TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

Farid Shamani

Design of Intellectual Property-Based Hardware Blocks Integrable With Embedded RISC Processors

Julkaisu 1498 • Publication 1498

Tampere 2017

Tampereen teknillinen yliopisto. Julkaisu 1498
Tampere University of Technology. Publication 1498

Farid Shamani

Design of Intellectual Property-Based Hardware Blocks Integrable with Embedded RISC Processors

Thesis for the degree of Doctor of Science in Technology to be presented with due permission for public examination and criticism in Tietotalo Building, Auditorium TB109, at Tampere University of Technology, on the 22nd of September 2017, at 12 noon.

Tampereen teknillinen yliopisto - Tampere University of Technology
Tampere 2017

Doctoral candidate:	Farid Shamani Laboratory of Electronics and Communications Engineering Tampere University of Technology Finland
Supervisor:	Professor Jari Nurmi Laboratory of Electronics and Communications Engineering Tampere University of Technology Finland
Pre-examiners:	Professor Michael Hübner Lehrstuhl für Eingebettete Systeme der Informationstechnik (ESIT) Ruhr-Universität Bochum Germany
	Adjunct Professor Pasi Liljeberg Department of Future Technology University of Turku Finland
Opponents:	Professor Peeter Ellervee Department of Computer Systems Tallinn University of Technology Estonia
	Adjunct Professor Pasi Liljeberg Department of Future Technology University of Turku Finland

ISBN 978-952-15-4009-7 (printed)
ISBN 978-952-15-4014-1 (PDF)
ISSN 1459-2045

Abstract

The main focus of this thesis is to research methods, architecture, and implementation of hardware acceleration for a Reduced Instruction Set Computer (RISC) platform. The target platform is a single-core general-purpose embedded processor (the COFFEE core) which was developed by our group at Tampere University of Technology. The COFFEE core alone cannot meet the requirements of the modern applications due to the lack of several components of which the Memory Management Unit (MMU) is one of the prominent ones. Since the MMU is one of the main requirements of today's processors, COFFEE with no MMU was not able to run an operating system. In the design of the MMU, we employed two additional micro-Translation-Lookaside Buffers (TLBs) to speed up the translation process, as well as minimizing congestions of the data/instruction address translations with a unified TLB. The MMU is tightly-coupled with the COFFEE RISC core through the Peripheral Control Block (PCB) interface of the core. The hardware implementation, alongside some optimization techniques and post synthesis results are presented, as well.

Another intention of this work is to prepare a reconfigurable platform to send and receive data packets of the next generation wireless communications. Hence, we will further discuss a recently emerged wireless modulation technique known as Non-Contiguous Orthogonal Frequency Division Multiplexing (NC-OFDM), a promising technique to alleviate spectrum scarcity problem. However, one of the primary concerns in such systems is the synchronization. To that end, we developed a reconfigurable hardware component to perform as a synchronizer. The developed module exploits Partial Reconfiguration (PR) feature in order to reconfigure itself. Eventually, we will come up with several architectural choices for systems with different limiting factors such as power consumption, operating frequency, and silicon area. The synchronizer can be loosely-coupled via one of the available co-processor slots of the target processor, the COFFEE RISC core.

In addition, we are willing to improve the versatility of the COFFEE core even in industrial use cases. Hence, we developed a reconfigurable hardware component capable of operating in the Controller Area Network (CAN) protocol. In the first step of this implementation, we mainly concentrate on receiving, decoding, and extracting the data segment of a CAN-based packet. Moreover, this hardware block can reconfigure itself on-the-fly to operate on different data frames. More details regarding hardware implementation issues, as well as post synthesis results are also presented. The CAN module is loosely-coupled with the COFFEE RISC processor through one of the available co-processor blocks.

Preface

The work presented here was carried out during 2014-2017 with the Department of Electronics and Communications Engineering at Tampere University of Technology (TUT).

Given the opportunity, I would like to express my deepest gratitude to Prof. Jari Nurmi, who made this possible to complete my M.Sc. degree first, and the Ph.D. degree in the second stage. It was my absolute pleasure to be a member of *Team Nurmi* for more than five years. I would also like to extend my many thanks to Dr. Tech. Tapani Ahonen, who always gave me precious suggestions, comments and feedbacks. It would have never been possible without his friendly support. I also acknowledge Dr. Tech. Jarno M. A. Tanskanen for giving the opportunity to expand my knowledge in other fields of science and technology, as well as the financial support of a portion of this work.

I would laso like to acknowledge the reviewers of this thesis: Prof. Michael Hübner from Ruhr-University Bochum, as well as the Adjunct Prof. Pasi Liljeberg from University of Turku for their invaluable comments to improve the quality of this thesis.

This work was financially supported by the ARTEMIS JU under grant agreement number 295371, as well as the Academy of Finland under contract number 258506 (DEFT: Design of a Highly-parallel Heterogeneous MP-SoC Architecture for Future Wireless Technologies). I would like to gratefully acknowledge the grants supported by the Finnish Cultural Foundation, along with the Jane and Aatos Erkko Foundation under the project Biological Neuronal Communications and Computing with ICT.

Over the last 6 years in Tampere, there was a great number of people who came to my life and left after a while. I learned from many of them, some made me suffer, and some left unforgettable memories of themselves. I would like to extend my appreciation to all my friends who came, who stayed, and who left; especially the ones who became happy with my every little achievement, and stayed right beside me when I was about to fail. Although there are too many of you to name here, I would like to specifically name Payman Aflaki, Milad Mosallaei, Zahra Abbaszadeh, Maede Arvani, Vida Fakour Sevom, Hadis Behzadifar, and Paula Rakowska for the nice moments we shared together.

I would like to express my unutterable gratitude to my family: Masoud, Ada, Saeed, Sepideh, Saghar, and especially my Mother, Soror Zamani, for their unconditional love, support, and kindness. Thank you for standing right beside me through all these tough years. Although we have been living all around the world for the last 15 years, your presences linger in my daily life. Words cannot describe how much I am proud to have such a supportive family.

Many many thanks to my closest relatives who have enlarged my family: my sisters-in-law Nasrin Aminian and Shideh Bakhshi, and brothers-in-law Amir Taheri and Mohsen

Shamani, for giving positive energy, alongside the non-stop love they shared. The same words are extended to my lovely nephews and nieces: Arman, Armin, Ava, Tara, Adrian, Toulu, Roz, Bardia, and Benita.

I lost my father when I just filled four. There are two persons in my life who never let me feel the lack of my father. The first and foremost, my beloved mother, Soror, who sacrificed her own life to raise her children. I do believe that the heaven would not be sufficient to accommodate her in the other life. I wish there would have been a sentence to express how thankful I am. In the second place, I would like to sincerely acknowledge my elder brother Masoud for fulfilling my father's place. All the support he has provided me over the years was the greatest gift anyone has ever given me. Thank you for sacrificing your own life for teaching me how to realize my own potentials.

My extreme appreciation is extended to Ada, as well. Thank you, Ada, for the pure love and your full support during these years. Nothing would have been possible without your unconditional love and support. Words are quite limited to express me deep inside, hence, the least I can do is to dedicate this work to you dear.

I would like to explicitly appreciate Saeed Shamani for being more than just a brother. Although you are sixteen years older than me, you are the best friend in the world that I have ever had, without even a second thought. Back when I was a kid, I swear I could never have imagined that one day we could build such a strong friendly relationship. Do you remember years ago when I was a teenage? I am talking about the time you were my *private teacher* in Mathematics and Physics. Whoever I am today and whatever I have achieved so far, is the fruit of the tree you planted many years ago.

I would like to thank Sepideh for pushing me towards studying at TUT. This would have not been possible without your unlimited encouragements and motivations. I have to admit that I owe you one of the most important achievement of my life.

Thank you, Saghar, for all the love and support you shared, especially when I was in Iran. I will never forget that you would always be there whenever I needed a shoulder to cry on. I spent the best days of my 20s with you.

To the future love of my life, thank you for being disappeared within these rough years and let me progress towards my ultimate goals. Whoever and wherever you are, it is now the time to show yourself up. ☺

31 August 2017, Espoo, Finland
Farid Shamani

Contents

Acronyms

ACK	Acknowledge
ALMs	Adaptive Logic Modules
ALUTs	Adaptive Look-Up Tables
APP	A Posterior Probability
ASIC	Application Specific Integrated Circuit
ATT	Address Translation Table
CAN	Controller Area Network
CP	Cyclic Prefix
CCB	Core Configuration Block
CISC	Complex Instruction Set Computer
CR	Cognitive Radio
CRs	Condition Registers
CRC	Cyclic Redundancy Check
CSMA/CR	Carrier Sense Multiple Access/Collision Resolution
CGRAs	Coarse-Grain Reconfigurable Arrays
DF	Direct Form
DLC	Data Length Code
DMA	Direct Memory Access
DSA	Dynamic Spectrum Access
DSP	Digital Signal Processor
DTLB	Data Translation Look-aside Buffer
EA	Effective Address
EOF	End-of-Frame
EPN	Effective Page Number

FIR	Finite Impulse Response
FPGA	Field Programmable Gate Array
GPP	General Purpose Processor
GPRs	General Purpose Registers
HDD	Hard Decision-based Detection
IFFT	Inverse Fast Fourier Transform
IC	Integrated Circuit
IDE	Identifier Extension
I/O	Input/Output
IP	Intellectual Property
ISA	Instruction Set Architecture
ITLB	Instruction Translation Look-aside Buffer
LAN	Local Area Network
LDPC	Low-Density Parity-Check
LRU	Least Recently Used
LTF	Long Training Field
LTS	Long Training Symbol
MAC	Multiply-Accumulate
ML	MultiplierLess
MMU	Memory Management Unit
MSR	Machine State Register
NIC	Network Interface Controller
NC-OFDM	Non-Contiguous Orthogonal Frequency Division Multiplexing
OFDM	Orthogonal Frequency Division Multiplexing
OOB	Out-of-Band
OPDF	Optimized Parallel Direct Form
OPPDF	Optimized Pipelined-Parallel Direct Form
OS	Operating System
OTDF	Optimized Transposed Direct Form
PAPR	Peak-to-Average Power Ratio

PCB	Peripheral Control Block
PDF	Parallel Direct Form
PID	Process ID
PPDF	Pipelined-Parallel Direct Form
PR	Partial Reconfiguration
PSR	Processor Status Register
RI	Register Insertion
RISC	Reduced Instruction Set Computer
RPN	Real Page Number
RTR	Remote Transmission Request
SDD	Soft Decision-based Detection
SDR	Software Define Radio
SNR	Signal-to-Noise Ratio
SOF	Start-of-Frame
SRR	Substitute Remote Request
STF	Short Training Field
STS	Short Training Symbol
TDF	Transposed Direct Form
TLB	Translation-Lookaside Buffer
UTLB	Unified Translation Look-aside Buffer
VCD	Value Change Dump
VHDL	Very high speed integrated circuits Hardware Description Language
VM	Virtual Memory
VPN	Virtual Page Number
ZPR	Zone Protection Register

List of Publications

Most of the contents of this thesis is based on the following publications in which the author of the thesis is the main author. The publications are referred to as [P. #] in the manuscript. All the following publications are appended at the end of the thesis.

I F. Shamani, R. Airoldi, T. Ahonen, and J. Nurmi, "FPGA Implementation of a Flexible Synchronizer for Cognitive Radio Applications", in *Proceedings of the 2014 Conference on Design and Architectures for Signal and Image Processing (DASIP)*, Madrid, Spain, pp. 1−8, Oct. 2014.

II F. Shamani, V. F. Sevom, T. Ahonen, and J. Nurmi, "FPGA Implementation Issues of a Flexible Synchronizer Suitable for NC-OFDM-Based Cognitive Radios", in *Elsevier Journal of Systems Architecture (SYSARC)*, Nov. 2016.

III F. Shamani, T. Ahonen, and J. Nurmi, "Synchronization in NC-OFDM-Based Cognitive Radio Platforms", in W. Hussain et al. "Computing Platforms for Software-Defined Radio", Springer International Publishing, pp. 189−207, 2017.

IV F. Shamani, V. F. Sevom, J. Nurmi, and T. Ahonen, "Design, Implementation and Analysis of a run-Time Configurable Memory Management Unit on FPGA", in *IEEE Nordic Circuits and Systems Conference (NORCAS): NORCHIP & International Symposium on System-on-Chip (SoC)*, Oslo, Norway, pp. 1−8, Oct. 2015.

V F. Shamani, V. F. Sevom, T. Ahonen, and J. Nurmi, "Integration Issues of a run-Time Configurable Memory Management Unit to a RISC Processor on FPGA", in *Elsevier Journal of Microprocessors and Microsystems (MICPRO)*, Dec. 2016.

VI F. Shamani, V. F. Sevom, T. Ahonen, and J. Nurmi, "FPGA Implementation and Integration of a Reconfigurable CAN-Based co-Processor to the COFFEE RISC Processor", in *IEEE Nordic Circuits and Systems Conference (NORCAS): NORCHIP & International Symposium on System-on-Chip (SoC)*, Copenhagen, Denmark, pp. 1−6, Nov. 2016.

1 Introduction

During the last decade, technology advancements have significantly affected people's lives. Unprecedented growth in science and technology has accomplished modernity. In that respect, a number of digital systems can be found in most of the electronics devices. The main engine of such systems is typically a programmable processor which controls other peripheral electronics components. In the world of *Embedded Systems* and *Systems-on-Chip (SoCs)*, these programmable processors are designed in a way to offer the maximum flexibility, minimum silicon area, fast operating frequency, and low power consumption. On the other hand, they do not necessarily perform well enough to meet some real-time requirements of embedded applications. Hence, the demand of real-time support in embedded applications is become more into picture day by day.

One possible solution is to employ hardware accelerators, which also called Functional Units (FUs), co-processing in parallel with the main processor [1]. In principle, co-processing accelerates the overall performance of the system by offloading computationally intensive task from the main processor. Therefore, components capable of co-processing are usually dedicated Intellectual Property (IP) blocks which can execute a certain function considerably faster than the main processor. Nevertheless, characteristics such as scalability, flexibility, and reconfigurability are some of the prominent features of a good design.

Decades ago, computer architectural designers were encountered to choose between flexibility and performance. If we consider Application Specific Integrated Circuit (ASIC) technology on one end and the General Purpose Processor (GPP) on the opposite end, there was a huge gap between these two design choices. Typically, GPPs are closer to the software side. They can execute any function consisted of several instructions due to the versatility of their instruction set. Thus, they are very flexible to perform any computable task, while their performance is very poor for performing some specific tasks. In principle, an ASIC is designed to perform some specific tasks very fast and efficient. Hence, ASICs has the potential to achieve higher speed, less silicon area, and less power consumption compared to the GPPs on performing a specific task. On the other hand, ASICs are not flexible in terms of architectural changes. Once an ASIC is fabricated, it cannot be altered to perform another application [2]. Therefore, ASICs are typically used for high-volume embedded systems, such as mobile phones. In addition, only few companies can afford to implement their products on ASICs due to the high production costs.

Reconfigurable Computing is intended to fill the gap between ASIC (hardware) and GPP (software) by achieving higher performance than software, while offering more flexibility compared to the hardware [3]. The main idea of employing reconfigurable architectures is back to 1950s where the Gerald Estrin proposed the concept of *Reconfigurable Architecture*

Blocks [4]. The idea was to use the main processor to control and monitor the behavior of an array of reconfigurable hardware blocks. We can consider that the 1980s-1990s was a renaissance in this field of research and several reconfigurable platforms were introduced, including Ramming Machine, Hartenstein's XPuter, and PAM Machine [5]. By the invent of the Field Programmable Gate Array (FPGA) in the mid 1980s and early 1990s, real-time signal processing (such as video and audio signal processing) which was too computationally intensive for microprocessors, came more into picture [6]. Indeed, the FPGAs, as one of the major reconfigurable platforms, could successfully fill the gap between software-oriented and hardware-oriented architectures. Although the FPGAs offer satisfactory performance for most of the embedded applications, the energy consumption of an FPGA-based embedded system should be carefully considered.

1.1 Objective and Scope of the Research

The primary objective of this work is to research the influence of hardware acceleration in the world of embedded systems. In that sense, some reconfigurable architectural solutions and research methods, with their respective hardware implementations suitable for a Reduced Instruction Set Computer (RISC)-based platform are developed. The main motivation, in that respect, is to provide a flexible approach to increase the overall performance of a computing platform, while the efficiency is maximized and power consumption is maintained at an acceptable level. In addition, a reconfigurable platform has the potential to be adapted in different operational environments.

As the case study, we take into consideration an embedded processor named *COFFEE* core for the proof-of-concept. The core is an open-source embedded processor in which reusability and configurability are two prominent characteristics of the core. So far, the core is integrated with several IP blocks as different co-processors to create an embedded platform. For example, the integration of a floating point unit as a co-processor was done in [7]. The floating point co-processor is called MILK. The COFFEE core which is tightly-coupled with the MILK co-processor is named CAPPUCCINO. In another attempt, the COFFEE is integrated with three co-processors to form a programmable baseband receiver platform [8]. In [9], the multi-core version of the COFFEE is used to build a platform suitable for Software Define Radio (SDR). The COFFEE is also used to create a template named CREMA which is based on the principle of Coarse-Grain Reconfigurable Arrays (CGRAs) [10]. More information regarding the COFFEE core is appended at the end of this thesis in Section Appendix A.

Although the COFFEE has exploited different configurations to create various platforms, it still requires more efforts to increase the versatility and application performance of the core. For example, the core is heavily suffering from the lack of a Memory Management Unit (MMU) [11]. As discussed earlier, the ultimate motivation behind this research work is to design several reconfigurable IP blocks loosely/tightly-coupled with the COFFEE processor. However, the integration of the developed IP blocks are not only limited to the target processor; they are integrable with any standard RISC architecture. The developed IP blocks fulfill the general characteristics of a good digital design previously mentioned. In addition, other design parameters, such as energy efficiency, are vastly taken into consideration in the hardware description of each IP block. Indeed, the energy efficiency is very crucial in battery-powered embedded applications. In our research group, we have a way of thinking that a good IP block is worth more than a good design.

The overall content of this thesis is composed of three main sections. In each section, we try to develop a reconfigure architecture specifically designed for a particular application. In the first section, a reconfigurable module which is capable of receiving and decoding one of the potential modulation techniques of the next generation wireless communication is developed. The target modulation is the Non-Contiguous Orthogonal Frequency Division Multiplexing (NC-OFDM) technique, in which the wireless spectrum can be utilized efficiently. Although the NC-OFDM has the potential to alleviate spectrum scarcity to some extent, synchronization in those systems is one of the major problems. In that respect, we propose a reconfigurable architecture which is able to tackle synchronization problem of an NC-OFDM-based system. One of the main characteristics of the synchronizer is the capability of reconfiguring itself during the run-time using Partial Reconfiguration (PR) feature.

In the second section, another reconfigurable architecture is designed to operate as an MMU. One of the main applications of the MMU is to enable virtual-to-physical address translations. In fact, the MMU is developed to improve the overall performance of the processor. The infrastructure of the MMU is based on employing several Translation-Lookaside Buffers (TLBs) simultaneously in different hierarchies. Each TLB has a different size in terms of the number of entries. The MMU reconfigures itself during the run-time to operate on different page sizes, from 1KB to 16MB in the virtual space.

The third section concentrates on another reconfigurable module which is able to operate in the context of Controller Area Network (CAN) protocol. The CAN module is capable of reconfiguring itself while operating to operate on different data frames, i.e. *standard data frame* and *extended data frame*. Furthermore, the CAN module guarantees the integrity of the data in several situations.

1.2 Author's Contribution to the Published Work

The context of this thesis has mainly been extracted from the Publications [P. I] – [P. VI] in which the author was the primary author. Prof. Nurmi supervised the overall work and gave invaluable comments and feedbacks on each of the publications.

Publication [P. I]: The purpose of this paper was to design and develop an IP-based synchronizer to alleviate one of the major problems in next generation wireless communications. We developed a novel method how to perform synchronization in such systems. The hardware implementation and considerations are done by the author. The idea of using Partial Reconfiguration (PR) technique arose by the second author, Dr. Tech. R. Airoldi. The third author, Dr. Tech. T. Ahonen, as well as the fourth author, Prof. J. Nurmi, gave technical comments, along with invaluable feedbacks.

Publication [P. II]: This paper was an extension to our previous work [P. I] with an invitation. In this paper, the first author put his best effort on optimizing the hardware implementation of the mentioned design. As a result, magnificent improvements in terms of maximum operating frequency were achieved, while the hardware resource allocation, alongside the power consumption drastically reduced. This IP block has the potential to be loosely-coupled with the COFFEE RISC processor. This work is mainly done by the first author, while the third author V.F. Sevom assisted in some simulations and verifications. The remaining authors gave very nice comments on overall work.

Publication [P. III]: In this publication, the first author discussed the synchronization problem in next generation wireless communications. The entire work is done by the first author, while Dr. Ahonen and Prof. Nurmi gave the final comments on the work.

Publication [P. IV]: The main contribution of this paper was to develop a run-time configurable memory management unit. The entire hardware implementation was done by the first author. The second author assisted in MatLab computations. Dr. Ahonen and Prof. Nurmi gave technical comments and supervised the entire work.

Publication [P. V]: This publication was an extension to our previous work in [P. IV]. In this paper, the integration issues of the proposed memory management unit with the COFFEE core (tightly-coupled) was investigated in detail. The overall work, including integration and optimizations, were performed by the first author. The second author assisted in some simulations and verifications with a new set of test vectors. The entire work was supervised by Dr. Ahonen, along with Prof. Nurmi.

Publication [P. VI]: In this publication, a reconfigurable Controller Area Network (CAN) protocol was developed which was loosely-coupled with the COFFEE RISC processor. The first author designed and implemented the entire CAN protocol as an IP block on FPGA. The integration with COFFEE processor was done by the first author, as well. The verification of the design was done by the help of the second author. This work was fully supervised by Dr. Ahonen and Prof. Nurmi.

1.3 Thesis Outline

The rest of this thesis is organized as follows. In Chapter 2, the author discusses the synchronization problem in next generation wireless communications. The-state-of-the-art, as well as the hardware implementation issues are widely discussed. Chapter 3 describes the developed memory management unit and its respective technical issues, e.g. how to integrate the MMU with the COFFEE processor. Chapter 4 explains the development of the proposed CAN module on FPGA with its respective integration with COFFEE RISC processor. Eventually, Chapter 5 summarizes the overall research work. Appendix A gives an insight to the COFFEE platform in terms of architecture, design considerations, hardware cost, etc.

2 Reconfigurable IP-Based NC-OFDM Synchronizer Module

The main context of this chapter is extracted from [PI], [PII] and [PIII]. This chapter provides a wide discussion about an alternative solution to cope with spectrum scarcity problem. In this regard, we proposed a flexible Non-Contiguous Orthogonal Frequency Division Multiplexing (NC-OFDM)-based synchronizer which is capable of reconfiguring itself on-the-fly using Partial Reconfiguration (PR) feature. In addition to the state-of-the-art, this chapter provides a deep insight to the hardware implementation which leads to revealing various constraints. Furthermore, the author provides different architectural choices with their respective trade-offs, e.g. using more silicon area to achieve higher operating frequency. The proposed synchronizer has the potential to be loosely-coupled with the COFFEE core as a co-processor.

2.1 Spectrum Scarcity Problem

The trend towards replacing wired systems with wireless devices is getting stronger. Generally speaking, wireless technology is being advanced day after day. The number of wireless users who demand higher data rate is increasing per day [12]. Since the spectrum is a finite resource, increasing number of wireless users is leading to spectrum scarcity problem. It is mainly due to the fact that the frequency bands between 10MHz to 6GHz are most suitable for wireless communications due to the characteristics of the electromagnetic waves. However, this wide frequency band is not sufficient to accommodate all today's wireless users at once [13].

2.2 Cognitive Radio as a Solution

One possible solution to resolve spectrum scarcity problem is to employ Dynamic Spectrum Access (DSA). Conceptually, the DSA is a promising technique which employs unused frequency bands allocated to a licensed user, e.g. TV broadcast, for secondary user activities [14]. These unoccupied frequency bands are also known as *white spaces*. In principle, secondary users willing to transmit within a licensed band are permitted to transmit their data via the available white spaces as long as they do not interfere with the incumbent licensed user. Cognitive Radio (CR) is a device capable of employing DSA technique. In context, the CR is a flexible platform in which most of the baseband processing is performed in programmable processing technologies such as GPP, Digital Signal Processor (DSP), and FPGA [9]. In CR, the transmitter is able to detect which communication channels are occupied and which are not. Indeed, the CR is always aware of its surrounding. For example, Figure 2.1 depicts a real measurement of the spectrum

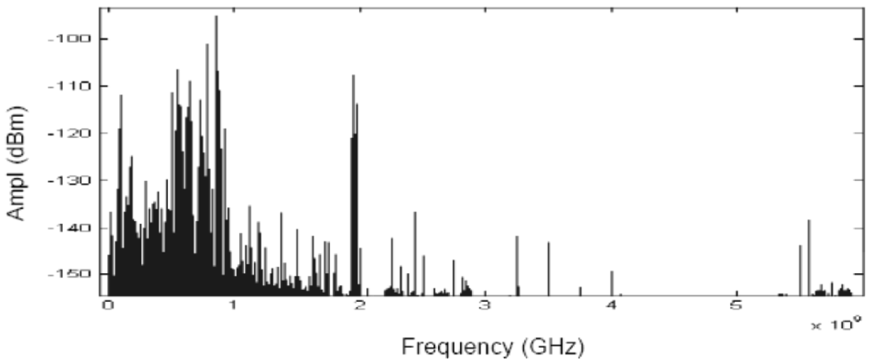


Figure 2.1: A portion of the spectrum allocated to a licensed user. Obviously, the licensed user does not efficiently utilize the spectrum [15, p. 163]

at downtown Berkeley. As it is obviously clear, the licensed user mostly utilizes the first 2GHz of the dedicated 6GHz spectrum, while the rest of the spectrum is not fully utilized. The CR has the ability to instantly employ unoccupied channels within the licensed spectrum for secondary transmissions while avoiding the ones occupied by the primary user. Moreover, the CR should guarantee that there will be no interference with the licensed user's subbands.

2.2.1 NC-OFDM, a Recently Emerged Technology

CRs have still their own challenges including how to sense the spectrum, which data transmission technique is the most efficient one, etc. So far, there have been several modulation techniques to be employed in CR of which the Orthogonal Frequency Division Multiplexing (OFDM) is the most favorable one [16]. The orthogonality between overlapped carriers is one of the major features that discriminates the OFDM from the other modulation techniques [17]. However, the OFDM technique has its own disadvantages such as Peak-to-Average Power Ratio (PAPR), sensitivity to carrier frequency offset, and extra overhead due to using a Cyclic Prefix (CP). In addition, since the OFDM operates in a contiguous spectrum mode, a 5MHz transceiver can only transmit when an idle 5MHz spectrum band is detected. If we take into the assumption that a 5MHz band is occupied by a narrowband transceiver, e.g. 200kHz, the entire 5MHz band is treated as busy. In this case, more than 90% of the bandwidth is being wasted by the narrowband transceiver to transmit at only 200kHz speed [18].

One alternative solution capable of tackling the above-mentioned problem is an upgraded form of the conventional OFDM technique named NC-OFDM. Indeed, the NC-OFDM technique has recently attracted many researchers due to its capability of turning off a subset of subcarriers which are not required for the transmission or those which might cause interferences with the adjacent users. Furthermore, the NC-OFDM has been nominated as a promising candidate for high data rate transmissions in the context of the CR [14]. Potentially, the NC-OFDM can be one of the best candidates towards the next generation wireless communications, 5G [19].

2.2.2 Synchronization, one of the Major Problems

As previously mentioned, NC-OFDM is a reliable candidate to exploit the spectrum more efficiently. Although this technique is able to cope with the spectrum scarcity problem, similar to other techniques, it has its own challenges. One of the primary challenges in NC-OFDM is how to synchronize the receiver with the transmitter. The main reason is that since the NC-OFDM transmitter is able to use any white space within the licensed spectrum (of course, by considering the constraints mentioned such as not interfering with the licensed user's subbands), the secondary user has the potential to be presented in any location within the spectrum. Nevertheless, one challenge is to detect the location of the secondary transmitter within the prime spectrum from the receiver's point of view. In addition, the NC-OFDM transmitter is able to switch off unnecessary subcarriers which result in an alteration to the time-domain representation of the signal. A tiny change in the waveform of a time-domain signal will generate a stochastic location for the preamble, pilot and data carriers. In this regard, the second challenge is to find the exact locations of the mentioned subcarriers in the receiver side.

2.3 Related Works

In principle, there are two well-known methods which have widely been used to synchronize both receiver and transmitter. A very straightforward solution is to dedicate a particular channel (a secondary channel) to reveal the essential synchronization characteristics of the transmitter to the receiver. In the literature, this solution is known as the *Out-of-Band (OOB)* communication. Authors in [20, 21, 22, 23, 24] have studied synchronization in OOB systems and proposed some techniques with respect to the out-of-band synchronization. In this method, since the receiver is always aware of the location of the secondary user in the entire spectrum, the synchronization is potentially similar to that of OFDM. However, the additional hardware cost, along with dedicating a portion of bandwidth to the secondary channel are two limiting factors which limit the functionality of the out-of-band data transmission in some practical situations [25].

In contrast to the out-of-band communications, *In-Band* data transmissions more attract researchers in the field of NC-OFDM systems. In this method, the prerequisite synchronization information is embedded in the same packet sent by the transmitter. It is now the receiver's concern how to extract that information in order to synchronize itself with the transmitter. In this thesis, we have also taken into account how to perform synchronization in an in-band environment. Similar to out-of-band systems, there are some studies regarding the in-band communication which few of them are briefly explained as follows.

In [26], a fractional bandwidth model is proposed in which a new format for the preamble is introduced. In frequency-domain, a specific pseudo-noise is generated in a way that the time-domain representation of the preamble comes with two identical halves with different sign bits. Apart from the OFDM based scheme, the interferences caused by the licensed users have not been fully considered. The authors took into the assumption that the power level of the primary user is lower than the secondary transmitter, while in reality, it is the secondary transmitter which should minimize its transmitting power to prevent interference with the adjacent primary user due to the sidelobe leakages [27].

In [25], the receiver takes A Posterior Probability (APP) algorithm into consideration to discover the active subchannels, while the interference caused by the licensed user is accounted. In the case of detecting an active subchannel, the receiver performs a Hard Decision-based Detection (HDD) algorithm to extract NC-OFDM symbols. A poor performance in a noisy channel, as well as the subchannels closed to the primary user, is one drawback of employing HDD algorithm. Therefore, the receiver takes into account a Soft Decision-based Detection (SDD) to increase the performance in such noisy environments.

Li et al. in [28], tried to improve the methodology introduced in [25]. According to their claim, the system code rate of the proposed algorithm is only 1/4, while only half of the subcarriers are active. They propose a Low-Density Parity-Check (LDPC) to improve system code rate right after the APP is finished. The authors mainly emphasized on how to generate the LDPC code, while the synchronization procedure is not exactly addressed. They have taken into the assumption that the receiver has a perfect solution to synchronize itself with the transmitter.

Saha et al. proposed a blind synchronization method in which the receiver is capable of locally regenerating the time-domain representation of the frequency-domain incoming signal [29, 30]. They also exploit a multiplier-less approach to detect active subcarriers in the frequency-domain. The multiplier-less approach provides an acceptable performance in packet detection stage while omitting a massive number of unnecessary complex computations. Furthermore, by having known the fundamental information regarding the primary user's activities in the spectrum, a binary mask is employed to filter out the primary user's active subcarriers in the frequency-domain. The rest of synchronization is inferred to be similar to the OFDM.

2.4 The State-of-the-art in NC-OFDM Synchronization

As previously stated, since the time-domain representation of the signal in an NC-OFDM-based system is altered, the synchronizer block should find another way to detect the secondary transmission. Although synchronization in an OFDM system can be performed in time-domain, frequency-domain or both, an OFDM receiver is more likely to perform synchronization in time-domain. The main reason is that synchronization in frequency-domain is very computationally expensive in the receiver. However, it seems that an NC-OFDM receiver has no possible solution than to perform synchronization in the frequency-domain, since the shape of the time-domain preamble is altered. In this work, we employ several techniques, e.g. multiplier-less approach, alongside the state-of-the-art features, e.g. partial reconfiguration, in order to keep the receiver synchronized with the secondary transmitter. Moreover, the synchronizer block is carefully designed for different scenarios, for example, when the silicon area is the limiting factor, when power consumption is the main factor, or reaching to the maximum operating frequency is the ultimate target.

2.4.1 The Conventional Method

Figure 2.2 shows the most common synchronization mechanism in an NC-OFDM system. Conceptually, the synchronization is performed in two major sections. First, the subcarrier detection and, second, preamble regeneration and packet detection. As previously studied, the receiver has no prior information about the subbands in which the secondary user is active. Nevertheless, the first step is to collect as much information about the spectrum as possible. Hence, the receiver gathers all the available subbands in the spectrum and starts sensing each of them to detect the active ones, using one of the sensing methods.

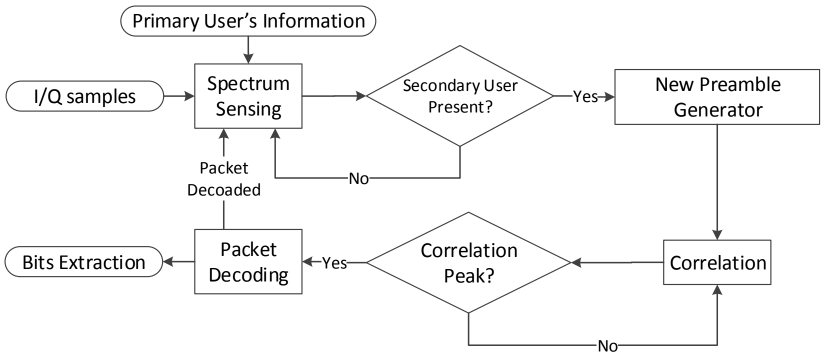


Figure 2.2: The most common synchronization scheme. Reconstructed from [P. I] © IEEE, 2014.

According to the Federal Communications Commission, secondary users willing to transmit in a licensed spectrum may have the preliminary information about the licensed user such as power, location, and signal structure [31]. Having fundamental information about the licensed user, those subcarriers which belong to the licensed user are masked. Once the licensed user is masked, the remaining active subcarriers are considered for the secondary transmitter. The first phase of the synchronization (secondary user detection) is successfully finished at this stage.

Now, it is possible to form the time-domain representation of the secondary signal by using a low-cost Inverse Fast Fourier Transform (IFFT) unit. Once the new preambles are generated, the correlator is fed by recently generated parameters and, subsequently, the maximum similarity between a buffered version of the incoming signal and the generated preamble is investigated. As soon as the correlator finds a peak, the entire packet is delivered to the packet decoding block to extract the data bits. In case that the correlator fails to detect a peak, there is a possibility that the secondary transmitter has terminated the connection on the set of subcarriers detected as active in the previous stage. Therefore, the sensing unit will start sensing the spectrum from the scratch to find active subbands occupied by the secondary transmitter.

2.4.2 The Proposed Synchronizer

The synchronizer proposed in this work follows the same dataflow as Figure 2.2 presents. The contribution beyond the state-of-the-art of this work is to employ a reconfigurable *multicorrelator* to perform both spectrum sensing and correlation on demand. From the architectural point of view, the multicorrelator is able to reconfigure its parameters to perform either autocorrelation function for spectrum sensing or crosscorrelation function for packet detection phase using Partial Reconfiguration feature.

Figure 2.3 illustrates how the multicorrelator operates in an NC-OFDM receiver. In the first phase (secondary transmission detection), the controller block (re)configures the multicorrelator to perform autocorrelation during the run-time to monitor the entire spectrum. Here, the autocorrelation function computes the maximum similarity between the incoming signal and a delayed version of itself. The achieved results are compared with a threshold. The magnitude of the threshold is determined by the receiver during the inter packet time slot. When the channel is idle, the magnitude of the noise is determined. Then, the noise level is subtracted from the overall energy of the signal, leading to an approximate threshold value. However, this method might not be practical in very low Signal-to-Noise Ratio (SNR) regions.

When the presence of the secondary user is detected by the receiver, all active subcarriers (excluding those related to the licensed user) are handed to the preamble regenerator unit to generate a time-domain form of the frequency-domain signals. Meanwhile, the controller reconfigures the multicorrelator using PR feature to perform crosscorrelation function (packet detection phase). Once the multicorrelator is set up with regenerated coefficients, the crosscorrelation is performed to find the boundaries of the packet. Since the crosscorrelation function is a computationally intensive task, Roberto Airoldi proposed a two-step threshold verification mechanism in [32]. This method is further investigated in next sections. Once the second threshold level is met by the Threshold Detector block, the packet is delivered to the Packet Decoding block to extract the data.

2.5 The Infrastructure of the Multicorrelator

The multicorrelator is composed of several fundamental subblocks of which the Finite Impulse Response (FIR) block is the most prominent one. As Figure 2.4 presents, the multicorrelator consist of a Memory block, FIR filter block, Threshold Detector block, and Controller block. Each block is briefly explained in the following subsections.

2.5.1 Memory Block

This block is a simple SRAM which is used to store regenerated coefficients. Each memory word is 32 bits long of which we assigned the first 16 bits for the *Real* part and the remaining 16 subsequent bits represent the *Imaginary* part of the incoming signal. For example, the value 0x1234ABCD in a memory block infers that 0x1234 is the real part and 0xABCD is the imaginary part of the signal, respectively. The main reason why we take 16-bit input signals into consideration is that 16-bit coefficients provide enough accuracy for the FIR filter to perform crosscorrelation [33]. From the hardware implementation point of view, we employed an unsynthesizable SRAM described by the Very high speed integrated circuits Hardware Description Language (VHDL) code for the simulation and a synthesizable IP block provided by Quartus II environment for synthesis purpose, respectively.

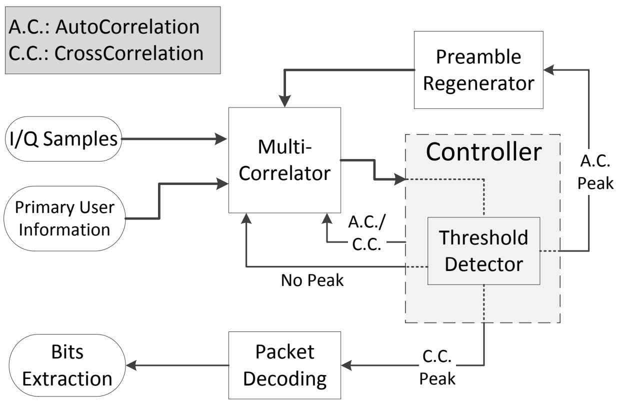


Figure 2.3: The proposed architecture for the synchronizer [P. II] © Elsevier, 2016.

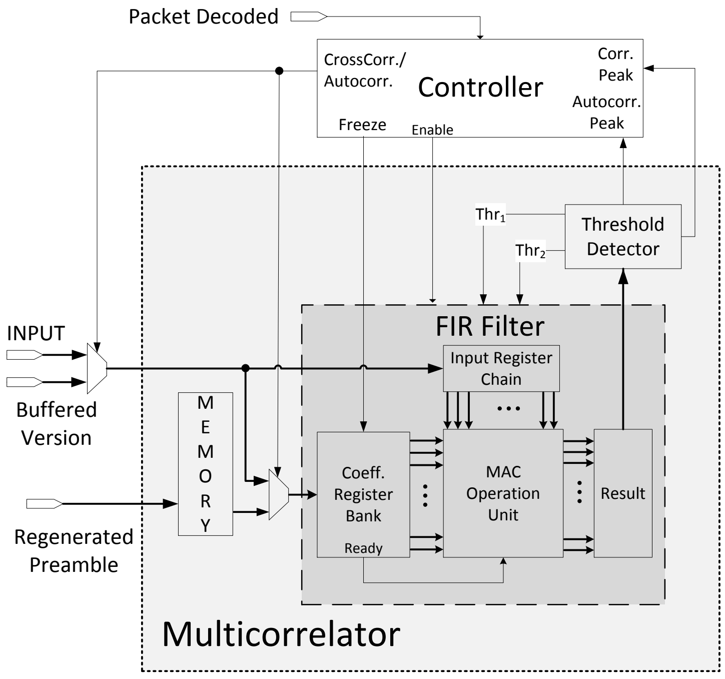


Figure 2.4: The infrastructure of the multicorrelator [P. II] © Elsevier, 2016.

2.5.2 Threshold Detector block

The main objective of this block is to compare the computed results of the FIR filter with a predefined threshold value. In the first phase (secondary transmitter detection) the filter computes the autocorrelation function. The autocorrelation measures when the similarity between the incoming signal and a delayed version of itself is in the maximum level. In that respect, the energy of the noise is calculated when the channel is idle. Once the channel becomes busy, the energy of the noise is subtracted from the total energy. As soon as the final result exceeds a threshold, the corresponding subcarrier is considered to be active. In contrast to autocorrelation, the threshold detector employs two threshold points when the filter is computing crosscorrelation function. It is mainly due to the fact that calculating crosscorrelation function is a very energy-hungry task. Therefore, we apply a 2-step verification algorithm proposed in [32]. As Figure 2.5 illustrates, in the first step the crosscorrelation of the first half of the signal is computed. The obtained results are compared with a preliminary threshold Thr_1. If the magnitude of the Thr_1 for the first-half calculation is met, the multicorrelator computes the second half and, subsequently, computes the overall results with a second threshold Thr_2. At this stage, a peak shows that there was a good correlation between the buffered signal and the regenerated preambles. Otherwise, the buffered version of the signal is discarded, even though the first threshold was met. A proper value of the preliminary threshold (Thr_1), as well as the original threshold (Thr_2), has a massive impact on the signal detection. For example, setting a low value for the Thr_1 might lead to false detection of the signal in low SNR regions. On the other hand, although a high value will guarantee the robustness of the calculation, it might result in packet loss since a low-power signal might be considered as noise. Therefore, a good approximation for the Thr_1 is 45% of the Thr_2 [32].

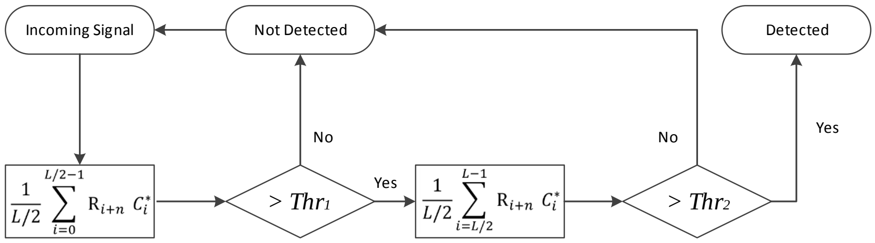

Incoming Signal
Not Detected
Detected
No
No
Yes
Yes
$\frac{1}{L/2}\sum_{i=0}^{L/2-1} R_{i+n} C_i^*$
$> Thr_1$
$\frac{1}{L/2}\sum_{i=L/2}^{L-1} R_{i+n} C_i^*$
$> Thr_2$

Figure 2.5: The 2-step verification algorithm [P. III] © Springer, 2017.

2.5.3 Controller Block

The controller block continuously monitors all the subblocks. Based on the information received by the threshold detector, the controller decides how to configure the multicorrelator. When the multicorrelator is detecting a secondary transmission, the controller configures the coefficients of the filter with the delayed version of the incoming signal. The length of the filter for calculating autocorrelation is 16-tap due to the structure of the preamble in IEEE 802.11a. In this standard, each packet starts with a predefined sequence of bits known as *preamble*. The preamble consists of two parts, a Short Training Symbol (STS) alongside a Long Training Symbol (LTS). The STS is composed of ten repetitive sequences of bits, each sequence is 16-bit long. Thus, the STS, which also known as Short Training Field (STF), provides a very good autocorrelation properties. The LTS is constructed from two identical 64-bit sequences. The LTS, or Long Training Field (LTF), provides more robust properties for exact timing acquisition [34]. Figure 2.6 illustrates the correlation characteristics of the preamble. The controller can simply configure the filter to compute the autocorrelation function by redirecting the incoming signal to the coefficients of the filter. In a similar way, the controller sets up the filter with regenerated preamble stored in the SRAM.

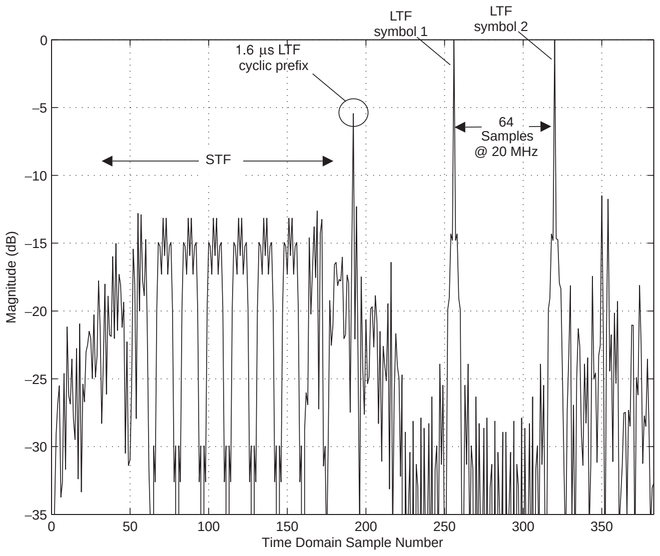


Figure 2.6: The correlation of the preamble with STS and LTS [35]

2.5.4 FIR Filter Block

As earlier mentioned, the FIR filter is the most resource-hungry and power-consuming block of the synchronizer. Therefore, we put our best effort to carefully design the filter to minimize the hardware costs. As previously studied, NC-OFDM-based CR should gather as much information about the spectrum as possible. Hence, the CR receiver has to employ a very large order of the FIR filter in the synchronizer (e.g. 4096-tap filter). Although having such a high-order FIR filter is feasible, it raises serious issues from the hardware limitations point of view. The first concern is the limited silicon area. Another problem is the massive energy consumption of the high-order filters due to performing a large number of complex Multiply-Accumulate (MAC) operations. In addition, a high-order filter has the potential to degrade the overall performance of the whole system when some crucial considerations are ignored during the design time. In the following, we will explain some techniques in order to overcome the above-mentioned issues to some extent.

An FIR filter periodically calculates the MAC operation as Equation 2.1

$$y(n) = c(n) * x(n) = \sum_{k=0}^{N-1} c(k)x(n-k) \tag{2.1}$$

where the $y(n)$ is the filter response, comprising the sum of products of a set of conjugated coefficients $c(k)$ with the filter input $x(n-k)$ delayed by k samples, on a window whose length is N. Therefore, we will have the complex operation as Equation 2.2

$$\begin{aligned}(Re_1 + Im_1 i) \times (Re_2 - Im_2 i) &= Re_1(Re_2 - Im_2 i) + Im_1 i(Re_2 - Im_2 i)\\ &= Re_1 Re_2 - (Re_1 Im_2)i + (Re_2 Im_1)i - (Im_1 Im_2)i^2\\ &= Re_1 Re_2 + Im_1 Im_2 + (Re_2 Im_1 - Re_1 Im_2)i\end{aligned} \tag{2.2}$$

where $Re_1Re_2 + Im_1Im_2$ and $Re_2Im_1 - Re_1Im_2$ are the Real and Imaginary calculations of the I/Q signals, respectively. By taking Equation 2.2 into consideration, we can figure out that an N-tap complex FIR filter requires $2 \times N$ multipliers, as well as $(2 \times N) - 1$ adders to compute the final results. On the other hand, implementing a multiplier requires more hardware resources than an adder. Additionally, the hardware implementation of the multiplier is very much slower than the adder, as well. Therefore, there have been some studies on the possibility to replace a multiplier with the cost of inserting additional adders. Karatsuba multiplication [36] as well as the Golub's method [37] were two famous multiplication methods in which a multiplier can be replaced by three adders. In Karatsuba multiplication, if we assume $P = (Re_1 + Im_1) \times (Re_2 - Im_2)$, $R = Re_1Re_2$ and $I = Im_1Im_2$, we will generate the Equation 2.3 as follows:

$$\begin{aligned}P &= (Re_1 + Im_1) \times (Re_2 - Im_2)\\ &= Re_1Re_2 + Re_2Im_1 - Re_1Im_2 - Im_1Im_2\\ &= R + Re_2Im_1 - Re_1Im_2 - I\\ Re_2Im_1 - Re_1Im_2 &= P - R + I\end{aligned} \tag{2.3}$$

Then, the Equation 2.4 is derived as:

$$\begin{aligned}(Re_1 + Im_1 i) \times (Re_2 - Im_2 i) &= (R - I) + (P - R + I)i \\ &= Re_1 Re_2 + Im_1 Im_2 \\ &\quad + [(Re_1 + Im_1)(Re_2 - Im_2) - Re_1 Re_2 + Im_1 Im_2]i \end{aligned} \tag{2.4}$$

Golub's method can be derived in a similar way to the Karatsuba's as Equation 2.5

$$\begin{aligned}(Re_1 + Im_1 i) \times (Re_2 - Im_2 i) &= [(Re_1 + Im_1)(Re_2 + Im_2) - Re_1 Im_2 - Re_2 Im_1] \\ &\quad + (Re_2 Im_1 - Re_1 Im_2)i \end{aligned} \tag{2.5}$$

At first glance, both methods require 5 multipliers, the multiplication results for both Re_1Re_2 and Im_1Im_2 are computed once, though. Hence, we can reuse those results in the other section of the equation.

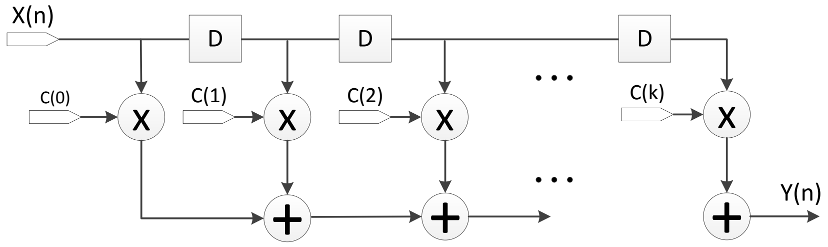
All these efforts try to replace only one multiplier with the cost of some adders. It is mainly because of that implementing a complex FIR filter is potentially 4 times more hardware resource expensive than a real-valued FIR filter. In the next section, we will investigate which one of these methods is the most suitable one for the target FPGA device.

Technically, the proposed multicorrelator operates in two different modes. First one is the high-precision mode in which the multicorrelator employs a MAC-based FIR filter and the second mode is the low-precision mode in which the multicorrelator degrades the precision of the filter using a MultiplierLess (ML) method. The multicorrelator reconfigures the FIR filter block on-the-fly using high-precision alongside the low-precision mode to compute crosscorrelation and autocorrelation functions, respectively. These two modes are further inspected in the following subsections.

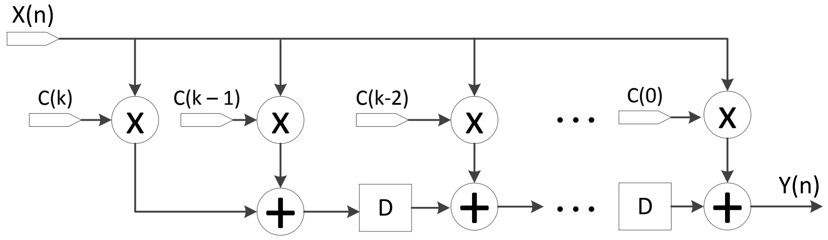
2.5.4.1 MAC-Based FIR Filter Architecture

In principle, FIR filters are constructed from three fundamental blocks known as multipliers, adders and delay elements. These three blocks can be integrated together in different shapes, each of them calculates the Equation 2.1. The most straightforward structure for the FIR filters is the Direct Form (DF) architecture. In DF, the input is propagated through the delay elements as Figure 2.7a depicts. The DF architecture heavily suffers from the long critical path in a design with a high-order FIR filter. For example, the critical path for a 256-tap FIR filter is equal to $T_{crit.} = T_m + (255 \times T_a)$, where the T_m and T_a are units of time required by the multiplier and adder to compute the corresponding results, respectively. Hence, we do not study this structure any further and just present some preliminary results after the synthesis in section 2.8.1.

Figure 2.7b is a well-known enhanced form of the DF architecture which is widely known as Transposed Direct Form (TDF). Technically, the main difference between TDF and DF is only the change in the location of the delay elements, as well as inverting the coefficient elements. Irrespective of the order of the FIR filter, the critical path of the TDF is always $T_{crit.} = T_m + T_a$ since the delay elements cut the long critical path introduced in high-order filters. However, the TDF architecture may suffer from the long interconnection problem



(a) The DF architecture suffers from the long critical path



(b) The TDF architecture mitigates the critical path, but potentially suffers from long input interconnection

Figure 2.7: The most common forms of the FIR filter [P. II] © Elsevier, 2016.

in the input path of a filter with a large number of taps. Moreover, TDF architecture is not capable of employing ternary adders due to the nature of its structure. We will explain when and where employing ternary adders are useful in section 2.7 in detail.

Since both DF and TDF architectures have some critical issues in high-order filter designs, we further investigate two other candidates to mitigate the long critical path, introduced in DF, along with the long interconnection problem, which is potentially introduced in TDF. These two methods are known as Parallel Direct Form (PDF) and Pipelined-Parallel Direct Form (PPDF). Figure 2.8 presents the PDF architecture which is another alteration to the DF format. Instead of employing a serial chain of adders, the PDF uses a specific structure similar to binary adders tree in which all the branches are processed in parallel. Hence, the critical path is limited to $T_{crit} = T_m + \left(\lceil \log_2^N \rceil \times T_a\right) = T_m + (8 \times T_a)$ for the example 256-tap FIR filter. This architecture does not introduce a long critical path, while it distributes the long interconnection through the binary tree. Similar to previous architectures, the PDF form has its own disadvantages of which hardware implementation complexity is one of the major ones.

The PPDF shown in Figure 2.9 is an enhanced form of the PDF structure in which all the branches are pipelined. Although both critical path and long interconnection problems are mitigated in this structure, the resource usage is heavily increased. The PPDF employs an additional delay element after each addition. Therefore, the critical path will remain at the minimum level of $T_{crit.} = T_m + T_a$. This architecture also has the same design complexity as of the PDF, alongside using a huge amount of hardware resources. Furthermore, this architecture cannot exploit ternary adders due to the nature of its structure. Further explanation about the ternary adders are given in section 2.8.2 in detail.

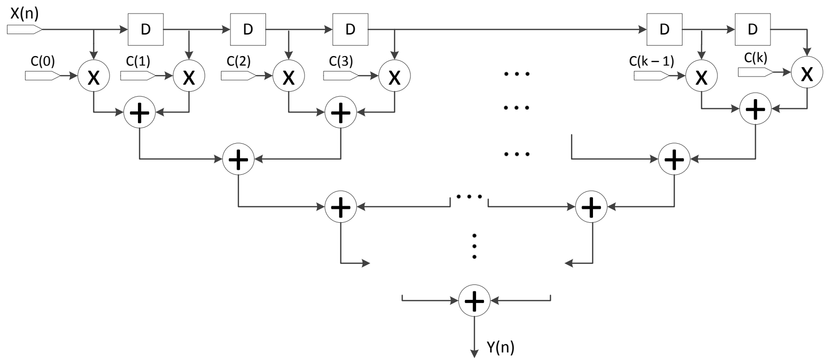


Figure 2.8: The PDF architecture uses a binary adder tree shape [P. II] © Elsevier, 2016.

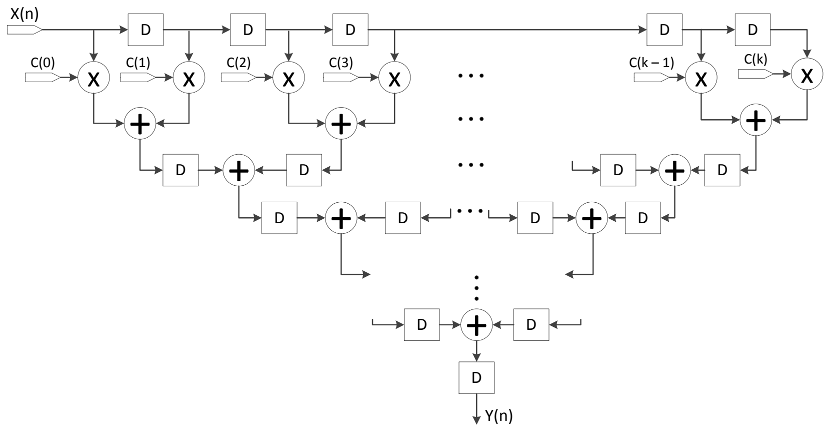


Figure 2.9: The PPDF architecture employs a bunch of delay elements to minimize both the critical path and the interconnections [P. II] © Elsevier, 2016.

2.5.4.2 ML-Based FIR Filter

The ML-based FIR filters have recently attracted more researchers since it requires a very low amount of hardware resources. The ML is an approximate computing technique where the most of the computations are mitigated [38]. In the context of the FIR filters, the coefficient multipliers are replaced by adders and shift elements, while the coefficients are represented in a form of summed powers of 2. According to Saha et al., experiences with MATLAB reveal that a sign bit correlation (instead of a full MAC operation) between the filter input and coefficients, provide sufficient accuracy to detect the secondary transmitter (autocorrelation) even in low SNR regions [29, 30]. In either case, the total number of registers, multiplier, and adders are dramatically decreased. Furthermore, the energy consumption of an ML-based correlator is massively lower than the MAC-based ones since a significant number of intensive MAC operations are mitigated. We will further investigate the overall results later on in section 2.8. We also observed that the hardware implementation of the sign bit correlation was successful to detect the secondary transmitter. Implementing an ML-based FIR filter is very straightforward without introducing additional complexity to the design. Hardware implementation of the sign-bit correlator is a simple XOR or XNOR gate depending on the designer's choice. Although the ML-based FIR filter presents satisfactory results to perform autocorrelation function, it still requires more practical studies in the field of communications since it is an approximate computing method.

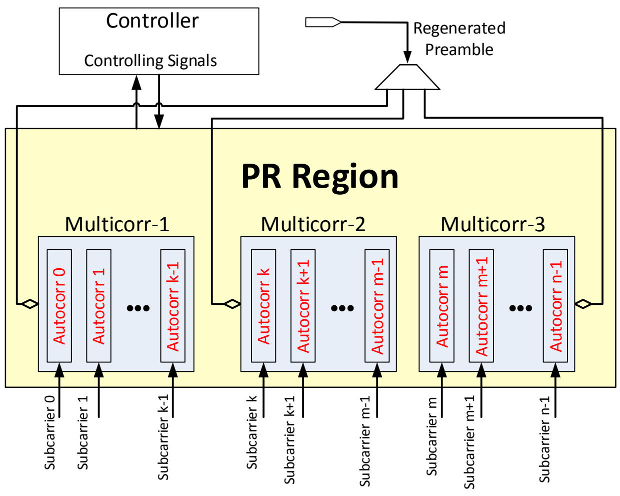
2.6 Partial Reconfiguration

Partial Reconfiguration is a powerful ability available on some of the FPGA devices in which a particular part of the FPGA can be reconfigured, while the remaining parts are operating normally. Partial Reconfiguration was first available on Xilinx XC6200 devices in 1995 [39]. Later on in mid-2010, Altera Corporation announced that their newly released 28nm Stratix V FPGA was equipped with a friendly method to do PR [40]. The PR is a specific feature which enables particular portions of the FPGA to be reconfigured on-the-fly without disrupting rest of the circuit during the run-time. This feature reserves a portion of the FPGA to perform PR on that region. The PR region may have different configurations, as well as the functionality, while everything else outside of this region has a normal operation. It implies that the PR region is dynamically changed, while the remaining portions of the FPGA are static.

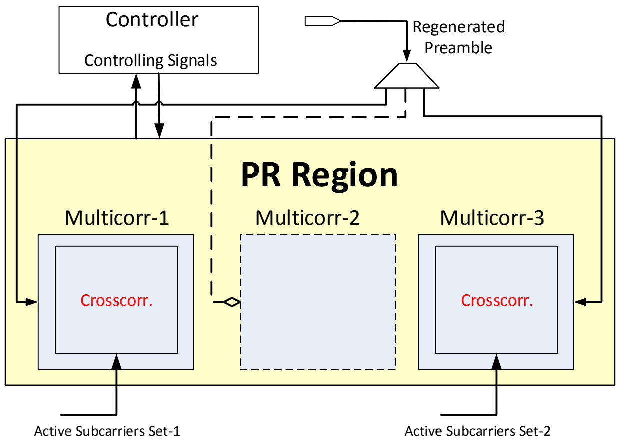
In this work, the synchronizer employs the PR feature to reconfigure itself to either perform autocorrelation or crosscorrelation functions on demand. Figure 2.10a illustrates an example of the multicorrelator infrastructure when it is (re)configured to perform autocorrelation function. In the same figure, the PR region consists of three identical multicorrelators. Three is the maximum number of multicorrelators that our target FPGA device allows implementing on hardware due to the resource constraints. Nevertheless, fewer multicorrelators can be assigned in PR region depending on the application, as well as the FFT window. Furthermore, the multicorrelator is (re)configured to detect any secondary transmission. Therefore, each multicorrelator is (re)configured with ML-based FIR filters. On the other hand, there are plenty of resources dedicated to each multicorrelator in the PR region, because each multicorrelator should also be able to perform crosscorrelation function in the following steps. In addition, since the ML-based technique is a very low-cost approach, each multicorrelator has sufficient resources available to perform autocorrelation on a wide-band spectrum. How wide a multicorrelator can be is totally dependent to other parameters such as the spectrum, FFT window, etc.

Figure 2.10b illustrates a scenario in which two multicorrelators have detected a secondary transmission, while the remaining one has failed to detect. In this example, the first multicorrelator alongside the third one is configured to perform crosscorrelation between the buffered version of the incoming signal and the regenerated time-domain preambles. Meanwhile, the CR can decide what the second idle multicorrelator should do. For instance, the CR might allow the idle multicorrelator to keep sensing the spectrum or release the hardware resources dedicated to the second multicorrelator for other purposes, e.g. hardware accelerators. Once the connection is disrupted, the PR region is reconfigured for all the multicorrelators to perform autocorrelation function and this circulation will be continued.

Another advantage of using PR feature is to set up a multi-standard receiver. Reconfiguration on-the-fly provides the ability to load various IEEE standards on the system, at will. For example, a single receiver can set up its parameters to operate in a variety of IEEE standards (such as 802.11, 802.15, 802.22, etc) without a single change in the hardware. Using PR feature enables such a CR to be more versatile and powerful than ever.



(a) The autocorrelation function



(b) The crosscorrelation function

Figure 2.10: The infrastructure of the multicorrelator when employs Partial Reconfiguration feature to perform either autocorrelation or crosscorrelation functions on demand [P. II] © Elsevier, 2016.

2.7 FPGA Constraints

In this section, we further inspect the hardware implementation, as well as the obstacles one might encounter during the implementation. Although NC-OFDM-based CR brings a variety of advantages, it has its own challenges and constraints. Some of the most important hardware constraints are inspected as follows.

