anpassbar auf viele/mehrere Probleme
- Rekonfigurierbarkeit
- geringe Entwicklungszeiten und kosten

Nachteile:

- vergleichsweise geringer Takt
- große Chipfläche
- hohe Stückkosten

D. DSL

Für Domain-Specific Languages (DSLs) findet sich in der Literatur keine bis aufs Wort einheitliche Beschreibung. Viele Autoren beschreiben DSLs in dem sie sich der Definition von Martin Fowler bedienen und diese erweitern.

Fowler definiert DSL in [12] als eine Programmiersprache mit einer beschränkten Ausdrucksstärke, die sich auf eine bestimmte Domäne konzentriert und folgende Kriterien erfüllen soll:

- Computer programming language: DSL wird von Menschen benutzt um dem Computer mitzuteilen, dass er etwas tun soll. Wie auch jede moderne Programmiersprache dessen Struktur zum besseren Verständnis für die Menschen aber auch dem Computer etwas gibt was er ausführen kann entwickelt wurde
- Language nature: DSL ist eine Sprache um einem Computer auszuführende Aktionen mitzuteilen und als solche sollte sie flüssig sein. Ihre Ausdruckskraft sollte nicht nur auf einzelnen Ausdrücken basieren, sondern

auch durch die Art wie sie zusammengesetzt werden können

- Limited expressiveness: Die Ausdruckskraft sollte sich nur auf das Wesentliche, dass nur die Domäne betrifft, beschränken und nicht, wie es bei einer GPL der Fall ist, von der Domäne unabhängige Funkionen abdecken. Ein Minimum an Funktionalität macht die DSL leichtgewichtiger und schneller erlernbar
- Domain focus: Eine in ihrer Funktionalität beschränkte Programmiersprache ist nur dann nützlich, wenn sie die Grenzen ihrer Domäne nicht erweitert. Der Fokus auf die Domäne macht eine DSL besonders

Unterschied zu GPL: Der wesentliche Unterschied einer domänenspezifischen Sprache(DSL) zu einer Allzweckprogrammiersprache ist ihre eingeschränkte Mächtigkeit. Eine DSL ist bei weitem nicht in der Lage alles das zu leisten was eine GPL zu leisten vermag. DSL zielt auf ein bestimmtes Problem einer bestimmten Domäne ab. GPL hingegen hat als Ziel Probleme unterschiedlicher Domänen zu lösen. Auch bei den GPL gibt es viele unterschiedliche Arten, die sich in ihrer Optimierung unterscheiden. Beispielsweise ist es einfacher mit C hardwarenah zu Programmieren als mit Ruby. Ruby hingegen ist auf die Entwicklung von Web-Applikationen optimiert. Die Benutzer einer DSL müssen sich keine Gedanken über die technischen Aspekte der Implementierung machen. Sie können sich voll und ganz auf fachliche Aspekte konzentrieren[13].

DSL-Klassifikation: DSL können auf unterschiedliche Weise in verschiedene Kategorien eingeteilt werden. Zum einen können sie in interne und externe DSL und zum Anderen in nicht-textuelle und textuelle DSL zusammengefasst werden. Dabei zählen interne und externe DSL zu den textuellen d.h. sie werden mit Hilfe von einer in Freitext verfassten Grammatik bedient. Die Klassifikation ist möglich, da diese unterschiedliche Ansätze zur Implementierung verfolgen[14].

Interne DSL erweitern die zugrundeliegende Implementierungssprache. Deren Quelltext ist gültig in der Sprache die sie erweitert. Die Erweiterung birgt den Nachteil, dass für ihren Einsatz die Laufzeitumgebung mitgeschleppt werden muss. Hingegen hat sie den Vorteil keinen eigene Parser, Scanner und andere bereits vorhandene Tools zu benötigen, da diese in der umgebenden Sprache vorhanden sind. Es ist einfacher eine interne DSL mit Sprachen, die eine lose Syntax haben zu erstellen, da diese notwendige Flexibilität bieten. Vertreter der internen DSL sind RAKE für Ruby, MAKE in Unixsystemen und andere mehr.

