UNIVERSITY OF CALIFORNIA
SANTA CRUZ

DEMIS: DYNAMIC EMI SHIFTING

A dissertation submitted in partial satisfaction of the
requirements for the degree of

DOCTOR OF PHILOSOPHY

in

COMPUTER ENGINEERING

by

Daphne Irene Gorman

June 2018

The Dissertation of
Daphne Irene Gorman is approved:

Professor Jose Renau, Chair

Professor Matthew Guthaus

Professor Heiner Litz

Tyrus Miller
Vice Provost and Dean of Graduate Studies

Table of Contents

List of Figures

List of Tables

Abstract

DEMIS: Dynamic EMI Shifting

by

Daphne Irene Gorman

Processors emit non-trivial amounts of electromagnetic radiation, creating interference at frequencies that are used by wireless communication technologies such as cellular, WiFi, and Bluetoooth. I introduce the problem of in-band radio frequency interference as a form of electromagnetic interference (EMI) to the computer architecture community as a technical challenge to be addressed.

My research is the first to provide insights in the new area of dynamically shifting the EMI generated by a processor by evaluating several platforms and showing the EMI is sensitive to many architectural and compilation parameters through exhaustive measurements. Using these measurements, I propose the new idea of Dynamic EMI Shifting (DEMIS), where architectural and/or compiler changes allow the EMI to be shifted at runtime by a processor. DEMIS processors dynamically move the interference from frequency bands used during communication to other, unused frequencies. Unlike previous works that leverage static techniques, DEMIS dynamically targets specific frequency bands; the type of techniques used in my research are only possible from an architectural perspective.

Despite the fact that the EMI generated by a processor is deterministic, modeling the EMI has proven to be a complex challenge. Moreover, EMI has been shown to be layout dependent (affected by the location of functional units on the chip and the lengths of the wires)

and binary dependent (affected by not only the application but also on the compilation options). I propose a Model for EMI from an SoC (MESC), a framework for modeling electromagnetic emissions from a core. MESC takes into account some layout information and the switching activity of a process to model the expected EMI emitted by an SoC. I validate MESC against a core running on an FPGA. My evaluation shows that MESC is able to predict EMI within 95% accuracy across time and across the frequency spectrum, even when using statistical sampling to obtain activity rates.

Using the results from MESC, I am able to propose that two different layouts of a single RTL can be leveraged to dynamically shift EMI using my technique EMI Core Hopper (EMI CHopper), a layout-based implementation of DEMIS. EMI CHopper uses a multi-core system, where each core has a different layout but the same RTL, and utilizes thread migration to have an application "hop" between cores to reduce in-band EMI on the fly. Leveraging MESC, EMI CHopper reduces in-band EMI by up to 50%, with only a small performance impact.

In order to implement DEMIS for existing systems, I propose utilizing higher level techniques, such as compiler optimizations and clock speeds. My evaluation over real systems shows a decrease of in-band EMI ranging from 3 to 15dB, with less than a 10% average performance impact. A 15dB EMI reduction for LTE can represent a bandwidth improvement of over 3 times for EMI bound communication.

The research presented in this thesis offers a new, architectural perspective on a prevalent problem in wireless communications. My findings based purely on architectural techniques show significant promise in improving wireless communications, and the tools I described offer

many opportunities for further research.

Acknowledgments

I would like to thank my committee for their valuable feedback and help throughout my graduate experience. In particular, my advisor Prof. Jose Renau who has provided me with help, guidance, and probably an unnecessary amount of leniency which allowed me to pursue this area of research.

My wonderful labmates in the MASC lab, especially Rafael, Gabriel, Elnaz, and Blake, encouraged me and kept me sane during crunch time before deadlines. Without these teammates and my other friends' help throughout the years, which ranged from working through a particularly complex piece of code with me to telling me to take a break and relax, I would never have made it through graduate school.

Last but not least, I want to thank my parents and sister for always being there for me and being supportive of my educational pursuits for many years. It was only through their influence that I was able to become the person I am today. Thank you for everything!

Chapter 1

Introduction

Communication breakdown

It's always the same

I'm having a nervous breakdown

Drive me insane!

Led Zeppelin

The advances in integrated circuits and digital devices in recent years have brought to market a growing number of connected devices, which are getting smaller each generation. To meet commercial and technological goals, the space being allocated to electronic components is only a fraction of the already small device size, which requires tighter integration. However, a processor in a SoC can produce electromagnetic interference (EMI) that can disrupt wireless communication, reducing the effectiveness of the device.

As mobile devices become more ubiquitous and wireless communication technologies become more diverse, we want our mobile devices to be able to interact with a wide variety

of communication technologies. I present to the computer architecture community the problem of in-band EMI caused by the processor. That is, when a processor is running, it may be emitting Radio Frequency (RF) interference, or EMI, at the same frequencies the device is using for wireless communication, causing interference. Unfortunately, the Shannon-Hartley theorem (explained more extensively in Section 2.1) states that this interference constrains the speed that data can be sent wirelessly.

Computer processors emit measurable amounts of electromagnetic radiation; enough that the EMI can be exploited as a security risk [15, 16]. In addition to being a security concern, EMI can also interfere with wireless communication, which is the focus of the work presented in this thesis. Even when running just the operating system, the processor will emit radiation at some frequencies. The EMI produced by a processor can create significant desensitization of the antenna, and is a well-known obstacle in the field, which many different publications and patents are dedicated to addressing [6, 21, 33, 38]. Currently, one way to mitigate this problem is to place the computer processor as far away as possible from the antennas [21, 35] to minimize the effect of the processor's EMI. However, modems are placed as close as possible to the antennas in order to mitigate losses before the signals are processed. With the introduction of more integrated chips such as the Snapdragon series [42], some of which include wireless communication processors, designers may no longer be able to physically separate the CPUs and the antenna in this way. This problem also tends to worsen with the emergence of smaller, more integrated devices such as wearables, as processors and RF components (particularly antennas) can no longer be physically separated. This is because even though the transmitted power may be less, the power recieved decreases linearly with transmit power and quadratically with dis-

tance, which will be discussed more in depth in Chapter **??** during my analysis of Equation 2.4.

Much of this work is based on the observation that wireless communication systems use many frequency bands, but not all the bands are used simultaneously. This observation can alleviate the proposed problem by moving RF interference out of the bands being used for communication in a given time.

To illustrate this potential, Figure 1.1 shows the interference level captured from a spectrum analyzer for a Exynos 5422 processor (a $28nm$ chip running with a clock frequency of $2GHz$) running SPEC2006 hmmer application. This type of plot has frequency in the x-axis and radiated power in dBm in the y-axis. The higher the radiated power the higher the EMI. The plot shows a SuperWiFi XR7 frequency band. From a communication point of view, we want the lowest possible amount of noise. Each line shows a different compilation option for hmmer from the SPEC2006 benchmark suite. The second option has a small 3% performance impact, but it has close to $6dB$ (or $4\times$ power) reduction of in-band noise. A more in depth analysis of this measurement will be provided in Chapter 3.

Since workload, compiler, architecture, and layout are necessary for understanding and addressing this problem, this belongs to the classical architectural domain where impact on execution performance and the interface between compiler/architecture/VLSI layers have different trade-offs. Computer architects are uniquely suited to provide solutions for minimizing in-band EMI from the processor.

Despite the fact that EMI is deterministic, there are no usable tools or methodology for modeling EMI for a given application, particularly at the micro-architecture level. This is because, despite its consistency, there are numerous factors that contribute to the EMI a proces-

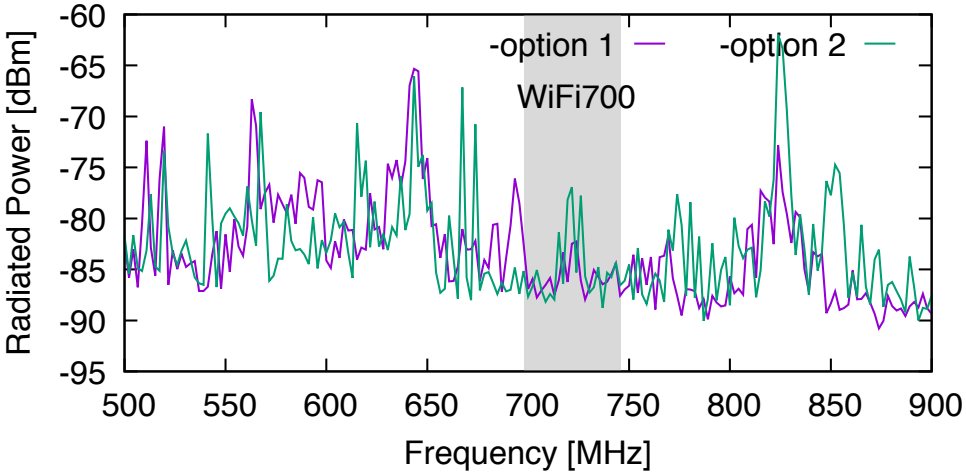

Figure 1.1: Changing compile options for the SPEC2006 hmmer benchmark on an ODROID-XU4 (Exynos 5422 chip) causes a significant difference in noise in the SuperWiFi XR7 frequencies (grey) with only a small change in performance.

sor produces. EMI can be theoretically modeled as a combination of electromagnetic radiation being produced by alternating current in each wire of a processor. In any modern design, the complexity both in number of wires and in the data patterns on them do not make this prediction an easy task. The use of a full-wave simulation software [3, 24, 43] would provide accurate results, but takes an extremely long time, as the processor's layout would have to be taken into account as well as the activity of each wire. Also, because GDSII is necessary for this type of simulation, these tools are unusable until just before tapeout, which may be too late for significant design changes. To some extent this is similar to requiring a full SPICE simulation for a whole chip, which is not really feasible. However, the extreme accuracy provided by these full-wave simulators is most likely unnecessary for addressing the problem of processor-generated interference on wireless technologies, since compensation mechanisms, such as redundancy and error-correcting codes, already exist for wireless communication and make "interference-free"

communication unnecessary. Designers lack usable models, and these models would be too complicated to actually run. Typically, designers do not realize they have an EMI problem until after a system has been prototyped and evaluated, which puts them in the unfavorable position of having to apply costly and usually ineffective patches, or going back and redesigning the system [21].

Furthermore, the EMI is also application/binary dependent, so the same layout will produce different EMI depending on the program executing. This adds additional challenges for designers, as it is difficult to account for the multitudinous, essentially infinite, applications a processor may execute. Then, modeling the EMI will necessarily imply characterization through specific benchmarks that can represent the EMI of a large range of applications. Thus, any model for EMI must be flexible enough to account for different processes.

In this thesis, I introduce the problem of in-band RF interference as a form of EMI generated by a processor, with an emphasis on micro-architecture. Included is a thorough exploration of the problem space as I analyze the effects specific architectural parameters have on EMI, as well as the concept of Dynamic EMI Shifting (DEMIS) to reduce in-band interference. Additionally, this thesis includes a technique for modeling the EMI a processor will generate as it executes an application and two implementation of DEMIS.

Chapter 2 provides background information relevant to my research. First, I provide a short description of wireless communication and some specific technologies. Then, I briefly cover some of the reasons EMI is produced in processors before providing a survey of other computer architectural research that focuses on EMI.

In Chapter 3, I provide an in-depth exploration of the architectural problem space

as it affects in-band EMI. This analysis consists of manipulating specific architectural and compilation parameters and measuring the resulting changes in RF interference.

MESC, the first Model for EMI from an SoC, is proposed in Chapter 4. MESC takes into account some basic layout information of a processor and the activity rates of individual nets for a process in order to approximate the expected EMI from that process.

Building on the model described in Chapter 4, Chapter 5 proposes the DEMIS technique for reducing in-band EMI emitted by a processor, EMI CHopper (EMI Core Hopper). EMI CHopper reduces EMI by causing processes to "hop" between cores that have the same RTL but different layouts.

The DEMIS technique for reducing in-band EMI, for existing processors is proposed in Chapter 6. This DEMIS implementation consists of shifting the EMI at runtime by changing architectural and/or compiler parameters during execution time.

Finally, Chapter 7 concludes with the significance and main findings of my research as well as some opportunities for future work.

Chapter 2

Background

> Get your facts first, then you can distort them as you please.
>
> Mark Twain

This chapter covers some background information about EMI, how it is produced in a core, and other useful information. I put special focus on the fact that communication does not use the entire frequency spectrum at a given time, but rather uses a single band. Thus, a processor does not need to minimize the EMI over the entire frequency spectrum, but rather just the frequencies being used for communication at that time.

Additionally, I will cover some of the current techniques and tools being used by architects today concerning EMI.

2.1 Wireless Communication Technologies

My work deals with the electromagnetic radiation from the processor that causes interference for the device's wireless communication. Since EMI may be an unfamiliar topic to parts of the computer architecture community, this section aims to provide an overview of the main concepts and technologies involved. This is not meant to be a complete introduction to the subject.

Wireless communication is an important feature of most modern computational systems, particularly for mobile devices. Wireless communication usually relies on allocating a specific frequency band in which data is transmitted according to a technology-specific protocol. Regardless of which specific protocol is being used, the theoretical amount of data that can be transferred through a channel depends on a certain number of factors, like the bandwidth and the Signal to Noise Ratio (SNR). This relation is governed by the Shannon-Hartley Theorem,

$$C = B * log_2(1 + \frac{S}{N}), \tag{2.1}$$

which states that the channel capacity (C), or the theoretical upper bound on the net bit rate, is affected by the bandwidth (B) and the SNR ($\frac{S}{N}$). Since SNR is the quotient between noise and signal strength, the lower the noise the higher the channel capacity. Figure 2.1 shows a visualization of the Shannon-Hartley Theorem for typical values for the LTE standard for cellular communication, with a bandwidth of nine megahertz.

Therefore, for a fixed bandwidth and signal strength, increasing the amount of noise will reduce the total channel capacity available, reducing the total amount of data that could be transmitted through that channel. Throughout my research, I explore the noise emitted by the

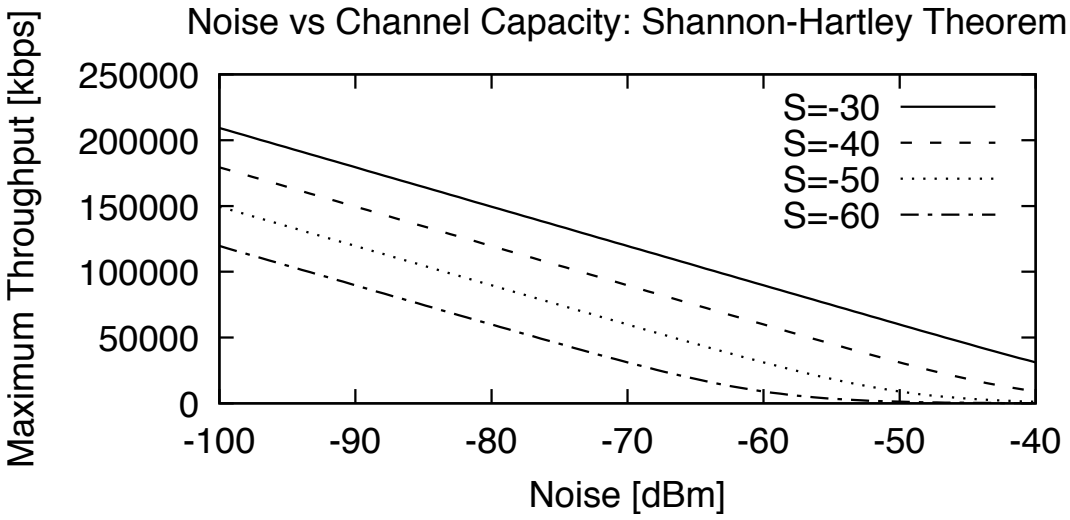

Figure 2.1: The amount of noise drastically changes the maximum theoretical throughput at a particular frequency at different signal strengths (S = signal strength in dBm).

processor that could interfere with the wireless communication technologies in a device.

The focus of this work is to improve the SNR by reducing the noise. Particularly, I focus on reducing the noise caused by processor generated EMI. Increasing the SNR will result in an improved channel capacity.

Unfortunately, EMI is produced any time current is present in a system, and thus computer processors constantly emit non-trivial amounts of noise. Fortunately, wireless communication is dependent on specific frequencies, and thus a device only needs certain frequency bands to be less noisy at a given time. Common sources of noise are the clock, its harmonics, memory accesses, and even power converters; these will be discussed later.

Table 2.1: Wireless technology frequency ranges and maximum bandwidth used at a given time.

Network	Range (MHz)	Max Channel Bandwidth (MHz)	Max Bandwidth (MHz)
LTE	698–5925	20	18[1]
Bluetooth	2400–2485	1	1
WiFi	2400–2500	22	40
WiFi 5GHz	5150–5850	160	160
Super WiFi	54–790	5	32
WiFi HaLow	755–928	4	16

2.1.1 LTE

LTE (Long Term Evolution) is the current standard for cellular communication and was developed with the main objective of improving the communication rates to and from a cell phone. However, with the invention of smartphones, the noise emitted from the phone's CPU can interfere with communication. This section provides a brief introduction to cellular communication technology LTE with a focus on components related to my research. LTE (and wireless communication in general) is commonly divided into frequency bands allocated to different users and uses.

The band selection is determined by many factors such as signal strength and band congestion and is usually out of the control of the user, since it is mostly defined by the tower, but higher-frequency bands are usually preferred since they can send more data.

Different times and situations may require different LTE frequency bands to be clear. For example, low frequencies can permeate solid objects better, so when using LTE inside, we would prefer to use a low-frequency band such as band 12 (699 to 716MHz). However, higher

[1]LTE can utilize a total of 18 MHz, but the frequencies do not necessarily have to be contiguous. Future LTE technologies may expand this 18 MHz to 19.8 MHz, as the largest channel bandwidth is actually 20 MHz, but the edges of the band remain unused in order to prevent noise in adjacent bands.

frequencies can send more data, so when outside, it would be better to use a higher frequency band such as band 4 (1710 to 1755MHz).

In order to connect to a network tower, a device utilizes two different radio links, one for uplink (where the data goes from the device to the tower) and one for downlink (tower to device) [22]. Most LTE bands are Frequency-Division Duplex (FDD) meaning that the uplink (UL) and downlink (DL) connections operate at different frequency bands between 10 and 400MHz apart, thus allowing them to happen at the same time [1], meaning communication happens at multiple frequencies simultaneously. However, some LTE bands use Time-Division Duplexing (TDD), which means that UL and DL occur at the same frequency, but at different times.

Depending on which network carrier a device is utilizing, it accesses to different LTE bands. Currently in the US, there is no overlap between the bands provided by AT&T (bands 2, 4, 5, 17) and Sprint (bands 25, 26, 41). Furthermore, different countries use different bands as well. For example, the US and most of Europe tend to use FDD bands, whereas China tends to use TDD bands. Thus, the amount of viable frequencies for an LTE connection is largely dictated by which cellular network provider a device is using.

Cell networks can trigger a band switch when the EMI interference is high. An infrequent noisy spike at a certain frequency is enough to drop a call or lose data, but is not necessarily enough for a cellular network to switch frequency bands. On the other hand, if some noise does trigger a band switch, it is likely that the mobile device is now operating on a slower band. For example, when using the T-Mobile network in the US, a device may be using band 4 because it had less interference and congestion, but because it is currently running a

process that creates in-band noise, the device will switch the noisier and more congested band 12. If the noise generated by the device were out of the band 4 frequencies, the device would have continued to operate on the higher band.

A noisy band may lead the network to trigger a band switch. The noise may be due to congestion, interference with other networks or simply because the CPU on the cell phone is emitting EMI. In my work, I show that architectural parameters can be manipulated to reduce the amount of in-band EMI while communication is happening to reduce interference from the core to the antenna.

The implication for my research is that to improve LTE bandwidth, only one or two bands of communication need to be cleared. The band to be cleared is not known a priori, and thus a dynamic band EMI management has high potential benefits.

2.1.2 Bluetooth

Another technology that is commonly implemented in mobile devices and could suffer from processor EM radiation is Bluetooth. Bluetooth operates at $2.4GHz$, and the entire Bluetooth spectrum spans $83MHz$. Each of the 79 Bluetooth channels has a bandwidth of $1MHz$ [10]. The reason for so many channels is that Bluetooth utilizes a Frequency Hopping Spread Spectrum (FHSS) technique, which utilizes each channel in a preset sequence negotiated by both the master and slave when they are first connected. The channel hopping occurs regularly according to that predetermined sequence known to both the transmitter and the receiver.

Bluetooth's high frequency limits the communication distance and also makes it more

susceptible to interference. Therefore, it is more crucial to prevent noise from a nearby source, such as that device's own CPU. Like in LTE, DEMIS and EMI CHopper can further reduce interference focusing on the bands being used at the moment.

2.1.3 WiFi and WLAN

Finally, another ubiquitous technology that may suffer from EMI from the CPU is WiFi, present in virtually every mobile device on the planet. WiFi operates at both the $2.4GHz$ and $5GHz$ (802.11a, n, and ac) frequencies. Older WiFi protocols use frequency hopping (similar to Bluetooth) or spread spectrum transmissions, while newer versions use Orthogonal Frequency Division Multiplexing.

Additionally, the 802.11af standard (also known as White-Fi or Super WiFi) operates in the 54 to $790MHz$ range (in bands licensed for TV, VHF, and UHF) and has been in use since 2014. White-Fi uses frequency channels with bandwidths ranging from 6 to $8MHz$, and can use up to four channels at once in one or two contiguous blocks.

Although operating at different bands and having different bandwidths, WiFi also divides the spectrum in different bands, dynamically assigned to each device. The communication speed of WiFi and WLAN can also be reduced due to the presence of in-band noise created by the CPU, and thus could also be improved by DEMIS and EMI CHopper.

2.2 EMI Causes in Processors

From an RF perspective, any wire with a time dependent current passing through it behaves as an antenna. Traditionally, in integrated circuit design, one of the only concerns is capacitive coupling, and sometimes wires being subject to bit-flips within a wire, which can cause data corruption and/or invalid behavior in the circuit. In this work, I look into wires as transmitters. I am thus interested in the EMI being emitted by each wire with respect to the wireless communication.