2.7.1 Insufficient DSP Blocks

Technically, today's FPGAs are equipped with several DSP blocks depending on the family board. DSP blocks can implement chain adders, multipliers and a combination of both (MAC operations) on the hardware. Practically, DSPs are used to calculate

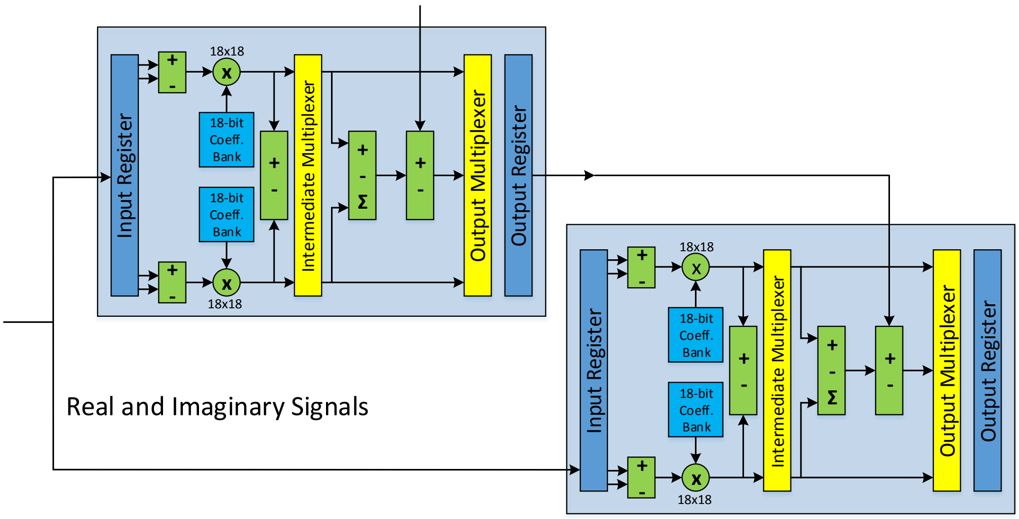


Figure 2.11: How the Stratix V cascades two DSP blocks to compute a complex multiplication [P. II] © Elsevier, 2016.

computationally intensive tasks such as digital filtering algorithms. However, such as other hardware elements, DSP blocks are finite hardware resources. Hence, utilizing a DSP block as much as possible is a very important issue from the hardware point of view. Depending on the FPGA technology, each DSP block tries to perform the MAC operation in the best efficient way. For example, DSP blocks on the Stratix II and Stratix III FPGA devices of the Altera Corporation are designed to compute Karatsuba algorithm more efficient than the conventional method [41]. In contrast to the above-mentioned FPGA families, Stratix V provides a different mode to calculate MAC operations. Additionally, Stratix V have a specific mode in the DSP structure to compute complex multiplication as the traditional method [42]. As we practically observed, DSP blocks of a Stratix V board are utilized more efficiently in the traditional method than neither Gulob's method nor Karatsuba's algorithm. Numerically, each complex multiplication requires two cascaded DSP blocks whereas three DSP blocks are required in both Gulob's and Karatsuba's methods. Figure 2.11 presents the infrastructure of a DSP block available on Stratix V series, as well as how the board cascades two DSPs to compute a complex multiplication.

The FPGA board we employed for prototyping is the Altera FPGA Stratix V family, series "5SGSMD5K2F40C2N". The maximum number of available DSP blocks for this series is 1590 units. Moreover, each multicorrelator is able to perform crosscorrelation on an FFT window size of 256. As mentioned above, each complex multiplication requires 2 DSP blocks to compute the result. Hence, each multicorrelator demands $256 \times 2 = 512$ DSP blocks to operate normally. Therefore, the target FPGA device does not allow the designer to implement more than three multicorrelators. This is the first limitation for such a platform from the hardware point of view. However, there are two approaches to tackle the insufficient DSP problem. A very straightforward solution is to enforce the synthesis tool to implement the MAC operation on look-up tables instead of implementing in the DSP. This method has the potential to increase hardware resources up to 10 times more, while the MAC operation is executed much slower than when the DSP block computes the results. Another alternative is to degrade the filter precision. For example, one can employ several 64-tap FIR filters instead of employing a 256-tap one. This method is also heavily dependent on the application, as well as the environment.

2.7.1.1 Silicon Area

The silicon area is another concern in such large designs. On FPGA, silicon area is translated to advanced look-up table based units that form the combinatorial logic of the design. As previously mentioned, the FIR filter is the most resource-hungry block in the synchronizer. In the previous subsection, we studied that the DSP blocks are very limited. In addition, a high-order FIR filter will also exhaust other fundamental components, such as registers. As the order of the filter becomes larger, the more hardware resources are required. Potentially, an FIR filter with a large number of taps can occupy most of the available hardware resources on the target FPGA. It does not allow to implement other fundamental components of a CR receiver, e.g. FFT, on the same FPGA die.

2.7.1.2 Power Consumption

The total power consumption is mainly depending on three major parameters known as static power (P_s), dynamic power (P_d) and Input/Output (I/O) power (P_{io}). Static power is the power required to operate the FPGA when there is no activity on the circuit. Therefore, the P_s is almost constant in each design. P_d is the power consumed when there is an activity on the circuit and transistors are toggling and parasitic capacitors are charged and discharged. P_{io} is a magnitude of the power required for a signal to start from the input, traverses the design and finally reaches to the output. Design issues such as long critical path, long interconnections, and bad hardware description have a proportional effect on P_d along with the P_{io}. As previously explained, a high-order FIR filter has the potential to carry all the mentioned design issues which proportionally increase the overall power consumption of the design.

2.8 Experimental Results with Further Discussion

The explained synchronizer block was described using VHDL description language; it was simulated using ModelSim software and synthesized using Quartus II 12.1 and 15.1 environments. Eventually, the synthesized design was implemented on Stratix V FPGA device speed grade 2 series. We achieved a preliminary result for each configuration and, then, we tried to improve the candidates by imposing some optimization techniques explained in next subsection. The preliminary results are reported by Quartus II 12.1, whereas the final results are achieved by Quartus 15.1.

2.8.1 Preliminary Results

Table 2.1 shows the preliminary results after synthesizing the multicorrelator configured with specifically mentioned FIR filter structures in terms of employing Adaptive Logic Modules (ALMs), DSP blocks, and registers, as well as the maximum achieved operating frequencies. The TDF configuration shows satisfactory results for the multicorrelator configured with a 256-tap FIR filter. The DF architecture requires almost the same amount of hardware resources as TDF, but the maximum operating frequency is not acceptable. Although the PDF architecture requires only about half of the resource elements and registers compared to the mentioned architectures, it seems that the long interconnection in binary adder tree does not allow this architecture to achieve better results than DF.

Table 2.1: Preliminary results for the multicorrelator with different configurations [P. II] © Elsevier, 2016.

	TDF	DF	PDF	PPDF	ML
Logic Utilization (ALMs)	12,483	12,490	7504	14,611	1876
Total DSP Blocks	512	512	512	512	0
Total Registers	16,378	16,883	8873	28,037	2886
Max. Freq. [MHz]	237	68	88	240	257

Table 2.2: Power Consumption Analysis (mW) [P. II] © Elsevier, 2016.

	TDF	DF	PDF	PPDF	ML
I/O Power	22.58	2108.14	33.92	22.86	26.20
Dynamic Power	344.17	200.59	265.89	160.10	9.42
Static Power	1070.06	1062.71	987.12	955.80	951.50
Total Power	1436.82	3371.43	1286.92	1138.75	937.13

The PPDF configuration requires slightly more logic than the TDF and DF, while the total number of registers is massively increased. Eventually, a multicorrelator with the ML-based FIR filters shows that it can be the best candidate to perform autocorrelation function.

Power dissipation analysis is reported based on the results generated by "PowerPlay Power Analyzer" tool in Quartus II environment. The analyzer reads the Value Change Dump (VCD) file, which contains signal transaction activities, generated by the ModelSim software.

Table 2.2 reports the total power consumption for each configuration. The TDF architecture has a higher dynamic power consumption than the other configurations. The DF has a massive power consumption on its I/O path due to the long critical path. The PDF architecture was successful to resolve the massive I/O power consumption observed in the DF structure. The PPDF is the greenest architecture among the MAC-based designs from the power consumptions point of view. Eventually, the ML architecture once again shows that it is also a good candidate to perform autocorrelation function in terms of overall power consumption.

2.8.2 Optimization Techniques and Final Results

In this subsection, we will not study the DF architecture due to showing unacceptable preliminary results, as well as the ML approach since it is not a robust algorithm for computing crosscorrelation function. It is worth to recall that since computing crosscorrelation is computationally much more intensive than the autocorrelation, we try to minimize the logic utilization, register usage, and power consumption while increasing the maximum operating frequency in other MAC-based candidates. We employed some optimization techniques to improve the final results as follows. Henceforth, we will consider Optimized Transposed Direct Form (OTDF), Optimized Parallel Direct Form (OPDF) and Optimized Pipelined-Parallel Direct Form (OPPDF) as the optimized version of their previously studied architectures, respectively.

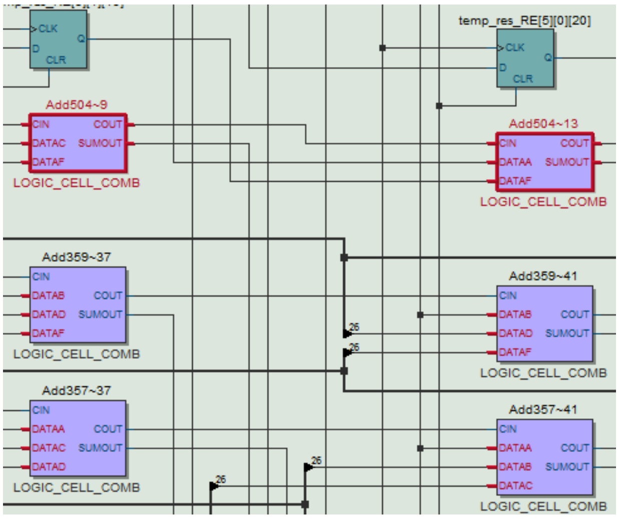
HDL Modification: The HDL modification is the foremost step of optimizing a design. We start modifying the codes to mitigate unnecessary registers, shortening the critical path, employing parallelism where applicable, etc.

Using Synthesis Tool Advisors: The Quartus II offers a set of advisors (such as timing optimization advisor, and resource optimization advisor) to the designer to synthesize the design in the most optimized way. For example, if the designer needs to optimize the design from power consumption point of view, the Power Optimization Advisor recommends several ways that may lead the same design to consume less power. We put our best effort towards the time optimization while keeping the resource optimization alongside the power optimization at a fair level. Table 2.3 summarizes the maximum achieved optimizations offered by the previously mentioned advisors.

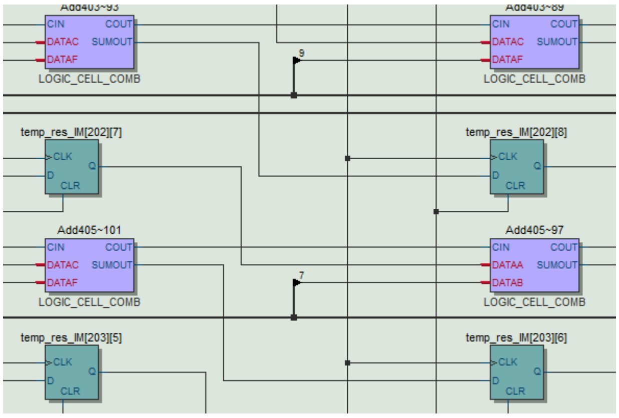
Using Ternary Adders: In 2005, Altera corporation encouraged digital designers to employ a ternary adder (3-input adder) structures instead of using binary ones (2-input adder) in their hardware implementations. According to a whitepaper published by Altera, exploiting ternary adders not only utilize fewer resource elements but also increase the maximum operating frequency [43]. Perhaps, in 2005 the synthesis tools were not mature enough to automatically replace and reroute binary adders with ternary adders. We also investigate this issue with different configurations of the MAC-based FIR filters. Although the hardware description of all configurations had been described to employ binary adders, the Quartus II 15.1 was automatically exploiting ternary adders whenever it was feasible. In order to discover this issue, we considered the Technology Map Viewer tools of the Quartus II. Figure 2.12a shows a part of the OPDF architecture where the ternary adders are employed whereas the Figure 2.12b shows a portion of the OTDF where the synthesis tool could not replace ternary adders with the binary one. The main reason that Quartus II failed to use ternary adders is the architectural formation in which each adder is surrounded by a delay element. The same scenario is valid for OPPDF architecture. The synthesis tool has become smart enough to replace ternary adders with binary adders. Describing ternary adders in a binary tree structure (as it is used in PDF and PPDF architecture) is a very complex and time-consuming task from the hardware designer's point of view.

Table 2.3: Summary of the maximum optimization gained by the synthesis tool [P. II] © Elsevier, 2016.

	TDF	**OTDF**	**PDF**	**OPDF**	**PPDF**	**OPPDF**
ALMs	12,483	8417	7504	7336	14611	9856
Saving	+32.57%		+2.24%		+32.54%	
Total Registers	16,378	16,407	8873	8925	28,037	21,197
Saving	−0.18%		−0.59%		+24.4%	
Max. Freq. [MHz]	237	258	88	94	240	268
Speed-up	+8.86% 1.09×		+6.62% 1.06×		+11.66% 1.11×	



(a) The OPDF architecture mainly exploits ternary adder structure. However, there are still a few binary adders which could not be implemented as ternary ones



(b) The OTDF architecture can only employ binary adders

Figure 2.12: A snapshot from Technology Map Viewer [P. II] © Elsevier, 2016.

DSP Packing and Intermediate Registers: When we were running timing analysis on each design, we discovered that not fully utilizing the DSPs is one of the main factors which jeopardizes the timing optimization. Meanwhile, we found out that each DSP block employs an embedded output register on the Stratix V board. These registers exist whether the designer uses them or not. Hence, we modified the HDL code in a way to exhaust the DSPs as much as possible, including using the free embedded register. Having utilized the DSPs, which is also known as DSP packing, not only magnificently improved the timing performance, also significantly affected the logic utilization, along with the register usages.

Furthermore, we took intermediate register insertion method into account in OPDF architecture. The OTDF, as well as the OPPTF, could not benefit from this method since the register elements were distributed between the adders. Hence, some intermediate

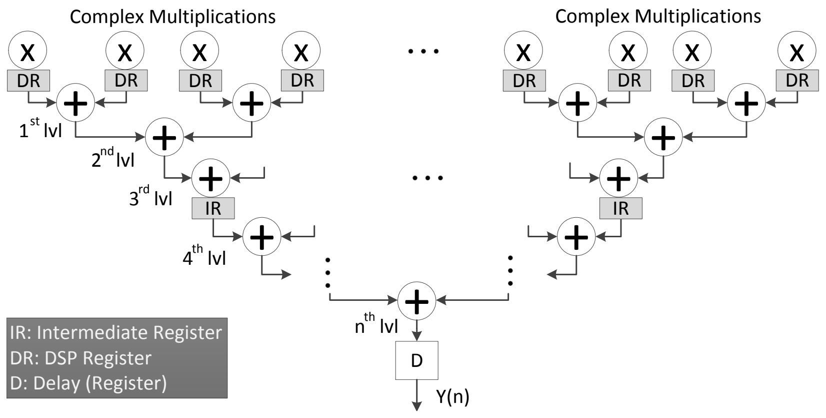


Figure 2.13: How the optimized PDF architecture benefits from *DSP packing* along with the *intermediate register insertion* method [P. II] © Elsevier, 2016.

registers were introduced to the OPDF architecture at certain levels. We discovered that the best locations for register insertion are at each 3^{rd} level, e.g. 1^{st} level, 4^{th} level, 7^{th} level, etc. Figure 2.13 shows how the OPDF architecture could benefit from both register insertion and DSP packing to achieve the maximum possible operating frequency. However, the OTDF and OPPDF architectures are the only ones that could benefit from the register packing method.

Table 2.4 presents each architecture after considering all the mentioned optimization techniques to achieve the highest operating frequencies at different temperatures. As the table shows, the final results are available for all architectures, while the preliminary results are only considered for the slow model at 85 ℃. Unfortunately, the other modes were not available for presentation. However, designs in the slow model at 85 ℃ can be considered as the most practical situation. Furthermore, it can be observed from the same table that the OPDF architecture obtained the most benefit from the optimization techniques by achieving 2.83× speed-up compared to the non-optimized architecture. Following by, the OPPDF and OTDF could obtain 1.22× and 1.18× speed-ups, respectively, by considering all optimization techniques.

Table 2.5 reports the power consumption in each cell alongside the total power consumption figures for each optimized MAC-based architecture in detail. The table depicts that

Table 2.4: The Maximum Operating Frequency Gained for Optimized MAC-Based Architectures

		Maximum Frequency [MHz]					
		OTDF		**OPDF**		**OPPDF**	
		Preliminary	Final	Preliminary	Final	Preliminary	Final
Slow Model (SM)	85 ℃	237	280	88	249	240	293
	0 ℃	–	289	–	258	–	315
Fast Model (FM)	85 ℃	–	378	–	347	–	387
	0 ℃	–	404	–	373	–	419
Speed-up at SM 85 ℃		1.18×		2.83×		1.22×	

Table 2.5: The power consumption of each cell after the optimization (mW)

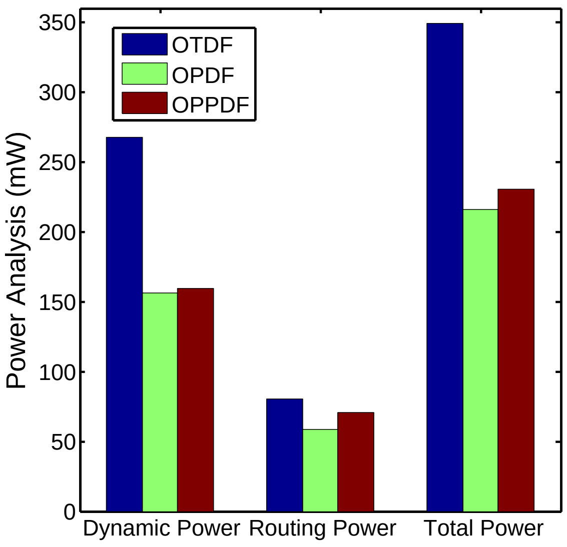
Power	**Arch.**	**Mem. Block**	**DSP Block**	**Comb. Cell**	**Clock Enable**	**Reg. Cell**	**I/O**	**Total Power**
Dynamic	*OTDF*	0.61	257.81	2.75	0	5.17	1.43	267.77
	OPDF	0.62	143.99	3.22	0	5.87	2.15	155.85
	OPPDF	0.62	143.99	3.34	0	7.57	3.91	159.43
Static	*OTDF*	–	–	–	–	–	1.11	1.11
	OPDF	–	–	–	–	–	1.11	1.11
	OPPDF	–	–	–	–	–	1.11	1.11
Routing	*OTDF*	0.01	0.46	23.46	24.84	11.54	19.95	80.26
	OPDF	0.02	1.71	2.19	23.83	30.47	0.54	58.76
	OPPDF	0.02	4.24	1.60	29.47	34.44	0.50	70.27
Total	*OTDF*	0.62	258.27	26.21	24.84	16.72	22.49	349.15
	OPDF	0.64	145.70	5.41	23.83	36.35	3.80	215.73
	OPPDF	0.64	148.23	4.94	29.47	42.01	5.52	230.81

the OPDF architecture achieved the lowest power consumption design with distinction compared to its counterpart configuration, OTDF, which is the most power-hungry architecture. Indeed, the power consumption of the OPDF form is drastically increased in DSP blocks, combinational cells, along with the I/O. Perhaps, the long interconnection, as well as the lack of exploiting ternary adders are two potential candidates to increase the overall power dissipation of the OTDF architecture. The reported power dissipation for the OPPDF architecture is slightly larger than the OPDF one in most of the cases. Nevertheless, we can categorize OPDF, OPPDF and OTDF architectures as the greenest, moderate and highest power consumers, respectively.

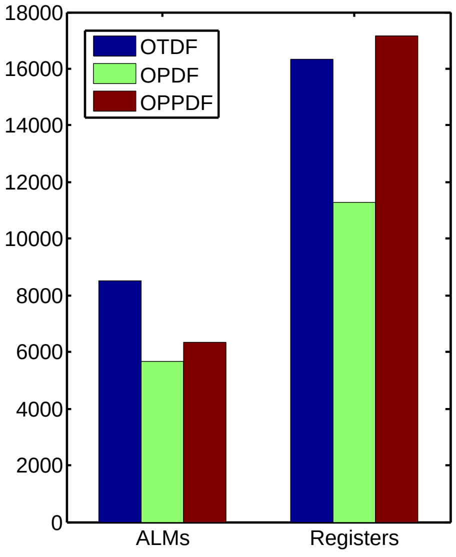
Figure 2.14 visualized all the achieved results in some bar charts. Figure 2.14a compares all the MAC-based architectures together in terms of power consumption. As previously discussed, the OPDF achieved the minimum power consumption within the MAC-based architectures from dynamic, routing and total powers point of view. Hence, the OPDF is the best candidate in systems where the *power consumption* is the limiting factor. In contrast to OPDF, the OTDF architecture dissipates around 350mW (about 61.85%) more than the lowest power consumer (OPDF) configuration. Additionally, the OPPDF-based filter dissipates slightly more power than the OPDF-based one.

Figure 2.14b compares the mentioned architectures together from the resource usage point of view. Based on the figures, the OPDF architecture requires fewer ALMs, as well as registers, than the other configurations. Nevertheless, again, the OPDF architecture is the best choice when the limiting factor is the *hardware costs* and *silicon area*. The OTDF along with the OPPDF architectures introduce a moderate hardware cost to the system. Hence, there is a trade-off between using either OPPDF or OTDF since OPPDF is a better choice in terms of ALMs, while the OTDF requires fewer registers.

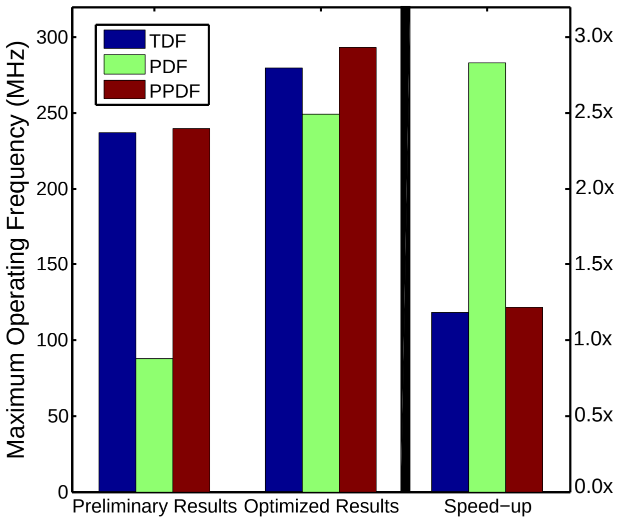
Figure 2.14c assesses the mentioned configurations from operating frequency point of view before and after applying optimization techniques. Although the OPDF configuration was improved the most after applying optimization techniques by reaching to a 2.83×



(a) Power dissipations



(b) Resource usages



(c) Maximum frequencies

Figure 2.14: The hardware costs for each architecture [P. II] © Elsevier, 2016.

speed-up, it could not obtain the maximum operating frequencies achieved by the two other candidates. Therefore, the OPPDF is the best choice in systems where reaching to the maximum available *operating frequency* is a very critical issue.

Table 2.6 presents a fair comparison between the candidates by ranking from 1^{st} one to 3^{rd} place. The comparison is based on the results which have been discussed so far. Based on the table, the best candidate for limited silicon area, along with the lowest possible power consumption is the OPDF form, while the best candidate in a speed-limited environment is the OPPDF. Eventually, the candidates are prioritized after an overall evaluation.

Table 2.6: MAC-based FIR filter from the designer's point of view

		OTDF		**OPDF**		**OPPDF**	
		Rank	Diff.	Rank	Diff.	Rank	Diff.
Silicon Area	*ALMs*	3^{nd}	+50.22%	1^{st}		2^{nd}	+11.88%
	Registers	2^{nd}	+44.88%	1^{st}		3^{nd}	+51.93%
Total Power		3^{rd}	+61.85%	1^{st}		2^{nd}	+7.00%
Speed		2^{nd}	0.96×	3^{rd}	0.85×	1^{st}	
Overall Evaluation		1^{st} OPDF, 2^{nd} OPPDF, 3^{rd} OTDF					

2.9 Concluding Remarks

In this chapter, we mainly described a new architecture for the synchronizer suitable for Non-Contiguous Orthogonal Frequency Division Multiplexer (NC-OFDM)-based Cognitive Radio (CR) receiver. In addition, we discussed some fundamental requirements in NC-OFDM system, as well as the current obstacles and challenges. The contribution to the state-of-the-art in synchronizer was to employ a multicorrelator to perform both autocorrelation (for secondary user detection) and crosscorrelation (for packet detection) functions on demand. Therefore, the multicorrelator had to reconfigure its parameters during the run-time. In order to do so, we employed Partial Reconfiguration (PR) feature to reconfigure a particular portion of the hardware on-the-fly, while the rest of the circuit is operating. Furthermore, the infrastructure of the synchronizer was studied. Moreover, a wide study on the architecture of the multicorrelator was provided. The major component of the multicorrelator was constructed based on the Finite Impulse Response (FIR) filters. Next, the idea behind using the MultiplierLess (ML) approach (to perform autocorrelation function) was explained. We further presented other Multiply-Accumulate (MAC)-based configurations to perform crosscorrelation function including Direct Form (DF), Transposed Direct Form (TDF), Pipeline Direct Form (PDF), and Parallel-Pipeline Direct Form (PPDF). Then a preliminary synthesis on each configuration was performed. Subsequently, excluding the DF due to presenting unacceptable results, we tried to optimize other MAC-based architectures to reach to the maximum possible operating frequencies, while keeping the overall hardware costs at the minimum level. Next, a fair comparison before and after applying the optimization techniques, alongside a wide study on all optimized architecture was presented. Eventually, we discovered that the Optimized PDF (OPDF) architecture is potentially the most suitable candidate in various environments, e.g. resource-limited systems. However, based on the designer's choice, the environments, as well as the hardware consideration, other candidates had the potential to be used in hardware description of the multicorrelator.

3 Reconfigurable IP-Based Memory Management Unit

The content of this chapter is mainly retrieved from publications [P. IV] and [P. V]. In this chapter, we discuss about design and hardware implementation of a reconfigurable IP-based MMU. The proposed MMU employs three Translation-Lookaside Buffers (TLBs) in two levels of hierarchy. The first level is composed of two micro-TLBs, an 8-entry Data Translation Look-aside Buffer (DTLB) for Data address translations operating in parallel with a 4-entry Instruction Translation Look-aside Buffer (ITLB) for instruction address translations. In the second level, an Unified Translation Look-aside Buffer (UTLB) exists to store the Least Recently Used (LRU) instruction page translations, as well as the data ones. All the TLBs are scalable during the synthesis time. Moreover, the MMU can reconfigure itself to operate on different page sizes during the run-time. Similar to Chapter 2, the hardware implementation issues, as well as some optimization techniques are widely discussed in this chapter, as well. Furthermore, the overall effect of scaling the UTLB with different configurations, along with the critical path analysis is inspected. Next, the integration issues of the developed MMU with a standard RISC processor is investigated. The case study which the MMU is tightly-coupled with the COFFEE RISC processor developed by our group.

3.1 The Virtual Memory

Decades ago, one of the major problems of the programmers was to properly fit each program to the main memory size. Hence, programs larger than the main memory should be manually fitted to the main memory by the programmer which required extra efforts. Conceptually, with the introduction of the Virtual Memory (VM), the mentioned problem was solved to some extent. Having exploited the VM mechanism, the Operating System (OS) takes responsibility to fit all the programs in the main memory. The VM is an unreal extension to the main memory in which various applications share the main memory, even though they are larger than the physical size of the main memory. In principle, the VM keeps the active portions of a program in the main memory, while transferring the remaining inactive parts into a secondary storage (such as hard disk) in the form of several *page files*. When the program requires access to an inactive portion stored on the hard disk, the VM provides some hardware-software approaches to relocate the required page files into the main memory. Indeed, the main memory looks like a cache for the secondary storage in the concept of VM. [44]

3.2 The Virtual Address

Today's processors operate in two different modes known as *Real/Physical mode* or *Virtual mode*. In real mode, the address produced by the processor refers to an exact location in the physical memory. Hence, when the processor requires referring to the main memory, the real/physical address is used. In the virtual mode, the processor produces a virtual address generated by the OS. The virtual address is intuitively larger than the physical address. Although the virtual address does not refer to any specific location on the main memory, the OS provides a hardware-software approach for translating the virtual address into the physical one.

3.3 How the OS Manages Virtual Addresses

The operating system maintains the page files of a program in a form of a *page table* residing in the main memory. Each program has its own page table active on the main memory. The page table maintains the corresponding physical addresses of the given virtual ones. When a program requires to access to an inactive portion stored on the hard disk, the OS searches through the page table to relocate required page files into the main memory. A *page fault* occurs when a referenced virtual page is not available in the page table, meaning that the indexed virtual address does not exist in the main memory. Consequently, the OS takes control and find the required page in the secondary storage. Next, the operating system decides where to place the requested page in the main memory. In general, the OS employs the LRU replacement policy, assuming that a page which has not been referenced for a while will less likely to be referenced in near future than a page which has been recently used. A page fault introduces an expensive penalty for the system since it will take millions of clock cycles to be addressed. [44]

3.4 Memory Management Unit

Technically, the MMU is a piece of hardware which can be implemented either as a part of the processor or a separate Integrated Circuit (IC). All the memory accesses are passed through the MMU. The primary feature of the MMU is to provide the procedure of virtual-to-physical address translation [45]. However, address translation is not the only application of the MMU. Memory protection, cache control, and fast address translation are just a few more advantages of a system which employs an MMU. Each MMU contains a special purpose cache to keep track of the most frequent page translations, known as Translation-Lookaside Buffer (TLB). It is recommended to explain what the TLB is and why it is important to employ the TLB, before going further into detail regarding the infrastructure of the MMU.

In the virtual mode, a program requires to index to the memory twice since the page tables are maintained in main memory. The first access occurs to obtain the physical address and the second attempt is to grab the required data. On the other hand, accessing the main memory, such as load and store, is one of the slowest instructions from the processor point of view. Searching through the memory blocks, as well as the characteristic of the main memory are two prominent reasons explaining that. In this situation, a key to improve the overall performance is to minimize accesses to the main memory. The first time access to the page table in the main memory is inevitable in order to get the physical address. But if we store the retrieved physical address in a cache, in the case of second attempt to the same physical address which is also known as the principle of locality, the second reference will be addressed by the cache (instead of the memory). Indeed, the processor does not necessarily require to make the second reference to the main memory, since the physical address is available in the cache. This special address translation cache is traditionally called as the TLB.

3.4.1 Related Works

There have been wide studies on the memory management unit due to its versatility. There are similar works regarding the implementation of an MMU on different FPGA platforms. In [46], the authors replace the TBL with a 16 million Address Translation Table (ATT) in a 128MB DRAM memory. The DRAM is resided on a Network Interface Controller (NIC). Similar to the TLB, the ATT stores a large number of virtual-to-physical page address translations. Each word of the ATT is 8 byte (64 bits) long. The NIC is responsible for mapping all the virtual addresses to the physical memory through a driver. Virtual-to-physical address translation is these systems are potentially energy expensive since DRAMs are the main candidates in total power consumption [47].

The implementation of an MMU on a Xilinx Vertex 5 FPGA board is studied in [48]. The mentioned MMU employs a 16-entry TLB, which uses the LRU replacement policy, to store virtual-to-physical page address translations. Furthermore, the authors observe that employing a small size TLB with only 4-entry would improve the overall performance of the system. The TLB returns the physical address in two clock cycles in the case of a successive translation. Moreover, updating an entry of the TLB will take 16 clock cycles. The miss penalty in the same design is heavily depended on the status of the OS. The best case, as well as the worst case miss penalty is reported as 600 and 227,000 clock cycles, respectively.

In [49], a reconfigurable architecture is proposed which takes into account a TLB with 512 entries, along with a Direct Memory Access (DMA) to perform virtual-to-physical address translation mechanism. A TLB hit would take place in 4 clock cycles whereas a miss penalty would take thousands clock cycles. In the case of a miss, the TLB is deactivated, while an interrupt is immediately issued to the processor. Once the operating system resolves the miss, the TLB will return to normal mode to address page translations mechanism. The authors claim that the hardware implementation of the mentioned TLB improves the overall performance up to 5 times faster than the pure software implementation of the same design.

3.4.2 The Proposed MMU

In this work, we concentrate on developing a reconfigurable IP-based MMU which can be integrated to any standard RISC processor. The term "reconfigurability" is referring to a portion of the MMU which can reconfigure itself to operate on different page sizes. In addition, the developed MMU employs three TLBs in two levels of hierarchy similar to the theory of caches. Scalability is another feature offered by the MMU to the system designer. All the TLBs are simply scalable during the design-time which introduces the maximum flexibility to the system. Another motivation behind this work is to integrate the developed MMU with the COFFEE RISC processor. One of the primary disadvantages of the COFFEE core was the lack of a memory management unit for the OS [50]. The integration issues are explained step by step in the following sections.

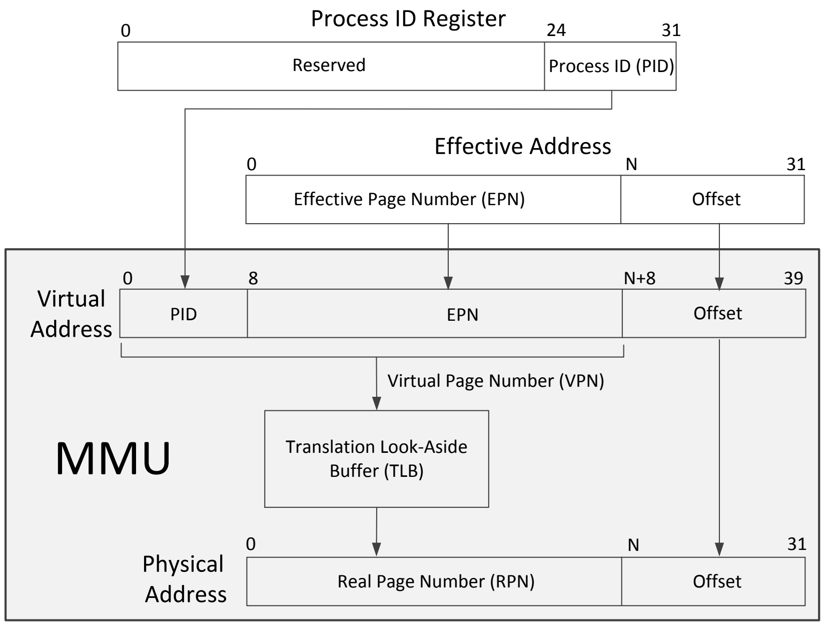


Figure 3.1: The address translation mechanism in the virtual mode [P. IV] © IEEE, 2015.

3.4.3 How the MMU Operates

Depending on the processor mode, the MMU operates either in real or virtual mode. In the real mode, as previously discussed, the 32-bit Effective Address (EA) is considered as a physical address which requires no translation. Nevertheless, the MMU does not process anything and bypasses the EA to the main memory. In contrast to the real mode, in the virtual mode, the MMU needs to translate the EA generated by the processor.

Figure 3.1 illustrates the virtual address translation mechanism step by step. The EA consists of two segments known as Effective Page Number (EPN) followed by an offset. During the translation, the MMU employs a combination of the EPN with an 8-bit Process ID (PID) field to form the Virtual Page Number (VPN). The PID is a non-negative identification number, uniquely assigned to each active process. The PID is driven from the PID register to resolve the overlapped area between several processes in virtual space. The MMU indexes to the TLBs with recently generated VPN. In the case of a successive hit in any TLBs, the Real Page Number (RPN) number is extracted from the corresponding TLB. Then, it is concatenated with the offset to form the physical address. The processor can now refer to the main memory using the physical address generated by the MMU.

In the EA, the boundary of the EPN and the offset fields are distinguished by the value N (see Figure 3.1). The magnitude of N is embedded in 3 specific bits within the EPN field known as the *page size*. We will further investigate the bit fields of the EPN, including the *page size*, in the following sections. Indeed, the MMU has no prior information regarding the page size of the virtual address before extracting the bits in the *page size* field. Once the *page size* bit field is revealed, the MMU reconfigures itself to operate relatively on a particular page size. Next, the MMU separates the offset field from the EA, considering the remaining bits consist the EPN field. In this implementation, the MMU is capable of

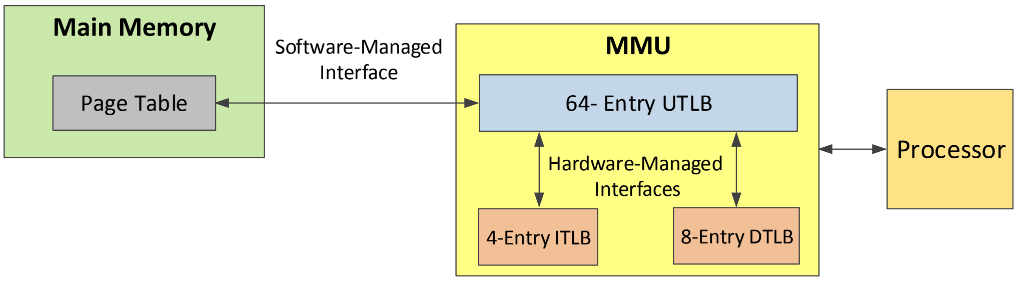


Figure 3.2: The basic organization of the page table residing in main memory along with the MMU.

reconfiguring itself to map 8 different page sizes, including 1, 4, 16, 64, and 256KB, as well as 1, 4, and 16MB, from the virtual space into the physical one.

3.5 The Infrastructure of the MMU

Figure 3.2 illustrates a basic scheme of the MMU and how it interacts with the page table. As previously explained, the page table residing in the main memory is completely managed by the software (the OS). Furthermore, the software manages the UTLB with 64 entries (the default format), however, it is located very close to the processor. Indeed, any modification to the UTLB, including initialization, validation, and entry replacements, is completely performed by the operating system. In addition to the UTLB, the MMU employs two micro-TLBs. By default, the MMU is configured with a 4-entry ITLB along with an 8-entry DTLB to cache the latest instruction and data address translations, respectively. Only the hardware (MMU) manages both micro-TLBs; initialization, entry replacement and validation is performed by the MMU. The fundamental idea behind using micro-TLBs is discussed in [51] in detail. Principally, the MMU exploits micro-TLBs to minimize the access conflicts of both data and instruction page translations, as well as speeding up the overall performance of the system.

3.5.1 TLB Organization

Figure 3.3 depicts the infrastructure of the MMU in more detail. In the real mode, the MMU just forwards the EA to the main memory assuming that the addressing is in real mode. In the virtual mode, the MMU first generates the VPN by concatenating the PID with the EA[EPN]. Next, the MMU inspects the corresponding micro-TLB to find a match with the generated VPN. In the case of a match, the MMU combines the RPN extracted by the micro-TLB with the offset to form the physical address. If a match was not found in the micro-TLB, subsequently, the MMU examines the UTLB. In the case of a hit, the physical address is produced in the same way to that of micro-TLBs. If a match was not found in the UTLB either, the MMU makes an *exception* to the operating system in order to report the cause. The entire translation process is carefully monitored by a controller. Moreover, the controller manages both hardware and software sides to modify the content of the TLBs. As a recall, each TLB has the following attributes:

The UTLB: The default configuration for the UTLB is 64-entries with a fully associative scheme. It is scalable during the synthesis time. The UTLB is fully managed by the operating system.

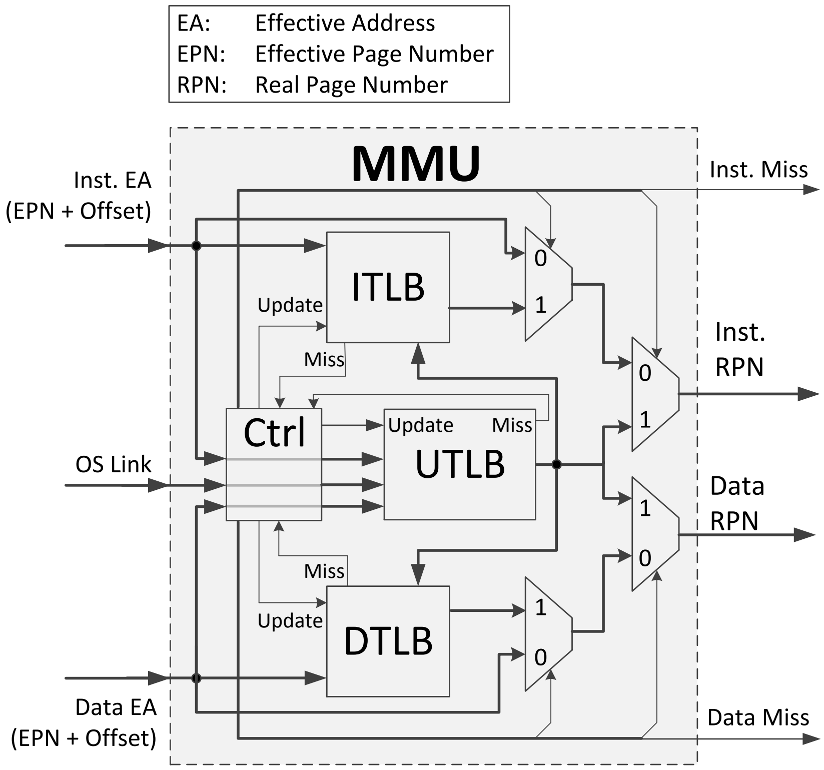


Figure 3.3: The infrastructure of the MMU with more details [P. IV] © IEEE, 2015.

The micro-TLBs: The ITLB/DTLB contains 4/8 entries to cache the most frequent instruction/data page translations. Same as the UTLB, the micro-TLBs are scalable during the synthesis time. Both micro-TLBs are managed by the MMU (hardware).

3.5.2 TLB Entry Organization

As Figure 3.4 presents, each entry in a TLB contains necessary information with respect to the virtual attributes, as well as the physical ones. Hence, each entry is divided into two sections, *TLBHI* which contains information about the virtual space alongside the *TLBLO* which contains the physical address attributes of the corresponding page. Each entry consists of 68 bits of which the first 36 bits are assigned to the TLBHI, and the remaining 32 bits are considered as the TLBLO. Technically, some of the bit fields which in both TLBHI and TLBLO (including U0, W, I, and M) are ignored in the virtual mode. Nevertheless, the most important bit fields are briefly explained as follows:

- TAG: The 22-bit "TAG" field is compared with the VPN[EPN]. The length of the comparison is defined by the *page size.*

- Page size: The 3-bit "page size" specifies the boundary between the offset and the EPN. It also clarifies how many bits in the TAG field should be compared with the given EPN. In this field, the value 0b000-0b111 respectively defines the range of the page size from 1KB to 16MB.

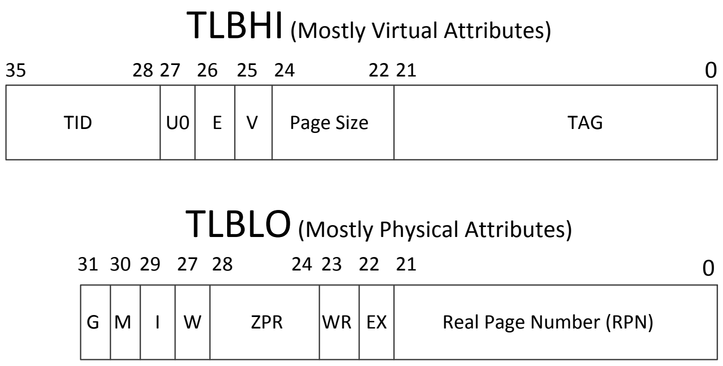


Figure 3.4: The organization of each entry in a TLB [P. IV] © IEEE, 2015.

- V: The single-bit "V" specifies whether the current page is valid in the main memory.
- E: The single-bit "E" defines the ordering of the bytes whether the page is accessed as little-endian or big-endian.
- TID: The 8-bit "TID" field is compared with the PID field of the current VPN.
- RPN: The 22-bit "RPN" represents the real page number of the corresponding VPN. The MMU generates the physical address using the RPN field.
- EX: The single-bit "EX" defines whether the instruction is executable (access control).
- WR: The single-bit "WR" indicates whether the page is read-only or writable (access control).
- ZPR: The 4-bit "ZPR" field selects one of the 16 different zone fields in the Zone Protection Register (ZPR) (access control).
- G: The single-bit "G" investigates if the current page is guarded. The term "guard" refers to a situation in which the speculative memory access such as instruction pre-fetch is prohibited.

3.5.3 Index Examination

As Figure 3.5 presents, an index to each TLB is examined in several steps to generate the physical address. Irrespective to the type of the TLB, all the entries are examined to find a match. In brief, a hit will occur when the index passes all the following examinations:

1. The TLB first investigates whether the corresponding entry is valid in the main memory before comparing other bit fields.
2. Then, the comparison between TLBHI[TID] and VPN[PID] is performed.
3. Next, the TLBHI[TAG] of the TLB is examined with the VPN[EPN] field of the index. The length of the comparison is based on the value given by the TLBHI[page size].

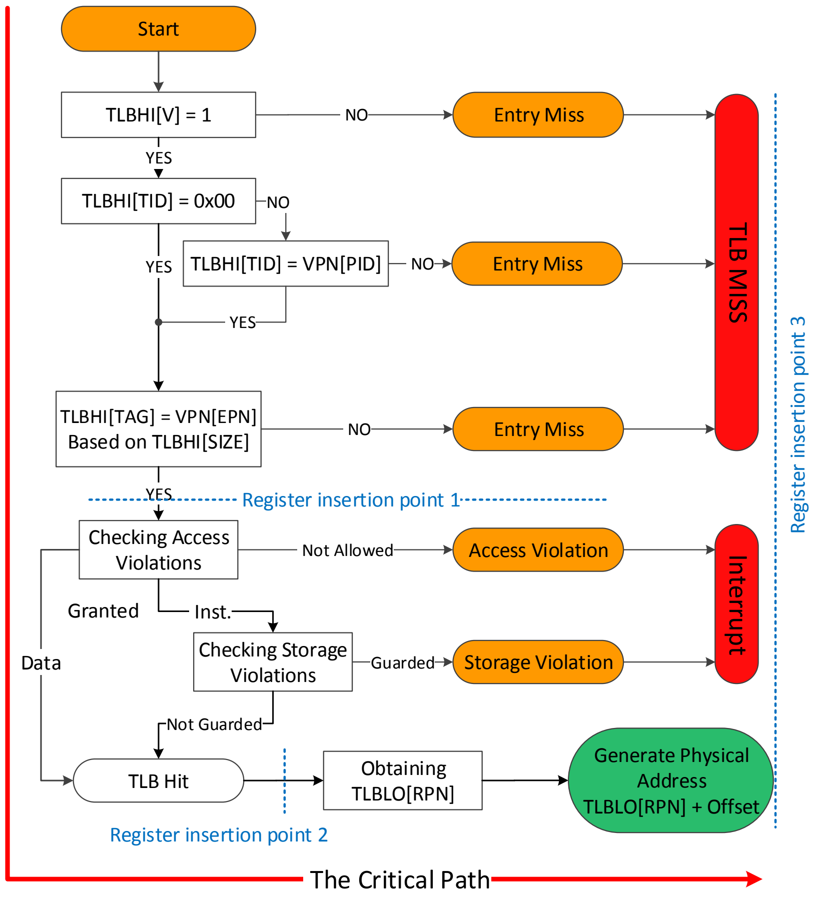


Figure 3.5: How an index examines the TLB. The longest critical path starts from *checking TLB Entry* to *Generate Physical Address* which is more visible in the UTLB. *Register Insertion Points* are the most suitable locations for inserting intermediate registers to minimize the critical path.

4. Eventually, the access violations followed by the storage violations are investigated.

In addition, TLBHI[TID] = 0x00 (see Figure 3.5) defines the current entry is a process-independent page translation, meaning that the page is accessible by all processes from the OS point of view. Hence, the comparison with the VPN[PID] is mitigated for these type of pages. In case a match between TLBHI[TAG] and VPN[EPN] was found, the corresponding TLB will check the access control bit fields. Next, the instruction VPNs should also be further examined to check whether the current page is not guarded, whereas the data VPNs bypass this stage. Moreover, Figure 3.5 presents the longest critical path, along with the register insertion technique exploited to minimize the critical path. We will further discuss about the timing analysis in the following Section 3.6.

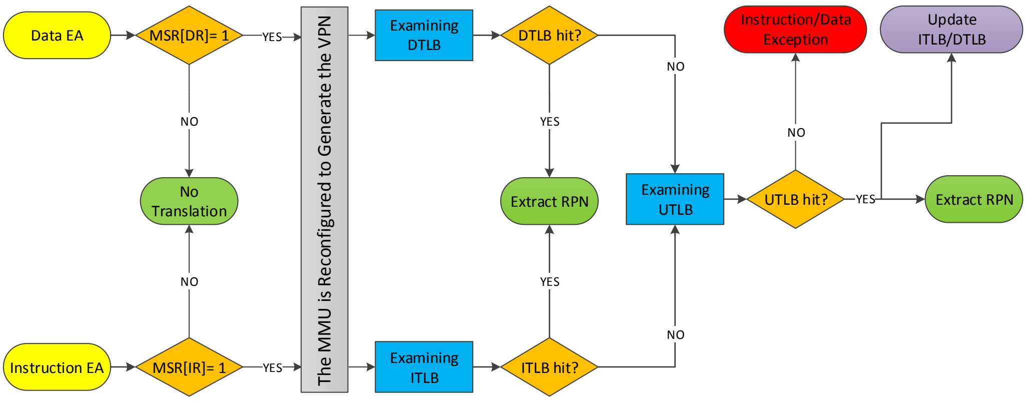


Figure 3.6: The virtual address translation flowchart

3.6 FPGA Implementations

The MMU is an IP module capable of reconfiguring itself on-the-fly to generate the appropriate VPN address under the control of the *page size* bit field of the EA[EPN]. Furthermore, all the TLBs are configurable and scalable during the synthesis time. We further investigate how scaling the UTLB would affect the overall performance. Therefore, we consider various configurations for the UTLB including 16, 32 and 64 entries.

3.6.1 Instruction/Data Page Translation Flow

Figure 3.6 illustrates how the MMU translates a virtual address into the physical one. The processor references to the MMU with both instruction and data virtual addresses. In the real mode, the MMU just bypasses both instruction and data addresses, being sure that both references are physical addresses which do not require translation mechanism. In the virtual mode, the MMU first distinguishes the offset and the EPN, followed by reconfiguring itself to operate on a proper page size. The MMU employs two specific bits (IR and DR) from the Machine State Register (MSR) to know in which mode the processor is running. Next, the MMU examines the appropriate micro-TLB to find a match with the corresponding VPN. It means that the MMU redirects the instruction address to the ITLB, whereas a data instruction is redirecting to the DTLB. In the case of a match, the RPN is extracted from the corresponding micro-TLB. In the other case, there is a miss occurred in the micro-TLB, meaning that the MMU should now investigate the UTLB to find the match. Additional delays are introduced to the system in the case of a micro-TLB miss. Similar to the hit in a micro-TLB, a successive RPN is generated after the UTLB hit. Meanwhile, the MMU is updating the micro-TLB in which the miss has occurred. In the case of a UTLB miss, the MMU reports an exception to the operating system via the interrupt handler. Accordingly, the MMU determines for what reason the miss was occurred, i.e. UTLB miss due to a missing entry, access violations, or storage violations (only for instruction pages).

In this implementation, we consider that the data address translation has priority over the instruction address translations. For example, in the case of simultaneous micro-TLB misses, the UTLB will address the DTLB miss, followed by the ITLB miss. We anticipate that the missing data would block the pipeline for the later instruction. Hence, the instruction fetch is not so urgent compared to the data.

Table 3.1: Minimum cycle counts for each virtual-to-physical address translation in an ideal case [P. IV] © IEEE, 2015.

	Data	**Instruction**
Micro-TLB hit	2	2
Micro-TLB miss, UTLB hit	5	5-7
Micro-TLB miss, UTLB miss	11	11-13

Table 3.1 reports the minimum clock cycle counts in each instruction/data address translation. A micro-TLB hit (real page extraction) occurs only in 2 clock cycles. In the case of a micro-TLB miss→UTLB hit, the physical address is generated within 5 clock cycles for the DTLB, and 5 to 7 clock cycles for the ITLB in the case of a simultaneous miss with the DTLB. If a miss also occurs in the UTLB, the fastest time that the OS can resolve the cause is 11 clock cycles for data and 11-13 clock cycles for instruction virtual addresses in an ideal case. The ideal case refers to a situation where the OS has the ability to instantly update the MMU. However, how the operating system resolves the cause is entirely depended on the status of the OS. Typically, today's operating systems can handle the situation in hundreds or even thousands clock cycles. Perhaps, future technologies, such as prediction, might lead to a point that the OS can update the MMU in a faster way.

When a miss is reported by the UTLB, the MMU issues an interrupt to the *interrupt handler* module. Consequently, the processor enters the real mode, while it informs the MMU by clearing the appropriate bit fields in the MSR. During the interrupt handling process, the MMU is disabled since all the references to the MMU are physical addresses. Once the operating system updates the MMU with the required instruction/data page address, the processor shall (re)enter the virtual mode. Having updated the UTLB, the MMU invalidates micro-TLBs by clearing the TLBHI[V] bit at each entry to protect instruction/data consistency between micro-TLBs and the UTLB.

3.6.2 The Critical Path Reduction

The default configuration for the UTLB in the MMU is 64 entries. As earlier explained, we experienced scaling the UTLB with three design-time configurations of 16, 32 and 64 entries. What we discovered was the fact that the performance of the MMU configured with 64-entry UTLB was drastically lower than other configurations. It is mainly caused due to a large number of entries. As a rule of thumb, the larger the number of entries, the longer the inspection time. Indeed, the hardware circuit is always bounded by the slowest operation of the design. Hence, we exploited two techniques to minimize the critical path, as well as breaking large computational tasks into several subtasks: *Register Insertion (RI)* and *Parallel Computing*, respectively.

The Register Insertion technique is quite similar to the one earlier discussed in Chapter 2.8.2 and is shown in Figure 2.13. The Register Insertion technique is one of the most promising solutions to cope with long critical paths in a design. The most recommended location for Register Insertion is visible in Figure 3.5.

The Parallel Computing is a technique exploited in computationally intensive tasks in which most of the calculations are executed concurrently. In Parallel Computing, a large computational task is divided into two or more subtasks which are inherently independent.

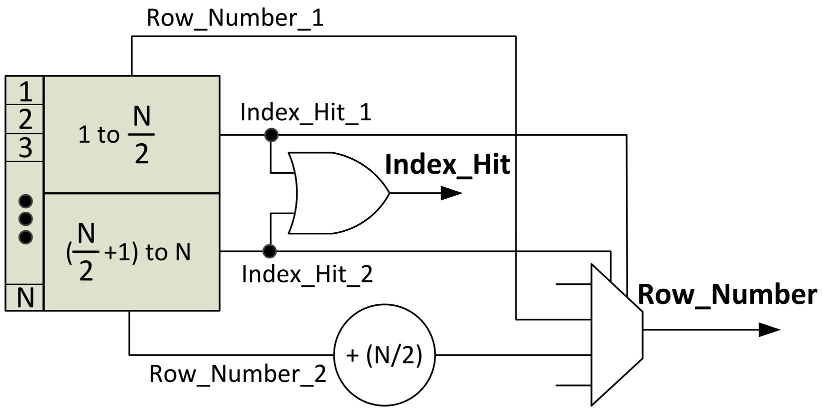


Figure 3.7: How to minimize the complexity of searching an index in the UTLB using Parallel Computing technique [P. IV] © IEEE, 2015.

Back to the UTLB implementation, Figure 3.7 presents how the UTLB exploits Parallel Computing to minimize the complexity of searching an index through the N number of entries. In this figure, an N-entry TLB is split into two sub-TLBs. During the index examination, the corresponding TLB searches for a matched entry in both sub-TLBs simultaneously. Once a hit occurs in either of the sub-TLBs, the corresponding physical attributes of the matched entry are simply extracted by the MMU. This method is very similar to unfolding technique widely used in DSP implementations. It is worth

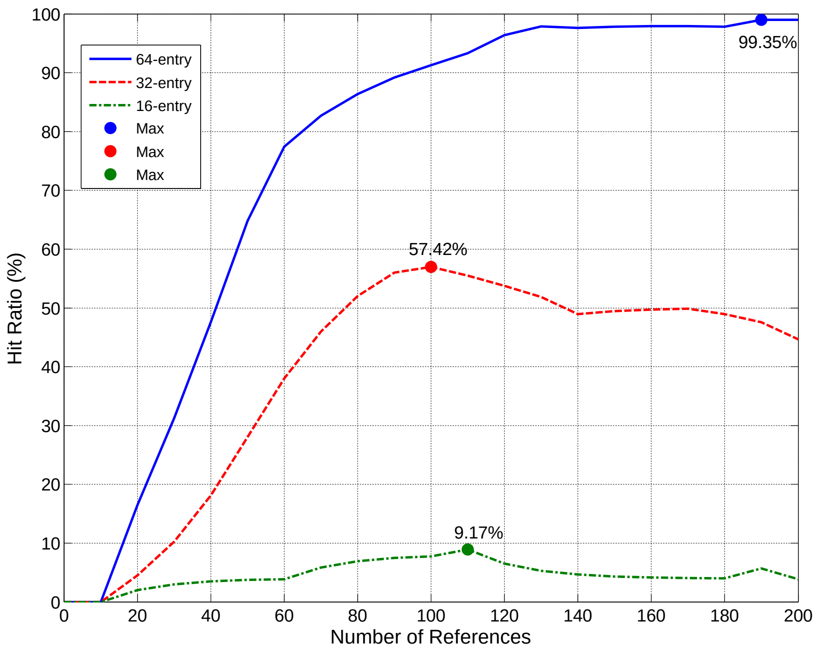


Figure 3.8: The hit rate with different number of entries in the UTLB for 200 random references after the boot-up. The curves are the smoothed version of original ones for intervals of 10 references [P. IV] © IEEE, 2015.

mentioning that parallelization has the potential to increase resource usage to some extent. Nevertheless, we only break the examination task at the UTLB into only two levels of parallelization in order to reach to the target operating frequency, while maintaining the hardware cost at an acceptable level. Indeed, the MMU could achieve the target operating frequency of 200MHz with 2 levels of parallelization. In that respect, further information is provided in section 3.8.

3.6.3 The Effect of Scaling the UTLB

As earlier mentioned , the MMU is configured with 16/32/64-entry UTLB in order to observe the effect of scaling the UTLB on the overall performance. Figure 3.8 depicts how the UTLB with different configurations responds to a subset of 200 stochastic numbers (including repetitive sequence) after the boot-up. The stochastic numbers present the worst-case scenario for the MMU, while better results are expected in real applications. Furthermore, we assume that all the references to the MMU have 1KB in page size and all the references are accessible. The random number generator is designed based on the specifications given in [52]. Primarily, we used random numbers to demonstrate the behavior of the overall system.

As Figure 3.8 illustrates, roughly the first 15 references to the UTLB are not returned by any hit due to the presence of compulsory misses. It seems that the operating system fills the UTLB with different references for approximately the first 15 references. Thereafter, future references have the potential to be returned with a hit since the UTLB is loaded with several entries. Based on the curves in the same figure, the 16-entry UTLB does not present acceptable result due to a large number of capacity misses. The 32-entry UTLB

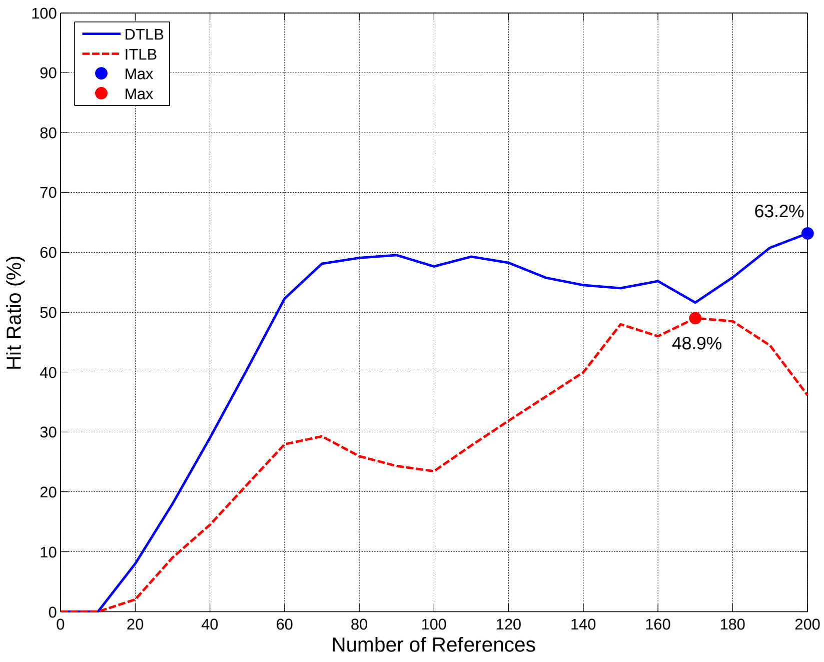


Figure 3.9: The effect of employing micro-TLBs for the same 200 random data/instruction patterns after the boot-up. The curves are the smoothed version of original ones for intervals of 10 references, as well [P. IV] © IEEE, 2015.

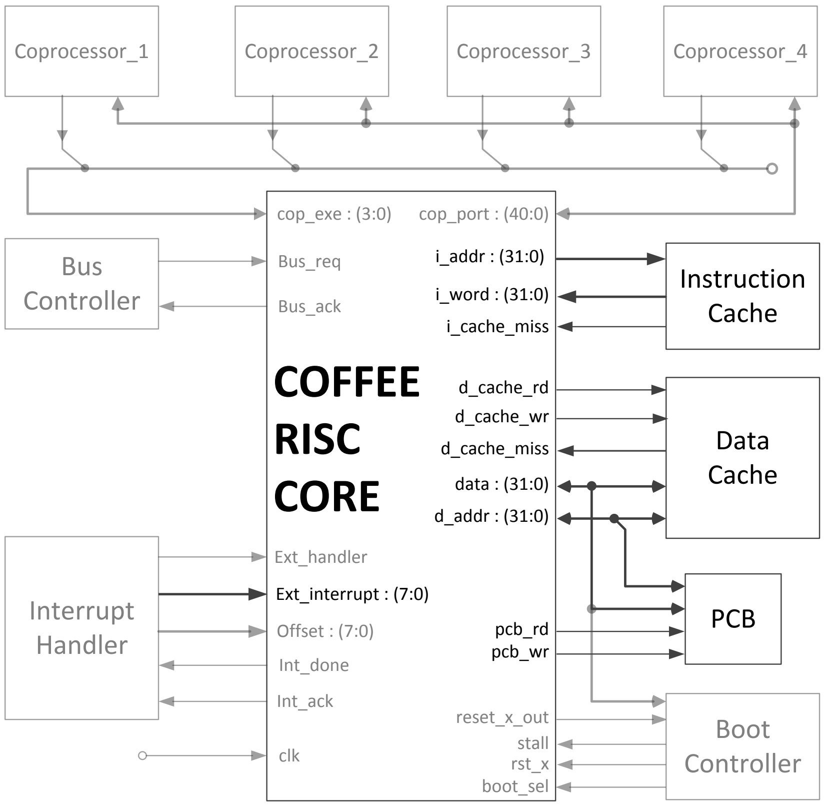


Figure 3.10: The integration of the COFFEE RISC core [P. V] © Springer, 2016.

seems more responsive to the same test vector, however, the approximately 57% hit rate is still not acceptable. We anticipate that the 32-entry UTLB has the potential to be employed in some real applications. The UTLB configured with 64 entries seems very promising to respond to the same test vector with 99% hit rate. In addition to that, the 64-entry UTLB shows a very stable hit rate after approximately 130^{th} reference onwards. It implies that the same hit rate is expected for larger test vectors.

