APPLIED HW/SW CO-DESIGN: Using the Kendall Tau Algorithm for

Adaptive Pacing

A Thesis

presented to

the Electrical Engineering Department Faculty of

California Polytechnic State University,

San Luis Obispo

In Partial Fulfillment

of the Requirements for the

Master of Science Degree in Electrical Engineering

By

Kenneth Chee

June 2013

© 2013

Kenneth Chee

ALL RIGHTS RESERVED

COMMITTEE MEMBERSHIP

TITLE: Applied HW/SW Co-design: Using Kendall Tau Algorithm
for Adaptive Pacing

AUTHOR: Kenneth Chee

DATE SUBMITTED: June 2013

COMMITTEE CHAIR: Dr. Bryan Mealy
(Associate Professor Dept of Electrical Engineering)

COMMITTEE MEMBER: Dr. Bridget Benson
(Assistant Professor Dept of Electrical Engineering)

COMMITTEE MEMBER: Dr. John Oliver
(Assistant Professor Dept of Electrical Engineering)

ABSTRACT

Applied HW/SW Co-design: Using Kendall Tau Algorithm for Adaptive Pacing

Kenneth Chee

Microcontrollers, the brains of embedded systems, have found their way into every aspect of our lives including medical devices such as pacemakers. Pacemakers provide life supporting functions to people therefore it is critical for these devices to meet their timing constraints. This thesis examines the use of hardware co-processing to accelerate the calculation time associated with the critical tasks of a pacemaker. In particular, we use an FPGA to accelerate a microcontroller's calculation time of the Kendall Tau Rank Correlation Coefficient algorithm. The Kendall Tau Rank Correlation Coefficient is a statistical measure that determines the pacemaker's voltage level for heart stimulation. This thesis explores three different hardware distributions of this algorithm between an FPGA and a pacemaker's microcontroller. The first implementation uses one microcontroller to establish the baseline performance of the system. The next implementation executes the entire Kendall Tau algorithm on an FPGA with varying degrees of parallelism. The final implementation of the Kendall Tau algorithm splits the computational requirements between the microcontroller and FPGA. This thesis uses these implementations to compare system-level issues such as power consumption and other tradeoffs that arise when using an FPGA for co-processing.

Keywords: Hardware/Software Co-design, Microcontroller, FPGA, Pacemaker, Pacing, Kendall Tau

ACKNOWLEDGMENTS:

I would first and foremost like to thank both my St. Jude Medical contact Eli Snell and my thesis advisor Bryan Mealy. They have guided me in the development of my thesis, academic and professional perspective.

I also greatly would like to thank my family for their never ending support for me. Surely without them I would have fallen and never gotten back up. Thanks mom and dad for your guidance in life.

This project was presented via Lily Laiho for the Meditec program at the California Polytechnic Institute of San Luis Obispo.

Contents

List of Tables	viii
List of Figures	ix
Chapter 1: Purpose	1
1.1 Introduction	1
1.2 Motivations.....	2
1.3 Main Contributors	3
1.4 Thesis Organization.....	3
Chapter 2: Background	5
2.1 Embedded Systems Perspective	5
2.2 Biomedical Perspective	9
2.3 Related Works	12
Chapter 3: Kendall Tau Algorithm	16
3.1 The Algorithm	16
3.2 Decomposition of the Kendall Tau Algorithm.....	18
Chapter 4: Implementation and Theory	21
4.1 Hardware	21
4.2 Implementation of Kendall Tau on a MCU	22
4.3 Implementation of Kendall Tau on a FPGA	25
4.4 Implementation of Kendall Tau on a MCU and FPGA	32

4.5 Measurement Methodology.....	35
Chapter 5: Results	40
5.1 Time to Calculate Kendall Tau Coefficient	40
5.2 Power Consumption	43
5.3 Difference in FPGA area consumption Single vs. Double.....	46
5.4 Difference in FPGA area consumption FPGA only vs. MCU/FPGA.....	46
Chapter 6: Conclusion.....	48
6.1 Overall conclusion.....	48
6.2 Future Work	48
References.....	51

