IMS
Institut für Mikroelektronische Systeme
Leibniz Universität Hannover

Masterarbeit

Power Optimization of Register File Accesses in a Hearing Aid Processor Using Genetic Optimization Algorithms

Leibniz Universität Hannover
Fakultät für Elektrotechnik und Informatik
Institut für Mikroelektronische Systeme
Fachgebiet Architekturen und Systeme

erstellt von

René, Weinmann
Matrikelnummer: 389570
geb. 14.08.1989 in Stuttgart, Deutschland

Hannover, 28. Juli 2017

Statement of originality

I hereby confirm that I have written the accompanying thesis by myself, without contributions from any sources other than those cited in the text and acknowledgments. This applies also to all graphics, drawings and images included in the thesis.

_____ Hannover, Fr 28 Juli, 2017

Rene Weinmann

Acknowledgment

Auch gibt es niemanden, der den Schmerz an sich liebt, sucht oder wünscht, nur, weil er Schmerz ist, es sei denn, es kommt zu zufälligen Umständen, in denen Mühen und Schmerz ihm große Freude bereiten können. Um ein triviales Beispiel zu nehmen, wer von uns unterzieht sich je anstrengender körperlicher Betätigung, außer um Vorteile daraus zu ziehen? Aber wer hat irgend ein Recht, einen Menschen zu tadeln, der die Entscheidung trifft, eine Freude zu genießen, die keine unangenehmen Folgen hat, oder einen, der Schmerz vermeidet, welcher keine daraus resultierende Freude nach sich zieht? Auch gibt es niemanden, der den Schmerz an sich liebt, sucht oder wünscht, nur, weil er Schmerz ist, es sei denn, es kommt zu zufälligen Umständen, in denen Mühen und Schmerz ihm große Freude bereiten können. Um ein triviales Beispiel zu nehmen, wer von uns unterzieht sich je anstrengender körperlicher Betätigung, außer um Vorteile daraus zu ziehen? Aber wer hat irgend ein Recht, einen Menschen zu tadeln, der die Entscheidung trifft, eine Freude zu genießen, die keine unangenehmen Folgen hat, oder einen, der Schmerz vermeidet, welcher keine daraus resultierende Freude nach sich zieht?

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

List of Algorithms

Kapitel 1

Einleitung

In den letzten Jahren steigt die Komplexität von Schaltungen kontinuierlich und scheint dabei dem Gesetz von Gordon Moor zu folgen, welches im Jahre 1965 veröffentlicht wurde. Jedoch hat diese Entwicklung auch eine Kehrseite, denn mit steigender Performance steigt auch gleichzeitig der Energiebedarf. Dies hat zur folge, dass nun der limitierende Faktor bei portablen Geräten bei der Stromaufnahme liegt und nicht mehr bei der Performance. Außerdem sind die Materialien für immer größer werdenden Batterien sehr selten und dementsprechend teuer. Aus diesem Grund ist die Verlustleistungsoptimierung von portablen Geräten ein immer wichtigeres Themengebiet und muss deshalb bestmöglich optimiert werden. Bei mobilen Geräten handelt es sich nicht nur um Smartphones oder Smartwatches sondern auch um medizinische Geräte wie in Fall dieser Arbeit um ein Hörgerät. Durch optimierte Verlustleistung ist es Nutzern möglich längere Zeit besser zu hören, welches die Lebensqualität deutlich erhöht. In dieser Arbeit soll die Verlustleistung eines solchen Hörgeräteprozessor minimiert werden, umso eine höhere Laufzeit zu ermöglichen. Hierbei soll die explizit die Verlustleistung von Registerspeicherzugriffen optimiert werden. Es gibt zwei Methoden die Leistungsaufnahme zu optimieren, zum Einen kann Veränderung der Hardware vorgenommen werden, oder zum Andern kann eine Verbesserung durch Software herbei geführt werden. Eine Optimierung der Hardware ist sehr zeitaufwändig und demzufolge teuer. Außerdem ist es bei vielen Geräten nicht möglich eine weitere Verbesserung durch Anpassen der Hardware

sicherzustellen. Aus diesem Grund wird auf eine Optimierung der Software gesetzt. Dies kann insbesondere bei gepipelineten Prozessoren zur Kompilierzeit geschehen. Diese Arbeit zeigt wie die Verlustleistung eines solchen Hörgeräteprozessor mithilfe eines genetischen Algorithmus zur Registerallokation optimiert werden kann.