2.2.1 EMI as an effect of processor layout

In general, the power and direction of the radiation depend on the form of an antenna and on the distance from which the interference is being measured. For instance, for very long wires, ($L \gg d$, where L is the wire length and d is the distance of interest), the radiation occurs uniformly throughout the wire axis, varying only with distance. In that particular case edge effects are usually ignored. Another example is with closed loop antennas, where the radiation is directional and perpendicular to the loop plane. In the specific case of interest of my research, L is a length within a die and d is a distance within a device, thus usually $d \gg L$. Also, there is a large number of wires that will act as an antenna array, possibly emitting at multiple frequencies, all with different magnitudes. The resulting EMI will, thus, be a combination of all the EMI emitted by each wire.

For a real system, the wireless communication antenna will be located in the same plane as the chip, since mobile devices tend to be relatively flat. This can reduce the EMI

observed by the antenna with regards to the maximum EMI observed in a plane parallel to the die plane. For the sake of simplicity, my model takes into account the maximum EMI, which is a more conservative approach since it is not known at design time which will be the direction of the die in respect to the antenna(s) in the device.

There are a few antenna equations that are relevant to my work regarding a processor layout's effect on interference, but the most relevant is the basic equation for antenna gain,

$$G = \frac{4\pi A}{\lambda^2}. \tag{2.2}$$

Equation 2.2 states that the gain of an antenna (G), or power multiplier, is directly proportional to the aperture, or effective area, A. While this works for aperture antennas, we also must consider linear antennas, for example a short dipole, which has a total radiated power of

$$P_{total} = \frac{\pi}{12} I_0^2 Z_0 (\frac{L}{\lambda})^2, \tag{2.3}$$

where I_0 is the current amplitude (the signal fed into the dipole), Z_0 is the admittance of free space $\approx 376\Omega$, and L is the length of the dipole, which to qualify as "short" must satisfy $L \ll \frac{\lambda}{10}$. Another important formula, known as Friis Transmission Formula,

$$P_R = \frac{P_T G_T G_R \lambda^2}{(4\pi d)^2} = \frac{P_T G_T G_R c^2}{(4\pi d f)^2}, \tag{2.4}$$

states that the received power P_R is the product of the transmitted power P_T multiplied by the TX and RX gains G_T and G_R multiplied by the wavelength squared (or multiplied by c^2 and divided by the frequency squared f^2) all divided by 4π times the distance between the two antennas d. This means that the received power is proportional to λ^2 or inversely proportional to f^2. In

decibels, this formula is

$$P_R = P_T + G_T + G_R + 20\log_{10}(\frac{\lambda}{4\pi d}). \tag{2.5}$$

These equations [36] are used throughout my research, particularly in designing MESC.

One thing that is important to note is that most EMI measurements are taken in the frequency domain. That is, EMI is typically depicted as the amount of power being radiated at each frequency. Most of the measured and modeled plots in this thesis are also in the frequency domain. In my work, I utilize the Fast Fourier Transform (FFT) to convert samples taken over time (in the time domain) into the frequency domain. One important aspect of taking an FFT over a set of time-domain values is that the FFT can only provide values for frequencies up to half of the sampling frequency, the Nyquist frequency. Therefore, to get the frequency domain up to a frequency f, the sampling rate need to be $2f$. As the primary objective in this work is to model and then reduce in-band EMI, using the frequency domain is a good way to visualize and isolate specific frequencies, as opposed to trying to determine frequencies from periodicity in time-domain plots.

The equation for the FFT is

$$X_k = \sum_{n=0}^{N-1} X_n \times e^{i2\pi nk/N}, \tag{2.6}$$

where X_n are time domain samples, X_k are frequency domain samples, N is the number of time domain samples, and k spans the integers from 0 to $N-1$. To get back into the time domain from the frequency domain, the Inverse FFT (IFFT),

$$X_n = \frac{1}{N} \times \sum_{k=0}^{N-1} X_k \times e^{i2\pi nk/N}, \tag{2.7}$$

can be used.

RF Integrated Circuits (ICs) are vulnerable to on-chip in-band interferers [5], and may contain circuit-level RF noise couplings that would have a significant impact on system-level performance of wireless communication performance. Unfortunately, finding the cause of interferers is extremely complicated and have so many technical aspects, that achieving a reliable estimation using computer simulations is impossible.

2.2.2 Layout and Metal

To apply these RF principles to a processor, I work under the assumption that metal wires in the processor act as "antenna elements" or "radiators" that emit EMI. One fundamental rule is that the radiated power of an antenna is proportional to the antenna aperture, or the effective size. This means that if an antenna has twice as much metal, we would expect twice as much power to be radiated, as the gain has doubled (see Equation 2.2). Applied to a processor layout, a net that is twice as long or twice as wide should contribute twice as much EMI. Furthermore, long traces may add inductance, which could increase the EMI.

Additionally, the EMI power is proportional to the integral of the current distribution along the antenna. Basically, this means that areas on the chip that have higher current will radiate higher power EMI. If a circuit contains a loop, that loop may couple with an external magnetic field and act as a strong transmitting antenna. However, this effect can be minimized by reducing the size of the loop. Also, loops are not widespread in chip designs due to inductive effects that are usually undesirable.

2.2.3 Square Waves

As my research focuses focuses on digital processors, a basic understanding of the EMI of square waves is required, as they have some unique properties that are not shared with simple sinusoidal waves.

One important observation about square waves is that square waves produce strong odd harmonics and weaker even harmonics. That is, for a square wave for a frequency of f we would expect large amounts of power at f, $3f$, $5f$, $7f$, etc.. However, at the even harmonics $2f$, $4f$, $6f$, we would expect some radiated power, but significantly less than at the odd harmonics. Figure 2.2 shows the wave created by adding the first ten odd harmonics of a square wave as in the equation $0.5 + \frac{2}{\pi}\sum_{k=1}^{10} sin(2\pi(2k-1)t)/(2k-1)$, which produces a square wave with a frequency of $1Hz$ and an amplitude of 1. Below that is the same square wave depicted in the frequency domain, which shows the odd harmonics to have significant power, but the even harmonics to be zero.

Another common occurrence are power spikes that occur at subharmonics ($\frac{f}{2}$, $\frac{f}{3}$, $\frac{f}{4}$, etc.). Potentially, we may even observe significant power at the harmonics of the subharmonics, which would could potentially put power spikes at unexpected frequencies.

An important observation is that in most data buses in a processor, the data is not truly periodic, that means that there is not a single frequency being produced over time. For instance, a bit of data can have a non-periodic pattern of values over time, like “00100011101” (where each bit corresponds to the value at a given clock cycle).

In order to design a realistic model, we must take into account all of these complexi-

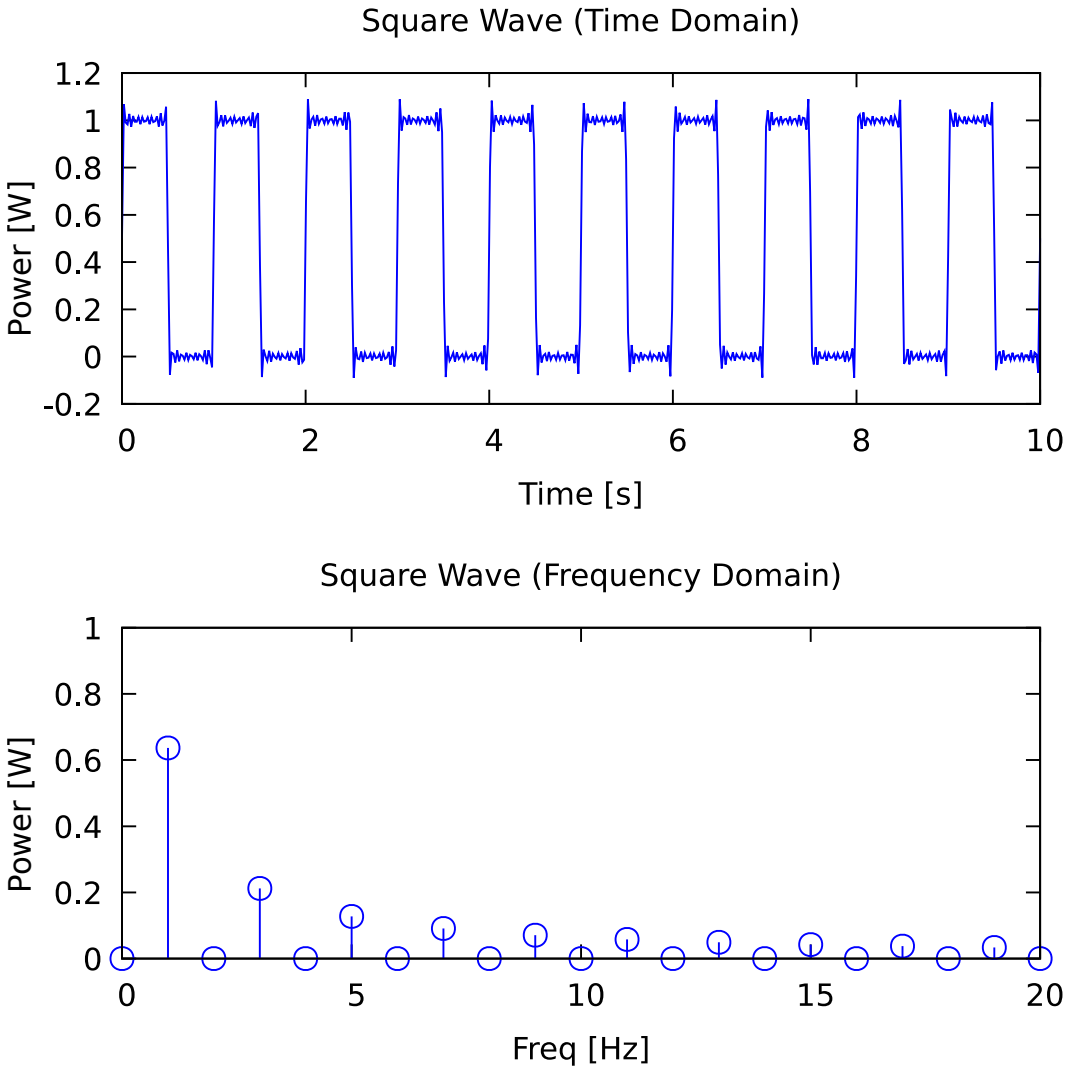

Figure 2.2: A $1Hz$ square wave in the time domain (top) and the frequency domain (bottom). A square wave is constructed from a sine wave and its odd harmonics.

ties. In my model, we start from a cycle-accurate execution trace of the wires in the processor and, assume it is a relatively square wave to perform a Fourier transformation, which allows us to determine which frequencies are being affected by a specific data pattern.

2.3 EMI in Computer Architecture

This section provides descriptions of some of the current research on EMI generated by processors. This section briefly covers topics from wireless communication, profiling, and security.

2.3.1 Profiling

Recently, there has been a push towards using EMI to profile code. Callan *et al.* [14] proposed ZOP: a zero-overhead approach to obtain profiling information via EMI measurements. ZOP first goes through a training phase in which it builds a model that associates different wave forms with different parts of the code. Using that model, ZOP can then go through its profiling phase, which monitors the EMI and can match the current EMI with what part of the code is being executed at that time.

Sehatbakhsh *et al.* [40] were able to exploit the processor's EMI to perform zero-overhead spectral profiling. They have shown a correlation between the amount of time a loop takes (T) and a frequency "spike" in the EMI ($f = 1/T$). Some of our findings may be influenced by this phenomenon.

However, the measurements presented in my work show that two processors can exhibit very different EMI, even when the two processors have the same RTL and are running the same benchmark compiled with the same options. This clearly shows that Spectral Profiling [40] has potential, but it is not obviously applicable, and I consider it future work for potential improvements.

2.3.2 Security

There has been substantial research on EMI and security, and many tools have been published that address the EMI security side channel. One notable publication proposes FASE [16], which is a methodology for finding periodic signals (such as a clock signal) whose amplitude is dependent on processor or memory activity. The authors found that the signals generated by a processor fall into three main categories: strong signals from voltage regulators and power filtering components at the switching frequencies of the regulators, signals from memory-refreshes, and high frequency clock signals and clock harmonics, especially DRAM clocks. They note that all three of these types of signals are affected by what the processor is doing. For example, the signals from the voltage regulators are affected by how much activity is occurring in the processor —the more activity, the higher power consumption, the stronger the signal. The signal caused by memory refreshes are caused by activity by the memory controller, and the EMI from the DRAM clocks are dependant on DRAM activity.

Another tool, SAVAT [15] determines the impact a single instruction has on the RF signal produced by running a program. The authors were able to distinguish single instructions via studying differences in EM radiation. Thus, showing that radiation patterns of some hardware is dependent on what software is running. We can conclude that manipulating the architectural components of a processor will modify the EM emissions, as the same program will be executed differently.

As opposed to the publications that show how vulnerable processors are to side channel attacks, some publications use EMI to discern whether or not a process has been modified.

For example, EDDIE detects code injections without introducing any overheads or changing the hardware or software [34].

2.3.3 Other EMI Models

Wang *et al.* have published a measurement procedure that searches for amplitude-modulated EMI [47]. The algorithm utilizes FASE and SAVAT in order to determine which circuits in a processor are most susceptible to EMI side channel attacks.

Werner *et al.* have proposed a method for determining the instruction-dependent magnetic field sources [48]. Their model determines the locations of the magnetic field sources at a single frequency. In fact, the user is encouraged to choose a "less noisy" frequency for this model. This is different from my proposed model MESC, which models the EMI over multiple frequencies from the processor, instead of modeling the processor from the EMI.

2.3.4 VLSI and System-Wide Tools

In general, not many tools for modeling or minimizing the EMI from a processor exist, even for a system-wide or VLSI perspective. Some PCB simulation tools contain rule checking for EMI, such as Hyperlynx [24], but that is only to comply with Federal Communications Commission (FCC) approval. Commercial full-wave simulation tools such as ANSYS [32], CST Microwave Studio [43], Ansoft HFSS [3] are all available and would provide in-depth and highly accurate predictions of the EMI created by a processor. However, these tools are costly and have long simulation times, and thus are not used in my work.

Most techniques implemented to minimize EMI from a system perspective address the

EMI generated by the clock. One widely used technique that is implemented in most modern processors is Spread Spectrum clock generation [26]. This lowers the EMI generated by the harmonics of a clock signal by modulating the clock. After modulation, the harmonic power decreases significantly. Another technique used to minimize the high frequency EMI produced by the system clock while keeping the skew and power minimal by changing buffer placement and sizing using dynamic programming [29].

Although not strictly EMI, EmerGPU [45] detects and mitigates resonance voltage noise (which causes EM noise) in GPUs. In order to reduce voltage noise, some techniques include reducing the slope of current changes via hardware or software mechanisms.

2.3.4.1 Interference from the clock

There has been a lot of work done on minimizing the interference from the clock [26, 27, 29, 30]. As clock speeds tend to be lower than the frequencies used by wireless communications, it is the clock harmonics that have an adverse affect on signal. Therefore, it is common for chips to have modulated clock signals, slightly changing the cycle time of the clock every cycle. By modulating the clock signal, the attenuation for the harmonics is significantly increased, and therefore the harmonics at the communication frequencies are much lower than for an unmodulated clock signal.

2.3.4.2 Static Techniques to Reduce EMI

Currently, for LTE, the interference reduction techniques being utilized include only static techniques[2]. These techniques address two separate cases: noise from on-die interferers, and noise from an external source (such as the antenna itself).

The most consistent on-die interferer is the clock. Even low-frequency clocks tend to interfere with communication frequencies, as even the twenty fifth harmonic can create frequency spikes that add significant in-band EMI. In fact, some devices fully power down their cores in order to avoid noise during the transmit (TX) phases of communication.

Off-chip interference can be caused by power coupling issues. Even with separate power converters, some parasitics still propagate from core to core. Additionally, during simultaneous TX and receive (RX) actions, noise from the transmit antenna can overpower the received signal, and thus multiple high- and low-pass filters are utilized, and in some cases, the TX signal is fully subtracted out of the received communication. Furthermore, DRAM is often the cause of interference, especially when placed off-chip. However, moving the DRAM on-chip creates less interference.

For Bluetooth, successful operation in the presence of external interferers such as microwave ovens or WiFi networks is to provide some shielding or distance from the cause of the noise [11]. As Bluetooth operates at a relatively high frequency, any in-band interference degrades quickly as distance increases, and even more rapidly through physical objects.

However, if the same device is utilizing both WiFi and Bluetooth, reducing the interference becomes more complex. Currently, collaborative techniques (such as alternating

[2]Information described in this and the following two paragraphs was acquired from personal contacts in industry.

transmissions between Bluetooth and WiFi, or managing packet transmissions based on signal strength) and non-collaborative techniques (such as classifying the Bluetooth channels and altering the channel hopping algorithm to avoid noisy channels) are being investigated [11].

Chapter 3

DEMIS Problem Space Exploration through Measurements

> To boldy go where no one has gone before!
>
> Jean-Luc Picard

In this chapter, I provide measurements of several processors running different applications to provide insights about EMI and better understand the problem space[1]. As expected (and shown in previous work [15, 16]), running different processes on the same core will produce different EMI. My work shows and quantifies that small changes in the application may cause significant EMI shifts. I show that running the same application on two different chips causes very different EMI, to the point that an application can have almost no EMI in a core, and that very same application produces strong interference when running on another core. Al-

[1] This work is part of my publication at The 50th International Symposium on Microarchitecture, October 2017 [23].

though some processors have the same RTL, if they are manufactured in different fabs, they may not produce similar EMI. Running the same application on different processors may yield vastly different EMI.

In particular, I show that the interaction between core, process, and application has a deterministic, but very unpredictable result in EMI interference for a given frequency band. I experimentally show that small architectural changes with small performance impact have a profound impact on EMI. The main contributions of this chapter are as follows:

- Showing that EMI produced by the core is dependent on architectural parameters
- Measuring several real devices to quantify the EMI produced
- Presenting the problem of in-band radio frequency interference as a form of EMI to the computer architecture community

3.1 Setup

In this section, I describe the experimental setup used in order to determine the effect architectural parameters have on EMI, to clearly define the problem space my research addresses. This section also includes a description of the hardware used and the benchmarks that were executed.

3.1.1 Test Equipment

The measurements were taken using a Near-Field Probe set made by Keysight Technologies, which was attached to an N9342C Handheld spectrum analyzer from Agilent Tech-

Table 3.1: System specifications for each device measured.

Device	Commercial Name	Processor	ARM Core	Operating System
A8_A10	Allwinner A10	Allwinner A10	Cortex A8	Ubuntu
A53_K620	HiKey (LeMaker version)	Kirin 620	Cortex A53	Linaro
A11_PI2	Raspberry Pi 2	Broadcom BCM2835	ARM11	Arch Linux, Raspian (Dual Boot)
A53_C2	ODROID-C2	Amlogic S905	Cortex A53	Arch Linux
A15_XU4	ODROID-XU4	Exynos 5422	Cortex-A15	Ubuntu MATE

nologies. This setup had a noise floor at $-120dBm$ and all measurements were taken in a Faraday cage. The deployment is depicted in Figure 3.1. Additionally, I fixed the probe onto each device for consistency, as there can be a substantial differences in measured power when measuring different locations.

The relevant specifications of each of the five devices measured are provided in Table 3.1. One of each type of device was measured. The rest of this thesis will refer to the devices as in the first column of Table 3.1.

3.1.2 Benchmarks

I measured each device while it was idling as well as while it was running a series of benchmarks. I utilized a set of in-house benchmarks, described in Section 3.2 and SPEC2006 [28] applications to determine if using architectural techniques would be effective for manipulating EMI in specific bands. From there, I utilized the SPEC2006 benchmarks to determine the effects on larger processes.

Each device ran all the benchmarks from the SPEC2006 benchmark suite[2] natively

[2]Due to compilation issues, gobmk, omnetpp, and xalancbmk were not included in this experiment.

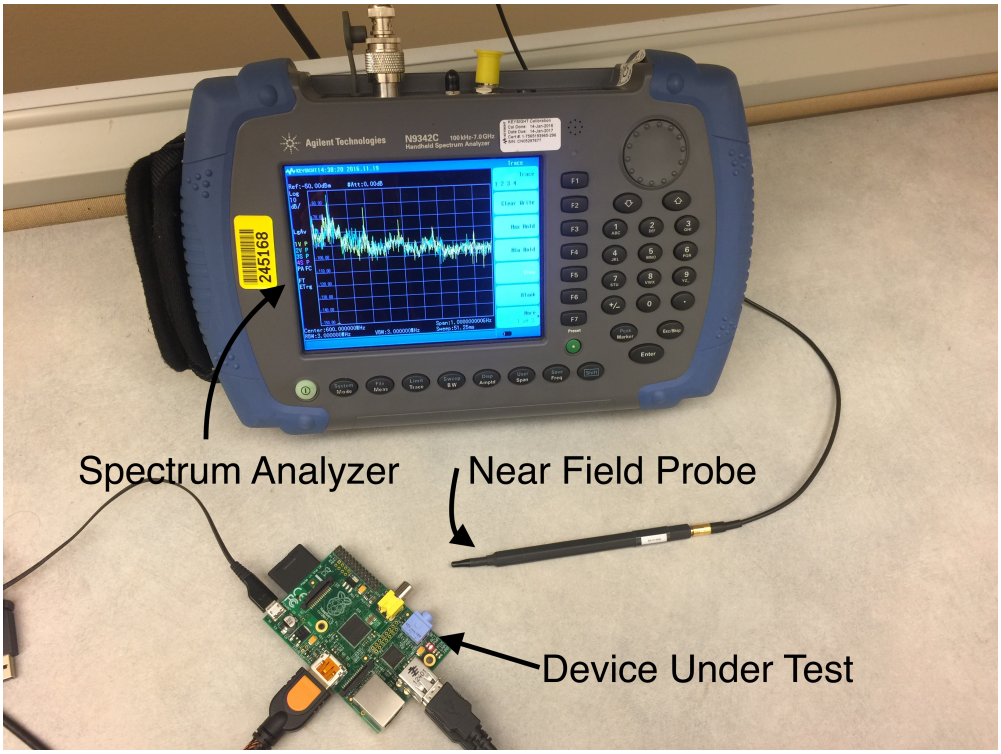

Figure 3.1: I utilized a spectrum analyzer and near-field probes to accurately measure the EMI from multiple hardware platforms running a set of benchmarks.

during measurements with the default settings for each device (without modifying the clock speed, etc.) as a baseline. Then, the benchmarks were run when changing different architectural parameters, including compiler optimizations.

