Universität Karlsruhe (TH)
Forschungsuniversität • gegründet 1825

Institut für Technische Informatik

Seminararbeit

Rekonfigurierbare Eingebettete Systeme - Bitstream Manipulation

von
Timo Sandmann

Universität Karlsruhe (TH)
Fakultät für Informatik
Institut für Technische Informatik (ITEC)
Chair for Embedded Systems (CES)
Prof. Dr. Jörg Henkel

(Erste Version)

Betreuer:	Prof. Dr. Jörg Henkel
Betreuender Mitarbeiter:	Lars Bauer
Tag der Anmeldung:	(Datum)
Tag der Abgabe:	(Datum)

Inhaltsverzeichnis

1 Einleitung

Die vorliegende Arbeit beschäftigt sich mit der Manipulation von Konfigurationsbitstreams rekonfigurierbarer Hardware. Betrachtet wird ausschließlich die dynamische partielle Rekonfiguration, also der Austausch von Hardwaremodulen zur Laufzeit des Systems, so wie sie auf aktuellen FPGA-Architekturen möglich ist. Neben einer allgemeinen Vorstellung konkreter Architekturen und deren Bitstream-Formaten wird im Wesentlichen aufgezeigt, welche Techniken eingesetzt werden können, um die dynamische Rekonfiguration von Hardware sicherer und flexibler gestalten zu können.

1.1 Motivation

Moderne Field Programmable Gate Arrays (FPGAs) bieten die Möglichkeit, Teile ihrer Konfiguration zur Laufzeit zu verändern. So können beispielsweise Module, die aktuell nicht benötigt werden, gegen andere ausgetauscht werden, die für anstehende Aufgaben gebraucht werden. Dadurch müssen nicht zu jeder Zeit alle Hardwareteile gleichzeitig aktiv sein, sondern nur die auch wirklich Nötigen. Als Beispiel sei hier eine Software Defined Radio Implementierung genannt, die durch die dynamische partielle Rekonfiguration erheblich vereinfacht werden kann. So ist es möglich, verschiedene Filtermodule je nach Bedarf zu laden und zu verwenden. Anschließend kann derselbe Konfigurationsspeicher für andere Module verwendet werden, auf weitere Details zu diesem Beispiel wird in Kapitel 5 näher eingegangen. Der erzielte Vorteil ist vor allem eine bessere Flächenauslastung des FPGAs. Dem steht allerdings auch der Nachteil entgegen, dass ein defekter Konfigurationsbitstream oder eine falsche Positionierung der Konfiguration das gesamte System unbrauchbar machen könnten. Aus diesem Grund sind Techniken zur Validierung des Konfigurationsupdates wünschenswert und betreffen somit direkt die Verarbeitung des Bitstreams, der die zu aktualisierende Hardware beschreibt. Neben der auf diese Weise zu gewinnenden Sicherheit sind aber auch Platzausnutzung und effiziente Kommunikation zwischen unterschiedlichen Modulen von entscheidender Bedeutung.

1.2 Ziele

Die dynamische partielle Rekonfiguration sollte im Idealfall drei Dinge erreichen: Erstens eine hohe Sicherheit, dass die aktualisierte Hardware gemäß ihrer Spezifikation korrekt funktioniert und dass eine Beeinflussung anderer Hardwareteile selbst dann ausgeschlossen ist, wenn der Updateprozess fehlschlug. Zum Beispiel im Automobilbereich erscheint eine Anwendung sonst ausgeschlossen. Zweitens ist eine möglichst optimale Ausnutzung der verfügbaren Fläche des FPGAs gewünscht. So geht mit der Minimierung von interner und externer Fragmentierung bei den entsprechenden Modulen eine Verringerung der benötigten

Größe des FPGAs und somit eine Ersparnis an Kosten und Energiebedarf einher. Drittens ist eine flexible und leistungsfähige Kommunikation zwischen den einzelnen Modulen des Gesamtsystems erforderlich, denn nur dann geht die Aufteilung in verschiedene, austauschbare Teile nicht zu Lasten der Performanz. Ein zu großer Kommunikationsoverhead würde den Einsatz in vielen zeitkritischen Anwendungsfällen verhindern.

1.3 Strukturierung

Das folgende Kapitel 2 beschäftigt sich zunächst mit der Hardware an sich. Es beschreibt den Aufbau verschiedener FPGA-Architekturen mit ihren Unterschieden, das Format des Konfigurationsbitstreams für jene und den zugehörigen Rekonfigurationsprozess im Detail. In Kapitel 3 werden verschiedenen Varianten zur Kommunikation der austauschbaren Hardwaremodule untereinander vorgestellt. Einerseits ist eine Minimierung des Platzbedarfs für Kommunikationskanäle gewünscht, andererseits aber eine Maximierung der Performanz und Flexibilität von Interesse. Kapitel 4 beschreibt die Möglichkeiten, ein dynamisch rekonfigurierbares System mit der nötigen Sicherheit gegen Ausfall oder Fehlfunktion auszustatten. Hier geht es im Wesentlichen um eine Integritätsprüfung des zu ladenden Moduls sowie den Schutz aller nicht zu aktualisierenden Teile gegen jegliche Veränderung. In Kapitel 5 wird näher auf den dynamisch zu konfigurierenden Bereich des FPGAs eingegangen. Im Vordergrund stehen hier die verschiedenen Möglichkeiten zur Einteilung des Rekonfigurationsbereichs, so dass eine möglichst hohe Flexibilität beim Moduleinsatz und eine gute Ausnutzung der Fläche auf dem FPGA gewährleistet werden können. Das 6. und letzte Kapitel fasst die vorgestellten Möglichkeiten zusammen und schließt mit einer abschließenden Bewertung.

Alle Abbildungen sind nur vorläufig und bisher lediglich Kopien aus dem zugehörigen Paper!

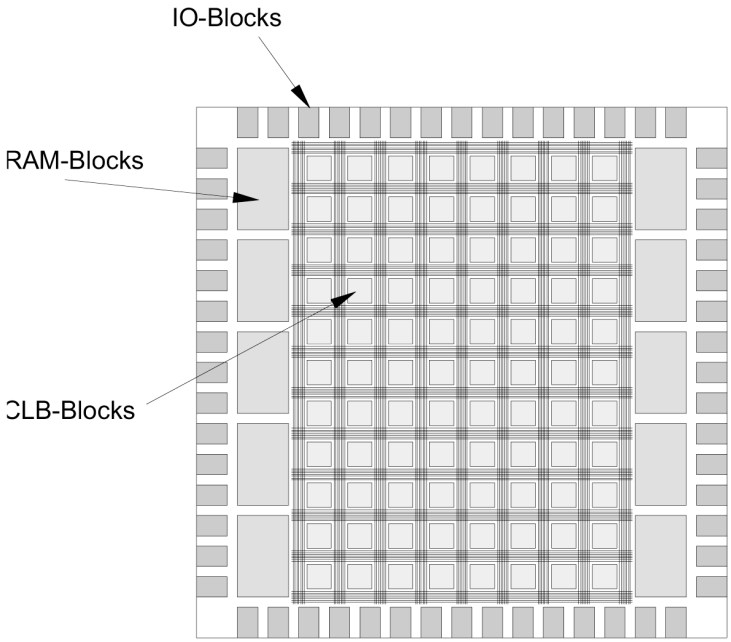
2 Hardware-Plattform

Dieses Kapitel beschreibt den Aufbau der FPGA-Architekturen Virtex II und Virtex 4 inklusive des für den Rekonfigurationsprozess erforderlichen Bitstreamformats und den Rekonfigurationsvorgang selbst. Aufgrund des verschiedenen Aufbaus von Virtex II und Virtex 4 FPGAs sind beide Architekturen getrennt beschrieben. Die hier vorgestellten Details dienen als Grundlage für die nachfolgenden Kapitel.

2.1 Aufbau verschiedener FPGAs

2.1.1 Virtex II (Pro)

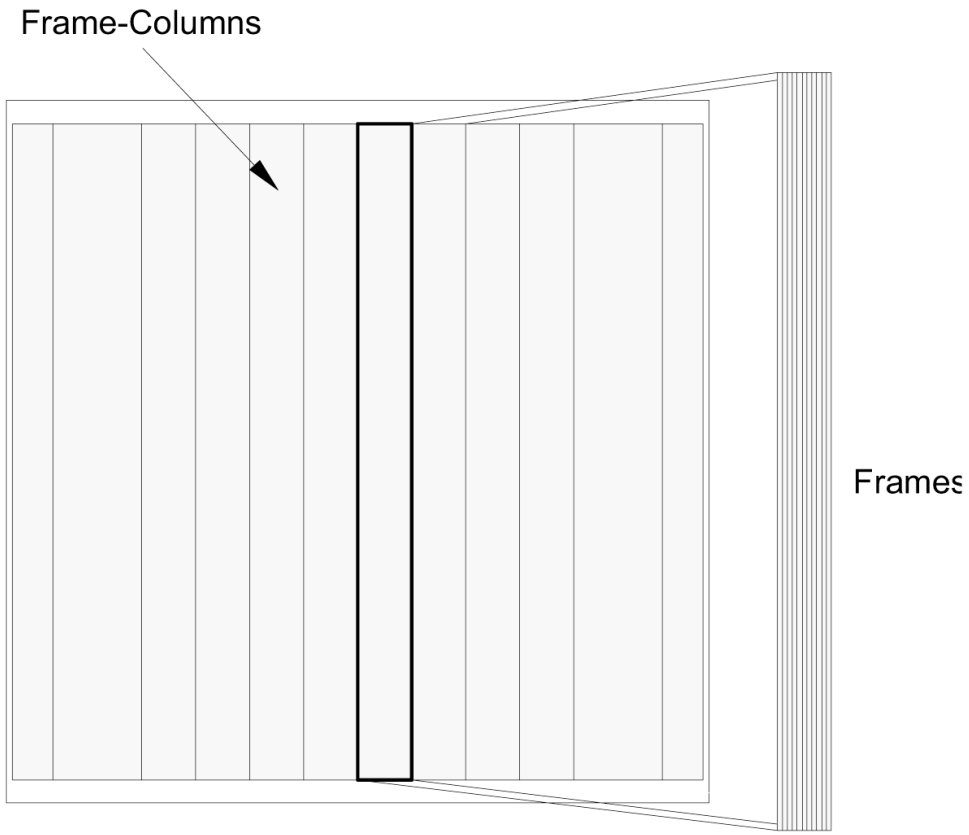
Ein Virtex II FPGA besteht im Wesentlichen aus zwei Schichten. Die Erste enthält die rekonfigurierbare Hardware an sich, die aus Logik-Blöcken („Cofigurable Logic Blocks“, kurz CLB), RAM- und I/O-Blöcken zusammengesetzt ist. Letztere dienen zur Kommunikation mit der Außenwelt. Des Weiteren gibt es noch Ressourcen zur Verbindung der einzelnen Blöcke untereinander. Alle aufgeführten Ressourcen sind in Abbildung 2-1 dargestellt, ebenso ihre Anordnung auf dem Chip:



a) Configurable Hardware Layer

Abbildung 2-1
Konfigurierbare Hardware Schicht eines Virtex II FPGAs

Die zweite Schicht enthält die Konfiguration für die erste Schicht. Folglich ist sie genauso in Spalten organisiert wie die Hardware Schicht. Es gibt aber keine Aufteilung in Spalten mit Zeilen, sondern eine Strukturierung in sogenannte Frames. Ein Frame beschreibt dabei immer die gesamte Höhe des FPGAs, umfasst also alle Zeilen aus Schicht 1. Mehrere Frames enthalten dabei die Konfigurationsinformationen für eine Spalte. Je nach Spaltentyp (CLB, RAM, I/O) sind unterschiedlich viele Frames nötig und ein Frame ist immer 1 Bit breit. Ein Frame ist zugleich die kleinste rekonfigurierbare Einheit, daher ist es nur möglich, alle Zeilen einer Spalte zu konfigurieren. Abbildung 2-2 zeigt schematisch die Unterteilung der Konfigurationsschicht in einzelne Spalten und Frames.



b) SRAM- Configuration Memory Layer

Abbildung 2-2
Konfigurationsschicht eines Virtex II FPGAs

Die Konfigurationsschicht ist als SRAM implementiert, so dass die Informationen ausgetauscht werden können. Dieser Rekonfigurationsprozess ist Kapitel 2.3 beschrieben. Die Informationen zum Aufbau eines Virtex II FPGAs sind im Wesentlichen [Hübner 06a] entnommen.

2.1.2 Virtex 4

Der grundsätzliche Aufbau eines Virtex 4 FPGAs ist dem eines Virtex II recht ähnlich, so gibt es auch hier eine Schicht für die Hardwarefunktionalität und eine Weitere für die Konfiguration. Wie aus [Becker 07] ersichtlich, besteht ein ganz entscheidender Unterschied jedoch darin, dass sich ein Frame nicht über die komplette Höhe erstreckt, also nicht sämtliche, sondern nur genau 16 Zeilen umfasst. Dadurch ist es möglich, nur einen Teil des FPGAs zu rekonfigurieren, der sowohl in der Breite als auch in der Höhe kleiner ist, als das gesamte FPGA. Ein Virtex 4 FPGA ist logisch in zwei Hälften unterteilt, eine Obere und eine Untere. Die Adressierung der Zeilen erfolgt aufsteigend von der Mitte aus nach oben bzw. unten, je nachdem, um welche Hälfte es sich handelt. Ein weiterer Unterschied zur Virtex II Architektur zeichnet sich dadurch aus, dass die I/O-Blöcke nicht ringförmig am äußeren Rand angeordnet sind, sondern als vertikale Streifen an der linken und rechten Seite sowie in der Mitte. Grafisch ist das in Abbildung 2-3 dargestellt:

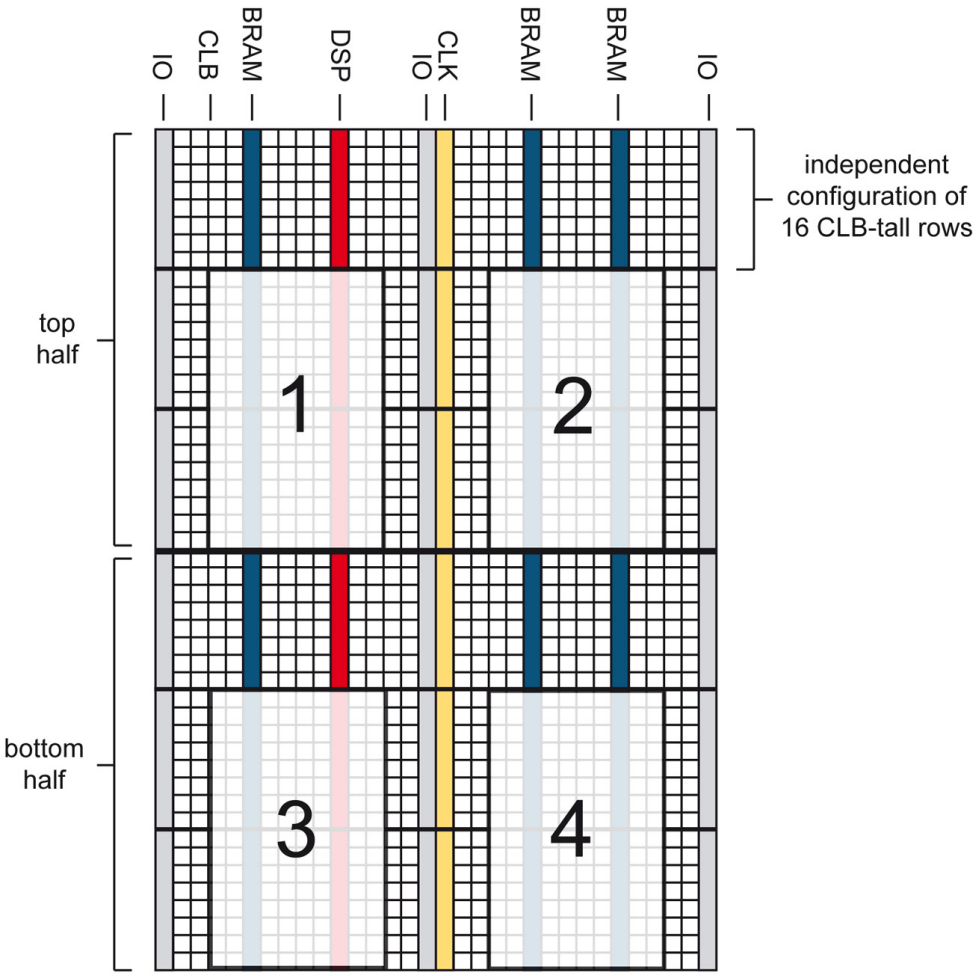


Abbildung 2-3
Aufbau eines Virtex 4 FPGAs

2.2 Bitstreamformat

2.2.1 Virtex II (Pro)

Der Konfigurationsbitstream beginnt immer mit einem Dummy-Wort, das nur dazu dient, die Puffer der Konfigurationslogik zu leeren. Das nächste Datenwort ist für die Synchronisation zuständig, mit dem die Konfigurationslogik beginnt, den Bitstream zu interpretieren. Anschließend folgen die eigentlichen Konfigurationsdaten. Mit dem Auftreten des abschließenden De-Synchronisations-Worts beendet die Logik die Rekonfiguration des FPGAs. Die eigentlichen Konfigurationsdaten (zwischen Synchronisations- und De-Synchronisations-Wort) sind in Paketen organisiert und jedes Paket beginnt mit einem Header. Je nach Anzahl der Datenwörter, die ein Paket umfasst, wird entweder ein Typ 1 oder Typ 2 Header verwendet. Im Detail sehen die Header wie in Abbildung 2-4 gezeigt aus:

Packet Header	Type	Operation (Write/Read)	Register Address	Byte Address	Word Count (32-bit data words)
Bits	[31:29]	[28:27]	[26:13]	[12:11]	[10:0]

Fig. 1. Type 1 packets

Packet Header	Type	Operation (Write/Read)	Word Count (32-bit data words)
Bits	[31:29]	[28:27]	[26:0]

Fig. 2. Type 2 packets

Abbildung 2-4
Mögliche Paket-Typen des Konfigurationsbitstreams

Je nachdem, welcher Art die zu konfigurierende Spalte des FPGAs ist, unterscheidet sich das Format der Konfigurationsbits, die auf den Header folgen. Für CLB- und BRAM-Frames zeigt die Abbildung 2-5 das Format, welches Virtex II FPGAs erwarten:

Table 1. CLB Frame Format

Pad/CLK	TopIOB	CLB X_aY_{max}		CLB X_aY_{max-1}		…	CLB X_aY_0		IOB bottom	Pad/CLK
16 bits	80 bits	Slice 40 bits	Slice 40 bits	Slice 40 bits	Slice 40 bits		Slice 40 bits	Slice 40 bits	80 bits	16 bits

Table 2. BRAM Frame Format

Pad/CLK	TopDCM	RAM X_aY_{max}	RAM X_aY_{max-1}	..	RAM X_aY_1	RAM X_aY_0	BottomDCM	Pad/CLk
16 bits	80 bits	320 bits	320 bits	..	320 bits	320 bits	80	16 bits

Abbildung 2-5
Format der CLB- und BRAM-Frames für Virtex II FPGAs

Da im Falle eines Virtex II FPGAs jeder Konfigurationsframe die komplette Höhe des FPGAs umfasst, besitzt jeder Frame des gleichen Typs eine konstante Größe, die nur von der maximalen Anzahl (Y_{max}) an Zeilen des jeweiligen FPGAs abhängt. Die Informationen zum Bitstreamformat für Virtex II FPGAs sind vor allem [Bok 07] und [Krasteva 06] entnommen.