1.1. Motivation

BLA BLA

1.2. State of the Art

BLA BLA

!Write something

1.3. Aufbau der Arbeit

For a better reading experience of this thesis, let us provide a short overview of the following chapters.

Chapter 2: Fundamentals !Write something

Chapter 3: Architecture !Write something

Chapter 4: Evaluation !Write something

Chapter 5: Conclusion !Write something

Kapitel 2

Grundlagen

Bla bla

2.1. Verlustleistung

Unter Verlustleistung in integrierten Schaltungen versteht man die in den Transistoren umgesetzte Leistung die in Form von Wärme verloren geht. Hierbei wird in statische P_{stat} und dynamische Verlustleistung P_{dyn} unterschieden.

2.1.1. Statische Verlustleistung

Jedes mal wenn eine Kapazität geladen oder entladen werden muss, entsteht eine dynamische VerlustleistungP_C. Die zweiter dynamische VerlustleistungP_SC entsteht beim Schaltevorgang von Transistoren insbesondere von Inverter-Schaltungen. In diesem Fall existiert beim Umschalten eine kurze Zeitspanne in der beide Transistoren durchgeschaltet sind. In diesem Fall besteht ein Kurzschlussstrom I_{SC}zwischen Versorgungsspannung V_{DD} und GroundV_{SS}. Die dynamische Verlustleistung ist proportional zur Schaltaktivitätα und somit auch zur Taktfrequenz f'.

2.1.2. Dynamische Verlustleistung

Sobald die Verlustleistung unabhängig von der Taktrate wird, kann diese als statische P_{Stat} bezeichnet werden. Dies ist der Fall, wenn die Transistoren so konzeptioniert sind, dass ein konstanter Strom zwischenV_{DD} undV_{SS}besteht. Dieser Strom ist untabhängig von der angelegten Gatespannung. Der dadurch auftretende Srom wird Leakagestrom genannt. Mit immer kleiner werdenden Strukturen wird dieser Strom deutliche wichtiger. In diesem Fall hängt der Strom von der Gatespannung ab und die statische wird eine dynamische Verlustleistung.

Die gesamte Verlustleistung ist die Summer der drei erwähnten Verlusten.

$$P = P_C + P_{SC} + P_{Stat}$$

2.2. Genetische Algorithmen

Genetische Algorithmen wurden ursprünglich entwickelt um evolutionäre Prozesse aus der Natur nachzuempfinden. Erstmals wurde diese Art von Algorithmen von John Holland 1975 entwickelt und untersucht. In der Natur müssen sich Lebewesen ständig an ihren Lebensraum anpassen und mit den Problemen der Natur leben. Um dies zu ermöglichen haben sich Lebewesen über Jahrtausende an ihre Umgebungen angepasst. Der Aufbau, die Fähigkeiten und das Erscheinungsbild eines Lebewesen ist von Geburt an vorgegeben, diese Information befindet sich in den Chromosomen verschlüsselt. Eine Evolution ist hierbei die Weitergabe dieser Informationen. Durch das decodieren der Chromosomen entsteht eine neue Lebensform. Natürliche Selektion ist hierbei die Anpassungsfähigkeit des Lebewesens an den vorgegebenen Lebensraum. Demzufolge überleben bzw. pflanzen sich nur die Generationen fort, welche sich gut an die Umstände der Umgebung angepasst haben. Die Mechanismen hinter der Evolution sind noch nicht komplett entschlüsselt, jedoch sind einige Verfahren bekannt welche im weiteren betrachtet werden. Durch das Vorbild der Natur sollen mit genetischen Algorithmen schwierige Sachverhalte lösen können. Hierbei stellen die Chromosomen eine Abbildung einer Lösung auf ein Problem dar. Durch eine

sogenannte Fitness-Funktion, kann ermittelt werden wie gut sich ein Chromosom an das gegebene Problem angepasst hat. Wie auch in der Natur müssen werden einzelne Chromosomen fortgepflanzt und bilden neue Generationen. Betrachtet man eine gewisse Anzahl an Generationen so spricht man von einer Population. Zu Beginn des Algorithmus startet man mit zufälligen Chromosomen, bis eine bestimmte Anzahl Generationen entstanden ist. Mit dieser Population kann nun die Fortpflanzung betrieben werden. Durch die Fortpflanzung werden die Chromosomen zweier Lebewesen an eine neue Generation weitergegeben, auch dieser Prozess ist bis heute nicht genau entschlüsselt jedoch können einige Merkmale abgebildet werden. Darunter fällt das Crossover und die Mutation.