In the interest of avoiding repetitiveness, the results provided in this section focus more the mcf, sjeng, and libquantum benchmarks. These benchmarks were chosen because they emphasize specific architectural parameters: mcf is memory intensive, with many RAM accesses; sjeng causes many branch mispredictions and calculations; and libquantum triggers many cache misses and prefetching.

3.1.3 Bands Analyzed

The noise was measured in the scope of wireless communication technologies and is reported as the difference between emitted power when running the benchmark and idling for five different RF communication frequency bands, described in Table 3.2.

Table 3.2: Wireless technology frequency ranges and maximum bandwidths used at a given time. This work will refer to each band as described in the first column.

Band	Technology	Lower Bound (MHz)	Upper Bound (MHz)
LTE 800 Lower	LTE band 18 (UL and DL)	815	875
LTE 700	LTE bands 12, 13, 14, 17 (UL and DL)	699	798
LTE 450	LTE band 31 (UL and DL)	452.5	467.2
SuperWiFi XR7	Ubiquity XtremeRange7 WiFi	698	746
SuperWiFi 600	SuperWiFi in TV spectrum	600	630

3.2 Insights and Measurements

This section uses physical measurements of real systems to identify the effects of architectural and compiler parameters on RF interference. Through measuring in-house benchmarks on the hardware described in Table 3.1, I was able to determine that different parameters affect the RF interference differently. This section will describe my findings from measuring the differences in interference when applying different architectural manipulations.

3.2.1 Measurement Repeatability and Accuracy

In order to ensure that the measurements taken are consistent, I measured the A53_K620 running the mcf benchmark 12 times. The results are provided in Figure 3.2.

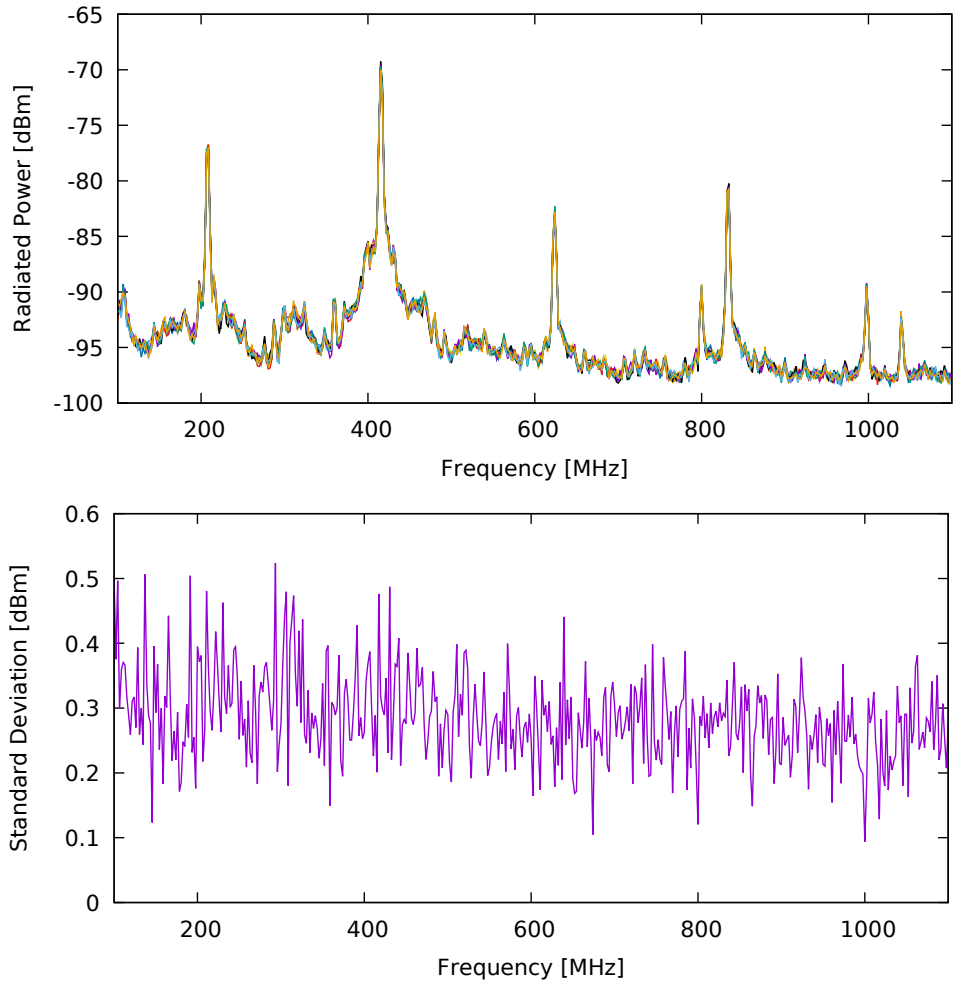

Figure 3.2: The EMI created by the A53_K620 running the mcf benchmark is consistent across multiple measurements. These measurements were taken on at different times on different days. The standard deviation of the radiated power across these measurements is negligible in comparison to the total EMI.

As we can see, the measurements are repeatable across different days and times, with a maximum standard deviation of about $0.5dBm$, a value that is a negligible amount compared to the measured power. Therefore, the rest of the measurements provided throughout this document will be presented without an error threshold.

3.2.2 Compiler Impact on EMI

To evaluate small binary changes impact on EMI, I use different compilation options (different schedulers or optimizations) that should have small performance impact to see the EMI effect. For example, using the -O2 or the -O3 compilation options, or changing the scheduling using the -mtune option.

Figure 3.3 shows the interference level and the IPC for the first 100 seconds of execution of mcf when executed on the A11_PI2. The IPC plot shows that the performance does not change when switching from -O2 (option 1) to -O3 (option 2) for mcf. Nevertheless, we see over $3dB$ interference reduction in the most noisy band (LTE 700) using -O2, while the LTE 450 band has an increase in interference with -O3. This means that for the mcf application, if we want to avoid interference in the LTE 450 band we should use -O3, whereas -O2 should be used if the communication is in the LTE 700 band.

I repeated the same experiment on the A8_A10 platform to understand the impact of hardware changes. The interference levels across the spectrum are shown in Figure 3.4, which also shows the baseline interference level when the core is powered on, but idle. Unlike in the A11_PI2 case, the -O2 improves interference compared to the -O3 option for both LTE 700 (by $2dB$) and LTE 450 (by $3dB$). Again, for mcf the -O2 and -O3 does not significantly change the performance in this platform.

To analyze its impact on another application, Figure 3.5 depicts the EMI of the A53_K620 processor running the sjeng benchmark after being compiled -O3 and -O2. In this case, the -O3 compilation option yields a 1% speedup. The interference shifts in frequency

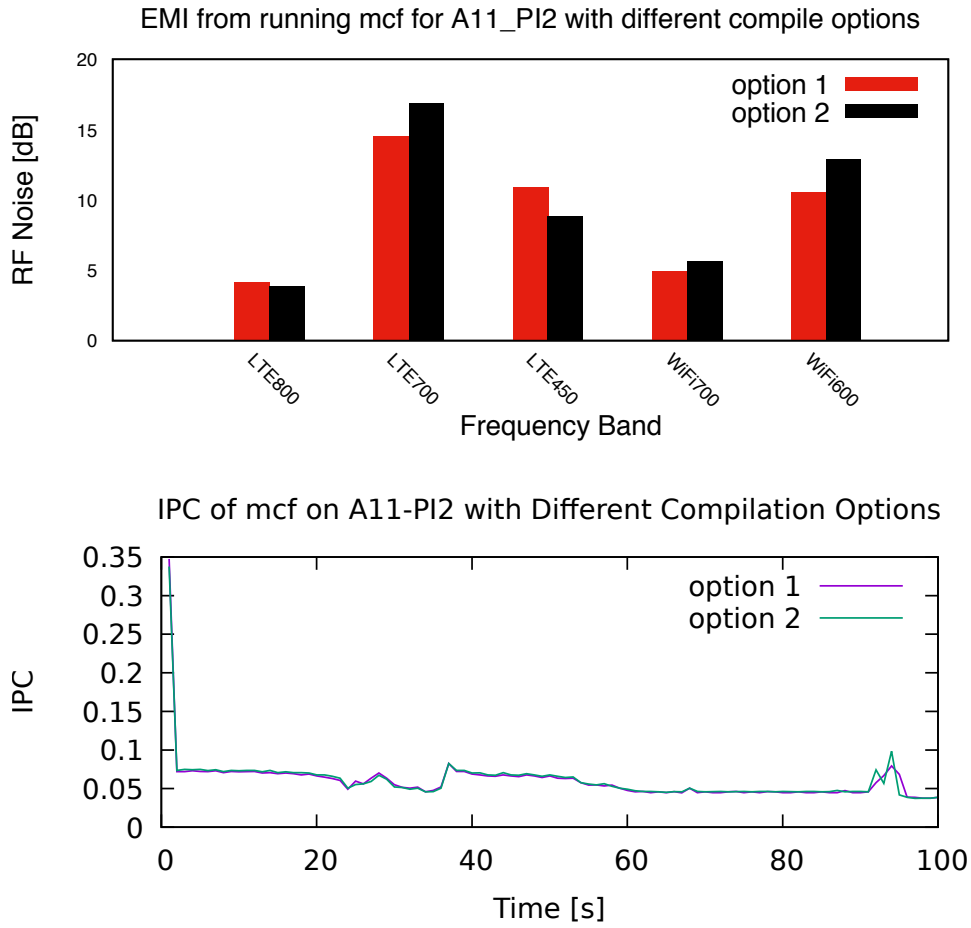

Figure 3.3: Running mcf with different compile options on the A11_PI2 yielded minimal difference in IPC, while still yielding a 3dB swing in maximum EMI at certain bands.

slightly, and in most cases the -O3 has a higher EMI. For this board mcf showed a similar behavior as sjeng with -O2 having less interference.

Figure 3.6 shows the case that only -mtune option is changed for hmmer application. I keep the default -march=native, which implies -mtune=native, and change the -mtune to cortex-a57. In this case, it has no performance difference when executing in the A11_PI2. The plots shows another case of clear shift in the frequency around 600MHz with little performance

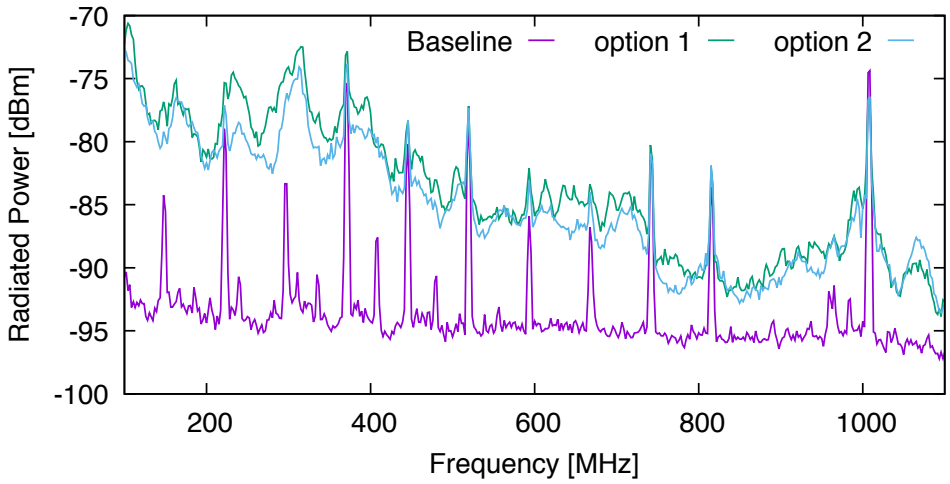

Figure 3.4: Different compilation options change the EMI in the mcf benchmark on the A8_A10 processor without performance impact.

impact.

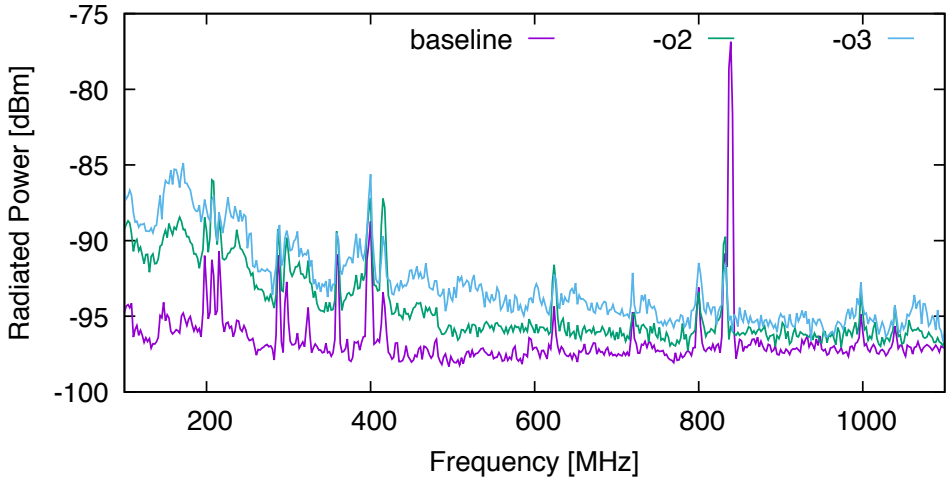

Figure 3.5: When running sjeng on the A53_K620 processor, the EMI being produced changed based on the optimization the benchmark was compiled with.

The maximum interference reduction for the 5 analyzed bands playing with -O2, -O3 and -mtune options was 8dB in the bzip2 benchmark (not shown in the plots). Also, as expected,

the compilation options can have a big performance impact. For example, in bzip2 the -mtune has over 30% improvement when applied in the A15_XU4 board (also not shown). For the applications of these findings, I exclude the cases with high performance impact.

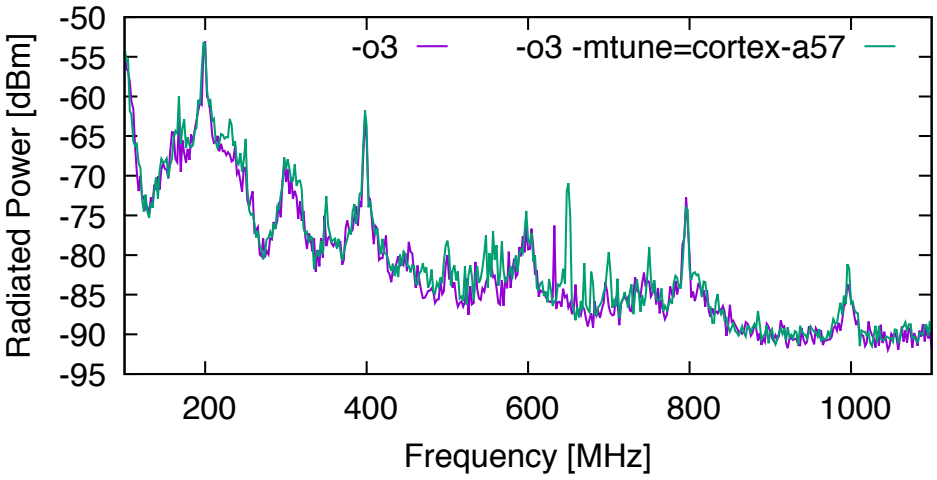

Figure 3.6: Compiling hmmer with different -mtune options has no performance impact but a change in EMI.

In an attempt to further understand the effect of the compiler on the EMI, we performed an exhaustive set of measurements on the A53_K620 board. I measured the EMI across multiple bands, disabling only one minor optimization from the full -O3 optimizations at a time. There are 11 gcc flags that differentiate between -O2 and -O3 with gcc 6.3.1.

Figure 3.7 shows the normalized execution time versus the normalized EMI for LTE 700 for each of the compilation options. Each dot corresponds to the -O3 optimization with one of the 11 flags disabled. Since we also keep the -O3 optimization, there are 12 points per benchmark. The x-axis shows execution time normalized to the fastest execution for that benchmark, and the y-axis shows the EMI improvement with respect to the worst EMI for LTE 700 in dB. Unfortunately, I was unable to discern a consistent correlation between optimization,

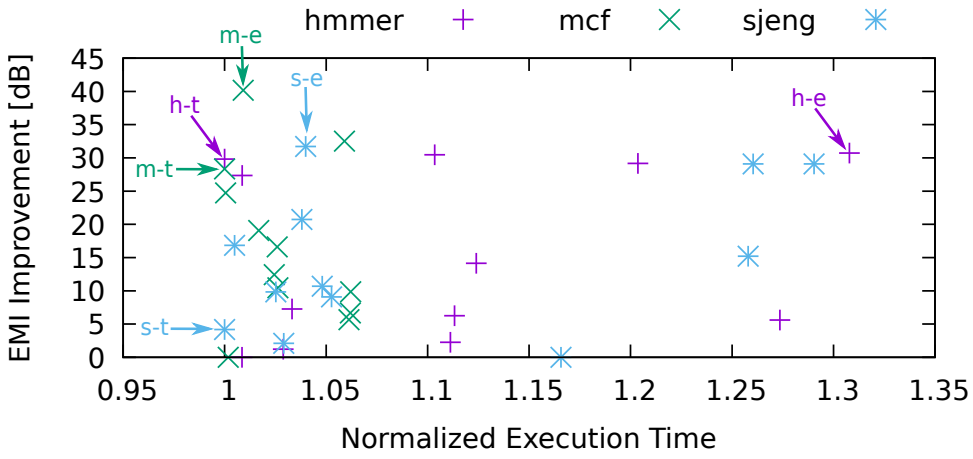

Figure 3.7: In-band EMI in the LTE 700 band emitted by the A53_K620 board with respect to execution time. I was unable to find a correlation between efficiency and EMI.

runtime, and EMI. For example, disabling the fgcse-after-reload optimization has a 40 dB EMI reduction with small slowdown for mcf, but only a 2 dB EMI reduction for hmmer, and 9 dB for sjeng (both also with small slowdowns).

In Figure 3.7, the points marking the fastest runtime and best EMI improvement are marked for each benchmark. For the hmmer benchmark, the point marked "h-e" for best EMI improvement ran with the fpeel-loops optimization disabled, and had a 1 dB improvement over the fastest run (marked "h-t"), which had the ftree-partial-pre optimization disabled. In the case of this benchmark, there is a 0.3x slowdown between the best EMI and the fastest execution time. The mcf benchmark ran fastest with the ftree-loop-vectorize optimization disabled ("m-t"), but with a 0.009x slowdown, disabling the fcse-after-reload optimization (marked "m-e") offered a 12 dB improvement. Lastly, the sjeng benchmark ran fastest with fpredictive-commoning disabled ("s-t"), but disabling the ftree-slp-vectorize optimization ("s-e") offers over 25 dB improvement with only a 0.03x slowdown.

Clearly, there is no correlation between the optimizations that is standard across benchmarks, even across different bands, there is no discernible relationship. Although potential for future work, we decided not to use multiple binaries, and we restrict ourselves to just -O2 vs -O3 in the rest of the thesis. Keeping many binary files would take up an excessive amount of space and raise tune/selection algorithm issues.

The main conclusion is that by adjusting the compilation options and constraining cases to a small performance impact, we can achieve up to 8dB interference reduction levels with many cases providing 3dB reduction. A source of difficulty managing the system is that the effect is not only compiler dependent but compiler/core/platform dependent. The same compilation options yield opposite results in different processors.

In Chapter 6 (which directly applies the techniques discussed in this chapter), when compiler techniques are mentioned, I mean to change the compiler options (-O2/-O3/-mtune) and select the binary with the lower interference in the band to protect. Although not covered, a JIT based system would be the best platform allowing to dynamically perform small binary changes to mitigate interference while monitoring the interference level impact.

3.2.3 Benchmark Impact on EMI

Clearly, EMI is different depending on the application [15, 16]. However, EMI also changes on a per-band basis. Figure 3.8 shows the average and maximum EMI in the five different RF bands on the A53_K620 and the A11_PI2. The interference depicted is the difference between emitted power when running the benchmark at each frequency with respect to an idling processor. However, it is hard to see a relationship between the in-band interference and

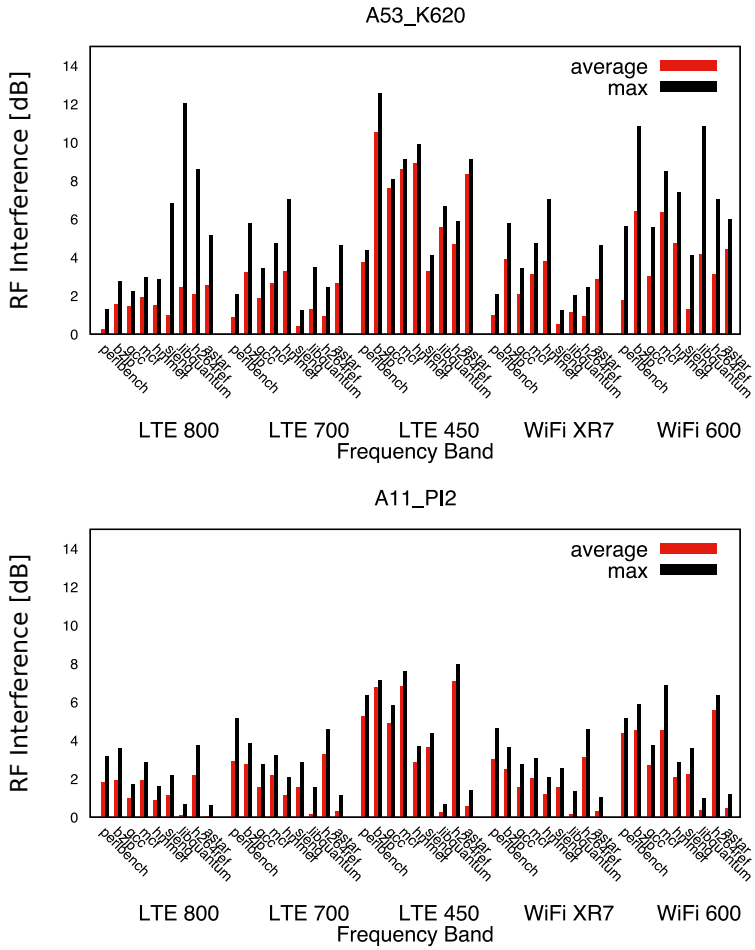

Figure 3.8: Interference created in each band running SPEC2006 benchmarks on the A53_K620 and the A11_PI2, without the base EMI created by the processors running only the operating system.

benchmark that is consistent across processors.