Externe DSL zeichnen sich dadurch aus, dass sie unabhängig von der Implementierungssprache sind und können eine eigene Syntax haben. Das macht jedoch zumindest die Entwicklung eigener Parser und Scanner notwendig. Wenn die Wahl der Syntax oder des Formats auf einen bereits vorhandenen und verbreiteten Standard fällt ist es sehr wahrscheinlich, dass es unterstützende Software wie Parser bereits gibt. Vertreter der externen DSL sind XML, SQL und andere mehr.

Im Falle der **Nicht-Textuellen DSL**, wie der Name schon sagt, handelt es sich bei dieser Klasse von DSL um eine, die als Eingabesprache eine grafische Repräsentation der textuellen vorzieht. Statt dem Text werden hier vordefinierte grafische Objekte benutzt. Nicht-Textuelle DSL sind aus folgenden Grünten entstanden [12]:

- Textuelle Notation bietet nur eingeschränkte Ausdruckskraft.
- Grafische Modelle stellen einige domänenspezifische Probleme besser dar.
- Die Manipulierung von grafischen Modellen ist für Fachexperten meistens einfacher
- In textuellen DSL wird die domänenspezifische Logik oft hinter komplexen Strukturen versteckt

Bewertung: Um zu entscheiden ob die Entwicklung einer DSL in Frage kommt muss man wissen, dass deren Entwicklung anfangs mit Investitionen verbunden ist, aber Nachhinein die Produktivität z.B. durch geringe Entwicklungszeiten der eigentlichen Produkte, erheblich steigern kann. Das Erstellen von domänenspezifischen Sprachen sollte spätestens in Frage kommen, wenn klar ist das eine bestimmte Art von Problemen oft wiederkehrt. Außerdem müssen die Vor- und Nachteile der DSL im allgemeinen gegeneinander abgewogen werden[14].

Vorteile:

- Innerhalb der Domäne sind DSL ausdrucksstark, den sie befassen sich mit den Abstraktionen, die der genauen Semantik der Domäne entsprechen
- Die Grenzen der Domäne zwingen die DSL kompakt zu bleiben
- DSL bedient sich höheren Abstraktionsniveaus und behandelt beispielsweise keine Implementierungstechniken
- DSL kann die Produktivität steigern indem es gestattet Programme schneller zu schreiben und dabei weniger Quelltext entstehen zu lassen
- DSL ist wegen des geringen Umfangs einfach zu erlernen.
- Die Implementierung mit einer DSL ist weniger fehleranfällig als mit einer GPL
- DSL fördert die Kommunikation zwischen den Domänenexperten und Entwicklern

Nachteile:

- Es ist nicht einfach Sprachen zu definieren, da es ein komplexer Vorgang ist und umfasst mehr Vorkenntnisse als die Definition des Alphabets
- Die Entwicklung kann finanziell, personell und zeitlich teuer sein bzw. werden, weil die Entwicklungszeit und der Umfang im Voraus schwer abzuschätzen sind
- DSL kann die Performanz eines zeitkritischen Programms negativ beeinträchtigen, da DSL immer eine zusätzliche Abstraktionsschicht bedeutet
- Die Akzeptanz der Benutzer ist nicht immer gegeben, da sie erlernt werden muss und damit gewohnte Strukturen bricht
- Es kann schwierig werden mehrere DSL mit einander zu kombinieren, weil sie entwickelt werden um unabhängig zu arbeiten

III. Eingrenzung

Die Umsetzung des gesamten HEVC mit Hilfe einer DSL kommt nicht in Frage wegen des sehr großen Umfangs und der Komplexität, dass den Rahmen dieser Arbeit sprengen würde. Deshalb soll aus dem HEVC für die Umsetzung der Zielsetzung zunächst ein Teil ausgewählt werden. Der Teil des HEVC, das in Frage kommt ist DCT. Denkbar währen aber auch andere.

Auch DSLs bieten viele verschiedene und umfangreiche Möglichkeiten der Umsetzung. Daher soll hier die Einschränkung gelten, dass die zu entwickelnde DSL sich der textuellen Darstellungsform bedienen soll.

FPGAs stellen in dieser Arbeit eine Zielplattform dar und sollen unter diesem Aspekt auch behandelt werden. Modellierung von Hardware kommt an dieser Stelle nicht in Frage. Es soll nur der Aspekt der Einspielung der Software auf den Chip betrachtet werden.