2.2.1. Crossover

Bei dem Crossover-Prozess, handelt es sich um die Fortpflanzung von Generationen. Hierbei ist ausschlaggebend welche Generationen miteinander gepaart werden. Auf den ersten Blick scheint es einleuchtend immer die Generation mit der besten Fitness zu paaren. Dies ist jedoch nicht immer sinnvoll, da so schnell ein lokales Minimum erreicht wird. Aus diesem Grund gibt es verschiedene Ansätze um geeignete Eltern für die neue Generation zu finden. Die am weitesten verbreitete Methode ist die Roulette Wheel Selection. Hierbei wird zufällig ein Elternpaar gewählt, wobei die Chance einer jeden Generation ausgewählt zu werden proportional zur Fitness ist. Dieses Verfahren trägt ihren Namen daher, dass es einem Roulette-Rad gleicht, wobei das Rad in Stücke unterteilt ist, welche die Größe proportional zur Fitness haben. Die Auswahl kann nun einem Drehen an einem Rad gleichgesetzt werden. Hierbei ist es wahrscheinlicher, dass die Generationen mit höher Fitness ausgewählt werden. Es ist jedoch auch möglich, dass Populationsmitglieder mit niedriger Fitness gepaart werden.

2.2.2. Mutation

Auch der Prozess der Mutation stammt aus der Natur. Hierbei besteht eine geringe Chance, dass Chromosomen verändert werden und Eigenschaften aufweisen die in keinem der Elternteile aufzufinden sind. Hierzu werden in dem durch Crossover erzeugtem Chromosom einzelne Gene durch Zufall verändert. Die Wahrscheinlichkeit einer zufälligen Veränderung ist hierbei wie in der Natur sehr gering.

2.2.3. Fitness

Die Fitness einer Generation gibt an wie gut sich das Chromosom an das gegebene Problem angepasst hat. Diese Bewertung ist sehr wichtig für die richtige Funktion des Algorithmus. Dabei muss die Fitness-Funktion so entwickelt werden, dass die Generationen unterscheidbar sind und eine Einordnung in der Population möglich ist.

2.3. SIMD Prozessor

2.3.1. VLIW

2.3.2. Pipelining

2.3.3. Scheduling

2.3.4. Hamming-Distanze

2.4. Register Allokation

2.4.1. virtuelle Register

Bei virtuellen Registern handelt es sich um Register die an beliebiger Stelle allokiert werden können, das heisst der Compiler kann selbst entscheiden wo im Registerfile er diese Variable allokiert. Dies hat den Vorteile, dass somit fuer den Code geeignete/optimale Register ausgewaehlt werden koennen. Hierbei kann darauf geachtet werden, dass beide Register-Files gleich ausgelastete sind und dass es moeglich beibt X2- oder MAC-Befehle (werden im naechsten kapitel beschrieben) allokiert werden koennen. Ausserdem koennen die Register in diesem Fall so allokiert werden, dass die Verlustleistungsaufnahme minimiert werd.

2.4.2. X2 Modus

2.4.3. MAC Modus

Kapitel 3

Implementierung

Nach genauerem betrachten der Formel fuer die dynamische Verlustleistung haengt diese von vier Faktoren ab, Spannung, Schaltaktivitaeten, Frequenz und der Lastkapazitaet. Die statische Verlustleistung wird nicht weiter betrachtet, da diese nur durch eine Veraenderung der Hardware verbessert werden koennte und dies ist nicht Teil dieder Arbeit. Da ebenfalls die Spannung und die Frequenz von der Architektur und anderen Funktionen vorgegeben wurde kann auch an diesen Parametern nichts geaendert werden. Im folgenden wird nun erst darauf eingegangen wie die Verlustleistung optimiert werden kann durch minimierung der Schaltaktiviaeten. Im weitern verlauf des Textes wird dann ebenfalls der Einfluss der Lastkapazitaeten ueberprueft. Um in ein Register zu schreiben, muss eine geeignete Adresse an den Adressbus angelegt werden. Da es sich bei der Architektur um 32bit-Register handelt muss hierfuer eine 5-bit Adresse angelegt werden. Dieser Adressbus kann nun auf Schaltaktivitaet optimiert werden. Schreibt beispielsweise eine Instruktion an die Adresse Null und die nachfolgende Instruktion an die Adresse 31, so waere dies der Worst-Case, da in diesem Fall alle 5-Bit umgeladen werden muessten. Befinden sich im Code jedoch virtuelle Register so koennen diese an beliebiger Stelle allokiert und somit die Schaltaktivitaeten der Leitungen verringert werden. Aus diesem Grund wurde zu Beginn eine Heuristik entwickelt bei der die Hammingdistanz der Adressleitungen verringert wird.