Thus far, all the measured data presented in this thesis have been the average of interference per frequency over a sustained amount of time. However, many programs go through different phases during execution. As the different phases tend to run different types of instructions, it stands to reason that the radiation at different frequencies will be different during different phases.

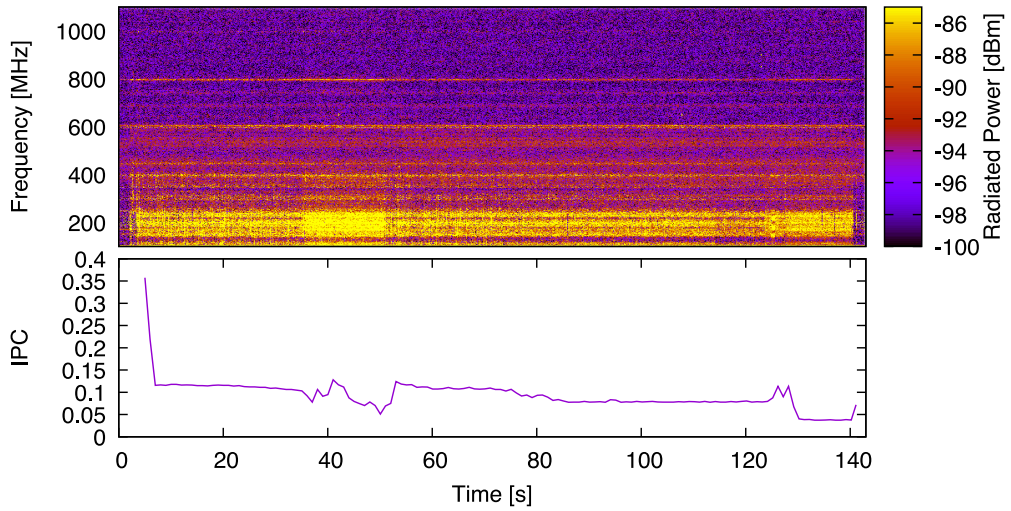

Figure 3.9: IPC and EMI for mcf on A11_PI2. The processor goes through phases with different EMI patterns. In this case, it seems that when there is lower IPC, there is more EMI.

Figure 3.9 shows these phases as the mcf benchmark is run on the A11_PI2 processor over time, the interference is shown as a color map per frequency and time. Interestingly, as the IPC decreases, there appears to generally be more RF emanations. From these results it is not clear why this behavior happens, but it could be related to changes in processor activity.

Figure 3.10 depicts the IPC and phases of the A53_K620 processor running the sjeng benchmark. In contrast to the A11_PI2 running mcf, in many cases the A53_K620 has more interference when the IPC is higher for the sjeng benchmark. Nevertheless, the clear EMI fluctuations are related to IPC changes but it is not a direct function of IPC because sometimes higher IPC means lower interference in some bands (which can be seen in in the LTE 800 Lower band in Figure 3.10.

In addition to only observing the phases per benchmark, I also performed some measurements over time with different input sets. Although the different input sets caused each benchmark to spend different amounts of time in each phase, the EMI during each phase re-

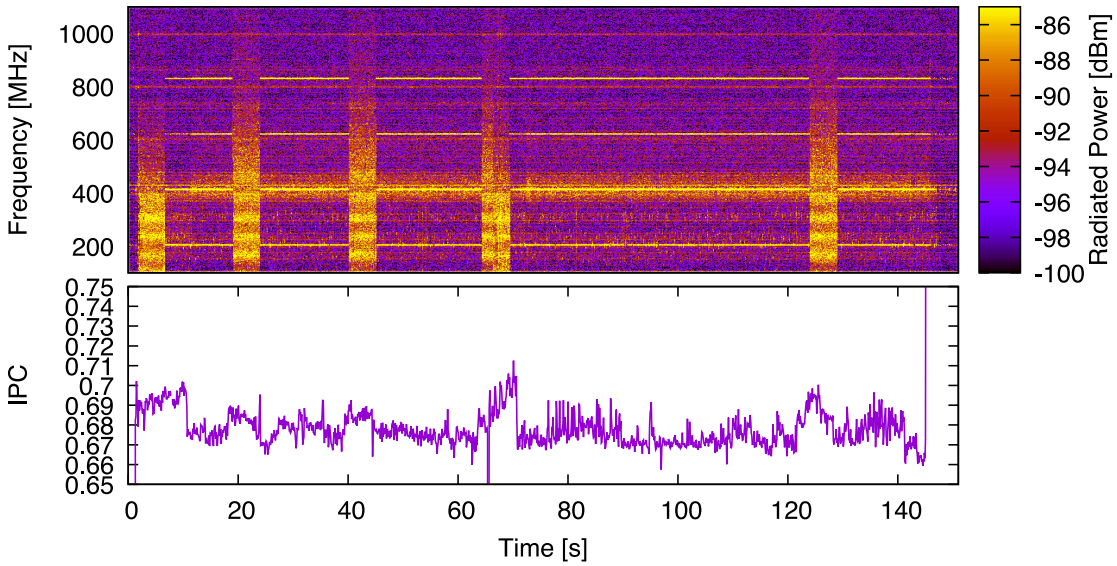

Figure 3.10: For the sjeng benchmark on the A53_K620, it is more clear that the processor goes through phases with different EMI patterns. When compared with the IPC, there is no clear higher/lower EMI correlation, just change.

mained unchanged for the benchmarks measured in this thesis.

The main conclusion is that there are clear program phases in IPC and EMI as the application executes, but the IPC phases are not necessarily correlated with EMI phases. Although sometimes an IPC increase results in an EMI increase, in many phases an IPC increase results in an EMI reduction. Adapting through phases has potential benefit and the phases can be detected with IPC phase changes.

3.2.4 Cache Impact on EMI

Cache accesses consistently affect the amount of RF interference being emitted by each chip. In order to observe how cache accesses affected the RF interference created by

each processor, we utilized a synthetic program that emphasizes accessing a large 2D array and injected a delay before the cache misses (Listing 3.1).

```
1  int main () {
2      int total = 0;
3      for(int i=0;i<8192;i++) {
4          for(int j=0;j<8192;j++) {
5              //asm("nop");
6              total += matrix[j][i];
7          }
8      }
9
10     printf("total=%d\n",total);
11 }
```

Listing 3.1: Function for testing code with frequent cache accesses. Delays (nop calls) were inserted at line 5. The numbers in lines 3 and 4 were manipulated based on the cache sizes for each processor in order to ensure the desired cache misses were occurring (L1 or 2).

The first thing I noticed was that there was a distinct difference in frequency and amplitude between L1 and L2 cache accesses, as shown in Figure 3.11, which I determined by manipulating lines 3 and 4 in Listing 3.1. Clearly, L2 cache accesses trigger more interference than L1 cache misses, presumably because an L1 cache access (miss) is required for an L2 cache access to trigger. However, at some frequencies L1 cache accesses are noisier, which may be attributed to the fact that L2 caches are slower and would therefore interfere with different frequencies. Unfortunately, a processor would take a huge performance hit should it forgo accessing the L1 cache in favor of the L2 cache, so I focused on techniques that would take less

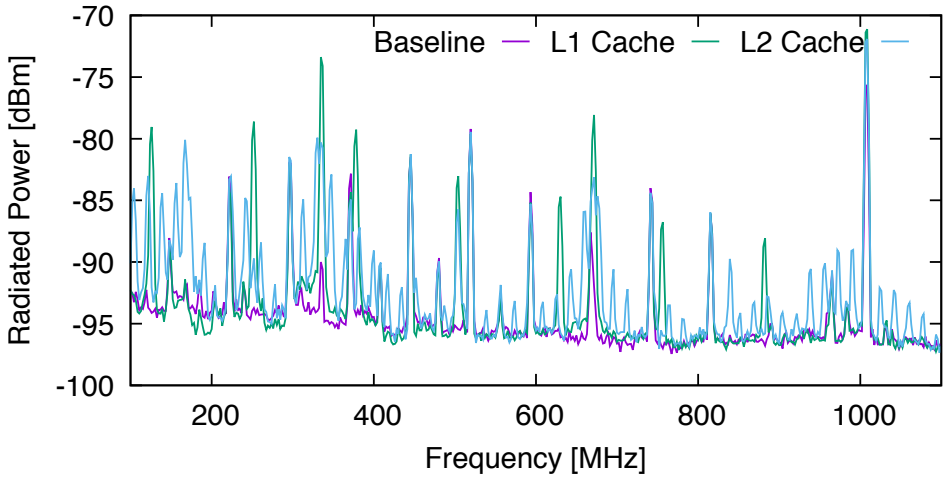

Figure 3.11: The interference created by accessing the L1 and L2 caches on the A8_A10 processor is significantly different.

critical performance hits, despite the EMI benefits.

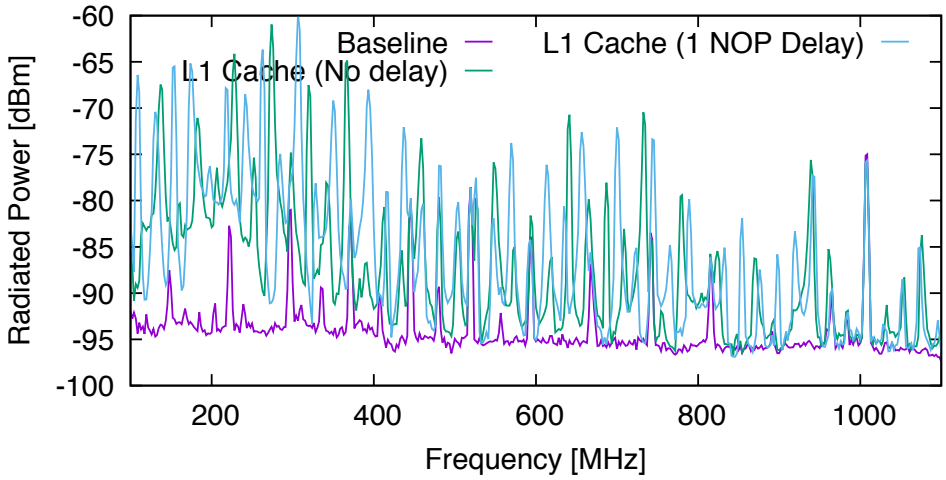

Figure 3.12: The interference created by accessing the L1 cache with and without delay on the A8_A10 processor is significantly different.

Instead of switching which cache to access, I tried injecting a minimal delay before accessing the L1 cache by uncommenting line 5 in Listing 3.1. When I delayed the cache

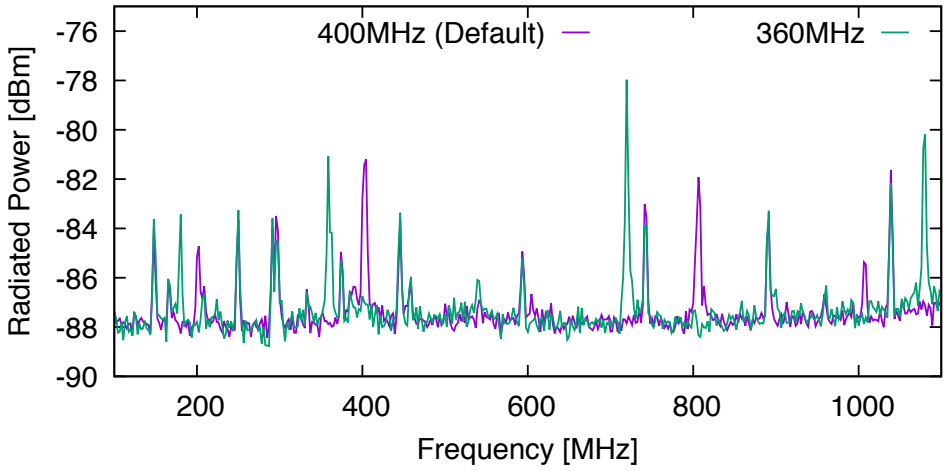

Figure 3.13: By changing the SDRAM speed for the A11_PI2, I was able to move the noisy peaks in frequency.

access slightly, I was able to reduce the interference, as shown in Figure 3.12. Clearly, there is a significant affect on the frequency response of the device: consistently a frequency shift of about $15MHz$. Injecting this minimal single nop delay was able to trigger a significant response, where the power spike has moved more than a Bluetooth channel and most LTE channels.

The main conclusion is that caches have a big impact in delay and misses in the L2 tend to have a higher impact at lower frequency bands. Although this has potential to be a good technique, the DEMIS technique in Chapter 6 does not trigger cache delays or misses, I was constrained to measurements in real systems; it is not clear how to introduce affect cache behavior at runtime without RTL access.

3.2.5 Memory Impact on EMI

On the A11_PI2, modifying the DRAM speed was a simple matter of modifying the BIOS parameters. I measured the EMI when the processor was running nothing but the OS. As

shown in Figure 3.13, modifying the $\mathrm{sdram\_freq}$ parameter in the config.txt file that serves as boot settings for the A11_PI2 changes the frequency of certain noisy spikes when idling.

These frequency spikes appear to be directly caused by memory accesses because they occur at the DRAM frequency and at its harmonics. When the DRAM frequency was shifted to $360MHz$, I was able to see the spikes shift accordingly: from $400MHz$ to $360MHz$, and from $800MHz$ (the first harmonic) to $720MHz$.

I measured the EMI produced by the A11_PI2 with the default DRAM speed of 400 MHz as well as with a 10% slower speed of $360MHz$. This time, however, I took these measurements while running the mcf and sjeng benchmarks. The results of these measurements are presented in Figure 3.14. The libquantum benchmark produced negligible EMI on the A11_PI2 processor, so the results are omitted.

Reducing the DRAM decreases interference by $3dB$ in the SuperWiFi 600 band, but adds a spike in the SuperWiFi XR7 band for the mcf benchmark. However, the default settings have less interference in the SuperWiFi XR7 band and more interference in the SuperWiFi 600 band for the same benchmark, so depending on what type of WiFi is being used, different DRAM speeds are better for this application.

Similar effects can be seen for the sjeng benchmark as well. For example, the default DRAM speed is better for the SuperWiFi XR7 and worse for the LTE 700 frequency bands than the slower DRAM speed.

The main conclusion is that DDR creates big spikes in interference. Even when the processor is idle there are big spikes because the DDR clock is kept running. If the band in use is affected by DDR, the only solution is to shift the DDR operating frequency. For the analyzed

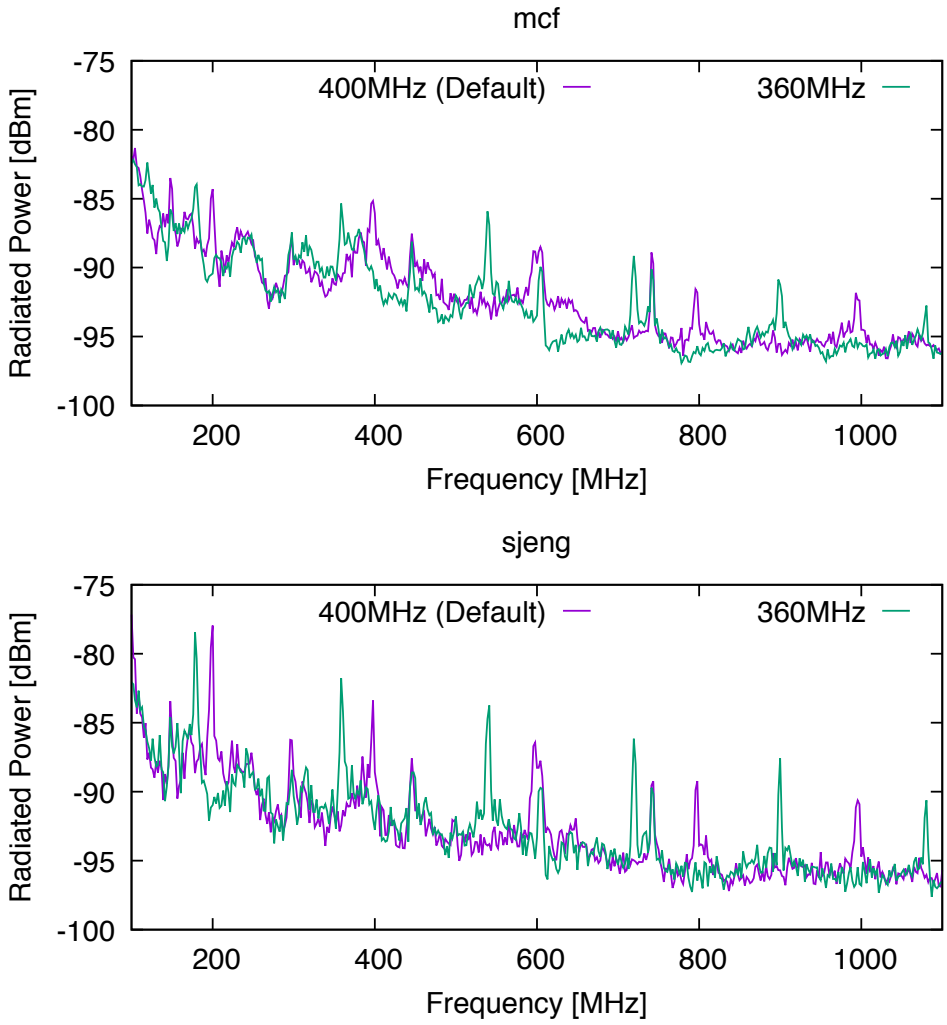

Figure 3.14: When running the mcf and sjeng benchmarks on the A11_PI2, changing the DRAM speed as little as 10% has a significant impact on EMI.

cases, small 10% DDR frequency change is enough to move the spikes out of most bands. For many benchmarks like sjeng, this has no performance impact but some benchmarks like mcf are very sensitive resulting in a 10% performance impact.

The DDR clock is one of the strongest EMI spikes I observed, even stronger than the processor clock. The reason is that processor clocks have a more effective modulation of the

clock signal. Future DDR designs may want to consider a better clock modulation requirement.

3.2.6 Execution Core Impact on EMI

As seen in Figure 3.8, even the same benchmark will emit different EMI when run on different processors. Therefore, I launched a more specific investigation on the impact the execution core itself has on EMI.

```
1  int main () {
2      int total = 0;
3      for(int k=0;k<10000;k++) {
4          for(int i=1;i<10000;i++) {
5              total += k/i;
6              //asm("nop");
7          }
8
9      }
10     printf("total=%d\n",total);
11 }
```

Listing 3.2: Function for testing computation heavy code. This code was used for testing the FPU by changing all data types from int to float. Delays were added by uncommenting line 6, and adding as many nops as desired.

In order to test the interference generated from using the FPU, I performed multiple divisions, once using the int data type and once using the float data type. The source code for the benchmark used is in Listing 3.2 in the A8_A10 hardware.

I noticed a substantial difference in EM radiation between the two test cases on the

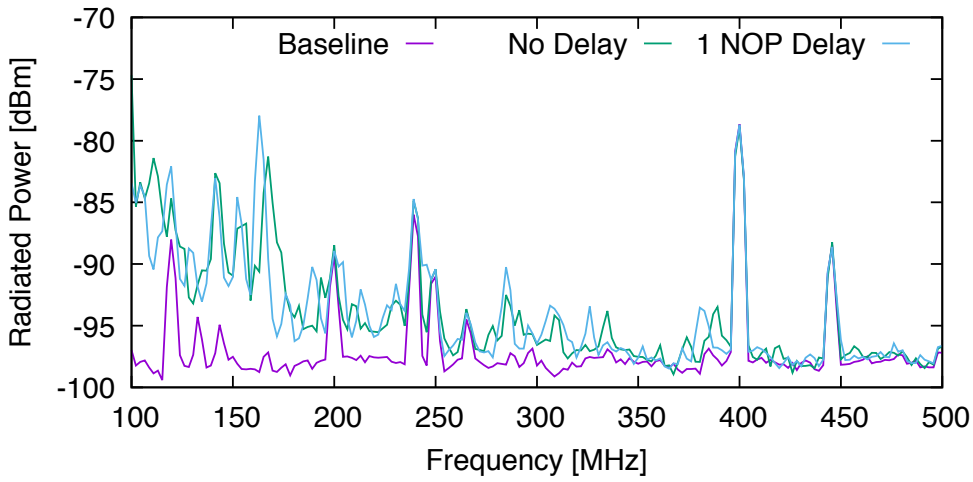

Figure 3.15: When utilizing the FPU as opposed to doing simply integer calculations on the A8_A10 processor, there was much less RF interference. Furthermore, different frequencies were affected differently.

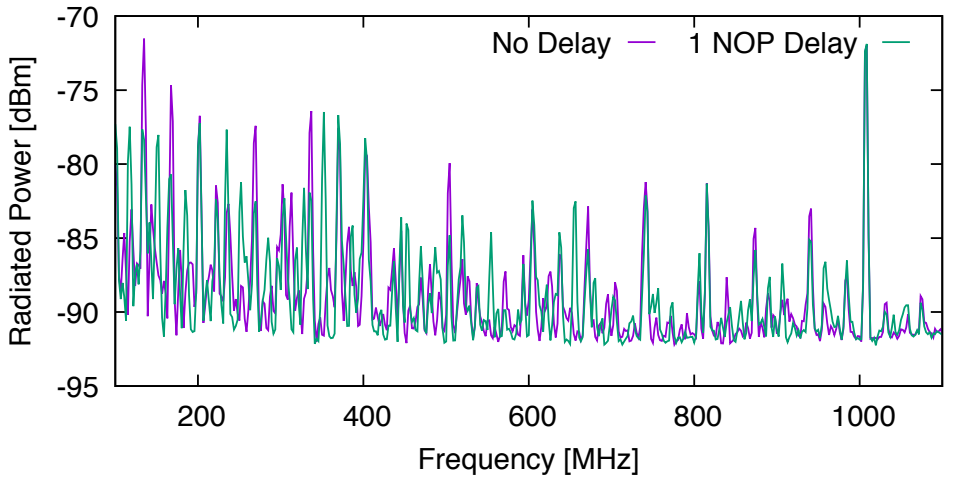

Figure 3.16: RF interference created when performing integer calculations on the A8_A10 processor. The frequency response differs with the addition of a single nop in the loop.

A8_A10 as in Figure 3.15. Not only did the interference decrease, but I also observed that the lower frequencies tended to exhibit a more consistent interference during floating-point calculations, whereas higher frequencies observed consistently more interference from integer-

only calculations.

By injecting even a single nop into the calculations (uncommenting line 6 in Listing 3.2), I was able to observe a distinct difference between frequency responses for both integer calculations and floating point calculations as seen in Figure 3.16.

