Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Ausarbeitung AW1 - WiSe 2010/11

Andre Jestel

Hardware-Software Partitionierung für SoC-Plattformen

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Andre Jestel

Hardware-Software Partitionierung für SoC-Plattformen

Ausarbeitung eingereicht im Rahmen von Anwendungen 1
im Studiengang Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr.-Ing. Bernd Schwarz
Gutachter : Prof. Dr.-Ing. Birgit Wendholt
Gutachter : Prof. Dr. Franz Korf

Abgegeben am 26. Februar 2011

Inhaltsverzeichnis

1 Einleitung

In vielen Anwendungsgebieten, wie es zum Beispiel bei der Signalverarbeitung in der Automobil- und Luftfahrtbranche der Fall ist, treten Problemstellungen auf, die System-Designer in der Entwurfsphase vor die Herausforderung stellt, die Ressourcenausnutzung sowie die Verarbeitungsgeschwindigkeit zu optimieren. Gleichzeitig gilt es, die Kosten und den zeitlichen Aufwand zu minimieren [3]. Bei der Umsetzung eines solchen Systems stellt sich die Frage, welche der Funktionalitäten am besten in Software, und welche in Hardware realisiert werden. Abbildung 1 stellt dar, dass eine System-Implementierung ausschließlich in Hardware die Kostengrenze übersteigt, während dagegen die Verarbeitungsgeschwindigkeit einer vollständige Software-Realisierung nicht den Anforderungen genügt [14]. Diese Entscheidungs- und Kompromissfindung basiert meist subjektiv auf den Erfahrungswerten der am Projekt beteiligten Entwickler. Wünschenswert wären objektive und automatisierte Verfahren zur Partitionierung, die unabhängig von einem bestimmten Anwendungsbereich und entkoppelt von spezieller Hardware bzw. (Co-)Prozessorsystemen arbeiten.

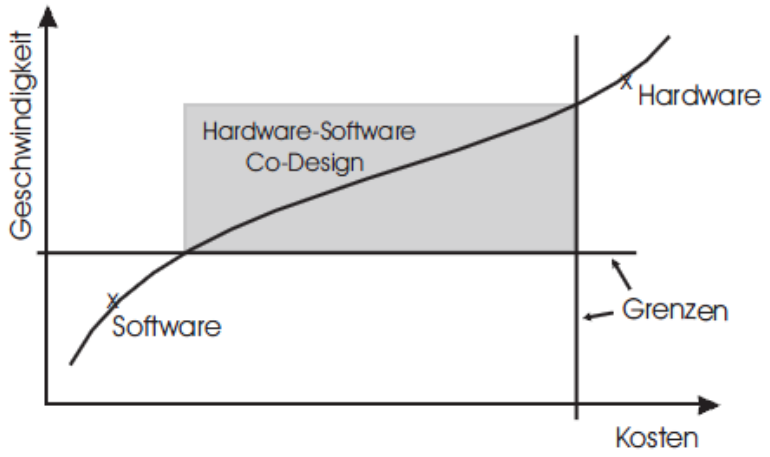

Abbildung 1: Entwicklungskosten-Geschwindigkeitsgraph mit Anforderungsgrenzen für die Implementierung in Hardware oder Software [10]

Ziel dieser Ausarbeitung, einen Überblick über die Entscheidungskriterien zur Hardware-Software Partitionierung sowie den aktuellen Stand der Technik von Verfahren zu vermitteln, die sich zur automatisierten Hardware-Software Partitionierung eignen. Aktuell existieren folgende Anwendungsbereiche für diese Verfahren:

- Im Forschungsschwerpunkt "Fahrerassistenz- und autonome Systeme" (FAUST) der HAW-Hamburg müssen die dort verwendeten Algorithmen zur Fahrbahn- und Umgebungserkennung große Mengen an Sensordaten zum Beispiel im Hinblick auf Kollisionsvermeidung oder Bahnplanung in Echtzeit weiterverarbeiten [11]. Dabei kommen eingebettete Systeme auf "System on a Chip" (SoC) Plattformen zum Einsatz. Bei Auswahl eines optimalen Hardware-Software Co-Designs sowie der Integration der Teilkomponenten sind stets Partitionierungsentscheidungen zu treffen.
- Für die Echtzeitanwendungen im "High Performance Embedded Computing" (HPEC) besteht ein Interesse daran, wie bei der Umsetzung auf einem "Symmetrischen Multiprozessorsystem" (SMP) mit Echtzeitbetriebssystem eine effiziente Parallelisierung der Software-Threads [1, 15] erzielt werden kann. Anschließendes Ziel dabei ist es die Parallelisierung sämtlicher Teilkomponenten, sowohl in Hardware als auch in Software zu maximieren (Abbildung 2). Dieser Prozess stellt ebenfalls eine Partitionierungsproblem dar, zu deren Lösung entschieden werden muss, wann welche Hardware-Ressourcen den Software-Threads zugeteilt werden.

