Wiederverwendungsgerechte Codegenerierung von FEC-Applikationen für dynamisch rekonfigurierbare Systeme

J. Schneider, V. Kotzsch

FhG IIS - Außenstelle Entwurfsautomatisierung
Zeunerstr. 38
D-01069 Dresden
{jsch|vincent}@eas.iis.fhg.de

Abstract. Dieser Beitrag beschreibt die Bereitstellung wiederverwendbarer Datenpfadkomponenten durch Generatorwerkzeuge für die Klasse der RS- *(Reed-Solomon)* Fehlerkorrekturcodes. Die Parameterisierbarkeit und Modularisierbarkeit der FEC- *(Forward Error Correction)* Applikation erfolgt werkzeuggestützt bez. der unterschiedlichen Codeparameter, optimierter Teilkomponenten und der Algorithmenauswahl. Bei der Partitionierung und der Steuerung der Applikation werden partiell/dynamische Rekonfigurationsabläufe ermöglicht. Die durchgeführten Untersuchungen basieren auf dem XILINX-Modular-Design-Flow (MDF) [1]. Für das Prototyping stand ein ADM-XRC Board der Firma Alpha Data zur Verfügung [15].

1 Einleitung

1.1 Motivation

Rekonfigurierbare HW- *(Hardware)* Architekturen unterscheiden sich bez. ihres internen Aufbaus, ihrer Arbeitsweise und ihrer Rekonfigurations-Eigenschaften erheblich voneinander. Eine große Anzahl dieser HW-Architekturen ist partiell/dynamisch rekonfigurierbar, wodurch sich eine Vielzahl von Vorteilen ergibt [1],[14]. Dadurch lassen sich z.B. adaptive [2] und fehlertolerante Syteme implementieren bzw. können umfangreiche Applikationen in Kombination mit Auslagerungsmechanismen innerhalb begrenzter HW-Ressourcen realisiert werden. Voraussetzung für die Nutzung dieser Rekonfigurations-Eigenschaft ist, daß die Applikation in voneinander unabhängige System-Komponenten partitionierbar ist, welche in ein zeitliches Ablaufschema eingeordnet werden können. Die Zeitablaufsteuerung wird durch speziell entwickelte applikations- und anwendungsspezifische Controller-Komponenten realisiert.

Um der Forderung nach kürzeren Entwicklungszeiten *(time-to-market)* auch beim Entwurf von Komponenten für partiell/dynamische System gerecht werden zu können, ist der Einsatz von Wiederverwendungsmethoden notwendig. Zu diesem Zweck wird ein Generatorwerkzeug vorgestellt, welches Entwurfsunterstützung bez. der Generierung wiederverwendbarer FEC-Komponeten und Kommunikations-Schnittstellen bietet und Empfehlungen für die Systempartitionierung unterbreitet.

Die weiteren Abschnitte dieses Beitrages sind wie folgt organisiert. Abschnitt 2 führt RS-Codes ein und beschreibt Möglichkeiten zur Partitionierung und Ablaufsteuerung von FEC-Applikationen für partiell/dynamisch rekonfigurierbare Systeme. In Abschnitt 3 erfolgt die Darstellung des Codegenerator-Architekturansatzes und die Beschreibung der wiederverwendungsgerechten Codegenerierung. Der verwendete Entwurfsablauf wird in

Abschnitt 4 beschrieben. Die Ergebnisauswertung folgt in Abschnitt 5. Einige Schlußfolgerungen und zukünftige Arbeiten werden in Abschnitt 6 diskutiert.