IV. Beispiel einer DSL

Unter der Beachtung der zuvor vorgestellten Eingrenzung könnte eine DSL, wie Sie in der Abbildung 5 zu sehen ist, aussehen. Es beschränkt sich auf DCT und ist in einer textuellen Form dargestellt. Dieses Beispiel bezieht sich auf zwei mögliche Herangehensweisen bei der Implementierung des Verfahrens. Zum einen bietet es die Möglichkeit eine Matrix wie eine aus [6] zu definieren. Dies soll eine hardwarenahe Darstellung des DCT-Verfahrens sein. In Verbindung mit den Regeln zur Transformation, soll eine Umwandlung einer größeren Matrix in eine kleinere möglich sein. Vorteilhaft an dieser Stelle ist, dass es andere Verfahren in eine Matrix einfließen konnten und würde trotzdem von der DSL abgedeckt werden. Eine weitere Möglichkeit DCT darzustellen ist in Form einer Formel. Eine mögliche Formel ist in der Abbildung 3 dargestellt. Die einzelnen Elemente dieser Formel könnten direkt in die Syntax der DSL einfließen. Die Bindung an die Blockgrößen kann durch spezielle Funktion ermöglicht werden, die sich einer Liste bedient in der die einsetzbaren Blockgrößen wie in der Z.33 bzw. Z.38 der Abbildung dargestellt. Eine Möglichkeit Nebenberechnungen oder feste Werte zu definieren soll die anschließend zu definierende Funktion schlanker und besser lesbar machen. Beispiele dafür finden sich in den ZZ.34 und 35. Durch die Definition der Konstante 'PI' wird die Semantik dieser Zahl in den ZZ.42 beibehalten. Grundlegende arithmetische Ausdrücke wie '*', '/', '+', '-' sowie die Operationen höherer Ordnung, wie zum Beispiel 'COS' (Kosinusfunktion), sollen übernommen werden und bekommen ihre Semantik in dem zu generierenden C++ Code. Eine etwas abgewandelte Darstellung bekommt die Summenfunktion. Dies ist jedoch der textuellen Darstellungsform geschuldet. Eingeleitet wird die Funktion nicht mit einem speziellen Zeichen sondern dem Schlüsselwort 'SUM' mit anschließender Angabe ihrer Randwerte in den eckigen Klammern. Schließlich kommt als Element in der Formel eine bedingte Abfrage vor. Dies wird mit Hilfe von einem Aufruf einer speziellen Funktion erledigt. Diese Funktion kann zu bestimmten Werten der übergebenen Parameter einen in dieser Funktion definierten Wert zurückliefern.