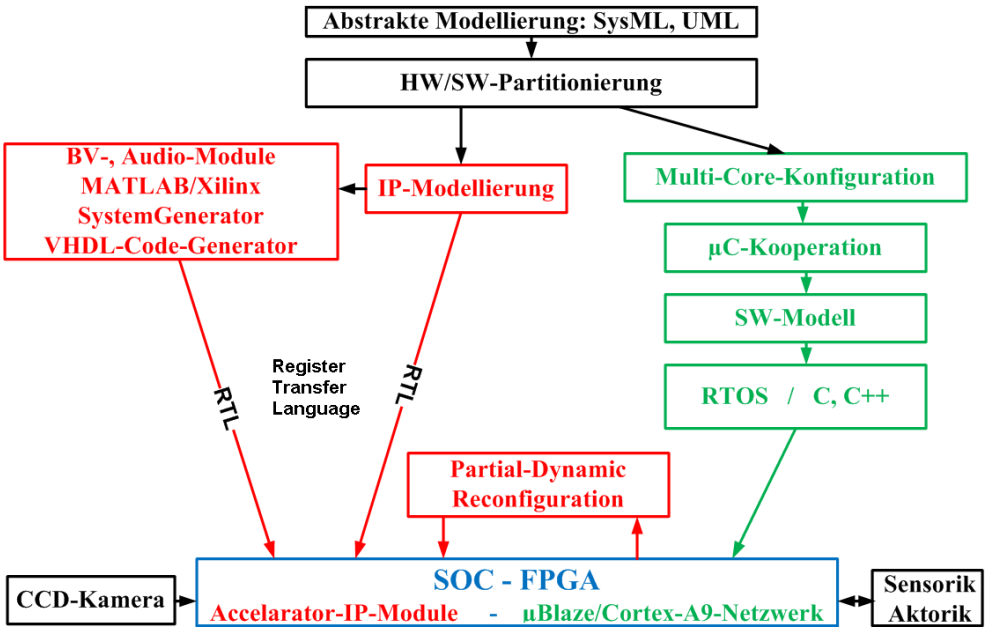

Abbildung 2: Dargestellt ist der thematische Rahmen mit aktuellen und zukünftigen HPEC-Arbeitsfeldern in denen Partitionierungsprobleme (Hardware-Software) auftreten

- Die für Anfang 2011 angekündigte *Extensible Processing Platform* [18] der Firma Xilinx®, die auf einem ARM Cortex™-A9 mit zwei Kernen basiert (Abbildung 3), unterstützt den Integrationsprozess der Hardware-Software Partitionen sowie die Erprobung von Partitionierungsverfahren. Steht die Partitionierung fest, können die Software-Komponenten auf der ARM-CPU ausgeführt werden, während die Hardware-Komponenten auf der selben Plattform synthetisiert werden können. Da sich somit ein komplett partitioniertes System auf dieser SoC-Plattform abbilden lässt, können die Partitionierungsergebnisse, die aus den Verfahren resultieren, dort erprobt und durchdrungen werden.

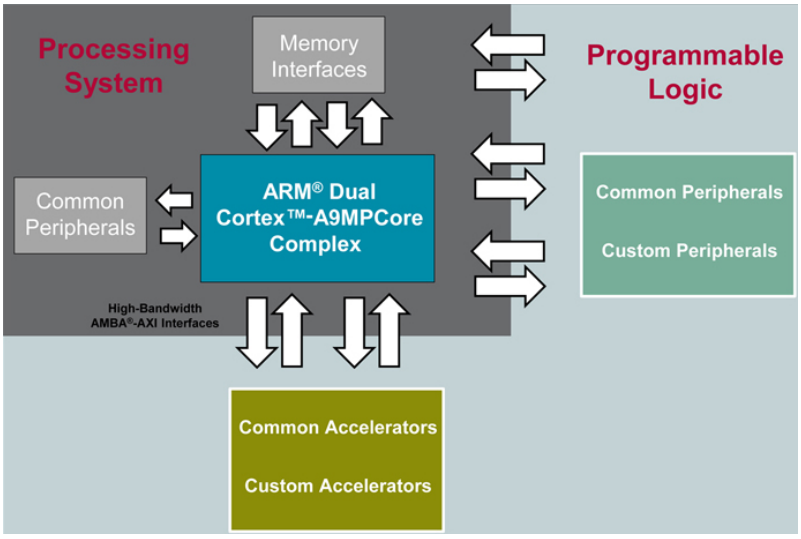

Abbildung 3: Die SoC-Plattform ermöglicht die Ausführung von Software auf einem Zweikern Prozessor und verfügt darüber hinaus über Schnittstellen zu Peripherie, Speicher und Hardwarebeschleuniger zur Synthese von VHDL-Modellen.

2 Vergleichsbeispiel von Hard- und Software Partitionierung

Metriken sind notwendig, um Kosten und Leistung eines Systems je nach Hardware- oder Softwareimplementierung quantitativ zu vergleichen. Hierzu wird anhand einer überschaubaren Funktion, die sowohl in Hardware als auch in Software realisiert werden kann, nach Ansätzen für Metriken gesucht. Betrachtet wird dazu ein numerisches Lösungsverfahren für Differentialgleichungen (DGLs). Einerseits stellt das Lösen von DGLs ein häufiges Teilproblem in der Informatik dar, andererseits lassen sich die Eigenschaften der verschiedenen Implementierungen durch die vorkommenden iterativen und parallelen Berechnungsschritte besonders gut veranschaulichen. Die lineare DGL 1. Ordnung: $y^{'}(x) = y(x)$ bei der der Betrag der Änderung gleich dem Funktionswert selbst ist und deren analytische Lösung der Exponentialfunktion e^x entspricht, stellt zusammen mit dem Startpunkt: $y(x_0) = y_0$ ein typisches Anfangswertproblem dar. Vorgestellt wird hier die numerische Lösung mit dem expliziten Eulerverfahren [7]. Die dazugehörigen Approximationsvorschriften lauten:

$$x_{n+1} = x_n + \Delta x \quad mit \quad x \leq a \qquad und \qquad y_{n+1} = y_n + \Delta x \cdot y_n = y_n \cdot (1 + \Delta x)$$

Neben den beiden Startwerten bestimmt die Schrittweite Δx zur Festlegung der Rechengenauigkeit sowie die rechte Intervallgrenze a als Endbedingung für den Algorithmus notwendig. Liegen dem Verfahren sämtliche Eingabeparameter vor, soll der neue Wert y_{n+1} nach jedem Iterationschritt als Zwischenergebnis berechnet werden. Es folgt nun die Betrachtung der Realisierungsvarianten dieser Rechenvorschriften. Bei der Auswahl des konkreten "Reduced Instruction Set Computer" (RISC) für die Softwareimplementierung als auch bei dem, für die Hardwaresynthese gewählten "Field Programmable Gate Array" (FPGA), wurde aufgrund der Einfachheit des Beispiels bewusst auf aktuelle *High-End* Produkte verzichtet. Stattdessen fiel die Wahl auf besonders kostengünstige und stromsparende Zielplattformen [6, 22] .