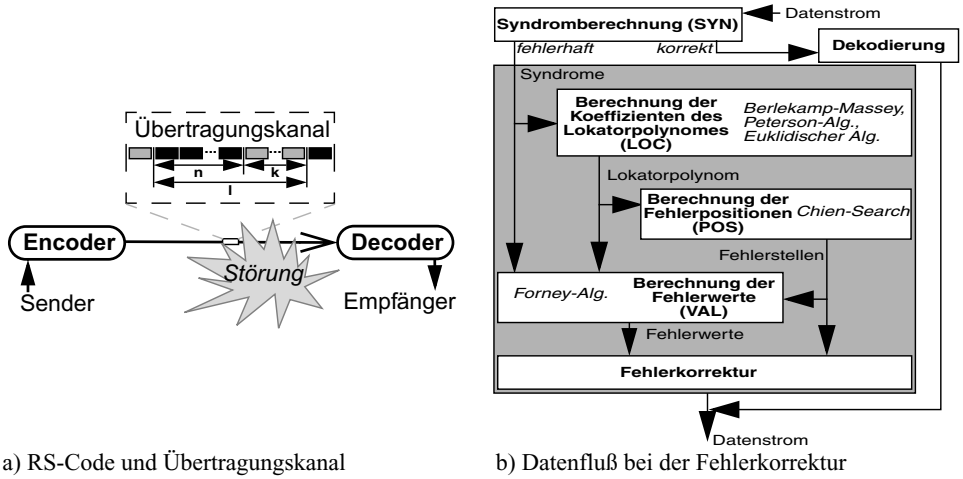
2 FEC-Applikationen

Eine Vielzahl der in der Nachrichten- und Datenverarbeitung eingesetzten Anwendungen verwenden Übertragungskanäle für die Kommunikation zwischen den Baugruppen. Bei diesen Kanälen kann es sich um Kabel- oder Funk-Netzwerke, Bussysteme auf Leiterkarten oder chipinterne Kommunikationsstrukturen handeln, auf welchen es zu Störungen kommen kann. Diese Störungen können durch eine Fehlerkorrektur unter Einhaltung der in Abschnitt 2.1 beschriebenen Fehlerkorrektureigenschaften beseitigt werden. Große praktische Relevanz für industrielle Anwendungsgebiete besitzen die sehr oft eingesetzten Blockcodes (BCH-Codes, RS-Codes) und Faltungscodes (Trellis, Viterbi, Turbo-Codes). Diese Codes sind hinsichtlich ihrer Parametervielfalt genau an die Zielapplikationen anpaßbar. Die variablen Parameter sind beispielsweise das verwendete Modularpolynom, der Fehlerkorrekturgrad *(f_k)*, Korrekturalgorithmen, Codeverkürzungen, Codeverkettungen oder eine Parallelisierung zur Erhöhung des Datendurchsatzes [3].

RS-Code besitzen für zahlreiche technische Anwendungen vielfältige Einsatzmöglichkeiten und bieten durch ihre Parameterisierbarkeit und Modularisierbarkeit ein hohes Potential für partiell/dynamische Rekonfigurationsabläufe. Sie fungieren als Referenz für Applikationen mit ähnlichen Eigenschaften [3],[4].

2.1 Einführung in RS-Codes

RS-Codes bilden die Grundlage für Fehlerkorrekturverfahren, welche zur Übertragungssicherung mit der Möglichkeit zur Korrektur fehlerhaft übertragener Datenblöcke durch angefügte Redundanz verwendet werden. Die Korrektur-Daten werden berechnet, als Datenblock an die Nutzdaten angehängt und mit diesen zusammen übertragen. Das Codewort besteht aus den Informations-Symbolen (Nutzdaten) und einer Anzahl von Korrektur-Symbolen. RS-Codes, entspechend Abbildung 1 angegeben durch *RS(l,n),* sind über einem erweiterten Galois-Feld *$GF(2^m)$* spezifiziert, wobei $l = 2^m - 1$, $n < l$ ist [3],[4],[11].



a) RS-Code und Übertragungskanal b) Datenfluß bei der Fehlerkorrektur

Abb. 1: RS-Code und Fehlerkorrekturablauf

Ein erweitertes *GF* enthält eine abgeschlossene Anzahl von Elementen, welche der Potenz *m* der Primzahl *2* entsprechen. Die Konstante *l* gibt die vom RS-Coder erzeugte Anzahl Symbole pro Block an, die Konstante *n* stellt die Anzahl der Informations-Symbole dar. Die Anzahl der Korrektur-Symbole berechnet sich aus der Differenz *k=l-n*, wobei hier vorausgesetzt wird, daß *k* geradzahlig ist. Unter dieser Voraussetzung ist f_k entsprechend For-

$$f_k = (l-n)/2 \tag{1}$$

mel (1) die maximale Anzahl korrigierbarer Fehler (Fehlerkorrekturgrad).

Ein *GF* ist eine algebraische Struktur, in welcher mathematische Operationen wie die Addition und Multiplikation möglich sind. Die Symbole α eines *GF* können durch ein primitives Modularpolynom *m(x)* erzeugt werden, sie werden üblicherweise als Potenzen von α angegeben. Im binärem Fall werden die Symbole durch *m* Bits repräsentiert. Die Formel (2) beschreibt eine Vorschrift für die RS-Codierung, wobei *c(x)* das Codewort-Polynom vom Grad *l-1*, *i(x)* das Informations-Polynom vom Grad *n-1* und *g(x)* das Generator-Polynom vom Grad *l-n* ist.

$$c(x) = i(x)x^{l-n} + [i(x)x^{l-n}]mod \cdot g(x) \tag{2}$$

Die Formel (3) beschreibt einen Ansatz für die Erzeugung des Generator-Polynoms *g(x)*.

$$g(x) = (x+\alpha)(x+\alpha^2)\ldots(x+\alpha^k) \tag{3}$$

Voraussetzung für die Decodierung ist, daß die Nullstellen von *g(x)* eine Folge (α^i, α^{i+1}, ... , α^{i+k-1}) bilden. Die Codierung und Decodierung kann durch das Verfahren der Polynom-Division entsprechend der Formel (2) erfolgen. Die Fehlerkorrektur des empfangenen Codewortes erfolgt sequentiell entsprechend der in Abbildung 2 angegebenen Schritte. Sie erfolgt in abgeschlossenen HW-Funktionseinheiten, sogenannten RS-Modulen. Weitere Ausführungen zu RS-Codes sind z.B. in [11] enthalten.