2.2.2 Virtex 4

Da ein Konfigurationsframe bei Virtex 4 FPGAs nicht die komplette Höhe des FPGAs, sondern lediglich 16 (CLB-)Zeilen umfasst, ist hier auch ein anderes Adressierungsschema nötig. Ein Bit kennzeichnet jeweils, ob es sich um ein Frame für die obere oder die untere Hälfte des Chips handelt. Außerdem sind noch Spalten- und Zeilennummern angegeben. Jeder Frame umfasst 1312 Bits und je nach Typ zu dem die Konfiguration gehört (CLB, DSP, BRAM), sind unterschiedlich viele Frames nötig. Der Aufbau der Routing-Informationen ist jedoch in jedem Fall identisch, jeweils die ersten 20 Frames werden hierauf verwendet. Anschließend folgen noch zwei Frames zur CLB-Konfiguration, ein Frame zu DSP-Konfiguration oder 64 Frames zur BRAM-Konfiguration. Die folgende Abbildung 2-6 zeigt die Anordnung schematisch:

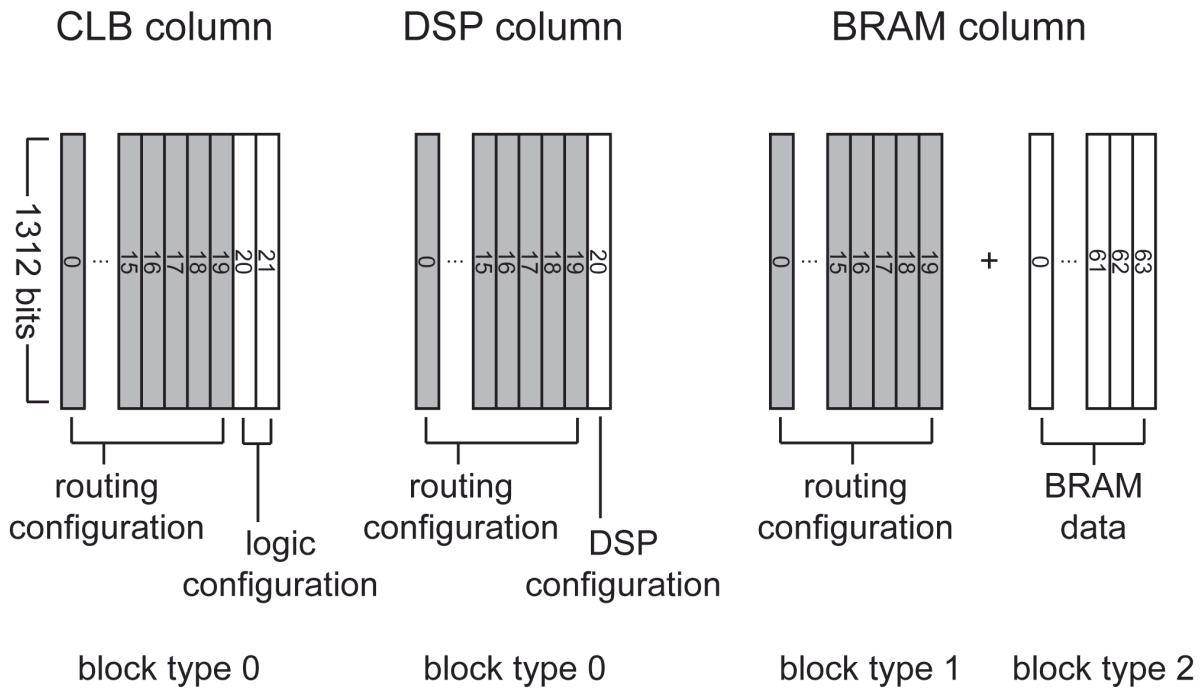


Abbildung 2-6
Format der CLB-, DSP- und BRAM-Frames für Virtex 4 FPGAs

Wie in [Becker 07] beschrieben ist, stehen sämtliche Konfigurationsbits für die untere Chiphälfte in rückwärtiger Ordnung (gegenüber der oberen Hälfte) im Bitstream. Dieselbe Quelle diente auch für die übrigen Informationen zum Bitstreamformat der Virtex 4 FPGAs.

2.3 Rekonfigurationsprozess

2.3.1 Kommandos

Der Rekonfigurationsprozess wird durch verschiedenen Kommandos gesteuert. Diese werden von der Konfigurationsschnittstelle „Internal Configuration Access Port“ (ICAP) ausgewertet. In [Bok 07] sind die folgenden Kommandos aufgeführt:

- RCRC Zum Zurücksetzen des CRC Registers
- SWITCH Zum Ändern der Clock Frequenz
- WCFG Zum Schreiben von Konfigurationsdaten
- RCFG Zum Lesen von Konfigurationsdaten
- SHUTDOWN Zum Einleiten der Shutdown-Sequenz
- START Zum Aktivieren der rekonfigurierten Hardware
- MFWR Zum Aktivieren des Multiple Frame Write Modus

Die zum ICAP gesendeten Kommandos sind insbesondere dann von Interesse, wenn überwacht werden soll, dass eine partielle Rekonfiguration nicht auf unerlaubte Bereiche schreibt.

2.3.2 Rekonfigurationslogik

Die ICAP-Schnittstelle eines Virtex II FPGAs besitzt hardwareseitig neben dem nötigen Taktsignal (clk) noch eine Aktivierungsleitung (ce), ein Schreib- / Lesesignal (write) und einen 8 Bit breiten Dateneingang (InData) für Schreibzugriffe sowie einen 8 Bit breiten Datenausgang (OutData) für Lesezugriffe und ein Busy-Signal (busy), für den Fall, dass der Puffer vollständig belegt ist.

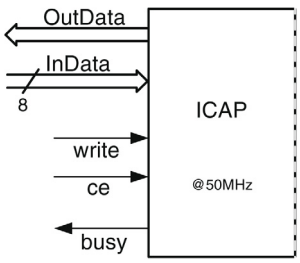


Abbildung 2-7
ICAP-Hardware eines Virtex II FPGAs

Die maximale Taktfrequenz beträgt 50 MHz, woraus sich eine maximale Datenrate von 8 Bit * 50 MHz = 400 MBit/s ergibt. Im Falle einer Virtex 4 Architektur sind beide Busse 32 Bit breit und die maximale Taktfrequenz beträgt 100 MHz. Hier liegt die obere Schranke für die Datenrate also bei 32 Bit * 100 MHz = 3,2 GBit/s.

Zur Kommunikation mit der ICAP-Schnittstelle stehen sechs Register zur Verfügung:

- CMD Das *Kommando Register*, in das auszuführende Kommandos geladen werden.
- FLR Das *Frame Address Register* enthält die Adresse des nächsten zu schreibenden Frames. Es wird automatisch inkrementiert.
- FLR Das *Frame Length Register* speichert die aktuelle Framegröße, die nicht nur für den eigentlichen Schreibzugriff erforderlich ist, sondern auch für die Auto-Inkrement-Funktion des Adressregisters.
- FDRI Beim *Frame Data Register Input* handelt es sich um eine Input-Pipeline-Stufe. Die hier eingehenden Daten werden in die Konfigurationsschicht des FPGAs geschrieben.
- MFWR Das *Multiple Frame Write Register* bietet die Möglichkeit, denselben Frame an verschiedene Speicheradressen zu schreiben. Der für unterschiedliche Orte gleichen Daten-Frame wird dazu hier gespeichert, während die gewünschten Adressen in das Frame Address Register geschrieben werden.
- CRC Das *CRC Register* enthält den erwarteten CRC-Wert, der dem Bitstream entnommen wurde. Zur Fehlerüberprüfung wird er mit dem berechneten Wert verglichen.

3 Modul-Kommunikation

Dieses Kapitel beschreibt verschiedene Möglichkeiten, wie austauschbare Hardwaremodule untereinander und mit dem Rest des Systems kommunizieren können. Die hier vorgestellten Varianten können als Kommunikations-Grundlage für die in den nachfolgenden Kapiteln beschriebenen Konzepte angesehen werden.