2.1 Softwarerealisierung für einen RISC

Der ARM7 [12] ist eine 1994 auf den Markt gebrachte 32Bit-Prozessorfamilie und ist bis Heute eingebetteten Systemen in hohen Stückzahlen im Umlauf (ca. 10 Milliarden [6]). Es gibt sie als Prozessoren oder als "Intellectual property" (IP), also zum Beispiel als Quellcode für Hardware Beschreibungssprachen zur Synthese auf SoCs. Die Realiserung des Beispielalgorithmus in C-Quellcode und die daraus resultierenden Assemblerbefehle des ARM7-GNU C-Compilers mit höchster Optimierungsstufe werden in Abbildung 4 gezeigt. Die Anzahl dieser Befehle sind ein Maß für die Ausführungsgeschwindigkeit des Entwurfs. Die *volatile*-Deklaration des Ausgangssignals sorgt dafür, dass der Compiler die *y_out* Variable nicht auf Grund von Optimierungen auf ein internes Register abbildet, sondern bei jedem Zugriff mit dem Speicher abgleicht. Als Teilkomponente ist ein ARM7-Prozessor somit in der Lage, die Anfangswerte zum Beispiel über einen Datenbus einzulesen, mit diesen unabhängig zu arbeiten und die Teilergebnisse ebenfalls über den Datenbus zurückzuschreiben.

Zum Vergleich der Leistung und Kosten mit anderen Soft- oder Hardwarelösungen werden die folgenden Parameter betrachtet:

- Entwicklungszeit und -aufwand durch Quellcodekomplexität: ca. 15 Zeilen
- Verarbeitungsgeschwindigkeit durch Laufzeit mit 100 Iterationen: $\frac{Anzahl\,Instruktionen}{Taktfrequenz} = \frac{9+7\cdot 100}{100 MHz} = 7{,}09\ \mu s$

Die in der Rechnung verwendete Taktfrequenz ist ein Durchschnittswert. Die tatsächliche Frequenz schwankt zwischen 50 und 200 MHz und hängt davon ab, welche Standardzellen-Bibliotheken und

```
int a_in, x0_in, dx_in, y0_in;
volatile int y_out;

int main( void )
{
  int a = a_in;
  int x = x0_in;
  int dx = dx_in;
  y_out = y0_in;

  while (x <= a) {
    x = x + dx;
    y_out = y_out * ( 1 + dx );
  }
  return 0;
}
```

```
main:
    ldr r3, a_in
    ldr r0, [r3, #0]
    ldr r3, x0_in
    ldr r2, [r3, #0]
    ldr r3, dx_in
    ldr r1, [r3, #0]
    ldr r3, y0_in
    ldr r3, [r3, #0]
    b   .while
.loop:
    add ip, r1, #1
    mul r3, ip, r3
    ldr ip, y_out
    add r2, r2, r1
    str r3, [ip, #0]
.while:
    cmp r2, r0
    ble .loop
    bx  lr
```

Abbildung 4: Software-Implementierung des Lösungsverfahrens in C und Assembler. Die Parameter- und Variablenbezeichnungen entsprechen den zuvor aufgestellten Approximationsvorschriften

Optimierungsoptionen bei der Synthese des Prozessors verwendet wurden [12]. Weiterhin muss berücksichtigt werden, dass nicht alle Instruktionen genau einen Taktzyklus dauern. Somit gilt die ermittelte Laufzeit als Abschätzung nach unten (*best case*).

2.2 VHDL-Modell für eine SoC-FPGA-Implementierung

Als Zielplattform dient hier die Spartan-3E Familie [22], die speziell für einen geringen Stromverbrauch und günstige Stückpreise entwickelt wurde. Weiterhin unterstützen diese FPGAs die Integration und Erprobung von zukünftigen Partitionierungsentwürfen durch die zusätzliche Verwendung von Softcore-RISC-Prozessoren, die auf das SoC-FPGA synthetisiert werden können. Die Umsetzung des Beispielalgorithmus wurde in der Hardwarebeschreibungssprache "Very High Speed Integrated Circuit Hardware Description Language" (VHDL) durchgeführt (Abbildung 5). Aus Gründen der Kompaktheit wurden Daten- und Steuerpfad zusammen in einem getakteten VHDL-Prozess modelliert. Ziel dieses Entwurfs ist es, zur Vergleichbarkeit mit der Softwarelösung, ein nach Außen äquivalentes Verhalten zu erreichen. Die über einen Datenbus nacheinander eingelesenen Eingabeparameter empha, x, dx und y sowie das herausgeschriebene Ergebnis emphy können hier auf die selbe Weise über die bidirektionale Schnittstelle *bus_inout* gelesen und geschrieben werden. Der interne Systemzustand *state* wird über einen Zustandsautomaten durch das 3Bit *state* Signal gesteuert, welches gleichzeitig den Tristatetreiber steuert.