3.6.4 The Effect of Employing Micro-TLBs

We applied the same test vector, that we used to examine the UTLB, to evaluate whether using micro-TLBs would be beneficial for the MMU. As Figure 3.9 illustrates, both DTLB and ITLB respond to the worst-case scenario (random numbers) to some extent. However, quite many compulsory and capacity misses do not allow the micro-TLBs to offer hit rate more than 63% and 48% in DTLB and ITLB, respectively.

3.7 Integration Issues

This section provides some essential information how the developed IP-based MMU can be integrated to any standard RISC computer. The case study is the COFFEE RISC processor which is described in Appendix A. Figure 3.10 presents a black box view of the COFFEE processor in which the main components required for integrating the MMU are more visible. Technically, there are two alternatives to integrate the MMU with the target processor. One possible solution is to integrate the MMU via the co-processor interface. Architecturally, the COFFEE references to the data of each co-processor within a 64-entry shared register bank of which each co-processor accesses to 16 (32-bit) registers. However, the address space in each co-processor is adequately wide to address all the required registers at the MMU. One of the primary drawbacks of the co-processor interface is the complicated timing issues which potentially occur in case the MMU is integrated with the core over the co-processor bus. Another constraint is the fact that the core cannot simultaneously reference to a co-processor to request for both data and instruction virtual address translations. It means that both data and instruction address translations can not be concurrently referenced.

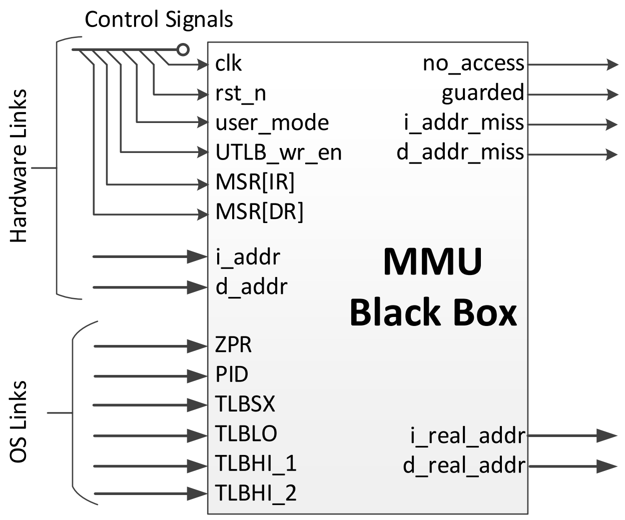
The other alternative is to integrate the IP-based MMU via the Peripheral Control Block (PCB) interface. The PCB interface offers some advantages over the co-processor bus. The foremost one is that no timing issue will be arisen during the translation process. The main reason is that the PCB interface is directly integrated to the shared memory bus. Another advantage is the faster access to the required data, specially when the OS updates the MMU. In addition, the fundamental idea in the design of the COFFEE core is to integrate peripheral modules such as the MMU via the memory-mapped shared data bus. Nevertheless, we will explain how to create a wrapper to integrate the MMU with the COFFEE processor via the PCB interface in the next subsection.

3.7.1 Hardware/Software Interconnections

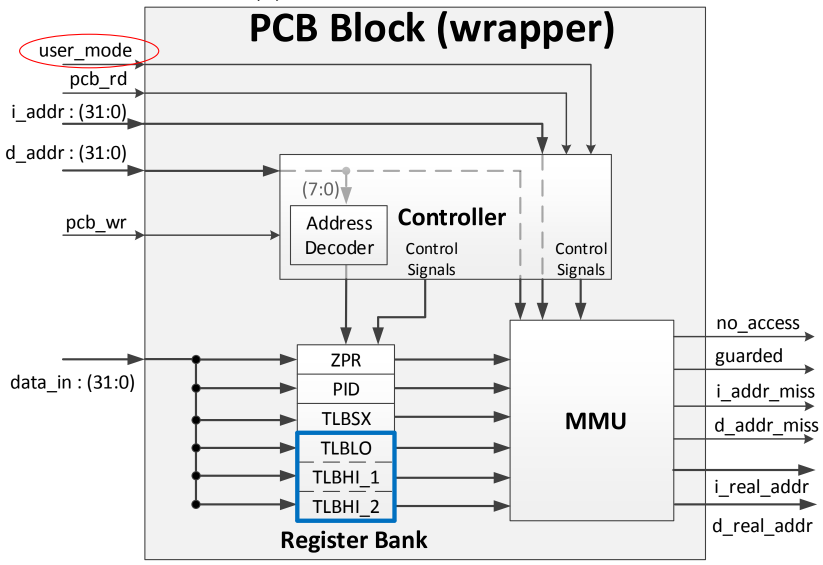
Figure 3.11a illustrates the IP-based MMU as a black box. The most of the I/O ports are named with appropriate labels in order to improve the readability. The input port *user_mode* indicates whether the current state of the processor is in either *user mode* or *super-user mode*. The super-user mode, also known as privileged mode, is a specific mode typically used by the operating system. In super-user mode, the OS can manipulate all the registers (such as ZPR) alongside overriding access and storage controls. The user mode refers to a situation when the processor is executing other applications. Indeed, not all the registers are accessible in this mode. As a recall, since the TLBHI consists of 36 bits, we broke the TLBHI field in two segments TLBHI_1 and TLBHI_2. Moreover, the operating system informs the MMU to replace the new entry at the UTLB through the TLBSX input signal. In addition, all the OS links in the same figure are 32 bits wide.

The MMU requires all the input ports shown in Figure 3.11a to successfully translate a virtual page into the physical one. However, several input ports, in particular the ones related to the OS, are not physically provided by the core. Indeed, there is no physical connection between the OS and the processor. In general, one possible way to exchange data between the processor and the OS is to use a set of specific registers in the processor accessible by the operating system. Hence, the OS reads/writes from/to some of those specific locations in the processor and, subsequently, the processor hands over the latest updates to the MMU. Finally, the MMU updates the UTLB with recently achieved information from the processor. In order to provide the above-mentioned links, we need to use a wrapper to mimic some missing inputs for the MMU, along with the processor's specific registers.

Figure 3.11b presents the wrapper (infrastructure of the PCB block) surrounding the MMU. In addition, a register bank is included in the wrapper which resembles the specific register of the processor where the OS can communicate with. The wrapper also includes



(a) The MMU as a black box



(b) The infrastructure of the PCB block

Figure 3.11: How a wrapper integrates the MMU with the COFFEE core. The OS links are presented as a register bank in which the TLBLO, TLBHI_1 and TLBHI_2 are atomic operations. The status of the processor is indicated through the signal *user_mode* [P. V] © Springer, 2016.

a controller to monitor all the transactions. In the following, the infrastructure of the wrapper is explained in detail.

In the first step, the COFFEE processor does not provide the *user_mode* signal for the MMU. Hence, one integration challenge is to find a way to update the MMU with the current status of the processor. The COFFEE core has an 8-bit read-only register known as Processor Status Register (PSR) in which the first bit, PSR[0] or PSR[UM], provides the user with the current status of the processor. Fortunately, the required information can be obtained by digging into the hardware description of the core. Hence, the user is permitted to modify the hardware description of the core to suit their interest due to the open-source nature of the COFFEE core.

The input signal *pcb_rd* driven from the processor indicates the addressing mode. When the processor logically sets the *pcb_rd = '0'*, the current addressing mode of the core is real. Hence, no translation is required and, consequently, the input *i_addr/d_addr* is redirected to the corresponding output port *i_real_addr/d_real_addr*. The logical value '1' on the *pcb_rd* informs the controller that the current addressing mode is virtual. Thus, the controller sets the *MSR[IR]/MSR[DR]* to the logic '1', informing the MMU to translate the references.

In the virtual mode, once the MMU fails to translate the given address due to a miss, the corresponding miss signal (instruction, data or both) is set. The COFFEE core is informed about the cause through the *i_addr_miss/d_addr_miss* output ports. In other cases, i.e. access and storage violations, the PCB block interrupts the core via the external interrupt ports (no_access and guarded signals in Figure 3.11b). We will further explain how the PCB block can interrupt the core later on. In any case of failure, the PCB informs the core, meaning that the operating system should resolve the issue. Next, the core acts as follows:

- The port *user_mode* is cleared to logic '0' to give the control to the operating system.
- The port *pcb_rd* is immediately cleared to turn off the address translation.
- The port *pcb_wr* is set to '1', as soon as the operating system is ready to update the MMU.

As earlier studied in subsection 3.7.1, the operating system can communicate with the processor via a set of specific registers of the processor. The COFFEE RISC core exploits some specific register banks such as Core Configuration Block (CCB) and PCB (see Appendix A). The OS can communicate with the processor via the CCB register bank. The address range of the CCB is 256 subsequent addresses starting from "0001 0000h" ending to "0001 00FFh". At this stage, CCB registers that the operating system communicates with is not a concern of the MMU.

The next 256 consecutive addresses are employed by the COFFEE core to reference to the PCB block. Nevertheless, the base address of the PCB is initially set by Hexadecimal value "0001 0100h" after the boot-up, followed by the next 256 consecutive addresses (up to "0001 01FFh"). We can roughly state that the operating system can communicate with the PCB block through the core in that way. In this implementation, we assumed that the OS references to the register bank in the wrapper (ZPR, PID, TLBSX, TLBLO, TLBHI_1 and TLBHI_2) via the address range "0001 0100h" to "0001 0105h", respectively. Once the input signal *pcb_wr* is set to logic '1', the controller is informed that the operating system is willing to communicate with the MMU. Nevertheless, the controller employs the *address decoder* block to update the appropriated block of the register bank. The eight least significant bits of the *d_addr* indicate which field of the register bank is being updated. For example, the value "0001 0102h" on the *d_addr* bus indicates that the operating system is willing to update the contents of the TLBSX register. The only exceptional condition during the updating process is that the operating system should update the TLBLO, TLBHI_1 and TLBHI_2 together in an atomic operation. Atomic operations

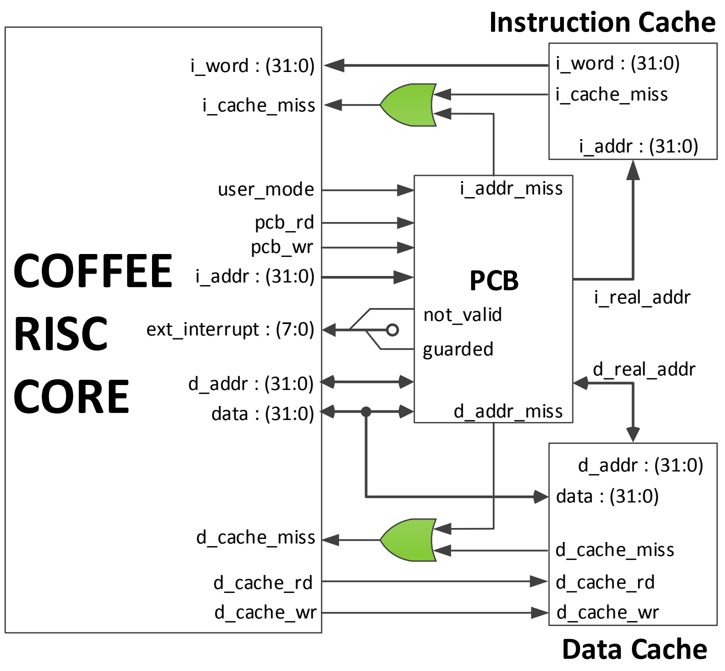


Figure 3.12: PCB integration with COFFEE RISC core. The PCB employs two bits of the *Ext_interrupt* signal to interrupt the core. Adding two OR gates requires to modify the hardware description of the core interfaces [P. V] © Springer, 2016.

are not permitted to be suspended by any means. Once the OS finishes updating the register bank, the *pcb_wr* signal is cleared and the controller subsequently applies the changes in the register bank to the MMU via the *UTLB_wr_en* (see Figure 3.11a).

3.7.2 Integrating the MMU With the COFFEE Core

As previously mentioned, the PCB interface has priority over the co-processor interface to integrate the MMU with the COFFEE. Figure 3.12 shows a block diagram how the MMU can be integrated with the core via the PCB interface. The data address bus *d_addr*, as well as the instruction address bus *i_addr*, are connected to the data and instruction caches via the PCB block. The core manages the behavior of the MMU via the *pcb_rd* and *pcb_wr* ports in order to prohibit unintentional read and write to the MMU. When the MMU experiences an access/storage violation, it can interrupt the core via the *not_valid/guarded* external port. Technically, the MMU reports the violation causes to the core via the last two available slots of the *ext_interrupt* interface. Unfortunately, there was no available signal to integrate the instruction miss, alongside the data miss via the *ext_interrupt* port. Hence, a new method should be implemented to inform the core about the missing virtual pages. Once again, we modified the hardware description of the core, this time the interfaces. We employed the instruction/data cache miss ports to inform the core that there is a missing page in the MMU. However, the *i_cache_miss*, alongside the *d_cache_miss* interface is reserved for the real cache misses. Hence, two simple *OR* gates (as Figure 3.12 shows) could resolve the conflict between the external outputs of the caches and the MMU. At this stage, the new challenge is how the processor can distinguish, e.g., whether the asserted miss is relevant to the cache or the

Table 3.2: How the Register Insertion, as well as Parallel Computing improves the overall performance of the 64-bit UTLB [P. V] © Springer, 2016.

	Baseline MMU	**Register Insertion**	**Parallel Computing**
Worst Critical Path (ns)	6.164	5.154	4.935
Clock Skew (ns)	−0.054	−0.130	−0.020
Maximum Freq. (MHz)	163.59	193.09	200.80
Speed-up	–	1.18×	1.23×

MMU. In this situation, we assume that the processor refers to the *ext_interrupt* port. If both *guard* and *no_access* signals are asserted, the incoming miss is asserted by the MMU. Otherwise, the miss is reported by the corresponding cache. Indeed, the controller overrides both *guard* and *no_access* signals to inform the processor about the miss. This scenario requires a heavy modification to the hardware description of the core. Hence, we have already postponed the mentioned assumption for the future when the operating system is completely provided for the core.

3.8 Synthesis Results

The following results are based on the same specifications and environment explained in Chapter 2, section 2.8. Table 3.2 presents the critical path analysis of the MMU configured with 64-entry UTLB. The RI technique increased the maximum operating frequency to 193MHz (about 18% speed-up). However, the RI introduces few extra delays during the translation process. The second level of the optimization, parallel computing, could rise the operating frequency above 200MHz (23% speed-up), meaning that the design could meet the ultimate operating frequency target. Hence, the MMU configured with 64-entry UTLB is the main interest in our design. The negative clock skew reveals that the MMU can potentially operate slightly slower than the introduced timing constraints.

Table 3.3 compares different configurations of the MMU with each other. The maximum operating frequency achieved by each configuration is about to 200MHz, 229MHz, and 267MHz, respectively for the MMU configured with 64-,32- and 16-entry UTLB. Although the MMU with 16-entry UTLB is the lightest implementation and the fastest design, the massive number of misses (compulsory and capacity misses) degrade the overall performance to the point that we do not recommend this architecture. The MMU configured with 32-entry UTLB has a decent potential to be employed in some practical applications. The MMU configured with 64-entry UTLB is the main target of our interest since it has a high performance and it could meet our target operating frequency (+200MHz).

Table 3.4 compares all architectures together in terms of energy consumption. All the configurations have almost the same figure for the static energy consumption, as well as the I/O one. The static energy is a characteristic of the FPGA chip, not the design. The dynamic energy consumption of the 32-entry UTLB is trivially more than the 16-entry UTLB, while the 64-entry UTLB almost doubles the dynamic energy consumption in comparison with 32-entry UTLB. It is interesting that the overall energy consumption of the 32-entry UTLB is lower than the 16-entry UTLB. Moreover, the total energy

Table 3.3: Summary of implementing MMU with different configurations [P. V] © Springer, 2016.

	Memory Management Unit		
	UTLB_64	UTLB_32	UTLB_16
Logic Utilization (ALMs)	3,459	2,092	1,373
Total Registers	5,263	3,221	2,213
Maximum Freq. (MHz)	200.80	228.57	267.38

consumption of the 64-entry UTLB is slightly more than the other two candidates.

Table 3.4: Energy consumption analysis for the first 200 references (μJ) [P. V] © Springer, 2016.

Thermal Energy	**Memory Management Unit**		
	UTLB_64	UTLB_32	UTLB_16
Static	14291.10	14278.20	14275.95
Dynamic	326.25	167.85	145.65
I/O	431.55	427.20	454.80
Total	15048.90	14873.25	14876.25

Table 3.5 reports the total energy consumption in each module with more detail. Since the result for the static energy in all configurations was quite similar and it is a characteristic of the FPGA chip, we exclude those results in this correspondence. Overall, the statistics show that both micro-TLBs consume very trivial energy compared to the rest of the components. The UTLB with 64 entries has a large impact on overall energy consumption of the system.

Table 3.5: Energy consumption of each component for the first 200 references (μJ) [P. V] © Springer, 2016.

	Dynamic Energy			**Routing Energy**			**Total Energy**		
	64	**32**	**16**	**64**	**32**	**16**	**64**	**32**	**16**
MMU	78.45	73.95	100.65	120.90	81.90	70.50	199.35	155.85	171.15
ITLB	7.80	7.95	7.50	4.65	4.20	4.65	12.45	12.15	12.15
DTLB	13.95	14.55	15.40	8.10	8.85	9.15	22.05	23.40	24.55
UTLB	**81.15**	29.10	18.45	**75.30**	11.10	11.85	**156.45**	40.20	30.30
Ctrl.	0.60	0.75	0.75	9.00	4.95	3.45	9.60	5.70	4.20
Total	181.95	126.30	142.75	217.95	111.00	99.60	**399.90**	**237.30**	**242.35**

Table 3.6 depicts the final summary of the system after integrating the COFFEE processor with the MMU configured with 64-UTLB. The final implementation requires 8403 ALMs of which about 55% is the hardware cost of implementing the core, 43% is employed by the MMU, while the remaining 2% ALMs is used in other components such as memory

interfaces and tri-state buffers. In addition, the overall system requires 10,884 registers of which about 50.5% are used by the COFFEE, while the remaining 49.5% are employed by the MMU. Moreover, the COFFEE RISC processor requires 57% of the combinational Adaptive Look-Up Tables (ALUTs), while about 40% is needed by the MMU. It is worth mentioning that integrating an MMU configured with 64-entry UTLB is as hardware expensive as the COFFEE core itself.

Table 3.6: Summary of the hardware implementation of the COFFEE processor integrated with the MMU [P. V] © Springer, 2016.

	ALMs	**Total Registers**	**Combinational ALUTs**
COFFEE	4,635 (55.16%)	5,493 (50.47%)	6,425 (57.25%)
MMU	3,594 (42.77%)	5,382 (49.45%)	4,535 (40.41%)
Other Blocks	174 (2.07%)	9 (0.08%)	263 (2.34%)
Total Design	8,403	10,884	11,223

Table 3.7 summarizes the maximum achieved operating frequency separately for the COFFEE, MMU and the integration of both. It is significantly important that the COFFEE core could successfully achieve the same operating frequency after the integration in the most practical environment (slow model at 85 ℃). The speed-optimized version of the COFFEE could run at 180MHz operating frequency, while this frequency is degraded by 2MHz, to 178MHz, after the integration. Moreover, the same table presents the worst critical path analysis of the COFFEE, MMU and the integration of both components in detail. It seems that the critical path of the design is propagated through the processor elements rather than the MMU. The 0.366ns positive slack in the core implies that the post-fit netlist can potentially meet the timing constraints introduced to the analyzer. On the other hand, the MMU alongside the COFFEE+MMU could roughly meet the timing requirements. Overall, the most speed-optimized version of the processor integrated with the MMU offers a negligible degradation in performance.

3.9 Concluding Remarks

This chapter presented developing an IP-based reconfigurable Memory Management Unit (MMU) on FPGA. The MMU was capable of reconfiguring itself to operate on different virtual page sizes, including 1, 4, 16, 64, 256KB, 1, 4, and 16MB. In addition, the MMU employed several Translation-Lookaside Buffers (TLBs) in two levels of hierarchy. In the first level, there were two hardware managed micro-TLBs working in parallel to cache a subset of latest instruction and data virtual page translations, respectively. In the second level, a software managed Unified TLB (UTLB) existed to cache the most recently used instruction and data address translations. All the mentioned TLBs were design-time configurable to offer the maximum flexibility to the MMU. Next, we experienced the effect of scaling the UTLB in different sizes (with 16, 32 and 64 entries) on the overall performance including hit rate, maximum operating frequency, resource usage, and energy consumption. We further investigated the critical path analysis, as well as some optimization methods such as concurrency and register insertion to minimize the critical path in order to reach to the target operating frequency. The MMU configured with 64-entries UTLB showed satisfactory results making it the preferred choice. In the

Table 3.7: Maximum operating frequencies as well as the worst critical path analyses of the COFFEE, MMU and their respective integration [P. V] © Springer, 2016.

			COFFEE	**MMU**	**COFFEE+ MMU**
Maximum Frequency (MHz)	*Slow Model*	85 °C	180	200	178
		0 °C	188	214	182
	Fast Model	85 °C	267	295	253
		0 °C	289	318	271
Worst Critical Path Analysis (ns)	*Data Delay*		5.604	4.935	5.716
	Interconn. Delay		4.077	3.623	4.313
	Cell Delay		1.527	1.312	1.403
	Clock Skew		-0.035	-0.020	-0.007
	Time Slack		0.366	0.020	0.072

next step, we investigated how to integrate the MMU to our target RISC processor, the COFFEE core. The integration required to modify the hardware description of the core to extract some crucial information such as processor status. In addition, a wrapper was needed to incorporate software accessible registers in the COFFEE core. Integrating the MMU with the COFFEE RISC core did not degrade the overall performance of the processor in terms of operating frequency. The 2MHz loss in the operating frequency was the only degradation of the core in the most practical environment. Eventually, the hardware implementation results after the synthesis were presented with respect to the hardware costs, energy consumption, etc.

4 Reconfigurable IP-Based Controller Area Network Module

The content of this chapter is extracted from the Publication [P. VI]. In this chapter, we mainly emphasize on developing a reconfigurable Controller Area Network (CAN) module. The CAN module is developed as an IP block integrable to any RISC processor, in particular, the COFFEE core. The synthesis results show that the CAN module is a very low-cost component which also consumes very low energy to operate. Later on, we will employ the CAN module as a co-processor loosely-coupled with the COFFEE core. This integration increases the versatility of the COFFEE core, even for the industrial purposes.

4.1 Background

The Controller Area Network (CAN) is an in-vehicle serial communication protocol developed by Bosch in mid-1980s [53]. The CAN protocol is an asynchronous serial communication system based on the principle of Carrier Sense Multiple Access/Collision Resolution (CSMA/CR) media access control [54]. The CAN protocol has a simple and robust bus architecture which is capable of operating up to 1Mbps rate. The CAN protocol is very practical in microcontroller-based systems where two (and more) nodes can communicate with each other without the supervision of a master node. Prior to the invention of the CAN, electronic devices used point-to-point wiring system to transfer data, which significantly increased the overall cost of the system. Potentially, the CAN protocol reduces the cost of a system by a factor of thousands. For example, a high-end luxury vehicle consumes 5km wires which is weighted over 100kg [55]. With the advent of the CAN protocol, point-to-point wiring system is simply replaced by a twisted pair medium to provide an efficient solution to the long and heavy wiring problems posed in modern vehicles. Figure 4.1 illustrates a system in which all the nodes are connected together via the CAN protocol.

4.2 Motivation

As discussed in previous chapters, the COFFEE RISC core is an open-source embedded processor. The processor has proved its versatility in embedded computing tasks, including telecommunication, embedded computing, etc. The ultimate goal is to make the core more versatile even for industrial purposes. For example, imagine an environment where the life of the technician steering a heavy industrial machinery is being jeopardized. We are intending to deploy the system in such a way that the technician can remotely control the truck through an industrial joystick, while the COFFEE processor maintains the

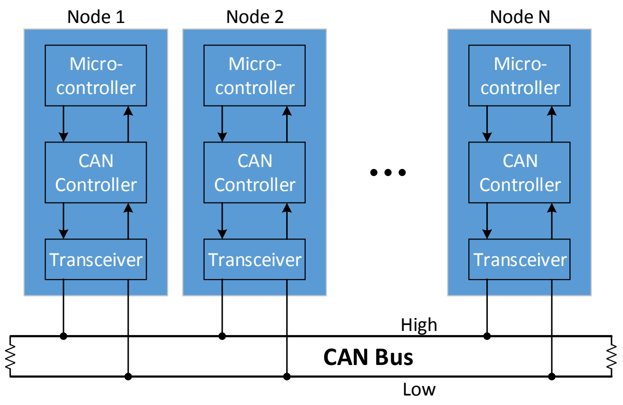


Figure 4.1: A CAN-based system. Reconstructed from [56].

machinery by detecting, receiving, and extracting the issued command. The technician can be either in a cockpit close to the machinery (using physical media) or the machine can be remotely controlled via the wireless media. In that respect, Bayilmis et al. have developed a platform in which the CAN2.0A was segmented over the IEEE 802.11b wireless Local Area Network (LAN) [57]. In another study, Shin et al. have proposed a hybrid network composed of a CAN protocol and ISA100.11a (industrial wireless standard) [58].

4.3 CAN Protocol Specification

Early in the 1990s, CAN protocol was standardized as an alternative solution to replace in-vehicle point-to-point wiring system. Nowadays, the CAN protocol is used in other devices such as aircraft engines, medical equipment, and cameras. CAN protocol enables real-time control mechanism in the system while maintaining the security at the highest level. The CAN protocol has many advantages of which a few of them are listed as follows [59]:

- *Low Cost*: CAN provides an inexpensive communication channel, so that each node has a single interface instead of several inputs.
- *Lightweight Network*: The CAN bus consists of a simple twisted pair with almost unlimited number of nodes attached to it.
- *Broadcast Communication*: All the nodes can see the transmitted message on the bus; many of them drop the message, while few of them decide to process the data.
- *Prioritized Messages*: All the CAN-based nodes are prioritized. Therefore, each message has its own priority, meaning that a node with higher priority transmits the data on the shared bus.
- *Robust Error Handling*: The CAN protocol employs the Cyclic Redundancy Check (CRC) to perform error handling. In addition, when a node detects an error, it will broadcast an error frame over the bus to inform other nodes.
- *Easy to Maintain*: Nodes can simply be added/removed from the network without introducing additional cost and complexity to the system.

- *Bit Overriding Mechanism*: Messages on the CAN bus are broadcast either as a *recessive* bit (logic '1') or *dominant* bit (logic '0') where the dominant bit can override the recessive bit.

4.4 The Structure of a CAN Frame

In general, four types of the frame can be broadcast in a CAN-based system. These four types are categorized as follows:

- *Data Frame*: is used to transmit the message of a node to another one.
- *Remote Frame*: is transmitted by any node which requires data from another node.
- *Error Frame*: is broadcast by any node over the CAN network to all nodes in the case of detecting an error.
- *Overload Frame*: is used when the receiving node cannot process all the receiving information being sent to it.

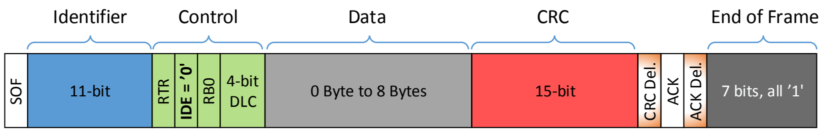
In this correspondence, we mostly emphasize on data frames since extracting messages is the first priority of the system. However, readers interested to discover more about other CAN frames are advised to refer to [60] in which Lawrenz has provided a wealthy reference.

4.4.1 Standard Data Frame

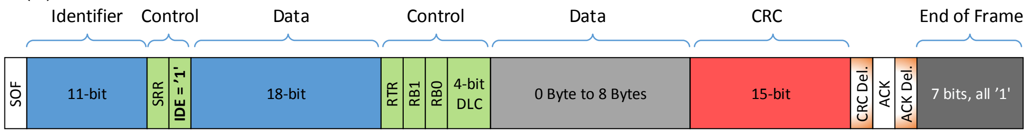
Figure 4.2a depicts the standard format of a data CAN frame. The first segment of each CAN frame is a single bit *Start-of-Frame (SOF).* Following the SOF, an 11-bit *arbitration field* is represented. This field is also known as *identifier* and it is unique for each node. The identifier reveals the identity of the node, priority of the message, and the logical address of the node. The 7-bit *control field* indicates information with respect to the type of the frame, whether it is a remote frame (Remote Transmission Request (RTR)), and if the identifier field is extended (Identifier Extension (IDE)). There is also a bit reserved for future extension. The next following 4 bits (Data Length Code (DLC)) in the control field determine the data length. The length of the *data field* varies between 0-8 bytes for each packet. The 16-bit *CRC field* is transmitted following the data. The CRC field confirms the integrity of the entire frame against different type of errors. The next three bits are *CRC delimiter*, *Acknowledge (ACK)*, along with the *ACK delimiter*, respectively. The last 7 bits in a standard data CAN frame are the *End-of-Frame (EOF)* which reports the transmission is terminated. It is worth mentioning that the EOF, as well as the delimiters, is presented as a sequence of recessive bits.

4.4.2 Extended Data Frame

Figure 4.2b illustrates the extended data frame. The overall structure of this frame is quite similar to the standard one, except the identifier field. In the extended mode, the arbitration field is extended to 29 bits. Hence, the extended frame is capable of addressing up to $2^{29} = 536,870,912$ nodes. The CAN module identifies an extended frame when it receives a recessive IDE (IDE='1'), and the standard frame when the IDE is sent as dominant (IDE = '0'). In this implementation, the CAN module is able to



(a) Standard data frame can support up to $2^{11} = 2048$ number of nodes.



(b) Extended data frame can support up to $2^{29} = 536,870,912$ number of nodes.

Figure 4.2: Data frames: standard frame as well as the extended frame

reconfigure itself to operate either on standard frames or extended frames during the run-time. However, the default configuration of the module is to operate on standard data frames. In addition, the extended frame employs an additional bit in the control field known as Substitute Remote Request (SRR). This bit is always set as recessive to prioritize the standard frame over the extended one in the case of two simultaneous arbitrations.

4.4.3 Bit Stuffing

In essence, any sequence of 5 consecutive bits of the same level should be followed by an opposite bit level. This procedure is called *bit stuffing*. The only exception is in the EOF field in which 7 recessive bits are transmitted at the end of the frame. Otherwise, a bit stuffing error is reported by the receiving node. The receiving node ignores the bit stuffing error, while the EOF is being transmitted.

4.5 Design Considerations

Decades ago, one could integrate a CAN-based product using either a standard controller chip, a standard microcontroller with a built-in CAN interface, or even an ASIC with CAN interface. Today, a better solution is added to the ones mentioned above which are the FPGA-based CAN modules [61]. So far, there have been many studies on IP-based CAN interfaces, however, researchers are still finding new solutions to improve the overall performance of the CAN interface module. We also developed our CAN-module as an IP block for a variety of reasons of which *full compatibility* with our target processor (COFFEE) is one of the most prominent ones. The main reason is that the COFFEE processor is made by our research group, hence, we know how the core behaves in different situations. We will further investigate the compatibilities later on. Nevertheless, designing a fully compatible CAN interface is one of the main interests. In addition to that, integrating the IP-based CAN module is not only limited to the COFFEE core. It is integrable with any other standard RISC processor, as well. Another advantage of the CAN module is that it can be (loosely) coupled with the COFFEE core via a dedicated co-processor slot. Therefore, there will be no additional overhead on the memory-mapped data bus since the co-processors are connected to the core via the co-processor bus. Henceforth, we refer to the CAN module by using the term *"CAN co-processor"*.

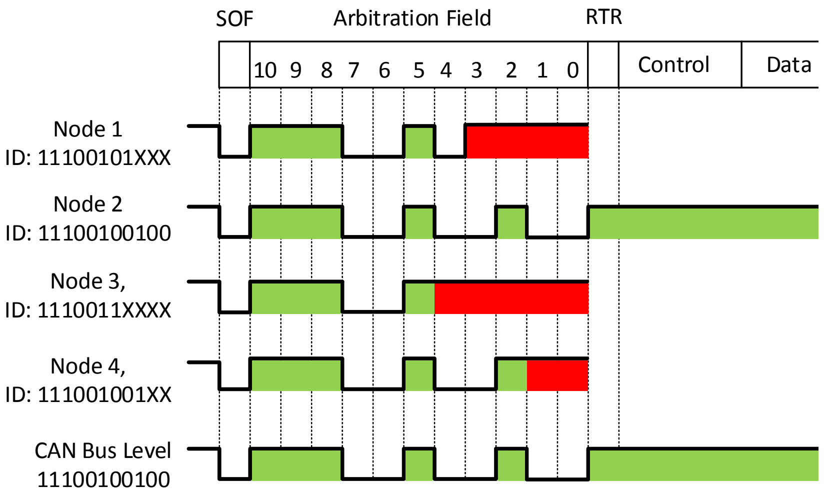


Figure 4.3: How the CAN nodes are prioritized to access the media.

4.5.1 Arbitration

In a CAN system, each node can simultaneously transmit data while listening to the media. Figure 4.3 presents how each node accesses the shared media based on the prioritized arbitration field. The identifications of the nodes are assigned by the system designer. The highest priority is defined in the LSB bits of the arbitration field. It means that a node with a dominant bit in the LSBs has the priority over the other ones with recessive bits. For example, consider the ID of the Node 2 "11100100100", alongside the Node 1 "11100101XXX" in the same figure. Since Node 2 transfers a dominant bit in the fourth LSB (ID[3]), it overrides Node 1 whose ID[3] is a recessive bit. Accordingly, Node 1 learns that another node with higher priority is transmitting at the same time, hence, it immediately stops its transmission. Henceforth, nodes with lower priorities, as well as the silent ones are the receivers of the message. The CAN bus is always granted to a node with the highest priority message.

4.5.2 Data Management

The COFFEE processor exploits four co-processors which share 64 numbers of 32-bit registers (4×16). In addition, we studied that the length of a CAN message varies between 0 to 8 bytes. In this implementation, the co-processor always provides the core with 8-byte data (64 bits), irrespective to the data length. Hence, the processor should refer to the co-processor in two consecutive attempts to obtain the entire data. On the other hand, there is a possibility that the processor is interrupted between the first and second attempts to the corresponding co-processor. Therefore, the co-processor is designed to enter the standby mode to prevent register overwriting in case the core is interrupted. Hence, the second reference from the processor to the co-processor is returned with the appropriate portion of the data. Data coherency is guaranteed by the co-processor as follows.

When the processor refers to the co-processor to fetch the data, it should read two specific registers from the co-processor shared register bank. Hence, the processor keeps the signal *cop_rd* set for two consecutive clock cycles. In case the processor is exactly interrupted at

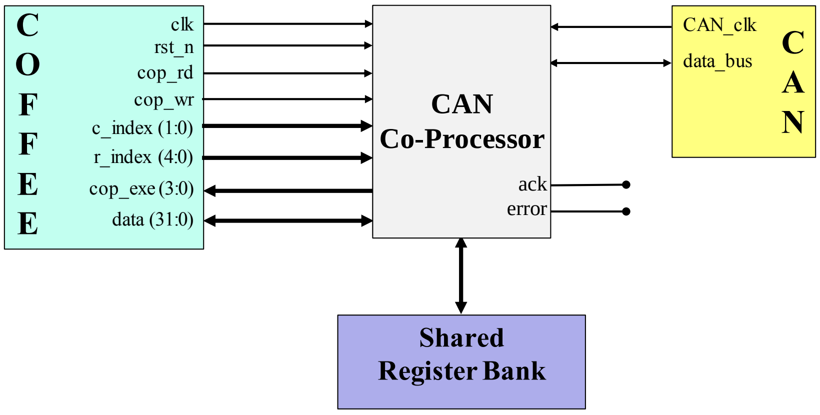


Figure 4.4: The integration of the co-processor with the COFFEE core, as well as the CAN system [P. VI] © IEEE, 2016.

the second reference, the processor will clear *cop_rd*. Instantly, the co-processor is alerted about the cause and it changes the state to standby mode. In this mode, any permission to write to the destination registers in the register bank is denied by the co-processor. Instead, the co-processor starts buffering the incoming packet. Once the internal buffer of the co-processor is about to overflow, the co-processor immediately interrupts the core in case the processor has not referenced the co-processor yet.

4.5.3 Error Handling

There are four types of errors that can occur in a CAN-based transmission. The co-processor is designed to detect and broadcast to all nodes when one of the following fault containments occurs.

- *bit error*: happens when the transmitted bit differs from what the receiving node expects, e.g. the data length is more than 8 bytes.
- *form error*: When the general format of the frame is altered (e.g. wrong logic value in delimiters).
- *bit stuffing error*: occurs when 6 consecutive bits of the same type have been found by the receiver (excluding the EOF).
- *Acknowledge error*: happens when the transmitter never receives an acknowledge from the receiver.
- *CRC error*: is detected by the receiver when the computed value in the CRC field is different from the received CRC.

The transmitter is able to detect the bit error, along with the acknowledge error, whereas the receiver detects the form error, bit stuffing error, and CRC error. As soon as an error is found by a node, an error frame is broadcast to the network, thus, all other nodes will be notified about the cause.

4.6 The FPGA Implementation and Integration

Despite the MMU, the CAN co-processor is simply integrated with the COFFEE processor without introducing any additional complexity. In addition, the core provides all the necessary signals to read/write from/to the co-processor. Figure 4.4 depicts how the co-processor is integrated with the COFFEE core, as well as to the CAN system. The co-processor operates with two different clock domains, one is driven from the core (clk) and the other one is globally driven from the CAN system (CAN_clk). The processor can read/write from/to the co-processor using the co-processor *data* bus. The processor communicates with the co-processor by using *cop_rd* and *cop_wr* signals. From the instruction set point of view, the communication between the processor and co-processors is performed with *movtc* and *movfc* instructions. Moreover, the processor refers to the shared co-processor registers by using signal *r_index*. The 2-bit *c_index* selects one of the four available co-processors. In this implementation, we employed the fourth co-processor slot to integrate the CAN module with the COFFEE processor. Hence, the logical value "11" on the *c_index* signal refers to the fourth co-processor slot. Furthermore, the co-processor interrupts the core via the *cop_exe* interface, e.g. in case the internal buffer is overflowing. On the other side of the co-processor entity, the CAN module can communicate with the rest of the nodes via the shared CAN bus. Furthermore, we extracted the *ack*, along side the *error* signal for the future use.

4.7 Synthesis Results

Similar to the MMU and NC-OFDM synchronizer, the same setting and environments are used to analyze the hardware implementation costs of the CAN co-processor. Table 4.1 provides the maximum achieved operating frequency of the co-processor in two different temperatures when the design is running on either slow or fast mode. As studied in previous chapter, the most practical environment is the slow model at 85 °C. The co-processor has the potential to operate at about 490MHz, while this figure is changed to 825MHz in the fast mode at the lower temperature. However, the circuit is restricted to the maximum operating frequency of 713MHz due to the limited toggling rate in the clock tree. It means that the FPGA board cannot offer more operating frequency than 713MHz in this design. Moreover, long interconnections are the main sources of introducing delays to the circuit.

Table 4.2 illustrates the hardware costs introduced by the co-processor, alongside the energy consumption of receiving and decoding one standard data frame. Implementing the co-processor on FPGA adds only 321 ALMs (7.12% additional cost) and 441 (8%) registers more to the system where the COFFEE is running. In terms of energy consumption, the co-processor has a very low energy consumption to receive, decode and extract the data of a standard data frame. It is worth mentioning that the co-processor consumes about 6% of the total energy for dynamic and I/O activities.

4.8 Concluding Remarks

In this chapter, the FPGA implementation of a run-time reconfigurable module capable of communicating over the Controller Area Network (CAN) protocol was discussed. Moreover, the developed module was integrated with the COFFEE RISC core as a loosely-coupled co-processor. The co-processor reconfigures itself during the run-time to operate on different data packets, i.e., standard and extended packets. The co-processor

Table 4.1: The maximum operating frequencies, along with the worst critical path analysis [P. VI] © IEEE, 2016.

			Co-processor
Maximum Frequency	*Slow model*	85 °C	493 (MHz)
		0 °C	523 (MHz)
	Fast model	85 °C	733 (MHz) restricted to 673 (MHz)
		0 °C	825 (MHz) restricted to 713 (MHz)
Worst Critical Path Analysis (ns)	*Cell Delay*		0.825 – 40.54%
	Interconn. Delay		1.210 – 59.46%
	Data Delay		2.035 – 100%
	Clock Skew		-0.019
	Time Slack		0.071

Table 4.2: Summary of the implementation in detail [P. VI] © IEEE, 2016.

	COFFEE	**Co-processor**	**Info**
ALMs	4503	321	7.12% overhead
Total Registers	5439	441	8% overhead
Static Energy	–	327.33 (μJ)	93.82%
Dynamic Energy	–	11.19 (μJ)	3.21%
I/O Energy	–	10.36 (μJ)	2.97%
Total Energy	–	348.88 (μJ)	100%

guarantees the integrity of the data in some exceptional situations such as two non-consecutive references from the processor. In that case, the co-processor changes the state to the standby mode to prevent the data registers being overwritten. Another advantage of this integration is that since the co-processor communicates with the core over the co-processor bus, no additional overhead is introduced to the memory-mapped data bus. Hence, the data bus stays as idle as possible. The synthesis results were presented in terms of hardware costs, maximum operating frequency, as well as the energy consumption. The co-processor introduced 7%-8% overhead to the hardware costs, along with a trivial increase in energy consumption. The main achievement of this implementation is the new ability of the COFFEE core to communicate with CAN-based peripheral devices. The integration increased the versatility of the target processor even in industrial usages.

5 Conclusion

The main contribution of the author of this thesis was to research hardware acceleration and develop different Intellectual Property (IP)-based modules integrable to any standard Reduced Instruction Set Computer (RISC)-based processor. The ultimate goal of this thesis was to improve the versatility of a particular processor named the COFFEE RISC core developed by the same research group at Tampere University of Technology.

The overall structure of this thesis was based on the research on hardware acceleration methodology. The design, implementation, and integration of several IP-blocks which were loosely/tightly-coupled with the COFFEE processor were carried out as the case study. Each IP block improved the versatility of the COFFEE core in embedded applications. In addition, the author made his best effort to provide valuable references for future extensions for the COFFEE processor.

5.1 Main Results

There were several achievements obtained in this research work. On the hardware acceleration methodology and architectures, one achievement was the state-of-the-art in developing a synchronizer block suitable for future wireless communications. The synchronizer was able to reconfigure itself on-the-fly using Partial Reconfiguration (PR) feature. The COFFEE RISC processor is now capable of employing the developed synchronizer to detect, receive and extract the data segment of a wireless packet in a Non-Contiguous Orthogonal Frequency Division Multiplexing (NC-OFDM) system. However, the synchronizer is mainly practical in a complete receiver framework. For example, the COFFEE can exploit the NC-OFDM synchronizer in parallel with other components such as Fast Fourier Transform (FFT) block to act as a receiver for future wireless technology. Such blocks have also been developed by the research group.

The second achievement of this work was the development of a configurable Memory Management Unit (MMU). The MMU enabled the virtual-to-physical address translation mechanism for the COFFEE. Moreover, the MMU employed several Translation Look-aside Buffers (TLBs) in two levels of hierarchies to speed up the address translation mechanism. Prior to this integration, the COFFEE was not able to manage any operating system, however, it is now possible to port an operating system (such as Linux) on the target processor. Porting an operating system is one of the absolute targets in the future.

The third achievement in this work was the development of an IP module as the Controller Area Network (CAN) protocol. The CAN module had the ability to reconfigure itself during the run-time to operate on the different type of the CAN-based data frames. Technically, the CAN module had a very light design with trivial energy consumption.

The CAN protocol is widely known as an in-vehicle protocol. Having integrated the CAN module, the COFFEE processor has become more versatile even for industrial purposes.

5.2 Future Developments

As stated earlier, there are still open research opportunities remaining to be resolved. One of the main contributions of the future research is developing and porting an operating system (such as Linux) for the target processor. This might be slightly challenging since developing an embedded operating system requires both hardware and software skills to understand the overall behavior of the system. Another future extension to the core is to develop the entire receiver architecture on one FPGA. This development is also very challenging since resource allocation is very critical in that system. Due to the complexity of synchronization in NC-OFDM systems, the synchronizer itself requires a lot of hardware resources to operate appropriately. Perhaps the PR feature could be an alternative solution to alleviate this problem.

Bibliography

[1] C. Brunelli, “Design of Hardware Accelerators for Embedded Multimedia Applications,” Ph.D. dissertation, Tampere University of Technology, 2008.

[2] M. Platzner, “Reconfigurable computer architectures,” *e&i Elektrotechnik und Informationstechnik*, vol. 115, no. 3, pp. 143–148, 1998. [Online]. Available: http://dx.doi.org/10.1007/BF03159565

[3] K. Compton and S. Hauck, “Reconfigurable computing: A survey of systems and software,” *ACM Comput. Surv.*, vol. 34, no. 2, pp. 171–210, Jun. 2002. [Online]. Available: http://doi.acm.org/10.1145/508352.508353

[4] G. Estrin, “Organization of computer systems: The fixed plus variable structure computer,” in *Western Joint IRE-AIEE-ACM Computer Conference*, ser. IRE-AIEE-ACM ’60 (Western). New York, NY, USA: ACM, 1960, pp. 33–40. [Online]. Available: http://doi.acm.org/10.1145/1460361.1460365

[5] C. Bobda, *Introduction to Reconfigurable Computing: Architectures, Algorithms, and Applications*, 1st ed. Springer Publishing Company, Incorporated, 2007.

[6] R. Tessier, K. Pocek, and A. DeHon, “Reconfigurable computing architectures,” *Proceedings of the IEEE*, vol. 103, no. 3, pp. 332–354, March 2015.

[7] C. Brunelli, F. Campi, J. Kylliäinen, and J. Nurmi, “A Reconfigurable FPU as IP Component for SoCs,” in *Proceedings International Symposium on System-on-Chip*, Nov 2004, pp. 103–106.

[8] L. Harju, “Programmable Receiver Architectures for Multimode Mobile Terminals,” Ph.D. dissertation, Tampere University of Technology, 2006.

[9] R. Airoldi, “Design and Implementation of Software Defined Radios on a Homogeneous Multi-Processor Architecture,” Ph.D. dissertation, Tampere University of Technology, 2013.

[10] F. Garzia, “From run-Time Reconfigurable Coarse-Grain Arrays to Application-Specific Accelerator Design,” Ph.D. dissertation, Tampere University of Technology, 2009.

[11] J. Kylliäinen, T. Ahonen, and J. Nurmi, “General-Purpose Embedded Processor Cores – The COFFEE RISC Example,” in *Processor Design: System-On-Chip Computing for ASICs and FPGAs*, 1st ed., J. Nurmi, Ed. Springer Publishing Company, 2007, pp. 83–100.

[12] J. Acharya, H. Viswanathan, and S. Venkatesan, “Timing Acquisition for Non Contiguous OFDM Based Dynamic Spectrum Access,” in *3rd IEEE Symposium on New Frontiers in Dynamic Spectrum Access Networks*, Oct 2008, pp. 1−10.

[13] A. F. Molisch, “Cognitive Radio,” in *Wireless Communications*. John Wiley & Sons, 2012, ch. 21, pp. 501−520.

[14] J. Gao, X. Li, W. Wang, and Y. Bai, “Non-Contiguous Channel Bonding for TV White Space Usage With NC-OFDM Transmission,” *Wireless Personal Communications*, vol. 86, no. 2, pp. 385−401, 2016.

[15] Y. Xing, H. Kushwaha, K. P. Subbalakshmi, and R. Chandramouli, “Codes and games for dynamic spectrum access,” in *Cognitive Radio, Software Defined Radio, and Adaptive Wireless Systems*, H. Arslan, Ed. Dordrecht: Springer Netherlands, 2007, ch. 6, pp. 161−187. [Online]. Available: http://dx.doi.org/10.1007/978-1-4020-5542-3_6

[16] Z. Kollar and P. Horvath, “Physical Layer Considerations for Cognitive Radio: Modulation Techniques,” in *IEEE 73rd Vehicular Technology Conference (VTC Spring)*, May 2011, pp. 1−5.

[17] I. Budiarjo, H. Nikookar, and L. P. Ligthart, “Cognitive Radio Modulation Techniques,” *IEEE Signal Processing Magazine*, vol. 25, no. 6, pp. 24−34, November 2008.

[18] S. Feng, H. Zheng, H. Wang, J. Liu, and P. Zhang, “Preamble Design for Non-Contiguous Spectrum Usage in Cognitive Radio Networks,” in *IEEE Wireless Communications and Networking Conference*, April 2009, pp. 1−6.

[19] R. Chávez-Santiago, M. Szydełko, A. Kliks, F. Foukalas, Y. Haddad, K. E. Nolan, M. Y. Kelly, M. T. Masonta, and I. Balasingham, “5G: The Convergence of Wireless Communications,” *Wireless Personal Communications*, vol. 83, no. 3, pp. 1617−1642, 2015.

[20] J. Liu, S. Feng, and H. Wang, “Comb-Type Pilot Aided Channel Estimation in Non-Contiguous OFDM Systems for Cognitive Radio,” in *5th International Conference on Wireless Communications, Networking and Mobile Computing*, Sept 2009, pp. 1−4.

[21] X. Zhou and R. Qiu, “An adaptive ssnchronization algorithm for non-contiguous ofdm cognitive radio systems,” in *IET International Communication Conference on Wireless Mobile and Computing (CCWMC 2011)*.

[22] A. M. Wyglinski, “Effects of Bit Allocation on Non-Contiguous Multicarrier-Based Cognitive Radio Transceivers,” in *IEEE Vehicular Technology Conference*, Sept 2006, pp. 1−5.

[23] R. Rajbanshi, A. M. Wyglinski, and G. J. Minden, “An Efficient Implementation of NC-OFDM Transceivers for Cognitive Radios,” in *1st International Conference on Cognitive Radio Oriented Wireless Networks and Communications*, June 2006, pp. 1−5.

[24] Z. Yuan, S. Pagadarai, and A. M. Wyglinski, “Feasibility of NC-OFDM Transmission in Dynamic Spectrum Access Networks,” in *IEEE Military Communications Conference (MILCOM)*, Oct 2009, pp. 1−5.

[25] D. Qu, J. Ding, T. Jiang, and X. Sun, “Detection of Non-Contiguous OFDM Symbols for Cognitive Radio Systems without Out-of-Band Spectrum Synchronization,” *IEEE Transactions on Wireless Communications*, vol. 10, no. 2, pp. 693–701, February 2011.

[26] J. Y. Won, H. G. Kang, Y. H. Kim, L. Song, and M. S. Song, “Fractional Bandwidth Mode Detection and Synchronization for OFDM-Based Cognitive Radio Systems,” in *IEEE Vehicular Technology Conference (VTC)*, May 2008, pp. 1599–1603.

[27] B. Huang, J. Wang, W. Tang, and S. Li, “An Effective Synchronization Scheme for NC-OFDM Systems in Cognitive Radio Context,” in *IEEE International Conference on Wireless Information Technology and Systems*, Aug 2010, pp. 1–4.

[28] L. Li, D. Qu, T. Jiang, and J. Ding, “Design of LDPC Codes for non-Contiguous OFDM-Based Communication Systems,” in *IEEE International Conference on Communications (ICC)*. IEEE, 2012, pp. 4712–4716.

[29] A. Dutta, D. Saha, D. Grunwald, and D. Sicker, “Practical Implementation of Blind Synchronization in NC-OFDM Based Cognitive Radio Networks,” in *Proceedings of the 2010 ACM workshop on Cognitive radio networks*. ACM, 2010, pp. 1–6.

[30] D. Saha, A. Dutta, D. Grunwald, and D. Sicker, “Blind Synchronization for NC-OFDM—When “Channels” are Conventions, not Mandates,” in *IEEE Symposium on New Frontiers in Dynamic Spectrum Access Networks (DySPAN)*. IEEE, 2011, pp. 552–563.

[31] “Unlicensed Operation in the TV Broadcast Bands,” https://apps.fcc.gov/edocs_public/attachmatch/FCC-08-260A1.pdf, Federal Communications Commission, p. 130, November 2008, Accessed on 10.02.2017.

[32] R. Airoldi and J. Nurmi, “Design of a Matched Filter for Timing Synchronization,” in *Conference on Design and Architectures for Signal and Image Processing*, Oct 2013, pp. 247–251.

[33] N. Khouja, K. Grati, B. L. Gal, and A. Ghazel, “Power consumption estimation models for fir decimation filters in multi-standard receivers,” in *IEEE GCC Conference and Exhibition (GCC)*, Feb 2011, pp. 617–620.

[34] J. Heiskala and J. Terry, “Synchronization,” in *OFDM Wireless LANs: A Theoretical and Practical Guide*. Sams, 2001, ch. 21, pp. 51–90.

[35] E. Perahia and R. Stacey, “PHY Interoperability With 11a/g Legacy OFDM Devices,” in *Next Generation Wireless LANs: 802.11n and 802.11ac*. Cambridge University Press, 2013, ch. 4.

[36] A. Karatsuba and Y. Ofman, “Multiplication of Multidigit Numbers on Automata,” in *Soviet physics doklady*, vol. 7, 1963, p. 595.

[37] J. W. Hartwell, “A Procedure for Implementing the Fast Fourier Transform on Small Computers,” *IBM Journal of Research and Development*, vol. 15, no. 5, pp. 355–363, Sep 1971.

[38] R. Airoldi, F. Campi, and J. Nurmi, “Approximate computing for complexity reduction in timing synchronization,” *EURASIP Journal on Advances in Signal Processing*, vol. 2014, no. 1, p. 155, 2014. [Online]. Available: http://dx.doi.org/10.1186/1687-6180-2014-155

[39] N. Palladino, “Investigating data throughput and partial dynamic reconfiguration in a commodity fpga cluster framework,” Master’s thesis, Rochester Institute of Technology, 2011.

[40] D. McGrath, “Altera to Offer Partial Reconfiguration at 28-nm,” http://www.eetimes.com/document.asp?doc_id=1313649, Febuary 2010, Accessed on 06.02.2017.

[41] Altera, “Advanced Synthesis Cookbook,” https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/manual/stx_cookbook.pdf, pp. 2–13, July 2011, Accessed on 16.02.2017.

[42] Altera, “Stratix V Device Handbook Volume 1: Device Interfaces and Integration,” https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/hb/stratix-v/stx5_core.pdf, pp. 3–16, December 2016, Accessed on 16.02.2017.

[43] Altera, “Stratix II DSP Performance,” https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/hb/stratix-v/stx5_core.pdf, January 2005, Accessed on 17.02.2017.

[44] D. A. Patterson and J. L. Hennessy, “Large and Fast: Exploiting Memory Hierarchy,” in *Computer Organization and Design: the Hardware/Software Interface*. Morgan Kaufmann Publishers, 2009, ch. 5, pp. 492–517.

[45] B. Cohen and R. McGarity, “The Design and Implementation of the MC68851 Paged Memory Management Unit,” *Micro, IEEE*, vol. 6, no. 2, pp. 13–28, 1986.

[46] W. Yongqing and Z. Minxuan, “Fully Memory Based Address Translation in User-Level Network Interface,” in *IEEE 3rd International Conference on Communication Software and Networks (ICCSN)*, 2011, pp. 351–355.

[47] D. Schmidt and N. Wehn, “DRAM Power Management and Energy Consumption: a Critical Assessment,” in *Proceedings of the 22nd Annual Symposium on Integrated Circuits and System Design: Chip on the Dunes*. ACM, 2009, p. 32.

[48] H.-C. Ng, Y.-M. Choi, and H. K.-H. So, “Direct Virtual Memory Access From FPGA for High-Productivity Heterogeneous Computing,” in *International Conference on Field-Programmable Technology (FPT)*. IEEE, 2013, pp. 458–461.

[49] A. Brandon, I. Sourdis, and G. N. Gaydadjiev, “General Purpose Computing with Reconfigurable Acceleration,” in *International Conference on Field Programmable Logic and Applications*, Aug 2010, pp. 588–591.

[50] J. Nurmi, “Introduction,” in *Processor Design: System-On-Chip Computing for ASICs and FPGAs*, 1st ed., J. Nurmi, Ed. Springer Publishing Company, 2007, pp. 1–6.

[51] J. Ball, “Designing Soft-Core Processors for FPGAs,” in *Processor Design: System-On-Chip Computing for ASICs and FPGAs*, 1st ed., J. Nurmi, Ed. Springer Publishing Company, 2007, ch. 11, pp. 229–256.

[52] W. Wijesinghe, M. Jayananda, and D. Sonnadara, "Hardware implementation of random number generators," in *Proceedings of the Technical Sessions*, vol. 22, 2006, pp. 28–38.

[53] K. H. Johansson, M. Törngren, and L. Nielsen, "Vehicle Applications of Controller Area Network," in *Handbook of Networked and Embedded Control Systems*, D. Hristu-Varsakelis and W. S. Levine, Eds. Boston, MA: Birkhäuser Boston, 2005, pp. 741–765. [Online]. Available: http://dx.doi.org/10.1007/0-8176-4404-0_32

[54] R. I. Davis, A. Burns, R. J. Bril, and J. J. Lukkien, "Controller Area Network (CAN) Schedulability Analysis: Refuted, Revisited and Revised," *Real-Time Systems*, vol. 35, no. 3, pp. 239–272, 2007. [Online]. Available: http://dx.doi.org/10.1007/s11241-007-9012-7

[55] M. Khanapurkar, P. Bajaj, and S. D. H. Wandhare, "Design Approach for VHDL and FPGA Implementation of Automotive Black Box Using CAN Protocol," *IJCSNS*, vol. 8, no. 9, p. 214, 2008.

[56] J. Diaz, E. Rodriguez, L. Hurtado, H. Cacique, N. Vazquez, and A. Ramirez, "CAN bus Embedded System for Lighting Network Applications," in *51st Midwest Symposium on Circuits and Systems*, Aug 2008, pp. 531–534.

[57] C. Bayilmis, I. Erturk, C. Ceken, and I. Ozcelik, "A CAN/IEEE 802.11 b Wireless LAN Local Bridge Design," *Computer Standards & Interfaces*, vol. 30, no. 3, pp. 200–212, 2008.

[58] S. Y. Shin and F. P. Rezha, "Extending CAN Protocol With ISA100.11a Wireless Network," in *International Conference on ICT Convergence (ICTC)*. IEEE, 2012, pp. 472–476.

[59] M. Farsi, K. Ratcliff, and M. Barbosa, "An overview of controller area network," *Computing & Control Engineering Journal*, vol. 10, no. 3, pp. 113–120, 1999.

[60] L. Wolfhard, *Can System Engineering: From Theory to Practical Application*. Springer Verlag, New York, NY, 1997.

[61] D. Leu, "FPGA-Based CAN Solutions," http://www.inicore.com/______pdf/fpga_based_can_solutions.pdf, p. 6, 2005, Accessed on 16.03.2017.

[62] J. Kylliäinen, J. Nurmi, and M. Kuulusa, "COFFEE - a Core for Free," in *Proceedings International Symposium on System-on-Chip,*, Nov 2003, pp. 17–22.

[63] T. Ahonen and J. Nurmi, "Hierarchically Heterogeneous Network-on-Chip," in *EUROCON, The International Conference on" Computer as a Tool"*. IEEE, 2007, pp. 2580–2586.

[64] "COFFEE RISC core - a COre For FrEE," http://coffee.tut.fi, Tampere University of Technology, Accessed on 07.02.2017.

[65] "COFFEE Core User Manual," http://coffee.tut.fi/docs/COFFEE_Core_USER_MANUAL.pdf, p. 85, 2007, Accessed on 02.03.2016.

Appendix A

The Architecture of the Platform

Some of the contents of this chapter are extracted from the Publications [P. V] and [P. VI]. The primary purpose of the following discussion is to illustrate an insight to the hardware platform which has been widely used in the rest of this correspondence. In addition, some fundamental background with respect to the importance of reconfigurable computing is presented, as well.

Embedded Processors

The term Embedded Processor refers to a microprocessor suitable to perform specific applications in an embedded system. An embedded processor is usually smaller in size and consumes less power compared to a GPP. Indeed, an embedded processor is dedicated to execute specific tasks in the, e.g., game consoles, washing machines, and mobile phones. Hence, low cost, small size, and low power consumption are some prominent characteristics of an embedded processor [50]. These types of the processor have a variety of architectures since they are specifically designed for the work they are intended to do. However, most of the architectures are based on the Harvard design which employs separate buses to access the instruction and data memories. In terms of Instruction Set Architecture (ISA), RISC, Complex Instruction Set Computer (CISC) and DSP are commonly used in embedded processors. The RISC, CISC, and DSP ISAs are mainly built for the compiler, human, and application specific devices, respectively [11].

COFFEE, a General-Purpose Embedded Processor

The COre For FrEE (COFFEE) processor is one of such embedded processors developed by our group at Tampere University of Technology [62]. The COFFEE processor has a single-core architecture which is a versatile general-purpose embedded processor suitable for embedded systems for telecommunications and multimedia applications. However, the multi-core version of the COFFEE has been developed under the name NineSilica [63], as well. COFFEE has a Harvard RISC-based architecture to fulfills compiler's requirements. The hardware description of the core is based on the VHDL, thus, it is portable on different technologies. Since the COFFEE core is an open-source processor, it is permitted to use, explore and modify the hardware description of the core to fulfill the system requirements.

The software development tools have also been developed at Tampere University of Technology. Software tools are primarily developed for Linux/Unix operating system, however, Microsoft Window-based users have the possibility to exploit the core using Cygwin to have a virtual Unix environment on Windows. All the required materials,

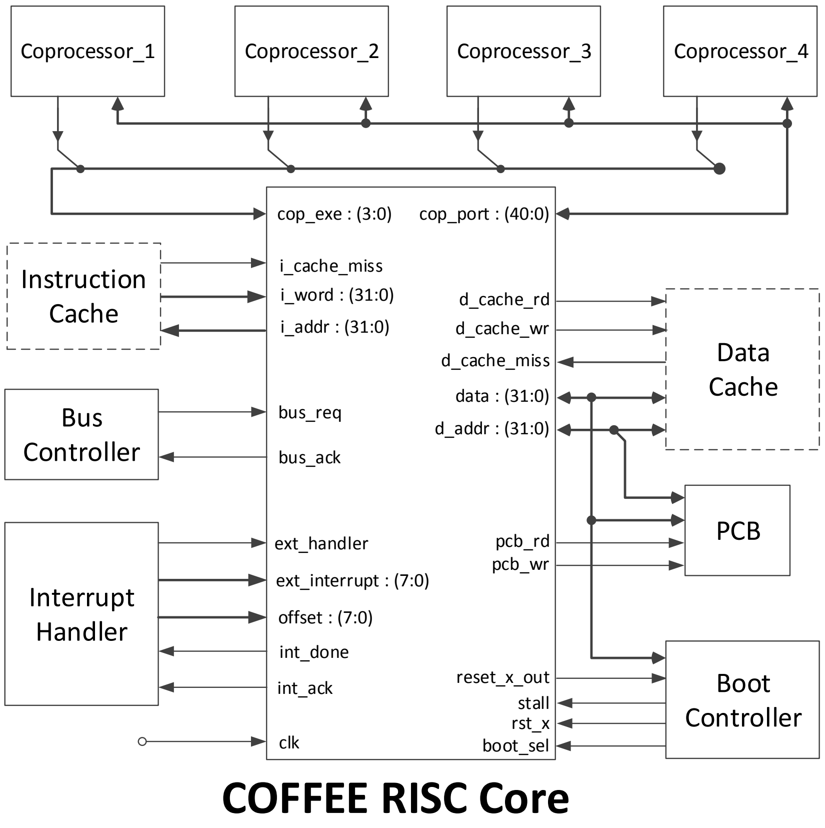


Figure 1: The interface of the COFFEE core [P. VI © IEEE, 2016.]

including the VHDL codes, user guide, compiler, etc. are available on the official website of the COFFEE core in [64].