List of Tables

Table	Page
1. This table shows the processing times for the various implementations	40
2. Shows the measured Current/Voltage/Instantaneous power for each implementation	44
3. The power cost to calculate the Kendall Tau Algorithm	45
4. Resource usage for the Single VS Double FPGA implementations.....	46
5. Resource usage of the FPGA single and MCU/FPGA	47

List of Figures

Figure	Page
1	A diagram showing the Atmega328P architecture. The arrows denote the flow of data between RAM and ALU in the Atmega32P..... 6
2	(a) The first commercial implantable cardiac pacemaker (b) and the modern day pacemaker developed by St Jude Medical..... 9
3	A photograph showing the internal battery and oscillator in a circa 1960 pacemaker..... 10
4	The features of a QRS complex that are displayed on an ECG..... 13
5	A graph showing the Number of calculations vs Set size to complete the Kendall Tau Algorithm..... 19
6	The atmega328p MCU on the Arduino Uno development board....21
7	The Spartan 3e FPGA mounted on the Nexys2 development board. 22

8	(a) A black box diagram showing the inputs and outputs of the MCU implementation, and (b) the MCU implementation showing the “pulse” and “pulse2” lines connected to a scope for read out and UART connection through US.....	23
9	A plot of the Number of Executed Instructions vs Set Size showing the growth function for the MCU implementation.....	24
10	(a) A black box diagram showing the inputs and outputs of the FPGA implementation, and (b) a photograph of the FPGA implementation showing the parallel data, and control lines.....	26
11	A black box diagram for the internal FPGA modules (FPGA implementation).....	27
12	A timing diagram showing the 4-stage handshake for REQ and ACK lines	28
13	The data points used per cycle of the Kendall Tau comparison for a single hardware set.....	29

14	The data points used per cycle of the Kendall Tau comparison for a double hardware set.....	31
15	A graph showing the Number of Executed Instructions vs Set Size in order to compare MCU, FPGA Single, FPGA Double implementations.....	32
16	a) A black box diagram showing the inputs and outputs of the MCU/FPGA implementation, and b) the MCU/FPGA implementation showing the parallel data, control lines and “pulse” and “pulse2” lines.....	33
17	A black box diagram for the internal FPGA modules (MCU/FPGA implementation).....	34
18	A graph showing the Number of Executed Instructions vs Set Size comparing MCU, FPGA Single, FPGA Double, MCU/FPGA implementations.....	35
19	A photograph showing the pulse and pulse2 pins from the FPGA to oscilloscope probes.....	37

20	A photograph showing the pulse and pulse2 pins from the MCU to oscilloscope probes.....	37
21	A photograph showing the USB wires to MCU and FPGA spliced with ammeter in series.....	39

Chapter 1: Purpose

This chapter provides general background information regarding the motivations and approach taken by this thesis. This background includes a non-technical description of how the medical device industry can benefit from hardware/software co-design principles.

1.1 Introduction

In the world of embedded systems, designers look minimize resource consumption and maximize computational performance. System designers can achieve this by combining and capitalizing on hardware and software capabilities into a single design effort known as hardware/software co-design. [3] This thesis describes a system that utilizes hardware/software co-design using a microcontroller (MCU) and Field Programmable Gate Arrays (FPGA) in an effort to minimize resource consumption (power and FPGA utilization) and maximize system performance. Software has a much shorter development time and higher runtime flexibility compared to hardware, because software runs on a general purpose processor. Hardware is the opposite in that it can be configured and tailored to specific tasks leading to a more efficient design. It is therefore highly desirable to have a system that combines hardware and software in order to gain the efficiency of hardware and yet keep the flexibility of software. [8][10]