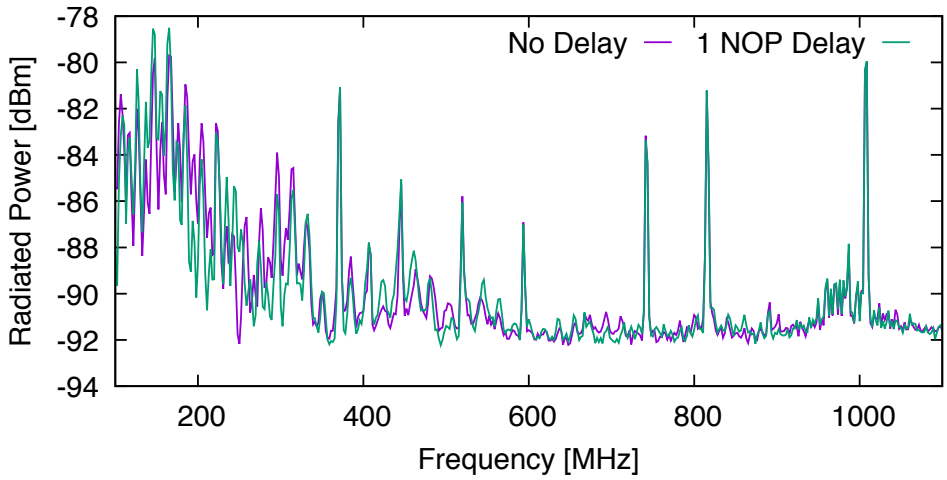

Figure 3.17: RF interference created when performing floating-point operations on the A8_A10 processor. The frequency response differs with the addition of a single nop in the loop.

As the interference generated from the integer calculations exhibited frequent changes in power in each frequency, it is easy to determine that the power spikes undergo a frequency shift when one nop is injected into the code. Furthermore, the frequency spikes tend to have less power in the benchmark with a delay by almost $7dB$ in the lower bands and $5dB$ at higher frequencies. The measured power is presented in Figure 3.16.

Figure 3.17 shows the difference in frequency responses when running the floating point benchmark with and without a nop delay. As opposed to the integer calculations, the floating point calculations have wide-band low-frequency interference. However, by injecting a nop into the inner for loop, I was able to move the interference to different frequencies.

Figure 3.18 shows the radiated power from the A53_C2 and A53_K620 processors when running the three benchmarks without changing any of the default settings on the boards. The benchmarks were compiled without any optimizations. It is clear in these two figures that each benchmark yields a distinct radiation pattern on each processor. Also, the benchmarks tend to behave similarly across processors. As the processors all run at different frequencies, we expect some variations, but in general it is clear that specific architectural components produce distinct interference patterns.

It is important to note that the A53_C2 and the A53_K620 processors are fabricated from the same RTL in the same fab and process node. Therefore, it is surprising that the EMI is so different between the two processors.

Additionally, on the A53_C2 processor, the libquantum benchmark produces less interference than sjeng, but on they are switched on the A53_K620 processor. I found this to be quite common throughout our investigation.

I was also able to measure these benchmarks on an out of order (OoO) processor, the A15_XU4 which features four OoO cores and four in-order cores. The results are provided in Figure 3.19. Interestingly enough, the OoO core appears to only emit noticeable radiation in frequencies less than $600MHz$, which is below most wireless communication bands. However, running the libquantum benchmark introduces multiple $10dB$ spikes into the LTE 450 band, which could cause significant connectivity degradation.

The main conclusion is cores have a high impact in processor interference, with some cores like the A15_XU4 having less interference at higher frequencies. The high power blocks like the FPU can produce interference but adding delays, which sort of behaves like a frequency

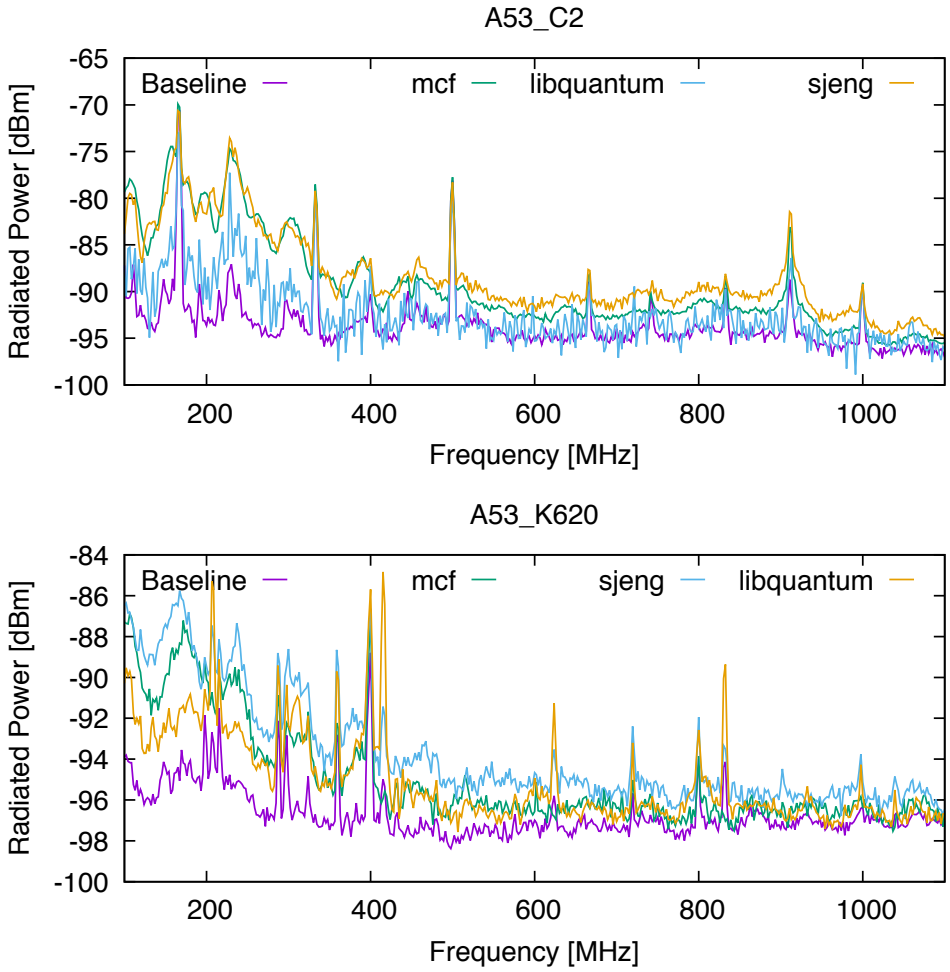

Figure 3.18: When running the different benchmarks on the A53_C2, and A53_K620 processors, the EMI being produced varied noticeably. A53_C2 and A53_K620 are fabricated from the same RTL.

modulation of the FPU, changes the interference level. Even the same RTL (A53) has a very different behavior in absolute and relative terms between applications. This means that other factors besides architecture can have a big impact, but as show small core fluctuations can compensate for the interference.

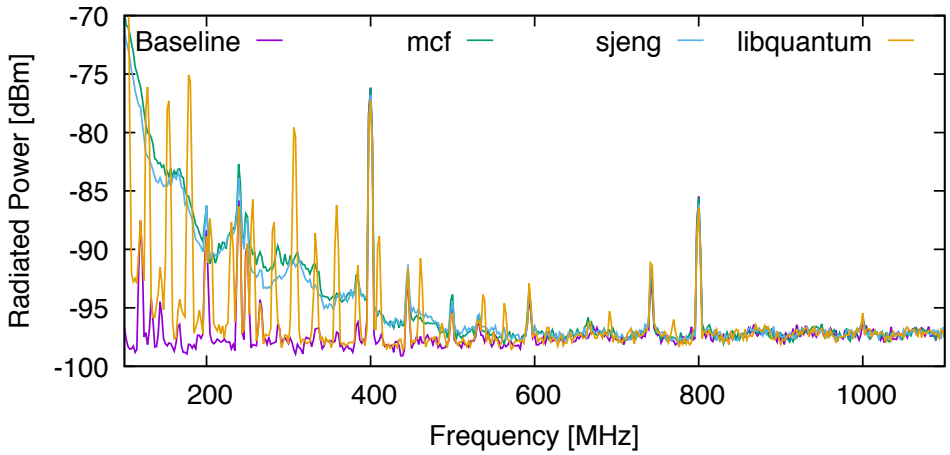

Figure 3.19: When using the A15_XU4, the EMI varied significantly across benchmarks. However, no significant EMI was produced above $600MHz$, and therefore is irrelevant to most wireless technology frequencies.

3.2.7 Further Insights

In addition to the hardware described in Table 3.1, some experiments were also performed on a MYiR z-turn Board, running Ubuntu on a Xilinx ZYNQ chip that includes an FPGA on chip. Unfortunately, the measurements were all too noisy to provide usable results, as the processing noise was overpowered by noise that I believe was generated by the FPGA itself. However, I was able to note that when starting and ending a process, all frequencies measured experienced a significant increase in noise (more than $30dB$). I hypothesize that this noise may have been caused by components being powered on or off during that time.

3.3 Conclusion for DEMIS Problem Space Exploration

In this chapter, I performed a thorough investigation of the problem space and proved that computer architectural parameters have a significant effect on EMI. Additionally, I offered

insights into multiple different architectural parameters, and investigate their effect on EMI. Of the multiple parameters studied, cache access schemes, clock speed, and small process delays had little impact on processor efficiency. However, small dynamic changes of these produced a significant reduction of in-band interference. These observations were expanded to include more system-level techniques for reducing in-band EMI such as changing compilation options.

Chapter 4

MESC: Model for EMI from an SoC

> Those who have knowledge don't predict. Those who predict don't have knowledge.
>
> Lao Tzu

In this chapter, I propose MESC, Model for EMI from an SoC[1]. MESC takes into account the activity rate of individual wires in a process and the layout to approximate the expected EMI from that process. This is in distinct contrast to previous works that utilize measured EMI to profile programs [14, 40] or to isolate the on-chip location of magnetic field sources [48], as I aim to model the EMI from the layout, not the other way around. MESC uses some basic initial power measurements of a device as well as statistical sampling of switching activity in order to model the expected EMI of a processor. MESC is a simpler model than full-wave simulators [3, 24, 43], with a more convenient runtime in order to allow design space

[1] This work is part of the publication I have submitted to The 51st Annual IEEE/ACM International Symposium on Microarchitecture.

exploration. In my evaluations, I used traces obtained by statistically (temporal) sampling an FPGA emulation to get activity, but traces from RTL simulation could also be used. I also propose spatial sampling, *i.e.*, not considering all the wires in a core, only considering the longest wires on a processor. The rationale behind this is that longer wires have more impact in the EM radiation.

To evaluate MESC, I perform FPGA emulations and measure the EMI using a spectrum analyzer. I then compare the expected EMI profile across different frequencies with the measured EMI. Even by using temporal and spatial sampling, MESC is able to predict with an average of less than 1% error. The frequencies at which we observe peaks of EMI are correctly predicted by MESC and the main source of error is with relation to the magnitude of the EMI. This is particularly important since predicting the frequency bands at which interference occurs is more important than predicting the exact magnitude of the EMI.

The main contributions of this chapter are:

- MESC: a novel methodology for modeling the EMI of a processor running on a processor
- Validation of MESC with FPGA measurements of a processor's EMI

4.1 MESC Flow

In this section, I discuss each step in MESC used to model the expected EMI. The EMI produced by a net depends on the net length and waveform present in the net. In theory, the shape of the net would also have an influence on EMI, but from preliminary tests, I noticed that this has a lower impact on the overall EMI produced. Sampling all the nets in a core, all the

time would generate a lot of overhead, both in terms of hardware needed and in performance impact. On the other hand, not all nets contribute equally to EMI.

4.1.1 Sampling Strategy

Because monitoring every net on a processor would be impractical, MESC only monitors a subset of the wires in the core. Since EMI is proportional to wire length, only long wires will emit a significant amount of EMI. Therefore, I chose which nets to monitor based on length: after excluding reset and clock nets, I choose the longest nets until at least 80% of all the metal in the processor was being probed. A histogram of the total length covered when choosing the longest nets is provided in Figure 4.1 (details in Section 4.2). For each net used, MESC requires information on length and width (metal layer). This is summarized as a single number. We call the combination of this information for all nets used "layout profile." Metal layer is important since different layers have different metal pitch, which alters the current (thicker wires have lower resistance per length unit). My implementation of MESC targets an FPGA, so metal layer is not taken into account, but would be necessary in an ASIC implementation of MESC.

Additionally, in the interest of minimizing the amount of data that is necessary, I decided to take samples of the switching activity as well, basically this means that at every period of time, the data in each wire of interest will be collected for a certain number of cycles. As shown in previous works [14, 23], the EMI changes as processors execute different phases of a process. For the purposes of this work, determining which execution phase is being processed is sufficient.

In my evaluation, I show that 1024 samples per monitored net and ≈ 600 nets are

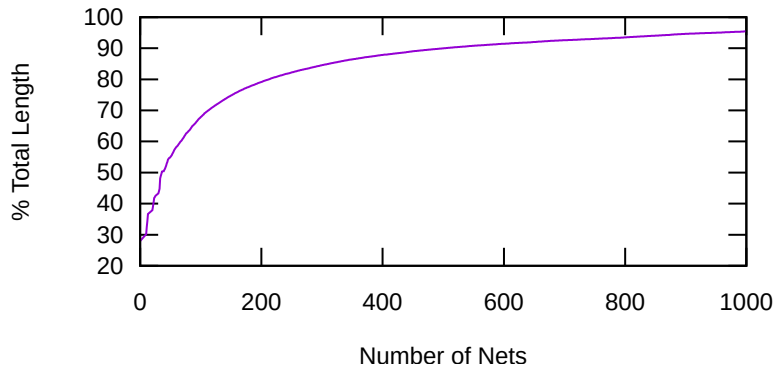

Figure 4.1: By choosing the longest nets first, MESC is able to cover 90% of the total length covered by all nets by using only the longest 600 nets of the total 646,952 nets.

enough to get accurate EMI estimate. Those figures can be be reduced at the expense of accuracy.

Once I have obtained the switching activities of each sample net in different time slices and the relative amount of metal that each net has, I have all the data necessary to build MESC.

4.1.2 Converting to the Frequency Domain

After retrieving the switching activities, I perform an FFT on each net of each sample, converting these time slices into the frequency domain. When putting together the MESC EMI for the entire processor, I must also take into account the relative amounts of metal each net has, in the case of an FPGA, how long the net is. Thus, each net is assigned a weight equal to its relative length.

Despite these nets being linear, and thus the total radiated power expected to be proportional to the square of the wire length ($P_{total} \propto L^2$ as in Equation 2.3), the FPGA is split up

into wire segments of equal length with buffers in between. Therefore, each net can be considered to be multiple radiators of equal dimensions instead of one single long wire. Therefore, in my implementation of MESC for FPGAs, the net is weighted by its length, not the square of its length. In the future, further analysis of wire lengths and buffer placement must be performed in order to create an appropriate weighting algorithm for ASICs, but due to resource and time constraints, I did not pursue this in my research. Additionally, for this implementation of MESC on an FPGA, each wire has the same signal strength and is therefore disregarded, but for ASICs, the drive strength of each wire will need to be taken into account.

The weight is multiplied by its frequency domain value before the weighted value of each net is added together for every frequency. This yields a total power over frequency from 0 to half of the sampling frequency, which in this case is the clock frequency, as our tools sample the values each clock cycle.

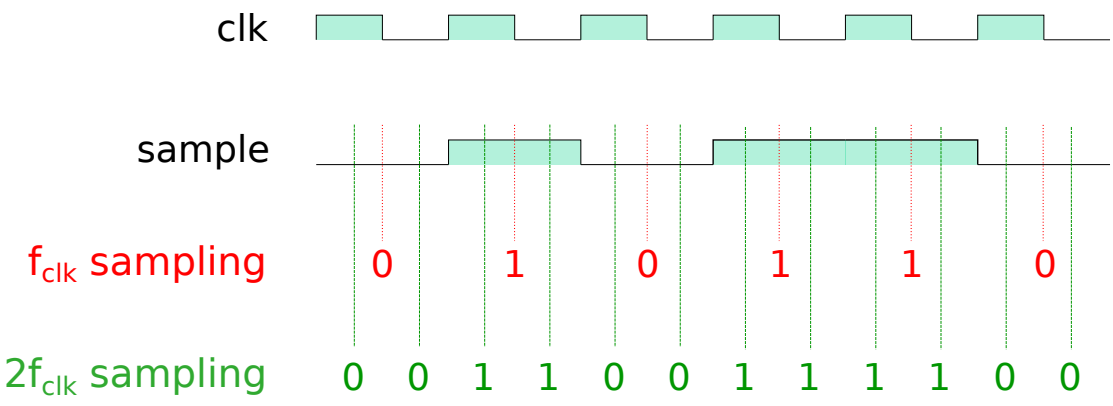

Figure 4.2: One clock cycle has the same value, so MESC can artificially increase the sampling frequency, thus increasing the accuracy of the model.

However, only being able to obtain the frequency domain until up to half of the clock

frequency is insufficient for this model. Therefore, to implement MESC, before taking the FFT, I repeat the value of each clock cycle to artificially increase the sampling frequency. I argue that, because we are dealing with square waves for this FPGA application, the value should remain unchanged throughout each clock period. For example, a switching activity sample containing the values '0 1 0 1 1 0' with a clock frequency of $1MHz$ would only yield a Fourier transform that corresponds to values up to $0.5MHz$. However, if we inject an extra sample per clock cycle, we can use the values '0 0 1 1 0 0 1 1 1 1 0 0' with a sampling frequency of $2MHz$, and we are able to get values in the frequency domain up to $1MHz$. This is shown in Figure 4.2. When implementing MESC for ASICs, we would have to take into account less square waves, and add values that are calculated based on the rise and fall times of the data for that specific processor. In the future, it would be possible to implement MESC for FPGAs that also account for less square waves, but for this application I found approximating the waves to be square was sufficient while adding simplicity.

For this implementation of MESC, I used 8 samples per clock cycle, as after investigation, I found that injecting more samples yielded a small increases in accuracy for a large increase in compute time.

4.1.3 Adding Harmonics

Once the samples have been converted to the frequency domain, MESC must compensate for each signal's harmonics. Remember from Chapter 2 that square waves are comprised of a signal and its odd harmonics. Thus, MESC adds the odd harmonics of the signal to account up to the highest desired frequency for the model, dividing the power by the value of the har-

monic. That is, for a signal at frequency f with strength p, the nth harmonic will be added at the frequency fn with strength p/n, provided that n is odd.

However, because the FFT takes into account harmonics, this is only necessary for frequencies above half the artificially increased sampling frequency. Adding these higher frequencies increases the model's accuracy without adding a significant compilation time, as this is a linear operation and the FFT is $O(n \log n)$.

4.1.4 Artifacts and Shielding

Packaging, board and other off-chip components affect the observed EMI. Off-chip components may create additional spikes in frequencies that were not expected by simply modeling on-chip wires; I call these artifacts. Also, packaging, cooling components and sockets may create an EMI filtering at certain frequencies, and I call this effect shielding.

In my model, I consider both artifacts and shielding. First, to detect artifacts, the chip must be on without processing anything. In the case of an FPGA, I was able to observe that these artifacts existed even when the FPGA had not been programmed. Figure 4.3 shows an example of artifacts that were measured on the FPGA used in my work. The FPGA was programmed with a simple bus, that was run at different frequencies ($40MHz$ and $50MHz$), as expected, an EMI spike is observed in those frequencies. However, there is a consistent artifact at frequency $60MHz$. We can therefore conclude that this spike does not come from the design implemented, but rather from some other component, either on the FPGA itself or on the board.

The process of detecting artifacts is static and can be done once in the lifetime of a chip. The process involves finding EMI peaks that are consistent regardless of the application

running.

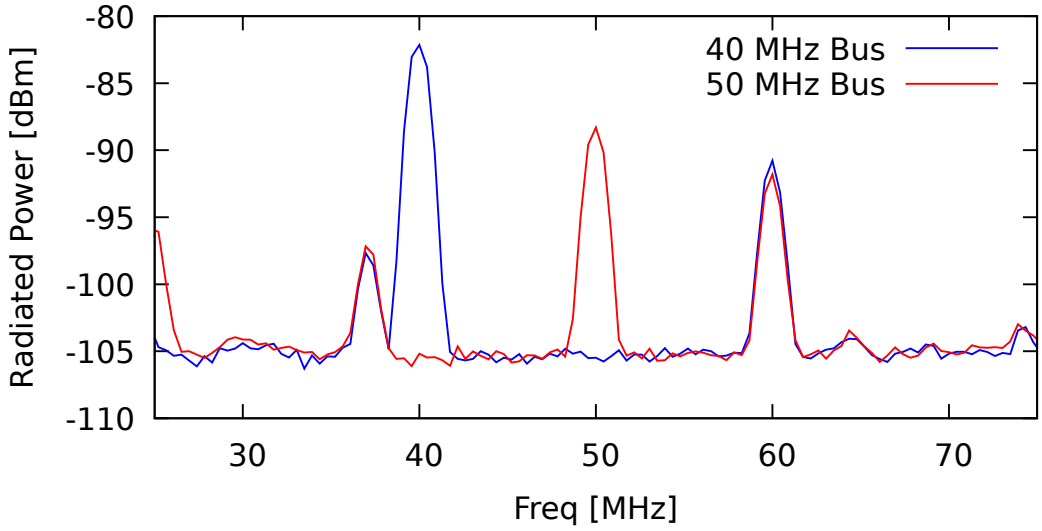

Figure 4.3: To detect artifacts, we run the FPGA at different frequencies. The figure shows the measurements artifacts at 46 and 60MHz for our FPGA setup when running a simple 40 or 50MHz bus.

Additionally, as I am unable to measure the EMI of the processor directly, I must take into account any shielding created by the housing (packaging, cooling, board, etc.). I am able to determine this by measuring the power emitted from a single bus with a consistent width and length carrying varying clock frequencies. With the peak power at each frequency, I am able to construct a linear function over frequency that must be taken into account when measuring the EMI. Figure 4.4 shows the EMI emitted by a single bus, designed in Vivado for this experiment, with consistent length and width switching at different frequencies. As shown, there is a significant difference in power as the frequency changes. However, I was able to determine an approximate logarithmic decay, with a power decrease of $20\log_{10}(f)$, which we

would expect after an inspection of the logarithmic Friis Transmission Formula (Equation 2.5).