3.1 Bus-Makros

Die dynamische partielle Rekonfiguration eines FPGAs schafft die Möglichkeit, Hardwarefunktionalität in Module aufzuteilen und diese je nach Bedarf zu laden bzw. auszutauschen. Allerdings muss dafür Sorge getragen werden, dass die Module auch untereinander und mit dem Rest des System kommunizieren können. Die hierfür nötigen Busse können jedoch nicht beliebig vom Modul selbst bestimmt werden, es muss eine definierte Schnittstellen geben, so dass ein Modul in einen beliebigen, für Module vorgesehenen Einschub platziert werden kann. In [Hübner 06b] wird zu genau diesem Zweck das Konzept der Bus-Makros vorgestellt. Solch ein Makro stellt eine fest definierte Schnittstelle dar, die sich für jeden Modul-Einschub an derselbe Stelle befindet. Somit können Module beliebig ausgetauscht werden, ohne dass ihnen selbst bekannt sein muss, wo sie eingesetzt wurden. In Abbildung 3-1 sind die Kommunikationskanäle durch Pfeile angedeutet:

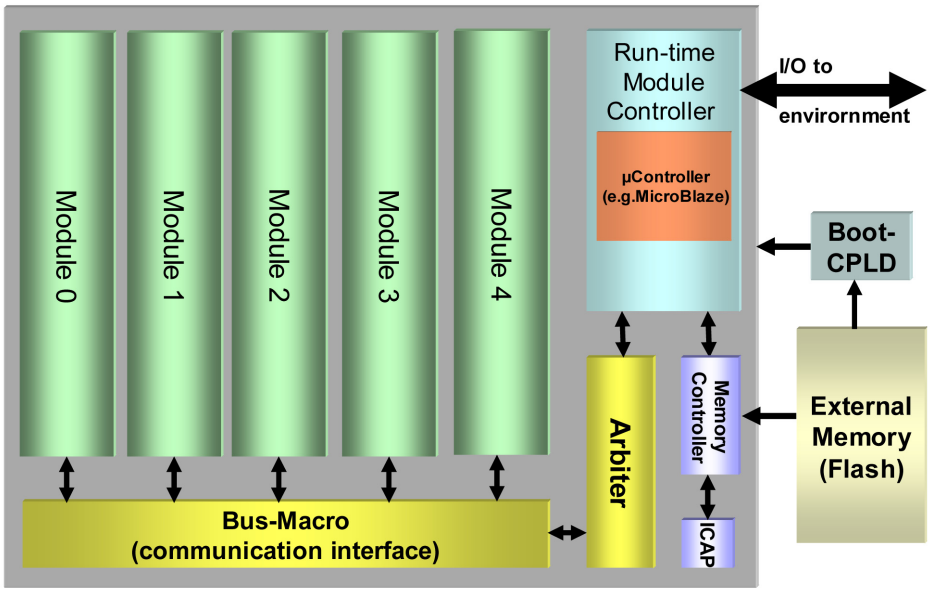


Figure 1. Slot Based Dynamic reconfigurable system

Abbildung 3-1
Schema eines modulorientierten Systems mit Bus-Makros zur Kommunikation

Die Autoren der o. g. Arbeit betonen neben der Notwendigkeit von klar definierten Kommunikationsschnittstellen die Wichtigkeit einer hohen Abstraktionsebene beim Modulentwurf insbesondere für die Akzeptanz des Verfahrens beispielsweise im industriellen Umfeld. Die vorgestellte Variante biete so die Möglichkeit, kleinere und kostengünstigere FPGAs zu verwenden, weil nur die tatsächlich benötigten Hardwareteile auch konfiguriert sein müssten. Außerdem lasse sich dadurch der Energieverbrauch senken.

Einen weiteren in [Carver 08] ausführlich untersuchten Aspekt stellt die Frage dar, an welcher Position sich die Bus-Makros befinden sollten, um ein möglichst optimales Timing des Datenpfades zu erreichen. So fanden die Autoren heraus, dass selbst eine manuelle Platzierung nicht die besten Ergebnisse liefere. Eine manuelle Platzierung, die den Empfehlungen des Herstellers folge, sei im Timing um den Faktor 4,5 schlechter als die best Mögliche. Sie entwickelten deshalb ein Tool, das den Entwurfsraum mit dem Simulated Annealing Algorithmus untersucht und eine möglichst optimale Platzierung der Makros liefern soll.

3.2 Routing-Tabellen

Einen alternativen Ansatz zur Kommunikation verfolgen die Autoren in [Silva 08]. Hier verbinden die definierten Schnittelen (*Kommunikationsendpunkt* genannt) nicht die Module mit einem allgemeinen Bussystem, sondern sie öffnen Kanälen direkt zwischen zwei unterschiedlichen Modulen, wie in Abbildung 3-2 verdeutlicht.

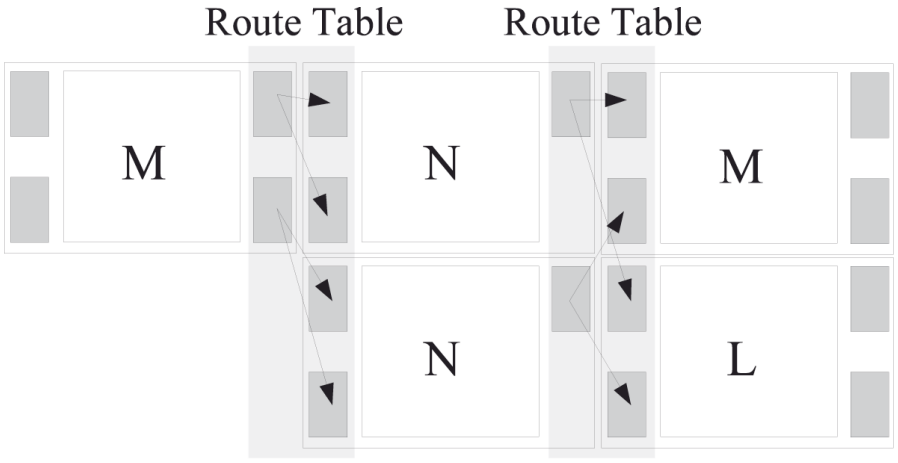


Abbildung 3-2
Kommunikationsschema das Module mit Hilfe von Routing-Tabellen verbindet

Dadurch ist eine sehr performante und effiziente Verbindung zwischen zwei Modulen möglich, die als eine Art Pipeline angesehen werden kann. Da ein Endpunkt eines Moduls auch mit mehreren Endpunkten von mehreren Modulen verbunden sein kann (siehe Abbildung 3-2), ergibt sich eine komplizierte Konfiguration der Verbindungsmatrix des FPGAs, die zum Zeitpunkt der Rekonfiguration errechnet werden müsste. Hier greift nun der in der o. g. Arbeit vorgestellte Ansatz der vorberechneten Routing-Tabellen, in denen alle

möglichen kürzesten Verbindungen bereits hinterlegt sind. Für jeden Endpunkt sind in der Tabelle Einträge vorhanden, die jeweils die Konfiguration der Verbindungsmatrix für eine Verbindung zu jedem anderen Endpunkt der anliegenden Spalte darstellen. Somit vereinfacht sich die Berechnung der Route zwischen zwei Endpunkten auf das Auslesen des entsprechenden Tabelleneintrags für Quell- und Zielendpunkt. Aufgrund der sich wiederholenden Struktur der CLB-Spalten wird nur eine Tabelle benötigt, die für alle Spalten verwendet werden kann. Die aus der Tabelle ausgelesen Konfigurationsdaten werden dann zur eigentlichen Rekonfiguration der Hardware in den entsprechenden Bitstream migriert. Für den Fall, dass nicht alle Module gleich breit sind, stehen sogenannte *Feed-Through-Module* zur Verfügung, welche die entstandenen Lücken durch eine simple Weiterleitung schließen:

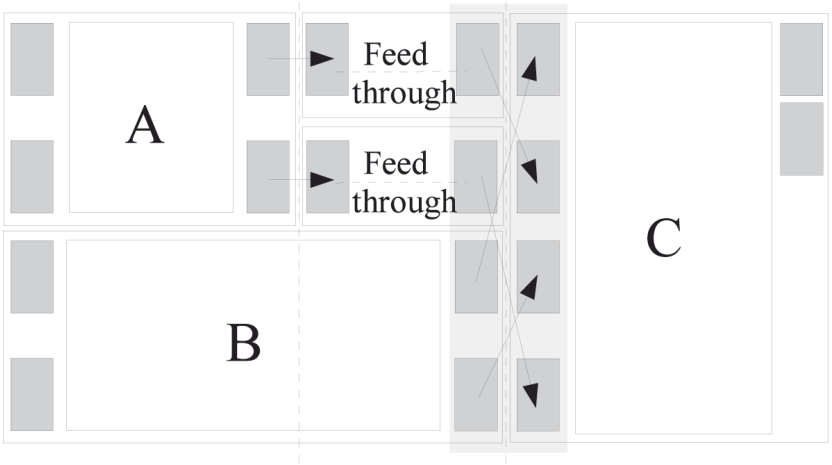
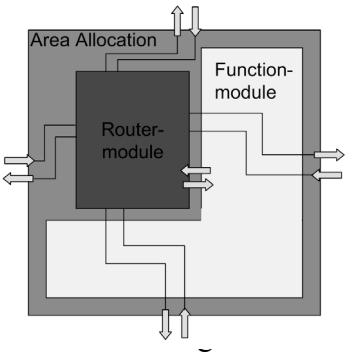


Abbildung 3-3
Routing-Beispiel mit zwei Feed-Through-Modulen

3.3 Online Routing

Im Gegensatz zu dem im vorherigen Kapitel 3.2 vorgestellten Ansatz mit vorberechneten Routing-Tabellen findet sich in [Hübner 06a] ein Verfahren, das sich durch ein zur Laufzeit hochgradig anpassbares Routing auszeichnet. Die Autoren entschieden sich in diesem Fall für ein Netzwerk von standardisierten Routern, die direkt in die Funktionsmodule integriert werden. Abbildung 3-4 zeigt solch ein Modul mit integriertem Router. Sollte die Logik eines Moduls mehr Fläche benötigen, lässt sich ein komplexeres Modul aus mehreren Einheiten der gezeigten Größe zusammensetzen. Dieses Zellenprinzip stellt sicher, dass sich die Kommunikationskanäle zu anderen Modulen immer an den bekannten Positionen befinden. So lässt sich ein Gesamtsystem aufbauen, wie beispielsweise in Abbildung 3-5 gezeigt. Modul A und B haben hier die minimale Größe und bieten die aus Abbildung 3-4 bekannten vier Kommunikationskanäle. Diese sind jeweils 8 Bit breit. Die Module A, D und E hingegen bestehen aus mehreren zusammengesetzten Funktionsmodulen und haben entsprechend mehr Kommunikationskanäle an den jeweils fest definierten Positionen.