2.2 Parameterisierung

RS-Codes sind vor allem in ihren mathematischen Parametern variabel. Der Grad des ihnen zu Grunde gelegten *m(x)* berechnet das *GF* und bestimmt damit die Bitbreite des RS-Codes. Ein weiterer Parameter, welcher entsprechend der Formel 2 über die Leistungsfähigkeit des RS-Codes entscheidet, ist f_k. Beide Parameter wirken sich vor allem auf den internen Aufbau der in jedem RS-Modul mehrfach vorhandenen *GF*-Grundoperationseinheiten (*GF-Addition, GF-Multiplikation, GF-Inverter, GF-Potenzierer*), auf die Häufigkeit ihrer Verwendung und damit auf den benötigten Flächenverbrauch aus. Tabelle 1 stellt eine Übersicht über die Verteilung der *GF*-Grundoperatoren in den in Abbildung 2 dargestellten Teilalgorithmen entsprechend unterschiedlicher Fehlerkorrekturgrade dar. Die Implementation fächenoptimierter *GF*-Multiplizierer-Module wurde bereits in [10] untersucht.

2.3 Modularisierung

Neben der aus Abschnitt 2.2 resultierenden Partitionierung der RS-Module in *GF*-Grundoperationseinheiten und weiterer notwendiger Logik zur Realisierung des Datenpfades für die RS-Codierung und -Decodierung soll in diesem Abschnitt die Partitionierung auf RS-Modul-Niveau diskutiert werden. Die Verwendung von RS-Codes ermöglicht große Freiheitsgrade hinsichtlich der Partitionierbarkeit in voneinander unabhängige RS-Module. Diese erlauben bez. der dynamischen Rekonfiguration vielfältige Rekonfigurationsabläufe

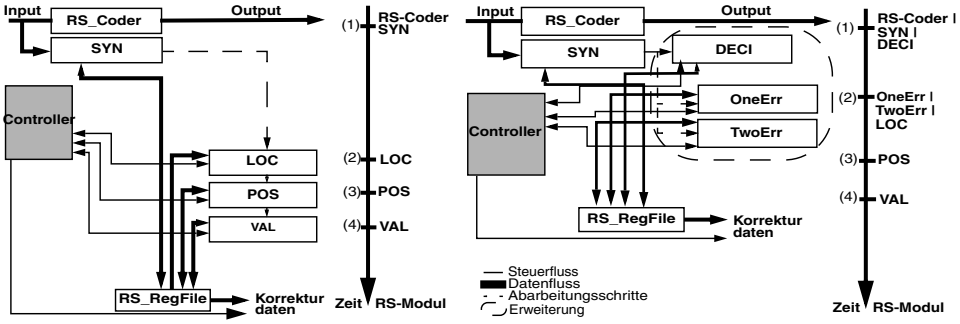
[6]. Die Abbildungen 2, 4 und 3 zeigen auf den am rechten Rand dargestellten Zeitleisten ebenfalls verschiedene Varianten dynamischer Rekonfigurationsabläufe der RS-Module.

	Fehlerkorrekturgrad (f_k)							
GF-Module	2	3	4	5	6	7	8	16
$Coder/SYN_{Add/Mul}$	4	6	8	10	12	14	16	32
LOC_{Add}	5	7	9	11	13	15	17	32
LOC_{Mul}	9	13	17	21	25	29	33	65
LOC_{Inv}	4	6	8	10	12	14	16	32
POS_{Add}	2	3	4	5	6	7	8	16
POS_{Mul}	4	6	8	10	12	14	16	32
VAL_{Add}	6	9	12	15	18	21	24	48
VAL_{Mul}	8	12	16	20	24	28	32	64
$VAL_{Inv/Pot}$	2	3	4	5	6	7	8	16

Tab. 1: Verteilung der GF-Operationseinheiten

2.3.1 RS-Fehlerkorrektur

In [6] wurde ein in Abbildung 2 dargestellter Lösungsansatz diskutiert, welcher bei selten auftretenden Fehlern (nachfolgend als geringe BER *(Bit Error Rate)* bezeichnet) dennoch eine Fehlerkorrektur bei beschränkten HW-Resourchen ermöglicht, indem die RS-Module sequentiell auf dem zur Verfügung stehenden Platz innerhalb der programmierbaren Logik entsprechend der dargestellten Zeitachse rekonfiguriert werden. Dieser Ansatz erlaubt es, auftretende Störungen - feststellbar durch das RS-Modul *SYN* - durch partiell/dynamische Rekonfiguration der RS-Module *LOC*, *POS* und *VAL* schrittweise zu korrigieren.



a) Standard-Fehlerkorrekturablauf b) erweiterter Fehlerkorrekturablauf

Abb. 2: RS-Codec-Blockschaltbilder mit Standard- und erweitertem Fehlerkorrekturablauf

2.3.2 Optimierung der RS-Fehlerkorrektur

Eine Optimierung des in Abschnitt 2.3.1 beschriebenen Ansatzes beruht auf Arbeiten in [9] und wurde in [5], [10] realisiert. Ziel des in Abbildung 4 dargestellten Ansatzes war es, eine Optimierung der Fehlerkorrekturalgorithmen für eine geringe Anzahl Fehler (< 2 Fehler) zu erzielen, welche in eine höhere Korrekturgeschwindigkeit und einen geringeren Flächenbedarf resultiert. Dazu wurden weitere RS-Module entwickelt, welche eine Einfehler- oder Zweifehlererkennung *(DECI)* bzw. eine Einfehler- oder Zweifehlerkorrektur *(OneErr, TwoErr)* durchführen. Steigt die *BER* an, dann kann auf den in Abschnitt 2.3.1 beschriebenen Ansatz zurückgegriffen werden und es können alle RS-Module *LOC*, *POS* und *VAL* wieder konfiguriert werden.