General Characteristics

Figure 1 presents the interface of the COFFEE RISC processor. The core exploits separate interfaces to refer to data and instruction memories due to its Harvard architectural structure. Besides the scalability and extendibility, reusability, as well as configurability are the two major characteristics of the COFFEE core. The interface of the COFFEE core is explained in the following subsections.

Co-processor

The COFFEE processor exploits four co-processor slots with 64 number of registers in a shared register bank. Each register in the register bank is a 32-bit word. In addition, all the four co-processors are attached to a separate data bus to communicate with the core. Moreover, each co-processor is able to interrupt the core via a dedicated signal. Each co-processor takes advantage of being operated in different clock domains.

Instruction/Data Cache

It is up to the designer to implement caches, as they are not parts of the core. Instead, the core employs scalable memories to suit different applications. In addition, the core supports multi-cycle access to manage large and slow memories.

PCB

Peripheral devices are supposed to be integrated with the processor via the PCB interface. The PCB is directly attached to the memory-mapped data bus. Hence, the COFFEE core accesses the PCB instead of the data memory by asserting *pcb_rd* and *pcb_wr* signals. The data memory is accessible via a dedicated *rd* and *wr* interfaces.

Shared Data Bus

The COFFEE core permits sharing the memory-mapped data bus with peripheral devices. Accordingly, a multiprocessor system with shared data memory can be created. However, connecting a large number of peripheral devices has the potential to decrease the overall performance of the core since the processor will be stalled each time the data bus is being used by other devices.

Boot Control

This component assists the core during the boot-up to appropriately configure itself. The output port *stall* is primarily used in battery-powered systems to save more power. When the core is stalled all the registers are frozen, while only the system clock will remain enabled.

Bus Control

An external device has to send a bus request *(bus_req)* to the core before accessing the main bus. This procedure is done using bus controller/arbitrator. The permission is granted by the core via the *bus_ack* interface.

Interrupt Handler

COFFEE processor is configured with an internal interrupt handler which is applicable for most of the applications. The internal interrupt handler has the possibility to support eight external interrupt resources. Moreover, the COFFEE can exploits four additional interrupt requests of the co-processor slots if those are not being used by any component.

An Insight to the Architecture of the Core

The COFFEE processor can operate in either 16-bit mode or 32-bit instruction mode. COFFEE is able to execute 66 different instructions in a 6-stage pipeline structure. The *Execute* stage is extendable to 3 stages to perform multiplication more efficiently.

The core exploits different registers in two sets, *SET_1* for user applications and *SET_2* for privileged software (such as operating system). The second set, SET_2, is not accessible by the user applications at all, while the operating system can access both sets. Each set contains a total number of 32 registers of both Special Purpose Registers

Table 1: Synthesis results for the most speed-optimized version of the COFFEE core on different FPGA Families [11].

	Stratix II EP2S130F1020C4	**Stratix V** 5SGSMD5K2F40C2	**Virtex-4** XC4VLX160FF1148-11
Registers	5472	5533	5273
Logics	6267	4586	7030
DSP Blocks	2	12	16
Frequency (MHz)	106.25	179.21	69.24
Logic depth	14	17	13
Interconnection Delay (%of cycle)	71.97%	77.33%	71.60%

(SPRs) and General Purpose Registers (GPRs). In addition, COFFEE employs eight Condition Registers (CRs) in order to monitor and control the execution of conditional instructions. Furthermore, the core has another register set, CCB, accessible by the load and store instructions. The CCB register bank offers the software configurability to the core. Finally, the COFFEE processor employs an eight-bit read-only PSR which mainly indicates the current mode of the processor, if the interrupt is enabled, the length of the instruction word, etc. In this thesis, we exploit some of the mentioned registers for integrating IP blocks. Readers are advised to read the user guide of the core in [65] for more information about the infrastructure of the core.

Hardware Implementation Costs and Details

The COFFEE RISC core is a VHDL-based processor which can potentially be implemented on any FPGA target device. In that respect, we synthesized the core on Altera Stratix II & V, as well as the Xilinx Virtex-4 FPGA devices. The used synthesis tools were Quartus II 5.0, Quartus II 15.1, and ISE 7.1i, respectively. In addition, the synthesis results of a

Table 2: Synthesis results of the COFFEE core on 90nm low-power ASIC technology [11].

	COFFEE (small)	**COFFEE (fast)**
Equivalent Gates	60,665	88,074
Actual STD Cells	20,511	28,901
Leakage Power	2.136μW	3.916μW
Number of Nets	22,109	29,915
Average fanout	2.465	2.217
Critical Path Information		
Register Setup	0.513ns	0.251ns
Propagation Time	27.541ns	6.430ns
Clock Frequency	35MHz	150MHz
Logic Depth	46	33

90nm low-power ASIC implementation is also provided for the resource-optimized version of the COFFEE, alongside the speed-optimized one.

Table 1 compares above-mentioned FPGA families together for the most speed-optimized version of the core. Figures in the table show that the Altera FPGAs concentrate more on higher operating frequency, while the Xilinx FPGA focuses more on power consumption [11]. It is worth mentioning that the Stratix II, along with the Virtex-4 are 90nm technology, whereas the Stratix V belongs to the 28nm technology node. Since the Stratix V is a very much newer technology, we only used the COFFEE processor on this FPGA family in the rest of this thesis.

Table 2 presents the synthesis results of implementing the COFFEE core on a 90nm low-power ASIC technology. The table compares the resource-optimized (small) version of the core versus the speed-optimized (fast) version. The figures are reported for the worst-case behavior of the processor, assuming a low supply voltage (0.95V) at high temperature (125℃). However, the fast version of the COFFEE in a more practical condition (1.4V at 25℃) has the potential to operate about three times faster, at an operating frequency around 500MHz.

Publications

Publication I

FPGA Implementation of a Flexible Synchronizer for Cognitive Radio Applications

Farid Shamani, Roberto Airoldi, Tapani Ahonen, Jari Nurmi
Department of Electronics and Communications Engineering
Tampere University of Technology
P.O.Box 553, FIN-33101, Tampere, Finland
{firstname.lastname}@tut.fi

***Abstract*—This paper presents a flexible timing synchronization scheme implemented on an Altera Stratix-V Field Programmable Gate Array (FPGA) device. The core content of the synchronizer is based on a reconfigurable Finite Impulse Response (FIR) filter which performs as a multicorrelator on demand. In concept of flexibility, the synchronizer is able to reconfigure its FIR filter block with Partial Reconfiguration (PR) feature, while the rest of the design is operating. Different synchronization architectures have been evaluated for the design, including MultiplierLess(ML)-based multicorrelator as well as Transposed, Sequential, Parallel and Pipelined-Parallel direct form FIR filters. All the developed architectures are compared to each other in terms of power consumption, chip area, maximum frequency. Synthesis results show that the ML-based multicorrelator achieves 93% better performance in terms of dynamic thermal power dissipation. The ML algorithm also decreases the logic utilization down to 1% of the chip area while the MAC-based architectures utilize almost 7% of the device.**

I. Introduction

Advancements in wireless technology have increased the demand for higher data rate access, which cause spectrum scarcity problem to come more into picture day by day [1]. One solution to cope with spectrum scarcity is to use white spaces between frequencies occupied by the primary users. This can be done using Dynamic Spectrum Access (DSA) method. DSA is an opportunistic approach that not only provides available spectrum to the secondary users, but also significantly improves spectrum utilization [2]. Devices capable of employing DSA are known as Cognitive Radio (CR). CR is an alternative solution to mitigate spectrum scarcity problem by enabling reuse of the licensed spectrum for the secondary users without disrupting the operations of incumbent primary users [3]. Non-Contiguous Orthogonal Frequency Division Multiplexing (NC-OFDM)-based CRs are new emerged promising technology capable of using spectrum more efficiently by continuously sensing the spectrum, employing white spaces for secondary transmission and having the ability to dynamically adopt their radio parameters [4]. An NC-OFDM-based CR is capable of deactivating unused subcarriers during the transmission. If the primary user does not use the entire spectrum, the secondary users could occupy those white spaces as long as they are not interfering with primary subbands. In a similar manner, when an offline primary user comes online, the secondary users must terminate their transmissions in frequency bands which might cause any interference with the primary user. Although CRs can mitigate spectrum scarcity to some extent, a variety of challenges have emerged where synchronization is the most prominent one [5]. The receiver has no information about frequency bands occupied by the secondary user due to the change in primary user location in the spectrum. Furthermore, according to primary user activities, the location of preamble, pilot and data carriers are varied over time. The issues mentioned above enforce the receiver to be always compromised about the existence of secondary users.

In order to establish the spectrum synchronization between receiver and transmitter in NC-OFDM systems, a few solutions have been proposed in [6]− [10]. These methods are considered for an Out-Of-Band (OOB) channel condition. In OOB systems, particular channels are dedicated to secondary transceivers where new information about the spectrum occupancies in terms of active and deactivated subchannels are transferred. Therefore, the receiver is always alert about secondary users activities with the payoff of having a special-purposed channel assigned to secondary transmitters meaning additional cost as well as wasting of bandwidth. Hence, OOB channels are not optimal candidates in some practical situations [13]. Irrespective of OOB transmission, up to the best knowledge of the authors of this paper, there are only a couple of researches in concept of in-band synchronization. The prerequisite synchronization data, for an in-band system, are embedded into the incoming signal itself.

In [11], a fractional bandwidth model has been proposed. Apart from OFDM based synchronization scheme, a specifically designed pseudo-noise (PN) sequence is generated in frequency domain. In time domain, preamble is represented with two identical halves whose sign bits are varied, while the interferences caused by the primary user have not been considered. Theoretically, this algorithm is based on the assumption that the power strength of the primary user signal is lower than secondary transmitter while in reality the secondary transmitter always keeps its transmission power lower than the primary user to mitigate interfering with primary band due to the sidelobe leakages [12].

In [13], the interference caused by primary user has been considered. Therefore, an A Posterior Probability (APP) algorithm is used to distinguish which subchannels are active. When an active subchannel is detected, a Hard Decision-based Detection (HDD) scheme is performed to detect NC-OFDM symbols. However, in subchannels far from primary user band the HDD performs properly, for those subchannels close to primary user band the performance is degraded drastically. Since HDD performance is degraded in noisy environments, another algorithm named Soft Decision-based

Detection (SDD) is performed to increase accuracy on those subchannels close to the primary user band. Therefore, the performance of the proposed method is strongly dependent on primary user activities.

According to [5], the system code rate of the algorithm proposed in [13] is 1/4 while only half of the subcarriers are active. Therefore, the authors try to improve the performance by employing an APP algorithm to identify active subcarriers and, then, use a Low-Density Parity-Check (LDPC) to improve system code rate. However, the authors emphasized more on how to create LDPC code, they did not explain the synchronization process. Furthermore, the proposed method requires additional hardware to decode received LDPC with iteration bound of maximum 80 times meaning more power consumption as well as increasing chip area.

Saha et al in [14] proposed a blind synchronization method where the receiver is able to regenerate time-domain preamble locally at the receiver by employing frequency-domain representation of the preamble. In this article, instead of performing a full 16-bit Multiply-Accumulate (MAC) operations in frequency-domain, the idea of using multiplier-less correlator has been investigated. The authors considered primary user activities and, hence, a binary mask extracts secondary transmission from incoming signal. As a result, primary user existence is eliminated after Fast Fourier Transform (FFT) is done. Thereafter, the rest of the synchronization process is similar to the one of OFDM.

In this paper, a flexible timing synchronizer for NC-OFDM-based CR is implemented on an Altera Stratix-V series (5SGSMD5K2F40C2N) FPGA board. The core content of the synchronizer consists of a multicorrelator which is able to perform both autocorrelation and crosscorrelation functions on demand. The synchronizer core is implemented using different architectures as a core. Therefore, in various situations different implementations can be considered. Moreover, with PR feature, synchronizer is able to reconfigure some portion of itself while the rest of the design is operating. In this paper, a novel implementation technique by availing the PR feature of Altera FPGA is proposed in an efficient manner, while a comparison study of employing various algorithms are considered as well. Therefore, a multi-standards synchronization scheme is presented on FPGA.

The rest of this paper is organized as follows. In Section II synchronization regarding to NC-OFDM-based CR systems are described. Improved synchronization algorithm is discussed in Section III. In Section IV implementation of the proposed synchronization scheme on FPGA is investigated in detail. Synthesis results with respect to different implementations are presented in Section V, followed by conclusions in Section VI.

II. NC-OFDM Synchronization

In all digital communication systems, synchronization is an essential mechanism in order to fetch useful data from the received signal. Synchronization is the process in which the receiver firstly detects any incoming data from the received signal and secondly distinguishes both the beginning and the end of the received packet. So far, designing a robust and accurate synchronization algorithm for NC-OFDM systems has been one of the major challenges for design engineers.

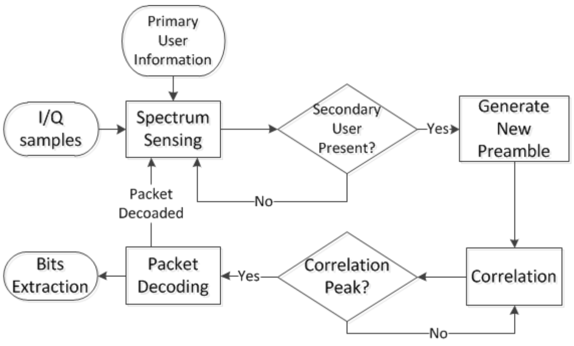


Fig. 1. NC-OFDM synchronization scheme

Synchronization in Orthogonal Frequency Division Multiplexing (OFDM) systems can be done either in time-domain, frequency-domain or both. In reality, the reason why most of the OFDM systems prefer to perform synchronization in time-domain is lack of time due to the packet-based system nature. Each packet starts with a known sequence named *preamble* which facilitates the synchronization process due to its repetitive nature [15]. Symbol timing is extracted using a correlator whose coefficients are exactly the same samples of the preamble in time-domain representation. What makes the NC-OFDM receiver fail at this stage is an alteration in time-domain representation of the predefined preamble due to the *non-continuity* of the encoded signal. In other words, any activity by the primary user enforces the transmitter to alter its transmitting frequencies which results in a change in time-domain waveform representation of the preamble. Generally, NC-OFDM receivers are encountered with two major challenges. Firstly, only a part of subcarriers are available and the rest are occupied by licensed user which makes many traditional synchronization techniques which are used for OFDM systems to be disabled. Secondly, secondary users must reduce their transmission power to mitigate interference with the licensed user due to the sidelobe leakages which makes secondary transmission strength to be weakened [12]. Therefore, CR receiver should be able to adopt its radio parameters with new environment conditions to re-establish the synchronization process in lower Signal-to-Noise Ratio (SNR) regions as well as subcarriers deficiency.

According to [14], Fig. 1 shows the synchronization pipeline for NC-OFDM-based CRs. As it is shown, synchronization is done in two major steps: Subcarrier Detection and Preamble Regeneration and Correlation. Since the CR receiver has no information about subcarriers employed by secondary transmitter, all subcarriers must be gathered at the receiving signal. Then, the whole spectrum is sensed using one of spectrum sensing methods. Next, the results are compared to a threshold to decide which subcarriers contain information. The spectrum sensing procedure is performed in a pipeline structure. The pipeline process will be stalled with any detection of an incoming packet. According to [16], secondary users willing to transmit over the licensed spectrum might have useful knowledge about the signal structure, power and location of the primary user. Therefore, it can be assumed that the secondary

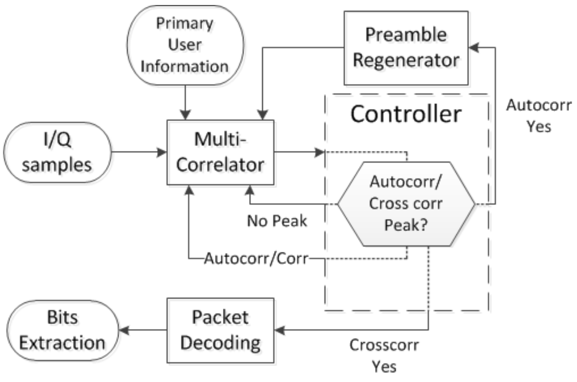


Fig. 2. Improved NC-OFDM synchronization scheme

receiver is eligible to filter out the primary user by having fundamental information about licensed spectrum. Following by the secondary transmission detection, time-domain coefficients of the correlator can be generated at the receiver using frequency-domain representation of the preamble. This is usually done using a low-cost Inverse Fast Fourier Transform (IFFT) unit. Once the time-domain correlator is initialized by new generated coefficients, the incoming waveform can be implied similar to OFDM and is synchronized using related synchronization techniques. However, a copy of the received signal must be buffered in advance to extract time samples from the beginning of the packet. Spectrum sensing unit shall be turned off to save energy as long as the correlation shows satisfactory results. If the correlator fails to detect secondary user's incoming packet, spectrum sensing unit must resume sensing the entire spectrum to provide preamble re-generator with the new secondary user information.

III. Improved Synchronization Algorithm

In this paper, a reconfigurable multicorrelator is proposed to perform both sensing the spectrum along with detecting the preamble. In order to speed up the system, several multicorrelator are used in a parallel manner. As Fig. 2 shows, a multicorrelator is employed in order to perform both sensing the spectrum based on signal energy detection as well as correlating time-domain regenerated preamble with the original signal. Multicorrelator performs autocorrelation between incoming signal and a delayed version of itself as long as the particular subcarriers are inferred as inactive. The autocorrelation results are compared to a threshold in parallel structure. As soon as a result exceeds the threshold, an active subcarrier is inferred to be detected and the controller will immediately be informed. Henceforth, other multicorrelators are discarded and the preamble regenerator unit regenerates the time-domain representation version of the frequency-domain preamble. Then, the buffered version of the In-Phase/Quadrature (I/Q) signals are fed to the active multicorrelator as the input signals and, similarly, the time-domain representation of the preamble (which is created by the re-generator) are fed to the coefficients of the multicorrelator. Thereafter, the controller issues the correlation command to the multicorrelator to compute the maximum similarity between the noisy incoming signal and

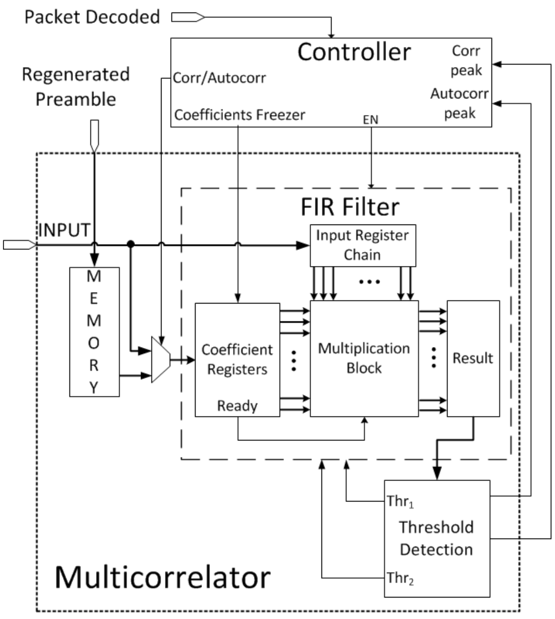


Fig. 3. The internal integration between the controller block and the multicorrelator

the clean version of the time-domain preamble. The obtained results are compared with another threshold to set the second peak, meaning that the preamble boundaries are found at the received signal. Thereafter, the entire packet is delivered to the packet decoding unit while the multicorrelator is stalled in order to save energy. As soon as the packet is decoded, the controller is informed and makes the proper action.

IV. FPGA Implementation of the Multicorrelator

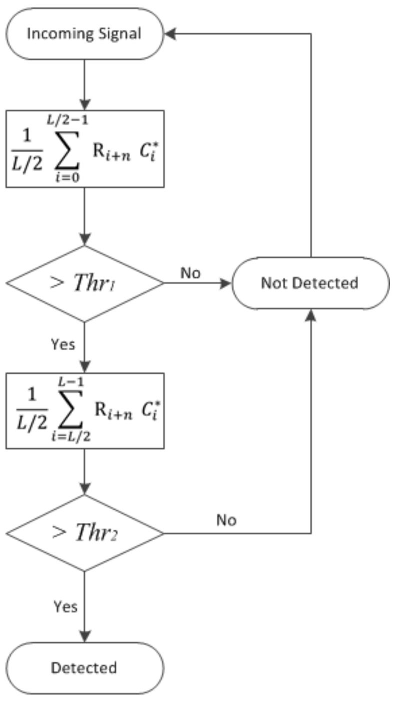
Synchronization in NC-OFDM is done in frequency domain by performing a crosscorrelation between predefined coefficients and received signal, followed by an autocorrelation between the received sequence and a delayed version of itself. Fig. 3 shows an inner overview of the synchronization block structure. Synchronization block is composed of several sub-blocks integrated together to form the entire design in which the major ones are described in following.

A. Memory Block

The memory block considered for this design is an SRAM initialized by frequency-domain representation of the preamble. Thus, the values stored at the memory are used for performing crosscorrelation. The SRAM has a 32-bit data width output port. It is assumed that the 16 Least Significant Bits (LSB) half includes the *Real* part of the preamble and, consecutively, the 16 Most Significant Bits (MSB) half contains the *Imaginary* part of the preamble. The memory block is able to store new values of the re-generated preamble as well.

B. Threshold Detection Block

There is one input and two output ports for *Threshold Detection* block. The input to this unit is a 16-bit signal includes the final result calculated by FIR filter. The decision is made based on a comparison between the signal energy with a 2-step predefined threshold level proposed in [17]. As Fig. 4 depicts, there are two threshold levels considered for the threshold block. The preliminary threshold Thr_1 and the original threshold Thr_2. Conceptually, the correlation of the first half of the incoming signal is calculated and compared with the value Thr_1. If the comparison exceeds the Thr_1, then, the second half will be calculated and compared with Thr_2. Otherwise, the computation is considered as undetected. As soon as the magnitude of the incoming signal energy exceeds Thr_2, a peak is inferred to be found and the threshold detection block informs the controller immediately to perform the proper action. Thereafter, the threshold detection block is looking for the second peak in a similar way which indicates a peak in crosscorrelation of the buffered version of the noisy signal with predefined/regenerated preamble. There are two output ports to the controller where the first one detects autocorrelation peak and the second one indicates the crosscorrelation peak, respectively. The choice of correct value for both Thr_1, Thr_2 have a impressive impact on proper detection in low SNR regions. Although a high value will improve the algorithm robustness, on the other hand, it might result in missing low power frames. On the opposite side, a low value might result in false detection of the frame and noise due to the lower SNR. As an instance, a good approximation for Thr_1 value could be 45% of Thr_2 for 2-step algorithm [17].



Incoming Signal
$\frac{1}{L/2}\sum_{i=0}^{L/2-1} R_{i+n} C_i^*$
$> Thr_1$
No
Not Detected
Yes
$\frac{1}{L/2}\sum_{i=L/2}^{L-1} R_{i+n} C_i^*$
$> Thr_2$
No
Yes
Detected

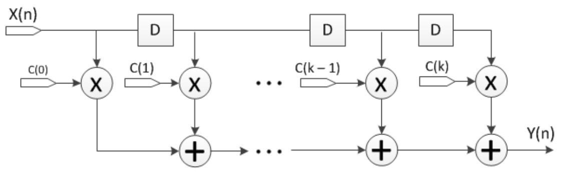
Fig. 4. 2-step threshold detection flowchart. [17]

C. Controller Block

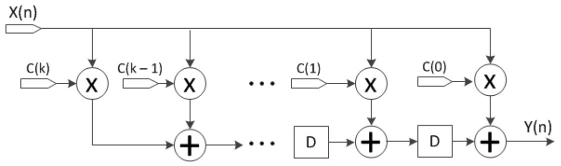
The *Controller* block is the most intelligent block in the design. One of the important responsibilities of this block is to make proper action based on received information by continuously monitoring the behavior of other blocks. At the start of the transmission, the synchronizer should perform auto-correlation between the incoming signal with a delayed version of itself. Therefore, the controller issues the autocorrelation command to the FIR filter and routes the input signal to the coefficients of the FIR filter with the delay length of D. It can be done using a multiplexer with two inputs where the first one is the incoming signal and the second one is the signal coming from memory. The controller sets the multiplexer to logic '0' and '1' in order to perform autocorrelation and crosscorrelation, respectively. As soon as the coefficients of the FIR filter are fed by the incoming signal, the controller issues the stall command to the FIR filter to freeze the input to the coefficients in order to avoid coefficients overloading. Thereafter, the controller waits for any response from *Threshold Detection* block. As soon as the autocorrelation peak is found, the controller will get alerted by its corresponding input. Then, the controller immediately changes the functionality of the multicorrelator to perform crosscorrelation between a buffered version of the incoming signal and the regenerated time-domain preamble. Crosscorrelation is performed in a similar way to autocorrelation. This time, the coefficients of the FIR filter are filled by the signal coming from memory, instead of being filled with the incoming signal. Then, the controller issues the read command to the memory block by issuing the corresponding address. Hence, the proper coefficient is loaded from the memory and is fed to the multicorrelator per clock cycle. When the multicorrelator is filled with the proper coefficients, the controller freezes the coefficient input to the multicorrelator and orders the FIR filter to perform proper action. Again, the controller waits for any alteration in its crosscorrelation input. Whenever the second peak is found, the controller disables the FIR filter and, consequently, any incoming packet will be discarded afterwards. The current packet is handed to the *Packet Decoding* unit to extract useful data. However, the IFFT and *Packet Decoding* units are postponed to the future work. When the data is decoded, the corresponding unit informs the controller to restart the synchronization process. Furthermore, the controller block is capable of reconfiguring the number of taps of the FIR filter at run time in terms of reconfigurability. This assists the FIR filter to reconfigure its taps based on the information provided by the controller.

D. FIR Filter Block

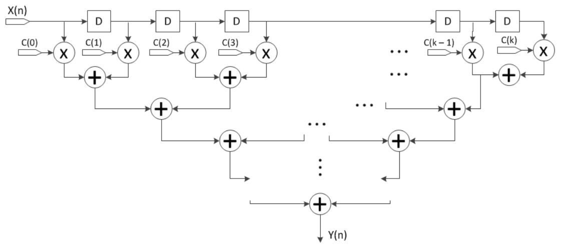
In concept of CR technology, reconfigurability is referred to FIR filters due to the basic idea of the CR which is how to enable a single hardware platform to support multi-standards communication on a single chip by replacing analog signal processing with digital signal processing. Due to the nature of NC-OFDM-based CR, highest order of the filter, for example 4096-tap FIR filter, is required for performing synchronization. Therefore, power consumption of the receiver is increased along with the chip area [18]. According to Equation (1), a FIR filter periodically calculates $y(n)$ as the complex products



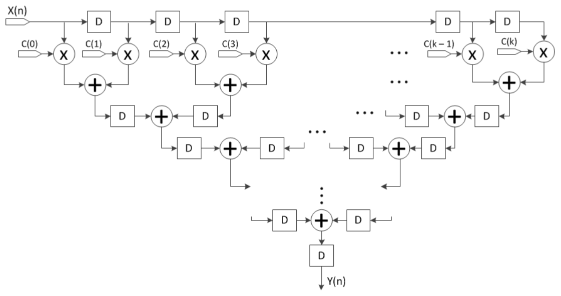
(a)



(b)



(c)



(d)

Fig. 5. Different FIR filter architectures, (a) SDF architecture, (b) TDF architecture, (c) PDF architecture and (d) PPDF architecture

of conjugated coefficients $c(k)$ with incoming signal $x(n-k)$ on a window whose length is N, which is also known as the number of filter taps or filter order. This operation is called MAC operation [19]. Conjugation is done by inversing the sign bit of the coefficients to the FIR filter.

$$y(n) = c(n) * x(n) = \sum_{k=0}^{N-1} c(k)x(n-k) \tag{1}$$

1) MAC-Based Multicorrelator: Basically, FIR filters are composed of three fundamental functional units. These three major parts are adders, multipliers and delay elements integrated and arranged in different ways to form different FIR architectures. Mostly, N-tap FIR filter consists of N delay elements, N multipliers and N-1 adders. Generally, there are two basic architectures for implementing FIR filters known as Sequential Direct Form (SDF) and Transposed Direct Form (TDF), each of which is able to obtain $y(n)$ [20]. In this paper, similar structures such as Parallel Direct Form (PDF) and Pipelined-Parallel Direct Form (PPDF) are investigated and implemented as well. Fig. 5 illustrates different architectures for FIR filter. As it is shown, in TDF architecture all coefficients $C_{(k)}$, including $(C_0, C_1, \ldots, C_k)$, must be multiplied with the input signal $X(n)$ simultaneously at time interval n while in other architectures a particular coefficient must be multiplied with the corresponding input samples. Therefore, excluding TDF architecture, an input chain registers as well as coefficient registers are required. Both coefficient and incoming signal inputs to the FIR filter are 32-bit signal vectors where we stipulated that the first part (including 16 LSB bits) is the Real part and, subsequently, the remaining 16 bits (MSB half) contains the Imaginary part of the corresponding input. Note that the multiplication block performs a complex multiplication which is more complex and power consuming in comparison with the simple one from the hardware point of view. In this work, we used Fixed-width truncation in which only 16 MSBs of 32-bit product are used as the output. FIR filters are intensive parts of a CR due to the massive workload of complex computations as well as operating at high speed to achieve highest sampling rate from power consumption point of view. In section V, resource allocation and power consumptions for different FIR filter architectures are described in detail.

2) ML-Based Multicorrelator: In a ML-based multicorrelator, the size of the correlator as well as the number of registers is depending on the number of input samples. According to [21], instead of performing 16-bit multiplication, a sign bit correlation between coefficients and input signal is done which can be simplified by either XNOR or XOR gate. Similarly, in [22] coefficients are represented in the form of summed powers of 2. In both cases, a massive number of registers and multiplications, which are the most power-hungry operations in FIR filters due to their dynamic power consumption, are mitigated. Although the multicorrelator uses only 1-bit instead of 16-bit, simulation results given by ModelSim software show that the performance of the FIR filter is still at a satisfactory level for this application. Authors in [14] acknowledged that the sign based correlator has satisfactory result in low SNR regions. Intuitively, it works better in higher SNRs regions. By implementing both multiplier-based correlator and sign-based correlator in MATLAB and testing both with 802.11g packets, the results were satisfactory. Based on that, in this work, a ML based multicorrelator implemented in hardware was fed by a random input. The results are given in terms of implementation summary, maximum frequency and power consumption in section V.

E. Partial Reconfiguration (PR)

PR is the ability in which a portion of FPGA is reconfigured while the rest of the design is operating. In other words, a particular region of the design might have multiple personas while everything outside of this region is normally operating. PR has several applications and, generally, is used in designs required to operate continuously while some particular regions can be reconfigured without disrupting the operating parts [23]. Since in this work only the synchronization block

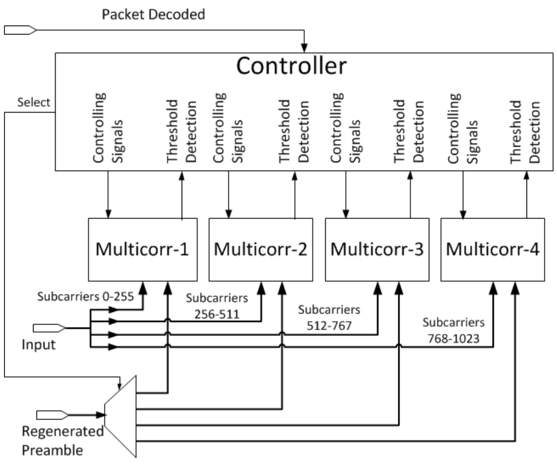


Fig. 6. How the controller is able to control, load and unload the desired multicorrelator

is emphasized, while the rest of the receiver parts are left for the future work, it has been assumed that the PR host is implemented internally. As soon as the PR host sends the PR request to the hard controller block inside the FPGA, when the device is ready to perform PR, the controller acknowledges the host through a "ready" signal. Thereafter, the host transmits the configuration bitstreams, which have been generated by the Quartus-II software beforehand, and waits for acknowledge from FPGA PR controller block. In case of any error occurrence, the controller warns the host. Similarly, if the PR process is performed successfully, the controller informs the host again. With PR feature, different IEEE standards can be easily uploaded to the system while it is operating. Therefore, a single synchronizer is able to reconfigure its radio parameters based on required standard.

Fig. 6 illustrates the integration of the controller with four multicorrelators. The controller controls each multicorrelator via several controlling signals including freezing, enabling, crosscorrelation, autocorrelation signals. The controller consecutively monitors the multicorrelators behavior along with the given threshold results. As soon as the controller finds a desired result, it will keep the corresponding multicorrelator active while discarding other multicorrelators using PR ability. This is done using above-mentioned procedure. The PR host issues the PR request command to discard other multicorrelators. If no error occurs, the FPGA will unload the other multicorrelators and, thereafter, plenty of Adaptive Lookup Modules (ALMs) will be free to be employed for other purposes. If the corresponding multicorrelator fails to determine satisfactory results, PR host requests to load all four multicorrelators to the FPGA again and restart sensing the incoming signal.

V. Synthesis Results

In this section, the proposed algorithm is written in Very-high-speed integrated circuit Hardware Description Language (VHDL), simulated using ModelSim software, compiled and synthesized by Quartus-II version 12.1 environment and implemented on Altera family targeting Stratix-V series 5SGSMD5K2F40C2N FPGA device. Synthesis results for a 256-tap FIR filter are reported in Table I in terms of logic utilization, total DSP blocks, total registers and maximum frequency. The results show that ML architecture is the best architecture in all aspects whereas SDF, as it has been expected, is the worst one. Table I reveals that PPDF utilizes more logic in terms of registers in comparison with other architectures due to its pipeline structure. In ML design, logic utilization is decreased drastically due to the avoidance of a variety of DSP blocks, registers, signals, etc. which preserve more than 82% chip area. The MAC structure FIR filters use 512 DSP blocks (32% of the total) because of multiplication blocks while in the ML-based designs none of the DSP blocks are used. Excluding SDF architecture, the maximum frequency, which is reported after place and route in Quartus-II software, is approximately similar for other architectures. However, the ML design achieves the highest frequency. Power dissipation analysis is reported based on the results given by PowerPlay Power Analyzer Tool in Quartus-II software. The analyzer directly reads the waveforms generated by the ModelSim running at 50 MHz. Static probability and toggle rate for each signal are calculated based on reported VCD file. Table II summarized the estimated results. SDF architecture has the most power dissipation among the other implementations due to its long critical path. As it is shown, core dynamic thermal power is drastically decreased in ML architecture due to avoiding a number of power-hungry multipliers and adders. The ML-based design saves more than 93% of dynamic power consumption in comparison with PDF form (the lowest power consumer architecture in MAC designs). Dynamic thermal power dissipation by hierarchy for various architectures are reported in Table III. As it is obviously clear, FIR filter is the most power-hungry block for MAC-based architecture, while the controller block as well as the threshold block are the least energy consumers. In ML architecture the power dissipation problem related to FIR filter block is solved by using a sign bit correlation instead of 16-bit multiply-accumulate operations. Table IV shows power dissipation for each cell in detail. The results show that ML architecture achieves the best result, almost in all aspects, in comparison with MAC-based architectures. The reason why the SDF has a massive power

TABLE I. Summary of different implementations for the multicorrelator

	TDF	SDF	PDF	PPDF	ML
Logic Utilization (ALMs)	12,483	12,490	12,597	14,611	1,876
Total DSP Blocks	512	512	512	512	0
Total Registers	16,378	16,883	19,764	28,037	2,886
Max. Freq. (MHz)	237.19	67.93	237.64	240.27	257.33

TABLE II. Power consumption analysis (mW)

	TDF	SDF	PDF	PPDF	ML
Total Thermal Power	1436.82	3371.43	1132.33	1138.75	937.13
Core Dynamic Thermal Power	344.17	200.59	153.86	160.10	9.42
Core Static Thermal Power	1070.06	1062.71	955.62	955.80	951.50
I/O Thermal Power	22.58	2108.14	22.86	22.86	26.20

TABLE III. THERMAL POWER DISSIPATION BY HIERARCHY (mW)

	TDF	SDF	PDF	PPDF	ML
Memory Block	0.78	0.81	0.79	0.80	0.79
Threshold Block	0.01	0.01	0.01	0.01	0.01
Controller Block	0.02	0.02	0.02	0.02	0.02
FIR Filter Block	230.20	165.57	125.38	128.80	4.18

TABLE IV. CELL POWER CONSUMPTION (mW)

	TDF	SDF	PDF	PPDF	ML
DSP Block	216.48	130.40	95.20	95.20	0
Combinational Cell	76.58	10.78	1.28	1.30	1.87
Register Cell	2.09	30.31	29.24	32.63	2.45
I/O	25.15	2090.09	4.31	4.23	7.44

dissipation can be explained due to the long critical path. When a signal is flowing from the input, it should traverse a multiplier and 255 adders to the output.

VI. CONCLUSION

In this paper, a flexible timing synchronization scheme for Cognitive Radio application was developed. The proposed synchronizer is capable of performing multicorrelation on demand. The core content of the synchronizer, in terms of architecture, was based on FIR filters. With the same random input signals along with the same coefficient sets, simulation results showed that multicorrelator had the same results irrespective of different architectures. Synthesis results inferred that the best architecture to be employed for FIR filter would be ML-based algorithm in terms of maximum frequency of 257.33 MHz, logic utilization of 1,876 Stratix V ALMs, thermal dynamic power consumption of 9.42 mW, in comparison with MAC-based architectures. As experiences showed, ML-based design has the simplest implementation as well as saving significantly dynamic power (more than 93%) by sacrificing little in terms of accuracy. In MAC-based architectures, the simplest architecture in terms of implementation belonged to TDF and SDF architectures, whereas the most complex ones were PDF and PPDF, respectively. However, there is always a trade-off between employing MAC architectures, excluding SDF, in environments where implementing ML architecture is not possible.

VII. ACKNOWLEDGMENT

This work was funded by the Academy of Finland under contract #258506 (DEFT: Design of a Highly-parallel Heterogeneous MP-SoC Architecture for Future Wireless Technologies). Portions of the work were also funded by CRAFTERS (ConstRaint and Application driven Framework for Tailoring Embedded Real-time Systems) project under contract #295371 in ARTEMIS program.

REFERENCES

[1] Acharya, J.; Viswanathan, H.; Venkatesan, S., "Timing Acquisition for Non Contiguous OFDM Based Dynamic Spectrum Access", 3rd IEEE Symposium on New Frontiers in Dynamic Spectrum Access Networks (DySPAN), pp.1,10, 14-17 Oct. 2008

[2] A. Dutta, D. Saha, D. Grunwald, D. Sicker. "Practical Implementation of Blind Synchronization in NC-OFDM based Cognitive Radio Networks", Proceedings of the 2010 ACM workshop on Cognitive radio networks, pp. 1-6, 2010

[3] Shulan Feng; Zheng, H.; Haiguang Wang; Jinnan Liu; Zhang, P., "Preamble Design for Non-Contiguous Spectrum Usage in Cognitive Radio Networks", IEEE Wireless Communications and Networking Conference (WCNC), pp.1-6, April 2009

[4] Rajbanshi, R.; Wyglinski, Alexander M.; Minden, G.J., "An Efficient Implementation of NC-OFDM Transceivers for Cognitive Radios", 1st International Conference on Cognitive Radio Oriented Wireless Networks and Communications, pp.1,5, 8-10 June 2006

[5] Li Li; Daiming Qu; Tao Jiang; Jie Ding, "Design of LDPC codes for non-contiguous OFDM-based communication systems", IEEE International Conference on Communications (ICC), pp.4712,4716, 10-15 June 2012

[6] Jinnan Liu; Shulan Feng; Haiguang Wang, "Comb-Type Pilot Aided Channel Estimation in Non-Contiguous OFDM Systems for Cognitive Radio", 5th International Conference on Wireless Communications, Networking and Mobile Computing (WiCom), pp.1,4, 24-26 Sept. 2009

[7] Xiao Zhou; Runhe Qiu, "An adaptive synchronization algorithm for Non-Contiguous OFDM cognitive radio systems", IET International Communication Conference on Wireless Mobile and Computing (CCWMC 2011), pp.102,106, 14-16 Nov. 2011

[8] Wyglinski, Alexander M., "Effects of Bit Allocation on Non-Contiguous Multicarrier-Based Cognitive Radio Transceivers", IEEE 64th Vehicular Technology Conference, pp.1,5, 25-28 Sept. 2006

[9] Rajbanshi, R.; Wyglinski, Alexander M.; Minden, G.J., "An Efficient Implementation of NC-OFDM Transceivers for Cognitive Radios", 1st International Conference on Cognitive Radio Oriented Wireless Networks and Communications, pp.1,5, 8-10 June 2006

[10] Zhou Yuan; Pagadarai, S.; Wyglinski, Alexander M., "Feasibility of NC-OFDM transmission in dynamic spectrum access networks", IEEE Military Communications Conference (MILCOM), pp.1,5, 18-21 Oct. 2009

[11] Jae Yeon Won; Hyun Gu Kang; Yun Hee Kim; Iickho Song; Myung-Sun Song, "Fractional Bandwidth Mode Detection and Synchronization for OFDM-Based Cognitive Radio Systems", IEEE Vehicular Technology Conference (VTC), pp.1599,1603, 11-14 May 2008

[12] Biao Huang; Jun Wang; Wanbin Tang; Shaoqian Li, "An effective synchronization scheme for NC-OFDM systems in cognitive radio context", IEEE International Conference on Wireless Information Technology and Systems (ICWITS), pp.1,4, Aug. 28 2010-Sept. 3 2010

[13] Daiming Qu; Jie Ding; Tao Jiang; XiaoJun Sun, "Detection of Non-Contiguous OFDM Symbols for Cognitive Radio Systems without Out-of-Band Spectrum Synchronization", IEEE Transactions on Wireless Communications, vol.10, no.2, pp.693,701, February 2011

[14] Saha, D.; Dutta, A.; Grunwald, D.; Sicker, D., "Blind synchronization for NC-OFDM - When "channels" are conventions, not mandates", IEEE Symposium on New Frontiers in Dynamic Spectrum Access Networks (DySPAN), pp.552,563, 3-6 May 2011

[15] T. Chiueh, P. Tsai, I. Lai, "Baseband Receiver Design for Wireless MIMO-OFDM Communications", 2nd Edition, 2012, Wiley-IEEE Press, 346 p.

[16] "White Space Database Administrators Guide", [WWW], Federal Communications Commission, [Accessed on 29.10.2013], Available at http://www.fcc.gov/encyclopedia/white-space-database-administrators-guide

[17] R. Airoldi; J. Nurmi, "Design of a matched filter for timing synchronization", Conference on Design and Architectures for Signal and Image Processing (DASIP), 2013, pp.247,251, 8-10 Oct. 2013

[18] Mahesh, R.; Vinod, A.P., "New Reconfigurable Architectures for Implementing FIR Filters With Low Complexity", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol.29, no.2, pp.275,288, Feb. 2010

[19] Ghosh, D.; Sharma, Deepak; Aziz, A., "A Novel Low Area and High Performance Programmable FIR Filter Design Using Dynamic Random Access Memory", 48th Midwest Symposium on Circuits and Systems, pp.1477,1480 Vol. 2, 7-10 Aug. 2005

[20] Shen-Fu Hsiao; Jun-Hong Zhang Jian; Ming-Chih Chen, "Low-Cost FIR Filter Designs Based on Faithfully Rounded Truncated Multiple

Constant Multiplication/Accumulation", IEEE Transactions on Circuits and Systems II: Express Briefs,vol.60, no.5, pp.287,291, May 2013

[21] Diaz, I.; Wilhelmsson, L.; Rodrigues, J.; Lofgren, J.; Olsson, T.; Owall, V., "A Sign-Bit Auto-Correlation Architecture For Fractional Frequency Offset Estimation in OFDM", Proceedings of 2010 IEEE International Symposium on Circuits and Systems (ISCAS), pp.3765,3768, May 30 2010-June 2 2010

[22] Pham, T.H.; Fahmy, S.A.; McLoughlin, I.V., "Low-Power Correlation for IEEE 802.16 OFDM Synchronization on FPGA", IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol.21, no.8, pp.1549,1553, Aug. 2013

[23] "Design Planning for Partial Reconfiguration Quartus-II Handbook Version 13.0", Vol. , May 2013, Altera Corporation, 46 p.

Publication II

Journal of Systems Architecture 000 (2016) 1–15

Contents lists available at ScienceDirect

Journal of Systems Architecture

journal homepage: www.elsevier.com/locate/sysarc

FPGA implementation issues of a flexible synchronizer suitable for NC-OFDM-based cognitive radios

Farid Shamani[a,*], Roberto Airoldi[b], Vida Fakour Sevom[a], Tapani Ahonen[a], Jari Nurmi[a]

[a] *Department of Electronics and Communications Engineering, Tampere University of Technology, Tampere, Finland*
[b] *Nokia Solutions and Networks, Espoo, Finland*

A R T I C L E I N F O

Article history:
Received 30 January 2016
Revised 20 October 2016
Accepted 15 November 2016
Available online xxx

Keywords:
NC-OFDM
Synchronization
Partial Reconfiguration
Cognitive Radio
FIR filter
FPGA implementation

A B S T R A C T

This paper presents a flexible timing synchronization scheme alongside the hardware implementation issues on an Altera Stratix-V Field Programmable Gate Array (FPGA) device. The core content of the synchronizer is based on Finite Impulse Response (FIR) filter which operates as a multicorrelator on demand. The term “flexibility” refers to a specific part of the synchronizer where the multicorrelator reconfigures its FIR filter block on-the-fly by employing Partial Reconfiguration (PR) feature. Moreover, different implementations have been evaluated for the multicorrelator, including MultiplierLess (ML) approach (an approximate computing technique only for autocorrelation purpose) along with the Direct From as well as the Transposed, Parallel, and Pipelined-Parallel Direct Form FIR filters. All the developed architectures are compared to each other in terms of power consumption, silicon area, and maximum frequency. Preliminary synthesis results show that the ML approach achieves better performance (including 94% less power dissipation, 75% less logic utilization as well as 67% fewer registers) than other architectures when performing autocorrelation function. Furthermore, the critical path is analyzed and appropriate optimization techniques (such as DSP register packing and intermediate register insertion) are applied to the best candidates of the architectures mentioned. As the best results, 2.83× speed-up, 56.57% less logic utilization along with 38.86% fewer registers are achieved for different architectures. Accordingly, we discover that the parallel form, as well as the pipelined-parallel one, achieve more interesting results than the transposed version in most of the cases.

1. Introduction and motivation

Advancements in wireless technology have increased the demand for higher data rate access which leads to spectrum scarcity problem [1]. The prime spectrum is become scarcer and scarcer each day. One possible solution to cope with spectrum scarcity is to use available white spaces between frequency bands occupied by the primary users. Fig 1 illustrates a real measurement taken in downtown Berkeley which shows that the licensed users occupied most of the spectrum while do not utilize it efficiently [2]. This can be done using Dynamic Spectrum Access (DSA) method. The DSA is an opportunistic approach which not only provides available spectrum to the secondary users, but also significantly improves spectrum utilization [3]. Devices capable of employing DSA technique are known as Cognitive Radio (CR). The CR is an alternative solution to mitigate spectrum scarcity problem by making the licensed spectrum reusable for the secondary users without disrupting the operations of incumbent primary users [4]. Non-Contiguous Orthogonal Frequency Division Multiplexing (NC-OFDM)-based CRs are a newly emerged promising technology capable of using spectrum more efficiently by continuously sensing the spectrum and utilizing white spaces for secondary transmitters. CRs have the ability to dynamically adopt their radio parameters depending on the new characteristics of the supreme spectrum [5]. An NC-OFDM-based CR is capable of deactivating unused subcarriers during the transmission. If the primary user does not use the entire spectrum, the secondary users could occupy those white spaces as long as they are not interfering with primary user’s subbands. In a similar manner, when an offline primary user comes online, the secondary users must terminate their transmissions in frequency bands which might cause any interference with the primary user. Although CRs can mitigate spectrum scarcity to some extent, a variety of challenges have emerged where synchronization is the most prominent one [6]. In brief, the receiver has no prior information of which frequency band the secondary trans-

* Corresponding author.
E-mail addresses: farid.shamani@tut.fi (F. Shamani), roberto.airoldi@nokia.com (R. Airoldi), vida.fakoursevom@tut.fi (V. Fakour Sevom), tapani.ahonen@tut.fi (T. Ahonen), jari.nurmi@tut.fi (J. Nurmi).

http://dx.doi.org/10.1016/j.sysarc.2016.11.006
1383-7621/© 2016 Elsevier B.V. All rights reserved.

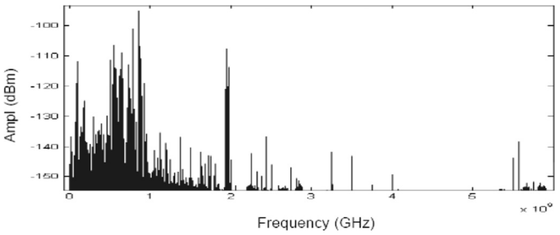


Fig. 1. A snapshot taken at Berkley. The primary user(s) mainly utilizes the first 2GHz of the prime spectrum [2].

mitter has dedicated to itself. Furthermore, based on the primary user's activities, the location of the preamble, pilot and data carriers of the secondary transmitter are varied over the time. The above-mentioned issues compromise the receiver with the existence of secondary users.

This paper is based on our previous work in [7] where a flexible timing synchronizer for NC-OFDM-based CR was implemented on an Altera Stratix-V FPGA board. The synchronizer is based on a reconfigurable multicorrelator which is able to perform both autocorrelation and crosscorrelation functions on-the-fly using Partial Reconfiguration (PR) feature. We take different architectures into account to implement the multicorrelator. Thus, the user has the ability to select the most profitable implementation which is essentially suitable for their applications. In addition, the synchronizer is able to reconfigure some portion of itself while the rest of the design are being operated by taking PR feature into consideration. Moreover, a novel implementation scheme for an NC-OFDM-based CR synchronizer is proposed by exploiting PR feature on an Altera FPGA board. Unlike the other studies in the same field which have more concentrated on system level architecture, we mainly focus on hardware implementation issues, restrictions, and limitations which have the most impact on the overall cost. However, some good references are provided in system level architecture, as well. Eventually, we choose the most suitable candidate that outperforms other architectures in our application. The main contributions of this work are as follows:

- Explaining the State-of-the-art of employing a multicorrelator using PR feature.
- Providing a wide study on hardware implementation costs (including power dissipation, logic utilization, etc.) for each architecture.
- Describing some optimization techniques to improve the performance while minimizing the hardware cost.
- Discussing about different situations where each architecture is applicable and beneficial as well as the situations where particular architecture(s) should be avoided.

The rest of this paper is organized as follows. Section 2 presents some related researches in this field. In Section 3 synchronization in NC-OFDM-based systems are described. The state-of-the-art in synchronization is discussed in Section 4. In Section 5 implementation of the proposed scheme on FPGA is investigated in detail. Section 6 provides a number of constraints in FPGA implementation. Experimental results (before and after optimization) as well as further discussion with respect to the hardware costs for different implementations are presented in Section 7 in detail, followed by conclusion in Section 8.

2. Related works

In principle, in order to establish the spectrum synchronization between receiver and transmitter in NC-OFDM systems, a few solutions have been proposed in [8–12]. These methods are considered for an Out-Of-Band (OOB) channel condition. In OOB systems, particular channels are dedicated to secondary transceivers where new information about the spectrum occupancies with respect to active subchannels along with the deactivated ones are transferred. Therefore, the receiver always knows about the activities of the secondary users with the payoff of having a special-purposed channel. Adopting a secondary channel has the potential to introduce additional cost as well as wasting of bandwidth. Hence, OOB channels are not optimal candidates in some practical situations [13]. In contrary to the OOB transmission, there are only a couple of researches on the concept of in-band synchronization to the best knowledge of the authors of this paper. In an in-band system, the prerequisite synchronization data are embedded into the incoming signal itself.

In [14], a fractional bandwidth model has been proposed. Apart from the OFDM-based synchronization scheme, a specifically designed pseudo-noise (PN) sequence is generated in frequency domain. In time domain, the preamble is represented with two identical halves whose sign bits are varied, while the interferences caused by the primary user have not been considered. Theoretically, this algorithm is based on the assumption that the power level of the primary user signal is lower than secondary transmitter while in reality the secondary transmitter always keeps its transmission power lower than the primary user to mitigate interfering with primary band due to the sidelobe leakages [15].

In [13], the interference caused by the primary user has been considered. Therefore, an A Posterior Probability (APP) algorithm is used to distinguish which subchannels are active. When an active subchannel is detected, a Hard Decision-based Detection (HDD) scheme is performed to detect NC-OFDM symbols. However, the HDD performs properly in subchannels far from the primary user band, but for those subchannels close to the primary user band the performance is degraded drastically. Since the performance of the HDD is degraded in noisy environments, another algorithm named Soft Decision-based Detection (SDD) is performed to increase accuracy on those subchannels close to the primary user band.

According to Li et al. in [6], the system code rate of the algorithm proposed in [13] is 1/4 while only half of the subcarriers are active. Therefore, the authors try to improve the performance by employing an APP algorithm to identify active subcarriers and, then, use a Low-Density Parity-Check (LDPC) to improve system code rate. However, the authors emphasized more on how to create LDPC code, they did not provide any explanation with respect to the synchronization process.

Saha et al.in [16] proposed a blind synchronization method where the receiver is able to regenerate time-domain preamble locally at the receiver by employing a frequency-domain representation of the preamble. In this article, instead of performing a full 16-bit Multiply-Accumulate (MAC) operations in frequency-domain, the idea of using multiplierless correlator has been investigated. The authors considered primary user activities and, hence, a binary mask extracts secondary transmission from incoming signal. As a result, after performing the Fast Fourier Transform (FFT), the primary user is filtered out. Thereafter, the rest of the synchronization process is similar to the one of OFDM.

3. NC-OFDM synchronization

In all digital communication systems, synchronization is an essential mechanism in order to extract useful data from the received signal. Synchronization is the process in which the receiver detects the incoming data and, subsequently, distinguishes the boundaries of the received packet. So far, designing a robust and accurate synchronization algorithm for NC-OFDM systems has been one of the major challenges for design engineers. Synchronization in Orthogo-

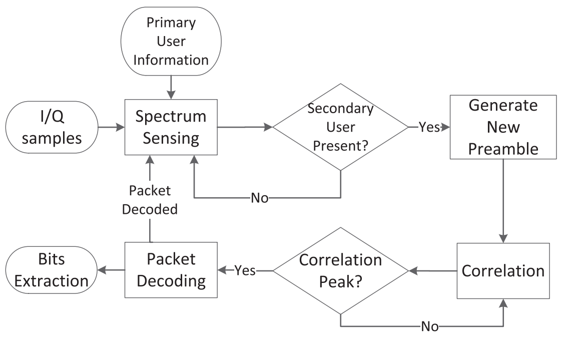


Fig. 2. NC-OFDM synchronization scheme.

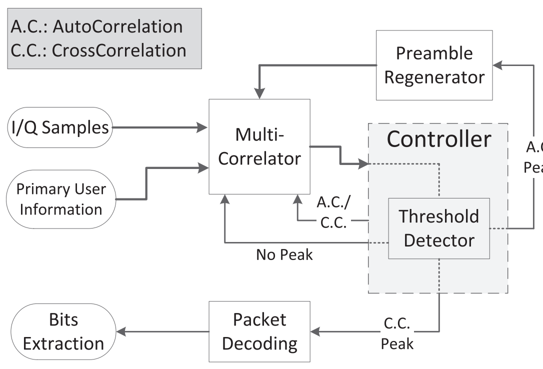


Fig. 3. Proposed NC-OFDM synchronization scheme.

nal Frequency Division Multiplexing (OFDM) systems is done either in time-domain, frequency-domain or both. In reality, the reason why most of the OFDM systems prefer to perform synchronization in time-domain is the lack of time. This is due to the fact that synchronization in frequency-domain is a time-consuming task in the receiver side. Each packet starts with a known sequence named *preamble* which facilitates the synchronization process due to its repetitive nature [17]. Symbol timing is extracted using a correlator whose coefficients are exactly the same samples of the preamble in time-domain representation. What makes the NC-OFDM receiver failed at this stage is the alteration in the time-domain representation of the predefined preamble due to the *non-continuity* of the signal. Indeed, a new activity by the primary user enforces the secondary transmitter to alter its transmitting frequencies which results in a change in the time-domain representation of the preamble waveform. Generally, NC-OFDM receivers are encountered with two major challenges. First, only a part of subcarriers is available for secondary transmission while the remaining subcarriers are occupied by the licensed user. This issue disables many traditional synchronization techniques which are used in OFDM systems. Second, secondary users must reduce their transmission power as low as possible to avoid interference with the primary user due to the sidelobe leakages. In this case, the secondary transmission power strength is weakened which has the potential to be overwhelmed by noise in some low Signal-to-Noise Ratio (SNR) environments [15]. Therefore, a CR receiver should be able to adopt its radio parameters with new environment conditions to re-establish the synchronization process in lower SNR regions as well as subcarriers deficiency.

Fig. 2 shows the synchronization pipeline for NC-OFDM-based CRs [16]. As it is shown, synchronization is performed in two major steps: Subcarrier Detection and Preamble Regeneration and Correlation. Since the CR receiver has no information about subcarriers employed by the secondary transmitter, all subcarriers must be gathered at the receiving signal. Then, the whole spectrum is sensed using one of the spectrum sensing methods. Next, the results are compared to a threshold to decide which subcarriers contain information. The spectrum sensing procedure is performed in a pipeline structure. The pipeline process will be stalled with any detection of an incoming packet. Fortunately, secondary users willing to transmit over the licensed spectrum might have useful knowledge about the signal structure, power and location of the primary user [18]. Therefore, it can be assumed that the secondary receiver is eligible to filter out the primary user by having fundamental information about licensed spectrum. Following by the secondary transmission detection, time-domain coefficients of the correlator can be generated at the receiver using the frequency-domain representation of the preamble. This is usually done using a low-cost Inverse Fast Fourier Transform (IFFT) unit. Once the time-domain correlator is initialized by newly generated coefficients, the incoming waveform can be implied similar to OFDM and is synchronized using related synchronization techniques. However, a copy of the received signal must be buffered in advance to extract time samples from the beginning of the packet. Spectrum sensing unit shall be turned off to save energy as long as the correlation shows satisfactory results. If the correlator fails to detect a secondary user's incoming packet, spectrum sensing unit must resume monitoring the entire spectrum to provide preamble regenerator with the new information with respect to secondary user activities.

4. Proposed synchronization scheme

In this paper, a reconfigurable multicorrelator is proposed to perform both sensing the spectrum along with detecting the preamble. In order to speed up the system, several multicorrelators are used in a parallel manner. As Fig. 3 shows, a multicorrelator is employed in order to perform both sensing the spectrum based on signal energy detection as well as correlating time-domain regenerated preamble with the original signal. Multicorrelator performs autocorrelation between the incoming signal and a delayed version of itself as long as the particular subcarriers are inferred as inactive. The autocorrelation results are compared to a threshold in parallel structure. As soon as a result exceeds the threshold, an active subcarrier is inferred to be detected and the controller will immediately be informed. Henceforth, other multicorrelators are discarded and the preamble regenerator unit regenerates the time-domain representation of the frequency-domain preamble. Then, the buffered versions of the In-Phase/Quadrature (I/Q) signals are fed to the active multicorrelator as the input signals and, similarly, the time-domain representation of the preamble (which is created by the regenerator) are fed to the coefficients of the multicorrelator. Thereafter, the controller issues the correlation command to the multicorrelator to compute the maximum similarity between the noisy incoming signal and the clean version of the time-domain preamble. The obtained results are compared with another threshold to set the second peak, meaning that the preamble boundaries are found in the received signal. Thereafter, the entire packet is delivered to the packet decoding unit while the multicorrelator is stalled in order to save energy. The decoding module will inform the controller as soon as the packet is decoded. Thereafter, the controller takes the action and prepares the multicorrelator to wait for the next packet.

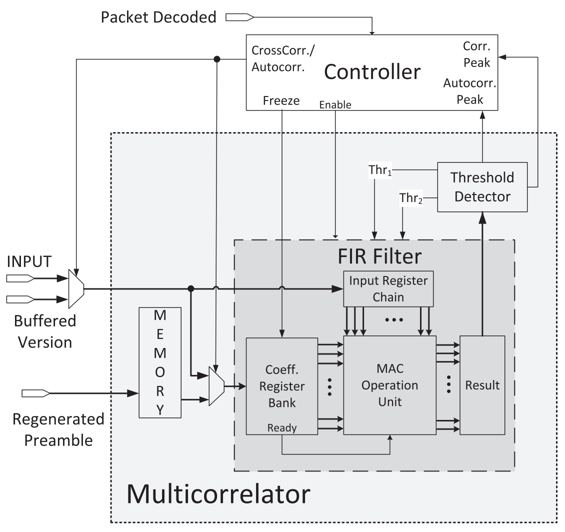


Fig. 4. The internal integration between the controller block and the multicorrelator.

5. FPGA implementation of the multicorrelator

Synchronization in NC-OFDM is established in the frequency-domain by performing a crosscorrelation between predefined coefficients and received signal, followed by an autocorrelation between the received sequence and a delayed version of itself. Fig. 4 shows the infrastructural overview of the synchronization block structure. Synchronization block is composed of several sub-blocks integrated together to form the entire design in which the major ones are described in following.

5.1. Memory block

The memory block considered for this design is an SRAM initialized by the frequency-domain representation of the preamble. Thus, the values stored in the memory are used for performing crosscorrelation. Furthermore, the SRAM is capable of keeping regenerated time-domain preamble, as well. The data output of the SRAM is 32-bit wide where we take into assumption that the 16 Least Significant Bits (LSB) half includes the *Real* part of the preamble and the 16 Most Significant Bits (MSB) half contains the *Imaginary* part. Authors in [19] estimate that using 16-bit inputs provide sufficient accuracy for the receiver. In this case study, we used an unsynthesizable VHDL-based SRAM (for simulation purpose only) which was initiated by the frequency-domain representation of the preamble in hexadecimal format. Since the 32-bit data driven by the SRAM contains both real and imaginary parts of the preamble, the precision of each part of the signal is sacrificed by mitigating the corresponding LSB part of that signal. For example, if we assume that the value 0x3BFCAB36 is the real part of the signal, the SRAM only provides the value 0x3BFC for the synchronizer. The imaginary part of the signal is obtained from the memory in the same manner. We also employed a synthesizable Intellectual Property (IP)-based SRAM provided by Quartus-II design software for synthesis purpose [20].