Als erstes Sprachfeature soll die Möglichkeit sein nicht nur Intervalle zu definieren sondern auch diese nach Werten zu durchsuchen. Das bietet den Vorteil zum Beispiel in den bedingten Anweisungen (Z.49-52) auf das Vorhanden sein eines Wertes innerhalb eines Intervalls reagieren zu können. Weiterhin wird es die Anzahl von diesen Anweisungen übersichtlich halten, da auf dieser Weise viele Fälle abgedeckt werden können.

```
1  model DCT {
2      matrix dx@16x16 {
3          {64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64},
4          {90, 87, 80, 70, 57, 43, 25, 9, -9, -25,-43,-57,-70,-80,-87,-90},
5          {89, 75, 50, 18,-18,-50,-75,-89,-89,-75,-50,-18, 18, 50, 75, 89},
6          {87, 57,  9,-43,-80,-90,-70,-25, 25, 70, 90, 80, 43, -9,-57,-87},
7          {83, 36,-36,-83,-83,-36, 36, 83, 83, 36,-36,-83,-83,-36, 36, 83},
8          {80,  9,-70,-87,-25, 57, 90, 43,-43,-90,-57, 25, 87, 70, -9,-80},
9          {75,-18,-89,-50, 50, 89, 18,-75,-75, 18, 89, 50,-50,-89,-18, 75},
10         {70,-43,-87,  9, 90, 25,-80,-57, 57, 80,-25,-90, -9, 87, 43,-70},
11         {64,-64,-64, 64, 64,-64,-64, 64, 64,-64,-64, 64, 64,-64,-64, 64},
12         {57,-80,-25, 90, -9,-87, 43, 70,-70,-43, 87,  9,-90, 25, 80,-57},
13         {50,-89, 18, 75,-75,-18, 89,-50,-50, 89,-18,-75, 75, 18,-89, 50},
14         {43,-90, 57, 25,-87, 70,  9,-80, 80, -9,-70, 87,-25,-57, 90,-43},
15         {36,-83, 83,-36,-36, 83,-83, 36, 36,-83, 83,-36,-36, 83,-83, 36},
16         {25,-70, 90,-80, 43,  9,-57, 87,-87, 57, -9,-43, 80,-90, 70,-25},
17         {18,-50, 75,-89, 89,-75, 50,-18,-18, 50,-75, 89,-89, 75,-50, 18},
18         { 9,-25, 43,-57, 70,-80, 87,-90, 90,-87, 80,-70, 57,-43, 25, -9}
19     }
20
21     transform dx@16x16 to 8x8 with
22         col step: first 8
23         row step: every 2
24
25     transform dx@16x16 to 4x4 with
26         col step: first 4
27         row step: every 4
28
29     transform dx@16x16 to 2x2 with
30         col step: first 2
31         row step: every 8
32
33     block_size bl = 4,16
34     const PI = 3.14159
35     const R = 2
36     range i = [0;2]
37
38     func id @block_size N do
39         X(u,v) = C(u) * C(v) * (
40                 SUM[k=0;-1+N](
41                     SUM[l=0;-1+N](
42                         f(k,l) * COS( (2*k+1)*u*PI/2*N ) * COS( (2*l+1)*v*PI/2*N )
43                     )
44                 )
45             )
46     end
47
48     func C(r) do
49         WHERE r < 10 ? 1 + 2 : 3 - 1 END
50         WHERE r >= 10 ? 2 * 1 : 1 / 2 END
51         WHERE r in [0;2] ? 2/1 : 1/2 END
52         WHERE r in i ? 2/1 : 1/2 END
53     end
54 }
```

Quelle: Programmable Logic Design[10]

Figure 5. Beispiel DSL-Script

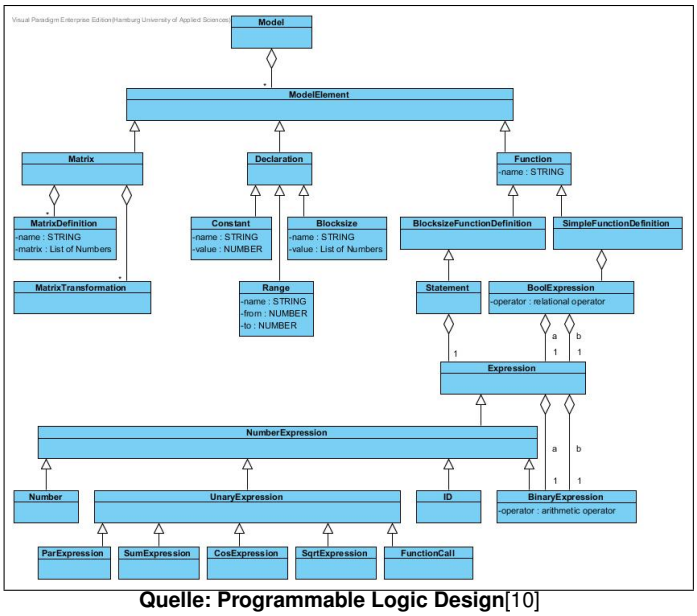
V. Entwurf

Für die zu erstellende Sprache soll folgendes, in der Abbildung 6 dargestelltes, Meta-Modell zugrunde liegen. Dieses Modell soll im Rahmen des Projektes weiterentwickelt werden. Es hat als Grundlage die Beispiel-DSL aus dem vorherigen Artikel. Es enthält wesentliche Teile dieser DSL, welche die beabsichtigte Funktionalität ermöglichen sollen.

Nachfolgend soll auf die Entwicklung des Meta-Modells eingegangen werden. Es werden Parallelen zwischen der Beispiel-DSL aus Abbildung 5 und dem Meta-Modell aus der Abbildung 6 aufgezeigt. Genauer gesagt wird das gewünschte Verhalten aus der Beispiel-DSL in das Meta-Modell übertragen.