Funktionsmodul mit integriertem Router

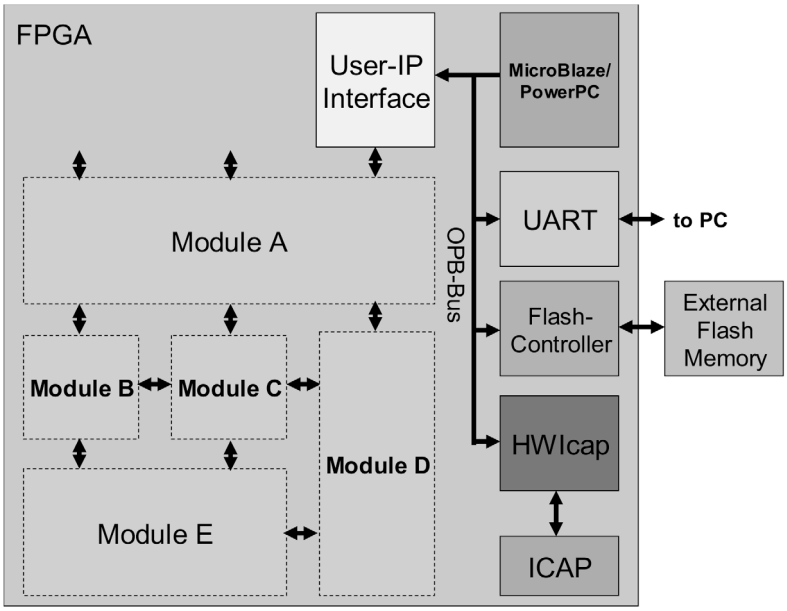


Abbildung 3-5
Beispielanordnung verschiedener Module

Die Router in den Funktionsmodulen sind nun in der Lage, Datenpakete zu jedem gewünschten Zielpunkt weiterzuleiten. Die Adressierung erfolgt hierbei nach einem Schachbrettmuster. Der Routingalgorithmus berechnet für jedes Paket den kürzesten Weg, indem er die Weglängen von Nord-, Süd-, Ost- und Westausgang miteinander vergleicht. Der gewonnenen Flexibilität und möglichen Steigerung der Transferdatenrate steht natürlich der zusätzliche Bedarf an Hardwareressourcen für die integrierten Router gegenüber. Im vorgestellten Beispiel wählten die Autoren eine Modulgröße von 220 CLBs, von denen 62 CLBs für den Router und weitere 28 CLBs für die Verbindungslogik belegt werden, so dass für die eigentliche Anwendungslogik nur noch 130 der 220 CLBs zur Verfügung stehen.

4 Sicherheitskonzepte für den Rekonfigurationsprozess

Aufbauend auf den vorangegangenen Grundlagenkapiteln werden hier Mechanismen vorgestellt, welche die Sicherheit eines dynamisch partiell rekonfigurierbaren Systems erhöhen können. Das Kapitel umfasst zwei große Bereiche, einerseits die Überprüfung gegen Veränderungen und Übertragungsfehler der zu aktualisierenden Konfigurationsdaten selbst und andererseits ein Verfahren zum Schutz der nicht von der Rekonfiguration betroffenen FPGA-Bereiche.

4.1 Integritätsprüfung

Um sicher zu stellen, dass sich ein rekonfiguriertes Hardwaremodul auch so verhalten wird wie erwartet, liegt es nahe, die Integrität des zum Update verwendeten Bitstreams zu überprüfen. Das in [Chaves 08] vorgestellte Verfahren ermöglicht es die Korrektheit der Konfigurationsdaten während der Rekonfiguration on-the-fly zu validieren. Von den Autoren wird hierzu ein Hash-Algorithmus auf den Konfigurationsbitstream angewendet. Die vorgestellte Implementierung ermöglicht die Auswahl zwischen zwei verschiedenen Hash-Methoden, die beide auf dem *Secure Hash Algorithm* (SHA) basieren: SHA-128 und SHA-256. Aufgrund der Überprüfung während des Rekonfigurationsprozesses (on-the-fly) bewirkt das gewonnene Maß an Sicherheit keine Performanzeinbußen. Ein weiterer von den Autoren angeführter Vorteil liegt in der einfachen Integration des Validierungsmoduls in ein bestehendes Design. Um den Rekonfigurationsprozess zu überwachen, arbeitet die zusätzliche Hardware eng mit dem in Kapitel 2.3.2 erläuterten ICAP-Modul zusammen. Die Daten werden per Snooping direkt vom Datenbus der ICAP-Schnittstelle abgegriffen und der Hash-Wert wird aktualisiert, sobald die nächsten 512 Bit bereitstehen. Sollte die Bitstreamlänge kein ganzzahliges Vielfaches von 512 Bit sein, werden die noch fehlenden Bits mit Nullen aufgefüllt, sobald mit dem Signal *readHash* der berechnete Hash-Wert abgefragt wird. Für den Fall, dass der berechnete Hash-Wert nicht mit dem im Bitstream hinterlegten übereinstimmt, stellt sich die Frage, was mit der anscheinend falsch konfigurierten Hardware geschehen soll. Die Autoren der o. g. Arbeit haben diesen Fall nicht weiter ausgeführt. Denkbar wäre z.B. den Rekonfigurationsprozess erneut zu starten, um Übertragungsfehler zu korrigieren. Falls es jedoch eine Übereinstimmung der Hash-Werte gibt, kann das geschilderte Verfahren die Authentizität des geladenen Hardwaremoduls sicherstellen.

4.2 Bereichsbeschränkung

Ein weiterer Punkt von entscheidender Bedeutung für die Sicherheit eines dynamisch partiell rekonfigurierbaren Systems ist das Problem der Einschränkung des Rekonfigurationsbereichs auf den zulässigen Bereich. So darf ein Hardwaremodul, das in einen für dieses Modul vorgesehenen Einschub geladen werden soll, keine anderen Bereiche der FPGA-

Konfiguration verändern können. Das schon aus Kapitel 4.1 bekannte Verfahren, den Konfigurationsdatenstrom per Snooping am ICAP-Bus zu überwachen, kann auch für dieses Problem Abhilfe schaffen. Ebenfalls in [Chaves 08] und [Bok 07] wird ein System zur automatischen Bereichsüberprüfung vorgestellt. Dieses überprüft bereits während des Rekonfigurationsprozesses, welcher Frame (siehe Kapitel 2.1 und 2.2) der Konfigurationsschicht des FPGAs neu geschrieben werden soll. Sollte es sich um einen Frame außerhalb eines als erlaubt definierten Bereichs handeln, generiert das System sofort ein Abbruchsignal, durch das die Rekonfiguration direkt abgebrochen wird. Da das ICAP-Modul ein Paket aus dem Konfigurationsbitstream (siehe Kapitel 2.2) erst dann verarbeitet, wenn das nächste Paket eingetroffen ist, bleibt dem System zur Bereichsprüfung die nötige Zeit den Rekonfigurationsprozess zu unterbrechen, bevor der Frame, dessen Adresse außerhalb der erlaubten Region liegt, geschrieben wird. Abbildung 4-1 zeigt die Anbindung zur ICAP-Schnittstelle:

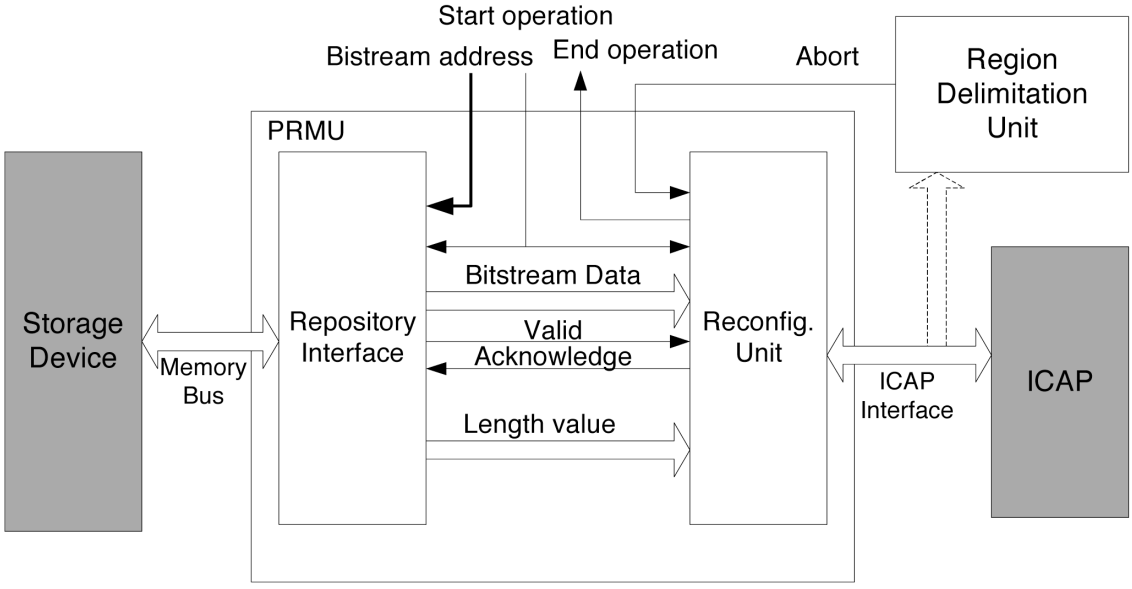


Abbildung 4-1
Rekonfigurationslogik mit Anbindung zum ICAP-Modul

4.3 Ressourcenbedarf und Performanz

Die vorgestellte Implementierung der Bereichsüberwachung belegt laut der in [Chaves 08] aufgeführten Ergebnisse 264 Slices eines Virtex II Pro 30 FPGAs und 512 Byte BRAM. Durch Hinzufügen der Integritätsprüfung des Bitstreams erhöht sich die benötigte Chipfläche je nach Hash-Algorithmus um 533 Slices (SHA-128) oder 849 Slices (SHA-256). Bei Verwendung des SHA-256 Algorithmus ist zusätzlich noch ein weiterer BRAM-Block erforderlich. Insgesamt ergibt sich also für die beiden Sicherheitsfeatures zusammen ein Bedarf an 797 bzw. 1113 Slices und 1 bzw. 2 BRAM-Blöcken. Es stellt sich außerdem keine Verlangsamung des Rekonfigurationsprozesses ein, da die nötige Logik (bei einem Takt von 100 MHz) eine Datenrate von 786 MBit/s erreicht, die deutlich über der maximalen Inputdatenrate der ICAP-Schnittstelle von 400 MBit/s liegt.