This thesis applies hardware/software co-design principles to the life saving functions of a pacemaker. A pacemaker senses the heart's condition, analyzes sensor metrics and provides an appropriate response. The pacemaker must respond in a small window of time, referred to as the *response window*. This window starts when the pacemaker receives sensory data and ends when a delivered therapy would no longer be effective (approximately 26ms). If the analysis of the measurements requires excessive time to calculate and exceeds the response window, the pacemaker delivers its therapy late and the patient may enter a critical condition or die. [2] By applying hardware/software co-design principles to decrease calculation time, we can maximize a pacemaker's resources, complete calculations before the response window ends, and possibly allow for the application of more sophisticated therapies to the patient. One such sophisticated therapy is referred to as *adaptive pacing*, which means that a pacemaker is able to change its stimulating voltage according to the dynamic needs of the heart. This thesis analyzes several approaches to accelerate the Kendall Tau Rank Correlation Coefficient calculations which we use to process a pacemaker's sensor metrics to determine adaptive pacing voltage levels.

1.2 Motivations

The Kendall Tau Rank Correlation Coefficient is a statistical measure which can be used to determine if the heart needs to be pulsed. However, with current/available medical device technology, the algorithm that calculates the

Kendall Tau Rank Correlation Coefficient exceeds the response window and delays the pacemaker therapy. The Kendall Tau Rank Correlation Coefficient algorithm serves as an example of where hardware is useful in accelerating the calculation time to meet system timing requirements. This acceleration frees up more of the response window; this extra time in the response window can be used for future development and implementation of pacemaker technology.

1.3 Main Contributors

California Polytechnic San Luis Obispo has been working with St Jude Medical via the Medical Engineering Development and Integrated Technology Enhancement and Integrated Technology Enhancement Consortium (MEDITEC) program. The MEDITEC program is a multidisciplinary industry/academic consortium, which provides funding, project ideas and contacts at St. Jude Medical for students to enhance the biomedical research field. St. Jude Medical is global medical device company and a large pioneer in the cardiac pacemakers and provided the theory of applying the Kendall Tau Rank Correlation Coefficient to cardiac sensing.

1.4 Thesis Organization

This thesis is organized as follows: Chapter 2 gives a background of the problem from a biomedical and embedded systems perspective while Chapter 3 provides an outline of the Kendall Tau Algorithm with hardware/software co-design

methodologies applied. Chapter 4 describes several implementations using MCUs and FPGAs and the results of the implementations are discussed in Chapter 5. Chapter 6 describes the findings in this thesis by providing a recommended implementation, summarizing major conclusions, and addressing possible future works.

Chapter 2: Background

This chapter provides a more in-depth description of the problem addressed by this thesis. In particular, this chapter displays the goals of this thesis from an embedded systems and biomedical perspective.

2.1 Embedded Systems Perspective

In general, software in embedded systems execute on a MCU, which is composed of instruction/data memory and arithmetic/logic elements (ALU). Instruction memory stores software instructions that control the flow of data in the ALU. Data cycles between the ALU and memory until the software instructions have determined that it has completed execution. While MCUs have made considerable advancements, there will always exist more sophisticated algorithms that push the performance limits of the MCU.

A MCU operates by cycling data between memory and the ALU. This cycling provides flexibility allowing the general purpose ALU reconfigure its operation on each cycle, however it also limits the MCU by taking time to store data. Custom hardware would be configured to the application and would pass data directly to the next arithmetic or logic element instead of memory. Figure 1 shows the cycling between memory and ALU with the dotted box. This reuse of an ALU creates overhead since instructions must be decoded and executed and data passed/saved in registers. The MCU shown in Figure 1 contains one ALU, which limits it to one instruction executing at a time.

Figure 1: A diagram showing the Atmega328P architecture. The arrows denote the flow of data between RAM and ALU in the Atmega328P.