Als erstes wird ein Dokument definieren, das im Meta-Modell als 'Model' bezeichnet wird. Dieses Dokument soll weitere Elemente der DSL in sich aufnehmen. Aufzunehmende Elemente werden als 'ModelElement' bezeichnet.

Weitere Unterteilung der Modellelemente findet einerseits in die Definition der Matrix, bezeichnet als 'Matrix', mit ihren Regeln, andererseits in den Abschnitt mit Definition von Konstanten, Intervalle sowie Blockgrößen. Diese sind im Modell unter 'Declaration' als 'Constant', 'Blocksize' und 'Range' zu



Quelle: Programmable Logic Design[10]

Figure 6. Meta-Modell für die DSL zur Beschreibung von DCT-Verfahren

finden. Und zuletzt in den Teil mit Funktionsdefinitionen mit der Bezeichnung 'Function'.

Nachfolgende Auflistung bietet weitere Informationen zu den Elementen des Modells:

- **Matrix:** Beinhaltet eine Matrixdefinition und Regeln zur Matrixumwandlung('MatrixTransformation'). Die erste besteht aus einem Namen zur Identifikation dieser Matrix und einer Liste von Zahlen als Inhalt dieser Matrix. Es können mehrere Matrizen aber auch mehrere Regeln definiert werden, die die Matrizen umwandeln können.
- **Deklaration:** In diesem Modellelement kann eine oder mehrere Konstanten ('Constant'), Intervalle ('Range') oder Blockgrößen ('Blocksize') definiert werden. Konstanten können durch einen Namen identifiziert werden und zeigen auf einen Zahlenwert. Intervalle haben einen Namen zur Identifikation sowie zwei Zahlenwerte, welche die Grenzen des Intervalls festlegen. Es können nur abgeschlossene Intervalle definiert werden. Blockgrößen beinhalten eine Liste von Zahlenwerten. Sie sind dazu gedacht um in der späteren Codegenerierung an die Blockgröße angepasste Funktionen zu erstellen. Der Zweck der Konstanten und Intervalle ist es die später zu implementierende Funktion zur Beschreibung zum Beispiel des DCT ŝauberżu halten. Sie können evtl. komplizierte Nebenrechnungen enthalten.
- **Funktion:** Funktionen werden durch einen Namen identifiziert. Sie werden Nachfolgend gesondert betrachtet, da sie einen umfangreichen Abschnitt bilden.

Wie auch in der Beispiel-DSL gibt es zwei unterschiedliche Arten von Funktionen. Eine Art reagiert entsprechend der Vorgabe auf bestimmte Eingangsparameter in dem es Bedingungen prüft. Zweite Art definiert das DCT-Verfahren in dem es durch grundlegende mathematische Operationen wie '+', '*', '/' und '-' sowie Operationen höherer Ordnung wie Kosinus ('COS') unterstützt wird. An dieser Stelle in der DSL können außerdem Funktionsaufrufe, Operation zum Ziehen der Wurzel, zur Berechnung von Summen vorkommen. Nähere Informationen sind in der nachfolgenden Auflistung dargestellt:

- **Klammerausdruck ('ParExpression'):** erlaubt Klammerung, welche wie üblich den Vorrang einer auszuführenden Rechenoperation anzeigt. Innerhalb der Klammern findet sich ein Ausdruck.
- **Summenausdruck ('SumExpression'):** ist der Summenfunktion aus Mathematik gleich.
- **Kosinusausdruck ('CosExpression'):** stellt Kosinusfunktion dar.
- **Wurzelausdruck ('SqrtExpression'):** als Sprachelement ermöglicht es in den Formeln Wurzelfunktion einzusetzen
- **Funktionsaufruf ('FunctionCall'):** erlaubt den Aufruf von eigenen Funktionen

Denkbar währen an dieser Stelle auch andere Funktionen wie zum Beispiel Logarithmus und Sinus. Diese sind jedoch zunächst nicht relevant.