Bei dieser VHDL-Lösung ergeben zum Vergleich die folgenden Parameter:

- Entwicklungszeit und -aufwand durch Quellcodekomplexität: ca. 45 Zeilen
- Verarbeitungsgeschwindigkeit durch Laufzeit mit 100 Iterationen: $\frac{Anzahl\,Taktzyklen}{Taktfrequenz} = \frac{4+100}{60MHz} =$ 1,73 μs

```
LIBRARY ieee;
  USE ieee.std_logic_1164.ALL;
  USE ieee.std_logic_signed.ALL;

ENTITY dgl_solver IS
    PORT ( bus_inout : INOUT
           std_logic_vector(31 DOWNTO 0);
           clk,reset : IN std_logic)
END dgl_solver;

ARCHITECTURE behavior OF dgl_solver IS
    SIGNAL a, x, dx, y :
    std_logic_vector(31 DOWNTO 0);
    SIGNAL state :
    std_logic_vector(2 DOWNTO 0);
BEGIN
  PROCESS (clk, reset) BEGIN

  END PROCESS;
  bus_inout <= y WHEN state(2) = '1'
               ELSE (OTHERS => 'Z');
END ARCHITECTURE behavior;
```

```
IF (reset = '1') THEN
  a <= (OTHERS => '0');
  x <= (OTHERS => '0');
 dx <= (OTHERS => '0');
  y <= (OTHERS => '0');
  state <= (OTHERS => '0');
ELSIF(clk'EVENT AND clk = '1')THEN
  CASE state IS
    WHEN "000" => a <= bus_inout;
                  state <= "001";
    WHEN "001" => x <= bus_inout;
                  state <= "010";
    WHEN "010" => dx <= bus_inout;
                  state <= "011";
    WHEN "011" => y <= bus_inout;
                  state <= "1--";
    WHEN OTHERS =>
    IF ( x < a ) THEN
      x <= x + dx;
      y <= y * ( 1 + dx );
    END IF;
  END CASE;
END IF;
```

Abbildung 5: Implementierung des Lösungsverfahrens in VHDL. Die Signalbezeichnungen entsprechen den zuvor aufgestellten Approximationsvorschriften

Die in der Rechnung verwendete Taktfrequenz wurde vom dem Synthesewerkzeug: *Xilinx ISE* entsprechend des kritischen Pfades, der hier durch den Multiplizierer und Addierer entsteht, und den Spartan-3E spezifischen Einstellungen ermittelt. Es ist festzustellen, dass Entwicklungszeit und -aufwand um den Faktor 3 größer sind als bei der Softwarerealisierung. Dagegen ist die Verarbeitungsgeschwindigkeit etwa um das 4-fache schneller. Liegt also ein laufzeitkritischer und datenunabhängiger Algorithmus vor, so fällt die Wahl für die Implementierung in Hardware aus. Im Gegenzug sind Algorithmen mit vielen datenabhängigen Fallunterscheidungen vorzugsweise in Software zu realisieren und ggf. so lange zu optimieren, bis die Anforderungen eingehalten werden können. Eine Auswahl von weiteren Metriken für eingebettete Systeme aus [14] und [4] lautet:

- **Leistungsaufnahme:** Insbesondere bei batteriebetriebenen Systemen gilt es, die Betriebsdauer zu maximieren.
- **Ressourcenbedarf:** Gewicht, Fläche (u.a. Anzahl der Pins) der Hardware sind zur Kostenreduzierung zu minimieren.
- **Wartbarkeit:** Neben der Quellcodekomplexität, gilt es, das System funktional zu erweitern als auch Testbarkeit und Fehlertoleranz zu garantieren.

Diese Metriken werden jeweils mit Koeffizienten gewichtet, um auszuwählen wie stark ein Kriterium in die Entwurfsentscheidung mit einfließen soll.

3 Automatisierte Verfahren zur Partitionierung

Ansätze und Algorithmen zur Automatisierung des Partitionierungsvorgangs eines ganzen Systems stehen in diesem Abschnitt im Vordergrund. Hier handelt es sich im Kern um eine Variation des aus

der Mathematik und der theoretischen Informatik stammenden *Partitionierungsproblems* [19]: Nehme man zuerst an, dass die gesamte Funktionalität eines Systems in n logische Funktionsblöcke $\{b_0, b_1, \ldots, b_{n-1}\}$, zum Beispiel durch *divide and conquer* Algorithmen [16]), aufgeteilt werden kann. Nehme man weiter an, dass jeder dieser Blöcke bereits identifiziert wurde und tatsächlich für jede der insgesamt m verfügbaren Hardware- und Softwareplattformen implementierbar ist. Dann folgt daraus eine theoretisch maximale Anzahl von m^n realisierbaren Kombinationen. Zum Auffinden der einen optimalen Partitionierung, sofern diese überhaupt Eindeutig ist, entsteht ein klassisches Extremwertproblem. Hierzu seien

$$K : \{i | i \in \mathbb{N}_0 \wedge i < m^n\} \to \mathbb{N} \quad und \quad L : \{j | j \in \mathbb{N}_0 \wedge j < m^n\} \to \mathbb{N}$$

zwei Abbildungen, die jeder der m^n Partitionierungen jeweils einen Wert nach einer zuvor definierten Metrik [21, 20] zuweisen, der den **K**osten bzw. der **L**aufzeit des auf diese Weise gefertigten Systems mit der jeweiligen Partitionierung entspricht. Dann lautet eine Formulierung der Aufgabenstellung: Bestimme jene Partitionierung(en), für die K und L jeweils unterhalb der anwendungsspezifischen Anforderungsgrenzen liegen und die Summe $K + L$ minimal wird. Hierbei handelt es sich um eine *NP-Schwere* Problemstellung, deren eindeutige Lösung also gemäß ihrer Komplexitätsklasse algorithmisch in Abhängigkeit von n nicht effizient berechenbar ist. Selbst wenn es nur um die Frage geht, ob nach Hardware oder Software ($\to m = 2$) partitioniert wird, wächst der Lösungsraum exponentiell mit der Größe des Systems. Bei den im folgenden vorgestellten Verfahren handelt es sich daher um Heuristiken, die mit einer kürzeren Laufzeit eine hinreichend gute Lösung finden.