5.2. Threshold detection block

The main task of the *Threshold Detection* block is to compare computed results calculated by the FIR filter with some threshold. In initial estimation for packet detection stage, the decision is made based on a comparison between the similarity between the incoming signal with a delayed version of itself (autocorrelation). The threshold detector performs a 2-step predefined threshold level proposed in [21] for precise timing acquisition (crosscorrelation). Fig. 5 depicts two threshold levels Thr_1 and Thr_2 used by the threshold detection block when the synchronizer is performing crosscorrelation function. Conceptually, the correlation of the first half of the incoming signal is calculated and compared with the value Thr_1. If the comparison result exceeds the Thr_1, the second half of the correlation function will be executed and subsequently the given result is compared to Thr_2. A crosscorrelation peak is found and reported to the controller, as soon as the similarity of the buffered version of the noisy signal with the clear version of the predefined preamble arises over the Thr_2 value. Otherwise, the overall computation is considered as undetected even though the Thr_1 was met. The controller takes control of the rest of the synchronization when the crosscorrelation peak is received. Therefore, two output ports of the threshold detection block indicate a peak in autocorrelation peak as well as a peak in crosscorrelation, respectively. Choosing an appropriate value for both Thr_1, Thr_2 has an impressive impact on a precise detection even in low SNR regions. Although a high value will improve the algorithm robustness, on the other hand, it might result in missing low-power frames. On the opposite side, a low value might result in false detection of the frame with the noise in lower SNR regions. For instance, a good

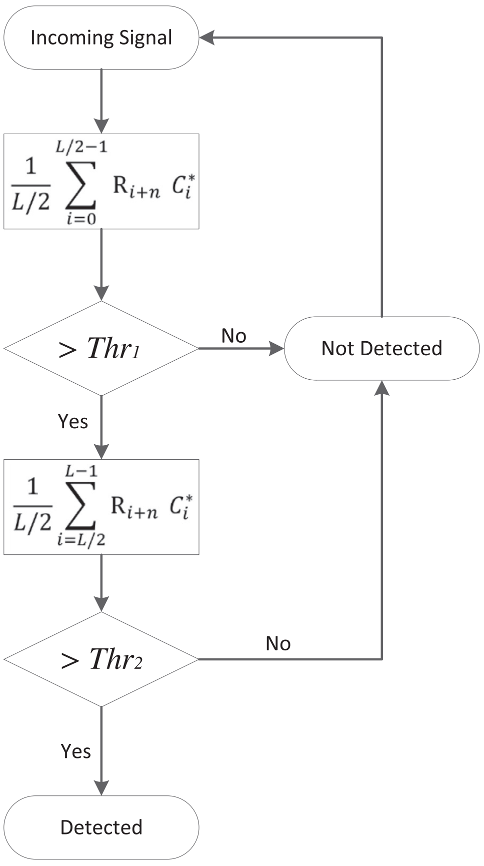

Incoming Signal
$\frac{1}{L/2}\sum_{i=0}^{L/2-1} R_{i+n} C_i^*$
$> Thr_1$
No
Not Detected
Yes
$\frac{1}{L/2}\sum_{i=L/2}^{L-1} R_{i+n} C_i^*$
$> Thr_2$
No
Yes
Detected

Fig. 5. 2-step threshold detection flowchart [21].

approximation for Thr_1 value could be 45% of Thr_2 for a 2-step algorithm [21].

5.3. Controller block

The *Controller* block has the highest responsibility in the design. The most prominent objective of the controller is to make a proper action based on received information by continuously monitoring the behavior of other blocks. At the start of the transmission, the synchronizer should perform autocorrelation between the incoming signal with a delayed version of itself. Therefore, the controller configures the synchronizer in such a way to perform autocorrelation function. It is simply done by routing the input signal to the coefficient registers of the FIR filter with a delay length of D (which in this case study is $D = 16$). This can be simply done using a multiplexer with two inputs where the first input is the incoming signal and the second one is the signal driving by the memory. The controller sets the multiplexer to logic ‘0’ and ‘1’ in order to perform autocorrelation and crosscorrelation, respectively. As soon as the coefficients of the FIR filter are fed by the incoming signal, the controller freezes the coefficient register bank of the FIR filter to avoid overloading coefficients overloading. Thereafter, the controller waits for receiving a response from *Threshold Detection* block. As soon as the autocorrelation peak is found, the threshold detector will inform the controller by setting the corresponding autocorrelation peak input. Then, the controller immediately changes the functionality of the multicorrelator to perform crosscorrelation between a buffered version of the incoming signal and the regenerated time-domain preamble. The Crosscorrelation function is executed in a similar way to autocorrelation. This time, the coefficients of the FIR filter are filled by the data driven by the memory, instead of being filled with the incoming signal. Then, the controller issues the read command to the memory block by issuing the corresponding address. Hence, the proper coefficient is loaded from the memory and is fed to the multicorrelator per clock cycle. When the multicorrelator is filled with the proper coefficients, the controller freezes the coefficient input to the multicorrelator and orders the FIR filter to perform the proper action. Once again, the controller waits for threshold detector to detect if there exists any crosscorrelation peak. Whenever the second peak is found, the controller disables the FIR filter and, consequently, any incoming packet will be discarded afterward. The current packet is handed to the *Packet Decoding* unit to extract the useful data. However, since the ultimate goal of the synchronizer is only packet detection, the *Packet Decoder* unit is not a concern. When the data is decoded, the corresponding unit informs the controller to restart the synchronization process. Furthermore, the controller block is capable of reconfiguring the number of taps of the FIR filter at run time in terms of reconfigurability. This assists the FIR filter to reconfigure its taps based on the information provided by the controller.

5.4. FIR filter block

Conceptually, the reconfigurability is referred to the *FIR Filter* section of the synchronizer. Moreover, the basic idea of the CR technology is how to enable a single hardware platform to support multi-standard communications on a single chip. This can be achieved by replacing analog signal processing with digital signal processing. Due to the nature of NC-OFDM-based CR, the highest order of the filter, e.g. 4096-tap, is required on the receiver side to detect secondary users within the spectrum. Technically, the power dissipation along with the silicon area of a secondary NC-OFDM-based receiver are two major concerns in such a system [22]. These problems are mainly emerged due to the power-hungry MAC operations performed by the FIR filter. Therefore, an appropriate hardware implementation of an FIR filter has the potential to have a massive impact on the overall performance of the receiver. Based on Eq. (1), a complex FIR filter continuously calculates $y(n)$ as complex products of conjugated coefficients $c(k)$ with the signal $x(n-k)$ on a window whose length is N which is also known as the MAC operation [23]. The window N is also referred to as the number of filter taps or filter order and the conjugation is performed by inverting the sign bit of the imaginary part of the coefficients of the FIR filter.

$$y(n) = c(n) * x(n) = \sum_{k=0}^{N-1} c(k)x(n-k) \tag{1}$$

By taking Eq. (2) into consideration, a complex multiplication is derived as

$$\begin{aligned}(R_1 + I_1 i) \times (R_2 - I_2 i) &= R_1(R_2 - I_2 i) + I_1 i(R_2 - I_2 i)\\ &= R_1R_2 - R_1I_2 i + R_2I_1 i - I_1I_2 i^2\\ &= R_1R_2 + I_1I_2 + (R_2I_1 - R_1I_2)i\end{aligned} \tag{2}$$

where R and I are the *Real* and *Imaginary* input signals, respectively. Please note that the $(R_2 - I_2 i)$ is the conjugated version of the signal. Therefore, each complex FIR filter computes both the Real and the Imaginary input signals by using four multipliers alongside two adders. In addition to that, the hardware implementation of a multiplier is much more area expensive and slower than implementing an adder [24].

There are other solutions to replace a multiplier with the cost of introducing few adders. For example, a complex multiplier can be replaced by three adders in Golub’s method [25]. In this method, a complex multiplication is calculated as Eq. (3):

$$\begin{aligned}(R_1 + I_1 i) \times (R_2 + I_2 i) &= [(R_1 + I_1)(R_2 - I_2) + R_1I_2 - R_2I_1]\\ &\quad + (R_1I_2 + R_2I_1)i\end{aligned} \tag{3}$$

where the Eq. (4) is the conjugated version of the Eq. (3) is:

$$\begin{aligned}(R_1 + I_1 i) \times (R_2 - I_2 i) &= [(R_1 + I_1)(R_2 + I_2) - R_1I_2 - R_2I_1]\\ &\quad + (R_2I_1 - R_1I_2)i\end{aligned} \tag{4}$$

By taking Golub’s method into consideration, each complex multiplication is performed by using three multiplication along with five addition. Potentially, complex multiplications not only alter the critical path, but also they have the potential to increase hardware resource usage two times more than a simple multiplication. Hence, resource utilization is significantly important when designing such a large complex FIR filter.

Karatsuba introduced a very similar method, which also known as Karatsuba MuLtiplication (KML), in which a multiplier can be replaced by few adders [26,27]. If we expand the KML to the complex multiplication in the example, Eq. (5) will be derived as:

$$(R_1 + I_1 i) \times (R_2 - I_2 i) = (R_1R_2 + I_1I_2) + (R_2I_1 - R_1I_2)i \tag{5}$$

If we assume $P = (R_1 + I_1) \times (R_2 - I_2)$, $R = R_1R_2$ and $I = I_1I_2$, then we will have:

$$\begin{aligned}P &= R_1R_2 + R_2I_1 - R_2I_1 - I_1I_2\\ &= R + R_2I_1 - R_2I_1 - I\\ R_2I_1 - R_2I_1 &= P - R + I\end{aligned} \tag{6}$$

Ultimately, Eq. (7) will be derived as:

$$\begin{aligned}(R_1 + I_1 i) \times (R_2 - I_2 i) &= (R - I) + (P - R + I)i\\ &= R_1R_2 + I_1I_2\\ &\quad + [(R_1 + I_1)(R_2 - I_2) - R_1R_2 + I_1I_2]i\end{aligned} \tag{7}$$

So far, we have introduced three different methods to calculate any complex multiplication. In general, complex multiplications not only alter the critical path, but also they have the potential to increase hardware resource usage up to four times more than a simple real multiplication. Hence, resource utilization is significantly important when designing such a large complex FIR filter. Later in Subsection 6.1, we will further explain which approach is the most suitable candidate in our design, based on the specific FPGA device.

Technically, the proposed multicorrelator operates in two different modes as follows:

5.4.1. MAC-based multicorrelator

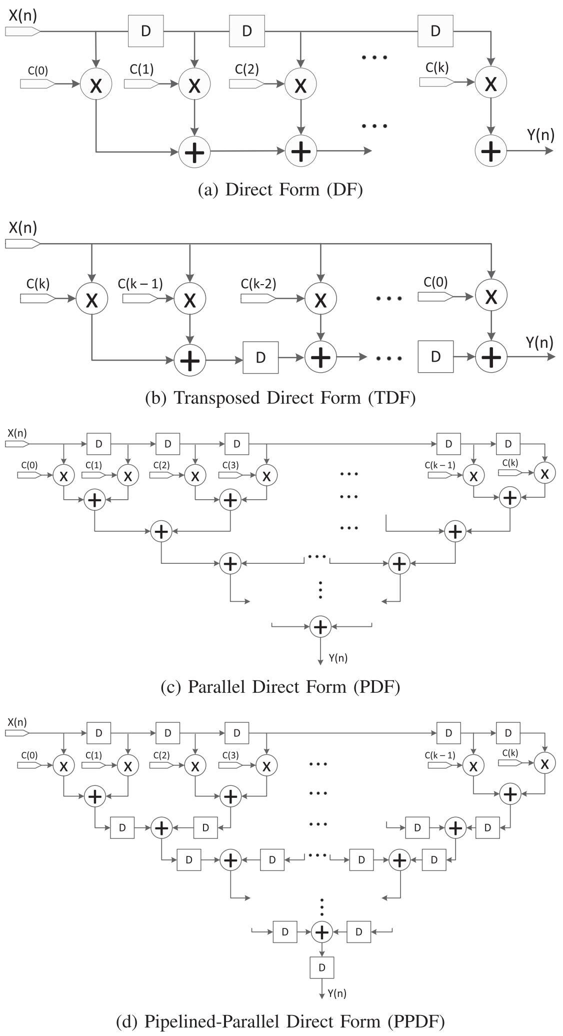
In principle, FIR filters are composed of three fundamental functional units. These three major parts are adders, multipliers and delay elements integrated and arranged in different ways to form different FIR architectures. In general, an N-tap FIR filter consists of N delay elements, N multipliers, and $N-1$ adders. FIR filters are mostly represented as two well-known architectures known as Direct Form (DF) and Transposed Direct Form (TDF), each of which is capable of computing $y(n)$ [28].

Fig. 6a illustrates the DF architecture where the input signal are propagated through the delay elements in the input path. Intuitively, this architecture employs a long serial adder tree, as well. Therefore, a DF FIR filter with a large number of taps introduces a long critical path started from the input to the output. If we take into account that each multiplication introduces T_m delay and each addition takes T_a units of time, the critical path of the DF is equal to $T_{crit} = T_m + 255T_a$ for a 256-tap FIR filter. Thus, the DF architecture is not a good candidate for high-order FIR filter designs and, henceforth, we will not study this architecture in the rest of this paper. However, we have provided some synthesis results in Section 7 which reveal why this architecture is not recommendable.

Fig. 6b shows an optimized version of the DF known as TDF configuration. In this architecture, since the delay elements are placed between adders, the critical path of the design is limited by $T_{crit} = T_m + T_a$, irrespective of the number of filter taps. This alteration requires the reversed order of the coefficients $C_{(k)}$, i.e. $(C_k, C_{k-1}, \ldots, C_0)$, to be fed to the system. Thus, the TDF does not include any register on the input path. However, a large TDF-based FIR filter has the potential to introduce a long interconnection on the input path [29]. In addition, this well-known architecture suffers from exploiting 3-input adders due to its architectural nature. We will investigate this issue later in Section 7.2 in detail.

In this work, similar structures such as Parallel Direct Form (PDF) and Pipelined-Parallel Direct Form (PPDF) are investigated and implemented, as well. Fig. 6c shows the PDF configuration where instead of implementing a serial chain adder, a binary adder tree is implemented. The idea behind this architecture is to cope with the long interconnection in the input path of the TDF model as well as the critical path introduced in the DF architecture to some extent. The number of required adder levels are proportional to the $\lceil \log_2 N \rceil$ where N is the number of taps. Consequently, the critical path of a design is dependent on N. For example, the critical path of a 256-tap FIR filter is reported as $T_{crit} = T_m + (\lceil \log_2 N \rceil \times T_a) = T_m + 8T_a$. The hardware implementation procedure of this architecture is more complex than the DF and TDF forms.

The PPDF architecture shown in Fig. 6d is the pipelined version of the PDF model where the critical path is more limited than the one in PDF. The inserted registers maintain the critical path at its minimum level ($T_{crit} = T_m + T_a$) by the cost of introducing a variety of register elements to the system. Moreover, the PPDF architecture has the same complexity as the PDF form. Section 7 provides more information relevant to the hardware cost with respect to the above-mentioned architectures. Although these architectures



(a) Direct Form (DF)
(b) Transposed Direct Form (TDF)
(c) Parallel Direct Form (PDF)
(d) Pipelined-Parallel Direct Form (PPDF)

Fig. 6. Different architectures for MAC operator unit.

(PDF and PPDF) are quite similar to the binary adder tree proposed in [30], bear in mind that we are dealing with complex FIR filter structures in this work.

In this work, irrespective to the architecture, coefficient inputs, as well as the incoming signal, are 32-bit signal vectors where we stipulated that the first half (including 16 LSB bits) is the Real part and, subsequently, the remaining 16 bits (MSB half) contains the Imaginary part of the corresponding input. Note that the multiplication block performs a complex multiplication which is more complex and power-hungry in comparison with the real-valued multiplication from the hardware point of view. In this work, we used fixed-width truncation in which only 16 MSBs of 32-bit products are assigned to the output. FIR filters are intensive parts of a CR due to the massive workload of complex computations as well as operating at high speed to achieve highest sampling rate in power consumption point of view. In Section 7, resource allocation and power consumption for different FIR filter architectures are described in detail.

5.4.2. *ML-based multicorrelator*

The Multiplier-Less (ML) technique is an *approximate computing* approach in order to exploit an energy efficient design by omitting computation of some less significant bits [31]. In an ML-based multicorrelator, the size of the correlator, as well as the number of registers, is depending on the number of input samples. According to [32], instead of performing 16-bit multiplication, a sign bit correlation between the coefficients and input signal is done which can be simplified to either an XNOR or XOR gate. Similarly, in [33] coefficients are represented in the form of summed powers of 2. In both cases, a massive number of registers and multiplications, which are the most power-hungry operations in FIR filters due to their dynamic power consumption, are mitigated. Although the multicorrelator uses only 1-bit word instead of 16-bit word, simulation results given by the ModelSim software show that the performance of the FIR filter is still at a satisfactory level for this application. Authors in [16] acknowledged that the sign-based correlator has reasonable result in low SNR regions, as well. Intuitively, it works better in higher SNRs regions. They have also claimed, by implementing both multiplier-based correlator and sign-based correlator in MATLAB and testing both with 802.11a/g packets, the results were still satisfactory. Based on that, we implemented the ML-based multicorrelator for autocorrelation purpose in hardware. Our observation of performing autocorrelation function based on ML approach has the potential to overcome the conventional MAC approaches. The implementation results of the ML-based along with the MAC-based multicorrelator are presented in Section 7 in terms of implementation summary, maximum frequency, power consumption, etc. However, the ML-based approach still requires more practical studies in the field of communications due to presenting an approximate computation [34].

5.5. *Partial reconfiguration (PR)*

The partial reconfiguration is an ability in which a portion of FPGA is reconfigured while the rest of the design is operating. This feature enables a particular region of the design to have multiple configurations while everything outside of this region is operating normally. Although the PR has several applications, in general, it is used in designs which require operating continuously while some particular regions can be reconfigured without disrupting the operating parts [35]. In this work, the PR host is implemented as an internal host, meaning that the core has the potential to execute the PR itself. When the device is ready to perform PR, the PR host sends the PR request to the hard controller block inside the FPGA. The controller acknowledges the PR host through a “Ready” signal. Thereafter, the host transmits the configuration *bitstreams* and waits for an acknowledge from the PR controller block. The configuration bitstreams are generated by the Quartus-II software during the synthesis time. Similar to other communication systems, if an error occurs, the controller informs the host immediately. In the same way, if the PR is successfully performed, the controller informs the host again.

Reconfiguring portions of the FPGA on-the-fly is the main advantage of using PR feature. Fig. 7 illustrates how multicorrelators perform autocorrelation functions over the wide spectrum. The controller controls each multicorrelator via several control signals including freezing, enabling, crosscorrelation, autocorrelation, etc. signals. Initially, each multicorrelator performs autocorrelation functions over a subset of subcarriers. As earlier studied in this paper, the ML-based approach achieves acceptable performance for autocorrelation function by replacing the conventional MAC operation with an inexpensive XOR/XNOR gate. instead of the conventional MAC operation. On the other hand, there are a huge amount of hardware resources dedicated to each multicorrelator in order to compute crosscorrelation function of a maximum 256-tap FIR

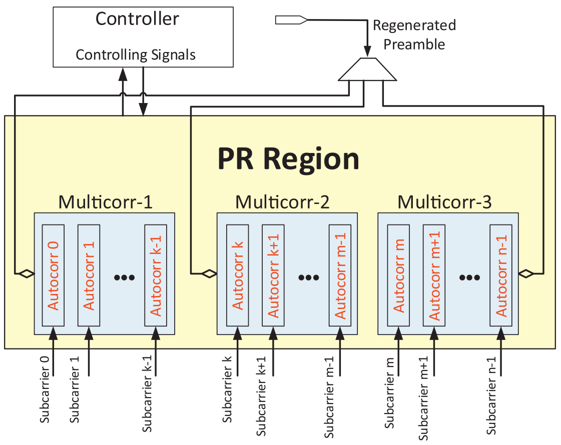


Fig. 7. How multicorrelators perform autocorrelation function using ML-based method. In this case, regenerated preamble ports are dummy interfaces.

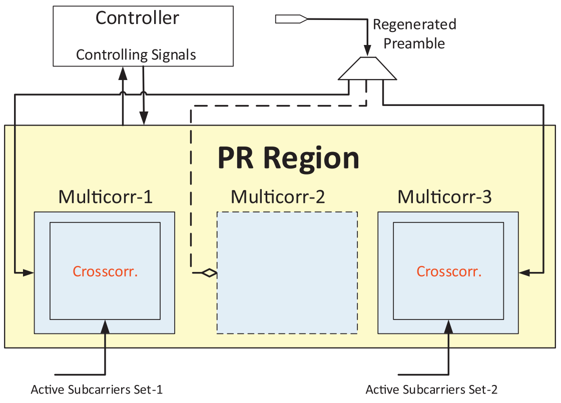


Fig. 8. How multicorrelators are reconfigured to perform crosscorrelation function using MAC-based method.

filter. Therefore, each multicorrelator has the potential to compute autocorrelation results over a large portion of the spectrum. At the same time, the controller consecutively monitors the results of the threshold detector unit of each multicorrelator. As soon as the controller detects the presence of the secondary user, it will send the appropriate command to the PR host to perform reconfiguration process. When the multicorrelators are configured to perform autocorrelation function, the “regenerated preamble” interfaces are considered as dummy ports. It means that the actual ports do exist although they are not connected to the multicorrelator.

Partial reconfiguration is a special feature which is not available on all FPGA boards. It also requires an additional license on top of the one needed for performing the synthesis. Please make sure if your target FPGA board has the capability to run PR before trying to use this feature.

Fig. 8 presents the situation when the presence of the secondary user is detected in two different frequency bands. In this case, the PR host reconfigures the PR region with two multicorrelators to compute 256-tap crosscorrelation function, while it discards the third multicorrelator. Therefore, there will be some free hardware resources which can be taken into other uses, e.g., as hardware accelerators. Once the data transmission is disrupted for any reason, such as the secondary user completed its transmission

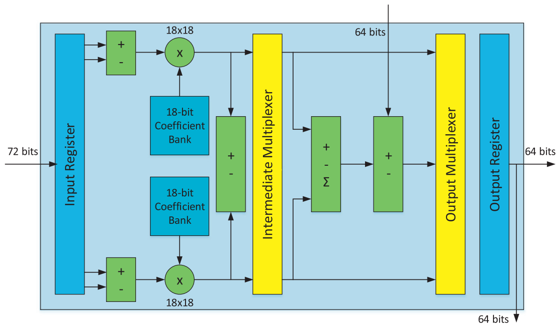


Fig. 9. The DSP architecture of the Stratix V in 18 × 18 complex mode. The architectural structure is suitable for traditional complex MAC operation. Adopted from [39].

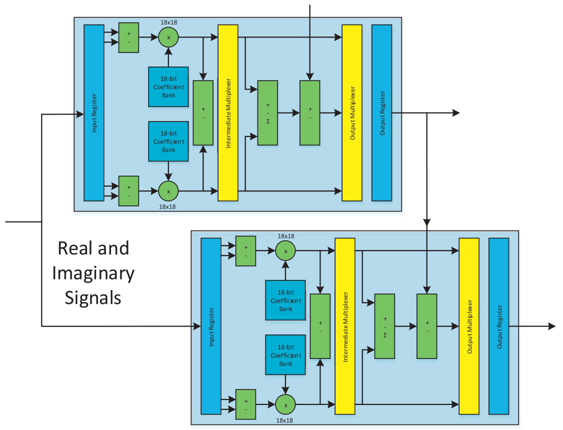


Fig. 10. How the synthesis tool cascades two DSP blocks to perform complex multiplication. The combination of using two DSP is suitable for performing complex multiplication based on traditional algorithm. Adopted from [39].

or the multicorrelators fail to determine acceptable results, the PR host reconfigures the PR region in a way to start sensing the spectrum again. In this figure, we only show three multicorrelators due to the design choice as well as the constraints of the target FPGA board. In next section, we investigate these issues more in detail.

Another advantage of employing PR feature is the capability of uploading different IEEE standards to the same receiver on-the-fly. For example, one multicorrelator is configured to receive IEEE 802.11x standard whereas another one can receive and decode IEEE 802.22 packets. A CR receiver is more versatile due to using such an impressive feature.

6. FPGA implementation constraints

As previously explained, an NC-OFDM-based CR receiver has to collect all the available subcarriers in the spectrum to perform a synchronization procedure. Therefore, the FIR filter section of the multicorrelator should employ the highest possible filter order, e.g., 4096-tap filter. In contrast to the software perspective where implementing a 4096-tap FIR filter is feasible, our experience revealed various limitations when the design is implemented on FPGA. In the following, a few of such constraints are listed:

6.1. Insufficient DSP blocks

Generally speaking, the Digital Signal Processing (DSP) blocks are suitable for implementing chain adders, multiplier and a combination of both. Technically, the main applications of the DSPs are in computationally intensive tasks such as digital filtering algorithms where a massive number of MAC operation are required. Depend on the target FPGA device, each DSP block is suitable to perform the MAC operation in the best efficient way. For example, Stratix II and Stratix III device families employ extremely efficient DSP blocks to compute Karatsuba algorithm [36]. Therefore, it is the best to exhaust the DSP block as much as possible before going to perform MAC operations in logic cells.

Our target FPGA device, Stratix V, provides variable-precision as well as fixed-precision DSP blocks. In the fixed-precision mode, each DSP block can operate various configurations of a MAC operation including three 9 × 9 multipliers, two 16 × 16 or one 18 × 18/27 × 27 multipliers. For example, Fig. 9 depicts the internal architecture of the Stratix V FPGA device in 18 × 18 complex mode. Furthermore, multiple DSP blocks can be cascaded together to perform complex computations [37]. Fig. 10 illustrates how the complex multiplication is done by cascading two DSPs.

As in earlier sections mentioned, there are different alternatives to calculate a complex multiplication. The multiplication algorithm that we take into account is the traditional method which requires 4 multipliers alongside 2 adders. Although other mentioned algorithms, Golub's and KML, take advantage of employing one multiplier less than the conventional method with the cost of inserting few adders, the Stratix V DSP block provides a specific mode exactly for computing complex multiplications as the traditional method [38]. Hence, the traditional complex multiplication algorithm seems to be the best candidate to fit into the DSP blocks. Moreover, we investigated the effect of employing both Golub's algorithm and KML method, instead of the traditional algorithm. Surprisingly, the hardware costs, including resource utilization, number of used registers, DSP blocks, etc., drastically increased. This is due to the fact that the DSP blocks cannot fully be utilized. When we further investigated, we discovered that the synthesis tool employs 3 DSP blocks to compute none-traditional complex multiplication algorithms. We anticipate that, perhaps, one reason could be that the synthesis tool utilizes two DSPs in medium-precision mode and the other one in high-precision mode. Further information with respect to the precision modes can be found in [39]. It is crucial to know the FPGA platform in detail prior to writing the hardware description of the design. The traditional complex multiplication method is suitable for Stratix V families. Perhaps, none-traditional algorithms (such as Golub's and KML) would be more efficient than the traditional algorithm in other platforms such as Stratix II families.

In our application, the I/Q signals (which are the real/imaginary part of the original signal) are 16-bit wide, each of which requires one DSP block to compute a complex MAC function. For example, a complex 256-tap FIR filter exploits 512 DSP blocks. The Altera's Stratix V FPGA board that we used provides only 1590 DSP blocks. It means the highest order for such a filter cannot exceed 795 taps. Since FIR filters are generally implemented as powers-of-two, i.e. 128-tap/256-tap, the FPGA board could only support complex FIR filter with up to 512 taps. Insufficient DSP blocks are the main limitation of the platform from the hardware point of view.

There are two approaches to cope with DSP limitation. One solution is to force the compiler not to use DSP blocks automatically. The synthesis tool will then implement the MAC operations as combinatorial logic using the FPGA look-up tables. This is potentially 10 times more expensive than exploiting existing DSP blocks. Another alternative is to sacrifice the precision of the filter. For instance, instead of having three 256-tap filters, we can exploit

Table 1
Preliminary results for different implementations of the multicorrelator.

	TDF	DF	PDF	PPDF	ML
Logic Utilization (ALMs)	12,483	12,490	7504	14,611	1876
Total DSP Blocks	512	512	512	512	0
Total Registers	16,378	16,883	8873	28,037	2886
Max. Freq. [MHz]	237	68	88	240	257

Table 2
Power consumption analysis (mW).

	TDF	DF	PDF	PPDF	ML
I/O Power	22.58	2108.14	33.92	22.86	26.20
Dynamic Power	344.17	200.59	265.89	160.10	9.42
Static Power	1070.06	1062.71	987.12	955.80	951.50
Total Power	1436.82	3371.43	1286.92	1138.75	937.13

Table 3
Power dissipation by hierarchy (mW).

	TDF	DF	PDF	PPDF	ML
Memory Block	0.78	0.81	1.21	0.80	0.79
Threshold Block	0.01	0.01	0.01	0.01	0.01
Controller Block	0.02	0.02	0.02	0.02	0.02
FIR Filter Block	230.20	165.57	192.38	128.80	4.18

twelve 64-tap filters with approximately the same amount of hardware resources. This strategy entirely depends on the application as well as the environment in which the CR receiver is being used.

6.2. Silicon area

The FIR filter is the major component of the multicorrelator which employs a large number of resources. Having an *N*-tap complex FIR filter not only requires $2 \times N$ DSP blocks, but also increases the total number of fundamental components such as shift registers, register bank, adders, etc. A large complex filter, e.g. 4096, has the potential to employ most of the available resources of the target FPGA.

6.3. Power consumption

The total power consumption of each configuration is dependent on three components: total static power, total dynamic power as well as the total I/O power of which the dynamic power along with the I/O power dominate the total FPGA power dissipation [40]. Static power refers to the power consumed when there is no circuit activity. Hence, the static power is approximately the same for all configurations. Dynamic power consumption is the power consumed by the FPGA when the transistors toggle and capacitive loads are charged and discharged, respectively. The I/O power of an FPGA includes the power consumed for the signal starting from the input, traversing the circuit and ending at the output. In our case study, one of the main resources of dynamic power dissipation is the massive workload of the large complex computations of a complex FIR filter. On the other hand, as the order of the complex filter grows, long interconnections are introduced to the system which have the potential to increase the I/O power consumption.

7. Experimental results and further discussion

In this section, the proposed multicorrelator described using Very-high-speed integrated circuit Hardware Description Language (VHDL), simulated by ModelSim software, compiled and synthesized by Quartus-II 12.1 and 15.1 environments and eventually implemented on Altera FPGA device targeting Stratix-V speed grade 2 family. It is worth mentioning that the target FPGA board has the capability to perform partial reconfiguration on demand.

7.1. Preliminary results

Synthesis results for a multicorrelator configured to a 256-tap FIR filter are reported in Table 1 in terms of logic utilization, total DSP blocks, total registers and maximum frequency. The results are purely presented based on the information obtained by the Quartus-II version 12.1 software without using any optimization technique. Unfortunately, the presented results for the PDF configuration in [7] are not the correct (as Fig. 6c depicts) due to a design error made in the HDL. Therefore, the updated results are presented in this section.

The results show that the ML architecture is the preferable configuration among the implemented architectures, whereas the DF configuration holds the worst records. Moreover, the PPDF architecture employs more registers in comparison with other architectures due to its pipeline-parallel-based structure. However, the PPDF architecture does not yield an impressive advantage by using extra registers. In ML design, logic utilization is decreased drastically since a large number of DSP blocks, as well as registers, are mitigated while the MAC-based architectures exploit 512 DSP blocks with more registers. Therefore, ML architecture preserves more than 82% silicon area in comparison with other architectures. However, the ML-based multicorrelator is a better candidate to perform autocorrelation function rather than the crosscorrelation. Although the PDF configuration does not yield acceptable results in terms of maximum frequency, it employs the least number of ALMs along with total used registers within MAC-based architectures. Excluding DF and PDF models, the maximum frequency in other architectures is approximately similar to each other. Technically, it seems that the TDF architecture is the preferable MAC-based configuration based on the given results.

Power dissipation analysis is reported based on the results given by PowerPlay Power Analyzer Tool in Quartus-II software. The analyzer directly reads the waveforms generated by the ModelSim running at 50 MHz. Static probability and toggle rate for each signal are calculated based on reported VCD file. Table 2 summarizes the estimated results. The DF architecture has the most power dissipation among the other implementations due to the long critical path existing in the I/O. As it is shown, the dynamic power is drastically decreased in ML architecture due to avoiding a number of power-hungry multipliers and adders. The ML-based design saves more than 94% of dynamic power consumption in comparison with PPDF form (the lowest power consumer in MAC-based architecture). PPDF architecture has the lowest power consumption among the MAC-based architectures by exploiting several levels of registers. Although the PDF model did not yield satisfactory operating frequency, it is an acceptable candidate in terms of power dissipation. The TDF configuration has the most power dissipation within the MAC-based architectures.

The Dynamic power dissipation by hierarchy for different architectures are reported in Table 3. As it is obviously clear, the FIR filter is the most power-hungry component in MAC-based architecture while the controller block, as well as the threshold block, is the least energy consumer. In the ML architecture, the power dissipation problem relative to the FIR filter block is resolved by using a sign bit correlation instead of 16-bit multiply-accumulate operations. Table 4 shows power dissipation for each cell in detail. The results show that ML architecture achieves the best result, almost in all aspects, in comparison with MAC-based architectures. The reason why the DF has a massive I/O power dissipation can be explained as the existence of the long critical path made in the

Table 4
Total power consumption for each cell(mW).

	TDF	DF	PDF	PPDF	ML
DSP Block	216.48	130.40	157.76	95.20	0
Combinational Cell	76.58	10.78	51.81	1.30	1.87
Register Cell	2.09	30.31	10.66	32.63	2.45
I/O	25.15	2090.09	15.79	4.23	7.44

Table 5
Summary of the maximum optimization gained by the synthesis tool.

	ITDF	IPDF	IPPDF
Logic Utilization (ALMs)	8417	7336	9856
Improvement	32.57%	2.24%	32.54%
Total Registers	16,407	8925	21,197
Improvement	−0.18%	−0.59%	24.4%
Max. Freq. [MHz]	258	94	268
Speed-up	8.86%	6.62%	11.66%
	1.09×	1.06×	1.11×

design. The signal should have adequate energy to be able to traverse the critical path made by a multiplier and 255 adders. Hence, the DF architecture, which had the minimum operating frequency, is not a wise configuration from the power consumption point of view, as well. Next section provides useful information with respect to the refinement techniques we used to improve the given results in the tables above.

7.2. Improved results

As previously discussed, we would avoid further studying the DF architecture due to its unacceptable results. Furthermore, the ML architecture is not a reliable candidate to perform crosscorrelation function. Hence, we tried to make our best effort to minimize the logic utilization as well as the power consumption, while increasing the maximum frequency of the other aptly named MAC architectures. It is worth mentioning that the FIR filters should perform more intensive computation to calculate the crosscorrelation function rather than the autocorrelation. Thus, the hardware cost of performing crosscorrelation is significantly more expensive than in autocorrelation. Therefore, we mainly concentrated on optimizing MAC-based architectures at this stage. The optimization procedures are as following:

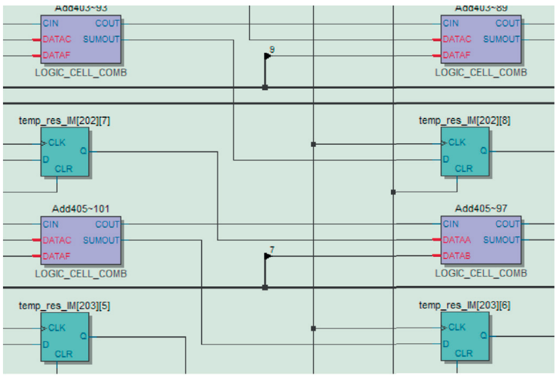
HDL Modification. Typically, the optimization techniques start with revising the HDL code. Modifications such as eliminating unnecessary registers, shortening the critical path using more parallelism, etc., assist in obtaining better performance. At this stage, we also found a bug in the architectural description of the PDF model where the preliminary MAC results were registered at the first level of the chained adders. Hence, the critical path was shortened by taking advantage of additional registers.

Optimization Techniques Offered by Synthesis Tool. Quartus-II provides a set of recommendations (including timing optimization, resource optimization, power optimization, etc. recommendations) to improve the overall performance of the design. At this stage, we pushed more towards the timing optimization, while minimizing the logic utilizations along with the power dissipation. Table 5 presents the maximum optimization gained by taking advantage of recommendations offered by the synthesis tool. Although the Quartus-II was successful to improve the logic utilization as well as the maximum frequency to some extent, it had a negative impact on employing total registers in Improved TDF (ITDF) along with the Improved PDF (IPDF) configurations. Furthermore, optimizing design using synthesis tool approach was quite successful for Improved PPDF (IPPDF) architecture. As a result, the IPPDF configuration gained 11.66% speed-up (1.11×), 24.4% less registers alongside 32.54% less logic utilization. On the other hand, the synthesis tool approach was not very successful for IPDF architecture in comparison to other models. Overall, Quartus-II optimization recommendations have the potential to improve some part of the design to some extent.

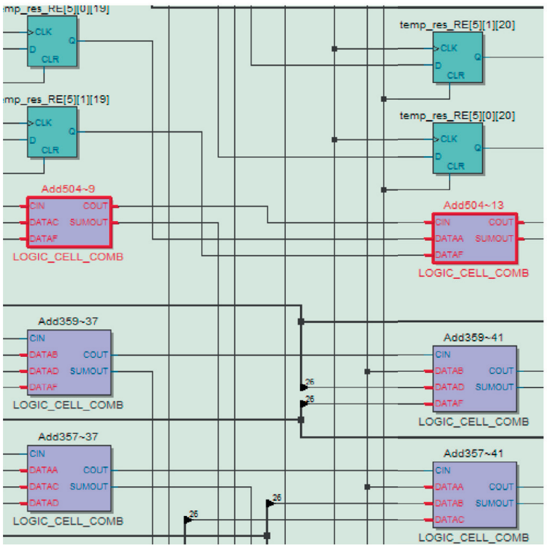
Exploiting Ternary Adders. In 2005, Altera corporation published a white paper which encouraged digital designers to exploit 3-input adders (also known as ternary adder structure) instead of 2-input ones (binary adder) [41]. It means, in order to utilize an adder in an efficient way, the designer should explicitly design the code, e.g., $d = a + b + c$. Back to that time, perhaps, the synthesis tools were not smart enough to put their efforts on replacing ternary adders with binary ones. We also investigate this issue to verify how today's synthesis tools manage it. Therefore, we took into consideration the generated netlist after place and route using the Technology Map Viewer offered by the Quartus tool. As Fig. 11a shows, the synthesis tool was not able to exploit ternary adders due to the nature of the ITDF architecture. Similarly, the IPPDF architecture suffers from the same problem. Fig. 11b illustrates a portion of the IPDF netlist. It seems that the synthesis tool changes the design in a way to exploit ternary adders whenever it is possible. However, a few binary adders exist in the netlist. The algorithm being used by the synthesis tool might be similar to [42,43]. The hardware description of a large adder tree which employs 3-input adders is a complicated task from the hardware designer's point of view.

Intermediate Register Insertion. In this stage, the slowest architecture (PDF model due to reporting the worst-case maximum frequency of 94 MHz) was the first target which required maximum optimization to reach the maximum possible frequency. After analyzing the critical path, we discovered that although the adder tree is limited to 8 levels (in this study), the design mainly suffers from the long combinational path produced by the chained adders. Inserting Intermediate Register (IR) is an alternative solution to minimize the critical path by the cost of using more registers. However, inserting IR at each level is exactly what we applied to implement PPDF architecture. Hence, the critical path was investigated more in detail to discover the best suitable locations for IR insertion. We investigated that inserting IR at 1st, 4th and 7th levels would increase the frequency to 240 MHz by introducing 19,051 registers alongside 8777 ALMS meaning that the achieved speed-up was 2.55× (155.32% faster) with the cost of 113.46% (2.13×) more registers as well as 19.54% (1.2×) more logic elements. Furthermore, we inspected that the design was also suffering from the critical path made by DSP blocks due to not being fully utilized. It means unregistered data path which goes through a DSP block does not fully utilize the embedded DSP block which drastically degrades the overall performance. Therefore, we exploited the free embedded register available inside the DSP block. The main advantage of such embedded register is providing a free register element which significantly helped us to remove the 1st level of IR. Hence, with the new conditions we had to move IRs one step backward. Eventually, the IPDF architecture could achieve the maximum frequency of 249 MHz (2.65× speed-up) by employing 11,282 (26.41%) more registers while employing 5671 (22.7%) fewer ALMs. Fig. 12 illustrates the above-mentioned scenario where we exploit DSP registers with no additional cost while distributing IRs in appropriate locations at every 3rd level.

The PPDF architecture could not benefit from intermediate register insertion technique due to the nature of its architectural



(a) The TDF architecture only employs binary adders



(b) The IPDF architecture mostly exploit ternary adders. However, there are still a few binary adders which could not be implemented as ternary adders.

Fig. 11. A snapshot from technology map viewer.

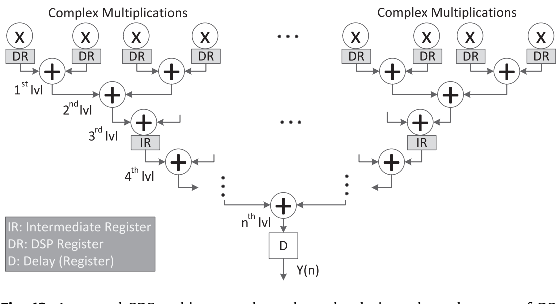


Fig. 12. Improved PDF architecture shows how the design takes advantage of DRs as well as distributed IRs to achieve the maximum possible frequency.

Table 6
Synthesis summary for of the maximum improvement applicable to the MAC-based architectures.

	ITDF	IPDF	IPPDF
Logic Utilization (ALMs)	8519	5671	6345
Improvement	31.75%	24.43%	56.57%
Total Registers	16,345	11,282	17,141
Improvement	0.20%	−27.15%	38.86%

Table 7
The maximum operating frequency gained for improved MAC-based architectures.

		Maximum Frequency [MHz]		
		ITDF	IPDF	IPPDF
Slow Model (SM)	85 °C	280	249	293
	0 °C	289	258	315
Fast Model (FM)	85 °C	378	347	387
	0 °C	404	373	419
Speed-up at SM 85 °C		1.18×	2.83×	1.22×

structure. Hence, the PPDF could only take advantage of DSP register packing. In order to do that, we eliminated all the registers located at the first level of the adder tree (visible in Fig. 12) while employing DSP register packing technique. This infers that the first level of the registers is shifted one step backward. As a result, a significant amount of logic modules along with a massive number of registers are released while the maximum operating frequency increased to 293 MHz.

Similar to PPDF, the TDF architecture could not take advantage of register insertion technique either. Therefore, we could not refine the TDF design by applying intermediate registers. What we could do to further optimize these designs was employing available DSP register technique. The TDF architecture takes advantage of employing DSP registers with no further structural alteration as well as the hardware cost. As a result, the TDF architecture could achieve the maximum operating frequency of 280 MHz on the slow model at 85 °C. The next drawback of this architecture is the architectural problem which cannot exploit 3-input adders.

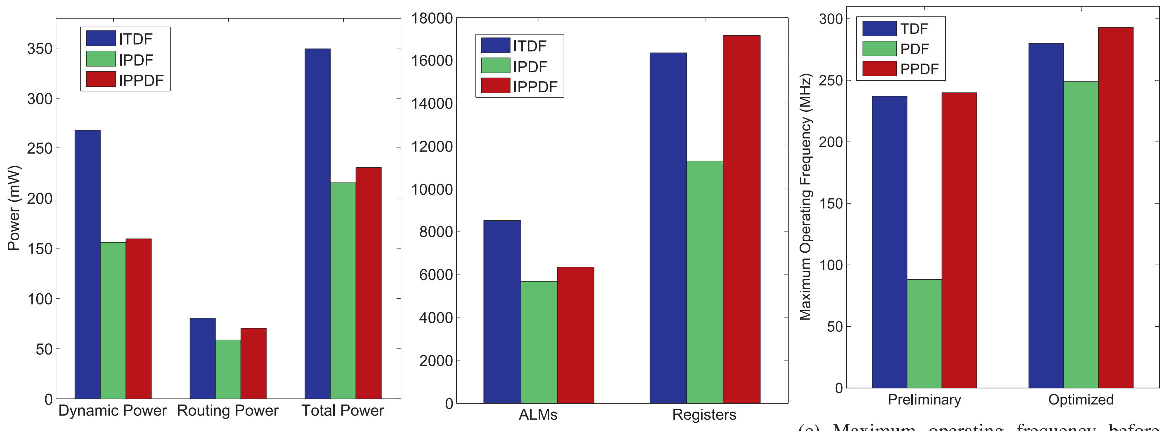
Table 6 summarizes the maximum achieved optimization by applying previously stated optimization techniques on selected MAC-based architectures. The presented results are based on the information obtained from the Quartus-II version 15.1 software. The results reveal that the IPPDF configuration takes the most advantages of applied improvement techniques by achieving 56.67% and 38.86% better results in terms of logic utilization and total registers, respectively. The IPDF model requires 24.43% fewer logic modules by the cost of employing 27.15% more registers to achieve acceptable results. The ITDF architecture achieves 24.43% fewer logic modules while the total used registers have a trivial refinement. Overall, the refinement techniques were partially successful in achieving less logic utilization as well as total used registers.

Table 7 presents the maximum frequency achieved after applying refinement techniques on namely MAC-based FIR filters. As it is shown, the IPDF configuration achieved the most speed-up of 2.83× compared to the PDF configuration in the slow model at 85 °C. However, the IPDF architecture employed more registers to target the speed-up. Moreover, 1.22× and 1.18× higher operating frequency could be achieved for IPPDF and ITDF, respectively. The multicorrelator configured with the IPPDF architecture is the fastest design (293 MHz), the second option is ITDF model (280 MHz) while the slowest configuration is IPDF (249 MHz).

Table 8 shows the total power dissipation in each cell of the mentioned architectures after the refinements in detail. It can be seen that the IPDF architecture along with the IPPDF has achieved

Table 8
Power consumption of each cell after optimization.

	Dynamic Power (mW)			Static Power (mW)			Routing Power (mW)			Total Power (mW)		
	ITDF	IPDF	PPDF	ITDF	IPDF	PPDF	ITDF	IPDF	PPDF	ITDF	IPDF	PPDF
Memory Block	0.61	0.62	0.62	–	–	–	0.01	0.02	0.02	0.62	0.64	0.64
DSP Block	257.81	143.99	143.99	–	–	–	0.46	1.71	4.24	258.27	145.70	148.23
Combinational cell	2.75	3.22	3.34	–	–	–	23.46	2.19	1.60	26.21	5.41	4.94
Clock enable	0	0	0	–	–	–	24.84	23.83	29.47	24.84	23.83	29.47
Register Cell	5.17	5.87	7.57	–	–	–	11.54	30.47	34.44	16.72	36.35	42.01
I/O	1.43	2.15	3.91	1.11	1.11	1.11	19.95	0.54	0.50	22.49	3.80	5.52
Total Power	267.77	155.85	159.43	1.11	1.11	1.11	80.26	58.76	70.27	349.15	215.73	230.81



(a) Power dissipation (b) Resource usage (c) Maximum operating frequency before and after applying optimization techniques

Fig. 13. Differences in hardware cost for each architecture.

less power consumption rather than the ITDF model in most of the cases. The ITDF architecture has an excessive power dissipations in comparison with mentioned candidates due to the massive power consumption on DSP blocks, routing as well as the I/O. The main reason for such a large difference is the existence of a long interconnection on the input path which not only affects the maximum frequency but also degrades the performance in terms of dynamic power dissipation. Based on given results, the greenest architecture from the power dissipation point of view is the IPDF model with 216 mW power, followed by the IPPDF and ITDF with 231 mW and 350 mW power consumptions, respectively.

Fig. 13a provides information for the designs where the limiting factor is mainly the power dissipation. In that case, the ITDF has the most power dissipation architecture, IPPDF model has a moderated power consumption while the IPDF has the least power dissipation. Fig. 13b presents information suitable for designs which are mainly constraint by silicon area. For such a design, the IPDF model is the most economic candidate while employing either ITDF or IPPDF have their own trade-offs. Fig. 13c compares above-mentioned architectures with each other in terms of maximum operating frequency. In addition, each architecture is compared to its preliminary version, as well. The IPPDF configuration provides the fastest available speed for the designs where the limiting factor is the speed. The ITDF architecture and IPDF model stand as the second and third candidates, respectively. In our case study, the ideal choice is the IPDF model by considering all above-mentioned trade-offs. As the second choice, we choose IPPDF model over the ITDF one, since it requires 25.52% fewer ALMs along with 33.9% less power dissipation while provides 4.64% higher operating frequency with the payoff 4.64% more registers. Eventually, ITDF (which was the most suitable architecture before the optimization) is now the third option. By taking Table 8 along with the Fig. 13 into consideration, we can conclude that the transposed direct architecture does not yield very acceptable results for high-order FIR filters from hardware costs point of view.

8. Conclusion

In this correspondence, a flexible synchronizer suitable for Non-Contiguous Orthogonal Frequency Division Multiplexing (NC-OFDM)-based Cognitive Radio (CR) systems was developed on an FPGA board. In this work, we mainly tried to study more about hardware implementation issues of the synchronizer. Apart from synchronization problems which exist in NC-OFDM modulation (at a higher level), we tried to reveal obstacles along with challenges in the hardware implementation of such a CR, as well. The synchronizer employed a multicorrelator to perform both autocorrelation and crosscorrelation functions on demand. The core content of the synchronizer was based on FIR filters. Several architectures, including Direct Form (DF), Transposed Direct Form (TDF), Pipeline Direct Form (PDF), Parallel-Pipeline Direct Form (PPDF) and MultiplierLess (ML), with respect to the FIR filters were implemented and analyzed. The DF architecture demonstrated the worst results almost in all cases and, hence, we did not study this architecture. Preliminary synthesis results inferred that the best architecture for performing autocorrelation function would be the ML-based FIR filter in terms of maximum frequency of 257.33 MHz, logic utilization of 1876 ALMs and dynamic power consumption of 9.42 mW, in comparison with MAC-based architectures. However, the ML architecture still requires more practical studies in the field of communications. In MAC-based designs, the fastest architecture was PPDF

(240 MHz) by employing a large set of registers. Thereafter, the PDF architecture did not reach to an acceptable frequency while employing the least logic modules as well as total registers. The preliminary results also show that the TDF configuration would be the most suitable candidate while its dynamic power consumption has the potential to be an issue. Next, we put our best afford to optimize the MAC-based architectures to reach to the maximum available frequency while minimizing the hardware costs including power consumption, less logic utilization, etc. In addition, we studied the effect of employing traditional complex multiplication algorithm (using 4 multipliers), Golub's method as well as Karatsuba (3 multipliers). The traditional algorithm utilized the DSP blocks more than other studied algorithms. Furthermore, we took advantage of optimization techniques such as DSP register packing and intermediate register insertion if applicable. Based on the results after the optimization, all mentioned architectures reached to speed-ups of 1.18×, 2.83× and 1.22× in improved version of TDF, PDF and PPDF, respectively. The synthesis results after optimizing the designs revealed something else. Although the improved PPDF architecture had an impressive resource reduction in terms of logic utilization (56.57%) as well as registers (38.86%), the improved PDF model could achieve the most economic design from hardware cost point of view while it maintained the maximum frequency at a satisfactory level (249 MHz, 2.83× speed-up). Hence, this architecture is the most suitable candidate in our application. The improved TDF architecture could not win the best design in any category and, nevertheless, there are other alternatives as the best solution in practical applications. Technically, improved PDF model by a trivial sacrifice in frequency would be the best candidate in environments where the silicon area alongside the power consumption is the limiting factor. Similarly, when the limiting factor is the maximum frequency, improved PPDF has the best performance among the MAC-based structures. Therefore, there is not a certain answer which configuration is the best candidate, since a trade-off always exists when employing the aptly named MAC-based architectures. As the final words, the achieved results were obtained based on the mentioned FPGA device. Although implementation on other platforms might lead to achieving different results, the overall figures should remain the same.

Acknowledgment

This work was funded by the Academy of Finland under contract #258506 (DEFT: Design of a Highly-parallel Heterogeneous MP-SoC Architecture for Future Wireless Technologies). A portion of this work has been supported by the Jane and Aatos Erkko Foundation, Finland, under the project Biological Neuronal Communications and Computing with ICT.

References

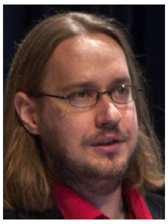
[1] J. Acharya, H. Viswanathan, S. Venkatesan, Timing acquisition for non contiguous OFDM based dynamic spectrum access, in: 3rd IEEE Symposium on New Frontiers in Dynamic Spectrum Access Networks (DySPAN), 14–17, Chicago, IL, 2008, pp. 1–10, doi:10.1109/DYSPAN.2008.57.
[2] Y. Xing, H. Kushwaha, K.P. Subbalakshmi, R. Chandramouli, Codes and games for dynamic spectrum access, in: H. Arsalan (Ed.), Cognitive Radio, Software Defined Radio, and Adaptive Wireless Systems, Ch. 6, Springer, 2007, p. 163.
[3] A. Dutta, D. Saha, D. Grunwald, D. Sicker, Practical implementation of blind synchronization in NC-OFDM based cognitive radio networks, in: Proceedings of the 2010 ACM Workshop on Cognitive Radio Networks, 2010, pp. 1–6.
[4] S. Feng, H. Zheng, H. Wang, J. Liu, P. Zhang, Preamble design for non-contiguous spectrum usage in cognitive radio networks, in: IEEE Wireless Communications and Networking Conference (WCNC, 2009, pp. 1–6, doi:10.1109/WCNC.2009.4917916.
[5] R. Rajbanshi, A.M. Wyglinski, G.J. Minden, An efficient implementation of NC-OFDM transceivers for cognitive radios, in: 1st International Conference on Cognitive Radio Oriented Wireless Networks and Communications, 8–10, 2006, pp. 1–5, doi:10.1109/CROWNCOM.2006.363452.
[6] L. Li, D. Qu, T. Jiang, J. Ding, Design of LDPC codes for non-contiguous OFDM-based communication systems, in: IEEE International Conference on Communications (ICC), 10–15, 2012, pp. 4712–4716, doi:10.1109/ICC.2012.6363932.
[7] F. Shamani, R. Airoldi, T. Ahonen, J. Nurmi, FPGA implementation of a flexible synchronizer for cognitive radio applications, in: Conference on Design and Architectures for Signal and Image Processing (DASIP), 8–10, 2014, pp. 1–8, doi:10.1109/DASIP.2014.7115603.
[8] J. Liu, S. Feng, H. Wang, Comb-type pilot aided channel estimation in non-contiguous OFDM systems for cognitive radio, in: 5th International Conference on Wireless Communications, Networking and Mobile Computing (WiCom), 24–26, 2009, pp. 1–4, doi:10.1109/WICOM.2009.5303081.
[9] X. Zhou, R. Qiu, An adaptive synchronization algorithm for non-contiguous OFDM cognitive radio systems, in: IET International Communication Conference on Wireless Mobile and Computing (CCWMC), 14–16, 2011, pp. 102–106, doi:10.1049/cp.2011.0856.
[10] A.M. Wyglinski, Effects of bit allocation on non-contiguous multicarrier-based cognitive radio transceivers, in: IEEE 64th Vehicular Technology Conference, 25–28, 2006, pp. 1–5, doi:10.1109/VTCF.2006.159.
[11] R. Rajbanshi, A.M. Wyglinski, G.J. Minden, An efficient implementation of NC-OFDM transceivers for cognitive radios, in: 1st International Conference on Cognitive Radio Oriented Wireless Networks and Communications, 8–10, 2006, pp. 1–5, doi:10.1109/CROWNCOM.2006.363452.
[12] Z. Yuan, S. Pagadarai, A.M. Wyglinski, Feasibility of NC-OFDM transmission in dynamic spectrum access networks, in: IEEE Military Communications Conference (MILCOM), 18–21, 2009, pp. 1–5, doi:10.1109/MILCOM.2009.5379824.
[13] D. Qu, J. Ding, T. Jiang, X. Sun, Detection of non-contiguous OFDM symbols for cognitive radio systems without out-of-band spectrum synchronization, in: IEEE Transactions on Wireless Communications, volume 10, 2011, pp. 693–701, doi:10.1109/TWC.2011.120810.101324.
[14] J.Y. Won, H.G. Kang, Y.H. Kim, I. Song, M. Song, Fractional bandwidth mode detection and synchronization for OFDM-based cognitive radio systems, in: IEEE Vehicular Technology Conference (VTC), 11–14, 2008, pp. 1599–1603, doi:10.1109/VETECS.2008.371.
[15] B. Huang, J. Wang, W. Tang, S. Li, An effective synchronization scheme for NC-OFDM systems in cognitive radio context, In IEEE International Conference on Wireless Information Technology and Systems (ICWITS), Aug. 28 2010-Sept. 3 2010, pp. 1–4, doi:10.1109/ICWITS.2010.5611980.
[16] D. Saha, A. Dutta, D. Grunwald, D. Sicker, Blind synchronization for NC-OFDM - when “channels” are conventions, not mandates, in: IEEE Symposium on New Frontiers in Dynamic Spectrum Access Networks (DySPAN), 2011, pp. 552–563, doi:10.1109/DYSPAN.2011.5936246.
[17] T. Chiueh, P. Tsai, I. Lai, Baseband Receiver Design for Wireless MIMO-OFDM Communications, 2nd, Wiley-IEEE Press, 2012, p. 346.
[18] White space database administrators guide, [WWW], Federal Communications Commission, [Accessed on 29.10.2013], Available at http://www.fcc.gov/encyclopedia/white-space-database-administrators-guide.
[19] K. Grati, N. Khouja, B.L. Gal, A. Ghazel, Power consumption models for decimation FIR filters in multistandard receivers, in: VLSI Design, Article ID 870546, Hindawi Publishing Corporation, 2012, p. 15, doi:10.1155/2012/870546.
[20] , Embedded Memory (RAM: 1-PORT, RAM:2-PORT, ROM: 1-PORT, and ROM: 2-PORT) User Guide, Altera Corporation, 2014.
[21] R. Airoldi, J. Nurmi, Design of a matched filter for timing synchronization, in: Conference on Design and Architectures for Signal and Image Processing (DASIP), 2013, 2013, pp. 247–251.
[22] R. Mahesh, A.P. Vinod, New reconfigurable architectures for implementing FIR filters with low complexity, in: IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, volume 29, 2010, pp. 275–288, doi:10.1109/ICCCI.2014.6921804.
[23] D. Ghosh, D. Sharma, A. Aziz, A novel low area and high performance programmable FIR filter design using dynamic random access memory, in: 48th Midwest Symposium on Circuits and Systems, volume 2, 2005, pp. 1477–1480, doi:10.1109/MWSCAS.2005.1594392.
[24] S. Pontarelli, P. Reviriego, C.J. Bleakley, J.A. Maestro, Low complexity concurrent error detection for complex multiplication, in: IEEE Transactions on Computers, volume 62, 2013, pp. 1899–1903, doi:10.1109/TC.2012.246.
[25] J.W. Hartwell, A procedure for implementing the fast fourier transform on small computers, in: IBM Journal of Research and Development 15.5, 1971, pp. 355–363, doi:10.1147/rd.155.0355.
[26] A.A. Karatsuba, Y. Ofman, Multiplication of multidigit numbers on automata, Soviet physics-Doklady 7, 1963, pp. 595–596, translated from, Doklady Akademii Nauk SSSR 145 (2) (1962) 293–294.
[27] A.A. Karatsuba, The complexity of computations, in: Proceedings of the Steklov Institute of Mathematics, vol. 211, 1995, pp. 169–183, Translated from Trudy Matematicheskogo Instituta Imeni V. A. Steklova, 211, 1995, pp. 186–202.
[28] M. Faust, C.H. Chang, Optimization of structural adders in fixed coefficient transposed direct form FIR filters, in: IEEE International Symposium on Circuits and Systems, Taipei, 2009, pp. 2185–2188, doi:10.1109/ISCAS.2009.5118230.
[29] R. Nanda, B. Nikolid, Digital filters, in: D. Markovic, R.W. Brodersen (Eds.), DSP Architecture Design Essentials, Springer, New York, 2012, p. 351.
[30] E. Jamro, K. Wiatr, FPGA implementation of addition as a part of the convolution, in: Proceedings Euromicro Symposium on Digital Systems Design, Warsaw, 2001, pp. 458–465, doi:10.1109/DSD.2001.952368.
[31] J. Han, M. Orshansky, Approximate computing: an emerging paradigm for energy-efficient design, in: 18th IEEE European Test Symposium (ETS), Avignon, 2013, pp. 1–6, doi:10.1109/ETS.2013.6569370.

[32] I. Diaz, L. Wilhelmsson, J. Rodrigues, J. Lofgren, T. Olsson, V. Owall, A sign-bit auto-correlation architecture for fractional frequency offset estimation in OFDM, In Proceedings of 2010 IEEE International Symposium on Circuits and Systems (ISCAS), May 30 2010–June 2, 2010, pp. 3765–3768, DOI: 10.1109/ISCAS.2010.5537730.
[33] T.H. Pham, S.A. Fahmy, I.V. McLoughlin, Low-power correlation for IEEE 802.16 OFDM synchronization on FPGA, in: IEEE Transactions on Very Large Scale Integration (VLSI) Systems, volume 21, 2013, pp. 1549–1553, doi:10.1109/TVLSI.2012.2210917.
[34] A. Chandra, S. Chattopadhyay, Design of hardware efficient FIR filter: a review of the state-of-the-art approaches, Available online, Engineering Science and Technology an International Journal, [Accessed on 11.01.2016](2015).
[35] Design planning for partial reconfiguration quartus-II handbook version 13.0, 2013, Altera Corporation, San Jose, CA, 46,
[36] Arithmetic, ch. 2, in: "Advanced Synthesis Cookbook", Altera Corporation, San Jose, CA, July, 2011, pp. 2–13. available at https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/manual/stx_cookbook.pdf.
[37] J. McAllister, FPGA-based DSP, in: S.S. Bhattacharyya, F. Deprettere, R. Leupers, J. Takala (Eds.), Handbook of Signal Processing Systems, 2nd, Springer New York, 2013, pp. 707–739. 1399 p.
[38] Stratix V device handbook, volume1: device interfaces and integration, [WWW], Altera Corporation, San Jose, CA, June 2016, [Accessed on 11.10.2016], available at https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/hb/stratix-v/stx5_core.pdf.
[39] Enabling high-performance DSP applications with stratix v variable-precision DSP blocks, [WWW], white paper, Altera Corporation, San Jose, CA, May 2011, [Accessed on 11.10.2016], available at https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/wp/wp-01131-stxv-dsp-architecture.pdf.
[40] Reducing power consumption and increasing bandwidth on 28-nm FPGAs, [WWW], Altera Corporation, San Jose, CA, March 2012, [Accessed on 19.1.2016], available at https://www.altera.com/en_US/pdfs/literature/wp/wp-01148-stxv-power-consumption.pdf.
[41] Stratix II DSP performance, [WWW], white paper, Altera Corporation, San Jose, CA, [Accessed on 25.07.2016, available at] https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/wp/wpstxiidsp.pdf.
[42] S. Jang, B. Chan, K. Chung, A. Mishchenko, Wiremap: FPGA technology mapping for improved routability, in: Proceedings of the 16th International ACM/SIGDA Symposium on Field Programmable Gate Arrays, New York, NY, USA, 2008, pp. 47–55.
[43] J. Cong, Y. Ding, Flowmap: an optimal technology mapping algorithm for delay optimization in lookup-table based FPGA designs, in: IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, volume 13, 1994, pp. 1–12, doi:10.1109/43.273754.