Other electronics communities including the high performance computing community have currently hit a wall due to power and energy density limits of MCUs and are searching for hardware/software co-design solutions. [4]

One of the current industry solutions to the MCU's power and performance limits is to create custom or specialized hardware to increase the overall system throughput. This system-level approach entails transferring some of the required computations to specialized hardware in an effort to meet scheduled deadlines. [19] [1] This specialized hardware can be tailored to a specific task and avoid decoding and storing instructions/data and passing data through unnecessary

hardware. Another advantage of this approach is that custom hardware can exploit an algorithm's inherent parallelism by executing multiple independent data sets on similar copies of hardware at a time.[9] In the biomedical industry, MCUs that have multiple cores and can handle multiple instruction, are the exception. [18]

Two types of hardware elements commonly utilized in industry are Field Programmable Gate Arrays (FPGAs) and Application Specific Integrated Circuits (ASICs). ASICs are custom-made integrated circuits (ICs) for which a mask and etch process is created. This process is very expensive and not cost effective for small runs. FPGAs are a class of ICs composed of many sets of look-up tables and gate arrays that are re-routed to implement different behavioral designs. [14] Both ASICs and FPGAs have the capabilities for implementing parallelism, however the ASIC performs more efficiently, because they do not route between reconfigurable look up tables (LUTs) and gate arrays as do FPGAs. The main drawbacks of ASICs are their expense and slow design and fabrication time, because a new mask and etch process is required for each version of the hardware. In contrast, a FPGA can be re-routed without an expensive mask and etch process, thus bypassing the main problem with ASICs. Accordingly, this thesis studies hardware implemented on an FPGA with the notion that all hardware models can be later synthesized as an ASIC. [7]

A simple example of exploiting parallelism is matrix multiplication where a 2×2 matrix is multiplied by another matrix of the same size. We decompose this

operation into two phases, (multiplication and addition), which leads to eight multiplications and four additions. We assume a simplified model where one cycle represents one operation (either multiplication or addition) meaning that multiplying two 2×2 matrices has a cost of 12 cycles. The multiplication operations are independent of each other and can thus execute in parallel. If we design this same multiplication on custom hardware using eight multipliers, we can execute this matrix operation in one cycle. This approach gives a final cost of five cycles (one cycle for parallel multiplication and four for addition). While a seven cycle savings may not seem to be a significant difference, the computational savings significantly increase as the matrices grow larger. For example multiplying two 5×5 matrices requires 225 cycles (125 multiplications and 100 additions); if we parallelize the multiplications, we have a cost of 101 cycles, thus saving 124 cycles. However, the speed savings of parallelism comes with a cost in hardware, requiring more adders, multipliers, registers to implement a given algorithm. [13][7][20]

The higher energy consumption of general purpose processors (both in embedded and high performance computing) is no longer acceptable. Therefore, computing communities are searching for more efficient computing methods, such as reconfigurable FPGAs, to provide application specific solutions.[4][3] This thesis presents an approach to leveraging the capabilities of a microprocessor in tandem with an FPGA.[20]

2.2 Biomedical Perspective

The pacemaker is a small medical device placed within the abdominal or chest cavity that delivers electrical pulses to control abnormal heart rhythms [6][5]. Patients receive pacemakers when their heart cannot maintain its own natural rhythmic pumping, which in time leads to premature death. This occurs when the muscles within the heart are desynchronized and work against each other which results in less or no blood being pumped. The pacemaker's primary objective is to deliver a pulse that exceeds a certain voltage referred to as the *capture threshold*, which causes a cascade effect between cells in the heart causing them to contract in unison and pump blood. This unison contraction of the heart is referred to as *capture* (CAP); the term *loss of capture* (LOC) refers to the condition where the voltage level does not reach this threshold to activate the cascade effect. [2]

Figure 2: (a) The first commercial implantable cardiac pacemaker (b) and the modern day pacemaker developed by St Jude Medical

The first implantable pacemaker was used on a patient in the 1960s. In this particular case the pacemaker paced at a preset voltage regardless of the heart's condition. Figure 2 shows a side by side comparison of a first generation

pacemaker next to a modern day pacemaker. Despite the fact that this design was relatively simplistic and crude by today's standard, it extended the patient's life for 18 months. [6] A major drawback of this early design was that the patient's capture threshold was only checked upon implant. However, a patient's capture threshold can fluctuate over time due to various physiological changes. When such change occurs, the pacemaker can potentially deliver insufficient voltage to the heart resulting in a loss of capture. [12] One other possible solution to this capture threshold issue was to set the pacemaker at maximum pacing voltage; however this approach prematurely depletes a pacemaker's battery and leads to avoidable implant surgeries. [2]

Figure 3: A photograph showing the internal battery and oscillator in a circa 1960 pacemaker.