3.1 Hierarchisches Clustering

Bei diesem Verfahren bilden die n Funktionsblöcke (*kursiv* dargestellt) und m Implementierungsplattformen (**fett** dargestellt) zusammen einen ungerichteten, gewichteten Graphen. Zu Beginn stellt jeder Knoten eine eigene Partition (Cluster) dar. Die Kantengewichte entsprechen dabei einem numerischen Wert, dessen Größe als Metrik für die Abhängigkeit dient. Je höher der Wert desto sinnvoller und wünschenswerter ist demnach eine Zusammenlegung der Teilkomponenten in eine gemeinsame Partition. Ist die Gruppierung zweier Knoten prinzipiell nicht realisierbar oder auf keinen Fall erwünscht, werden die Kanten mit 0 gewichtet und werden für die Darstellung weg gelassen. Bei jeder Zusammenlegung werden die zusammenfallenden Kanten auf je eine Kante zu jedem Nachbarn reduziert und diese mit dem arithmetischen Mittel der vorigen Werte neu gewichtet. Als Endbedingung wird ein Grenzwert für die Kantengewichte festgelegt. Gibt es keine verbleibenden Kanten mehr, die diesen Grenzwert überschreiten, terminiert der Algorithmus und die aktuelle Partitionierung entspricht dem Endergebnis.

Zur Veranschaulichung soll ein einfaches Beispielszenario dienen (Abbildung 6). Ausgangssituation ist die Frage, wie die Implementierung eines (Teil-)Systems, welches aus dem Lösungsverfahren für eine DGL des vorigen Kapitels und einem Bildverarbeitungsalgorithmus zur Hinderniserkennung besteht, in Hardware auf einen FPGA und in Software für einen RISC partitioniert werden soll. Da der Hardwareentwurf für das Lösen der DGL bereits bekannt ist, soll diese Funktion eher auf dem FPGA realisiert werden ($\to$ Kantengewichtung 25 zu 4). Entsprechend anders verhält es sich bei der Funktionalität der Hinderniserkennung für die noch keine genauen Kriterien vorliegen. Bekannt ist lediglich, dass die Verarbeitung und Auswertung vieler Sensorwerte umfangreiche Logikoperationen erfordert, was für die Bevorzugung eines Softwareentwurfs spricht ($\to$ Kantengewichtung 20 zu 8). Weil dazu auch die Fähigkeit des DGL-Lösungsverfahrens genutzt werden kann und soll, werden auch diese Knoten verbunden und hier mit 16 gewichtet.

Beginnend mit dem größten Kantengewicht, werden *DGL-Solver* und **FPGA** im ersten Schritt zusammengefasst. Damit fallen die Kantengewichte auf $\frac{16+8}{2} = 12$ für den *Obstacle-Detector* und $\frac{4+0}{2} = 2$ für den **RISC**. Wieder mit dem größten Kantengewicht beginnend, werden im zweiten Schritt *Obstacle-Detector* und **RISC** gruppiert und die verbleibende Kante entsprechend mit $\frac{2+12}{2} = 7$ neu gewichtet.

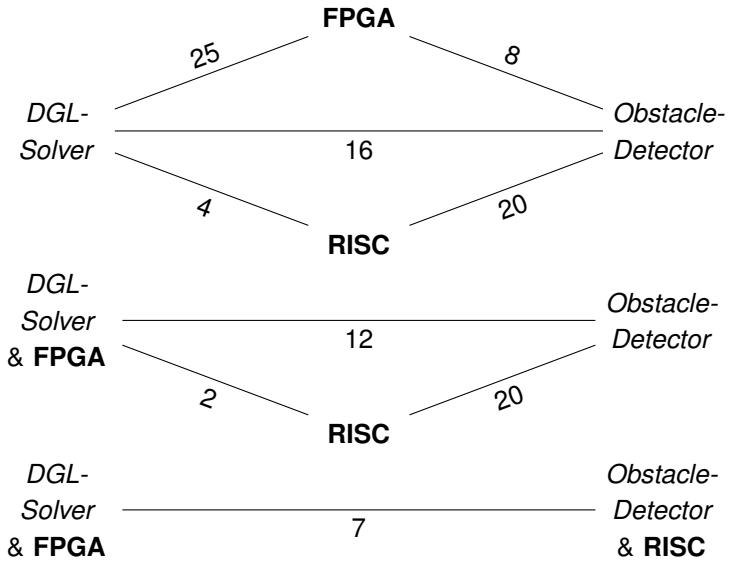

Abbildung 6: Ablauf des hierarchischen Clustering für das Beispielszenario in zwei Schritten in der Reihenfolge von oben nach unten

Liegt dieses letzte Kantengewicht unterhalb des Grenzwertes für das Zusammenfügen von Teilkomponenten, ist der Algorithmus erfolgreich beendet. Sollten die letzten Gewichte noch zu groß sein, müsste das Verfahren mit veränderten Start- oder Zwischengewichten einen oder mehrere Schritte wiederholen. Für das hierarchische Clustering existieren Algorithmen [16] mit quadratischen Laufzeiten von $O((n+m)^2)$. Der Nachteil bei diesem Verfahren ist die unter Umständen sehr schwierige Erfassung von geeigneten Anfangskantengewichten. Weiterhin besteht die Möglichkeit, den Graphen zusätzlich mit Kommunikationskanälen zur Koordination der Teilkomponenten, wie zum Beispiel Bussysteme oder gemeinsame Speicher, zu ergänzen, um somit eine Ausgangsbasis für die Integration der so partitionierten Komponenten zu erlangen.