Farid Shamani was born in Tehran, Iran, in 1983. He received his Bachelor's degree in Computer-Hardware Engineering from Islamic Azad University-Central Tehran Branch (IAUCTB), Iran, in 2007. Shamani received his Master's degree with distinction from the Department of Digital and Computer Electronics from Tampere University of Technology (TUT), Finland, in 2013. In 2013, he joined Professor Jari Nurmi's research group in the Department of Electronics and Communications Engineering at TUT. He is currently a Ph.D. student in the Department of Electronics and Communication Engineering at Tampere University of Technology under the supervision of Prof. Jari Nurmi. His research interests primarily include Processor Design, Computer Architecture, Multicore/Manycore System-on-Chips, Digital Electronic Designs, FPGA and DSP Implementations.

Roberto Airoldi received his M.Sc. degree in Microelectronics in 2009 from University of Bologna (Bologna, Italy). Since 2008 he has been working at Tampere University of Technology (Tampere, Finland), initially as research assistant and then as researcher. Since 2009 he has been carrying out his Ph.D. studies at Tampere University of Technology under the supervision of Prof. Jari Nurmi. Currently Airoldi is the vice-chairman of the IEEE GOLD Finland/Tampere section and the IEEE Student Branch at Tampere University of Technology. His main research interests are Multi-Processor Systems-on-Chip, Software Defined Radio, Cognitive Radio and Wireless communications.

Vida Fakour Sevom was born in Mashhad, Iran, 1989. She received her B.Sc. in Biomedical Engineering majoring Bioelectric at Islamic Azad University of Mashhad, 2011. She received her M.Sc. degree in bioengineering from Tampere University of Technology (TUT), Tampere, Finland, in 2015. She is currently working toward the Ph.D. degree with the Department of Electronics and Communication Engineering, TUT. Her main research interests include computer vision, image and video processing, hardware software co-design and digital electronics design.



Tapani Ahonen is a Senior Research Fellow (Vanhempi tutkija) at Tampere University of Technology (TUT) in Tampere, Finland, where he has held various positions since 2000. Since 2004 he has been co-managing a group of about 30 researchers. He is a part-time Lecturer (Nebenberuflicher Lektor) at Carinthia Tech Institute (CTI) - University of Applied Sciences in Villach, Austria since 2007. In 2009 2010 Ahonen was a Visiting Researcher (Chercheur Invité) at Université Libre de Bruxelles (ULB) in Bruxelles, Belgium. His work is focused on proof-of-concept driven computer systems design with emphasis on many-core processing environments. Ahonen has an M.Sc. in Electrical Engineering and a Ph.D. in Information Technology from TUT. Positions of trust recently held by Dr. Ahonen include technical board member of the EU co-funded project Cutting edge Reconfigurable ICs for Stream Processing (CRISP), Finance Chair of the 2009 IEEE Workshop on Signal Processing Systems (SiPS), editorial board member and Guest Editor of the International Journal of Embedded and Real-Time Communication Systems (IJERTCS), and Program Co-Chair of the 2010 Conference on Design and Architectures for Signal and Image Processing (DASIP), He performs reviews for various international journals and participates in program committees of many high-quality conferences on a regular basis. Ahonen has an extensive international publication record including edited books and journals, written book chapters and journal articles, invited talks in high-quality conferences, as well as full-length papers and paper abstracts in conference proceedings.

Jari Nurmi is a professor of Computer Systems at Tampere University of Technology (TUT). He has held various research, education and management positions at TUT and in the industry since 1987. He got a Ph.D. degree from TUT in 1994. His current research interests include System-on-Chip integration, on-chip communication, embedded and application-specific processor architectures, and circuit implementations of digital communication, positioning and DSP systems. He is leading a group of about 20 researchers. Dr. Nurmi is the general chairman of the annual International Symposium on System-on-Chip (SoC) and its predecessor SoC Seminar in Tampere since 1999, and a board member of SoC, FPL, and NORCHIP conference series. He was also the general chair of FPL 2005 and SiPS 2009 conferences. He was the head of the national TELESOC graduate school 2001 2005. He is the author or co-author of about 250 international papers, editor of Springer book Processor Design: System-onChip Computing for ASICs and FPGAs, co-editor of Kluwer book Interconnect-centric Design for Advanced SoC and NoC, associate editor of the International Journal of Embedded and Real-Time Communication Systems, and has supervised over 100 MSc and Licentiate theses and 12 Doctoral theses. He is a senior member in IEEE Circuits and Systems Society, Computer Society, Signal Processing Society, Solid-State Circuits Society and Communications Society. In 2004, he was one of the recipients of Nokia Educational Award, and the recipient of Tampere Congress Award 2005. He was awarded one of the Academy of Finland Research Fellow grants for 2007 2008.

Publication III

Chapter 10
Synchronization in NC-OFDM-Based Cognitive Radio Platforms

Farid Shamani, Tapani Ahonen, and Jari Nurmi

10.1 Introduction

With advancements in wireless technology, applications demand higher data rates. On the other hand, spectrum scarcity is becoming a major problem due to the increment of the spectrum users [1]. As Fig. 10.1 illustrates, a real measurement taken in downtown Berkeley shows that licensed users (also known as primary users) occupied most of the prime spectrum while do not utilize it efficiently. The Dynamic Spectrum Access (DSA) is a promising solution to cope with spectrum scarcity problem. Conceptually, DSA exploits the white spaces between frequencies occupied by several licensed users. Figure 10.2 depicts how DSA enables the secondary usage of the white spaces within a licensed spectrum without interfering primary user's activities. DSA approach also improves spectrum utilization significantly [6].

With the advent of the Software Defined Radio (SDR) in 1991, transceiver could carry out the entire baseband processing in software. Indeed, the SDR is a platform in which, at least, a portion of the implementation is held in software [24]. A more intelligent and advanced version of the SDR known as Cognitive Radio (CR) which was proposed by Joseph Mitola in 1999 [13]. The CR is a radio platform which is always aware of its environment and can rapidly change its operating parameters by considering new characteristics of the spectrum. The user is not even notified when the CR is changing its corresponding parameters. The CR is an alternative solution to mitigate spectrum scarcity problem by reusing the portion of the spectrum

F. Shamani (✉)
Tampere University of Technology, Korkeakoulunkatu 1, 33720, Tampere, Finland
e-mail: farid.shamani@tut.fi

T. Ahonen • J. Nurmi
Department of Electronics and Communications Engineering, Tampere University of Technology, Tampere, Finland
e-mail: ukaata@gmail.com; jari.nurmi@tut.fi

W. Hussain et al. (eds.), *Computing Platforms for Software-Defined Radio*,
DOI 10.1007/978-3-319-49679-5_10

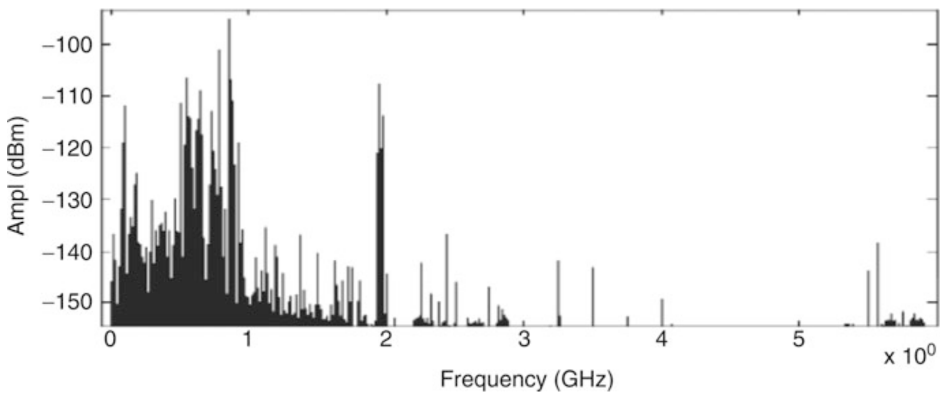


Fig. 10.1 Spectrum utilization snapshot at downtown Berkeley [25]

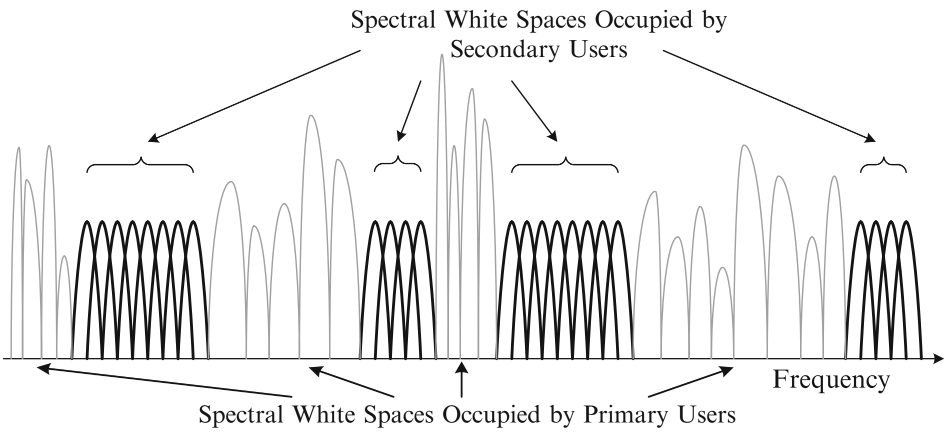

Spectral White Spaces Occupied by Secondary Users
Frequency
Spectral White Spaces Occupied by Primary Users

Fig. 10.2 Spectrum utilization using DSA technology [24, p. 151]

assigned to the licensed user for secondary users without disrupting the operations of any nearby primary user [7]. At first glance, the above-mentioned approaches look simple. However, a simple alteration in frequency-domain arises several challenges in the transceiver of which the synchronization is one of the prominent ones. Briefly, the receiver has no idea that the transmitter sends the data in which frequency band.

10.2 Synchronization Issues in NC-OFDM Systems

Synchronization is an essential issue in all digital communication systems in order to extract the data out of the received signal. Therefore, one of the main challenges for digital design engineers is to establish a reliable, robust, and accurate synchronization. Synchronization is a process in which the receiver detects the existence of an incoming packet and then distinguishes the boundaries of the received packet.

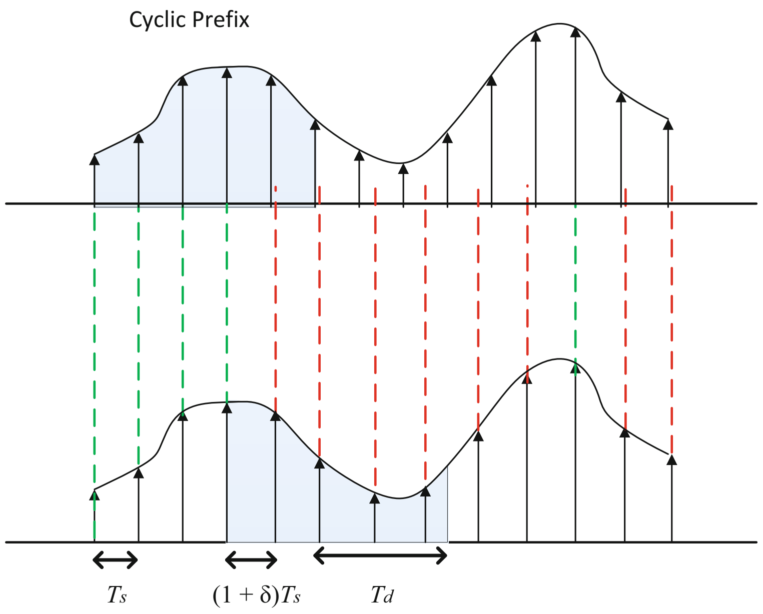


Fig. 10.3 The effect of a bad synchronizer in an ideal channel [20]

Synchronization errors yield in either time, frequency, or both. Conceptually, the receiver must run the sampling process on the incoming wave stream in certain time intervals. Any alteration in sampling time misleads the receiver to properly extract the data. Figure 10.3 illustrates the picture when the receiver is not able to execute the sampling process on time. As it can be seen, sampling at a period of $(1 + \delta)T_s$ causes a δ shift in time which leads to an incorrect sampling.

10.2.1 OFDM Versus NC-OFDM

NC-OFDM is an extension of the conventional OFDM modulation where unused subcarriers are deactivated to eliminate interference made by the secondary user with adjacent licensed users. Therefore, understanding the architecture of an OFDM transceiver is strongly required before stepping forward to NC-OFDM systems. As Fig. 10.4 shows, a high-speed data stream $X(n)$ is demultiplexed into N_u parallel ones $x^{(k)}(n), k = 0, \dots, N_u$. This procedure is usually performed by using a serial-to-parallel converter to form a set of data subcarriers. Then each subcarrier is individually modulated using either QAM or PSK modulation technique to produce $y^{(k)}(n), k = 0, \dots, N_u$ [24]. Typically, as long as the receiver knows the modulation pattern, each subcarrier is modulated with the same constellation [3]. Thereafter, the baseband OFDM waveform $s^{(\ell)}(n), \ell = 0, \dots, N$ is constructed as an N-input IDFT unit with $N \geq N_u$ defined as Eq. (10.1). The remaining $N - N_u$ unused inputs

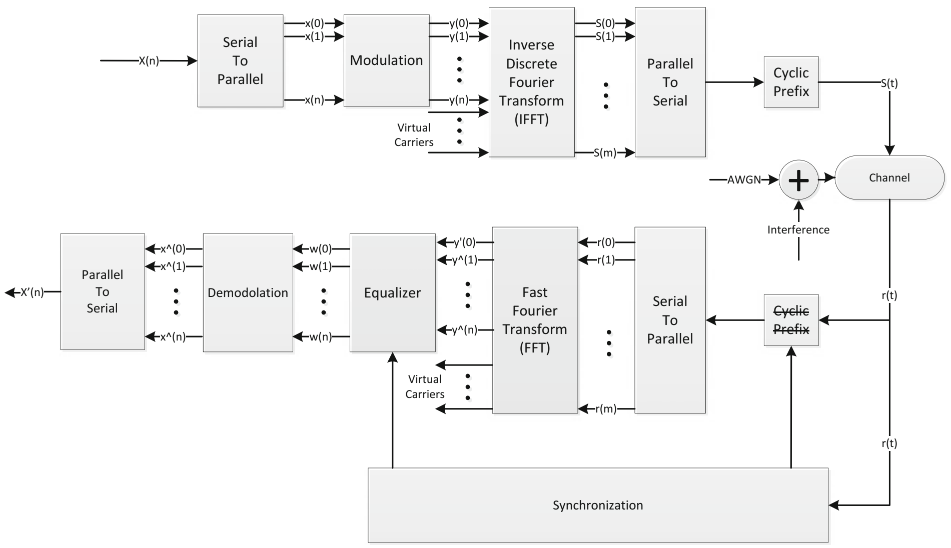


Fig. 10.4 OFDM transceiver architecture [19]

(Also known as VR) are set to zero. VCs are employed as guard bands to cope with the Inter-Symbol Interference (ISI) problem. ISI is imposed to the system due to the interferences caused by transmission power of the adjacent subcarriers. Typically, IDFT is implemented using an IFFT function. Eventually, the transmitter inserts a CP before converting all the subcarriers to the composite signal $s(n)$ [15].

$$s^{(\ell)}(n) = \frac{1}{N}\sum_{k=0}^{N-1} y^{(k)}(n)e^{j2\pi k\ell/N} \tag{10.1}$$

The receiver performs the inverse procedure to the received signal. The first step is to remove the CP from the received signal $r(n)$. Next, considering Eq. (10.2), a serial-to-parallel demultiplexer converts the serial stream to parallel ones. Thereafter, the information is extracted by performing DFT function on the received parallel waveforms $r^{(\ell)}(n)$. DFT is performed using an FFT function to produce $\hat{y}^{(k)}(n)$. Next step is to compensate channel distortion imposed to the subcarriers using an equalizer. The equalized subcarriers $\omega(n)$ are demodulated and, subsequently, the serial steam $x'(n)$ is fetched by employing a parallel-to-serial converter on parallel streams $x^{(k)}(n)$ [15].

$$\hat{y}^{(\ell)}(n) = \sum_{\ell=0}^{N-1} r^{(\ell)}(n)e^{(-j2\pi k\ell/N)} \tag{10.2}$$

Figure 10.5 presents the architecture of an NC-OFDM transceiver. By taking into consideration both Figs. 10.4 and 10.5, the fundamental difference in both architectures is the synchronizer. In contrast to the OFDM where the synchronization was

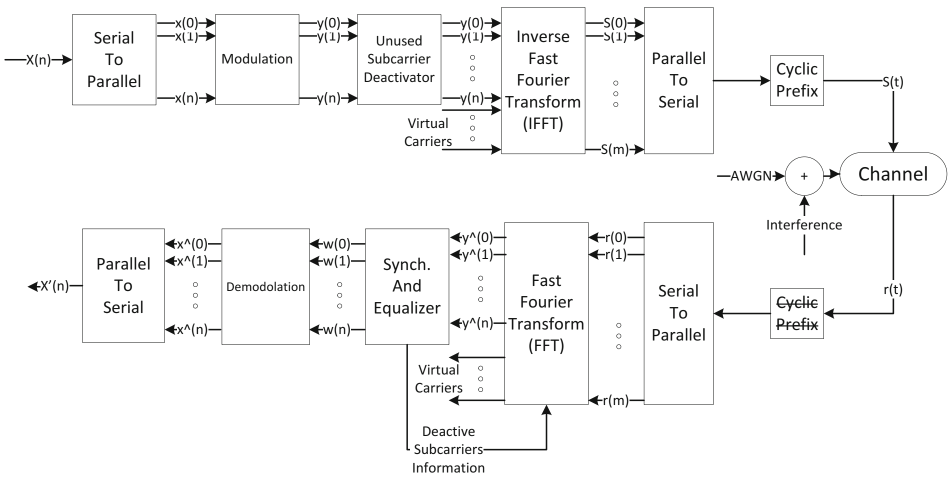


Fig. 10.5 NC-OFDM transceiver architecture [19]

done prior to the Fast Fourier Transform (FFT) (synchronization in time-domain), an NC-OFDM receiver performs the synchronization following the FFT (synchronization in frequency-domain). The main reason which enforces the receiver to perform synchronization in frequency-domain is a simple change in time-domain representation of the signal. This is mainly due to the time-domain stochastic alteration of the preamble, pilot, and data carriers of the secondary user based on primary user activities. In addition, all secondary users must reduce their transmission power to avoid interfering with the licensed user due to the sidelobe leakages caused by the OFDM nature.

10.2.2 *Synchronization Methods*

As previously discussed, contrary to OFDM, in NC-OFDM systems a precise location for signal carriers are not guaranteed. Moreover, the location of the licensed user is changed across the spectrum over the time. Therefore, an NC-OFDM receiver has no prior information about the subcarriers employed by the secondary transmitter. This is one of the key challenges emerged from synchronization point of view. Some researches proposed Out-Of-Band (OOB) architecture to inform the receiver about the new characteristics of the spectrum [11, 17, 23, 26, 27]. In this methodology, a special-purpose pre-dedicated channel is established between the transmitter and receiver to reveal information with regard to the active subchannels using by the secondary user as well as inactive ones. However, OOB methodology seems not to be a suitable solution since the dedicated channel may not be available in some practical situations [16].

Conceptually, in contrast to OOB mechanism, the prerequisite synchronization information of a secondary transmitter is embedded to the incoming packet itself. However, the secondary receiver still has the problem to detect a secondary transmission. Fortunately, the Federal Communication Commision (FCC) provides secondary users willing to transmit over the licensed spectrum with transmission information of the primary user, including the structure of the signal, power, location of the primary user, etc. [21]. Having substantial information about the primary user's activities enables the secondary receiver to filter out subcarriers occupied by the primary user. Following discusses few proposed method with regard to in-band communication in NC-OFDM systems.

In [16], the existence of the primary user as well as the corresponding interferences is discussed. Authors explain an A Posterior Probability (APP) algorithm to discover active subchannels. A threshold-based Hard Decision-based Detection (HDD) scheme is used to detect NC-OFDM symbols as soon as an active subchannel is detected. However, the HDD operates quite confident for the subset of subchannels far away from primary user band while the performance is drastically degraded for subchannels closed to the primary user band. In order to mitigate the performance degradation to some extent, a Soft Decision-based Detection (SDD) is performed to increase accuracy for (noisy) subchannels adjacent to the primary user. However, the existence of the primary user still causes severe synchronization problems. Eventually, the performance of the proposed method is heavily dependent on the primary user activities, power, etc.

In [10], authors claim that the system code rate of the proposed algorithm in [16] is 1/4 when only half of the subcarriers are active. Hence, in order to improve the system code rate, they proposed a Low-Density Parity-Check (LDPC) after performing the APP algorithm. The authors emphasized more on how to create the LDPC code while a perfect spectrum synchronization is taken into assumption. Since the obtained results are based on simulations, in hardware perspective the proposed method requires additional hardware element to decode the received LDPC. In addition, employing more hardware element might increase the energy consumption as well as the silicon area.

In [22], a fractional bandwidth model has been proposed in which a different preamble pattern is designed. The information of the active subbands generated by a Pseudo-Noise (PN) sequence is transmitted over the channel. Therefore, the receiver identifies and detects the active subbands in the frequency-domain. In time-domain, the preamble is represented with two identical halves whose sign bits are inverted. However, the proposed method is based on considering only contiguous subbands (OFDM-based CR system) which is not practical in NC-OFDM systems. Moreover, in reality the NC-OFDM secondary transmitter always keeps its transmission power lower than the licensed user to mitigate interfering with the primary band due to the sidelobe leakages [9].

Saha et al. [6, 18] proposed a blind synchronization method where the receiver is capable of regenerating time-domain preamble of the frequency-domain representation of the same preamble. Furthermore, they considered the idea of employing multiplier-less correlator instead of performing a full 16-bit Multiplier-Accumulate

(MAC) operation in frequency-domain. Furthermore, the primary user's activities are taken into account and, consequently, subchannels occupied by the primary user are filtered using a binary mask. As a result, the existence of the primary user is eliminated right after performing the FFT. Henceforward, the rest of the synchronization process is similar to the one of OFDM.

In OFDM systems, synchronization can be performed either in time-domain, frequency-domain, or both. Since synchronization in frequency-domain requires many cycles to compute, an OFDM receiver prefers to perform synchronization in time-domain [4]. Furthermore, each packet starts with the so-called *preamble*. The preamble is repetitive sequence known for both transmitter and receiver. Due to the repetitive nature of the preamble, time-domain representation of the preamble has a good autocorrelation property. Nevertheless, any change in time-domain representation of the preamble will cause the NC-OFDM receiver to fail detecting a packet. On the other hand, a secondary transmitter has to alter its transmitting frequencies over the time (in the frequency-domain) depending on the licensed user's activities. A simple change in frequency-domain results in a complete change of the time-domain waveform of the preamble. This is the reason why an NC-OFDM receiver has no idea about the time-domain representation of the preamble. Therefore, synchronization based on preamble detection should be performed in frequency-domain in NC-OFDM-based systems. In addition, NC-OFDM receivers are compromised with two main challenges, as well. First, only a portion of the subcarriers are available (the rest are occupied by the licensed user). Second, they should face a low SNR region due to the low transmission power of the secondary user (to mitigate interferences with the licensed user) [9]. The above-mentioned obstacles disable conventional synchronization methods which were feasible in OFDM systems. Figure 10.6 depicts the architecture of an NC-OFDM receiver. By taking into account the block diagram, synchronization is performed in several steps explained in the following [18].

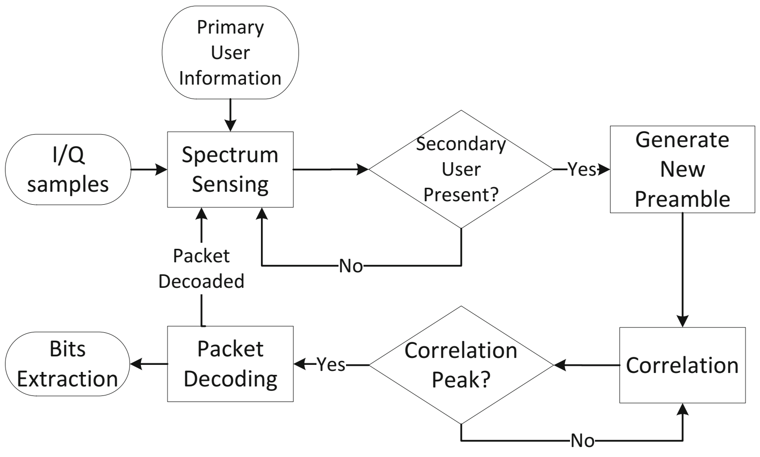


Fig. 10.6 NC-OFDM synchronization scheme [20]

10.2.3 Active Subcarrier Detection

The first step is to filter those subcarriers occupied by the licensed user. Since characteristics of the licensed user are known for the secondary receiver [21], it can be simply done by excluding primary user's subcarriers when collecting all the subcarriers (after the FFT). The hardware circuit of such a filter can be a simple XOR gate. Henceforth, a set of collected subcarriers implies the entire spectrum where the secondary user might have been active. At this stage, all subcarriers are employed for spectrum sensing in a pipeline manner using one of the spectrum sensing methods. The corresponding spectrum sensing results of all subcarriers are reported to a decision maker unit. The decision maker unit compares computed results with an approximate threshold. The threshold can be approximately calculated as follows: when there is no incoming signal or during the inter-packet duration, the output of the FFT is the magnitude of the noise level. The decision maker unit is able to dynamically approximate a threshold by subtracting the noise level from the FFT output. The last step in this phase is to stop spectrum sensing as soon as the decision maker unit detects the results that have risen over the threshold. Note that a copy of the incoming packet is also buffered to extract time-domain samples in the next step.

10.2.4 Preamble Regeneration in Time-Domain

The time-domain representation of the frequency-domain coefficients are generated by employing a low-cost IFFT unit. Once the time-domain preamble is generated, the coefficients of the correlator are re-initialized by new samples. Henceforward, the synchronization mechanism is quite similar to OFDM. The similarity of the new coefficients with the buffered version of the incoming packet (which stored in previous step) is computed to detect the boundary of the signal. As soon as a peak is found, the correlation is inferred to be successful. The last step is to call decoding unit to extract the data out of the received bits.

10.2.5 Failure in Correlation

When the correlation computations do not yield appropriate results, for whatever reason, the spectrum sensing mechanism is started to obtain a new set of subcarriers which is occupied by the secondary transmitter. There are several reasons which might desynchronize the receiver with the transmitter. A low secondary signal SNR is one of the main reasons. This happens due to the fact that a secondary user near to a primary user should minimize its transmission power as low as possible. A high power primary user has the potential to overwhelm any adjacent secondary transmitter. Furthermore, a secondary transmitter might have been using a white space spectrum which belonged to a turned-off primary. When the primary user shows an activity on the white space band, the secondary user should immediately disrupt its transmission. In such circumstances, the receiver must be intelligent and agile enough to start sensing the channel from the beginning.

10.3 Proposed Synchronization Dataflow

In this section, we propose a reconfigurable platform for sensing the spectrum as well as the preamble detection/generation. The term "reconfigurability" mainly refers to the correlator architecture to perform autocorrelation and cross-correlation on demand. In addition, instead of using wide-input multicorrelator (e.g., 4096-tap), we employ several smaller ones in parallel. Figure 10.7 illustrates the block diagram of the proposed receiver architecture. However in order to keep the figure clean, only one multicorrelator is shown in the figure. In this design, a *controller unit* monitors the behavior of the whole system. At the instantiation, multicorrelators are set up by the controller to perform spectrum sensing. Spectrum sensing is done by executing the autocorrelation function of the incoming signal with a delayed version of itself. Each multicorrelator is assigned to compute a particular portion of the spectrum. Computation results obtained by all the multicorrelators are compared to an approximate threshold. As soon as the computation results of a multicorrelator are exceeded over the threshold, the decision maker unit reports the event to the controller. Next, the controller issues the command to *preamble regenerator unit* to generate time-domain representation of the frequency-domain

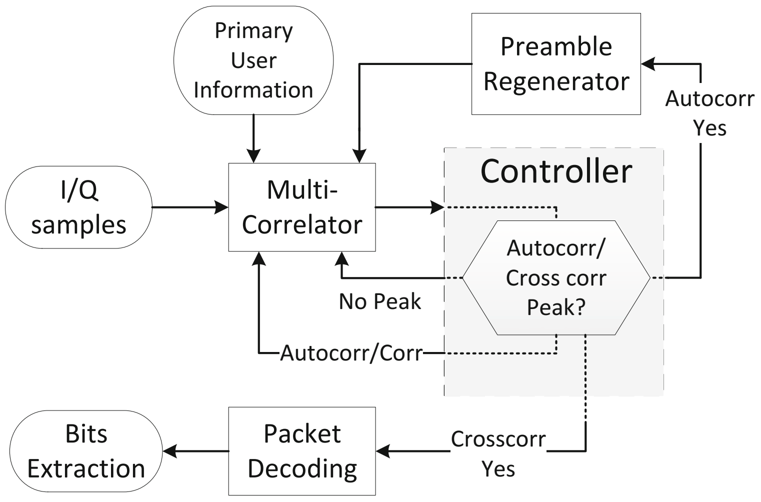


Fig. 10.7 Proposed NC-OFDM synchronization scheme [20]

preamble. In the meantime, the controller sets up the corresponding multicorrelator to compute cross-correlation function while dismisses the other multicorrelators. Once the time-domain version of the preamble is generated, the coefficients of the active multicorrelator are set by the time-domain coefficient of the generated preamble while the buffered version of the In-Phase/Quadrature (I/Q) signals is fed as the input signal. When the multicorrelator is re-initiated, the maximum similarity between the noisy buffered signal and the clean version of the time-domain preamble is computed. Again, the obtained result is compared to a second threshold by the decision maker unit. Once the second peak is detected, the controller infers that the packet is detected and it is ready to be handed to the decoder unit. When the decoder unit extracts the data bits, it reports to the controller to make the proper action.

10.4 FPGA Implementation of the Multicorrelator

Figure 10.8 presents an overview of the proposed NC-OFDM synchronizer. The core content of the multicorrelator is based on a *Memory Block* to store predefined preambles as well as the regenerated preambles, a *MAC* Operator for computing complex Sum-of-Products (SoPs), a *Decision Maker* unit to compare obtained results with a threshold, followed by a *Controller* to monitor the entire system; each block is described in the following in detail.

10.4.1 Memory Block

In this design, the considered memory block is inferred as an SRAM initialized by frequency-domain representation of the preamble. The SRAM is used to store predefined preamble values in frequency-domain representation as well as the regenerated preambles. The SRAM data are mainly used as a sequence of coefficients for cross-correlation purposes. The SRAM has a 32-bit data width output port where the first half (16 least significant bits) includes *Real* part of the preamble and, consecutively, the second half (16 most significant bits) contains the *Imaginary* part. One coefficient is inserted to the FIR structure unit per clock cycle.

10.4.2 Decision Maker Unit

The main objective of the *Decision Maker* unit is to compare results computed by the MAC operator with a threshold. The decision maker employs different threshold for autocorrelation and cross-correlation purposes. When the multicorrelator is set to perform autocorrelation function, the decision maker estimates the noise level

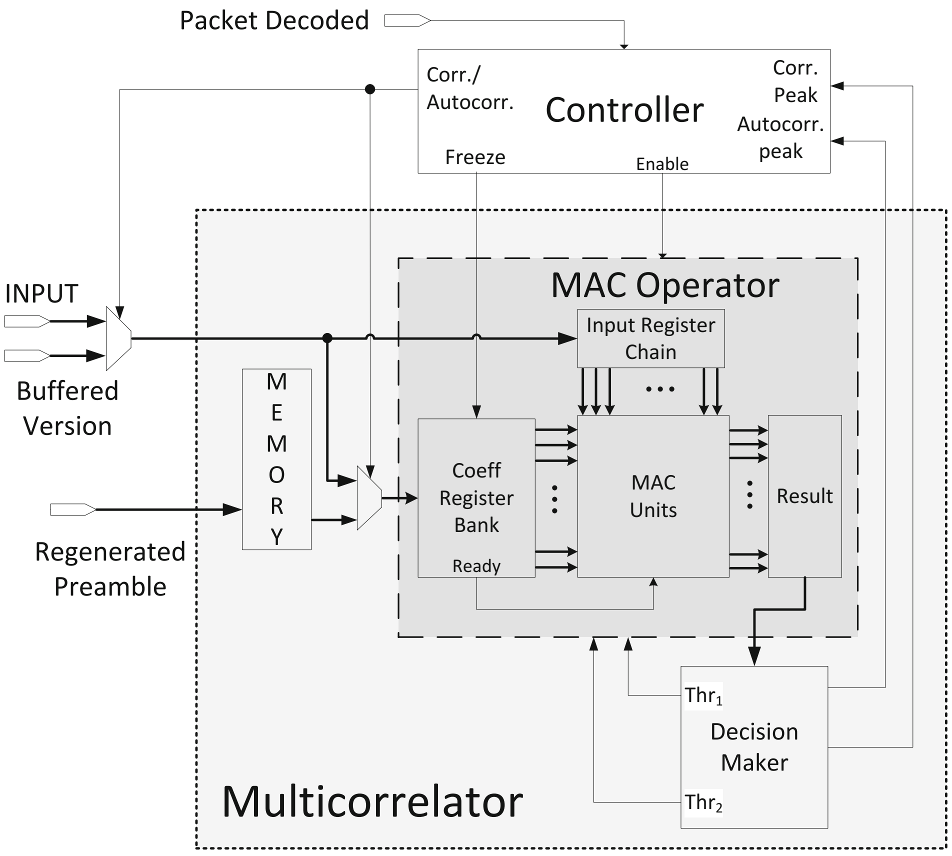


Fig. 10.8 The infrastructure of the proposed NC-OFDM synchronizer

during the silent time and, based on that, an approximate threshold is set. When there is an incoming packet, the noise level is subtracted from the FFT output and the final 16-bit result is compared to the approximate threshold. When the result exceeds the threshold, an autocorrelation peak is found, meaning that the transmitter is sending a packet. In cross-correlation mode, the decision maker unit makes the decision based on a comparison between the results given by the MAC operator with a 2-step threshold level. The main idea of employing 2-step threshold level is explained in [2] in detail. Figure 10.9 shows how the decision maker unit decides whether a peak is found. There are two threshold level: a preliminary threshold Thr_1 alongside an original threshold Thr_2. When the first half of the incoming signal exceeds Thr_1, the decision maker unit issues a command to the MAC operator to compute the rest of the calculation. This is generally made to minimize unnecessary calculations of the MAC operations. Indeed, MAC operations are the most power-hungry operation in this design. If the first half of the MAC operations meets the Thr_1, the second half of the MAC operations are executed. The packet is detected if the MAC operation result exceeds the Thr_2. Otherwise, the decision maker unit infers that the packet

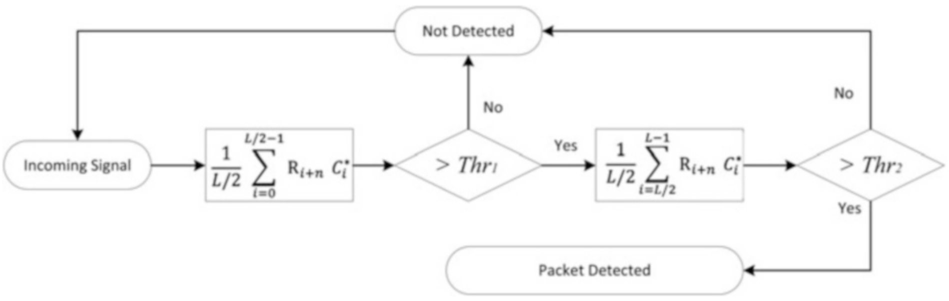

Not Detected
No
No
Incoming Signal
$\frac{1}{L/2}\sum_{i=0}^{L/2-1} R_{i+n} C_i^*$
$> Thr_1$
Yes
$\frac{1}{L/2}\sum_{i=L/2}^{L-1} R_{i+n} C_i^*$
$> Thr_2$
Yes
Packet Detected

Fig. 10.9 2-step threshold detection flowchart [20].

was not detected and, consequently, inform the controller to discard undergoing procedures. The decision maker unit informs the controller with two separate signals each of which indicates a particular peak that is found either in autocorrelation or cross-correlation. Choosing a proper value for both Thr_1 and Thr_2 has an impressive impact on correct packet detection even in low SNR regions. A high-value threshold will improve the algorithm robustness, however, it might result in missing frames with lower energy. On the opposite side, a low value might mislead the decision to detect the noise in lower SNR regions. For instance, a good approximation for Thr_1 value could be 45 % of the original Thr_2 in 2-step algorithm [2].

10.4.3 Controller Block

The controller block is the most responsible block in the design. The main task of this block is to make proper action by continuously monitoring the behavior of the system. When the receiver is waiting for a packet, the synchronizer continuously performs autocorrelation function between the incoming signals with a delayed version of itself. Therefore, the controller sets the MAC operator to perform auto-correlation function. In order to perform the autocorrelation function, the controller routes the incoming signal with the delay length of D to the coefficient register bank through a multiplexer. Whenever the coefficient register bank is appropriately loaded, the controller freezes the register bank to avoid register overloading. Henceforward, the controller waits on the decision maker unit to respond. Once the autocorrelation peak is confirmed by the decision maker, the controller prepares the environment for the multicorrelator to perform cross-correlation function between a buffered version of the packet and regenerated time-domain preambles. The cross-correlation function is executed in the same way as the autocorrelation function. This time, the coefficient register bank is loaded by the regenerated preambles which are stored in the memory. Only one coefficient can be loaded to the register bank per clock cycle. Based on the MAC operator architecture, the multicorrelator performs cross-correlation function at the same time the coefficients are being loaded. The functionality of the MAC operator is explained in next subsection. When the coefficients of the multicorrelator are fed by proper values, the controller freezes the register bank and waits for any prompt from decision maker unit. Once the second peak is confirmed, the controller disables the MAC operator and hands the packet to the decoding unit to extract the data. When the data is extracted, the decoding unit informs the controller to restart the synchronization process.

10.4.4 MAC Operator Unit

Due to the nature of the NC-OFDM, the highest order of the tap, e.g. 4096-tap, is required to perform synchronization over the entire spectrum. Therefore, an NC-OFDM receiver has a massive power dissipation along with a huge silicon usage [12]. The MAC operator unit is quite similar to Finite Impulse Response (FIR) filters from infrastructure point of view. As Eq. (10.3) presents, an FIR filter of order N calculates $y(n)$ as the SoP of coefficients $h(k)$ with the input $x(n-k)$. The hardware implementation of the MAC operator unit also calculates $y(n)$ as the SoP of complex conjugate coefficients $c(k)$ with incoming signal $x(n-k)$ on a window whose length is N [8]. The Conjugation is simply done by inverting the sign bit of the imaginary part of the coefficients. As previously discussed, both coefficient and incoming signal are 32 bits long where we stipulated that the first half is the Real part and the remaining 16 bits are the Imaginary part. It is also worth taking into consideration that the products are complex are complex multiplications which have the potential to double resource usage resource usage from the hardware point of view. Furthermore, we considered fixed-width truncation method where only 16 most significant bits are taken into consideration as the final result of the MAC operation. The MAC operator unit is the most intensive block of the multicorrelator due to the massive workload of complex computations. The infrastructure of the MAC operator is described in the following subsections.

$$y(n) = c(n) * x(n) = \sum_{k=0}^{N-1} c(k)x(n-k) \tag{10.3}$$

10.4.4.1 MAC-Based Operator

In general, the MAC operator is composed of three fundamental functional units aptly named adders, multipliers, and delay elements. An N-tap MAC operator is consisted of N delay elements, N multipliers, as well as $N-1$ adders. Furthermore, different architectures are obtained by different arrangements of these functional units. Technically, there are two well-known architectures for implementing MAC operator known as Direct Form (DF) and Transposed Direct Form (TDF), each of which is capable of calculating $y(n)$. We also experience different structures namely

Parallel Direct Form (PDF) as well as Pipelined-Parallel Direct Form (PPDF). Figure 10.10 pictures the above-mentioned architectures with their pros and cons. Figure 10.10a presents the DF architecture where the delay elements are located through the input path. A large N-tap DF introduces a long critical path due to the nature of its architecture. If we assume that each addition takes T_a and each multiplication takes T_m unit of time and number of taps are equal to 256 (as it is in this implementation), the critical path of the DF architecture is equal to $T_{\text{crit}} = T_m + 255T_a$. Therefore, it is not a wise choice for large MAC operator designs at all.

Figure 10.10b shows the TDF architecture which is the optimized version of the DF form. In this architecture, since the delay elements are located between the adders, the critical path is bounded by $T_{\text{crit}} = T_m + T_a$ irrespective to the number of taps. In the TDF version, the inverse version of the coefficient chain $C_{(k)}$, e.i. $(C_k, C_{k-1}, \ldots, C_0)$, is multiplied with the input signal $X(n)$. Thus, this architecture does not require a register chain in its input path (see Fig. 10.8). However, the TDF version has the potential to introduce a long interconnection on its input path.

Figure 10.10c illustrates the PDF form where at the first glance it looks similar to the DF version. The idea behind the PDF architecture is to cope with long interconnection input as well as the critical path to some extent. Despite the DF form, PDF distributes the adders in several parallel levels (hierarchy levels). The number of required levels is proportional to the $\lceil \log_2^N \rceil$ where N is the number of taps. In this case study, since $N = 256$, 8 levels of adders are required. Consequently, the critical path of a 256-tap MAC operator is equal to $T_{\text{crit}} = T_m + \left(\lceil \log_2^N \rceil \times T_a\right) = T_m + 8T_a$. The hardware implementation procedure of this architecture is more complex than the DF and TDF forms.

Figure 10.10d depicts the pipelined version of the PDF form where a register is inserted between each two adders. Although the inserted registers maintain the critical path at its minimum level ($T_{\text{crit}} = T_m + T_a$) while preventing long interconnections, a variety of register elements are introduced to the system. Moreover, the PPDF architecture has the same complexity as the PDF form.

10.4.4.2 Multiplier-less-Based Operator

A study in [5, 14] shows that the M-L-based multicorrelator design offers comparable synchronization performance. In this approach, a sign bit correlation between coefficients and input signal is considered instead of performing 16-bit multiplication. The sign bit correlation can be implemented in hardware by a simple XOR gate. The M-L form reduces a substantial number of registers as well as the DSP blocks. Although the M-L-based multicorrelator takes only 1-bit into account, simulation results obtained by the ModelSim software show that the performance of the MAC operator is at a satisfactory level in our case study. Furthermore, Saha et al. [18] acknowledged that the sign-based correlator has satisfactory performance even in low SNR regions, as well. Based on these evidences, we can claim that the

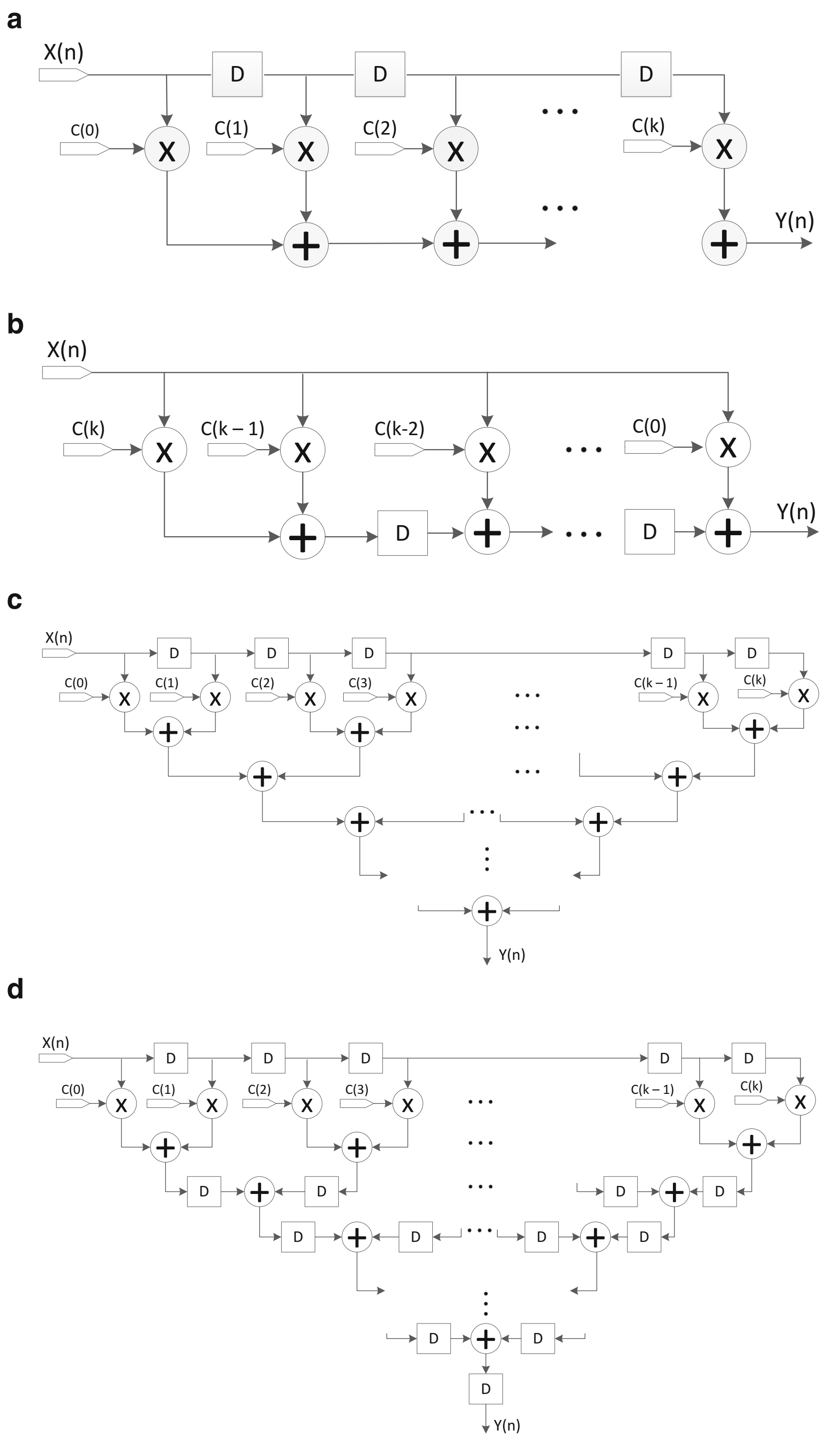


Fig. 10.10 Different architectures for MAC operator unit [20]. (**a**) Direct Form (DF). (**b**) Transposed Direct Form (TDF). (**c**) Parallel Direct Form (PDF). (**d**) Pipelined-Parallel Direct Form (PPDF)

M-L-based multicorrelator is the best choice in our case study. More discussion with regard to the synthesis results of the multicorrelator with different architectures are addressed in Sect. 10.5.

10.5 Synthesis Results

This section discusses with respect to the proposed multicorrelator configured with different architectures. We used VHDL description language to implement the multicorrelator, ModelSim software for simulation, and verification progresses, Quartus-II environment to synthesize the design and generate the netlist. We also took into account the Altera family FPGA targeting Stratix-V speed grade 2 series to prototype the design. Table 10.1 shows synthesis results in terms of logic utilization, DSP blocks, total registers, and maximum frequency. The results show that the M-L-based multicorrelator achieved the best results in most cases whereas the DF can be considered as the worst design approach. As previously explained, the PPDF architecture consumes more resources due to the use of more register elements. The M-L approach drastically reduces logic utilization due to mitigating substantial DSP blocks, registers, etc. In this case study, the M-L-based multicorrelator preserved more than 75 % logic elements as well as 78 % reduction in total registers in comparison with MAC-based architectures. All MAC-based architectures use 512 DSP blocks (32 % of the total) because of the multiplication purposes whereas the M-L-based design does not require any DSP block. The maximum frequency is approximately similar in all architectures, excluding the DF architecture. The M-L-based design achieved the highest frequency between all configurations. Power dissipation analysis is reported based on the results given by Quartus-II PowerPlay Power Analyzer Tool. The analyzer directly reads the SAIF generated by the ModelSim for the multicorrelator running at 50 MHz. The static probability and toggle rate are calculated based on the generated VCD file for detecting a packet

Table 10.1 Synchronizer synthesis results

	TDF	DF	PDF	PPDF	ML
Logic utilization (ALMs)	12,483	12,490	7504	14,611	1876
Total DSP blocks	512	512	512	512	0
Total registers	16,378	16,883	8873	28,037	2886
Max. freq. (MHz)	238	68	88	240	257

Table 10.2 Thermal power dissipation (mW)

	TDF	DF	PDF	PPDF	ML
Total	1436.82	3371.45	1286.90	1138.75	937.15
Dynamic	344.15	200.60	265.90	160.10	9.40
Static	1070.05	1062.70	987.10	955.80	951.50
I/O	22.60	2108.15	33.90	22.85	26.20

Table 10.3 Thermal power dissipation by hierarchy (mW)

	TDF	DF	PDF	PPDF	ML
Memory block	0.78	0.81	1.21	0.80	0.79
Threshold block	0.01	0.01	0.01	0.01	0.01
Controller block	0.02	0.02	0.02	0.02	0.02
FIR filter block	230.20	165.55	192.40	128.80	4.20

Table 10.4 Cell power consumption (mW)

	TDF	DF	PDF	PPDF	ML
DSP block	216.50	130.4	157.75	95.2	0
Combinational cell	76.60	10.80	51.80	1.3	1.85
Register cell	2.10	30.30	10.65	32.65	2.45
I/O	25.1	2090.10	15.80	4.25	7.45

after the boot-up. Table 10.2 summarized the estimated results. The DF architecture has the most power dissipation rather than the other implementations due to the long critical path through its I/O. The dynamic thermal power is noticeably decreased in M-L-based approach. It is generally due to the fact that the M-L architecture mitigates massive number of complex multiplications. Furthermore, the M-L-based design saves more than 94 % of the dynamic power consumption in comparison with the PPDF form. The PPDF architecture is the lowest power-hungry architecture in MAC-based designs.

Dynamic power dissipation by hierarchy is reported in Table 10.3 for different architectures. The MAC operator unit is the most power-hungry block in MAC-based architecture while the controller block as well as the decision maker are the least power consumer, respectively. The power dissipation was reduced due to employing a sign bit correlation instead of 16-bit multiply accumulate operations. Table 10.4 depicts the power dissipation in each cell. The results show that the M-L-based multicorrelator achieves a better result in comparison with MAC-based architectures. The reason why the DF has a massive power dissipation can be explained due to the long critical path which exist in the I/O.

10.6 Conclusion

This chapter discussed about fundamental principles with regard to synchronization issues in Non-Contiguous Orthogonal Frequency Division Multiplexing (NC-OFDM) systems. The architecture of an OFDM transceiver as well as an NC-OFDM transceiver were explored. In contrast to the OFDM system where the synchronization is based on time-domain preamble, an NC-OFDM receiver should regenerate the time-domain representation of the preamble in frequency-domain. Following by, the proposed synchronization data flow as well as the state of the art in synchronizer explained. The main idea was to perform spectrum sensing along with packet detection by a single multicorrelator. The second idea was to segment the supreme spectrum into several subchannels and employ several multicorrelator (instead of searching a wide spectrum). The core content of the synchronizer was based on MAC operations which were quite similar to FIR filters. The MAC operator unit implemented with different structures including Direct Form, Transposed Direct Form, Parallel Direct Form, Pipelined-Parallel Direct Form, Multiplier-Less architectures. The synthesis results inferred that the Multiplier-Less-based multicorrelator had better performance in terms of maximum frequency, silicon area, dynamic power consumption, etc., compared to other configurations. On the other hand, the worst case architecture was Direct Form in all aspects in our case study while there was a trade-off for remaining architectures. As experiences showed, the simplest MAC-based architecture in terms of implementation belonged to Transposed Direct Form and Direct Form, whereas the most complex ones were Pipelined-Parallel Direct Form and Pipelined Direct Form, respectively.

References

1. Acharya, J., Viswanathan, H., Venkatesan, S.: Timing acquisition for non-contiguous OFDM-based dynamic spectrum access. In: 3rd IEEE Symposium on New Frontiers in Dynamic Spectrum Access Networks (DySPAN), Chicago, IL, pp. 1–10 (2008)
2. Airoldi, R., Nurmi, J.: Design of a matched filter for timing synchronization. In: Conference on Design and Architectures for Signal and Image Processing (DASIP), pp. 247–251, 8–10 October 2013
3. Bobrowksi, K.M.: Practical implementation consideration for spectrally agile waveforms in cognitive radio. Ph.D. thesis, Worcester Polytechnic Institute (2009)
4. Chiueh, T., Tsai, P., Lai, I.: Baseband Receiver Design for Wireless MIMO-OFDM Communications, 2nd edn., p. 346. Wiley-IEEE Press, Singapore (2012)
5. Diaz, I., Wilhelmsson, L., Rodrigues, J., Lofgren, J., Olsson, T., Owall, V.: A sign-bit auto-correlation architecture for fractional frequency offset estimation in OFDM. In: Proceedings of IEEE International Symposium on Circuits and Systems (ISCAS), pp. 3765–3768, 30 May–02 June 2010
6. Dutta, A., Saha, D., Grunwald, D., Sicker, D.: Practical implementation of blind synchronization in NC-OFDM based cognitive radio networks. In: Proceedings of the 2010 ACM Workshop on Cognitive Radio Networks, pp. 1–6 (2010)
7. Feng, S., Zheng, H., Haiguang, W.; Jinnan, L., Zhang, P.: Preamble design for non-contiguous spectrum usage in cognitive radio networks. In: IEEE Wireless Communications and Networking Conference (WCNC), pp. 1–6, April 2009
8. Ghosh, D., Sharma, D., Aziz, A.: A novel low area and high performance programmable FIR filter design using dynamic random access memory. In: 48th Midwest Symposium on Circuits and Systems, vol. 2, pp. 1477–1480, 7–10 August 2005
9. Huang, B., Wang, J., Tang, W., Li, S.: An effective synchronization scheme for NC-OFDM systems in cognitive radio context. In: IEEE International Conference on Wireless Information Technology and Systems (ICWITS), pp. 1–4, 28 August–03 September 2010
10. Li, L., Qu, D., Jiang, T., Ding, J.: Design of LDPC codes for non-contiguous OFDM-based communication systems. In: IEEE International Conference on Communications (ICC), pp. 4712–4716, 10–15 June 2012

11. Liu, J., Feng, S., Wang, H.: Comb-type pilot aided channel estimation in non-contiguous OFDM systems for cognitive radio. In: 5th International Conference on Wireless Communications, Networking and Mobile Computing (WiCom), pp. 1–4, 24–26 September 2009
12. Mahesh, R., Vinod, A.P.: New reconfigurable architectures for implementing FIR filters with low complexity. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **29**(2), 275–288 (2010)
13. Mitola, J.: Cognitive radio for flexible mobile multimedia communications. In: IEEE International Workshop on Mobile Multimedia Communications (MoMuC '99), pp. 3–10 (1999)
14. Pham, T.H., Fahmy, S.A., McLoughlin, I.V.: Low-power correlation for IEEE 802.16 OFDM synchronization on FPGA. IEEE Trans. Very Large Scale Integr. (VLSI) Syst. **21**(8), 1549–1553 (2013)
15. Pun, M., Morelli, M., Jay Kuo, C.-C.: Multi-Carrier Techniques for Broadband Wireless Communications. A Signal Processing Perspective, vol. 3, p. 257. Imperial College Press, London
16. Qu, D., Ding, J., Jiang, T., Sun, X.J.: Detection of non-contiguous OFDM symbols for cognitive radio systems without out-of-band spectrum synchronization. IEEE Trans. Wirel. Commun. **10**(2), 693–701 (2011)
17. Rajbanshi, R., Wyglinski, A.M., Minden, G.J.: An efficient implementation of NC-OFDM transceivers for cognitive radios. In: 1st International Conference on Cognitive Radio Oriented Wireless Networks and Communications, pp. 1–5, 8–10 June 2006
18. Saha, D., Dutta, A., Grunwald, D., Sicker, D.: Blind synchronization for NC-OFDM - when "Channels" are conventions, not mandates. In: IEEE Symposium on New Frontiers in Dynamic Spectrum Access Networks (DySPAN), pp. 552–563, 3–6 May 2011
19. Shamani, F.: Design of a flexible timing synchronization scheme for cognitive radio applications. M.Sc. thesis, Tampere University of Technology, p. 75 (2013)
20. Shamani, F., Airoldi, R.; Ahonen, T.; Nurmi, J.: FPGA implementation of a flexible synchronizer for cognitive radio applications. In: Conference on Design and Architectures for Signal and Image Processing (DASIP), Madrid, pp. 1–8 (2014)
21. White Space Database Administrators Guide, [WWW]: Federal Communications Commission (2013). Available at http://www.fcc.gov/encyclopedia/white-space-database-administrators-guide. Accessed on 29 October 2013
22. Won, J.Y., Kang, H.G., Kim, Y.H., Song, I., Song, M.S.: Fractional bandwidth mode detection and synchronization for OFDM-based cognitive radio systems. In: IEEE Vehicular Technology Conference (VTC), pp. 1599–1603, 11–14 May 2008
23. Wyglinski, A.M.: Effects of bit allocation on non-contiguous multicarrier-based cognitive radio transceivers. In: 64th IEEE Vehicular Technology Conference, pp. 1–5, 25–28 September 2006
24. Wyglinski, A.M., Nekoyee, M., Houauthor, T.: Cognitive Radio Communications and Networks, p. 714 . Academic, Oxford (2010)
25. Xing, Y., Kushwaha, H., Subbalakshmi, K.P., Chandramouli, R.: Codes and games for dynamic spectrum access. In: Arsalan, H. (ed.), Cognitive Radio, Software Defined Radio, and Adaptive Wireless Systems, chap. 6, p. 163. Springer, Dordrecht (2007)
26. Zhou, X., Qiu, R.: An adaptive synchronization algorithm for non-contiguous OFDM cognitive radio systems. In: International Communication Conference on Wireless Mobile and Computing (CCWMC), pp. 102–106, 14–16 November 2011
27. Zhou, Y., Pagadarai, S., Wyglinski, A.M.: Feasibility of NC-OFDM transmission in dynamic spectrum access networks. In: IEEE Military Communications Conference (MILCOM), pp. 1–5, 18–21 October 2009

Publication IV

Design, Implementation and Analysis of a Run-Time Configurable Memory Management Unit on FPGA

Farid Shamani, Vida Fakour Sevom, Jari Nurmi, Tapani Ahonen
Department of Electronics and Communications Engineering
Tampere University of Technology
P.O.Box 553, FIN-33101, Tampere, Finland
{firstname.lastname}@tut.fi

***Abstract*—In this paper, we describe the design of a configurable Memory Management Unit (MMU) and its prototype implementation on a Field Programmable Gate Array (FPGA). We present analytical results of scaling the size of the second level software-managed Unified Translation Lookaside Buffer (UTLB) in terms of effect on the overall hit rate. Three design-time configurations with 16, 32, and 64 entries were used for this study. Critical path analysis of the logical design running on Altera Stratix-V FPGA is presented together with a description of optimization techniques employed in order to improve static timing performance. These optimization techniques assist in reaching 22.75% speed-up compared to non-optimized design. Moreover, maximum operating frequencies of 265, 225 and 200 MHz were achieved for UTLB sizes of 16, 32, and 64 entries, respectively. We quote worst case energy consumption figures with random input stimuli together with FPGA resource utilization characteristics for the above mentioned configurations. For resource-constrained or speed-critical hardware designs the 32-entry UTLB configuration provides a decent trade-off while the 16-entry configuration poses unsatisfactory performance. However, our target operating frequency of 200 MHz was eventually reached also for the 64-entry UTLB and hence it is our preferred option for most instantiations.**

Keywords: FPGA Implementation, Memory Management Unit, Virtual-to-Physical Address Translation, Run-Time Configurable MMU.

I. Introduction

In sophisticated embedded systems, main memory access has the potential to cause some extra delays. Even the fastest processors should wait on an access to a slow memory. Hence, speeding up main memory accesses are one of the important concerns of processor designers [1]. Just like caches provide fast access to the active portions of the programs, the main memory can have the potential to act as a cache for Virtual Memory (VM). An investigation in [2] shows that exploiting an L2 cache in parallel with an L1 cache which offers a hit rate of 90%, brings the hit rate up to 95%. Although the hit rate is only improved by a factor of 5% (in two-level cache system), the miss rate is reduced to half in comparison to the single-level cache system (decreased from 10% to 5%). If this fact can be expanded to the VM, instructions will be executed faster.

A. Motivation

One of the main applications of the virtual memory is to share the main memory among multiple programs. The VM allows each program to reference to a particular part of the main memory while avoids the same program to access to the rest of the memory dedicated to other programs. Many years ago, programmers were responsible to manually fit programs which were larger than the main memory storage. Virtual memory eliminated substantial burden on programmers by moving inactive portion of the program out of the physical memory. As soon as other active programs require some space in the main memory, the inactive portion of the running programs are stored in a secondary storage using virtual memory mechanism. Therefore, virtual memory enables multitasking by relocation over the main memory [3].

Conceptually, in virtual mode, the processor produces a virtual address which is convertible to the physical one using a combination of hardware and software approaches. Fig. 1 shows *address translation* procedure. This mechanism is usually performed by the Memory Management Unit (MMU). The primary function of the MMU, also can be denoted as the most important one, is to translate virtual addresses into physical addresses [4]. The MMU should guarantee the protection of each program as well. On the other hand, the MMU is sufficiently rapid to enable all transactions to the main memory [5]. Theoretically, any VM access takes at least twice longer than the main memory access. It requires one memory access to obtain the physical address and a second attempt to get the data. Therefore, by keeping these accesses in a special cache, successive memory access will be executed much faster than the first one. Accordingly, the most frequent page translations are kept in a special cache which is referred to as a Translation Look-aside Buffer (TLB). In virtual mode, the MMU examines its contents to find a matched Virtual Page Number (VPN). When a desired entry is found, the appropriate physical page number is extracted from the corresponding entry and, then, combined with the offset to form the physical

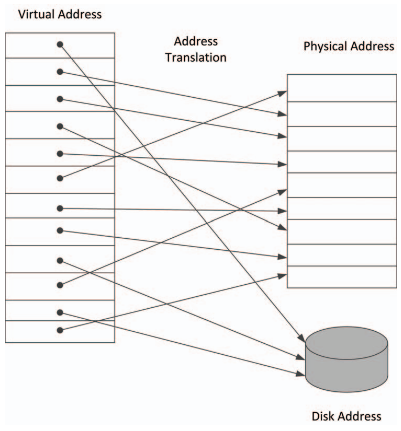


Fig. 1. How virtual memory is mapped to the main memory [3].

address. The processor employs the obtained physical address to reference to a specific location in the memory.

B. Related Works

There are similar works on the same topic implementing an MMU on different FPGA families. In [6], authors proposed a method based on Network Interface Controller (NIC) which includes a bulk memory. A 128MB DRAM memory stores 16 millions of Address Translation Table (ATT) entries (instead of TLB) which include information with regard to virtual-to-physical address translations. Each entry of the ATT is composed of 64 bits. After the first run, the NIC driver maps all the virtual memories to the physical memories. Subsequently, the ATT is maintained only by the driver. However, since DRAMs (along with the CPU) are the major candidates for total system energy consumption [7], the address translation in such systems in an expensive mechanism in terms of energy consumption.