2.3.3 Codeverkettungen

Um einer sehr hohen *BER* entgegenwirken zu können, ist der Einsatz von verketteten Kanalcodes möglich. Diese Codes bestehen aus einem „äußeren“ und aus einem „inneren“ Code. Beispiele solcher Codes sind sowohl bei Audio-CDs (äußerer Code: RS(28,24) über $GF(2^8)$, innerer Code RS(32,28) über $GF(2^8)$) als auch bei Digital Video Broadcasting *(DVB)* mit einem äußeren Code von RS(204,188) über $GF(2^8)$ und einem inneren Code mit VITERBI-Algorithmus zu finden [3]. Potential für partiell/dynamische Rekonfigurationsabläufe ist hier sowohl im Coder als auch im Decoder vorhanden. Hierbei wäre z.B. Coderseitig zuerst die Berechnung des äußeren Codes möglich, danach findet nach Rekonfiguration die Abarbeitung des inneren Codes statt. Code-Module, welche durch ihren f_k unterschiedlich leistungsfähig sind, können bei Bedarf entsprechend der in Abbildung 3 dargestellten Zeitachsen adaptiv gegeneinander ausgetauscht werden, wodurch je nach *BER* die Verwendung leistungsfähigerer Codes möglich wird. Eine Kombination mit dem in Abschnitt 2.3.1 und Abschnitt 2.3.2 beschriebenen Lösungsansatz ist möglich.

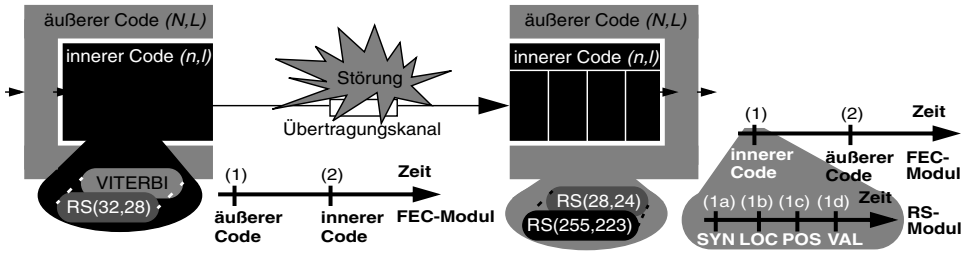


Abb. 3: sequentielle Codeverkettung und zugehörige Rekonfigurationsabläufe

3 Wiederverwendungsgerechte Codegenerierung

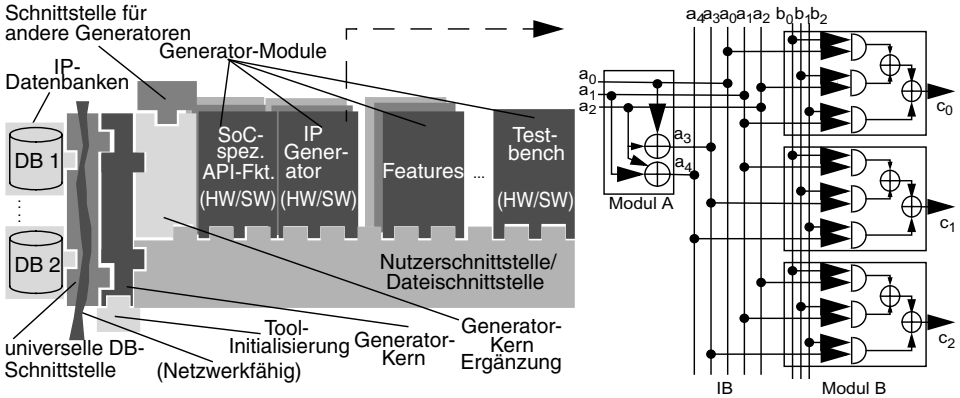
Zur Beschleunigung des Entwurfes der stark parameterisierbaren und modularisierbaren FEC-Applikationen soll als Lösungsansatz ein Werkzeug vorgestellt werden, welches eine Kombination aus komfortabler IP-Verwaltung und Generierung auf Soft-IP-Niveau unter Beachtung der SoC- *(System-on-Chip)* Architektur und deren Rekonfigurations-Möglichkeiten vorsieht. Die durch das Werkzeug erzeugten Soft-IPs können entsprechend dem Abschnitt 4 in unterschiedliche Entwurfsabläufe übernommen und von den zugehörigen Werkzeugen verarbeitet werden. Das Werkzeug trägt dadurch zur Entwurfsbeschleunigung bei, daß es ausgewählte Aufgaben wie z.B. die Bereitstellung von Schaltungs-Modulen, die Schnittstellengenerierung und die Systempartitionierung für dynamische Rekonfigurationsabläufe übernimmt, wodurch der manuelle Codierungsaufwand minimiert wird.