3.2 Simulated Annealing / Kerninghan-Lin Algorithmus

Das simulierte Abkühlen wurde inspiriert von den Abkühlungsprozessen in der Festkörperphysik, wie sie zum Beispiel bei glühendem Stahl ablaufen. Dabei entspricht der Wärmebegriff per Definition der Geschwindigkeit, mit der sich die Atome in einem Stoff hin- und herbewegen. Dagegen ist die momentane, strukturelle Ausrichtung der Atome untereinander, ein Maß für die Festigkeit eines Stoffes. Nach dem Erhitzen eines Stoffes führt ein langsames Abkühlen, durch das natürliche Bestreben der Atome nach der energetisch günstigsten Struktur, somit zu einer massiveren Festigkeit als es bei einer zu schnell abgekühlten, plötzlich eingefrorenen Atomstruktur der Fall ist. Die Temperatur stellt sozusagen ein Maß dafür dar, wie wahrscheinlich energetisch *ungünstige* Atombewegungen stattfinden.

Analog ist es vorstellbar, die Aufenthaltsräume der Atome als Implementierungsplattformen aufzufassen und die sich bewegenden Atome als einzelne Funktionsblöcke, die von einer der Hardware- oder Software-Plattformen auf eine andere hin und zurück verschoben werden können. Der Temperaturverlauf von Heiß nach Kalt ist dann als der zeitliche Verlauf des Verfahrens von Anfang bis Ende zu interpretieren und anstelle der Energetik werden die eigenen Metriken zur Bewertung des Gesamtzustands herangezogen.

Der Algorithmus lautet dann wie folgt:

- Verschiebe die Blöcke von den Plattformen iterativ und zufällig mit hoher Wahrscheinlichkeit immer dann, wenn diese Verschiebung zu einem hohen Kostengewinn führt und lehne eine

solche Verschiebung mit hoher Wahrscheinlichkeit ab, wenn sie zu einem hohen Kostenverlust führen würde.

- Verringere außerdem langsam nach jedem Schritt die Grundwahrscheinlichkeit dafür, dass eine Verschiebung akzeptiert wird. Das bedeutet, dass die Wahrscheinlichkeit für eine kostensteigernde Umgruppierung mit der Zeit immer kleiner wird und schließlich gegen 0 geht.
- Dennoch ist es nicht auszuschließen, dass eine vermeintlich negative Verschiebung trotzdem akzeptiert wird. Dies dient dazu lokale Minima zu überwinden (Abbildung 7). Das Verfahren terminiert, sobald die Grundwahrscheinlichkeit einen definierten Wert unterschreitet.

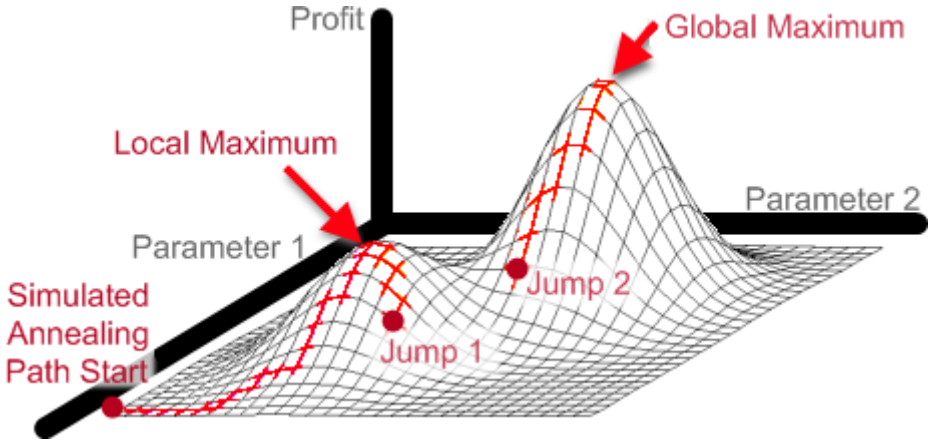

Abbildung 7: Optimierungsablauf mit Simulated Annealing durch Überspringen von lokalen Extrema [2]

Bei dem rein deterministischen Kerninghan-Lin Algorithmus [10] modifiziert man das Verfahren dahingehend, dass die verwendeten Zufallskomponenten, die beim Umgruppieren der Funktionsblöcke verwendet werden, durch festgelegte Metriken ersetzt werden und verbietet man dazu das mehrfache Verschieben einer Komponente. Anstelle von zufälligen Gruppierungen, werden alle Komponenten einmalig und nacheinander gemäß des größten Kostengewinns, zunächst *versuchsweise* verschoben. In einer Tabelle merkt man sich dabei für jeden Schritt, wie die Kosten der aktuellen Partition wären, hätten die bisherigen Umgruppierungen tatsächlich stattgefunden. Der Algorithmus wählt nun iterativ solange die günstigsten Partitionen aus, bis keine neue Partition mit geringeren Kosten mehr gefunden werden kann. Dieses robuste Verfahren besitzt eine Zeitkomplexität von $O(m \cdot n^3)$ und wird bereits zum Entwurf moderner, hochintegrierter Systeme genutzt: In [10] beträgt die Dauer für die automatische Partitionierung eines eingebetteten, echtzeitfähigen Stereobildanalysesystems mit 4000 Zeilen VHDL-Quellcode in 30 Iterationsschritten etwa 2 Minuten.