VI. Zusammenfassung und Ausblick

Dieser Arbeit hatte zum Ziel Grundlagen und theoretisches Konzept auszuarbeiten mit deren Hilfe sich die DSL im Projekt entwickeln lässt. Zunächst wurde das Ziel der Arbeit in der Einleitung definiert. Als nächstes sind die Grundbegriffe erläutert worden. Mit anschließender Eingrenzung wurden die Grenzen dieser Arbeit festgelegt. Und schließlich Modelle entwickelt worden in Form einer Beispiel-DSL und eines Meta-Modells. Dabei fasst das Meta-Modell auf eine Abstrakte weise die Funktionen der DSL zusammen und Beispiel-DSL gibt der DSL mögliches Erscheinungsbild und eröffnet damit die Möglichkeit sich die DSL genauer vorstellen zu können.

Als weiteres Vorgehen soll im Projekt des Masterstudiums die DSL mit Hilfe von Xtext realisiert werden. Dabei sollen in diesem Bericht erarbeiteten Aspekte unmittelbar in die Realisierung einfließen. In dem aus dem Meta-Modell unter Berücksichtigung der Beispiel-DSL eine Grammatik entwickelt wird. Die Semantik der DSL soll C++ Code erzeugen aber auch eine Möglichkeit bieten eine Transformation in weitere Sprachen vorzunehmen, dass sich mit Hilfe einer Software aufbereiten und auf die FPGAs transportieren lässt.

References

[1] J. Ohm and G. Sullivan, "High efficiency video coding: the next frontier in video compression [standards in a nutshell]," Signal Processing Magazine, IEEE, vol. 30, no. 1, Jan 2013, pp. 152–158.

[2] G. J. Sullivan, J.-R. Ohm, W. Han, and T. Wiegand, "Overview of the high efficiency video coding (hevc) standard." IEEE Trans. Circuits Syst. Video Techn., vol. 22, no. 12, 2012, pp. 1649–1668.

[3] HEVC: an introduction to High Efficiency Video Coding, Vcodex, 2013.

[4] T. Ma, C. Liu, Y. Fan, and X. Zeng, "A fast 8x8 idct algorithm for hevc," in ASIC (ASICON), 2013 IEEE 10th International Conference on, 10 2013, pp. 1–4.

[5] X. Ji, S. Kwong, D. Zhao, H. Wang, C.-C. Kuo, and Q. Dai, "Early determination of zero-quantized 8x8 dct coefficients," Circuits and Systems for Video Technology, IEEE Transactions on, vol. 19, no. 12, 12 2009, pp. 1755–1765.

[6] P. Leila, "Internship report," University Of The West Of Scotland, Tech. Rep., 2013.

[7] N. George, H. Lee, D. Novo, T. Rompf, K. J. Brown, A. K. Sujeeth, M. Odersky, K. Olukotun, and P. Ienne, "Hardware system synthesis from domain-specific languages," in Field Programmable Logic and Applications (FPL), 2014 24th International Conference on, Sept 2014, pp. 1–8.

[8] C. Kulkarni, G. Brebner, and G. Schelle, "Mapping a domain specific language to a platform fpga," in Design Automation Conference, 2004. Proceedings. 41st, July 2004, pp. 924–927.

[9] O. Reiche, M. Schmid, F. Hannig, R. Membarth, and J. Teich, "Code generation from a domain-specific language for c-based hls of hardware accelerators," in Proceedings of the 2014 International Conference on Hardware/Software Codesign and System Synthesis, ser. CODES '14, 2014, pp. 17:1–17:10.

[10] Programmable Logic Design, Xilinx, 2008.

[11] W. Vanderbauwhede, M. Margala, S. Chalamalasetti, and S. Purohit, "A c++-embedded domain-specific language for programming the mora soft processor array," in Application-specific Systems Architectures and Processors (ASAP), 2010 21st IEEE International Conference on, July 2010, pp. 141–148.

[12] M. Fowler and R. Parsons, Domain-specific languages, 1st ed. Addison-Wesley, 2011.

[13] M. Voelter, S. Benz, C. Dietrich, E. Birgit, H. Mats, L. Kats, E. Visser, and G. Wachsmuth, DSL Engineering : Designing, Implementing and Using Domain-Specific Languages. dslbook.org, 2013.

[14] D. Ghosh, DSL in action. Manning, 2011.