5 Strukturierung des Rekonfigurationsbereichs

Dieses Kapitel stellt verschiedene Möglichkeiten zur Strukturierung der dynamisch rekonfigurierbaren Region eines FPGAs vor. Wie bereits aus Kapitel 2 bekannt, gibt es hier grundlegende Unterschiede zwischen den verschiedenen FPGA-Architekturen. Es erfolgt in diesem Kapitel eine grobe Klassifizierung der unterschiedlichen Möglichkeiten auf den zugrundeliegenden Architekturen Virtex II und Virtex 4.

5.1 Eindimensionale Positionierung

Die einfachste Möglichkeit der Platzierung von austauschbaren Hardwaremodulen ergibt sich mit einer eindimensionalen Strukturierung. Ein Beispiel hierfür stellt das schon aus Kapitel 3.1 bekannte Modul-Konzept dar, in dem die rekonfigurierbaren Hardwaremodule in vordefinierte Einschübe platziert werden können. Der in [Hübner 06b] vorgestellte Rekonfigurationsprozess ermöglicht es, Teile des Systems zur Laufzeit gegen andere Teile austauschen zu können. Abbildung 5-1 zeigt einen möglichen Rekonfigurationsablauf auf der Zeitachse:

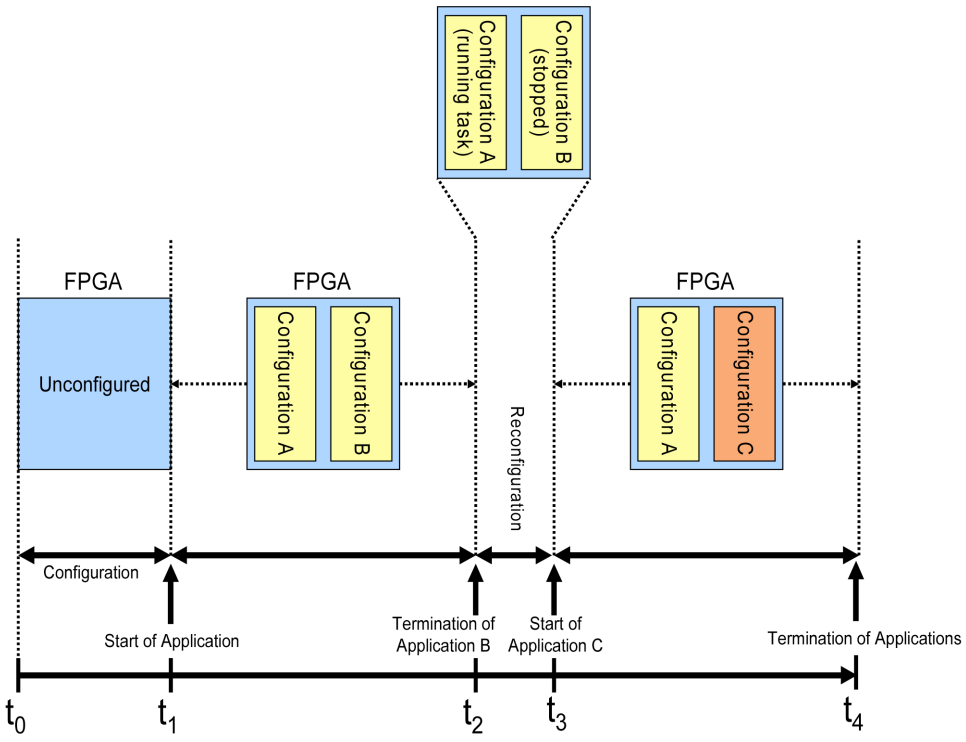


Figure 3. Process of Dynamic and Partial Reconfiguration

Abbildung 5-1
Zeitlicher Ablauf einer dynamischen partiellen Rekonfiguration

Das zum Zeitpunkt t_2 nicht weiter benötigte Modul *B* wird in diesem Beispiel gegen ein zum Zeitpunkt t_3 erforderliches Modul *C* ausgetauscht.

Aufgrund der eindimensionalen Anordnung der Module ergeben sich bei diesem Verfahren keine Einschränkungen oder Unterschiede durch die verwendete FPGA-Architektur. Die verwendete Strategie zur Modulplatzierung entspricht im Wesentlichen der Anordnung der Rekonfigurationseinheiten bei Virtex II FPGAs, wie in Kapitel 2.1 vorgestellt. Sie lässt sich jedoch genauso auch im Falle von Virtex 4 FPGAs anwenden. Die physische Umsetzung der eindimensionalen Moduleinschübe ist in dem in Abbildung 5-2 gezeigten Floorplan direkt erkennbar:

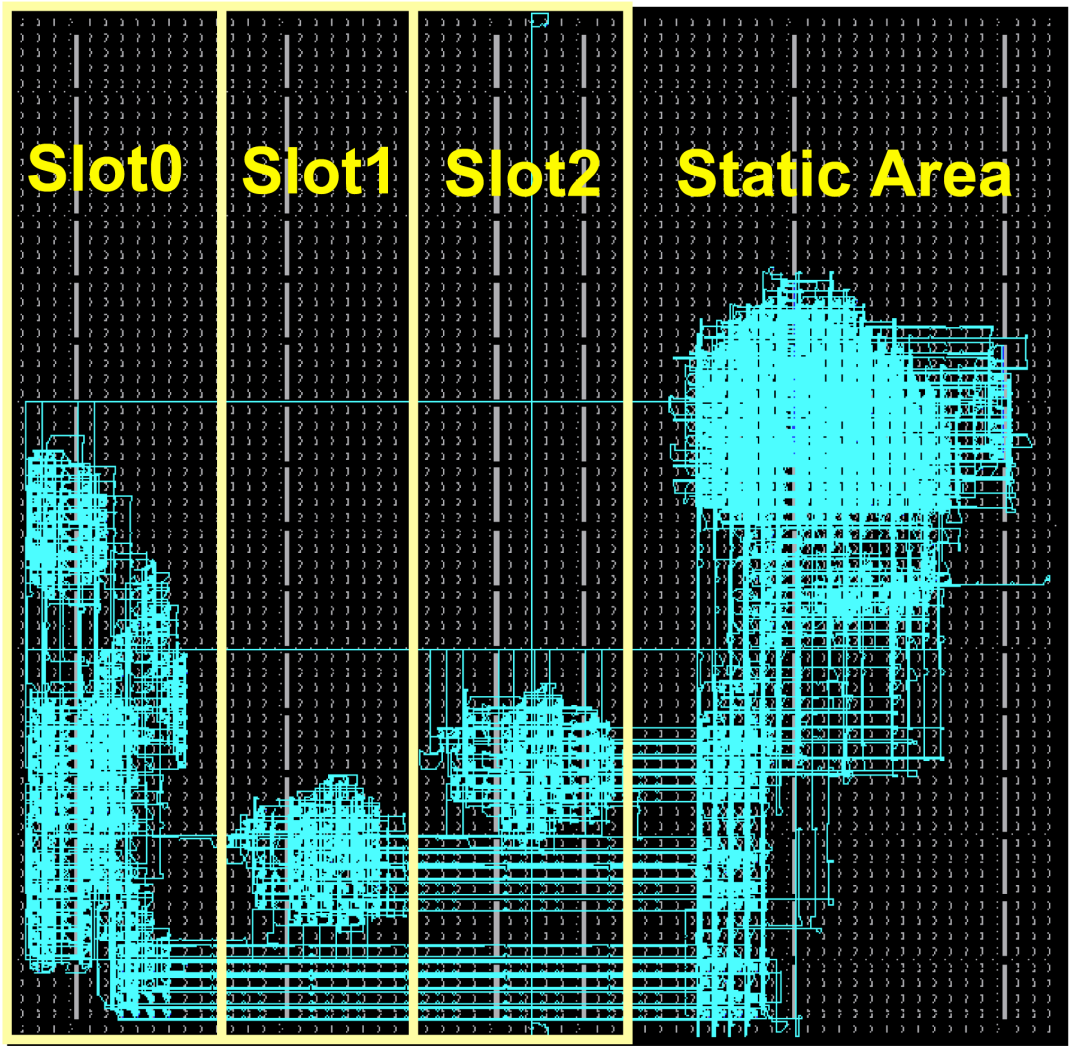


Abbildung 5-2
Floorplan der rekonfigurierbaren und statischen Bereiche des FPGAs

Solch eine eindimensionale Positionierungsstrategie der rekonfigurierbaren Hardwaremodule harmoniert sehr gut mit dem Kommunikationskonzept der Bus-Makros aus Kapitel 3.1.