Hu-Cheung Ng et al. implemented an MMU on Xilinx Virtex 5 FPGA board in [8]. The MMU has a TLB which includes virtual to physical address translations. The TLB has 16 entries and employs Least Recently Used (LRU) replacement policy. They also claimed that another small cache which contains 4 entries improved the speed of the memory accesses. The implemented MMU only supports 4kB memory page size. A successful translation takes 2 cycles whereas any update to the TLB takes 16 cycles. A TLB miss penalty is greatly varied between 600 to 227,000 cycles depending on the status of the Operating System (OS).

In [9], a reconfigurable architecture which employs a single 512-entry TLB for address translation mechanism alongside a Direct Memory Access (DMA) unit is proposed. In case of a miss occurrence, the TLB is locked and the FPGA will be interrupted. Consequently, the FPGA will update the TLB with the corresponding missed entry. Thereafter, the TLB is unlocked and the page translation mechanism will be continued. The authors acknowledged that any TLB miss would take thousands of FPGA cycles whereas an address translation with a TLB hit would take 4 cycles. They also claimed that the hardware implementation (on Xilinx Virtex 5 FPGA device) significantly speed-up the performance (approximately five times) over the pure software implementation. However, they have not mentioned the amount of virtual page space supported by the TLB.

Authors in [10] introduced a specialized TLB design which includes a Buffer Search Ram Cache (BSRC) along with a Virtual-to-Physical (V2P) unit where the V2P is mainly responsible for the address translation mechanism. A complete V2P block composed of eight 32-bit entries was integrated by Content Addressable Memories (CAMs) as a specialized memory to act as a hardware search engine. It takes different patterns as search keys and returns the corresponding addresses where the patterns are stored. This architecture, due to the use of tightly coupled RAM structures, returns a translated address in 1 or 2 cycles while deleting or updating a new entry is a complex operation which requires several steps. Analysis results as well as hardware costs are given based on the implementation on Stratix IV FPGA family specification. Of course exploiting a CAM module instead of a simple RAM will result in a very complex implementation (by a factor of 32) which is a very limiting factor in scarce memory resource designs [10].

In this paper, in contrast to the mentioned related works, since the process of address translation is quite similar to the cache mechanism, the implemented MMU employs two levels of TLBs. On the first level, a 4-entry micro TLB for instruction memory references operates alongside an 8-entry micro TLB for data memory references. Both of these micro TLBs are hardware managed. On the second level there is a unified software-managed TLB for both instruction and data page address translations. Run-time configurability allows the MMU to operate on eight different page sizes from 1 KB to 16 MB. The MMU is implemented on Altera Stratix-V series FPGA DSP board [11] based on the specification given in [12] in absolute terms.

The rest of this paper is organized as follows. In Section II an overview to the address translation mechanism by memory management unit is given. In Section III Implementation of the MMU is discussed in detail. Section IV explains analysis results, followed by conclusions in Section V.

II. An Overview to the MMU

Fig. 2 shows how the MMU operates in two different modes known as real and virtual modes. In the real mode, the 32-bit Effective Address (EA), calculated by the processor, is recognized as the physical address which is used to directly access the physical memory (no address translation is required). In virtual mode, the address translation mechanism is more complex compared to the real mode. Fig. 3 illustrates the virtual address translation procedure. The EA obtained from the processor is segmented into two fields of *Effective Page Number* (EPN) and the *offset*. The value N is the boundary

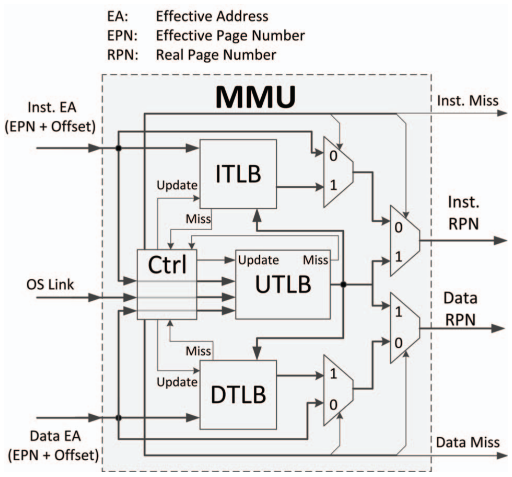


Fig. 2. The MMU behavior with respect to the real mode and virtual mode

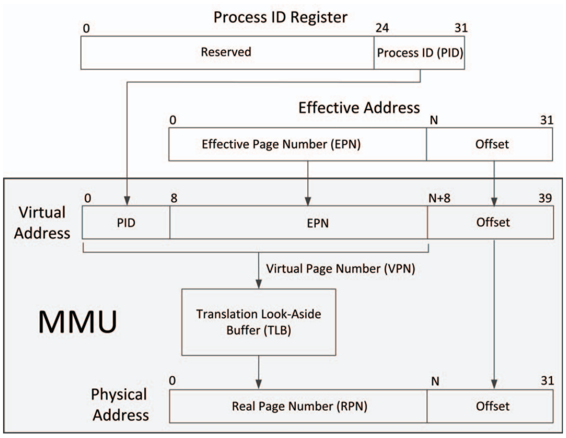


Fig. 3. Address translation procedure in virtual mode

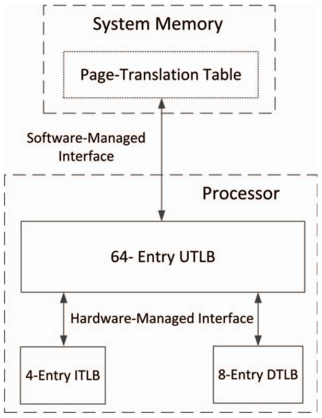


Fig. 4. System memory and processor organization

to distinguish the EPN segment from the offset one. The value of N is calculated based on 3 specific bits within the EPN segment known as *page size* (further information in section III-A). The resulting page size is equal to the $\log_2^n$ where n is the number of bits dedicated to the offset field and, consequently, $N = 31 - n$. For instance, in order to address a 16 KB (2^{14}) page size, $n = 14$ shows that 14 bits are dedicated to the offset field while the remaining 18 bits ($N = 17$) are assigned to the page number field. Virtual Address is a concatenation of the EA (address calculated by the processor) and the Process ID (PID) driven from the PID register. The PID is a unique ID for each process which is used to resolve the overlapped area between different processes in virtual space. Therefore, the PID field along with the EPN segment of the virtual address contain information with regard to the virtual address space. Nevertheless, the MMU employs PID and EPN fields, as the Virtual Page Number (VPN), to extract the Real Page Number (RPN). The concatenation of the RPN and the offset (which has not been a part of the translation process) is the physical address by which the processor references the memory. Moreover, the MMU is capable of mapping eight different fixed page sizes, including 1, 4, 16, 64, 256 KB, 1, 4, and 16 MB, from the virtual space into physical space via a page table. The operating system maintains the page table.

III. FPGA Implementation

In this implementation, we experiment with the effect of using UTLB with different configurations, i.e. 16-, 32- and 64-entry, to cache a subset of instruction and data page translations. However, the 64-entry UTLB is mostly emphasized in this case study. On the other hand, UTLB is completely maintained by the OS. Modifications to the UTLB such as initialization, entry validation and entry replacement (in case of a UTLB miss) are only handled by OS. In addition, all of the entries are accessible by the MMU. The MMU also consists of a 4-entry Instruction TLB (ITLB) along with an 8-entry Data TLB (DTLB) to cache the most frequent instruction and data page translations, respectively. These micro TLBs are managed completely by hardware meaning that the MMU is responsible to initialize, replace and validate the contents of each micro TLB. The basic idea of exploiting micro TLBs is explained in [13] in detail. Micro TLBs are employed to minimize access conflicts with the UTLB as well as speeding up the translation process. Fig. 4 shows how TLBs are organized by the operating system and the processor. The following explains each TLB attribute in brief:

- *Unified TLB*: The UTLB contains 64 entries. It is fully associative, meaning that the instruction and data page translations can be stored in any location. The instantiation and modification of the UTLB are completely

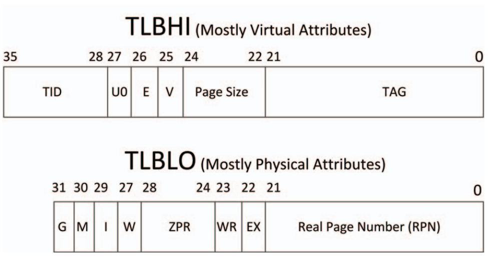


Fig. 5. TLB entry organization

managed by the OS.

- *Micro TLBs*: The ITLB/DTLB contains four/eight instruction/data translation entries. They are fully associative. They store the four/eight most frequently accessed instructions/data from the UTLB. The micro TLBS are used to minimize contention between instruction/data translation and UTLB update operations. The instantiation and modification of the micro TLBs are completely managed by the hardware.

A. *TLB Entry Organization*

When the processor is running in virtual mode, the MMU exploits its TLBs for address translation mechanism. Each TLB entry contains necessary information to identify a virtual page, specify its physical page, and determine the protection as well as the storage characteristics of the page. As Fig. 5 illustrates, each TLB entry consists of 68 bits split into two sections. The first 36 bits (TLBHI) mostly contain useful information about virtual address attributes. Subsequently, the remaining 32 bits (TLBLO) mostly contain information with respect to physical address attributes. The most important bit fields within TLBHI and TLBLO in this implementation are described as following:

- *The most important TLBHI bits in this implementation:*
  - *TAG*: (22 bits) TAG field is compared with the EPN portion of the EA based on the corresponding *page size* field.
  - *Page Size*: (3 bits) Specifies the page size used to define how many bit ranges of the TAG field should be compared with EPN. The size value 0b000 up to 0b111 represents page sizes of 1KB up to 16 MB, consecutively.
  - *V*: Determines whether the current entry is valid.
  - *TID*: (8 bits) This field is compared with the corresponding PID field.
- *The most important TLBLO bits in this implementation:*
  - *RPN*: (22 bits) This field determines the corresponding physical address in case of a TLB hit.
  - *ZPR*: (4 bits) This field selects one of the 16 different zone fields of the Zone Protection Register (ZPR).
  - *G*: Determines whether the current entry is guarded.

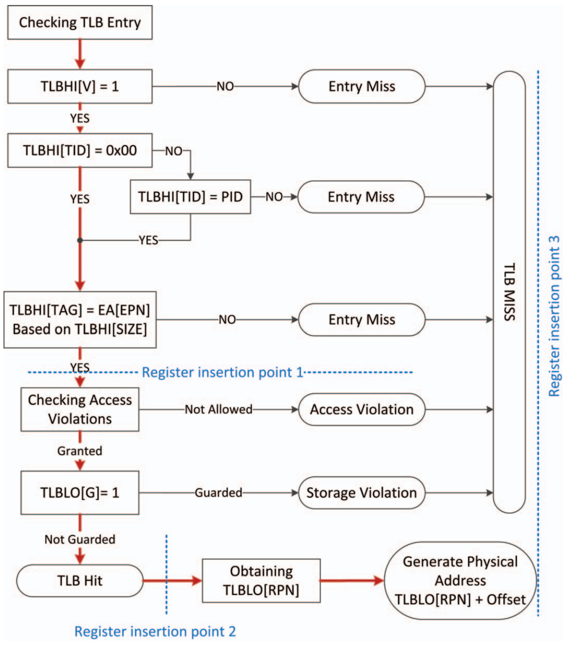


Fig. 6. How each TLB entry responds to a reference. For the UTLB, thicker arrows show the longest critical path propagated through the design. Suitable places for register insertion are represented as well.

B. *TLB Index Examination*

As Fig. 6 illustrates, when examining a page translation entry in either of TLBs, all valid entries in the corresponding TLB are checked simultaneously. A TLB hit will occur if all the following criteria are met:

- The entry is valid.
- The TAG field is identical to EA[EPN] under the control of the page size field.
- The TID field is identical to PID.
- Access to the page is granted and the page is not guarded.

If one of the above conditions is not met, a TLB miss occurs which results in an exception. It is worth mentioning that a TID value of 0x00 represents a process-independent translation from operating system point of view. Therefore, the MMU ignores comparing TID field with the PID. In this case, only the TAG and EA[EPN] are compared. Moreover, A PID value of 0x00 identifies a process which can not access any page. Obviously, the consequence of such a PID is an entry miss.

When a hit occurs, the MMU investigates access control bits of the corresponding TLB entry prior to accessing physical address. Similarly, the MMU inspects the storage attribute fields as well to determine whether the access to the page is allowed. Eventually, if all the criteria are met, a TLB hit will take place.

Fig. 6 shows how each TLB entry is verified during the address translation process. It is promising that a hit will occur

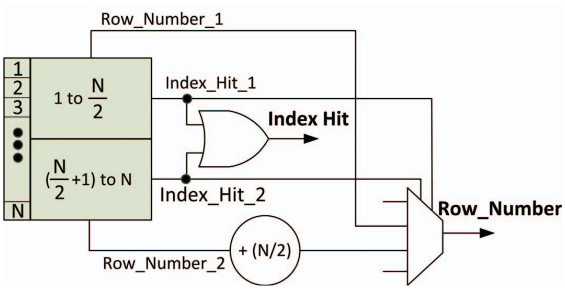


Fig. 7. How to break large loops into two smaller ones

when the incoming EPN passes five consecutive examination stages. Subsequently, a TLB miss is occurred when at least one of the following criteria is met:

- A matching TLB entry is not found.
- A matching TLB entry is found, but the contents are not valid.
- A matching TLB entry is found, but the access to the page is denied.
- A matching TLB entry is found, but the page is guarded.

In the presence of a UTLB miss (irrespective of the miss type), a failure is detected by the MMU and, consequently, an interrupt is issued which reports the cause to the *interrupt handler*. Thereafter, the interrupt handler mechanism is executed to resolve the corresponding problem. In the meantime, the processor enters in the real mode by clearing both IR and DR bits in the Machine State Register (MSR). When the necessary updates have been loaded into the UTLB, the processor will turn back to operate in virtual mode. Therefore, during the interrupt handling process, the MMU is disabled until the UTLB is reloaded with proper entries. Any update to an entry in the UTLB forces the MMU to invalidate all entries in both micro TLBs. Instruction and data consistency between micro TLBs and UTLB will be guaranteed in such a way. Furthermore, Fig. 6 presents the longest critical path propagated through the design with thicker arrows. After monitoring and analyzing the design, the longest critical path belongs to the UTLB due to the relatively large amount of loop iterations. In order to shorten the long critical path as much as possible, we took advantage of two well-known optimization techniques: *register insertion* and *large loops mitigation*. Fig. 6 also illustrates the most suitable locations for register insertion. These points are discovered after performing timing analysis intently.

Fig. 7 depicts how searching through the entire N-entry TLB can be split into several smaller loops (in this figure two smaller ones). For instance, in order to search an index through the 64-entry UTLB, a large 64-iteration loop is divided into two 32-iteration ones. In case of an entry hit, each loop provides useful information relative to whether a matched entry is found in the corresponding loop by setting *Index_Hit_1/2* flag as well as the respective matched entry row number (*Row_Number_1/2*). If the VPN address to the UTLB is found at the 20th entry (location is within the first loop), the Index_Hit_1 flag is set and consequently Row_Number_1 = 20 is assigned to the Row_Number value. If the desired VPN address is found at the 50th entry (second loop), the Index_Hit_2 is set and consequently Row_Number_2 = 18 + 32 (=50) is assigned to the Row_Number value. The effect of using such optimization techniques in order to minimize the critical path will be discussed in Section IV.

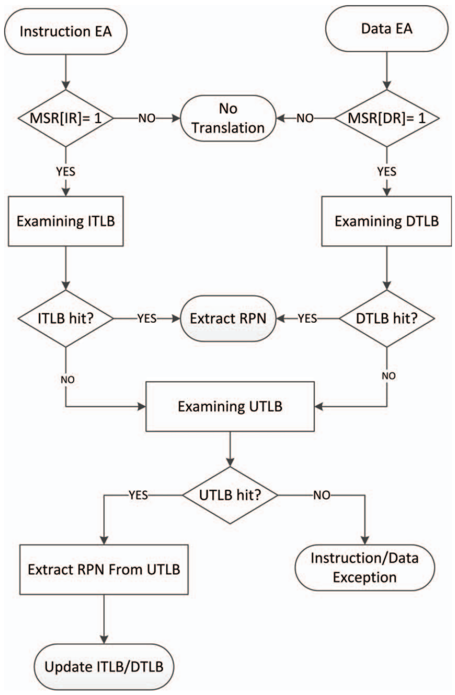


Fig. 8. MMU address translation flow

C. MMU Address Translation Flow

Fig. 8 shows the address translation flow through TLBs. When the virtual mode is activated for either instruction or data side, the MMU translates a combination of PID and EA into the physical address. Thus, it first examines the appropriate micro TLB to find a matched VPN. If a valid entry is found, it is used to access the physical memory. A micro TLB hit will take place in two cycles. If no entry is found, the MMU examines the UTLB for the corresponding virtual address. Additional delays are presented each time the UTLB is accessed due to a micro TLB miss. If both micro TLBs experience simultaneous misses, the DTLB has priority over the ITLB to access the UTLB. In such a situation, the UTLB will first address the data side by searching all the entries for a matched data page translation and then evaluates the instruction EPN. Therefore, in this implementation, the

TABLE I
MINIMUM AND MAXIMUM CYCLE COUNTS FOR EACH VIRTUAL-TO-PHYSICAL ADDRESS TRANSLATION IN AN IDEAL CASE

	Data	Instruction
Micro TLB hit	2	2
Micro TLB miss, UTLB hit	5	5-7
Micro TLB miss, UTLB miss	11	11-13

DTLB miss penalty is 5 cycles while the miss penalty for ITLB varies between 5 to 7 cycles (7 cycles in case of a simultaneous miss with DTLB). Thereafter, the physical address is fetched from the UTLB. In the meantime, the appropriate micro TLB entry is updated with the new entry as well. As earlier stated, in case of a UTLB miss an exception is reported to the operating system via the interrupt handler. How each exception is typically addressed depends on the operating system status, it will take six clock cycles to be resolved in an ideal case. Table I summarizes total cycle counts for each virtual-to-physical address translation in an ideal case.

The Least Recently Used (LRU) replacement policy is considered for both micro TLBs. Each micro TLB employs a specific counter which counts each reference to the TLB. In case of a TLB hit, the LRU counter of the corresponding entry is reset while the remaining LRU counters are advanced by one. When a TLB miss occurs, the entry with the highest LRU counter number is the best candidate to be replaced by the new entry fetched from the UTLB. Lifetime of each LRU counter within the TLB is equivalent to 2 $\times$ number of entries assigned to the same TLB minus one. For example, DTLB (in this implementation) has eight entries; hence, the maximum number achieved by the LRU counter is $2 \times 8 - 1 = 15$. The entries with the highest LRU counter have not been referenced for a while. Thus, they are assumed to be good candidates for replacement. In order to keep micro TLBs coherent with the UTLB, the controller (see Fig 2) will immediately invalidate the contents of both micro TLBs. Invalidation is simply performed by clearing the valid bit (TLBHI[25] in Fig. 5) of all entries right after encountering the UTLB miss. The UTLB miss will disable MMU address translation flow as well. As soon as the operating system completes updating the UTLB, MMU is unlocked and virtual address translation flow can be restarted if needed.

IV. SYNTHESIS RESULTS

The described MMU was written in Very-high-speed integrated circuit Hardware Description Language (VHDL), simulated using ModelSim software, compiled and synthesized by Quartus-II version 13.1.1 environment and finally implemented on Altera family targeting Stratix-V GS mainstream speed grade 2 FPGA device. Table II summarizes the effect of a long critical path before and after the optimization techniques discussed in Section III-B when the UTLB is configured for 64 entries. The longest critical path (6.164 ns) is recorded when no optimization technique is performed. The optimization Level_1 presents new results when the register

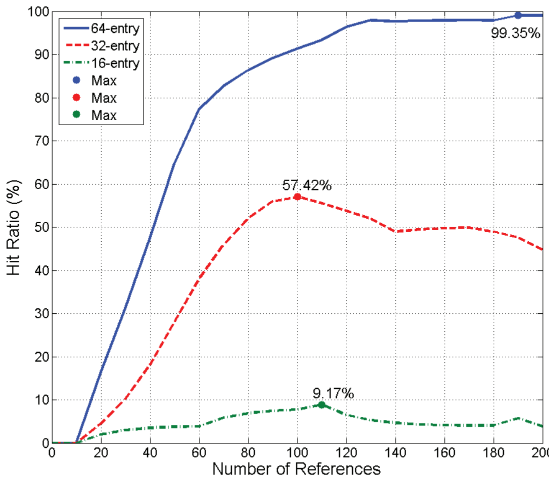


Fig. 9. The hit rate with different number of entries in the UTLB for 200 random references after the boot-up. The curves are smoothed version of the original ones for intervals of 10 references.

insertion strategy is considered. Although inserting registers in the middle of entry verification process has the potential to introduce additional delay, the critical path was reduced more than 1 ns (5.154 ns). In optimization Level_2, breaking a large loop into smaller ones technique is used in addition to register insertion strategy. However, splitting a large loop into two smaller ones (in this case study, 64 iteration to 2$\times$32 ones) has a negligible impact on shortening the critical path in comparison with register insertion technique.

TABLE II
CRITICAL PATH ANALYSIS. LEVEL_1 INDICATES THE REGISTER INSERTION STRATEGY AND LEVEL_2 REPRESENTS LOOP SPLITTING TECHNIQUE

	No Optimization	Level_1	Level_2
Worst Critical Path (ns)	6.164	5.154	4.935
Clock Skew (ns)	-0.054	-0.130	-0.020
Maximum Freq. (MHz)	163.59	193.09	200.80
Speed up	—	18.03%	22.75%

Fig. 9 illustrates the UTLB hit rate for the first 200 random accesses after the boot-up when the UTLB is configured with 64, 32 and 16 entries, respectively. Random accesses are considered to enable the worst-case scenario for the implemented MMU. The random number generator is quite similar to the specification given in [14]. It is obvious that the first indexes to the UTLB, regardless of how many entries it has, should result in misses since the UTLB is empty (compulsory misses). All curves are smoothed each 10 references to the corresponding TLB to present a more visible shot in the figure. The curves show that the 64-entry UTLB has the best hit rate (up to 99%) while 32-entry UTLB has the hit rate of maximum 57%. The hit rate is degraded due to the variety of capacity misses

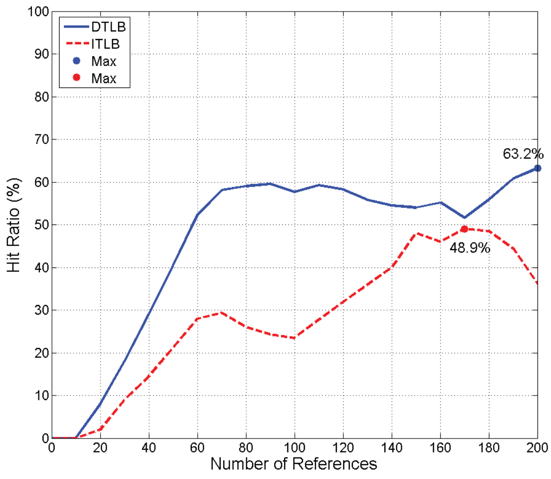


Fig. 10. The DTLB/ITLB hit rate with 64-entry UTLB for 200 random data/instruction references after the boot-up. The curves are smoothed version of the original ones for intervals of 10 references.

TABLE III
SUMMARY OF THE IMPLEMENTATION ANALYSIS

	Memory Management Unit		
	UTLB_64	UTLB_32	UTLB_16
Logic Utilization (ALMs)	3,459	2,092	1,373
Total Registers	5,263	3,221	2,213
Maximum Freq. (MHz)	200.80	228.57	267.38

occurring in 32-entry UTLB. Capacity misses are drastically affected by the design, when the UTLB is configured for 16 entries. A 16-entry UTLB does not respond to more than 9% of the references. The 16-entry UTLB seems to be useless in this case. Since our target is to maximize the MMU hit rate, 64-entry UTLB is the first choice of the interest.

Fig. 10 shows how DTLB and ITLB response to the first 200 corresponding references after the boot-up. Although micro TLB hit rates are not ideal, it is worth mentioning to recall that the investigation is based on random instruction/data references to the micro TLBs to make the worst-case happen. The smoothed curves (accumulating hits for intervals of 10 random references) illustrate both compulsory and capacity misses occurring in ITLB as well as DTLB. Nevertheless, it seems that the MMU takes advantage of micro TLBs to some extent.

In this implementation, we also exploit using UTLB with different number of entries including 16, 32 and 64 entries. Analysis results are reported in Table III in terms of logic utilization, total registers and maximum operating frequency for the described MMU composed of two micro TLBs operating in parallel with a 16-/32-/64-entry UTLB. The maximum operating frequencies of approximately 200, 225 and 265 MHz are achieved with different UTLB configurations. As the rule of thumb of all caches, the larger the UTLB, the slower operating frequency. Furthermore, 64-entry UTLB employs more resources and registers for its implementation. In our case study, 64-entry UTLB shows satisfactory results while 16-entry UTLB is a kind of wasting resources in total. There is a trade-off between using UTLB with either 32 or 64 entries. If, for example, there is no restriction on resources usage, 64-entry is the best candidate to be implemented for UTLB for the sake of higher hit rate.

TABLE IV
ENERGY CONSUMPTION ANALYSIS FOR THE FIRST 200 REFERENCES (μJ)

Thermal Energy	**Memory Management Unit**		
	UTLB_64	UTLB_32	UTLB_16
Static	14291.10	14278.20	14275.95
Dynamic	326.25	167.85	145.65
I/O	431.55	427.20	454.80
Total	15048.90	14873.25	14876.25

Energy consumption analysis is reported based on the results achieved by PowerPlay Power Analyzer tool in Quartus-II software. The analyzer directly reads the Switching Activity Interchange Format (SAIF) generated by ModelSim for the MMU design running at 50 MHz. Static probability and toggle rate for each signal are calculated based on generated VCD file for the first 200 random address translations after the boot-up which can be considered as the worst-case scenario. Random numbers force most of the transistors to toggle per clock cycle. Table IV summarizes the estimation results based on different UTLB configurations. Based on the energy consumption analysis report, dynamic energy is drastically increased while the MMU is configured with 64-entry UTLB due to triggering more transistors. The trade-off between configuring UTLB with 32 entries or 64 entries is still established here. It seems that a 32-entry UTLB might be a good candidate in low-power designs where the hit rate can be sacrificed. The 16-entry UTLB is not recommended from the energy consumption point of view as well.

Dynamic energy consumption for each cell regarding to the MMU block and its sub-blocks are reported in Table V. Statistics show that the MMU itself consumes most of the energy (273, 229.5 and 244.8μJ with respect to 64-, 32- and 16-entry UTLB) to manage the address translation flow for the first 200 references. Followed by, UTLB, DTLB and ITLB feature the highest energy consumptions, respectively. However, DTLB, ITLB and controller blocks consume approximately the same energy regardless of the UTLB configuration. Moreover, the MMU can take advantage of exploiting micro TLBs to speed up the translation mechanism by consuming a trivial amount of energy. Choosing a proper configuration for UTLB has an impressive impact on dynamic as well as routing energy consumption. The 64-entry UTLB consumes a large amount of energy in comparison with other configurations.

V. CONCLUSION

In this paper, an FPGA implementation of a run-time configurable Memory Management Unit (MMU) was described. This

TABLE V
ENERGY CONSUMPTION OF EACH CELL FOR THE FIRST 200 REFERENCES (μJ)

	Dynamic Energy			**Static Energy**			**Routing Energy**			**Design Total Energy**		
UTLB Entries	**64**	**32**	**16**	**64**	**32**	**16**	**64**	**32**	**16**	**64**	**32**	**16**
MMU	78.45	73.95	100.65	73.65	73.65	73.65	120.90	81.90	70.50	273.00	229.50	244.80
ITLB	7.80	7.95	7.50	0	0	0	4.65	4.20	4.65	12.45	12.15	12.15
DTLB	13.95	14.55	15.40	0	0	0	8.10	8.85	9.15	22.05	23.40	24.55
UTLB	**81.15**	29.10	18.45	0	0	0	**75.30**	11.10	11.85	**156.45**	40.20	30.30
Controller	0.60	0.75	0.75	0	0	0	9.00	4.95	3.45	9.60	5.70	4.20
Total Energy	181.95	126.30	142.75	73.65	73.65	73.65	217.95	111.00	99.60	**473.55**	**310.95**	**316.00**

MMU employs two levels of Translation Look-aside Buffers (TLBs). On the first level, a 4-entry micro TLB operates alongside an 8-entry micro TLB to keep track of the most recently used instruction and data address translations, respectively. Both of these micro TLBs are hardware managed. A software-managed Unified TLB (UTLB) exists on the second level to store both instruction and data address translations. The MMU supports eight different page sizes from 1 KB to 16 MB in virtual mode. Micro TLBs were observed to speed up address translation with a low energy overhead. Three UTLB configurations of 16, 32, and 64 entries were studied with respect to overall hit rate, operating speed, energy usage, and resource utilization. The critical path of the logic design was optimized by introducing explicit concurrency as well as retiming through register-to-register logic balancing. Maximum operating frequencies of 265, 225 and 200 MHz were achieved for the 16-, 32-, and 64-entry UTLB configurations, respectively. Using worst-case, i.e., random stimuli, the 64-entry UTLB provides the best worst-case hit rate of up to 99%. The 32-entry UTLB cuts down on resource utilization and energy usage of the MMU design. However, with worst case hit rate of up to 57% only, system-wide energy utilization and performance will be sacrificed by the frequent misses. While a 16-entry UTLB could operate at a somewhat higher speed, its worst case hit rate of up to 9% translates into system wide drawbacks on performance and energy usage. We met our target frequency of 200MHz on Altera Stratix V family of devices for all the studied configurations, making the 64-entry UTLB our preferred choice, while a 32-entry configuration remains a viable option for resource constrained systems.

FUTURE WORK

The MMU is currently being integrated with the COFFEE RISC processor developed at Tampere University of Technology [15]. In order to better serve multi-core systems, we will further modify and expand the MMU for sharing over a Network-on-Chip (NoC). In this work we will also adapt the design to take advantage of the extreme bandwidth obtainable through a memory on top of logic organization of 3D Stacked Integrated Circuits (3DSIC). This configuration is often referred to as 2.5D and is the most likely candidate for successful mass production in the near future, while prototype chips have been available for some time now.

ACKNOWLEDGMENT

The research leading to these results has received funding from the ARTEMIS JU under grant agreement number 295371. Portions of the work were also funded by the Academy of Finland under contract # 258506 (DEFT: Design of a Highly-parallel Heterogeneous MP-SoC Architecture for Future Wireless Technologies)

REFERENCES

[1] K. Hwang, N. Jotwani, ”Advanced Computer Architecture, 2nd Edition, Teta McGraw Hill, 2000, 725 p.
[2] J. Handy, ”The Cache Memory Book”, 2nd Edition, Academic Press Inc., 1998, United Kingdom, 229 p.
[3] D.A. Patterson; J.L. Hennessy, ”Computer Organization and Design”, 4th edition, Morgan Kaufmann Publishers, 2009, 703 p.
[4] B. Cohen; R. McGarity, ”The Design And Implementation of the MC68851 Paged Memory Management Unit”, in IEEE Micro, vol.6, no.2, pp.13,28, April 1986
[5] G. Stefan; F. Draghici, ”Memory management unit-a new principle for LRU implementation”, in 6th Proceedings on Mediterranean Electrotechnical Conference, vol.2, pp.989-992, 1991.
[6] Wang Yongqing; Zhang Minxuan, ”Fully memory based address translation in user-level network interface”, in 3rd International Conference on Communication Software and Networks (ICCSN), pp.351-355, 2011.
[7] D. Schmidt; N. Wehn, ”DRAM Power Management and Energy Consumption: a Critical Assessment”, in Proceedings of the 22nd Annual Symposium on Integrated Circuits and System Design: Chip on the Dunes (SBCCI), pp.32, 2009.
[8] Ho-Cheung Ng; Yuk-Ming Choi; So, H.K.-H., ”Direct virtual memory access from FPGA for high-productivity heterogeneous computing”, in International Conference on Field-Programmable Technology (FPT), pp.458-461, 2013.
[9] Brandon, A.; Sourdis, I.; Gaydadjiev, G.N., ”General Purpose Computing with Reconfigurable Acceleration”, in International Conference on Field Programmable Logic and Applications (FPL), pp.588-591, 2010.
[10] R. Ammendola; A. Biagioni; O. Frezza,; F. L. Cicero; A. Lonardo; P. S. Paolucci; D. Rossetti; F. Simula; L. Tosoratto; P. Vicini, ”Virtual-to-Physical address translation for an FPGA-based interconnect with host and GPU remote DMA capabilities”, in International Conference on Field-Programmable Technology (FPT), pp.58-65, 2013.
[11] ”Stratix V Device Handbook Volume 1: Device Interfaces and Integration”, [WWW], Altera, [Accessed on: 25.Nov.2014], Available at http://www.altera.com/literature/hb/stratix-v/stratix5_handbook.pdf, July 2014.
[12] ”PowerPC Processor Reference Guide Embedded Development Kit”, EDK 6.1, September 2003, Xilinx, 570 p.
[13] J. Ball, ”Designing Soft-Core Processors for FPGAs” in: J. Nurmi, ”Processor Design: System-On-Chip Computing for ASICs and FPGAs”, 1st ed., Springer Publishing Company, pp. 229-256, 2007, 513 p.
[14] W. A. S. Wijesinghe; M. K. Jayananda; D. U. J. Sonnadara, ”Hardware Implementation of Random Number Generators”, in Proceedings of the Technical Sessions, vol. 22, pp. 28-38, 2006.
[15] J. Kylliäinen; J. Nurmi; M. Kuulusa, ”COFFEE - a core for free”, in Proceedings of International Symposium on System-on-Chip, pp.17-22, 19-21 Nov. 2003.

Publication V

Contents lists available at ScienceDirect

Microprocessors and Microsystems

journal homepage: www.elsevier.com/locate/micpro

Integration issues of a run-time configurable memory management unit to a RISC processor on FPGA

Farid Shamani*, Vida Fakour Sevom, Tapani Ahonen, Jari Nurmi

Department of Electronics and Communications Engineering, Tampere University of Technology, Tampere, P.O. Box 553, FIN-33101, Finland

ARTICLE INFO

Article history:
Received 13 April 2016
Revised 16 September 2016
Accepted 2 December 2016
Available online 5 December 2016

Keywords:
FPGA implementation
Memory management unit
Virtual-to-physical address translation
Run-time configurable MMU
RISC processor

ABSTRACT

This paper presents the integration issues of a proposed run-time configurable Memory Management Unit (MMU) to the COFFEE processor developed by our group at Tampere University of Technology. The MMU consists of three Translation Lookaside Buffers (TLBs) in two levels of hierarchy. The MMU and its respective integration to the processor is prototyped on a Field Programmable Gate Array (FPGA) device. Furthermore, analytical results of scaling the second-level Unified TLB (UTLB) to three configurations (with 16, 32, and 64 entries) with respect to the effect on overall hit rate as well as the energy consumption are shown. The critical path analysis of the logical design running on the target FPGA is presented together with a description of optimization techniques to improve static timing performance which leads to gain 22.75% speed-up. We could reach to our target operating frequency of 200 MHz for the 64-entry UTLB and, thus, it is our preferred option. The 32-entry UTLB configuration provides a decent trade-off for resource-constrained or speed-critical hardware designs while the 16-entry configuration poses unsatisfactory performance. Next, integration challenges and how to resolve each of them (such as employing a wrapper around the MMU, modifying the hardware description of the COFFEE core, etc.) are investigated in detail. This paper not only provides invaluable information with regard to the implementation and integration phases of the MMU to a RISC processor, it opens a new horizon to our processor to provide virtual memory for its running operating system without degrading the operating frequency. This work also tends toward being a general reference for future integration to the COFFEE core as well as other similar processor architectures.

1. Introduction

In sophisticated embedded systems, main memory access has the potential to cause some extra delays. Even the fastest processors should be stalled while accessing to the main memory. Hence, speeding up the main memory accesses are one of the most important concerns of the processor designers [1]. One of the main applications of the Virtual Memory (VM) is to share the main memory among multiple programs. The VM allows each program to reference a particular part of the main memory while it prevents the same program accessing to the rest of the memory dedicated to other programs. Technically, as the caches provide faster access to the active portions of a program, the main memory can potentially act as a cache for the VM. A wide research in [2] shows that exploiting an L2 cache in parallel with an L1 cache which offers a hit rate of 90%, brings the hit rate up to 95%. Although the hit rate is only improved by a factor of 5% (in two-level cache system), the miss rate is reduced to half in comparison to the single-level cache system (decreased from 10% to 5%). This is the basic idea that we took into consideration to expand the VM on several hierarchal levels. Using this scenario, instructions have the potential to be executed faster.

1.1. Motivation

Many years ago, programmers were responsible for manually fitting programs which were larger than the main memory storage. The virtual memory eliminated the substantial burden on programmers by moving inactive portions of the program out of the physical memory. As soon as the other programs require more main memory, the inactive portions of the running program(s) are stored in a secondary storage using the virtual memory mechanism. Therefore, the virtual memory enables multitasking by instantly relocation over the main memory [3].

Conceptually, in the virtual mode, the processor produces a virtual address which is convertible to the physical one using a combination of hardware and software approaches. Fig. 1 shows the

* Correspondind author.
E-mail addresses: farid.shamani@tut.fi (F. Shamani), VidaFakour.Sevom@tut.fi (V. Fakour Sevom), Tapani.Ahonen@tut.fi (T. Ahonen), Jari.Nurmi@tut.fi (J. Nurmi).

http://dx.doi.org/10.1016/j.micpro.2016.12.001
0141-9331/© 2016 Elsevier B.V. All rights reserved.

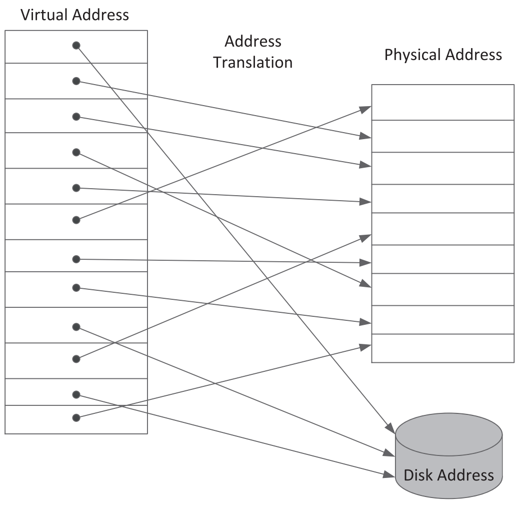


Fig. 1. How virtual memory is mapped to the main memory [3].

address translation procedure. This mechanism is usually performed by the Memory Management Unit (MMU). The primary function of the MMU, which can also be denoted as the most important one, is to translate the virtual addresses into the physical ones [4]. The MMU should guarantee the protection of each program, as well. On the other hand, the MMU is sufficiently rapid to enable all transactions to the main memory [5]. Theoretically, any VM access takes at least as twice long as the main memory access. It requires one memory access to obtain the physical address and a second attempt to get the data. Therefore, by keeping these accesses in a special cache, subsequent memory access will be executed much faster than the first one. Accordingly, the most frequent page translations are kept in a special cache which is referred to as a Translation Look-aside Buffer (TLB). In the virtual mode, the MMU examines its contents to find a matched Virtual Page Number (VPN). When the desired entry is found, the appropriate physical page number is extracted from the corresponding entry and, then, is combined with the offset to form the physical address. The processor employs the obtained physical address to reference a specific location in the memory. On the other hand, rapid technology advancement has enabled digital system designers to design and implement embedded processors capable of performing particular functions [6]. One such embedded processor is the *Core For FREE (COFFEE)* developed by our group at Tampere University of Technology. All the required materials to run the COFFEE processor (including the latest source codes, the corresponding compiler, etc.) as well as the instruction how to use the core can be found on the university website in [7]. In this work, we used the most updated version of the core released on 17-November, 2008. As the name states, the developed processor is an *open-source* core which is suitable for embedded computing. However, the processor lacks the integration of an MMU to support a virtual-to-physical address translation mechanism [8]. Indeed, configuring the COFFEE RISC processor with a fully compatible MMU is one the main motivations for this work. Potentially, the developed MMU can be integrated into other standard RISC-based platforms, as well.

1.2. Related works

There are other studies in which an MMU has been implemented on the FPGA devices for different applications. In [9], the authors proposed a method based on a Network Interface Controller (NIC) which includes a bulk memory. A 128 MB DRAM memory stores 16 million of Address Translation Table (ATT) entries (instead of TLBs) which include information with regard to virtual-to-physical address translations. Each entry of the ATT is composed of 64 bits. After the first run, the NIC driver maps all the virtual memories to the physical ones. Subsequently, the ATT is maintained only by the driver. However, since DRAMs (along with the CPU) are the major candidates for total system energy consumption [10], the address translation in such systems is an expensive mechanism in terms of energy consumption.

Hu-Cheung Ng et al. implemented an MMU on a Xilinx Virtex 5 FPGA board in [11]. The MMU has a TLB which includes virtual-to-physical address translations. The TLB has 16 entries and employs the Least Recently Used (LRU) replacement policy. They also claimed that another small cache which contains 4 entries improved the speed of the memory access. The implemented MMU only supports a fixed page size of 4 KB. A successful translation takes 2 cycles whereas any update to the TLB takes 16 cycles. A TLB miss penalty is very varied between 600 to 227,000 cycles depending on the status of the Operating System (OS).

In [12], a reconfigurable architecture which employs a single 512-entry TLB for address translation mechanism alongside a Direct Memory Access (DMA) unit is proposed. In the case of a miss occurrence, the TLB is locked and the FPGA will be interrupted. Consequently, the FPGA will update the TLB with the corresponding missed entry. Thereafter, the TLB is unlocked and the page translation mechanism will be continued. The authors acknowledged that any TLB miss would take thousands of FPGA cycles whereas an address translation with a TLB hit would only take 4 cycles. They also claimed that the hardware implementation (on Xilinx Virtex 5 FPGA device) significantly sped up the performance (approximately five times) over the pure software implementation. However, they have not mentioned the size of the virtual page space supported by the TLB.

The authors in [13] introduced a specialized TLB design which includes a Buffer Search Ram Cache (BSRC) along with a Virtual-to-Physical (V2P) unit where the V2P is mainly responsible for the address translation mechanism. A complete V2P block composed of eight 32-bit entries was integrated with Content Addressable Memories (CAMs) as a specialized memory to act as a hardware search engine. It takes different patterns as search keys and returns the corresponding addresses where the patterns are stored. This architecture, due to the use of tightly coupled RAM structures, returns a translated address in 1 or 2 cycles whereas deleting or updating a new entry is a complex operation which requires several steps. Both the analysis results and the hardware costs are given based on the implementation on Altera Stratix IV FPGA family specification. Of course exploiting a CAM module instead of a simple RAM will result in a very complex implementation (by a factor of 32) which is a very limiting factor in designs with scarce memory resources [13].

This paper is based on our previous work in [14]. In our earlier work, we developed our MMU on two levels of TLBs, since the process of address translation was quite similar to the caching procedure. On the first level, a 4-entry micro-TLB for instruction memory references operated alongside an 8-entry micro-TLB for data memory references. Both of these micro-TLBs were hardware managed. On the second level, there was a 64-entry unified software-managed TLB to cache both instruction and data page address translations. All employed TLBs were design-time configurable and scalable. The MMU was able to configure itself (mainly it reconfigured the TLBs) to operate on eight different page sizes from 1 KB to 16 MB during the run-time. The MMU was implemented on a 28 nm Altera Stratix-V FPGA board [15] based on the specification given in [16] in absolute terms. In this extended version, we try to demonstrate how the MMU can be integrated into any standard RISC processor as an IP block, what challenges are confronted and how we can tackle each of them. Our case study is the COFFEE RISC processor developed by our group. The main contributions of this paper are briefly as follows:

- Employing two scalable micro-TLBs along side a scalable unified TLB to speed up the overall performance.
- Configurability of the MMU to reconfigure its TLBs to operate on different page sizes.
- Providing a wide study on hardware implementation costs (including power dissipation, logic utilization, etc.)
- Describing some optimization techniques to improve the performance.
- Presenting integration challenges such as providing a wrapper, modifying the hardware description of the core, etc. in order to integrate the IP-based MMU to a standard RISC processor (COFFEE core as the case study).

The rest of this paper is organized as follows. Section 2 describes the COFFEE core in brief. Section 3 presents an overview of the address translation mechanism used by the memory management unit. In Section 4, the hardware implementation of the MMU is discussed in detail. Section 5 explains the design issues which have been considered prior to the integration phase. Next comes Section 6, which gives an insight how to integrate the MMU to the COFFEE core as a peripheral device. Section 7 discusses about synthesis results of the MMU and its respective integration to the COFFEE core on the FPGA, followed by the conclusion in Section 8.

2. The COFFEE core in brief

The COFFEE core is an embedded processor developed by our group at Tampere University of Technology [17]. Technically, the core is a Harvard RISC-based architecture suitable for System-on-Chip (SoC) and embedded systems. Reusability and configurability are the two main characteristics of the developed core. The COFFEE RISC core is capable of executing 66 different instructions in a 6-stage pipeline. Furthermore, 4 co-processors with different clock domains are attached to the core via a third bus to assist the processor in particular applications. In addition, the core also exploits a Core Configuration Block (CCB) alongside a Peripheral Control Block (PCB) to support software configurability and communication with peripheral devices, respectively. In typical applications, the practical operating frequency of the best optimized implementation of the COFFEE (in all aspects) is in the range of 100 MHz while the speed-optimized version operates at about 150 MHz for a 90 nm low power technology [8]. Fig. 2 depicts how the COFFEE RISC core is integrated with other blocks including the PCB. The main input and output ports to integrate the MMU as a PCB block are in bold while the rest of the components are transparent in the figure. In Section 5, we will completely explain how the previously developed MMU is integrated into the COFFEE core using PCB block interface. We also discuss about the main intention of using PCB block over other interfaces (e.g., co-processor bus). In addition, some pros and cons of employing the PCB block are presented, as well.

3. An overview of the MMU

Fig. 3 shows how the MMU operates in two different modes, known as the real and the virtual modes. In the real mode, the 32-bit Effective Address (EA), calculated by the processor, is recognized as the physical address which is used to directly access the physical memory (no address translation is required). In the virtual mode, the address translation mechanism is more complex than the real mode.

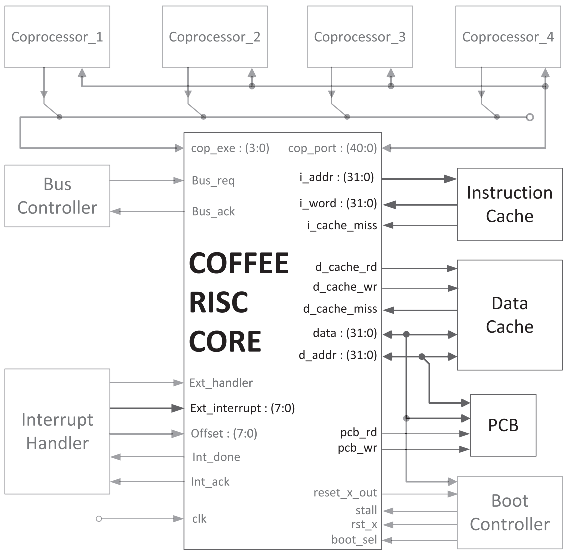


Fig. 2. The COFFEE RISC core integration. The highlighted components as well as input/output ports are involved in the MMU integration as a PCB block.

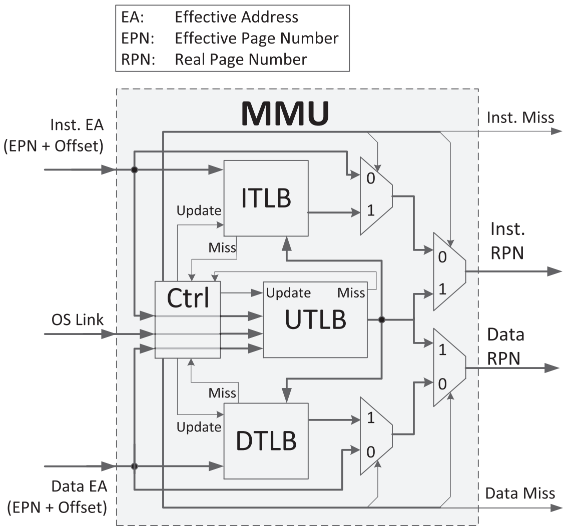


Fig. 3. The MMU behavior with respect to the real mode and the virtual mode.

Fig. 4 illustrates the virtual address translation procedure. The EA obtained from the processor is segmented into two fields of *Effective Page Number* (EPN) and the *offset*. The value N is the boundary to distinguish the EPN segment from the offset one. The value of N is calculated based on 3 specific bits within the EPN segment known as *page size* (further information in Section 4.1). The resulting page size is equal to 2^n where n is the number of bits dedicated to the offset field and, consequently, $N = 31 - n$. For instance, in order to address a 16 KB (2^{14}) page size, $n = 14$ shows that 14 bits are dedicated to the offset field while the remaining 18 bits ($N = 17$) are assigned to the page number field.

The virtual Address is a concatenation of the EA (address calculated by the processor) and the Process ID (PID) driven by the PID

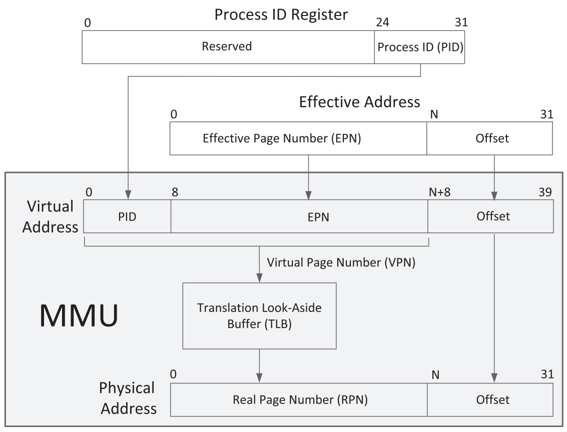


Fig. 4. Address translation procedure in the virtual mode.

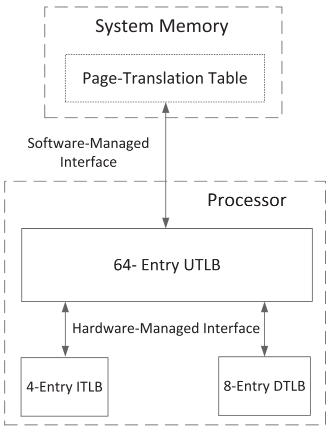


Fig. 5. System memory and processor organization.

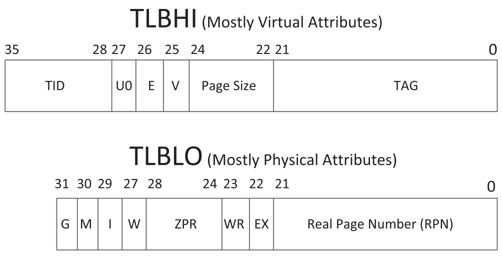


Fig. 6. TLB entry organization.

register. The PID is a unique ID for each process which is used to resolve the overlapped area between different processes in the virtual space. Therefore, the PID field along with the EPN segment of the virtual address contains information with regard to the virtual address space. Nevertheless, the MMU employs PID and EPN fields, as the Virtual Page Number (VPN), to extract the Real Page Number (RPN). The concatenation of the RPN and the offset (which has not been a part of the translation process) is the physical address by which the processor references the memory. Moreover, the MMU is capable of mapping eight different fixed page sizes, including 1, 4, 16, 64, 256 KB, 1, 4, and 16 MB, from the virtual space into the physical space via a page table. The operating system maintains the page table.

4. FPGA Implementation of the MMU

In this implementation, we experimented with the effect of using the UTLB with different configurations, i.e. 16-, 32- and 64-entry, to cache a subset of instruction and data page translations. In our case study, in fact, the 64-entry UTLB is mostly emphasized since it presents more interesting results than the two other configurations. The UTLB is completely maintained by the OS. Modifications to the UTLB such as initialization, entry validation and entry replacement (in the case of a UTLB miss) are only handled by OS. In addition, all of the entries are accessible by the MMU. The MMU also consists of a 4-entry Instruction TLB (ITLB) along with an 8-entry Data TLB (DTLB) to cache the most frequent instruction and data page translations, respectively. These micro-TLBs are managed completely by the hardware meaning that the MMU is responsible for initializing, replacing and validating the contents of each micro-TLB. The basic idea of exploiting micro-TLBs is explained in [6]. Micro-TLBs are employed to minimize access conflicts with the UTLB along with speeding up the address translation process. Fig. 5 shows how TLBs are organized by the operating system and the processor. The following explains each TLB attribute in brief:

- *Unified TLB*: The UTLB contains 64 entries. It is fully associative, meaning that the instruction and data page translations can be stored in any location. The instantiation and modification of the UTLB are completely managed by the OS.
- *Micro-TLBs*: The ITLB/DTLB contains four/eight instruction/data translation entries. They are fully associative. They store the four/eight most frequently accessed instructions/data from the UTLB. The micro-TLBS are used to minimize contention between instruction/data translation and UTLB update operations. The instantiation and modification of the micro-TLBs are completely managed by the hardware.

4.1. TLB entry organization

When the processor is running in the virtual mode, the MMU exploits its TLBs for the address translation mechanism. Each TLB entry contains the necessary information to identify a virtual page, specify its physical page, and determine the protection as well as the storage characteristics of the page. As Fig. 6 illustrates, each TLB entry consists of 68 bits split into two sections. The first 36 bits (TLBHI) mostly contain useful information about the virtual address attributes. Subsequently, the remaining 32 bits (TLBLO) mostly contain information with respect to the physical address attributes. The most important bit fields within TLBHI and TLBLO in this implementation are categorized in the following:

- ***TLBHI:***
  - *TAG* (22 bits). The TAG field is compared with the EPN portion of the EA based on the corresponding *page size* field.
  - *Page Size* (3 bits). This field specifies the page size used to define how many bit ranges of the TAG field should be compared with EPN. The size value 0b000 up to 0b111 represents page sizes of 1KB up to 16 MB, consecutively.
  - *V*. This bit determines whether the current entry is valid.
  - *TID* (8 bits). This field is compared with the corresponding PID field.
- ***TLBLO:***
  - *RPN* (22 bits). This field determines the corresponding physical address in the case of a TLB hit.

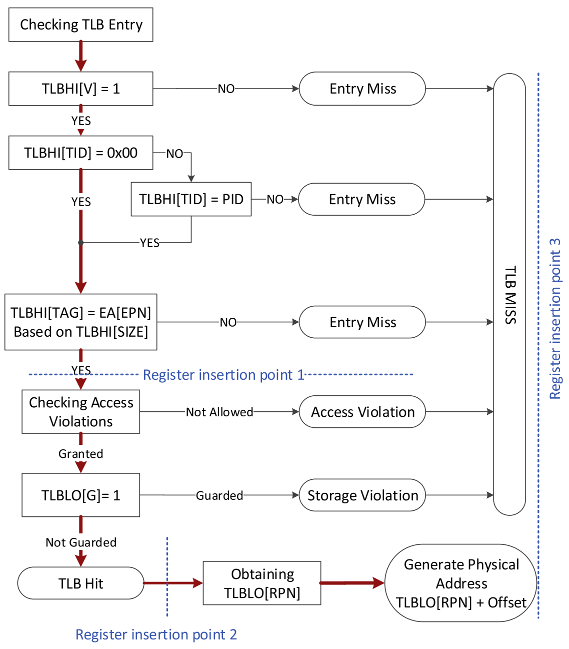


Fig. 7. How each TLB entry responds to a reference. In the case of the UTLB, the thicker arrows (shown in red) illustrate the longest critical path propagated through the design. Suitable locations for register insertion are presented, as well. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

– *ZPR* (4 bits). This field selects one of the 16 different zone fields of the Zone Protection Register (ZPR).
– *G*. This bit determines whether the current entry is guarded. The term “guard” is referred to the situation where the speculative memory access, such as instruction pre-fetch, is not permitted.

4.2. TLB index examination

As Fig. 7 illustrates, when examining a page translation entry in either of TLBs, all valid entries in the corresponding TLB are checked simultaneously. A TLB hit will occur if all the following criteria are met:

- The entry is valid.
- The TAG field is identical to EA[EPN] under the control of the page size field.
- The TID field is identical to PID.
- Access to the page is granted and the page is not guarded.

If one of the above conditions is not met, a TLB miss occurs which results in an exception. It is worth mentioning that a TID value of 0x00 represents a process-independent translation from the operating system point of view. Therefore, the MMU ignores comparing TID field with the PID. In this case, only the TAG and EA[EPN] are compared. Moreover, A PID value of 0x00 identifies a process which cannot access any page. Obviously, the consequence of such a PID is an entry miss.

When a hit occurs, the MMU investigates the access control bits of the corresponding TLB entry, prior to accessing the physical address. Similarly, the MMU inspects the storage attribute fields to determine whether the access to the page is allowed. Eventually, if all the criteria are met, a TLB hit will take place.

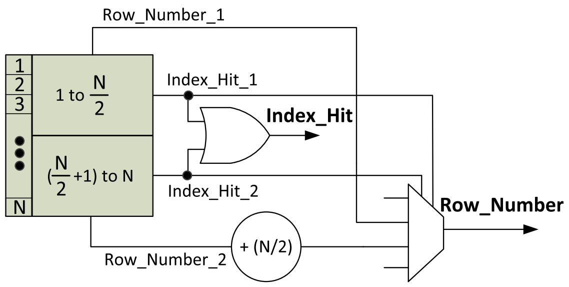


Fig. 8. How to break large loops into two smaller ones.

Fig. 7 shows how each TLB entry is verified during the address translation process. It is guaranteed that a hit will occur when the incoming EPN passes five consecutive examination stages. Subsequently, a TLB miss is occurred when at least one of the following criteria is met:

- A matching TLB entry is not found.
- A matching TLB entry is found, but the contents are not valid.
- A matching TLB entry is found, but the access to the page is denied.
- A matching TLB entry is found, but the page is guarded.

In the presence of a UTLB miss (irrespective of the miss type), a failure is detected by the MMU and, consequently, an interrupt is issued which reports the cause to the *interrupt handler*. Thereafter, the interrupt handler mechanism is executed to resolve the corresponding problem. In the meantime, the processor enters in the real mode by clearing both IR and DR bits in the Machine State Register (MSR). When the necessary updates have been loaded into the UTLB, the processor will return to operate in the virtual mode. Therefore, during the interrupt handling process, the MMU is disabled until the UTLB is reloaded with proper entries. Any update to an entry in the UTLB forces the MMU to invalidate all entries in both micro-TLBs. Instruction and data consistency between micro-TLBs and UTLB will be guaranteed in this way. Furthermore, Fig. 7 presents the longest critical path propagated through the design with thicker arrows shown in red. After monitoring and analyzing the design, the longest critical path belongs to the UTLB due to the relatively large amount of loop iterations to investigate all the entries, e.g., 64 times in the case of a 64-entry UTLB. Indeed, the hardware circuit is always bounded by the worst-case scenario. In order to shorten the long critical path as much as possible, we took advantage of two well-known optimization techniques: *register insertion* and *large loops mitigation*. Fig. 7 also illustrates the most suitable locations for register insertion. These points are discovered after performing timing analysis intently.

Fig. 8 depicts how searching through the entire N-entry TLB can be split into several smaller loops. In this figure, a large 64-iteration loop is divided into two 32-iteration ones in order to search an index in the UTLB. In the case of an entry hit in either of loops, the respective index flag (Index_Hit_1 or Index_Hit_2) is set meaning that a matched entry is found. Accordingly, signal Row_Number will reference to the exact row where the matched entry is found (*Row_Number_1 or Row_Number_2*). For example, if a match is found at the 20^{th} entry (location is within the first loop), the Index_Hit_1 flag will be set and consequently Row_Number_1 = 20 will be assigned to the Row_Number value. If the desired VPN address is found at the 50th entry (second loop), the Index_Hit_2 is set and consequently Row_Number_2 = 18 + 32 (=50) is assigned

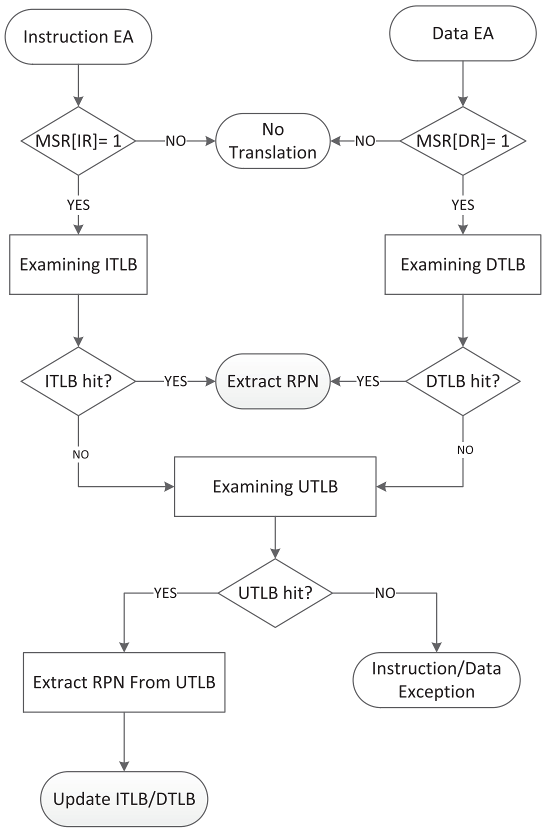


Fig. 9. MMU address translation flow.

to the Row_Number value. In concept, this method is quite similar to the *unfolding* technique which is known as *loop unrolling* in general programs. The effect of using such optimization techniques in order to minimize the critical path will be discussed in Section 7.

4.3. MMU address translation flow

Fig. 9 shows the address translation flow through TLBs. When the virtual mode is activated for either instruction or data side, the MMU translates a combination of PID and EA into the physical address. Thus, it first examines the appropriate micro-TLB to find a matched VPN. If a valid entry is found, it is used to access the physical memory. A micro-TLB hit will take place in two cycles. If no entry is found, the MMU examines the UTLB for the corresponding virtual address. Additional delays are presented each time the UTLB is accessed due to a micro-TLB miss. If both micro-TLBs experience simultaneous misses, the DTLB has priority over the ITLB to access the UTLB. In such a situation, the UTLB will first address the data side by searching all the entries for a matched data page translation and then evaluates the instruction EPN. Therefore, in this implementation, the DTLB miss penalty is 5 cycles while the miss penalty for ITLB varies between 5 to 7 cycles (7 cycles in the case of a simultaneous miss with DTLB). Thereafter, the physical address is fetched from the UTLB. In the meantime, the appropriate micro-TLB entry is updated with the new entry as well. As stated earlier, in the case of a UTLB miss an exception is reported to the operating system via the interrupt handler.

Table 1
Minimum and maximum cycle counts for each virtual-to-physical address translation in an ideal case.

	Data	Instruction
Micro-TLB hit	2	2
Micro-TLB miss, UTLB hit	5	5–7
Micro-TLB miss, UTLB miss	11	11–13

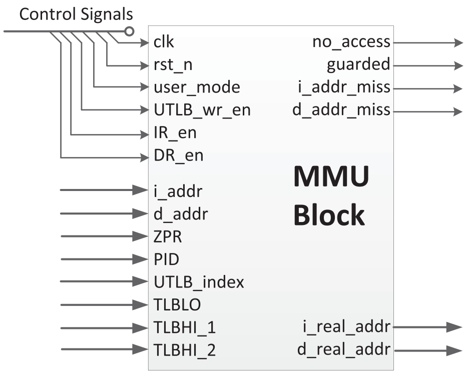


Fig. 10. The developed MMU as a black box.