3.3 Evolutionäre Algorithmen

Nach dem Vorbild der Populationen aus der biologischen Evolution wird hier zur Partitionierung eine Menge von Partitionen erzeugt. Diese werden iterativ durch Auswahlverfahren verbessert, in dem sie sich an ihre Umgebung, hier die Randbedingungen und Anforderungen des Systems, stetig anpassen. Die typischen Auswahlverfahren sind:

- Selektion: Steuert die Suchrichtung der Evolution durch das Gewichten der Metriken.
- Kreuzung: Trennt die Partitionen (eng verknüpfte seltener als locker verbundene) und rekombiniert sie mit dem Ziel, dass aus den guten Eigenschaften zweier Teile ein neues Teil mit verbesserten Eigenschaften resultiert.

- Mutation: Erzeugt zufällige Variationen, um gegebenenfalls starre oder über-spezialisierte Partitionen erneut weiter entwickelbar zu machen.

Evolutionäre Algorithmen gehören zu den Standardverfahren der kombinatorischen Optimierung und können für Probleme eingesetzt werden, für die es keine anderen Lösungsverfahren gibt, wie es zum Beispiel in der Luftfahrt zur Ermittlung von Tragflächenprofilen der Fall ist. Allerdings lassen sich Laufzeit und Qualität der Lösung im Vorfeld aufgrund des hohen Zufallsanteils kaum abschätzen.

4 Fazit

4.1 Zusammenfassung

Ein Embedded-System-Designer sieht sich stets vor der Aufgabenstellung, die Entwicklungskosten- und Rechenleistungsrahmenbedingungen beim Entwurf für ein Projekt einhalten zu müssen. Dazu ist es fast immer erforderlich, für die einzelnen Teilkomponenten des Systems zu entscheiden, ob und wie diese auf die verfügbaren Hardware- und Softwareplattformen zu partitionieren sind. Daraus ergeben sich folgende Teilaufgaben:

1. Zerlegung der Gesamtfunktionalität des System in Funktionsblöcke.
2. Ermittlung der Implementierungsvarianten für jeden Block.
3. Auswahl und Vergleich der Partitionierungen gemäß gewichteter Metriken.
4. Entwurf von geeigneten Kommunikationskanälen, welche die Partitionen verbinden.

Zur anschließenden Optimierung des Systems müssen die Teilschritte mehrfach wiederholt werden und verzahnt ablaufen. Der damit verbundene Arbeitsaufwand verlangt nach automatisierten Verfahren. Eine mathematisch exakte Lösung ist aufgrund der *NP-Schwere* des Partitionierungsproblems unter realistisches Bedingungen nicht effektiv berechenbar. Bei den hier vorgestellten Algorithmen handelt es sich daher um heuristische Verfahren mit unterschiedlichen Laufzeiten, deren Einsatz je nach Anwendungsfall abzuwägen ist. Als Parameter sind je nach geforderter Genauigkeit unterschiedlich viele definierte Metriken zu übergeben, nach denen die Verfahren eine qualitative und quantitative Bewertung der Partitionierungen vornehmen können. Beim *Hierarchischen Clustering* können die Entwickler ihre intuitiven Erfahrungswerte in Form von Empfehlungen durch das Festlegen der Kantengewichte, an den Algorithmus weitergeben.

4.2 Ausblick

In dem aktuellen "Sensor Controlled Vehicle" (FAUST-SCV) Forschungsschwerpunkt, wird ein fahrerloses Transportsystem als Versuchsträger für passive und aktive Fahrerassistenzfunktionen entwickelt [5]. Dabei werden die Algorithmen für die Laserscanner gestützte Navigation sowie für die Kursführung teils in Hardware und teils Software implementiert. Im Rahmen des weiterführenden HPEC-Projektes (Abbildung 2) ist geplant eines der vorgestellten Hardware-Software Partitionierungsverfahren zur Erprobung für dieses Transportsystem auszuwählen, zu realisieren und zu bewerten. Dies soll zum besseren Verständnis der Bedeutung und Auswirkung der Metriken beitragen. Das größte Risiko dabei besteht in der Herausforderung die Bewertungsfunktionen und Metriken von den Zielplattformen, auf denen die Teilkomponenten implementiert werden, unabhängig zu halten. Denn das Verhalten von Hard- und Software unterscheidet sich signifikant zwischen den eingesetzten SoC-FPGAs und Prozessoren [19].

Weiterhin ist kaum abzuschätzen, welche zusätzlichen Problemstellungen sich bei der Software-Thread-Partitionierung mit einem Mehrkern-Echtzeitbetriebssystem ergeben können. Um dieses (Teil-) Aufgabengebiet isoliert von der übergeordneten Hardware-Software Partitionierung zu untersuchen, ist vorgesehen, verschiedene Konfigurationen mit mehreren Microblaze-Prozessoren zu erforschen [9], die in den aktuellen FAUST-Systemen genutzt werden. Da die in der Einleitung angesprochene *Xilinx Extensible Processing Platform* [18] zum Beginn der weiterführenden Projekte voraussichtlich noch nicht erhältlich ist, bieten sich für die parallele Einarbeitung in die ARM Cortex™-A9 Prozessoren, Entwicklerboards wie das *PandaBoard* [17] an, dessen Schaltplän frei verfügbar sind und über zahlreiche Peripherieanschlüsse wie USB, Ethernet, WLAN, HDMI, DVI und einen speziellen Kamera-Anschluss verfügt. Im Anschluss soll der Aufbau vom Technologie-Kompetenzen sowie Entwicklungs-*KnowHow* im Bereich der eingebetteten Systeme und die Vertiefung in Bereich der symmetrischen Multiprozessorsysteme, deren aktuelle Bedeutung für die Leistungssteigerung von Software noch weiter zunehmen wird, erarbeitet werden.