3.1. Heuristik

Kapitel 4

Architektur

4.1. Aufbau der Architektur

4.1.1. MIPS Prozessor

4.1.2. Register File Organisation

In dieser Arbeit wird eine Architektur mit einem 64Byte Register verwendet. Die beiden Instruktionsdekoder für die Issue slots 0 und 1 In this thesis an architecture with an 64 byte register file (RF) is used. The two instruction decoders for issue 0 and issue 1 share this flexible register file (RF). In order to enlarge the bottleneck of the register file ports in an VLIW architecture, the register file is divided into two smaller files.

Kapitel 5

Evaluation

Implemented techniques/designs/hardware/software need some sort of validation regarding propriety and performance. The way chosen to establish the correct functionality and/or performance have to be described and justified regarding their suitability.

5.1. Evaluation

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Aenean commodo ligula eget dolor. Aenean massa. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Donec quam felis, ultricies nec, pellentesque eu, pretium quis, sem. Nulla consequat massa quis enim. Donec pede justo, fringilla vel, aliquet nec, vulputate eget, arcu. In enim justo, rhoncus ut, imperdiet a, venenatis vitae, justo. Nullam dictum felis eu pede mollis pretium. Integer tincidunt. Cras dapibus. Vivamus elementum semper nisi. Aenean vulputate eleifend tellus. Aenean leo ligula, porttitor eu, consequat vitae, eleifend ac, enim.

Kapitel 6

Conclusion

This chapter contains concluding remarks whether the desired results could be met or why this wasn't the case. Furthermore future optimizations can be documented here.

6.1. Evaluation

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Aenean commodo ligula eget dolor. Aenean massa. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Donec quam felis, ultricies nec, pellentesque eu, pretium quis, sem. Nulla consequat massa quis enim. Donec pede justo, fringilla vel, aliquet nec, vulputate eget, arcu. In enim justo, rhoncus ut, imperdiet a, venenatis vitae, justo. Nullam dictum felis eu pede mollis pretium. Integer tincidunt. Cras dapibus. Vivamus elementum semper nisi. Aenean vulputate eleifend tellus. Aenean leo ligula, porttitor eu, consequat vitae, eleifend ac, enim.

Anhang A

Source Code

Auszug A.1: Combinational part of the checker for state transition inspection

```
1  ...
2   -- combinational transition checking
3   process (next_state, curr_state, rst_n)
4   begin  -- process
5
6     transition_error <= '0';
7
8     case next_state is
9
10      when INIT =>
11        null;
12
13      when READY =>
14        if curr_state /= INIT
15          and curr_state /= READY
16          and curr_state /= IFG then
17          transition_error <= '1';
18        end if;
19
20      -- we check the next state against the current state
21      when PREAMBLE =>
22        if curr_state /= READY
23          and curr_state /= PREAMBLE then
24          transition_error <= '1';
25        end if;
26
27      when SFD =>
28        if curr_state /= PREAMBLE then
29          transition_error <= '1';
30        end if;
31
32      when DEST =>
33        if curr_state /= SFD
34          and curr_state /= DEST then
35          transition_error <= '1';
36        end if;
37
38      when SRC =>
39        if curr_state /= DEST
40          and curr_state /= SRC then
```

```
41            transition_error <= '1';
42          end if;
43
44        when PAYLOAD =>
45          if curr_state /= SRC
46            and curr_state /= PAYLOAD then
47            transition_error <= '1';
48          end if;
49
50        when PAD =>
51          if curr_state /= PAYLOAD
52            and curr_state /= PAD
53            and curr_state /= FLUSH then
54            transition_error <= '1';
55          end if;
56
57        -- flush state can always be accessed in case of tx-error
58        when FLUSH =>
59          null;
60
61        when CRC =>
62          if curr_state /= PAYLOAD
63            and curr_state /= PAD
64            and curr_state /= FLUSH
65            and curr_state /= CRC then
66            transition_error <= '1';
67          end if;
68
69        -- ifg state can always be accessed in case of tx-error
70        when IFG =>
71          null;
72
73        when others => transition_error <= '1';
74      end case;
75
76    end process;
77  ...
```