5.2 Zweidimensionale Positionierung

5.2.1 Verteilung auf homogene Bereiche

Die zweidimensionale Positionierung ermöglicht eine flexiblere Modulanordnung und somit auch eine bessere Ausnutzung der Fläche des FPGAs, es ergeben sich aber auch zwei zusätzliche Probleme: Erstens muss die FPGA-Architektur eine vertikale Unterteilung in mehrere Konfigurationsframes zulassen, dies ist bei der Virtex II Familie beispielsweise nicht gegeben (Kapitel 5.2.3 zeigt allerdings eine mögliche Lösung). Zweitens müssen die vom FPGA bereitgestellten Ressourcen (CLB, DSP, BRAM) der Zielregion für das zu ladenden Modul mit den vom Modul erwarteten Ressourcen übereinstimmen. Daher ergibt sich die zusätzliche Einschränkung, dass der Rekonfigurationsbereich bereits entsprechend strukturiert sein muss. Die Plätze für die Hardwaremodule müssen also *homogen* sein, wenn eine beliebige Modulanordnung möglich sein soll. Kapitel 5.2.2 stellt eine weitere Möglichkeit vor, wie diese Einschränkung aufgehoben werden kann.

Der eigentliche Rekonfigurationsprozess läuft wie in [Silva 08] vorgestellt in mehreren Schritten ab, zunächst muss eine gültige Zielposition ausgewählt werden. Anschließend müssen die für das Modul erforderlichen Kommunikationskanäle bereitgestellt und angeschlossen werden, für die zweidimensionale Positionierung eignen sich die Verfahren aus Kapitel 3.2 und 3.3 am besten. Um das neue Modul nun auch als Hardware zu aktivieren, muss sein angepasster Bitstream an das ICAP-Modul gesendet werden. Da im Allgemeinen nicht davon ausgegangen werden kann, dass alle Modul gleich groß sind, ist es unter Umständen erforderlich, die im zweidimensionalen Rekonfigurationsbereich entstandenen Lücken mit den in 3.2 bereits erwähnten *Feed-Through-Modulen* zu schließen.

5.2.2 Verteilung auf heterogene Bereiche

Das in [Becker 07] vorgestellte Verfahren zur zweidimensionalen Positionierung von Hardwaremodulen hebt die im vorangegangenen Kapitel aufgestellte Einschränkung der Forderung einer Ressourcengleichheit auf.

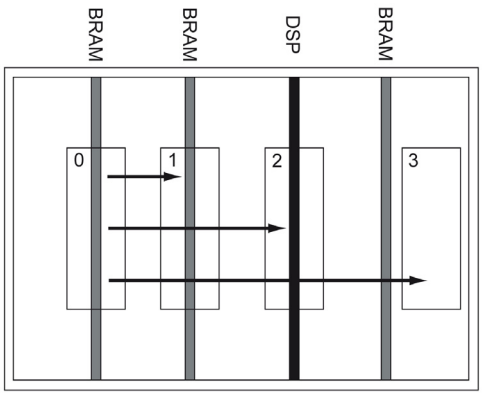


Abbildung 5-3
Relokation auf identische und nicht identische Regionen

Das von den Autoren entwickelte Konzept ermöglicht wie in Abbildung 5-3 dargestellt nicht nur eine Relokation von Region 0 auf die identische Region 1, sondern auch auf die nicht identischen Regionen 2 und 3. Es beruht darauf, dass sich der Aufbau der Konfigurationsschicht des FPGAs (siehe Kapitel 2.1) auch bei unterschiedlichen Ressourcen in der Hardware-Schicht nicht unterscheidet. Der in Abbildung 5-4 dargestellte Ausschnitt aus beiden Schichten umfasst in Region 1 wie in Region 2 Routingressourcen (R) und Logikressourcen vom Typ A. Logikressourcen vom Typ B sind jedoch nur in Region 1 vorhanden, Region 2 enthält an entsprechender Stelle Logikressourcen vom Typ C:

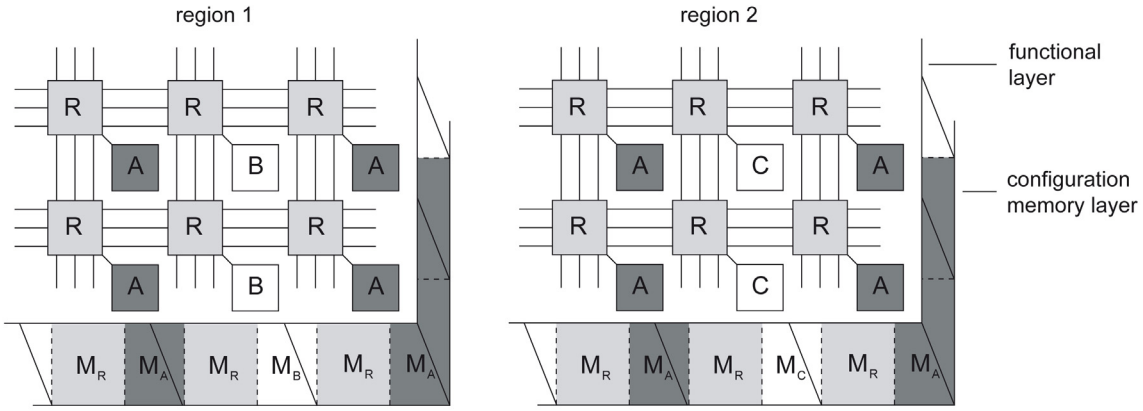


Abbildung 5-4
Kombinierte Darstellung von Hardware- und Konfigurationsschicht mit den unterschiedlichen Logikressourcen A, B und C

Ein Modul, das in beiden Regionen einsetzbar sein soll, darf natürlich keine Ressourcen vom Typ B und C verwenden. Damit zur Laufzeit beispielsweise eine Relokation von Region 1 nach Region 2 stattfinden kann, modifiziert das Laufzeitsystem den Konfigurationsbitstream nun so, dass alle Frames, die zu Ressource B gehörten, in Frames vom Ressourcen-Typ C geändert werden. Das entspricht in Abbildung 5-4 der Ersetzung von M_B durch M_C. Der in Abbildung 2-6 gezeigte Aufbau eines Bitstreams bei Virtex 4 FPGAs erleichtert das Vorgehen, da die Routingkonfiguration grundsätzlich identisch ist und somit nur die letzten Frames einer Spalte verändert werden müssen. Das folgenden Beispiel aus [Becker 07] demonstriert eine Relokation von einer Region mit einem DSP-Block zu einer anderen Region mit einem BRAM-Block. Abbildung 5-5 zeigt die nötigen Schritte der Bitstreamanpassung, bei der sich die Quellregion über die CLB-Spalten 8 bis 10 in Zeile 1 erstreckt. Als Zielregion soll der Bereich von Spalte 23 bis 24 in Zeile 1 dienen. In der Konfiguration für die erste CLB-Spalte müssen lediglich die Adressen angepasst werden, um die Verschiebung von Spalte 8 zu Spalte 23 zu realisieren. Die 22 Datenframes können unverändert übernommen werden. Bei der nächsten Spalte handelt es sich um die DSP-Ressource, die in der Zielregion nicht vorhanden ist. Daher wird hier zunächst die Adressierungsart angepasst und die neue Zieladresse berechnet. Es ergibt sich eine Verschiebung von Block-Typ 0 (CLB / DSP) zu Block-Typ 1 (BRAM) und eine Spaltentransformation von 9 zu 2. Außerdem benötigen auch die Datenframes eine Anpassung, es werden nur die ersten 20 Frames übernommen, welche die Routing-Informationen darstellen, diese sind für alle Spaltentypen identisch. Mit der letzten Spalte des Beispiels wird wie mit der Ersten verfahren, hier ergibt sich eine Verschie-

bung von Spalte 10, Zeile 1 zu Spalte 24, Zeile 1. Die Datenframes können wieder unverändert übernommen werden.