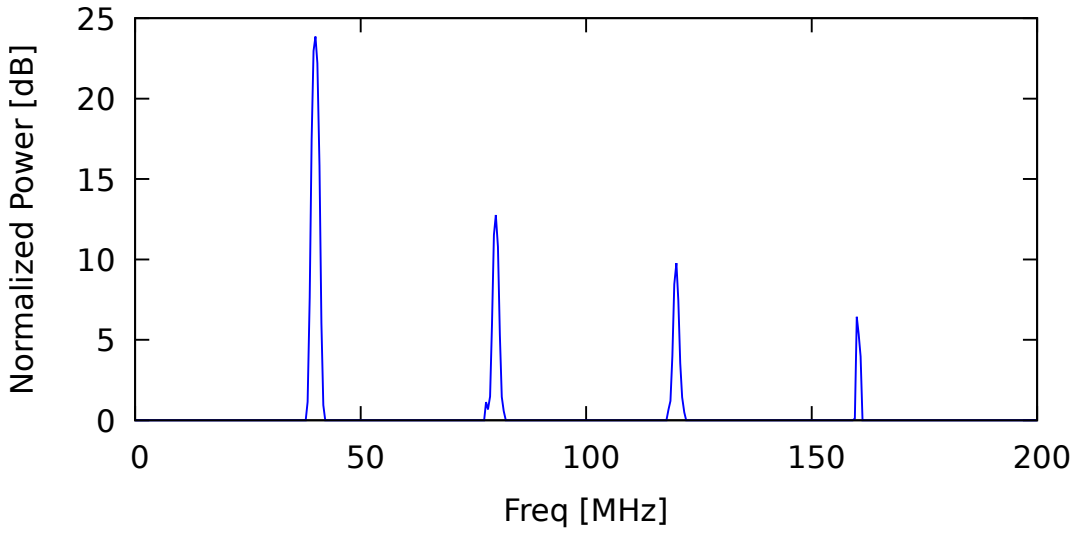

Figure 4.4: Interference of a single bus toggling at 40, 80, 120, and 160MHz. From this, I can infer that our measurement setup is not without some shielding as a function of frequency, which must be compensated for in MESC for accuracy.

4.1.5 Other physical factors

There are other factors that affect EMI. In general, any parameter that can affect the current will also affect EMI. Here, I discuss how some parameters are expected to affect EMI. Incorporating those parameters in MESC is left as future work, since they would require an ASIC tapeout.

Dynamic Voltage Frequency Scaling (DVFS) is a technique that adjusts the supply voltage during execution [13]. The current, and thus the EMI, is expected to be scaled linearly with voltage in wires. Since this is a dynamic technique, it needs to be taken into account during

the model computation.

Buffers are sometimes added in long wires to improve the overall delay of the path [2]. The layout profile for MESC in this work only takes into account the total size of a wire and not the presence of buffers. However, adding buffers to a wire will increase the overall current and therefore the EMI. In my FPGA based evaluation, it was not possible to evaluate the addition of buffers to a wire, since buffers were already pre-allocated in the FPGA, however to get more accuracy in the model, buffers would also need to be considered.

MESC should also be extended when the metal lines in an ASIC have different widths. In the current model, I assume all the metals to have the same length. This is fine in this evaluation because we pick the longest wires, and in an FPGA all of them have the same width. In a ASIC, this is not necessarily the same.

Another important issue that may arise in particular for ASICs is slew and load. Slew is the rate at which a signal changes in the beginning of a cycle. A signal that has high slew or signals with high load will be "less square." This will mostly affect harmonics of the main signal frequency. Since harmonics have lower amplitude, the error due to non-square waves is expected to be small. In cases where more accuracy is needed, this can be considered by calculating the load in each of the wires of interest.

MESC could be extended to consider the DVFS, voltage fluctuations, buffer insertion, and different ASIC metal lines. As the evaluation shows, this does not seem to be an issue for FPGAs. A potential tapeout for ASICs could create a validation setup for the model.

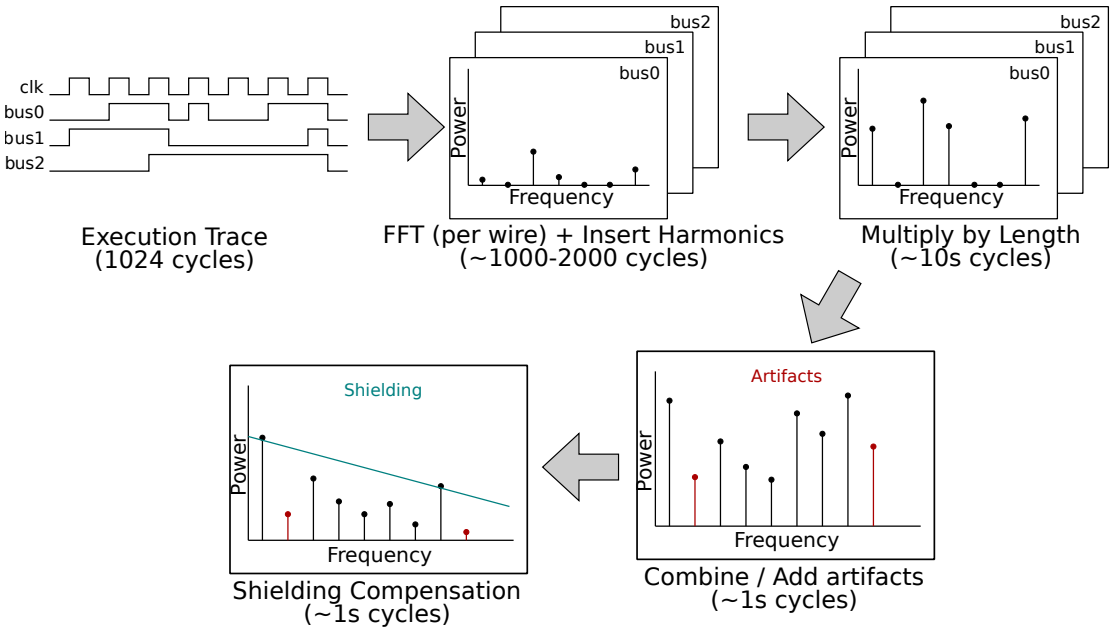

Figure 4.5: MESC consists of collecting execution traces for each of the sampled wired, calculating the FFT individually, and some processing before combining the FFTs. The final EMI model is provided after inserting the artifacts and compensating for shielding effects.

4.1.6 Final Flow

The final MESC flow consists thus into collecting execution traces for the sampled wires at sampled intervals, calculating the FFT of the traces in each wire, adding harmonics, multiplying by wire length and combining all the wires FFTs into a single FFT. The artifacts and shielding are added to the final FFT. Figure 4.5 depicts the flow.

4.1.7 Alternate MESC Application: EMI vs Power Tradeoff

In Chapter 5, I propose EMI CHopper as a use-case for MESC. However, MESC enables architects to model EMI for other applications as well. Thus far, EMI has been only

evaluated in architecture papers from measurements in real hardware. With the possibility of modeling EMI without real hardware may allow EMI estimation to be included in higher level architectural exploration.

For instance, another application for MESC would be to determine the effect of increasing the clock frequency, but only sending the values over the bus every couple of cycles, which should change the EMI produced by the clock without changing the processor speed. One drawback would be that there would be an increase in power consumption as a tradeoff for manipulating the EMI.

4.2 Setup

Before I go into the evaluation of MESC, I will discuss the measurement setup, some challenges associated with the measurements taken, and how they were handled throughout the construction of the model.

Since my approach requires two different layouts, I decided to take measurements out of an FPGA to avoid taping-out a chip. Therefore, the clock frequencies used are much lower than what would be feasible in an ASIC. Thus, in my research, I was unable to use real communication frequencies due to the constraints of running processes on an FPGA core instead of an ASIC processor. However, I believe that this proof of concept is sufficient to show that MESC (and EMI CHopper, proposed in Chapter 5) is feasible on a real chip.

All EMI measurements were taken using a N9342C handheld spectrum analyzer with a Near-Field probe set from Keysight Technologies. I used the Digilent Genesys 2 Kintex-7

FPGA Development Board to develop and verify MESC. In order to evaluate the Spec2006 benchmarks using different core layouts, I employed the use of the OpenPiton [8] core with different placement optimizations for our FPGA. OpenPiton was synthesized using Vivado 2017.4. The run frequency of OpenPiton in the FPGA is 66.6MHz, and for simpler experiments without cores the clock frequency is variable and mentioned in the evaluation on a per-experiment basis.

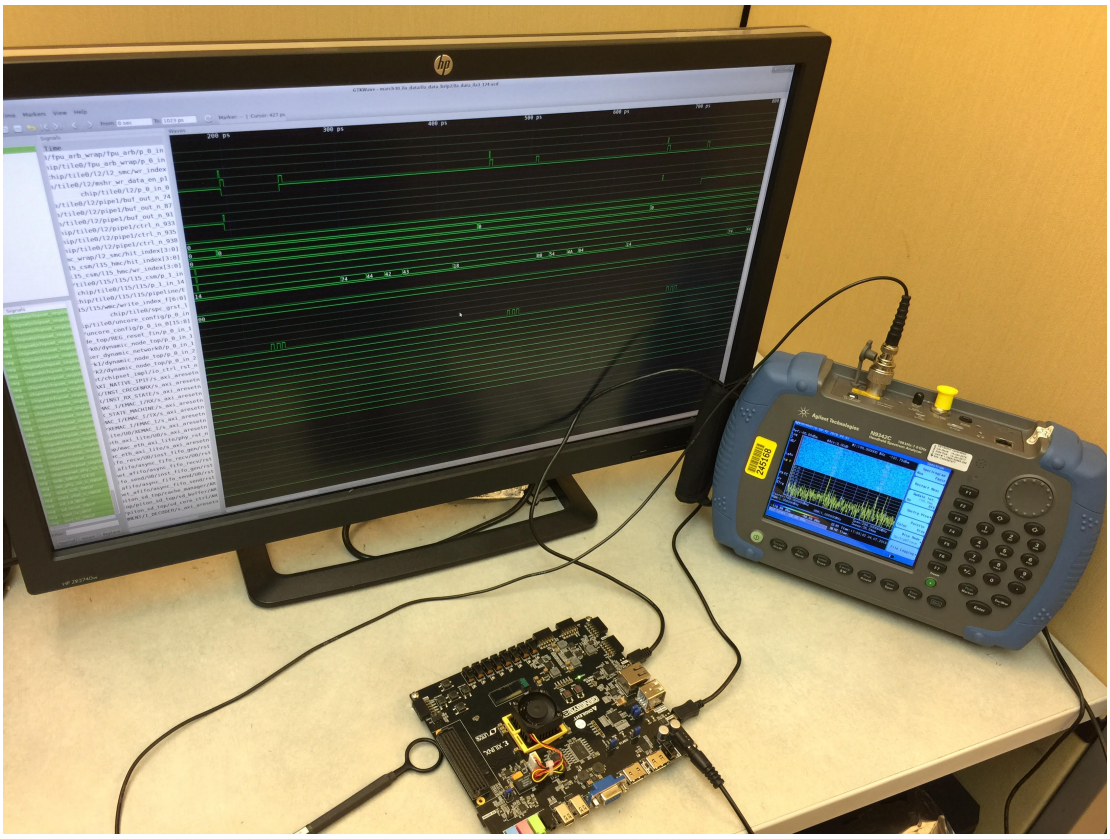

Figure 4.6: Depicts the measurement setup used for FPGA measurements.

4.2.1 Setup Verification

My first task was to ensure the RF principles described in Chapter 2 were applicable to my setup. I had some concerns that applying some of the principles described in Chapter 2 to an FPGA might yield some inconsistent results, as "wires" on an FPGA are broken up with buffers and routing resources such as crossbars and multiplexers. Additionally, FPGAs also contain a considerable number of additional electrical components, such as logic for clock generation, memories, IO connects, that could be active even if not used. Thus, I started with a series of smaller tests to see how implementing circuitry on an FPGA compared with the theoretical expectations.

My first tests consisted of creating buses of varying length, number of bits, and shape that transmitted clock signals at multiple different frequencies. As expected, buses that were twice as long or twice as wide emitted twice the amount of power at the signal frequency, an increase of dB.

Once I determined that the buses radiated EMI as expected, I implemented a debug core in Vivado that monitored the activity of each net on the FPGA. From the switching activity of each net, as one might find in a Value Change Dump (VCD) file, I was able to perform a discrete FFT of the values of each net at every clock cycle in order to obtain the signal's distribution in the frequency domain.

Using these simple tests, I was able to monitor the activity of all the nets, but moving forward, it was clear that monitoring every net on a processor would not be feasible.

4.3 Evaluation

I start this evaluation by validating MESC. Figure 4.7 shows a comparison of the output of MESC vs the measured EMI from the namd benchmark on OpenPiton running on a FPGA. The model yields an estimated EMI that is quite accurate, with only 5dB maximum difference between MESC and the measured output. It is reasonable to conclude from these results that MESC is able to accurately estimate the magnitude of the EMI and, most importantly, the frequency at which EMI peaks occur.

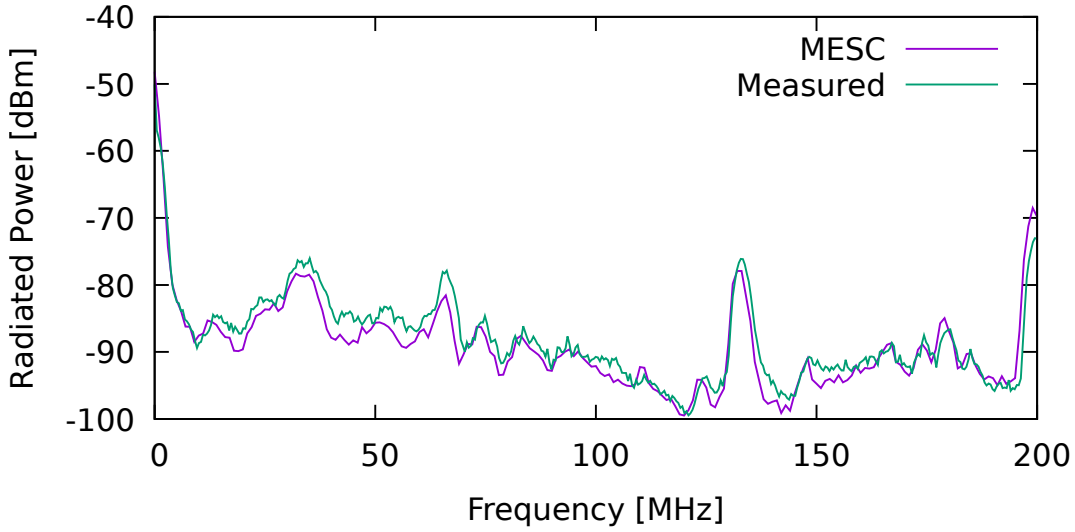

Figure 4.7: MESC of namd compared with the actual measured EMI of namd running with the same layout. The maximum difference between two is 10dB.

To go more into details on the accuracy and limitations of the MESC, I want to analyze other examples. The first point that I want to analyze is the effect of shielding in the model. The next set of results use a simple design with three buses, each running at different toggle rate.

One switches at a rate of $200MHz$, one that switches at a rate of $57MHz$, and one that switches in a "random" 7-bit pattern that keeps repeating itself. These activities were chosen in order to ensure that the harmonics would not overlap. Figure 4.8 shows the measured and estimated EMI for the three buses.

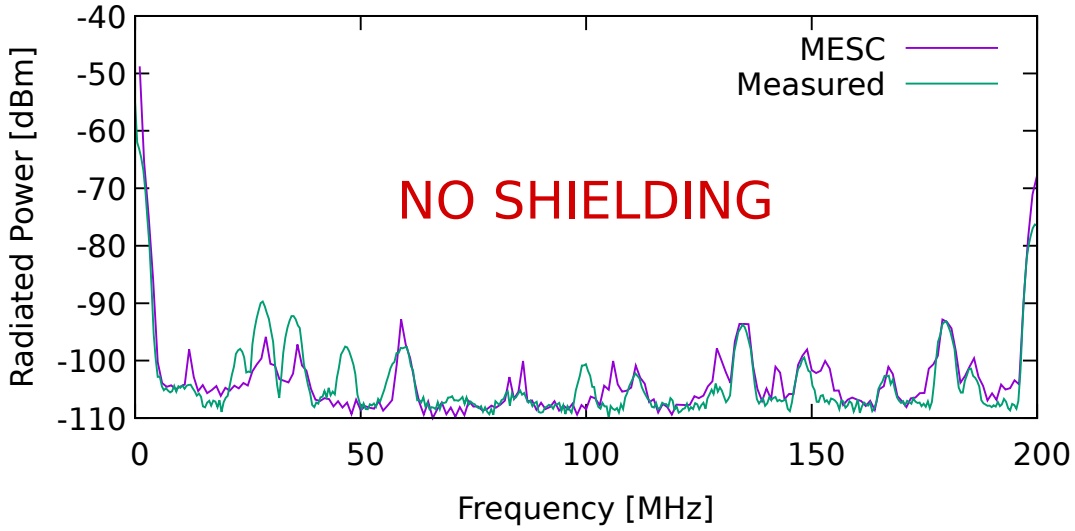

Figure 4.8: Compares MESC, without compensating for shielding, with the measured EMI of a small test case with three buses after adding the artifacts. From this figure, I concluded that compensating for shielding effects improves MESC accuracy.

As depicted in Figure 4.8, I started with small test cases to verify the accuracy of my model. One challenge for comparing measured EMI with my model was some of the inconsistencies in the amount of power being measured as the frequency changed. Figure 4.8 clearly shows that, at the lower frequencies, the peaks tend to be a lower magnitude than the actual measured EMI, whereas the higher frequencies tend to match up more accurately.

Next, I evaluate the need for artifact insertion in the model. Figure 4.7 also has a peak

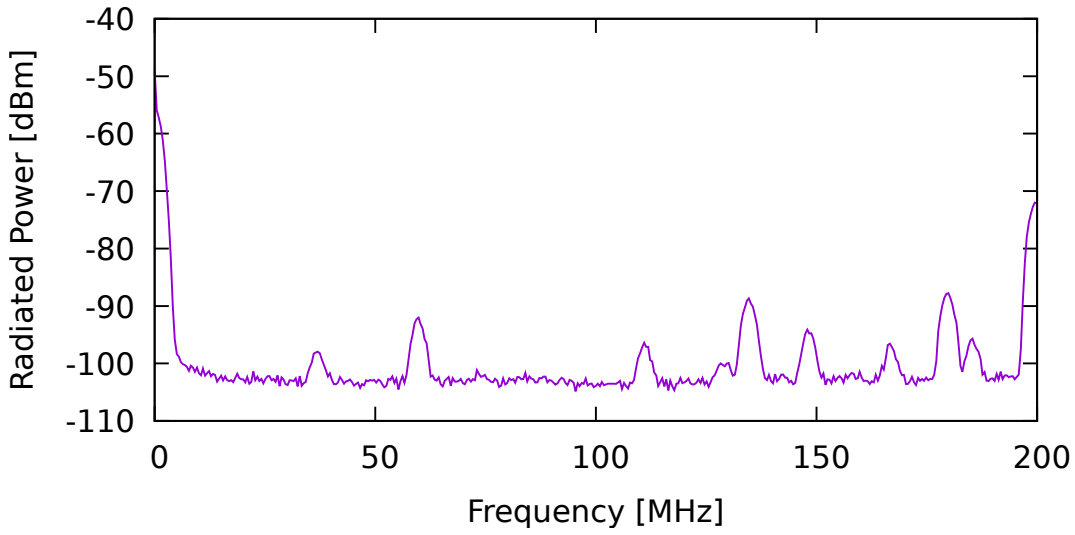

Figure 4.9: Shows the power emitted by the FPGA before it is programmed. When the device is on, but even before it is programmed, it emits non-uniform EMI, with 9 considerable power spikes in our frequency range.

at 135MHz that did not originally appear in my model. Unfortunately, this deviation is unable to be detected by MESC as it is being produced consistently by the FPGA, even when the FPGA is not programmed. Thus, this peak is not being produced by the core that MESC is modeling, but rather by the device in which it is being implemented. Figure 4.9 shows the measured EMI when the FPGA is not programmed with any circuitry at all. As shown, there is a significant spike in EMI at 40, 60, 120, 135, 150, 170, 180, and 200MHz. Therefore, MESC must add these consistent peaks produced by a device after calculating the EMI from the activity rates. Without inserting those artifacts into the final estimated EMI, there would be an extra source of error in MESC. For other applications, artifacts also need to be measured and considered when estimating EMI.

Once the artifacts and the shielding have both been integrated into MESC, I was able to run the set of SPEC2006 benchmarks on the OpenPiton processor. With that information, I was able to determine that MESC provides accuracy within 5% across all benchmarks, with an average deviation of less than 1%.

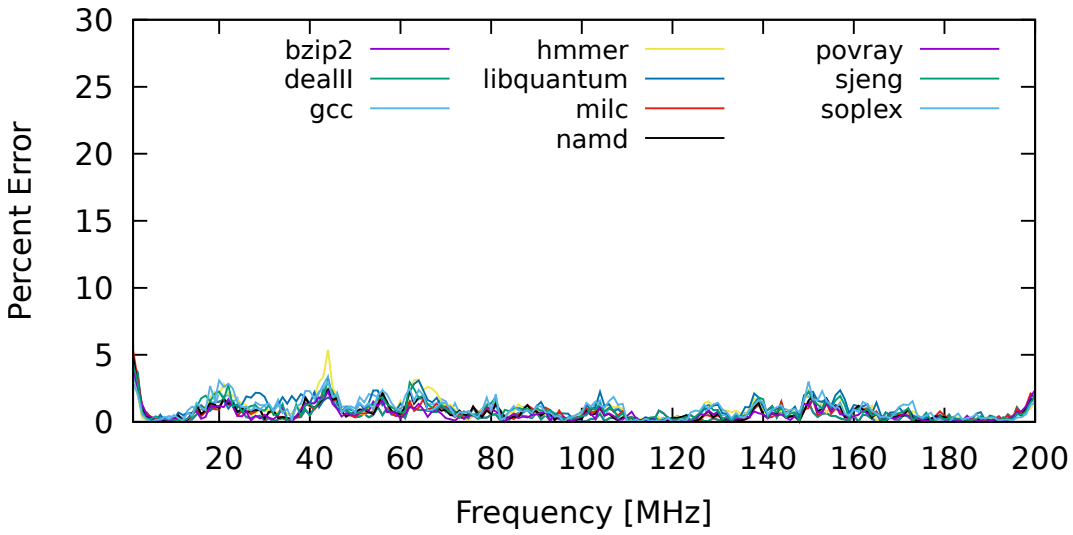

Figure 4.10: Shows the percent error over frequency for each benchmark.