Pacemakers today intermittently check for changes in the heart's capture threshold. Although this is advantageous over pacing at a constant voltage for the lifetime of the device, it does not sense the exact moment when the change in the heart occur. For example if a doctor presets the pacemaker to check the capture threshold on a daily basis, the beneficial or detrimental changes that may occur

during exercise routine will go unnoticed. Thus, this intermittent checking makes it difficult to identify physiological change or health issues in the heart.

The ideal pacemaker would check the heart's status and determine the minimum capture threshold on a beat-by-beat schedule. This method of determining the capture threshold is referred to as *adaptive pacing* [2]. By using an adaptive pacing method, we can find a solution that uses the lowest possible capture threshold necessary to pace the heart. By checking the capture threshold on a beat-by-beat basis may consume more power for calculation but saves significantly more by pacing the heart at the minimum capture voltage. Therefore the pacemaker maximizes battery life, minimizes the number of implant surgeries and show important trends in capture thresholds over time by logging corresponding data. This approach creates a need for an algorithm that can accurately interpret heart signals and provide the required voltage needed to meet the capture threshold.

The Kendall Tau algorithm has been proven effective in finding a correlation coefficient that can be translated into a heart's capture voltage. [2] However, currently pacemakers only perform intermittent checks for the capture threshold and not beat by beat checks, because the calculation time would exceed the response window.

2.3 Related Works

This section describes some examples of how the biomedical field uses hardware/software co-design. In particular we cite a case where hardware/software co-design provided greater processing capabilities.

2.3.1 Parallelization of an Electrocardiogram

An Electrocardiogram (ECG) shows the QRS complex (electrical activity of the heart) in real time. Doctors can subsequently use this information to interpret a patient's conditions. Current ECGs are able to show a QRS complex with simple filtering in real time. However, due to current computation limits of MCUs, complex results derived from the frequency breakdown and statistical averages cannot be displayed in real-time. If a doctor requires these results, the data is only available via post processing. Figure 4 shows an example of a typical QRS complex. The 'P' wave denotes the action potential or initial voltage that sets off the cascade of muscle contractions in the heart. The 'Q' pulse represents the depolarization that happens just before the actual contraction of the muscle which is denoted by 'R'. The 'S' represents the over travel as the heart begins re-polarization. The 'T' represents the portion of the heart as it re-polarizes and cannot be stimulated. The features of the QRS complex shows the activity of the heart and serve as indicators for heart health and disease.

Figure 4: The features of a QRS complex that are displayed on an ECG.

One approach take from [15], uses a parallel architecture to implement the Pan Thompkins algorithm, ACF-based algorithm and Fast Fourier Transform to real-time. These algorithms cannot be processed in real-time; however the parallel architecture described by this work allows independent calculations to be executed simultaneously, which allows for higher sampling rates while meeting real-time constraints.

In particular, the approach presented in [15] uses a parallel architecture of the ST200 DSP MCU which is composed of four cores. Each MCU core has its own private memory that is connected on a shared bus. As expected, execution time decreases and the power consumption increases. The parallel architecture achieves a 775% (72 million cycles to 8M cycles) savings in cycles and 156% power savings compared their benchmark of an ARM7TDMI which is their single

core architecture. At 12 cores the power consumptions is 400% higher than the four core approach. This solution achieves higher performance than the single core implementation, however still suffers from cycling data between ALU and memory.

In contrast, this thesis provides a higher performance solution by using a FPGA instead of multiple MCUs. While adding parallel MCUs give parallelization capabilities it significantly increases power consumption by adding general purpose hardware which is typically not 100% utilized by the particular application. This increase in power consumption is not suitable for pacemakers where power usage is highly constrained. A FPGA co-processing solution maybe a more suitable for pacemakers, because it allows the designer to directly define the required hardware and therefore minimizing power consumption.