3.1 Codegenerator Architekturansatz

Die Abbildung 4a stellt für das entwickelte Generatorwerkzeug den modularen Ansatz vor und gibt einen Überblick über die Funktionsweise ausgewählter Generator-Module. Der Ansatz erlaubt es, sogenannte Generator-Module an das Werkzeug zu adaptieren, durch welche der Umfang der generierbaren Applikations-Module erweitert werden kann. Eine gezielte Beeinflussung der Parameterisierung und Modularisierung des Entwurfes kann durch Konfigurationsdateien und Benutzerschnittstelle erfolgen. Nachfolgend werden ausgewählte Generator-Module beschrieben, welche zur Entwurfsverbesserung sowohl in der Bereitstellung der RS-Module als auch für die SoC-Implementation beitragen.

3.1.1 Generierung von GF-Operationseinheiten

Der interne Aufbau der GF-Einheiten hängt vom verwendeten *m(x)* ab und kann entsprechend [10] bez. der SoC-Umsetzung stark variieren. Aus diesem Grund ist die Entwicklung von Generator-Modulen erforderlich, welche eine gezielte Generierung der *GF*-Einheiten entsprechend der unterschiedlich verwendbaren Code-Parameter ermöglichen.



a) Generator-Architektur b) Generierter GF-Multiplizierer

Abb. 4: SoC-Generatorwerkzeuges und GF-Multiplizierer nach E.D. Mastrovito [8]

Für die Generierung von *GF-MUL*-Operationseinheiten wurden die in [8] und [12] beschriebenen Verfahren umgesetzt. Diese ermöglichen eine bitparallele Multiplikation unter Verwendung rein kombinatorischer Logik, welche pro Takt ein Ergebnis liefern.

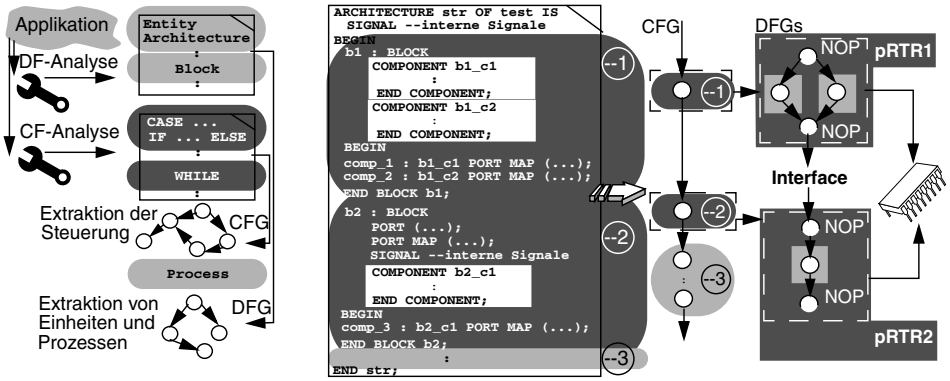
Generiert man in Abhängigkeit von *m(x)* Einheiten, dessen Grundmodule (Modul A, Modul B) über ein Netzwerk „IB *(Interconnection Bus)*“ verbunden sind, so erhält man z.B. für *GF(2)* mit $M(x)=x^3+x+1$ einen GF-Multiplizierer entsprechend der Abbildung 4b, welcher die GF-Elemente $A(x) = a_0 + a_1x + a_2x^2$ und $B(x) = b_0 + b_1x + b_2x^2$ miteinander multipliziert. Die gezielte Generierung dieser *GF*-Operationseinheit wirkte sich dahingehend positiv aus, daß schon bei einem 3-Bit RS-Coder eine Flächenersparnis von 50% und die Verdopplung des Arbeitstaktes gegenüber alternativen Ansätzen erreicht wurde [10].

3.1.2 Generatorbasierte Partitionierung

Der in diesem Abschnitt beschriebene Ansatz dient zur werkzeugbasierten Modul-Partitionierung für pRTR- *(partial Run Time Reconfiguration)* Systeme und ist in Generator-Modulen umsetzbar. Hilfsmittel dieses Ansatzes zur Erzeugung von pRTR-Modulen stellen Sequenzgraphen dar, welche einer Hierachie gerichteter Graphen entsprechen [13]. Diese Graphen (CFG/DFG: *control/data flow graph*) besitzen zusätzliche Knoten (Startknoten/Endknoten: *NOP*, Modulaufruf: *CALL*, Verzweigung: *BR*, Iteration: *LOOP*). Mit Hilfe eines parserbasierten Ansatzes, lassen sich diese Sequenzgraphen auf Grundlage der HDL- *(Hardware Description Language)* Beschreibung erzeugen. Der Parser ist zur Analyse der Systembeschreibungen nach verschieden komplexen Strukturen verantwortlich:

- grobe Granularität *(z.B. benötigt für die Rekonfiguration mehrerer/einer Applikation)*
  - Analyse nach Top Entity/Architecture Statements mehrerer Applikationen,
  - Analyse nach Entity/Architecture, Statements/Block Statements einer Applikation.