Finally, I verify that MESC is consistent across a set of SPEC benchmarks running on the OpenPiton core. I measure EMI with the spectrum analyzer and calculate the estimated EMI from MESC. Then I take the percentage difference for each frequency. Figure 4.10 shows the percent error over frequency of MESC compared with the measured EMI. The average error across all benchmark is about 1% and the maximum observed error was of 5%.

One interesting thing to note in Figure 4.10 is that around both 20 and 45MHz, there is what seems to be a systematic error that is making the modeled EMI smaller than the measured

EMI. Even though this error is small, it is consistent across benchmarks. This type of systematic error could be easily compensated for, even without knowing what the source of the error is. However, for this work, I decided not to do this compensation, since the magnitude of the error is small and I was unable to find a cause.

4.4 Conclusion for MESC

MESC is an EMI model that architects can integrate with their simulators. The model requires some layout information of the SoC and the ability to get traces for the longest wires. This information allowed me to build an EMI model that can be used in different ways.

My evaluation shows measurements for MESC with less than 5% error compared against multiple OpenPiton FPGA layouts on real FPGA Xilinx hardware. The model details how to choose samples, select wires, perform FFTs, and account for artifacts and package shielding.

Chapter 5

EMI CHopper: DEMIS for Layout

> Don't be too proud of this technological terror you've constructed.
>
> Darth Vader

Once I designed and verified a usable model, I leveraged MESC from Chapter 4 to model the EMI of a core with different layouts, I propose EMI Core Hopper (EMI CHopper), a technique to reduce in-band interference by "hopping" between cores with the same RTL but different layouts[1]. EMI CHopper uses the expected EMI provided by MESC in order to determine which of the different core layouts would produce less in-band interference, and may switch a process to the other core depending on the tradeoffs between processing efficiency and RF interference. I show that EMI CHopper is able to reduce the in-band interference by up to 50%, with very low impact on performance.

The main contributions of this chapter are:

[1]This work is part of the publication I have submitted to The 51st Annual IEEE/ACM International Symposium on Microarchitecture

- Proposing EMI CHopper: a technique for in-band EMI via thread migration
- Validation and evaluation of EMI CHopper

5.1 Related Work

Thread migration, or "core hopping" has been a topic of research for years, with applications ranging from power and leakage optimizations to performance improvements, but I am the first to propose utilizing this technique for reducing in-band EMI.

Kumar *et al.* [31] propose to use heterogeneous cores that implement the same Instruction Set Architecture (ISA). A thread can be migrated from a core to another to reduce power. Authors report a reduction in power of 39% with only a 3% reduction in performance. To enable this type of migration, a Heterogeneity-Aware scheduler [41] is also proposed. The schedule bases its decision on a signature that estimates the performance of a thread in each core. The same type of schedules could be adapted to be used by EMI CHopper, however, in my research, I purely focus on reducing the in-band EMI, and not to combine this with performance metrics. Thus, the only metric used to determine whether to migrate a thread is the estimated EMI of the thread in each core.

A similar approach is evaluated in Alpha cores [9], where only two core sizes are used. One important lesson is that, although there may be advantages of migrating the threads, it is necessary to take into account the overhead of switching the context from a core to the other.

PIE [46] uses a more elaborated performance predictor to decide whether and to

which cores threads should be migrated. It collects CPI stack, MLP and ILP profile information to estimate performance in each core. The recommended approach is to use dynamic scheduling, where migration overhead needs to be taken into account. The paper also shows that there is an impact on whether or not a shared Last Level Cache (LLC) is present or not. A shared LLC can reduce the performance overhead of migration of cache to less than 2%.

I propose using these techniques or similar ones to implement EMI CHopper, but evaluating these techniques lies outside the scope of my research.

5.2 Flow

Now that we can estimate the EMI emitted by a core using MESC, I want to use the estimate to reduce the EMI emitted in a specific communication band. The main objective is to clean bands that are being used for communication of interference. EMI CHopper proposes hopping from core to core to reduce in-band EMI. This is usually referred to as thread migration and has been proposed for performance and power [9, 31, 46]. In this chapter, I propose using two cores that are architecturally identical, *i.e.*, will yield the same performance and power, but that have different layouts. The difference in layout will affect how EMI is produced for a given process running in each core, which is the basis for our proposal.

EMI CHopper works as a dynamic scheduler that estimates EMI for each available core, using MESC, when wireless communication is being used. The layout profiles from MESC for each core are known at design time and are static. If EMI in the band of interest would be reduced by running the application in a different core, EMI CHopper may decide to

migrate the thread. To prevent migrating for trivial reductions in EMI, EMI CHopper takes a threshold parameter, and thus the migration only occurs if the reduction in EMI is larger than the threshold.

5.2.1 Modeling thread migration cost

It is important to note that there is some execution overhead associated with migrating a thread [9, 18]. The thread migration cost may be elevated, in particular when considering rebuilding the state of the cache and branch predictor [9, 46]. There is a clear tradeoff between paying the cost of transferring cache, branch predictor, etcbetween cores or going through a warm-up period, where these structures will be organically filled during execution. The most natural solution is to not transfer those states and let the execution take care of it, which is assumed throughout this thesis.

The migration cost comes from various sources, ordered here in order of magnitude (least impact first):

1. Migrating the architectural state (PC, RF, etc)
2. Warming-up branch predictor tables, prefetcher state, etc
3. Warming-up the cache state

Overall, a few clock cycles can be assumed to transfer the architectural state between cores [9]. For branch predictors, prefetcher and other predictor tables, it is harder to estimate the impact, but since their state is not transferred in this approach, it will show as a reduction in IPC due to mispredictions. For caches, the impact can be potentially throughout a large number of cycles,

since caches take much longer to warm-up [18]. However, by using a shared last level cache (LLC), even after migration, the penalty of misses will be largely reduced [46]. In fact, when using a shared LLC, even by migrating every 2.5ms, the performance impact is expected to be lower than 1%.

The impact on performance is dependent on the frequency at which the threads are migrated [18]. For wireless communication and EMI reduction, there is no need for fine grained thread migration, thus I do not expect frequent migrations. For EMI CHopper, I limit the migration to at most one thread migration per second, which should reduce any impacts on performance. In my evaluation, I show that even 1s migration is frequent enough for any communication purposes.

5.2.2 Implementing EMI CHopper in real hardware

EMI CHopper requires each core to keep track of the activity in the longest wires, and then calculate FFT for each wire. This requires some dedicated hardware. Since keeping 1024 samples for 600 wires would imply in too much storage, EMI CHopper proposes taking samples of each wire at a time. Thus, EMI CHopper requires only 1024 bits of storage, pipelined FFT implementation and some arithmetic circuitry per core.

Taking one EMI CHopper sample will thus take 600 wires $\times$1024 cycles $=\approx 600K$ cycles. At $1GHz$, this means that is possible to calculate over $16k$ samples per second. This is more than enough for 1 hop every second proposed by EMI CHopper. However, more buffering and a larger number of FFT accelerators can be used to speed-up the process.

The overall implementation of EMI CHopper is illustrated in Figure 5.1. Each core

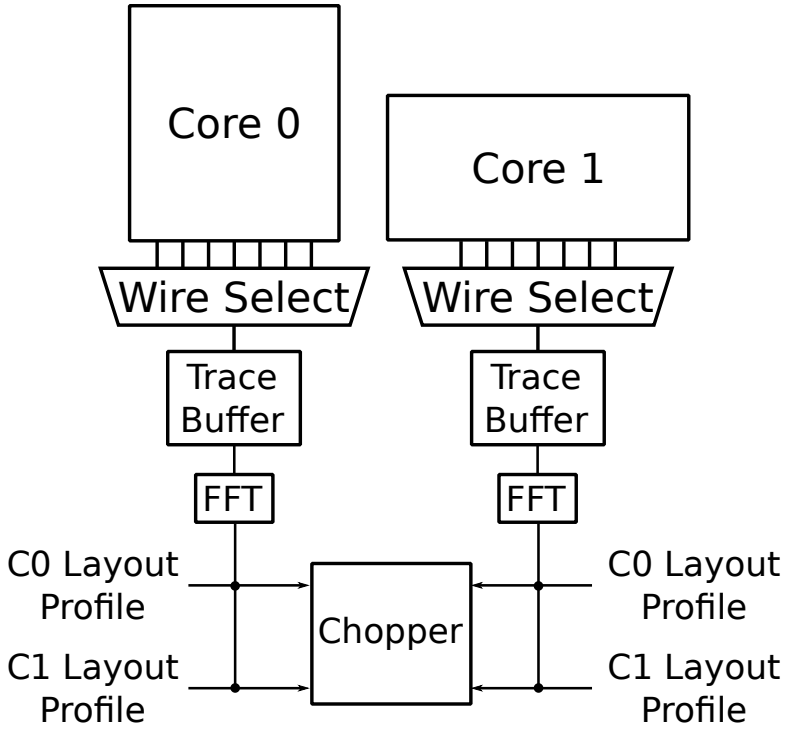

Figure 5.1: EMI CHopper requires a single FFT and arithmetic operations in addition to a small storage to calculate MESC for each application in each core and decide whether or not to migrate threads from a core to the other.

provides EMI CHopper with the execution traces for the monitored wires. The FFT is performed individually per wire trace and then multiplied by wire length from the layout profile information. The aggregated data will guide the EMI CHopper decision to hop applications from one core to the other.

5.3 Evaluation

After determining the accuracy and validity of MESC, I am able to apply it to implement and verify EMI CHopper. In this section, I evaluate the potential of EMI CHopper to reduce EMI in specific frequency bands.

The two layouts were generated by slightly changing the placement parameters in Vivado. Manual placement or floorplanning could also have been used but would, most likely result in a change in the frequency achieved by synthesis. The parameters that I used allowed the core to be run at the same frequency, which was desirable to evaluate EMI CHopper.

Figure 5.2 shows the EMI produced by sjeng in each of the layouts. In the top the EMI is shown as a heat map for frequencies between 0 and $200MHz$ over time and in the bottom for $100MHz$ only, but for two layouts. My model shows a significant change in EMI as an application goes through different phases. Furthermore, when comparing two different layouts, we can clearly see that in one case, after the phase change, the EMI increases, but the opposite occurs for the other layout. In the figure, a phase change occurs right after 15 seconds of execution, and Layout 1 has a decrease in EMI at the $100MHz$ frequency, whereas Layout 2 has an increase for $100MHz$.

To assess the impact on performance, I look into how frequently EMI CHopper would have a thread migrate from a core to the other. For each benchmark, I obtained switching activity samples for 30 seconds in order to evaluate EMI CHopper. I was able to obtain the MESC model over time for each of these samples on the two layouts described above. From there, I analyzed the number of hops a thread would make to reduce EMI. Figure 5.3 shows the number of migrations a thread would perform if it were to switch cores every time the core on which it is executing would have higher in-band EMI than the other core within a threshold. Across all benchmarks, dealII had the most hops (18), and namd had the least, only 5.

However, because there is a migration overhead associated with moving a process from one core to another, I also investigated changing constraints to how frequently a thread

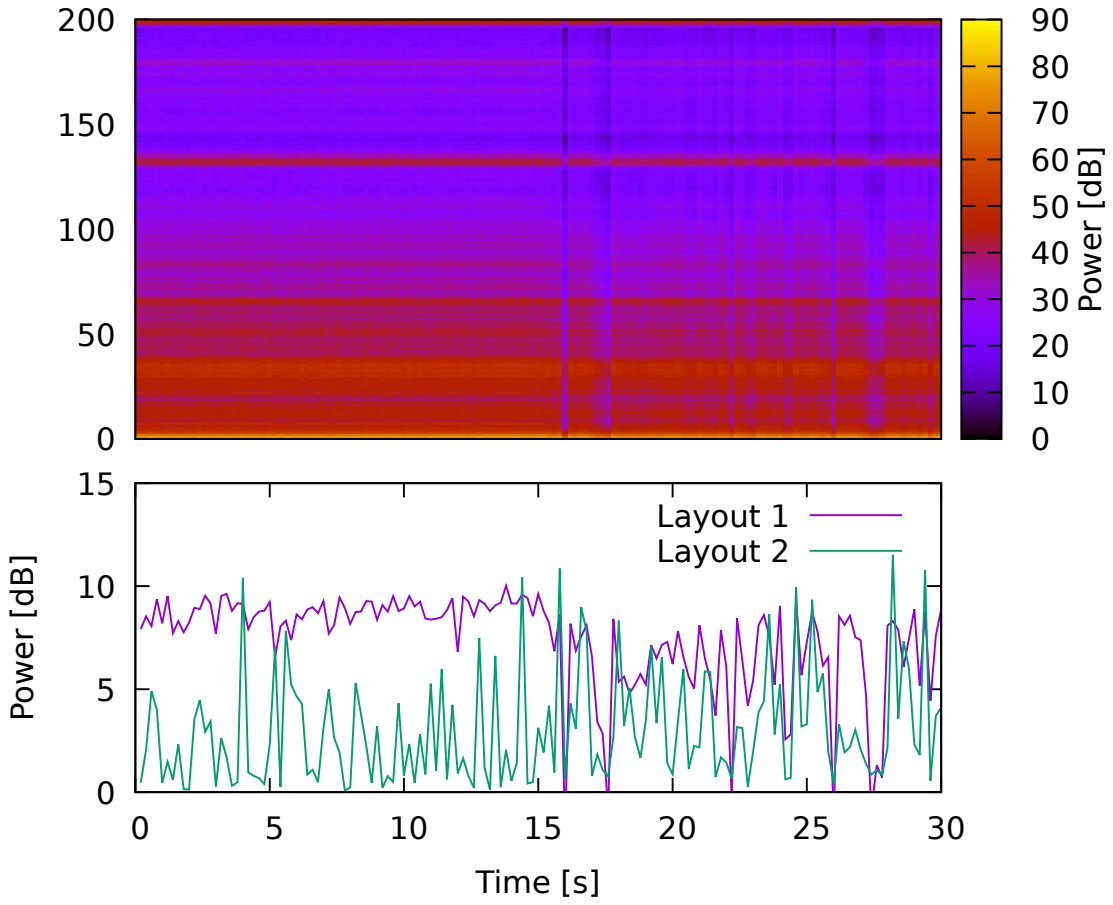

Figure 5.2: Interference over time for two different layouts, 100MHz, sjeng. On the top is the EMI for a single layout over time at all frequencies from 0 to 200MHz. On the bottom is the EMI over time for two different layouts for 100MHz.

can hop between core. Figure 5.4 shows the correlation between the migration frequency and the EMI reduction at 100MHz.

For my default constraint (a maximum of one migration per second), I was able to

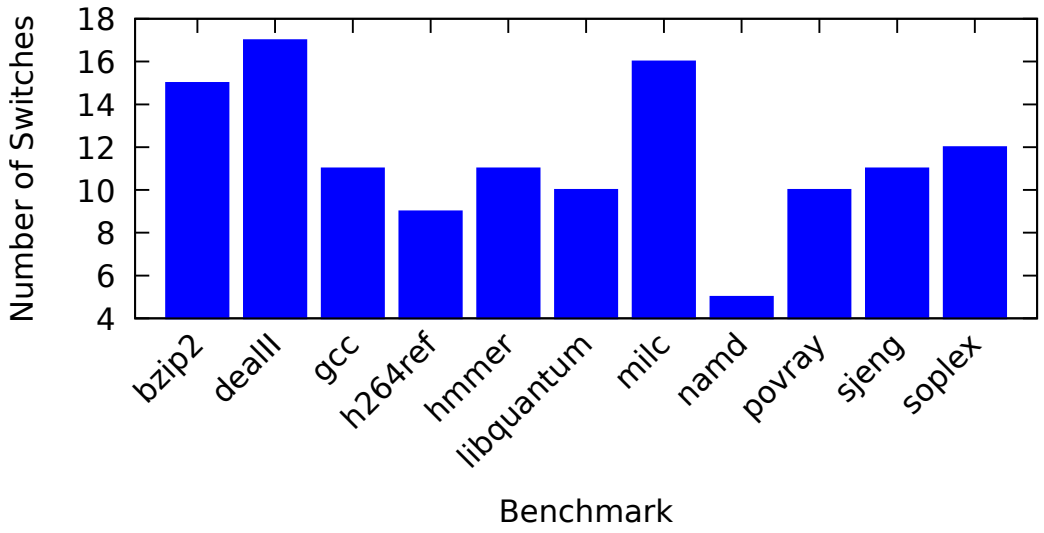

Figure 5.3: EMI CHopper migrates the thread between cores in order to reduce EMI. I limit the number of switches per benchmark for a maximum of one migration every 1 second. The figure shows the number of times EMI CHopper decided to migrate each application over a period of 30 seconds.

see an average reduction of 37% of the EMI power across all frequencies, or about $2dB$ by reducing the constraints to a maximum migration frequency of $10\times$ per second (once every $0.1s$). However, there was an average reduction of 47% in EMI power for the $100MHz$ band.

5.4 Conclusion for EMI CHopper

I propose EMI CHopper, a method that uses MESC to reduce in-band interference for a multi-core processor. EMI CHopper proposes a homogeneous multi-core with two different core layouts, but the same RTL. It shows that migrating between two cores at low granularity (like every second) can reduce power across all frequencies by 37%. If a single frequency band

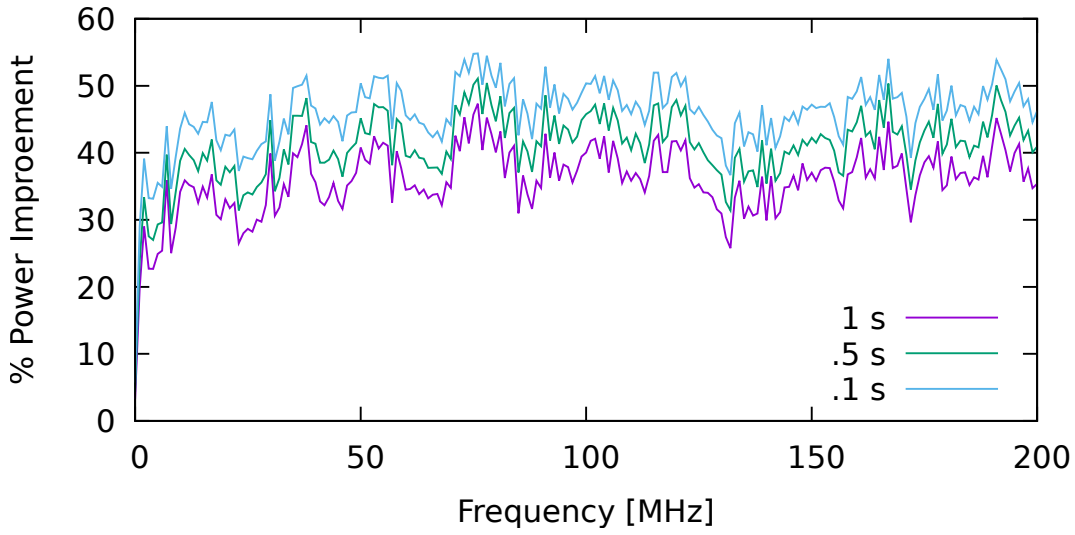

Figure 5.4: I evaluated how the decision of only allowing one switch every second affected the EMI. Switching every $0.1s$ would reduce the EMI through time, but the difference is small. When migrating more often, the impact on performance is expected to be higher.

is targeted, the reduction is even higher.

Chapter 6

DEMIS for Existing Processors

> Actually, all I have is the phrase "I have a foolproof plan." Beyond that, I'm wide open
>
> Shawn Spencer

This work proposes a DEMIS enabled platform for existing processors, which leverages the insights about techniques from Chapter 3 to dynamically shift the interference to out of a band of interest after a processor has already been taped out.[1] Notice that DEMIS does not reduce noise in all the bands like static EMI techniques. Instead it reduces the interference in the band currently used for communication. Figure 6.1 shows my proposal for the DEMIS system, which will monitor its own EMI and adjust its execution accordingly. The layout of this system is based on the Snapdragon 821 layout, which contains the Qualcomm Snapdragon X12 LTE modem and the Qualcomm Kyro CPU on die, but has an additional small DEMIS unit (in

[1] This work is part of my publication at The 50th International Symposium on Microarchitecture, October 2017 [23].

grey) that provides directives to the CPU based on the interference.

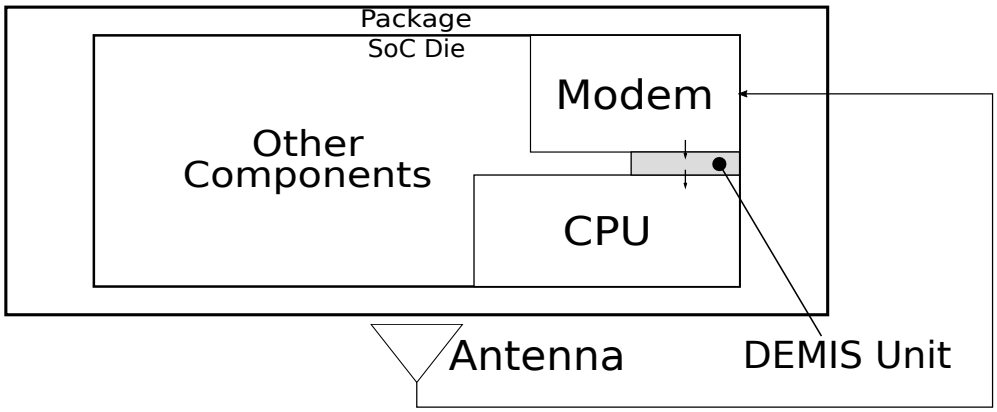

Figure 6.1: Proposed DEMIS architecture monitors the EMI and provides directives to the processor to change emitted interference.