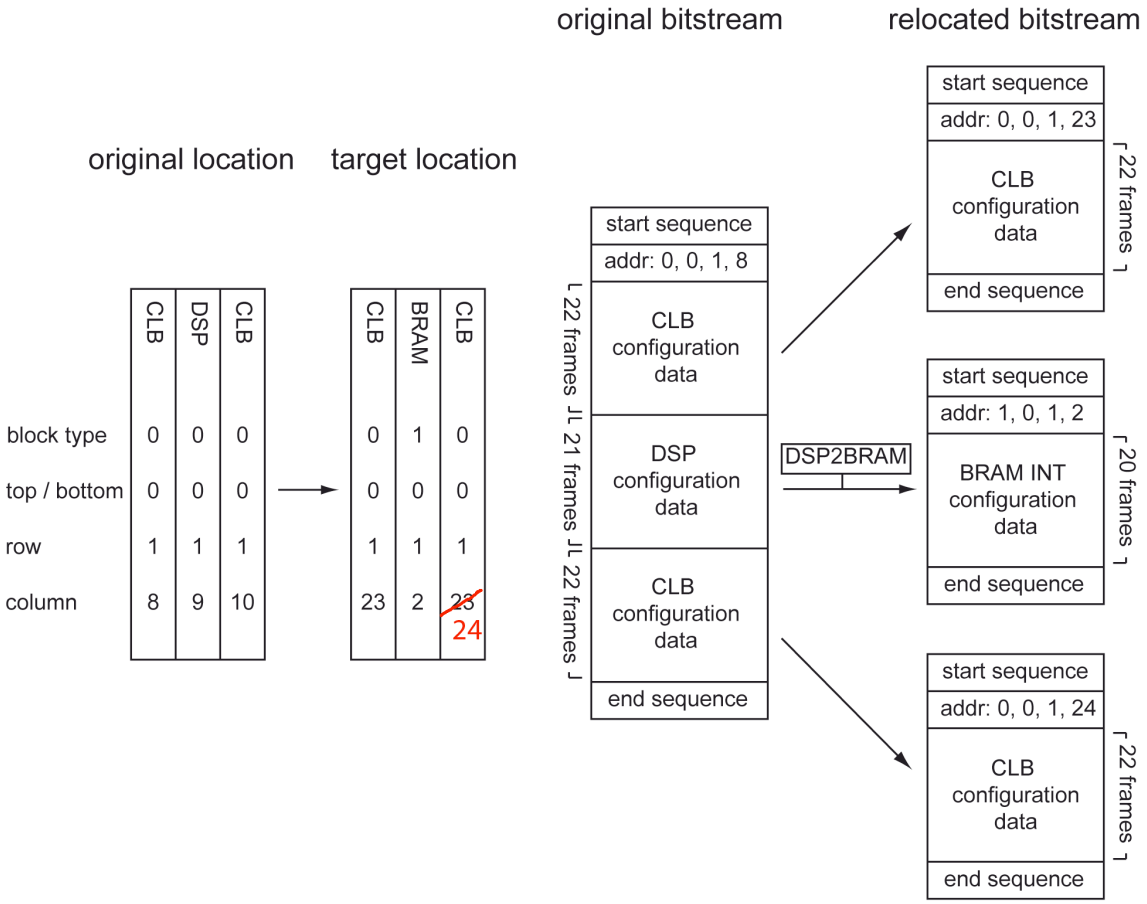


Abbildung 5-5
Bitstreamanpassung für eine Relokation von DSP zu BRAM

Die folgende Untersuchung aus [Becker 07] stellt die Vorteile dieses Konzepts qualitativ dar. Es wurde ein Software Defined Radio Modul entwickelt, das aus zwei heterogenen rekonfigurierbaren Regionen und vier Filtermodulen (low pass, band pass, high pass, all pass) besteht. Die Module belegen jeweils ca. 250 CLBs, was einer Fläche von 16 x 16 CLBs entspricht. Für solche Module der Breite 16 ergebe sich nach den Angaben der Autoren auf einem Virtex 4 FX140 FPGA lediglich eine Möglichkeit zur Relokation, wenn Quell- und Zielregion völlig identisch sein müssten. Für ein Modul der Breite 32 ergebe sich sogar gar keine Relokationsmöglichkeit. Reduziere man nun die Anforderungen an das Modul und setze nur zu 90% identische Regionen voraus, gebe es für das Modul der Breite 16 bereits elf Möglichkeiten zur Relokation und für das 32er Modul immerhin zwei. Eine weitere Reduzierung der Anforderungen auf nur noch zu 80% identische Regionen führe zu 34 alternativen Positionen für das Kleine und elf für das große Modul.

5.2.3 Die *read-modify-write* Methode

Um eine vertikale Relokation von Hardwaremodulen auch auf Virtex II FPGAs zu ermöglichen, lässt sich die in [Sedcole 06] beschriebene Methode verwenden. Das Konzept beruht auf der „*glitchtless dynamic reconfiguration*“ genannten Eigenschaft des FPGAs: Wird ein Konfigurationsbit geschrieben, das denselben Wert enthält wie vor dem Update, dann ist die Funktionsfähigkeit der zugehörigen Hardwareressource zu keiner Zeit beeinträchtigt. Dadurch ist es möglich, einen Teil einer CLB-Spalte zu rekonfigurieren, indem zunächst die aktuelle Konfiguration des Frames ausgelesen, dann der zu aktualisierende Teil im Bitstream geändert und abschließend der komplette Frame zurückgeschrieben wird. Der Rekonfigurationsprozess besteht also aus einem Lese-, Modifizierungs- und Schreibvorgang, was zum Namen *read-modify-write* Methode führt. Da die Rekonfiguration frameweise erfolgt, wird lediglich Speicherplatz benötigt um einen Frame zu puffern.

6 Zusammenfassung

Ein abschließendes Kapitel, das die vorgestellten Arbeiten und Konzepte noch ein mal kurz zusammenfasst, gegeneinander abgrenzt und bewertet.

Zum Thema der Bitstream Manipulation für dynamisch partiell rekonfigurierbare Systeme wurden neben den Grundlagen wie Aufbau von FPGAs und den zugehörigen Bitstreams drei Schwerpunkte vorgestellt. Der erste wesentliche Punkt betrifft die Kommunikation der rekonfigurierbaren Module untereinander und mit der Außenwelt. Die Kommunikationskonzepte sind zwar eng mit den technischen Grundlagen verbunden, spielen aber auch auf höherer Abstraktionsebene eine entscheidende Rolle, da nicht jedes Verfahren gleich gut für jeden Anwendungsfall geeignet ist.

Ein weiterer Schwerpunkt ist durch die Strukturierung des Rekonfigurationsbereichs gegeben. Hier gibt es zwei grundlegende Konzepte der Modulpositionierung, nämlich eine ein- oder zweidimensionale Anordnung. Im Falle einer zweidimensionalen Region für die rekonfigurierbaren Hardwaremodule sind einige Details aus dem grundsätzlichen Aufbau der FPGA-Architektur zu beachten, insbesondere die Granularität der Konfigurationsschicht (Framegröße) und die Unterscheidung zwischen verschiedenen Ressourcetypen (CLB, DSP, BRAM).

Das einfachste Kommunikationskonzept der Bus-Makros integriert sich gut in eine eindimensionale Positionierung und erfordert den geringsten Overhead. Um die komplexeren Positionierungsmodelle auch ausnutzen zu können, bietet sich in diesem Fall ein aufwendigeres, aber dafür flexibleres und in dem meisten Fällen auch performanteres Kommunikationsmodell an. Vielen Anwendungsfällen wird vermutlich ein statisches Routing genügen, hier bietet das Modell der offline vorberechneten Routing-Tabellen weitreichende Möglichkeiten. Eine noch größere Flexibilität stellt schließlich das vorgestellte Online-Routing Modell bereit.

Der dritte Schwerpunkt liegt auf der Sicherheit des Rekonfigurationsprozesses und ist von höherer Ebene gesehen recht unabhängig von den anderen beiden Schwerpunkten. Aus Sicht der Implementierung ist er jedoch eng mit ihnen verbunden. Der Sicherheitsaspekt lässt sich auf zwei wichtige Bereiche verteilen, einerseits die Sicherherstellung der Korrektheit des neu geladenen Hardwaremoduls gegenüber dem Originalbitstream im Speicherpool und andererseits die Unterbindung von Veränderungen der Systemhardware, die nicht innerhalb des für das zu ladende Modul gültigen Bereichs liegen. Letzteres ist mit relativ wenig Hardwareaufwand verbunden und daher auch entsprechend leicht zu realisieren. Die Integritätsprüfung erfordert deutlich mehr Hardwareaufwand, wenn sie in Echtzeit parallel zur Rekonfiguration erfolgen soll, bietet aber auch einen hohen Gewinn an Sicherheit.

Insgesamt ist für alle Teilbereiche der Aufbau der FPGA-Architektur und das Format des Rekonfigurationsbitstreams von entscheidender Bedeutung. In allen Fällen müssen die Konfigurationsdaten des Bitstreams zumindest ausgewertet, meistens aber sogar verändert werden.

Abbildungsverzeichnis

Literaturverzeichnis

[Becker 07] Tobias Becker, Wayne Luk and Peter Y.K. Cheung “Enhancing Relocatability of Partial Bitstreams for Run-Time Reconfiguration”, International Symposium on Field-Programmable Custom Computing Machines, 2007, Pages: 35-44.

[Bok 07] Kees van der Bok, Ricardo Chaves, Georgi Kuzmanov, Leonel Sousa, Arjan van Genderen “Dynamic FPGA Reconfigurations with Run-Time Region Delimitation”, Workshop on Circuits, Systems and Signal Processing (ProRISC), 2007, Pages: 201-207.

[Carver 08] Jeff Carver, Neil Pittman, Alessandro Forin “Relocation and Automatic Floor-planning of FPGA Partial Configuration Bit-Streams”, Technical Report MSR-TR-2008-111 Microsoft Research Microsoft Corporation, 2008.

[Chaves 08] Ricardo Chaves, Georgi Kuzmanov, Leonel Sousa “On-the-fly Attestation of Reconfigurable Hardware”, Field Programmable Logic and Applications, 2008, Pages: 71-76.

[Hübner 06a] Michael Hübner, Christian Schuck, Matthias Kühnle, Jürgen Becker “New 2-Dimensional Partial Dynamic Reconfiguration Techniques for Real-time Adaptive Microelectronic Circuits”, IEEE Computer Society Annual Symposium on VLSI: Emerging VLSI Technologies and Architectures, ISVLSI'06, 2006, Pages: 97-102.

[Hübner 06b] M. Hübner, J. Becker “Exploiting Dynamic and Partial Reconfiguration for FPGAs — Toolflow, Architecture and System Integration”, SBCCI’06, 2006.

[Krasteva 06] Y.E. Krasteva, E. de la Torre, T. Riesgo, Didier Joly “Virtex II FPGA Bitstream Manipulation: Application to Reconfiguration Control Systems”, Field Programmable Logic and Applications, 2006, Pages: 1-4.

[Silva 08] Miguel L. Silva, Joao Canas Ferreira “Generation of Partial FPGA Configurations at Run-Time”, Field Programmable Logic and Applications, 2008, Pages: 367-372.

[Sedcole 06] P. Sedcole, B. Blodget, T. Becker, J. Anderson and P. Lysaght „Modular dynamic reconfiguration in Virtex FPGAs“, Field Programmable Logic and Applications, 2006, Pages: 157-164.