- mittlere Granularität *(z.B. benötigt für die Rekonfiguration einer Applikation)*
  - Analyse nach Prozeß-Statements mehrerer/einer Applikation - siehe Abbildung 5a.
- feine Granularität *(z.B. benötigt für die Rekonfiguration der Koeffizienten eines Filters)*
  - Prozeß-Analyse auf definierte Basisoperationen.



a) mittlere Granularität b) Analyse und Selektion von pRTR-Modulen

Abb. 5: Analysevorgang für die Partitionierung von pRTR-Modulen

Nach der Analyse kann eine Umstrukturierung der HDL-Quellen auf Basis der internen Struktur der Sequenzgraphen bez. vorgegebener Rekonfigurations-Constraints erfolgen. Dazu werden die durch die Analyse selektierten HDL-Statements entsprechend der Abbildung 5b den DFG/CFG-Knoten zugeordnet. Durch die Verwendung der zusätzlichen Knoten lassen sich Steuerungs- und Datenflüsse detaillierter gestalten. So besteht die Möglichkeit durch die Knoten *CALL* und *BR* innerhalb des CFG einen Aufruf von pRTR-Modulen zu initiieren. Innerhalb des DFG ist es notwendig, an den Start- und Endknoten *(NOP)* des Sequenzgraphen Zwischenspeicher oder Schnittstellen-Einheiten einzufügen.

3.1.3 Generierung von SoC-Interface-Modulen

Weitere Entwurfsunterstützung können Generator-Module durch die Generierung von Interface-Modulen bereitstellen. Interface-Module sind notwendig, damit pRTR-Module entsprechend der Abbildung 7 über festgelegte Schnittstellen miteinander kommunizieren können Ansätze dieser Interface-Module wurden für programmierbare SoC bereits in [6] und [7] entwickelt. Die für diese Arbeit verwendete Xilinx-Virtex-FPGA-Architektur benötigt Interface-Module, welchen „Bus Macros“ als Grundlage dienen [1]. Je nach Applikation müssen diese Interface-Module entsprechend angepaßt werden. Zu diesen Anpassungen zählen u.a. die Breite des Kommunikationsinterfaces, Signalflußrichtung und die Platzierung innerhalb des gesamten Entwurfes. Eine Bereitstellung universeller wiederverwendbarer Schnittstellenmodule mit Standard-Bitbreiten von z.B. 4, 8, 16 oder 32 Bit und Zusatzlogik zur Programmierung von Signalflußrichtung ist ebenfalls möglich.

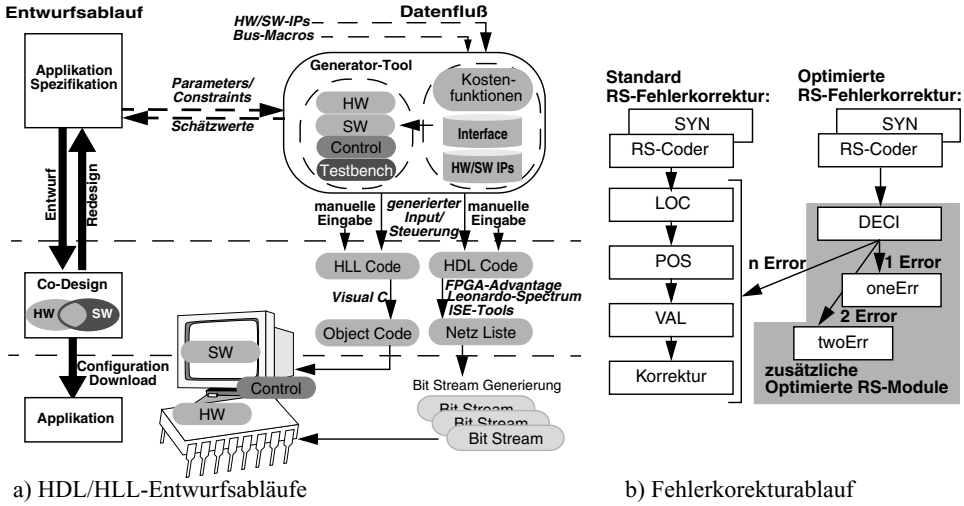
4 Erweiterter Entwurfsablauf

Der verwendete erweiterte Entwurfsablauf ist in vereinfachter Form in Abbildung 6a dargestellt. Es wird sowohl ein HDL-Entwurfsablauf für den Entwurf der HW incl. der pRTR-Module als auch ein HLL-Entwurfsablauf *(High Level Language)* für die Entwicklung der PC-Kommunikation incl. der Rekonfigurationsablaufsteuerung benötigt.

Das am Anfang des erweiterten Entwurfsablaufes angebundene Generatorwerkzeug stellt auf Grundlage von Nutzereingaben Schaltungs-Komponenten und Interface-Module spezifikationsgerecht bereit und unterstützt den Entwerfer bei der Partitionierung des Entwurfes für partiell/dynamische Rekonfigurationsabläufe. Neben weiteren, vom Entwerfer manuell entworfenen Schaltungs-Modulen, dient die Ausgabe des Generatorwerkzeuges als Eingabe zur Verarbeitung in kommerziellen Entwurfswerkzeugen. Synthese, Platzierung, Verdrahtung und die Compilierung der SW erfolgten durch den Einsatz dieser Werkzeuge.