How each exception is typically addressed depends on the operating system status, it will take six clock cycles to be resolved in an ideal case and hundreds (even thousands) cycles in normal cases. Table 1 summarizes total cycle counts for each virtual-to-physical address translation in an ideal case. The ideal case is referred to the situation where an UTLB miss can be instantly updated in the fastest possible time. However, such these cases might not ever happen with today's available operating systems. The average time when the operating system can resolve these issues are currently being investigated in our research group.

The Least Recently Used (LRU) replacement policy is considered for both micro-TLBs. Each micro-TLB employs a specific counter which counts each reference to the TLB. In the case of a TLB hit, the LRU counter of the corresponding entry is reset while the remaining LRU counters are advanced by one. When a TLB miss occurs, the entry with the highest LRU counter number is the best candidate to be replaced by the new entry fetched from the UTLB. The lifetime of each LRU counter within the TLB is equivalent to 2 $\times$ number of entries assigned to the same TLB minus one. For example, DTLB (in this implementation) has eight entries; hence, the maximum number achieved by the LRU counter is $2 \times 8 - 1 = 15$. The entries with the highest LRU counter have not been referenced for a while. Thus, they are assumed to be good candidates for replacement. In order to keep micro-TLBs coherent with the UTLB, the controller (see Fig. 3) will immediately invalidate the contents of both micro-TLBs. Invalidation is simply performed by clearing the valid bit (TLBHI[25] in Fig. 6) of all entries right after encountering the UTLB miss. The UTLB miss will disable MMU address translation flow, as well. As soon as the operating system completes updating the UTLB, the MMU is unlocked and the virtual address translation flow is resumed.

5. Integration issues and facts

Fig. 10 illustrates the previously described MMU as a black box where the most important ports are described in the following:

- user_mode: Indicates whether the processor is operating in super-user mode ('0') or user mode ('1').

- UTLB_wr_en: When this bit is set, one of the UTLB entries is planned to be updated or replaced by the operating system.
- IR_en/DR_en: Instruction/Data Relocation. These bits enable the virtual address translation mechanism.
- i_addr/d_addr : Instruction/Data Address. These inputs are the respective virtual addresses.
- ZPR: Zone Protection Register is used to override the access protection in a TLB.
- PID: Process ID is a unique ID for each process to resolve the overlapped area shared between processes in the virtual address space.
- UTLB_index: Is the link between the operating system and the UTLB.
- TLBLO: Updates the low part of a specific UTLB entry (the physical address attributes).
- TLBHI_1/2: Update the high part of a specific UTLB entry (the virtual address attributes).
- no_access/guarded: Arise appropriate exceptions when the access to the page is not permitted by any means.

Typically, when the IR_en/DR_en bit is set, the MMU enters the virtual mode meaning that the current 32-bit i_addr/d_addr is considered as a virtual address. Hence, the appropriate physical address can be fetched from corresponding i_real_addr/d_real_addr as soon as the MMU has completed the translation process. In contrast, when the IR_en/DR_en bit is clear, the MMU enters the real mode and bypasses i_addr/d_addr to the corresponding output i_real_addr/d_real_addr. The operating system uses the UTLB_index to make a reference to a particular entry in UTLB in order to make any modification or replacement. The appropriate modifications are applied to the UTLB through the TLBLO, TLBHI_1 and TLBHI_2 signals. The only assumption taken into account is that updating a UTLB entry has to be performed as an atomic operation. It means the operating system must not suspend the execution of instructions led to address TLBLO, TLBHI_1 and TLBHI_2 registers. Moreover, it should be mentioned that none of the I/O ports of the MMU can be left unassigned.

There are two possible ways to integrate a peripheral device (such as the MMU) to the COFFEE RISC core. One method is to integrate the MMU via the co-processor interface. A very good example can be found in [18] where the authors presented the integration of a design-time configurable Floating Point Unit (FPU) called Milk co-processor to the COFFEE core through the co-processor bus interface. The co-processor bus is a third 32-bit bus (in addition to the instruction and data buses) which is connected to the COFFEE core with a separate address space. One constraint is that the provided address space is limited to address only 64 addresses in total (4 co-processors with up to 16 registers). Although the provided address space would be sufficient to support all the required registers by the MMU, the basic idea behind the design of the COFFEE core is to connect all peripheral devices over the memory-mapped shared data bus. On the other hand, integrating a peripheral device over the co-processor bus takes advantage of having access to the data one pipeline stage earlier than the memory-mapped shared data bus. This is due to the fact that there is no access right restriction on the co-processor bus while complicated timing issues for the same design may occur. Unintentional timing issue is the main limitation why we prefer not to use the co-processor bus for integrating the MMU to the core.

Another alternative is to integrate the peripheral device via the PCB block interface. The main advantage of this interface over the co-processor bus is that there is no timing issue on the shared bus. Another advantage of the PCB block is that it is directly attached to the (shared) data bus interface of the COFFEE core. It enables faster access to the required data, particularly when the operating system needs to update the MMU. However, this reveals as a shortcoming

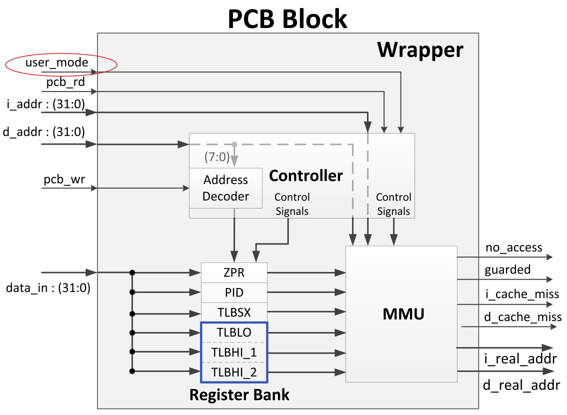


Fig. 11. The wrapper mimics CCB registers of the COFFEE RISC core through the register bank. The blue square in Register Bank block depicts the atomic operative registers. The processor has to indicate its status through the signal *user_mode*. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

that any unintentional transaction on the data memory bus may results in PCB misbehavior. Therefore, COFFEE core controls the PCB block by employing *pcb_rd* and *pcb_wr* signals (see Fig. 2). All accesses to the memory block reserved for the peripheral device to read from/write to the memory are granted by pcb_rd and pcb_wr signals, respectively. It is obvious that these two signals can not be activated simultaneously. Similarly, *d_cache_rd* and *d_cache_wr* signals assert data memory accesses directly.

There are only four signals driven by the core to the PCB block which are *data, d_addr, pcb_rd* and *pcb_wr*. However, the MMU still requires a few more signals connected to its I/O ports of which *clk* and *rst_n* signals are directly derived from the global clock and reset signals. The remaining ports should be supplied by a wrapper from/to the core.Fig. 11 illustrates how the wrapper connects the MMU to the COFFEE core via the PCB slot. As the figure shows, some of the input ports to the MMU are provided by an internal register bank. Furthermore, a controller block tightly controls the behavior of both register bank and the MMU through a set of controlling signals. The following subsections explain each block as well as the wrapper interconnections.

5.1. Controller block

Typically, the controller unit mainly takes care of the data flow through the design. The controller employs two different sets of control signals (*control signals* Fig. 11)to manage the register bank as well as the MMU. The following explains how the controller takes advantage of some input signals to moderate the behavior of the MMU.

5.1.1. pcb_rd

This input is driven by the core and indicates whether the address translation mechanism is required. Hence, the controller connects the pcb_rd signal directly to both IR_en and DR_en ports of the MMU via the control signals set. It infers that whenever the MMU is enabled, both instruction and data addresses are receiving from the virtual space. Otherwise, both addresses are considered as the physical addresses. When the signal pcb_rd is asserted, the physical instruction and data addresses will be sent to the core after a certain number of clock cycles. The minimum and maximum latencies to obtain the appropriate physical addresses are investigated in [14].

5.1.2. *pcb_wr*

The input port *pcb_wr* is another control signal which indicates whether an update or replacement to one of the UTLB entries is planned by the operating system. In contrast to the signal pcb_rd, pcb_wr is not directly connected to the MMU port. When the signal pcb_wr is asserted, it does not necessarily mean that the operating system is modifying the content of the MMU. For example, any modification to the register bank can only be applied when the pcb_wr signal is set while it has nothing to do with the MMU. The controller unit only enables *UTLB_wr_en* signal when the following conditions are met: first, the signal pcb_wr is set and, second, an atomic operation is executing. When the content of the TLBLO register in the register bank is changed, the controller is informed of the execution of an atomic operation.

5.1.3. *user_mode*

Signal *user_mode* is a tricky signal driven by the core. The controller takes advantage of this signal to inform the MMU whether the processor has the potential to operate on super-user mode or user mode. Typically, operating system is the only software run in super-user mode and other applications are only allowed to run in user mode [16]. Since the super-user mode allows the program to access all registers as well as executing all instructions, information with regard to processor status is one of the most critical inputs to the MMU. As it is shown in Fig. 11, input signal user_mode is not provided by the COFFEE core by default. Therefore, it should be dug out of the core. The Processor Status Register (PSR) is an 8-bit read-only register of which bit number 1 (PSR[0] or PSR[UM]) provides the exact information that we are looking for [19]. Thanks to the open-source nature of the COFFEE core, the signal *user_mode* can be extracted directly from the *PSR[UM]*. Henceforth, any change to the status of the processor will inform the MMU to alter its behavior when needed.

5.1.4. *i_addr, d_addr*

These input signals are directly connected to the corresponding input ports of the MMU (similar to the pcb_rd). When the processor is running in real mode (pcb_rd = ’0’), the MMU bypasses these two signals to the respective outputs assuming that these signals are physical addresses.

5.2. *Address decoder*

As previously stated, the COFFEE RISC core exploits different set of register banks of which CCB is the one that provides software compatibilities. The initial value for the CCB base address (ccb_base) is “0001 0000h” after the reset. The next 256 consecutive addresses are reserved for CCB register bank (address range “0001 0000h” to “0001 00FFh”). Thereafter, the next 256 subsequent addresses (address range “0001 0100h” to “0001 01FFh”) refer to the PCB block. In this implementation, we assume that the operating system employs the address range “0001 0100h” to “0001 0105h” to reference to ZPR, PID, TLBSX, TLBLO, TLBHI_1 and TLBHI_2, respectively. When the signal pcb_wr is asserted, the address decoder block examines the eight least significant bits to extract which block inside the register bank should be updated by the content of the *data_in* bus (see Fig. 12). For example, when the pcb_wr signal is set, the value “02h” on d_addr bus indicates that the TLBSX register is the target which should be updated by the content of the data_in bus. Therefore, the controller issues the appropriate commands.

5.3. *Register bank*

The *register bank* block mimics CCB registers of the COFFEE core for the MMU inside the wrapper. CCB registers are employed

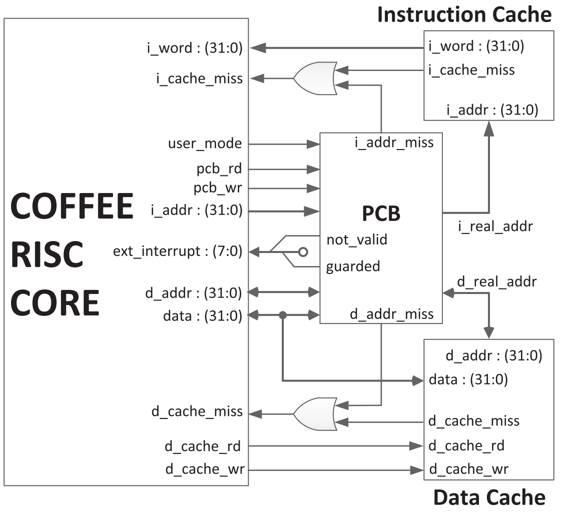


Fig. 12. PCB integration with COFFEE RISC core. The PCB employs 2 bits of the *Ext_interrupt* signal to interrupt the core.

by the operating system to communicate with the hardware. As soon as the operating system updates CCB registers, the processor will apply new changes to the PCB register bank by overwriting the appropriate register. It is worth mentioning that the last three registers shown in the blue square in Fig. 11(TLBLO, TLBHI_1 and TLBHI_2) have to be updated in an atomic operation. It means the core can not suspend the execution of the instructions which are meant to make any modification to the UTLB.

6. Integrating the MMU to the core via the PCB interface

Fig. 12 depicts how the PCB block (consisting of the MMU surrounded by a wrapper) is integrated into the COFFEE RISC core. It is obvious that the unidirectional instruction bus (i_addr) as well as the bidirectional data bus (d_addr) are connected to the corresponding caches via the PCB block. It means that there is no physical interconnection between the core and caches. That is why the MMU forwards the instruction/data address to the corresponding cache when the processor is operating in real mode. In the virtual mode, the processor produces virtual addresses which are first translated into the physical ones by the MMU and then forwarded to the appropriate cache. In the case of a successful address translation, the relative cache provides the core with the *instruction word* or the *data* via the direct physical interconnection between core and the caches. The MMU has the capability to interrupt the core when there is any violation during the translation mechanism, e.g. the page is not accessible or the page is guarded, via the *not_valid* and *guarded* signals. Therefore, the processor informs the operating system to resolve the problem through the exception handling mechanism. The MMU communicates to the processor with respect to the page violations via the available *ext_interrupt* interface of the core. Furthermore, the MMU informs the processor with respect to the instruction/data address translation misses by asserting i_addr_miss and d_addr_miss signals, respectively. These signals are connected to the COFFEE through the i_cache_miss and d_cache_miss interfaces of the core. Since these interfaces are reserved for instruction and data cache misses, we used an *OR* gate to resolve the conflicts. On the processor side, when either of the cache miss interfaces are triggered, the processor can not distinguish whether the current miss is asserted by the cache or the

Table 2
Critical path analysis. Level_1 indicates the register insertion strategy and Level_2 represents loop splitting technique.

	No optimization	Level_1	Level_2
Worst critical path (ns)	6.164	5.154	4.935
Clock skew (ns)	−0.054	−0.130	−0.020
Maximum freq. (MHz)	163.59	193.09	200.80
Speed-up	–	18.03%	22.75%

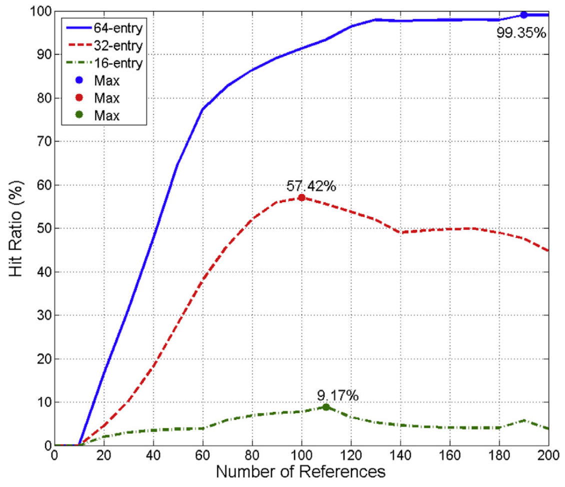


Fig. 13. The hit rate with different number of entries in the UTLB for 200 random references after the boot-up. The curves are the smoothed version of original ones for intervals of 10 references.

MMU. In such a situation, we assumed that the processor monitors the corresponding ext_interrupt interfaces connected to the MMU. If both guarded and not_valid signals have been set, the current cache miss is related to the MMU. Otherwise, the cache miss is asserted by the relative cache itself.

7. Synthesis results with respect to the MMU implementation

The described MMU was written in Very-high-speed integrated circuit Hardware Description Language (VHDL), simulated using ModelSim software, compiled and synthesized by Quartus-II 13.1.1 environment and finally implemented on a 28 nm Altera FPGA device targeting Stratix-V GS mainstream speed grade 2. Table 2 summarizes the effect of a long critical path before and after the optimization techniques discussed in Section 4.2 when the UTLB is configured for 64 entries. The longest critical path (6.164 ns) is recorded when no optimization technique is performed. The optimization Level_1 presents new results when the register insertion strategy is considered. Although inserting registers in the middle of entry verification process has the potential to introduce additional delay, the critical path was reduced more than 1 ns (5.154 ns). In optimization Level_2, breaking a large loop into smaller ones technique is used in addition to register insertion strategy. However, splitting a large loop into two smaller ones (in this case study, 64 iteration to 2 × 32 ones) has a negligible impact on shortening the critical path in comparison with register insertion technique.

Fig. 13 illustrates the UTLB hit rate for the first 200 random accesses (including repetitive numbers) after the boot-up when the UTLB is configured with 64, 32 and 16 entries, respectively. Random accesses are considered to represent the worst-case scenario for the implemented MMU. However, we should expect better result for references to the TLB in real applications. In order to do so,

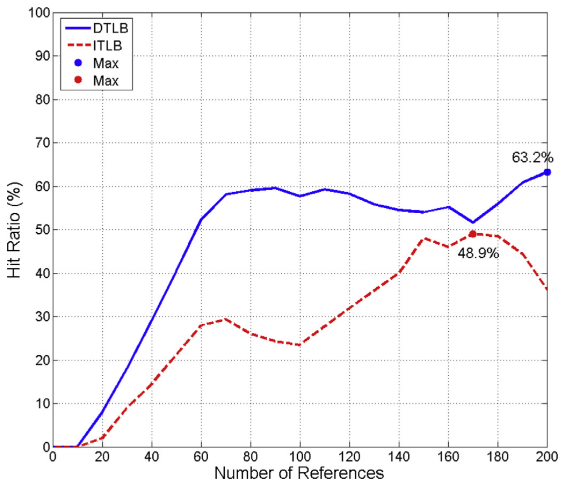


Fig. 14. The DTLB/ITLB hit rate for 200 random data/instruction references after the boot-up. The curves are the smoothed version of original ones for intervals of 10 references.

we took into consideration that the MMU only deals with 1KB page size where all the pages are accessible by any means. The random number generator is quite similar to the specification given in [20]. It is obvious that the first references to the UTLB, regardless to the number of entries, should result in misses since the UTLB is empty (compulsory misses). All curves are smoothed at each 10 references to the corresponding TLB to present a more visible shot in the figure. The curves show that the 64-entry UTLB has the best hit rate (up to 99%) while 32-entry UTLB has the hit rate of maximum 57%. The hit rate is degraded due to the variety of capacity misses occurring in 32-entry UTLB. Capacity misses are drastically affected by the design, when the UTLB is configured for 16 entries. A 16-entry UTLB does not respond to more than 9% of the references. The 16-entry UTLB seems to be useless in this case. Since our target is to maximize the MMU hit rate, the 64-entry UTLB is the first choice of the interest. Furthermore, the 64-entry UTLB shows a very stable behavior after (approximately) 130th reference onwards. It implies that we will have the same hit rate for the test vectors more than 200 references. Therefore, 200 random numbers are only considered to demonstrate the behavior of the overall system.

Fig. 14 shows how DTLB and ITLB response to the first 200 corresponding references after the boot-up. Although micro-TLB hit rates are not ideal, it is worth mentioning to recall that the investigation is based on random instruction/data references to the micro-TLBs to make the worst-case happen. The smoothed curves (accumulating hits for intervals of 10 random references) illustrate both compulsory and capacity misses occurring in ITLB as well as DTLB. Nevertheless, it seems that the MMU takes advantage of micro-TLBs to some extent.

In this implementation, we also exploit using UTLB with different number of entries including 16, 32 and 64 entries. Analysis results are reported in Table 3 in terms of logic utilization, total registers and maximum operating frequency for the described MMU composed of two micro-TLBs operating in parallel with a 16-/32-/64-entry UTLB. The maximum operating frequencies of approximately 200, 225 and 265 MHz are achieved with different UTLB configurations. As the rule of thumb of all caches, the larger the UTLB, the slower operating frequency. Furthermore, 64-entry UTLB employs more resources and registers for its implementation. In

Table 3
Summary of the implementation analysis.

	Memory management unit		
	UTLB_64	UTLB_32	UTLB_16
Logic utilization (ALMs)	3459	2092	1373
Total registers	5263	3221	2213
Maximum freq. (MHz)	200.80	228.57	267.38

Table 4
Energy consumption analysis for the first 200 references (μJ).

Thermal energy	Memory management unit		
	UTLB_64	UTLB_32	UTLB_16
Static	14291.10	14278.20	14275.95
Dynamic	326.25	167.85	145.65
I/O	431.55	427.20	454.80
Total	15048.90	14873.25	14876.25

our case study, the 64-entry UTLB shows satisfactory results while the 16-entry UTLB is a kind of wasting resources. There is a trade-off between using UTLB with either 32 or 64 entries. If, for example, there is no restriction on resources usage, 64-entry is the best candidate to be implemented for UTLB for the sake of higher hit rate.

Energy consumption analysis is reported based on the results achieved by PowerPlay Power Analyzer tool in Quartus-II software. The analyzer directly reads the Switching Activity Interchange Format (SAIF) generated by ModelSim for the MMU design running at 50 MHz. Static probability and toggle rate for each signal are calculated based on generated VCD file for the first 200 random address translations after the boot-up which can be considered as the worst-case scenario. Random numbers force most of the transistors to toggle per clock cycle. Table 4 summarizes the estimation results based on different UTLB configurations. Based on the energy consumption analysis report, dynamic energy is drastically increased while the MMU is configured with 64-entry UTLB due to triggering more transistors. The trade-off between configuring UTLB with 32 entries or 64 entries is still established here. It seems that a 32-entry UTLB might be a good candidate in low-power designs where the hit rate can be sacrificed. The 16-entry UTLB is not recommended from the energy consumption point of view either.

Dynamic energy consumption for each cell regarding the MMU block and its sub-blocks are reported in Table 5. Statistics show that the MMU itself consumes most of the energy (273, 229.5 and 244.8 μJ with respect to 64-, 32- and 16-entry UTLB) to manage the address translation flow for the first 200 references. Followed by, UTLB, DTLB and ITLB feature the highest energy consumptions, respectively. However, DTLB, ITLB and controller blocks consume approximately the same energy regardless of the UTLB configuration. Moreover, the MMU can take advantage of exploiting micro-TLBs to speed up the translation mechanism by consuming a trivial amount of energy. Choosing a proper configuration for UTLB has an impressive impact on dynamic as well as routing energy consumption. The 64-entry UTLB consumes a large amount of energy in comparison with other configurations.

Table 6 shows the synthesis results after integrating the mentioned MMU to the COFFEE RISC processor. As the table reveals, the total design requires 8403 ALMs of which about 4635 (55%) is used to implement the core, 3594 (43%) is employed by the MMU and the remaining 174 (2%) is considered for intermediate components such as memory blocks, three-state drivers, etc. Furthermore, the total design requires 10,884 registers of which 50.5% is the processor's share and 49.5% is used by the MMU. Similarly, COFFEE RISC processor roughly requires 6425 (57%) combinational Adaptive Look-Up Tables (ALUTs), while the MMU needs 4535 (40%) of the total dedicated resources. It seems that implementing the MMU (which is configured with 64-entry UTLB) is as hardware expensive as implementing our target processor.

Table 7 presents the maximum achievable operating frequencies and their respective critical path analyses for the COFFEE core, MMU and their integration. Maximum frequencies are measured based on the speed-optimized version of all three designs at two different temperature in fast and slow timing models. However, the most practical (worst possible case) operating frequency can be considered as the slow model at 85 °C. Although the MMU is capable of operating around 200 MHz, the speed-optimized version of the COFFEE processor could not achieve more than 180 MHz in this case study. Therefore, the maximum affordable frequency of the total design (COFFEE+MMU) is being restricted by the lowest operating frequency which is around 178 MHz. The critical path analysis reports the interconnection delays are the major sources of lengthening the critical path in all implementations. In addition, the worst critical path is generated in the core itself, when the COFFEE is integrated into the MMU. A negative clock skew in all design reveals that the circuits have the potential to operate at lower frequencies, as well. The COFFEE processor has 0.366 positive slack value, meaning that the post-fit netlist can potentially override the constraints we introduced to the analyzer, while the MMU as well as the COFFEE+MMU architectures could roughly meet the timing requirements. Overall, we could set up the most speed-optimized version of the processor with the MMU with a negligible degradation in performance. The integration of an MMU configured with 64-entry TLB is potentially as hardware expensive as our processor.

In order to compare the achieved results with an Application Specific Integrated Circuit (ASIC) version of the design, Kuon et al. have provided an investigation on the overall performance gap of a particular design between a 90 nm FPGA and a 90 nm ASIC in [21]. Accordingly, a 90 nm ASIC implementation is 35× smaller and 3.4× −4.6× faster compare to the same design on a 90 nm FPGA. In addition, the 90nm ASIC implementation consumes 14× less dynamic power as well as 87× less static power. According to Nouri et al. in [22], if we take 60% speed up into consideration, the same gap ratios are expected for a design on a 28 nm ASIC and a 28 nm FPGA.

8. Conclusions

In this paper, an FPGA implementation and integration of a run-time configurable Memory Management Unit (MMU) to the COFFEE RISC processor which had been developed by our research group at Tampere University of Technology was described. The MMU employed two levels of Translation Look-aside Buffers (TLBs) of which a 4-entry micro-TLB operated alongside an 8-entry micro-TLB (both hardware-managed) on the first level to keep track of the most recently used instruction and data address translations, respectively. On the second level, a software-managed Unified TLB (UTLB) stored both instruction and data address translations. The MMU was capable to support eight different page sizes from 1 KB to 16 MB in the virtual mode at the design time. Micro-TLBs were observed to speed up address translation with a low energy overhead. Three UTLB configurations of 16, 32, and 64 entries were studied with respect to the overall hit rate, operating speed, energy usage and resource utilization. The critical path of the logic design was optimized by introducing explicit concurrency as well as retiming through register-to-register logic balancing. Maximum operating frequencies of 265, 225 and 200 MHz were achieved for the 16-, 32-, and 64-entry UTLB configurations, respectively. Using worst-case, i.e., random stimuli, the 64-entry UTLB provides the best worst-case hit rate of up to 99%. The 32-entry UTLB cut

Table 5
Energy consumption of each component for the first 200 references (μJ).

	Dynamic energy			Static energy			Routing energy			Design total energy		
UTLB Entries	64	32	16	64	32	16	64	32	16	64	32	16
MMU	78.45	73.95	100.65	73.65	73.65	73.65	120.90	81.90	70.50	273.00	229.50	244.80
ITLB	7.80	7.95	7.50	0	0	0	4.65	4.20	4.65	12.45	12.15	12.15
DTLB	13.95	14.55	15.40	0	0	0	8.10	8.85	9.15	22.05	23.40	24.55
UTLB	**81.15**	29.10	18.45	0	0	0	**75.30**	11.10	11.85	**156.45**	40.20	30.30
Controller	0.60	0.75	0.75	0	0	0	9.00	4.95	3.45	9.60	5.70	4.20
Total energy	181.95	126.30	142.75	73.65	73.65	73.65	217.95	111.00	99.60	**473.55**	**310.95**	**316.00**

Table 6
Summary of the hardware implementation of the COFFEE processor integrated with the MMU.

	Logic utilization (ALMs)	Total registers	Combinational ALUTs
COFFEE	4635 (55.16%)	5493 (50.47%)	6425 (57.25%)
MMU	3594 (42.77%)	5382 (49.45%)	4535 (40.41%)
Other blocks	174 (2.07%)	9 (0.08%)	263 (2.34%)
Total design	8403	10,884	11,223

Table 7
Maximum operating frequencies as well as the worst critical path analyses of the COFFEE, MMU and their respective integration.

			COFFEE	MMU	COFFEE+ MMU
Maximum Frequency (MHz)	*Slow*	85 °*C*	180	200	178
	Model	0 °*C*	188	214	182
	Fast	85 °*C*	267	295	253
	Model	0 °*C*	289	318	271
Worst Critical Path Analysis (ns)	*Data delay*		5.604	4.935	5.716
	Interconn. delay		4.077	3.623	4.313
	Cell delay		1.527	1.312	1.403
	Clock skew		−0.035	−0.020	−0.007
	Time slack		0.366	0.020	0.072

down resource utilizations and energy usage of the MMU design. However, with worst case hit rate of up to only 57%, system-wide energy utilization and performance would be sacrificed by frequent misses. While a 16-entry UTLB could operate at a somewhat higher speed, its worst case hit rate of up to 9% translates into system-wide drawbacks on performance and energy usage. We met our target frequency of 200 MHz on Altera Stratix V speed grade 2 family device for all studied configurations, making the 64-entry UTLB our preferred choice, while a 32-entry configuration remains a viable option for resource constrained systems. Furthermore, the MMU (configured with 64-entry UTLB) was integrated into the Peripheral Control Block (PCB) interface of our processor. Since a link to the operating system had not been provided by the processor, we had to explicitly modify the hardware description of the core to extract required information such as current status of the processor. In addition, the MMU was surrounded by a wrapper to compensate other operating-system-wise interfaces including process ID, zone protection, etc. This integration had approximately the same hardware cost as the processor had alone by introducing additional 42.77% ALMs alongside 49.45% registers to the overall design, while preserving the same operating frequency for the most speed-optimized version of the core. This integration enabled the core to access the external memory as well as exploiting the virtual address space for the operating system. Moreover, this paper presented comprehensive information relative to the PCB integration. This would be a fine resource for future researches on integrating any peripheral device to the target core as well as the other standard RISC processors.

Future work

Currently, our software engineers are working to port the Linux operating system on the COFFEE RISC processor integrated with the MMU. This will provide us to experience the behavior of the COFFEE+MMU in order to generate new real-time results. Moreover, we will further modify and expand the MMU for sharing over a Network-on-Chip (NoC) in order to better serve multi-core systems. In this work we will also adapt the design to take advantage of the extreme bandwidth obtainable through a memory on top of logic organization of 3D Stacked Integrated Circuits (3DSIC). This configuration is often referred to as 2.5D and is the most likely candidate for successful mass production in the near future, while the prototype chips have been available for some time now.

Acknowledgments

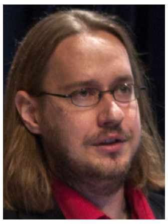
The research leading to these results has received funding from the ARTEMIS JU under grant agreement number 295371. Portions of the work were also funded by the Academy of Finland under contract # 258506 (DEFT: Design of a Highly-parallel Heterogeneous MP-SoC Architecture for Future Wireless Technologies). The first author gratefully acknowledges the grant support awarded by Finnish Cultural Foundation as well as the Jane and Aatos Erkko Foundation, under the project Biological Neuronal Communications and Computing with ICT.

References

[1] N.J. K. Hwang, Advanced Computer Architecture, 2nd Edition, Teta McGraw Hill, 2000.
[2] J. Handy, The Cache Memory Book, 2nd Edition, Academic Press Inc., 1998.
[3] D. Patterson, J. Hennessy, Computer Organization and Design, 4th Edition, Morgan Kaufmann Publishers, 2009.
[4] B. Cohen, R. McGarity, The design and implementation of the MC68851 paged memory management unit, Micro. IEEE 6 (2) (1986) 13–28.
[5] G. Stefan, F. Ăghici, Memory management unit-a new principle for LRU implementation, in: Proceedings on 6th Mediterranean Electrotechnical Conference, IEEE, 1991, pp. 989–992.
[6] J. Ball, Designing Soft-Core Processors for FPGAs, in: J. Nurmi (Ed.), Processor Design: System-On-Chip Computing for ASICs and FPGAs, 1^{st} edition, Springer Publishing Company, 2007, pp. 229–256.
[7] T.U. of Technology, COFFEE RISC core - a COre For FrEE, Accessed on 02.Mar.2016, (http://coffee.tut.fi).
[8] J. Kylliäinen, T. Ahonen, J. Nurmi, General-Purpose Embedded Processor Cores – The COFFEE RISC Example, in: J. Nurmi (Ed.), Processor Design: System-On-Chip Computing for ASICs and FPGAs, 1st ed. edition, Springer Publishing Company, 2007, pp. 83–100.
[9] W. Yongqing, Z. Minxuan, Fully memory based address translation in user-level network interface, in: IEEE 3rd International Conference on Communication Software and Networks (ICCSN), 2011, pp. 351–355.
[10] D. Schmidt, N. Wehn, Dram power management and energy consumption: a critical assessment, in: Proceedings of the 22nd Annual Symposium on Integrated Circuits and System Design: Chip on the Dunes, ACM, 2009, p. 32.
[11] N. Ho-Cheung, C. Yuk-Ming, S.H. Kwok-Hay, Direct virtual memory access from FPGA for high-productivity heterogeneous computing, in: International Conference on Field-Programmable Technology (FPT), IEEE, 2013, pp. 458–461.
[12] A. Brandon, I. Sourdis, G. Gaydadjiev, General purpose computing with reconfigurable acceleration, in: International Conference on Field Programmable Logic and Applications (FPL), IEEE, 2010, pp. 588–591.
[13] R. Ammendola, A. Biagioni, O. Frezza, F. Cicero, A. Lonardo, P. Paolucci, D. Rossetti, F. Simula, L. Tosoratto, P. Vicini, Virtual-to-physical address translation for an FPGA-based interconnect with host and GPU remote DMA capabilities, in: International Conference on Field-Programmable Technology (FPT), IEEE, 2013, pp. 58–65.
[14] F. Shamani, V.F. Sevom, T. Ahonen, J. Nurmi, Design, implementation and analysis of a run-time configurable memory management unit on FPGA, in: International Symposium on System-on-Chip (SoC) & Nordic Chip (NORCHIP): Nordic Circuits and Systems Conference (NORCAS), 2015, pp. 1–8.
[15] Altera, Stratix V Device Handbook Volume 1: Device Interfaces and Integration, 2015, (https://www.altera.com/en_US/pdfs/literature/hb/stratix-v/stx5_core.pdf). Accessed on 07.06.2016.
[16] PowerPC Processor Reference Guide Embedded Development Kit, Xilinx, 2003, p. 570.
[17] J. Kylliäinen, J. Nurmi, M. Kuulusa, COFFEE - a core for free, in: Proceedings on International Symposium on System-on-Chip, 2003, pp. 17–22.
[18] C. Brunelli, F. Campi, J. Kylliäinen, J. Nurmi, A reconfigurable FPU as IP component for SoCs, in: Proceedings on International Symposium on System-on-Chip, 2004, pp. 103–106.
[19] COFFEE Core User Manual, 2007, (http://coffee.tut.fi/docs/COFFEE_Core_USER_MANUAL.pdf). Accessed on 02.03.2016.
[20] W. Wijesinghe, M. Jayananda, D. Sonnadara, Hardware implementation of random number generators, in: Proceedings of the Technical Sessions, vol. 22, 2006, pp. 28–38.
[21] I. Kuon, J. Rose, Measuring the gap between FPGAs and ASICs, IEEE Transac. Comput.-Aided Des. Integr. Circuits Syst. 26 (2) (2007) 203–215.
[22] S. Nouri, W. Hussain, J. Nurmi, Design and evaluation of correlation accelerator in IEEE-802.11a/g receiver using a template-based coarse-grained reconfigurable array, in: Nordic Circuits and Systems Conference (NORCAS): NORCHIP International Symposium on System-on-Chip (SoC), 2015, pp. 1–6.

Farid Shamani was born in Tehran, Iran, in 1983. He received his Bachelors degree in Computer-Hardware Engineering from Islamic Azad University-Central Tehran Branch (IAUCTB), Iran, in 2007. Shamani received his Masters degree with distinction from the Department of Digital and Computer Electronics from Tampere University of Technology (TUT), Finland, 2013. In 2013, he joined Professor Jari Nurmi's research group with the Department of Electronics and Communications Engineering at TUT. He is currently a Ph.D student with the Department of Electronics and Communication Engineering at Tampere University of Technology under the supervision of Prof. Jari Nurmi. His research interests primarily include Processor Design, Computer Architecture, Multicore/Manycore System-on-Chips and Digital Electronics Design, FPGA and DSP Implementations.

Vida Fakour Sevom was born in Mashhad, Iran, 1989. She received her B.Sc. in Biomedical Engineering majoring Bioelectric at Islamic Azad University of Mashhad, 2011. She received her M.Sc. degree in bioengineering from Tampere University of Technology (TUT), Tampere, Finland, in 2015. She joined Professor Nurmi's research group as a Research Scientist in 2014. Currently, she is working as a trainee in Nokia Technologies. At the same time, she is pursuing her Ph.D degree with the Department of Signal Processing, TUT. Her main research interests include computer vision, image and video processing, hardware software co-design and digital electronics design.



Tapani Ahonen is a Senior Research Fellow (Vanhempi tutkija) at Tampere University of Technology (TUT) in Tampere, Finland, where he has held various positions since 2000. Since 2004 he has been co-managing a group of about 30 researchers. He is a part-time Lecturer (Nebenberuflicher Lektor) at Carinthia Tech Institute (CTI) - University of Applied Sciences in Villach, Austria since 2007. In 2009-2010 Ahonen was a Visiting Researcher (Chercheur InvitÃ³e) at UniversitÃ³e Libre de Bruxelles (ULB) in Bruxelles, Belgium. His work is focused on proof-of-concept driven computer systems design with emphasis on manycore processing environments. Ahonen has an MSc in Electrical Engineering and a Ph.D in Information Technology from TUT. Positions of trust recently held by Dr. Ahonen include technical board member of the EU co-funded project Cutting edge Reconfigurable ICs for Stream Processing (CRISP), Finance Chair of the 2009 IEEE Workshop on Signal Processing Systems (SiPS), editorial board member and Guest Editor of the International Journal of Embedded and Real-Time Communication Systems (IJERTCS), and Program Co-Chair of the 2010 Conference on Design and Architectures for Signal and Image Processing (DASIP), He performs reviews for various international journals and participates in program committees of many high-quality conferences on a regular basis. Ahonen has an extensive international publication record including edited books and journals, written book chapters and journal articles, invited talks in high-quality conferences, as well as full-length papers and paper abstracts in conference proceedings.

D.Sc.(Tech) Jari Nurmi works as a Professor at Tampere University of Technology, Finland since 1999, in the Faculty of Computing and Electrical Engineering. He is working on embedded computing systems, wireless localization, positioning receiver prototyping, and software-defined radio. He held various research, education and management positions at TUT since 1987 (e.g. Acting Associate Professor 1991–1994) and was the Vice President of the SME VLSI Solution Oy 1995–1998. Since 2013 he is also a partner and co-founder of Ekin Labs Oy, a research spin-off company commercializing technology for human presence detection, now headquartered in Silicon Valley as Radiomaze, Inc. He has supervised 19 Ph.D. and over 130 M.Sc. theses at TUT, and been the opponent or reviewer of over 30 Ph.D. theses for other universities worldwide. He is a senior member of IEEE, member of the technical committee on VLSI Systems and Applications at IEEE CAS, and board member of Tampere Convention Bureau. In 2004, he was one of the recipients of Nokia Educational Award, and the recipient of Tampere Congress Award 2005. He was awarded one of the Academy of Finland Research Fellow grants for 2007–2008. In 2011 he received IIDA Innovation Award, and in 2013 the Scientific Congress Award and HiPEAC Technology Transfer Award. He is a steering committee member of four international conferences (chairman in two), and participates actively in organizing conferences, tutorials, workshops, and special sessions, and in editing special issues in international journals. He has edited 5 Springer books, and has published over 300 international conference and journal articles and book chapters.

Publication VI

FPGA IMPLEMENTATION AND INTEGRATION OF A RECONFIGURABLE CAN-BASED CO-PROCESSOR TO THE COFFEE RISC PROCESSOR

Farid Shamani, Vida Fakour Sevom, Tapani Ahonen, Jari Nurmi

Department of Electronics and Communications Engineering,
Tampere University of Technology
P.O.Box 553, FIN-33101, Tampere, Finland
firstname.lastname@tut.fi

ABSTRACT

This paper presents the design, implementation and integration of a reconfigurable Intellectual Property (IP)-based co-processor for the COFFEE RISC processor on a Field Programmable Gate Array (FPGA) device. The co-processor is capable of communicating with peripheral devices through the Controller Area Network (CAN) protocol. Although the main target of the co-processor is standard CAN frames, it is able to reconfigure itself in run-time to receive and decode the extended CAN frames. One of the main advantages of the co-processor is full compatibility with the COFFEE processor's requirements. Furthermore, the data transmission between the core and co-processor introduces no workload on the main data bus. In this implementation, the co-processor could achieve the maximum operating frequencies of 493MHz up to 825MHz in slow and fast timing models, respectively. As a result, the implemented co-processor gains (at least) 2.8x speed-up in comparison with the COFFEE processor by consuming a trivial dynamic energy. The ultimate achievement of this work is the new horizon expanded to the COFFEE processor to communicate with CAN-based peripheral devices which makes the core more versatile even for industrial purposes.

Keywords: FPGA Implementation, RISC Processor, Reconfigurable Co-processor, Controller Area Network.

I. INTRODUCTION AND MOTIVATION

Field Programmable Gate Arrays (FPGAs) enable digital system designers to implement embedded processors which are capable of executing specific functions [1]. The COFFEE RISC processor is one of such processors developed in our research group at Tampere University of Technology [2]. The COFFEE is a versatile embedded processor suitable for a variety of use cases including telecommunication, embedded computing, etc. Our new intention is to verify the versatility of the core for industrial purposes. In that sense, the first application is to control a heavy industrial machinery in such a place that the human's life is being jeopardized. In this situation, the technician controls the truck through an industrial joystick while the COFFEE processor will maintain the issued commands. For example, when the technician moves the joystick upward, the COFFEE processor should receive the bitstream, analyzes the data and sends the appropriate command to move the truck forward. Therefore, establishing a real-time and robust communication channel between the COFFEE core and the machine is an essential task. However, one of the main shortcomings of the core is the lack of a reliable protocol to communicate with such peripheral devices similar to the mentioned joystick. This task is maintained by a reconfigurable co-processor integrated to the COFFEE core via a dedicated co-processor bus. The co-processor is accounted to ensure data transmissions between the core and peripheral device over the Controller Area Network (CAN) protocol.

CAN is a serial communication in-vehicle protocol which was originally developed by Bosch company in 1985 [3]. The CAN protocol has attracted many researchers due to the innovation made in electronics communications. Previous electronic devices were connected to each other by point-to-point wiring system which significantly increases the total cost of the system. For example, a high-end luxury car has the potential to consume approximately 5 km of wiring with nearly 100kg of weight [4]. The CAN communication system replaced this wiring system by a simple twisted-pair physical medium [5]. Communicating over the CAN protocol also offers a real-time control system while preserving the security at a high-level [6]. This attribute fulfills the primary requirement of the COFFEE core.

Traditional methods to integrate a CAN-based product to the system were either as a standard CAN chip, a microprocessor with a built-in CAN interface or a custom Application-Specific Integrated Circuit (ASIC) with a CAN interface. Nowadays, with the advent of FPGAs, different types of Intellectual Property (IP)-based CAN interfaces can be integrated to the system on a single chip [7].

Although there are even open-source CAN controllers, we developed our CAN module as a co-processor for plenty of reasons of which *full compatibility* is the most appealing one. In addition, the target processor communicates with peripheral CAN-based device without introducing any additional workload on the main data bus, since the co-processor is integrated to the core via a dedicated co-processor bus. The main idea behind integrating over the co-processor bus is to keep the data bus as idle as possible.

In this paper, we have mainly concentrated on hardware considerations as well as the integration issues of the proposed co-processor with our target processor named COFFEE. Since COFFEE is a standard embedded RISC processor, this work shall also be adoptable in other similar platforms. The rest of this paper is organized as follows. In Section II some features of the CAN protocol are explained. Section III briefly describes the COFFEE RISC core. The major characteristics of the proposed co-processor as well as some fundamental knowledge about the CAN protocol are discussed in detail in Section IV. Section V explains hardware considerations alongside the FPGA implementation of the co-processor. Section VI presents the synthesis results, followed by conclusions in Section VII.

II. CAN PROTOCOL FEATURES

The CAN protocol has different application ranges from high speed networks to low cost wiring systems [8]. Although CAN protocol was first developed for vehicles, nowadays it is applied in other applications such as fuel systems, aircraft engines, medical equipment and cameras. CAN is a message-based protocol in which each message has its own prioritized address [9]. Modern CAN buses are flexible in a way that a new node can be added to the network without introducing additional cost to the system [6]. Having employed the CAN protocol offers variety of advantages to the system including: low-cost and light-weight physical medium, fast transmission time for the node, message priority, error detection and correction, etc [10]. In these systems, the bus value is either *recessive* or *dominant* where the dominant bit overrides the recessive bit. In our case study, the recessive bit denotes the logic value '1' and the dominant bit refers to the logic value '0'. More information with regard to the CAN-based systems can be found in [11].

III. COFFEE RISC CORE IN BRIEF

The COFFEE core is an open-source 32-bit RISC-based processor suitable for System-on-Chip (SoC) and embedded systems. Furthermore, the core is applicable to embedded computing in a wide range of applications including multimedia processing and communication purposes. Reusability and configurability are the two main characteristics of the core. The COFFEE processor is capable of executing 66 different instructions in a 6-stage pipeline from instruction set architecture point of view. Instructions can be executed in either 16-bit or 32-bit mode. Furthermore, four co-processors with different clock domains, which exploit a shared register bank (up to 32 registers), are attached to the core via a separate co-processor bus. The practical operating frequency of the best optimized implementation of the COFFEE is in the range of 100MHz while the speed-optimized version operates about 150MHz for a 90nm low power standard-cell technology [1]. Implementing COFFEE processor on a 28nm Altera Stratix-V FPGA board (5SGSMD5K2F40C2) has the cost of 4,503 Adaptive Logic Modules (ALMs), 5,439 registers with 172MHz maximum operating frequency. All the required materials, including the hardware description code, user manuals, the corresponding compiler, etc. can be found at the official website of the Tampere University of Technology in [12]. The latest version of the COFFEE processor is integrated with a Memory Management Unit (MMU) proposed in [13]. This version has not been fully released by the time of writing this manuscript.

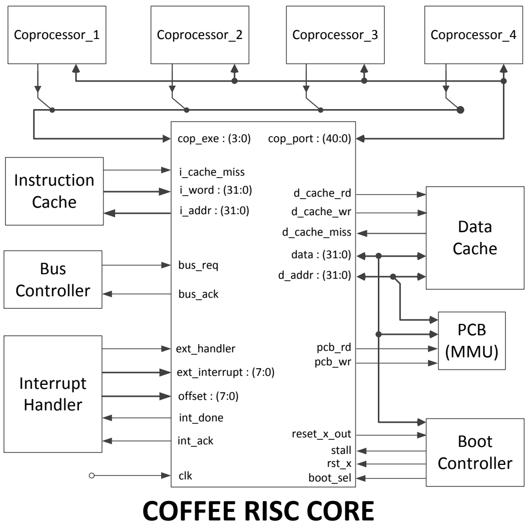


Fig. 1. The COFFEE RISC core integration

Fig. 1 depicts the interface of the COFFEE RISC core and also how it is integrated with other blocks. As the figure shows, there are four co-processor slots which are connected to the core through a shared bus. In this work, we employ the fourth co-processor slot to integrate our module to the main core.

IV. GENERAL CHARACTERISTICS OF THE CO-PROCESSOR

The developed co-processor is a reconfigurable IP-based module integrated to the COFFEE processor in order to provide communication facilities with peripheral devices through the CAN protocol. The co-processor is able to receive and transmit all the four type of frames used in a CAN-based system listed in the following:

- *Data Frame*: the main type of frame which contains the data.
- *Remote Frame*: the co-processor transmits this type of frame if it requires data from a certain node.
- *Error Frame*: as soon as the co-processor recognizes a misleading bit, it will broadcast the occurrence of an error to all nodes.
- *Overload Frame*: provides extra delay(s) between two successive data/remote frames (if required).

Fig. 2 presents the structure of a data CAN frame. Each frame starts with *start of frame* (SOF) bit, followed by a 11-bit *arbitration field* (also known as *identifier*). Arbitration field is a unique sequence of bits for each node which identifies the node, prioritizes the message and shows its logical address. The *control field* contains 7 bits indicating whether the frame is a remote frame, the arbitration field is extended, the following bit is reserved for future use and the last four bits contains the length of the data, respectively. There are two arbitration formats of 11 and 29 bits long. The frame with the 11-bit arbitration field is called *Standard Frame*, allows the system to address 2^{11} (2,048) different nodes. In some applications, the arbitration field can be extended to 29 bits, which is also known as *Extended Frame*, meaning that the system can handle up to 2^{29} (336,870,912) nodes. The *data field* contains 0 to 8 bytes of data, depending on the data length bits in the control field. Next comes the 16-bit *CRC field* which contains 15 bits Cyclic Redundancy Check (CRC) for checking the frame integrity, followed by a CRC delimiter bit. The *acknowledge field* contains two bits of which the first bit is the ack slot and the second one is the delimiter. The 7-bit *end of frame* (EOF) confirms that the frame has successfully delivered.

In this case study, we assumed that the co-processor only deals with standard frames. However, the co-processor has the ability to reconfigure its parameters to receive and transmit extended frames on demand. In addition, the co-processor always provides 64 bits data for the processor regardless of the data length. The extracted data is stored in two consecutive registers of the register bank shared with other co-processors. Therefore, the COFFEE processor should reference to the corresponding co-processor twice in order to fetch the entire 64-bit data. On the other hand, instructions referencing to any of the four co-processors, which are integrated to the COFFEE core, are not considered as an *atomic instruction* from the Instruction Set Architecture (ISA) point of view. It means that the core has the potential to refer to the co-processor in two non-consecutive references, e.g., in the case of an interrupt.

Start Of Frame	Arbitration Field	Control Field	Data Field	CRC Field	Acknowledge Field	End Of Frame

Fig. 2. CAN data frame

Hence, we designed the co-processor to ensure that the second non-consecutive reference to the register bank will be returned with the proper portion of the data. In this case, the co-processor not only ensures the data coherency, but also provides data integrity by not overwriting new values to the register bank. Furthermore, the co-processor can interrupt the COFFEE core in some particular cases, for instance, when its buffer is about to overflow. These are only two examples of *compatibility* which are offered by the proposed co-processor.

The co-processor is configured with the error management mechanism to detect and broadcast different type of errors which might have been happened to the frame during the packet transmission. In this regard, the co-processor is capable of identifying erroneous frames as follows:

- *bit-logic error*: which occurs when the transmitted bit has not been received with the same value that it was supposed to be received (e.g. the length of the data is more than 8 bytes).
- *form error*: where the structure of the frame is violated (e.g. wrong logic value for delimiters).
- *bit stuffing error*: happens when more than 5 consecutive bits of the same level have been detected (excluding end of frame field).
- *CRC error* where the CRC calculation is different than the received checksum.

Once the co-processor recognizes any type of above-mentioned errors, it will broadcast the error message by overwriting the bus with 6 consecutive dominant bits.

There are two strategies that the co-processor can choose to perform CRC calculation. One scenario is that the co-processor starts CRC calculation at once right after receiving the last bit of the CRC field. In this case, the co-processor will suffer from a massive computation workload, since the CRC calculation should be acknowledged in time (during the ACK slot). Another alternative is that the co-processor starts CRC calculation right after receiving the sixteenth bit. In this approach, CRC is calculated in several clock cycles instead of being calculated at once. In this implementation, the co-processor employs the second approach to compute CRC calculation in time.

V. THE FPGA IMPLEMENTATION AND INTEGRATION

Fig. 3 illustrates how the co-processor is integrated to the COFFEE core. Overall, the I/O ports of the co-processor are divided into two categories: the CAN-oriented ports at the right side of the entity as well as the COFFEE-oriented

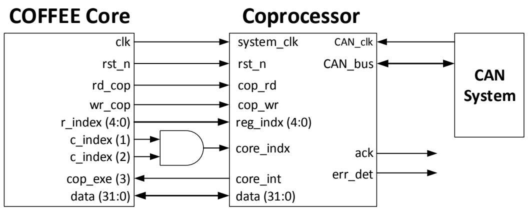


Fig. 3. The interface of the co-processor

Table I. The maximum operating frequencies as well as the worst critical path analysis

			Co-processor
Maximum Frequency	*Slow model*	85 °C	493 (MHz)
		0 °C	523 (MHz)
	Fast model	85 °C	733 (MHz) restricted to 673 (MHz)
		0 °C	825 (MHz) restricted to 713 (MHz)
Worst Critical Path Analysis (ns)	*Cell Delay*		0.825 — 40.54%
	Interconn. Delay		1.210 — 59.46%
	Data Delay		2.035 — 100%
	Clock Skew		-0.019
	Time Slack		0.071

ports at the left side. The co-processor has two different clock domains as well. The *CAN_clk* is the global clock derived by the CAN system. The co-processor negotiates with the CAN-based nodes by this clock. In addition to CAN_clk, the co-processor writes the extracted data to the appropriate registers where the processor communicates with the co-processor by using *system_clk*. Furthermore, as previously stated, the co-processor has the potential to detect the erroneous packet and broadcast the cause to other nodes *by overriding the CAN bus*. Therefore, the output ports ”ack” and ”err_det” with respect to the acknowledge slot and packet integrity violation are not physically used in our case study. The main idea of extracting these two output ports is for future use.

Technically, the COFFEE processor has been configured with three 32-bit buses for accessing the instruction cache, data cache and co-processors, respectively. From the ISA point of view, the COFFEE processor refers to each co-processor via movtc/movfc instructions for writing to and reading from a particular co-processor. The processor exchanges data with the co-processor through the available 32-bit bidirectional co-processor data bus. Signals wr_cop and rd_cop specify the direction of the data transmission. The 5-bit r_indx signal points to the specific register in the co-processor's register bank which is supposed to be written/read by the processor. In this case study, we stipulated that our target co-processor writes/reads its data to/from the last two registers of the register bank (R30 and R31). Furthermore, the processor references to each of the four available co-processors with 2-bit *c_indx* signal. Since we integrated our co-processor to the COFFEE via the 4^{th} co-processor slot, COFFEE should set c_indx to logic ”11” to reach to the corresponding co-processor. Indeed, this signal informs the co-processor that the main processor is referencing to its registers. Therefore, the co-processor can interrupt the core when, for example, the valid data has not been written to the registers yet. The co-processor interrupts the processor via the available interrupt signal (cop_exe(3)).

VI. SYNTHESIS RESULTS

The hardware implementation of the co-processor was in Very-high-speed integrated circuit Hardware Description Language (VHDL). The simulation tool used for preliminary verification was ModelSim software 10.2. We used Quartus-II 15.1 environment to synthesize the design and the target FPGA was an Altera Stratix V device of series 5SGSMD5K2F40C2.

Table I presents the worst critical path analysis as well as the maximum operating frequencies achieved by the co-processor at two different temperatures in fast and slow timing models. Accordingly, the co-processor operates at about 493MHz in slow model running at 85 °C and, consecutively, 523MHz at 0 °C. In fast model, the co-processor operates at 733MHz and 825MHz at 85 °C and 0 °C, respectively. However, both maximum operating frequencies in fast model are respectively restricted to 673MHz and 713MHz due to the limited toggling rate in the clock tree. The worst critical path analysis shows that about 60% of the total propagation time are made by interconnection delays whereas the remaining 40% are introduced by the cells themselves. The negative clock skew reveals that the co-processor can operate at slower frequencies. Eventually, co-processor achieved 0.071 ns positive time slack, meaning that the design could roughly meet the timing constraints introduced to the analyzer. Overall, the co-processor not only offers (at least) 2.8× speed-up in comparison with the COFFEE core (only in terms of operating frequency), but also it massively reduces the target processor's workload by receiving the packet, decoding it and extracting the data in hardware. Otherwise, the software should take care of decoding the packet. Furthermore, the co-processor guarantees that the processor will reference to the correct registers inside the co-processors' register bank in some specific cases (e.g. two non-consecutive accesses to fetch the data).

Table II depicts the synthesis results in terms of logic utilization, total used registers and energy consumption for a successful data delivery to the COFFEE core. As pre-

Table II. Summary of the implementation details

	Co-processor
Logic Utilization	321 (ALMs) — 7.12% overhead
Total Registers	441 — 8% overhead
Static Energy	327.33 (μJ)
Dynamic Energy	11.19 (μJ)
I/O Energy	10.36 (μJ)
Total Energy	348.88 (μJ)

viously mentioned, the baseline COFFEE implementation employs 4,503 ALMs along with 5,439 registers. Based on the achieved results, integrating the co-processor to the COFFEE processor introduces 321 more ALMs, i.e. 7.12% overhead, to the core along with 441 additional registers (8%). Energy consumption is analyzed using PowerPlay Power Analyzer tool in Quartus-II software. The analyzer directly reads the Switching Activity Interchange Format (SAIF) generated by the ModelSim for the co-processor running at 450MHz. Static probability and toggle rate for each signal were reported based on the generated Value Change Dump (VCD) file for a successful packet delivery composing of packet detection, frame integrity, data extraction and handing over the data to the main processor. From the Table II, it can be observed that the main source of energy consumption is the static energy (about to 328μJ) which is mainly dissipated within the unused portions of the chip. On the other side, the dynamic energy consumption, is only 11.2(μJ), i.e., 3.2% of the total energy. In addition, the co-processor consumes about 10.4(μJ) energy in its inputs and outputs.

In order to compare the achieved results with an ASIC model of the proposed co-processor, Kuon et al. have provided a wide study which compares the performance gap of a particular design between a 90nm ASIC and a 90nm FPGA in [14]. Base on their claims, the design on a 90nm ASIC implementation is 35$\times$ smaller and 3.4$\times$-4.6$\times$ faster compared to the same design on a 90nm FPGA. Furthermore, the 90nm ASIC implementation consumes 14$\times$ less dynamic power alongside 87$\times$ less static power. Nouri et al. in [15] claims if we take 60% speed up into consideration, the same gap ratios are expected for a design on a 28nm ASIC and a 28nm FPGA. By taking above-mentioned scenario into account, we can have a rough estimation on the hardware costs of the proposed co-processor on a 28nm ASIC design.

VII. CONCLUSION

In this paper, we presented an FPGA implementation and integration of a co-processor to the COFFEE RISC core in order to establish a Controller Area Network (CAN)-based communications channel between the processor and peripheral devices. The peripheral device was an industrial joystick in this study. The co-processor is an Intellectual Property (IP)-based module which could reconfigure itself to transmit and receive both standard and extended CAN frames. The main advantage of the co-processor was full compatibility with the COFFEE core's requirements. For instance, the co-processor could guarantee the data integrity in the case of two non-consecutive references from the processor to fetch the data. Another advantage of the proposed co-processor (in comparison with other available IP cores) was the fact that it could be integrated over the co-processor bus of the COFFEE processor, instead of using the main data bus. This would help to minimize the congestion on the data bus as well as keeping the bus as idle as possible. The synthesis results were presented in terms of hardware costs, maximum frequency, energy consumption, etc. The co-processor (operating at 493MHz) operated at least 2.8x faster than the COFFEE processor while introducing 7%-8% overhead to the hardware cost along with a trivial energy consumption. The main achievement of presented implementation was the new ability of the target processor to communicate with CAN-based peripheral devices. This integration improved the target processor's versatility even for industrial purposes.

ACKNOWLEDGMENT

This work was funded by the Academy of Finland under contract number 258506 (DEFT: Design of a Highly-parallel Heterogeneous MP-SoC Architecture for Future Wireless Technologies). Portions of the work were also funded by the ARTEMIS JU under grant agreement number 295371. The first author gratefully acknowledges the grant support awarded by Finnish Cultural Foundation as well as the grant support by Jane and Aatos Erkko Foundation under the project Biological Neuronal Communications and Computing with ICT.

REFERENCES

[1] J.P. Kylliäinen; T. Ahonen; J. Nurmi, "General-Purpose Embedded Processor Cores – The COFFEE RISC Example" in: J. Nurmi, "Processor Design: System-On-Chip Computing for ASICs and FPGAs", 1st ed., Springer Publishing Company, pp. 83-100, 2007, 513 p.

[2] Kylliäinen, J.; Nurmi, J.; Kuulusa, M., "COFFEE - a core for free", in Proceedings of International Symposium on System-on-Chip, 19-21 Nov. 2003, pp. 17-22, DOI: 10.1109/ISSOC.2003.1267707.

[3] K.H. Johansson; M. Törngren; L. Nielsen, "Vehicle Applications of Controller Area Network", in: D. Hristu-Varsakelis; W.s. Levine, "Handbook of Networked and Embedded Control Systems", Birkhäuser, Boston, pp. 741-765, 2008.

[4] M. Khanapurkar; P. Bajaj; S. Dahake; H. Wandhare, "Design Approach for VHDL and FPGA Implementation of Automotive Black Box using CAN Protocol", International Journal of Computer Science and Network Security (IJCSNS'08), VOL.8 No.9, September 2008.

[5] B. Donchev; M. Hristov, "Implementation of CAN Controller With FPGA Structures", Proceedings of the 7th International Conference The Experience of Designing and Application of

CAD Systems in Microelectronics (CADSM), 2003, pp. 577-580, DOI: 10.1109/CADSM.2003.1255163

[6] ”Controller Area Network(CAN), Overview”, white paper, National Instruments, 2014, Available at http://www.ni.com/white-paper/2732/en/

[7] D. Leu, ”FPGA-Based CAN Solutions”, Inicore Inc, 2005, 6 p.

[8] ”Bosch CAN Specification”, Ver. 2.0, Robert Bosch GmbH, Germany, 1991.

[9] R. I. Davis and N. Navet, ”Controller area network (CAN) schedulability analysis for messages with arbitrary deadlines in FIFO and work-conserving queues”, 9th IEEE International Workshop on Factory Communication Systems (WFCS), Lemgo, 2012, pp. 33-42. DOI: 10.1109/WFCS.2012.6242538

[10] M. Farsi; K. Ratcliff; M. Barbosa, ”An Overview of Controller Area Network,” in Computing & Control Engineering Journal, vol. 10, no. 3, June 1999, pp. 113-120, DOI: 10.1049/cce:19990304

[11] W. Lawrenz, ”CAN System Engineering From Theory to Practical Applications”, 2nd ed., Springer, 2013, 353 p, DOI: 10.1007/987-1-4471-5613-0.

[12] ”COFFEE™ RISC core - a COre For FrEE”, Tampere University of Technology, Finland, [WWW], accessed on 02.Mar.2016, available at http://coffee.tut.fi.

[13] F. Shamani, V. F. Sevom, J. Nurmi and T. Ahonen, ”Design, Implementation and Analysis of a run-Time Configurable Memory Management Unit on FPGA,” Nordic Circuits and Systems Conference (NORCAS): NORCHIP & International Symposium on System-on-Chip (SoC), 2015, Oslo, 2015, pp. 1-8, DOI: 10.1109/NORCHIP.2015.7364375

[14] I. Kuon and J. Rose, ”Measuring the Gap Between FPGAs and ASICs”, in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 26, no. 2, Feb. 2007, pp. 203-215, DOI: 10.1109/TCAD.2006.884574

[15] S. Nouri, W. Hussain and J. Nurmi, ”Design and Evaluation of Correlation Accelerator in IEEE-802.11a/g Receiver Using a Template-Based Coarse-Grained Reconfigurable Array”, Nordic Circuits and Systems Conference (NORCAS): NORCHIP & International Symposium on System-on-Chip (SoC), 2015, Oslo, 2015, pp. 1-6, DOI: 10.1109/NORCHIP.2015.7364374

Tampereen teknillinen yliopisto
PL 527
33101 Tampere

Tampere University of Technology
P.O.B. 527
FI-33101 Tampere, Finland

ISBN 978-952-15-4009-7

ISSN 1459-2045