Literatur

[1] BORDASCH, Heiko: *Multiprozessor System on Chip*. Hochschule für Angewandte Wissenschaften Hamburg: Seminar Anwendungen 1 Wintersemester 2009/10. 2009. – URL http://users.informatik.haw-hamburg.de/~ubicomp/projekte/master09-10-aw1/bordasch/bericht.pdf. – Zugriffsdatum: 2.10.2010

[2] DAMA, Max: *Trading Optimization: Simulated Annealing*. Website. 2008. – URL http://www.maxdama.com/2008/07/trading-optimization-simulated.html. – Zugriffsdatum: 21.10.2010

[3] EBERT, Christof ; SALECKER, Jürgen: *Embedded Software-Technologies and Trends*. IEEE Computer Society. 2009

[4] ERNST, R. ; HENKEL, J. ; BENNER, T.: *Hardware-software cosynthesis for microcontrollers*. IEEE Design and Test of Computers. 1994

[5] FAUST-PROJEKT: *Sensor Controlled Vehicle*. Website. 2011. – URL http://www.informatik.haw-hamburg.de/faust.html. – Zugriffsdatum: 25.01.2011

[6] HARDWARE-AKTUELL: *Systeme mit RISC CPU*. Website. 2010. – URL http://www.hardware-aktuell.com/lexikon/Reduced_Instruction_Set_Computing. – Zugriffsdatum: 03.11.2010

[7] HARTMANN, Peter: *Mathematik für Informatiker*. Vieweg, 2006. – ISBN 3-8348-0096-1

[8] HAUCK, Scott ; DEHON, Andre: *Reconfigurable Computing: The Theory and Practice of FPGA-Based Computation*. Morgan Kaufmann, 2007. – ISBN 978-0123705228

[9] HUERTA, Pablo ; CASTILLO, Javier ; PEDRAZA, Cesar ; CANO, Javier ; MARTINEZ, Jose I.: *Symmetric multiprocessor systems on FPGA*. IEEE Computer Society. 2009

[10] KAZUBIAK, Jens: *Automatisierte Hardware-Software Partitionierung am Beispiel eines eingebetteten, echtzeitfähigen Stereobildanalysesystems in Kraftfahrzeugen*, Otto-von-Guericke-Universität Magdeburg - Fakultät für Elektrotechnik und Informationstechnik, Dissertation, 2008. – URL http://diglib.uni-magdeburg.de/Dissertationen/2008/jenkaszubiak.pdf. – Zugriffsdatum: 28.09.2010

[11] KOLBE, Felix: *System-on-a-Chip Konzepte für die Telemetrie- und Steuerungskomponenten in einem Formula Student Rennwagen*. Hochschule für Angewandte Wissenschaften Hamburg: Seminar Anwendungen 1 Wintersemester 2008/09. 2008. – URL http://users.informatik.haw-hamburg.de/~ubicomp/projekte/master08-09-aw1/kolbe/bericht.pdf. – Zugriffsdatum: 5.10.2010

[12] LIMITED, ARM: *ARM7 Processor Family*. Website. 2010. – URL http://www.arm.com/products/processors/classic/arm7. – Zugriffsdatum: 18.10.2010

[13] MARTIN, Grant ; BAILEY, Brian ; PIZIALI, Andre: *ESL Design and Verification: A Prescription for Electronic System Level Methodology*. Morgan Kaufmann, 2007. – ISBN 978-0123735515

[14] MARWEDEL, Peter: *Embedded System Design*. Springer, 2005. – ISBN 978-0387292373

[15] MIGNOLET, Jean-Yves ; WUYTS, Roel: *Embedded Multiprocessor Systems-on-Chip Programming*. IEEE Computer Society. 2009

[16] MÖLLER, Dietmar P. F.: *Vorlesungsfolien: Hardware/Software Co-Design (HSCD)*. Universität Hamburg - Fachbereich Informatik. 2007. – URL http://www.informatik.uni-hamburg.de/TIS/files/VL_HSCD_Modul_MV4.8_2_D.pdf. – Zugriffsdatum: 14.10.2010

[17] PANDABOARD.ORG: *PandaBoard*. Website. 2011. – URL http://pandaboard.org/. – Zugriffsdatum: 05.02.2011

[18] SANTARINI, Mike: Xilinx Architects ARM-Based Processor-First, Processor-Centric Device. In: *Xcell Journal* (2010), März, S. 6–11

[19] SASS, Ronald ; SCHMIDT, Andrew G.: *Embedded Systems Design with Platform FPGAs: Principles and Practices*. Morgan Kaufmann, 2010. – ISBN 978-0123743336

[20] TEICH, Jürgen ; HAUBELT, Christian: *Digitale Hardware/Software-Systeme: Synthese und Optimierung*. Springer, 2007. – URL http://www.springerlink.com/content/w5r0j2. – Zugriffsdatum: 17.10.2010. – ISBN 978-3540468226

[21] WOLF, Wayne: *Hardware and Software Co-design*. Elsevierr, 2007. – ISBN 0-12-369485-X

[22] XILINX®, Inc.: *Spartan-3E FPGA Family: Data Sheet*. Website. 2009. – URL http://www.xilinx.com/support/documentation/data_sheets/ds312.pdf. – Zugriffsdatum: 19.10.2010