Die für den HDL-Entwurfsablauf benutzten Werkzeuge sind für den HW-Entwurf der „Mentor-HDL-Designer“, für die Synthese „FPGA-Advantage incl. Leonardo Spectrum“ oder alternativ der „Synopsys-FPGA Compiler II“. Die Initialzuweisung der Ressourcen, das Floorplanning mit Restriktionen des MDF, das Mapping und Routing der pRTR-Module und des gesamten Entwurfes und die Erzeugung der Bitstreams erfolgt unter Verwendung der Xilinx-ISE Werkzeuge [1].

Der HLL-Entwurfsablauf verwendet den Visual C Compiler und Programm-Bibliotheken des Board-Herstellers, welche API-Funktionen für Kommunikation und zur Steuerung der Rekonfigurationsabläufe zur Verfügung stellen. Diese können vom Entwerfer in eigenen C-Programmen eingesetzt werden. Ergänzungen bez. der API-Funktionen wurden dahingehend vorgenommen, das zusätzliche Funktionen zur Steuerung der partiell/dynamischen Rekonfiguration implementiert wurden. .



a) HDL/HLL-Entwurfsabläufe b) Fehlerkorekturablauf

Abb. 6: Erweiterung der Entwurfsabläufe und RS-Fehlerkorrektur mit pRTR-Funktionalität

Das eingesetzte Prototyping-Board ist ein ADM-XRC-Board der Firma Alpha-Data [15]. Es besitzt einen Xilinx-Virtex 1000 FPGA. Das ADM-XRC-Board ist mit dem Host-PC über eine zusätzliche PCI-Adapter-Karte verbunden. Die Kommunikation ist über API-*(Application Programming Interface)* C-Funktionen frei programmierbar. Weitere technische Details des Prototyping-Boards sind in [15] beschrieben.

5 Ergebnisse

Der RS-Codec soll entsprechend der folgenden Randbedingungen abgearbeitet werden:

- die *BER* des Übertragungskanals ist sehr gering,
- treten Fehler auf, so treten sehr selten mehr als zwei Fehler auf,
- „Standard-Konfiguration“ sind die in Abbildung 6b rechts dargestellten Algorithmen zur Fehlerkorrektur, welche eine Fehlerkorrektur von bis zu zwei Fehlern ermöglichen
- bei mehr als zwei Fehlern werden die in Abbildung 6b links dargestellten Standard-Fehlerkorrekturalgorithmen verwendet.

Die Bereitstellung der Blockstruktur der in Abbildung 6b dargestellten RS-Module incl. der *GF*-MUL Operationseinheit und der Interface-Module erfolgte für unterschiedliche *GF* durch das Generatorwerkzeug und lieferte damit den Input für den in Abschnitt 4 beschriebenen Entwurfsablauf. Die Ergebnisse der Untersuchungen in Bezug auf die Flächenausla-

	RS-Coder					
Module	2^4 (f_k=*2*)	rt (*ms*)	2^6 (f_k=*8*)	rt (*ms*)	2^8(f_k=*8*)	rt (*ms*)
$Coder_{Fläche\ (Slices)}$	119		344		398	
$Coder_{Bitstream\ (kByte)}$	87	2,18	190	4,75	169	4,23
$SYN_{Fläche\ (Slices)}$	85		104		113	
$SYN_{Bitstream\ (kByte)}$	85	2,13	98	2,45	95	2,38
$LOC_{Fläche\ (Slices)}$	186		1156		1801	
$LOC_{Bitstream\ (kByte)}$	92	2,30	298	7,45	313	7,83
$POS_{Fläche\ (Slices)}$	119		755		822	
$POS_{Bitstream\ (kByte)}$	92	2,30	270	6,75	253	6,33
$VAL_{Fläche\ (Slices)}$	132		476		1186	
$VAL_{Bitstream\ (kByte)}$	93	2,33	223	5,58	239	5,98
$DECI_{Fläche\ (Slices)}$	127		1961		2601	
$DECI_{Bitstream\ (kByte)}$	91	2,28	296	7,40	310	7,75
$OneErr_{Fläche\ (Slices)}$	86		119		168	
$OneErr_{Bitstream\ (kByte)}$	87	2,18	106	2,65	145	3,63
$TwoErr_{Fläche\ (Slices)}$	141		295		410	
$TwoErr_{Bitstream\ (kByte)}$	92	2,30	191	4,78	198	4,95

Tab. 2: Untersuchungsergebnisse der RS-Module

stung, Größe der Bitstreams und deren Rekonfigurationszeit (*rt*) für RS-Coder mit unterschiedlichem *GF* sind in Tabelle 2 gegenübergestellt. Die vollständige Rekonfiguration des Xilinx-Virtex 1000 FPGA würde im Vergleich zur Rekonfigurationszeit der partiellen Bitstreams rund 18,75 ms betragen. Die untersuchten RS-Coder hatten bei einer 3 Bit Realisierung f_k=*2* und für 8 Bit f_k=*8.* Der Flächenbedarf von RS-Modulen, in welchen als *GF*-Operationseinheiten eine hohe Anzahl von *GF*-INV bzw. *GF*-POT (*VAL*) enthalten sind, steigt bei höheren Bitbreiten sehr stark an. Das liegt in der Tatsache begründet, daß diese noch mit Hilfe von LUTs *(Look-Up Table)* realisiert sind. Hier ist die Entwicklung weiterer Generator-Module von Vorteil, welche *GF*-INV und *GF*-POT entsprechend der Nutzerparameter als kombinatorische Schaltung generieren.