The main contributions presented in this chapter are:

- Proposing DEMIS for existing processors, a dynamic methodology based on EMI measurements to reduce in-band EMI during runtime via manipulation of architectural and compiler parameters
- Evaluation of the impact DEMIS has on EMI and processor performance

6.1 Related Work

As shown throughout my research, the EMI signature of a process is dependent on several factors and is different when run on processors with the same RTL but manufactured in different fabs. This makes it virtually impossible to know beforehand the EMI signature of a process. Furthermore, a processor will not know which bands will be used for communication during the runtime of that process. Hence the decision of which techniques to use to reduce

in-band noise needs to be done at runtime. This section highlights some current techniques that a DEMIS processor could utilize, but evaluation of these techniques is out of the scope of this thesis and is reserved for future work.

With DEMIS, I leverage the fact that the EMI signature of a process can be changed by changing architectural parameters such as introducing delays, changing compiler options and so on. This is not possible with regular compilation/execution flows, as those changes need to be made at compile time and cannot be changed during runtime. However, there are tools a DEMIS processor can utilize to change how a process is executed at runtime such as Just In Time compilers (JITs) and interpreted languages. For non-JIT systems, it is possible to switch binaries at runtime, assuming multiple binaries compiled with different compilation options are available. Though this is not standard operation, it is not hard to do so form an OS perspective.

Most mobile devices already use JITs, *e.g.*, the discontinued Dalvik [12] and the new JIT introduced in Android 7.0 Nougat [19]. Furthermore, many JITs have been developed for Java [4, 44, 50]. JITs are also widely used during internet browsing on all platforms, with a strong emphasis on JavaScript [17,20,25,37,39,49]. Although these tools are almost exclusively geared towards efficiency, they can be repurposed to minimize in-band EMI.

In addition to JITs, there are other tools that dynamically switch how a process is being executed. For example, Dynamo [7] switches between binaries during execution in order to capitalize on runtime optimization opportunities. There are also many other tools that switch binaries at runtime.

I propose utilizing these techniques for implementing a DEMIS processor, but evaluation of these techniques does not fall within the scope of my research.

6.2 Evaluation

While Chapter 3 provided insights, this section evaluates a DEMIS approach. In this section, I evaluate the efficacy of the techniques proposed in Chapter 3 using the same hardware running SPEC2006 benchmarks. Once again, I utilized the hardware described in Table 3.1 and performed measurements using the same N9342C Handheld Spectrum Analyzer from Agilent Technologies and Keysight Technologies' Near Field Probe Set.

I propose utilizing the techniques in Section 6.1 for a DEMIS-enabled core to mitigate the EMI from the processor during execution. As shown in Figure 6.1, a DEMIS processor will be able to monitor its own EMI, and when the in-band interference exceeds a certain threshold, will switch to another processing configuration. Because DEMIS is only useful for devices with wireless communication, I suggest using the existing antenna on the device to monitor the EMI, leveraging the data from the modem to determine when the processor is generating significant interference. However, the purpose of this chapter is to show that using computer architectural techniques is effective in mitigating in-band EMI, and therefore designing and implementing this fully DEMIS-enabled core is beyond the scope of my research.

To evaluate DEMIS, I run all the different compile options, core, and DDR frequencies. I evaluate the effectiveness for each of the 5 analyzed bands (LTE 800, LTE 700, LTE 450, WiFi XR7 and WiFi 600). For the baseline, I pick the compile option and frequencies with the highest performance. For the DEMIS solution, I pick the configuration with the highest interference reduction as long as it that does not have more than 10% slowdown. To illustrate the selection process, Figure 6.2 shows all the options for just the LTE 800 band in an Exynos

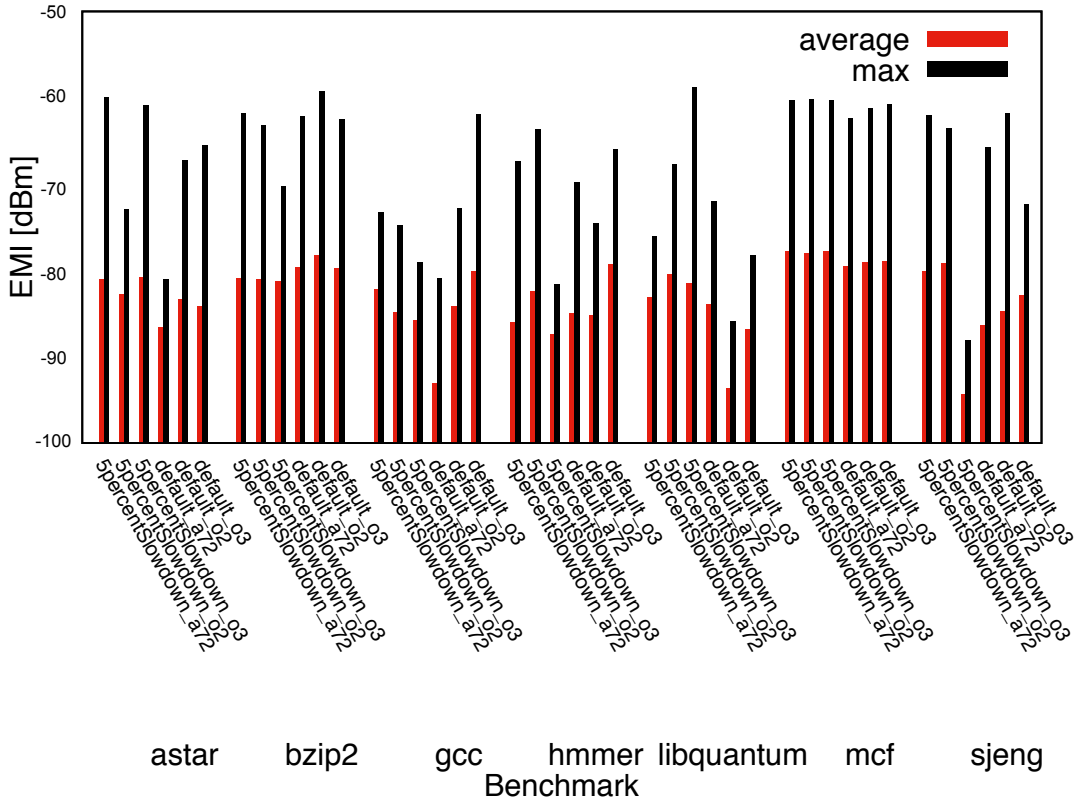

Figure 6.2: Interference for all the configurations in the LTE 800 band with the A15_XU4 core.

A15 core. Each bar shows the interference level. For the baseline and hmmer run, the fastest (speed not shown on the plot) is the configuration with default CPU speeds and using the -O2 compilation option with the default scheduling algorithm. The lowest EMI with less than 10% slowdown is the configuration with the CPUs running with a 5% slowdown and using the O3 optimization. While the first configuration is selected for the baseline hmmer point, the second configuration is selected for the DEMIS hmmer point.

I apply this process over all the bands. The result is shown in Figure 6.3. This plot summarizes the main results with a 6% average EMI reduction, with an average of only 5.5%

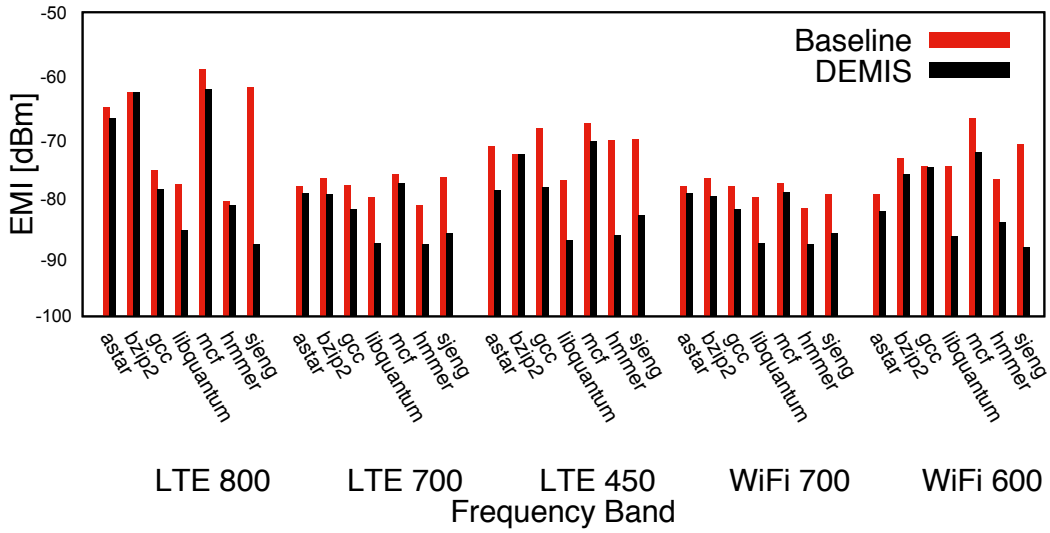

Figure 6.3: DEMIS has a significant EMI reduction across all the bands on the A15_XU4.

performance reduction

While the main results only show data for the A15 core, I performed the same process on all the platforms. Different cores have different benefits, the A11_PI2 has an average interference reduction of 6.5% across the five bands analyzed with the highest reduction being in the SuperWiFi XR7 band with a $8dB$ reduction. This is done with less than 10% slowdown.

6.3 Conclusion for DEMIS for Existing Processors

The main conclusion is that DEMIS provides opportunities to reduce interference across the different frequency bands. Doing architectural and frequency changes I have been able to measure EMI reductions from 3 to over 8dB across multiple platforms. I do so capping

the maximum slowdown under 10% with a modest average of 6% across SPEC2006 applications. I think that these results open a new opportunity to dynamically manage EMI.

Chapter 7

Conclusion and Future Work

> I love deadlines. I like the whooshing sound they make as they fly by.
>
> Douglas Adams

Throughout this thesis, I have addressed the wireless communications problem of in-band EMI from an architectural perspective.

In Chapter 3, I offer insights into multiple different architectural parameters and investigate their effect on EMI, including compilation settings, applications, caches, memory, and execution core. My work on this topic introduced to the computer architecture community the wireless communications challenge of EMI, and the opportunities for applying architectural techniques to address the challenge in dynamic solutions.

In Chapter 6, I proposed and evaluated DEMIS, my purely architectural solution that integrates the dynamic manipulation of the techniques described in Chapter 3 to reduce the interference in the frequency bands used by cellular, WiFi, and Bluetooth communication tech-

nologies. My results show that a $15dB$ EMI reduction for LTE can represent over $3\times$ bandwidth improvement for EMI bound communication.

Chapter 4 describes and validates MESC, the first architectural framework for modeling emission from a core. MESC takes into account the switching activity of a process and some layout information to provide an approximate model for EMI. MESC is validated against a core running on an FPGA, and I

Chapter 5 proposes EMI CHopper, which uses a multi-core system - where each core has the same RTL but different layouts - and causes thread migrations between these cores to reduce in-band EMI when it interferes with wireless communications. Using EMI CHopper, I was able to reduce in-band EMI by up to 50%, with a low impact on performance.

7.1 Future work

My research shows that more research on RF interference emitted by processors with on-chip FPGAs and out of order processors should be conducted. Additionally, current wireless standards take into account a small amount of noise. Therefore, improving the SNR as I do using my techniques would open up opportunities for improved bandwidth, as these standards would be able to take into account a higher maximum throughput as the EMI decreases. Furthermore, I would recommend looking into dynamically switching between binaries of the same code (with different compilation arguments) during runtime, and if it would be a feasible solution for mitigating in-band EMI as a program enters noisy phases. Additionally, DRAM clock modulation and introducing a time delay between two cores are interesting prospects that

I was unable to fully test due to lack of resources.

Furthermore, the introduction of MESC allows for many opportunities of future work. For example, the power vs EMI tradeoff described in Chapter 4 can utilize MESC in order to determine the effect of increasing the clock frequency, but only sending values over the bus every couple of cycles in an attempt to change the EMI from the clock, but not requiring changing the processor speed. However, there would be a significant power consumption increase, so MESC can be used as a tool for analyzing these tradeoffs. Another application for MESC would be for architects to study the effects of introducing activity in wires, which would increase power but would move the EMI frequency. Similarly, researchers can combine MESC with a DVFS and estimate the EMI reduction.

Expanding on EMI CHopper, which assumes duplicates of the same RTL but different layouts, a logical evolution would be to have to different types of cores.

The research presented in this thesis offers a new, architectural perspective on a prevalent problem in wireless communications. My findings based purely on architectural techniques show significant promise in improving wireless communications, and the tools I described offer many opportunities for further research.

Bibliography

[1] 3GPP. 3GPP TS 36.211: E-UTRA:Physical channels and modulation. www.3gpp.org/dynareport/36211.htm.

[2] C. Alpert and A. Devgan. Wire segmenting for improved buffer insertion. In *Proceedings of the 34th Annual Design Automation Conference*, DAC '97, pages 588–593, New York, NY, USA, 1997. ACM.

[3] H. Ansoft. ver. 11. *Ansoft Corporation, Pittsburgh, PA*, 2007.

[4] J. Aycock. A brief history of just-in-time. *ACM Computing Surveys (CSUR)*, 35(2):97–113, 2003.

[5] N. Azuma, T. Makita, S. Ueyama, M. Nagata, S. Takahashi, M. Murakami, K. Hori, S. Tanaka, and M. Yamaguchi. In-system diagnosis of RF ICs for tolerance against on-chip in-band interferers. In *Test Conference (ITC), 2013 IEEE International*, pages 1–9. IEEE, 2013.

[6] B. Baker. EMI problems? part two: Where does EMI come from?

2012. http://www.edn.com/electronics-blogs/bakers-best/4369076/EMI-problems-Part-two-Where-does-EMI-come-from-.

[7] V. Bala, E. Duesterwald, and S. Banerjia. Dynamo: A Transparent Dynamic Optimization System. In *ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 1–12, Vancouver, Canada, Jun. 2000.

[8] J. Balkind, M. McKeown, Y. Fu, T. Nguyen, Y. Zhou, A. Lavrov, M. Shahrad, A. Fuchs, S. Payne, X. Liang, M. Matl, and D. Wentzlaff. Openpiton: An open source manycore research framework. In *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '16, pages 217–232, New York, NY, USA, 2016. ACM.

[9] M. Becchi and P. Crowley. Dynamic thread assignment on heterogeneous multiprocessor architectures. In *Proceedings of the 3rd Conference on Computing Frontiers*, CF '06, pages 29–40, New York, NY, USA, 2006. ACM.

[10] Bluetooth. What is bluetooth technology? https://www.bluetooth.com/what-is-bluetooth-technology. Accessed: 2016-04-04.

[11] Bluetooth. Wi-fi and bluetooth - interference issues. Technical report, HP, January 2002. Accessed: 2016-04-04.

[12] D. Bornstein. Dalvik VM internals. In *Google I/O developer conference*, volume 23, pages 17–30, 2008.

[13] T. D. Burd and R. W. Brodersen. Design issues for dynamic voltage scaling. In *Low*

Power Electronics and Design, 2000. ISLPED '00. Proceedings of the 2000 International Symposium on, pages 9–14, July 2000.

[14] R. Callan, F. Behrang, A. Zajic, M. Prvulovic, and A. Orso. Zero-overhead profiling via em emanations. In *Proceedings of the 25th International Symposium on Software Testing and Analysis*, ISSTA 2016, pages 401–412, New York, NY, USA, 2016. ACM.

[15] R. Callan, A. Zajić, and M. Prvulovic. A practical methodology for measuring the side-channel signal available to the attacker for instruction-level events. In *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 242–254. IEEE Computer Society, 2014.

[16] R. Callan, A. Zajic, and M. Prvulovic. FASE: finding amplitude-modulated side-channel emanations. In *Computer Architecture (ISCA), 2015 ACM/IEEE 42nd Annual International Symposium on*, pages 592–603. IEEE, 2015.

[17] J. Conrod. A tour of V8: full compiler, Dec. 2013.

[18] T. Constantinou, Y. Sazeides, P. Michaud, D. Fetis, and A. Seznec. Performance implications of single thread migration on a chip multi-core. *SIGARCH Comput. Archit. News*, 33(4):80–91, Nov. 2005.

[19] P. Dean. Here's everything new in android nougat 7.1. 2016.

[20] X. Fuqiao, F. Scholz, K. Scarfone, B. Peksag, T. Schneidereit, E. Shepherd, and C. Leary. Tracing JIT, May. 2014.

[21] R. Getz and B. Moeckel. Understanding and eliminating EMI in microcontroller applications. *National Semiconductor*, 1996.

[22] N. Gompa. Deep dive: What is LTE? http://www.extremetech.com/mobile/110711-what-is-lte, April 2015. Accessed: 2016-04-04.

[23] D. I. Gorman, M. R. Guthaus, and J. Renau. Architectural opportunities for novel dynamic EMI shifting (DEMIS). In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-50 '17, pages 774–785, New York, NY, USA, 2017. ACM.

[24] M. Graphics. Hyperlynx signal integrity simulation software, 2004.

[25] B. Hackett and S.-y. Guo. Fast and precise hybrid type inference for javascript. *SIGPLAN Not.*, 47(6):239–250, Jun. 2012.

[26] K. B. Hardin, J. T. Fessler, and D. R. Bush. Spread spectrum clock generation for the reduction of radiated emissions. In *Electromagnetic Compatibility, 1994. Symposium Record. Compatibility in the Loop., IEEE International Symposium on*, pages 227–231. IEEE, 1994.

[27] K. B. Hardin, J. T. Fessler, and D. R. Bush. A study of the interference potential of spread spectrum clock generation techniques. In *Electromagnetic Compatibility, 1995. Symposium Record., 1995 IEEE International Symposium on*, pages 624–629. IEEE, 1995.

[28] J. L. Henning. SPEC CPU2006 benchmark descriptions. *SIGARCH Comput. Archit. News*, 34(4):1–17, Sept. 2006.

[29] X. Hu and M. R. Guthaus. Clock tree optimization for electromagnetic compatibility (EMC). In *Proceedings of the 16th Asia and South Pacific Design Automation Conference*, pages 184–189. IEEE Press, 2011.

[30] S. I-The, A. Chen, and J. Keip. Spread spectrum and PLL technology combine to reduce EMI. *RF DESIGN*, 23(4):20–25, 2000.

[31] R. Kumar, K. I. Farkas, N. P. Jouppi, P. Ranganathan, and D. M. Tullsen. Single-isa heterogeneous multi-core architectures: The potential for processor power reduction. *Microarchitecture, IEEE/ACM International Symposium on*, 0:81, 2003.

[32] A. U. Manual. Ansys. *Inc. Modeling, CFX*, 11, 2000.

[33] M. A. McHenry, D. Roberson, and R. J. Matheson. Electronic noise is drowning out the internet of things. *IEEE Spectrum: Technology, Engineering, and Science News, Available at: http://spectrum. ieee. org/telecom/wireless/electronic-noise-is-drowning-out-the-internet-of-things.[Accessed: 3 Jun. 2016]*, 2015.

[34] A. Nazari, N. Sehatbakhsh, M. Alam, A. Zajic, and M. Prvulovic. EDDIE: EM-based detection of deviations in program execution. *SIGARCH Comput. Archit. News*, 45(2):333–346, Jun. 2017.

[35] L. J. Oh. RF desense story 2. 2017. https://www.linkedin.com/pulse/rf-desense-story-2-leo-janghwan-oh.

[36] S. J. Orfanidis. Electromagnetic waves and antennas, 2008. *Unpublished, available: http://www. ece. rutgers. edu/orfanidi/ewa*, 2004.

[37] K. Patil. JaegerMonkey architecture, Aug. 2011.

[38] E. Rolf, A. Petersson, O. Samuelsson, and P. Nelderup. Desense with adaptive control, Aug. 6 2009. US Patent App. 12/025,254.

[39] F. Scholz. SpiderMonkey internals, May. 2014.

[40] N. Sehatbakhsh, A. Nazari, A. Zajic, and M. Prvulovic. Spectral profiling: Observer-effect-free profiling by monitoring EM emanations. In *Microarchitecture (MICRO), 2016 49th Annual IEEE/ACM International Symposium on*, pages 1–11. IEEE, 2016.

[41] D. Shelepov, J. C. Saez Alcaide, S. Jeffery, A. Fedorova, N. Perez, Z. F. Huang, S. Blagodurov, and V. Kumar. Hass: A scheduler for heterogeneous multicore systems. *SIGOPS Oper. Syst. Rev.*, 43(2):66–75, Apr. 2009.

[42] Q. S. F. D.-C. Snapdragon. Chipset, 2010.

[43] M. Studio. CST-computer simulation technology. *Bad Nuheimer Str*, 19:64289, 2008.

[44] T. Suganuma, T. Ogasawara, M. Takeuchi, T. Yasue, M. Kawahito, K. Ishizaki, H. Komatsu, and T. Nakatani. Overview of the IBM java just-in-time compiler. *IBM systems Journal*, 39(1):175–193, 2000.

[45] R. Thomas, N. Sedaghati, and R. Teodorescu. EmerGPU: Understanding and mitigating resonance-induced voltage noise in GPU architectures. In *2016 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 79–89, April 2016.

[46] K. Van Craeynest, A. Jaleel, L. Eeckhout, P. Narvaez, and J. Emer. Scheduling heterogeneous multi-cores through performance impact estimation (PIE). In *Proceedings of the 39th Annual International Symposium on Computer Architecture*, ISCA '12, pages 213–224, Washington, DC, USA, 2012. IEEE Computer Society.

[47] C. Wang, R. Callan, A. Zajic, and M. Prvulovic. An algorithm for finding carriers of amplitude-modulated electromagnetic emanations in computer systems. In *2016 10th European Conference on Antennas and Propagation (EuCAP)*, pages 1–5, April 2016.

[48] F. Werner, D. A. Chu, A. R. Djordjevic, D. I. Olcan, M. Prvulovic, and A. Zajic. A method for efficient localization of magnetic field sources excited by execution of instructions in a processor. *IEEE Transactions on Electromagnetic Compatibility*, 60(3):613–622, June 2018.

[49] A. Wingo. v8: a tale of two compilers, Jul. 2011.

[50] B.-S. Yang, S.-M. Moon, S. Park, J. Lee, S. Lee, J. Park, Y. C. Chung, S. Kim, K. Ebcioglu, and E. Altman. LaTTe: A java VM just-in-time compiler with fast and efficient register allocation. In *Parallel Architectures and Compilation Techniques, 1999. Proceedings. 1999 International Conference on*, pages 128–138. IEEE, 1999.