Die Berechnung der Rekonfigurationskosten erfolgt unter Verwendung der Formel (4). Die Länge des Bitstreams und die Geschwindigkeit der Programmierung eines partiellen Bitstreams sind direkt proportional zueinander [1]. Die in der Formel (4) angegebenen Größen besitzen folgende Bedeutung:

- t_{reconf_part}: Rekonfigurationszeit eines partiellen Bitstreams,
- $perf_{reconf\_inf}$: Geschwindigkeit der Rekonfigurationsschnittstelle (PCI-Bus=40MHZ),
- bs_{reconf_part}: Größe der Rekonfigurationsdaten des partiellen Bitstreams in Byte,
- bs_{FPGA}: Größe der Rekonfigurationsdaten des gesamten Bitstreams in Byte.

$$t_{reconf\_part} = \frac{\left(\frac{bs_{FPGA}}{perf_{reconf\_inf}}\right) \cdot bs_{reconf\_part}}{bs_{FPGA}} \tag{4}$$

Die System-Geschwindigkeit $perf_{reconf\_HW}$ berechnet sich unter Einbeziehung der Rekonfigurationsabläufe t_{reconf_part}, unter Berücksichtigung der Modullaufzeit $perf_{HW}$ und der Modul-Umschaltzeit (Zeit für die Auswahl eines neuen pRTR-Moduls durch die Steuerung bzw. den Host-PC) entsprechend Formel (5).

$$perf_{reconf\_HW} = \sum_{i=1}^{n} (t_{reconf\_part} \cdot perf_{HW} \cdot t_{schedul}) \tag{5}$$

6 Zusammenfassung und Ausblick

Untersuchungsgegenstand dieser Arbeiten waren u.a. die Untersuchung von Algorithmen aus dem Bereich der Kanalcodierung für partiell/dynamische Rekonfigurationsabläufe. Es wurde ein Generator-Ansatz beschrieben, welcher eine Entwurfsunterstützung durch Modul- und Interface-Generierung und eine Einbeziehung partiell/dynamischer Rekonfigurationsabläufe ermöglicht. Der Beitrag stellt für mehrere RS-Module Untersuchungsergebnisse bez. Rekonfigurationsgröße- und Geschwindigkeit der Bitstreams vor. Zukünftige Arbeiten werden sich mit der Weiterentwicklung des Generatorwerkzeuges beschäftigen.

Literatur

[1] D. Lim, M. Peattie: „Two Flows for Partial Reconfiguration: Module Based or Small Bit Manipulation“, Application Note XAPP290 (v1.0), May, 2002
[2] U. Tangen: „Selbstorganisation und Evolution elektronischer Hardware“, GMD Spiegel, 4.2000
[3] H. Klimant, R. Piotraschke, D. Schönfeld: „Informations- und Kodierungstheorie“, Teubner Verlag, 2. Auflage, 2003
[4] J. Schneider: „Forward Error Correction (FEC) in ATM-Netzwerken“, Diplom, TUD 1998
[5] J. Schneider, V. Kotzsch, M. Koegst: „Algorithmen-Optimierung und HW-Prototypingvon Fehlerkorrektur-Verfahren am Beispiel von Reed-Solomon Kodes“, DASS'03, Dresden, Mai, 2003
[6] A. Haase, J. Schneider u.a.: „Design of a Reed Solomon decoder using partial dynamic reconfiguration of Xilinx Virtex FPGAs-a case study“, DATE'02, Paris, France, 4-8 March, 2002
[7] J. Schneider, M. Boden, M. Koegst, S. Rülke: „Eine wiederverwendungsgerechte Entwurfsmethodik für rekonfigurierbare SoC-Architekturen“, GI-Workshop, Tübingen, 2002
[8] E. D. Mastrovito: „VLSI Architectures for Compuations in Galois Fields“, PhD Thesis, Linköping University, Schweden, Februar 1991
[9] M. Koegst, St. Rülke: „Conditions for the Number of Errors in a Reed-Solomon Codec“, 3th int. symposium on mobile multimedia systems and applications, Netherlands, 2002
[10] J. Schneider, V. Kotzsch, M. u.a.: „Wiederverwendungsgerechte HW-Modellierung und Optimierung von Fehlerkorrektur-Schaltungen am Beispiel eines Reed-Solomon Koders/Dekoders“, Report: SFB 358-B1-1/03, 2003
[11] W. W. Peterson: „Error-correcting codes“, MIT Press, 1994
[12] C. Paar: „Effiziente VLSI-Architekturen für bit-parallele Arithmetik in endlichen Körpern“, Diss., Uni Essen, 1994
[13] J.Teich: „Digitale HW/SW-Systeme“, Springer-Verlag,1997
[14] Xilinx: „http://www.xilinx.com“
[15] Alpha Data: „http://www.alpha-data.com“