We acknowledge that a comparison between the Pan Thompkins algorithm, ACF-based algorithm and Fast Fourier Transform are significantly different from the Kendall Tau algorithm which is accelerated in this thesis. However the relations of performance, power consumption and parallel processing is still applicable. If we compare multiple approaches to parallelism it will give a general idea of the trade-offs between them.

2.3.2 Other Significant Work

At the time this thesis was written there are not any public papers regarding the application of HW/SW co-design to a pacemaker hardware. Some papers in HW/SW co-design that were noteworthy is [1], [4] and [19]. Noteworthy papers regarding the concept of adaptive pacing are [12].

Chapter 3: Kendall Tau Algorithm

This chapter gives an overview of the Kendall Tau Algorithm. In particular, we describe the basic algorithm and show an approach to decomposing this algorithm in an effort to prepare for hardware co-processing.

3.1 The Algorithm

St Jude Medical currently uses the Kendall Tau algorithm to compare sensor heart data to a patient's heart template to obtain a correlation coefficient that gauges the voltage required to reach the capture threshold of the heart. Pacemakers generate this template by logging past averages of the patient's capture and loss of capture thresholds. (Defined in section 2.2)

The Kendall Tau algorithm is a type of statistical measure that falls under a non parametric classification, meaning that the algorithm is independent of probability distribution assumptions. [11] The non parametric classification is beneficial for our purpose because the Kendall Tau Algorithm is not affected by outliers, unequal variances, non-normality and non-linearity as much as parametric algorithms. Parametric equations rely on established distributions; if the statistical set analyzed does not follow the distribution, the results are significantly affected. The parametric equations' reliance on distribution makes it an undesirable analysis tool for modeling cardiac signals that may have intermittent noise or offset over time. [2]

The Kendall Tau Algorithm calculates a coefficient τ that signifies the association of two sets of points. Eq.1 shows the Kendall Tau Algorithm.

$$\tau = \frac{(\# \text{ concordant pairs}) - (\# \text{ discordant pairs})}{(0.5n(n-1))} \quad (\text{Eq.1})$$

In Eq.1 the variable ‘n’ represents the size of the sets to be compared. Eq.2 determines if a pair is concordant by subtracting two points of index ‘i’ and ‘j’ in set ‘x’ and multiplied by the corresponding difference of two points of index ‘i’ and ‘j’ in set ‘y’. If the result is positive, the pair is concordant; if negative, the pair is discordant. This accumulation of pairs gives a metric of how similar the measured heart signal is to a pre-saved “capture” and “loss of capture” templates. Eq.2 shows the equation to test for concordant and discordant pairs.

$$z = (x[i] - x[j]) * (y[i] - y[j]) \quad \begin{array}{l} \text{If } z \text{ is positive they are concordant (Eq.2)} \\ \text{If } z \text{ is negative they are discordant} \\ \text{If } z \text{ is zero they are neither} \end{array}$$

Eq.3 represents the calculation of growth as a set size increases for the Kendall Tau Algorithm, where ‘n’ stands for the number of entries in the set and ‘d’ stands for the number of instructions required per iteration. The variable ‘s’ represents the static number of instructions required regardless of the set size. Therefore the computational complexity required to solve the Kendall Tau coefficient grows quadratically as the set size increases.

$$\left(\frac{(n-1)^2 + (n-1)}{2}\right)d + s \quad (Eq.3)$$

3.2 Decomposition of the Kendall Tau Algorithm

In an effort to organize the Kendall Tau algorithm for parallel implementation, we decompose it into groups of similar operations. Eq.4-7 shows the decomposition.

$$\tau p1 = \text{sgn}(x(i) - x(j)) \quad \text{range of } i = 2 - 20 \text{ range of } j = 1 - i \quad (Eq.4)$$

$$\tau p2 = \text{sgn}(y(i) - y(j)) \quad \text{range of } i = 2 - 20 \text{ range of } j = 1 - i \quad (Eq.5)$$

$$\tau p = \Sigma (\tau p1 * \tau p2) \quad (Eq.6)$$

$$\tau = \frac{(\tau p)}{(0.5 n(n-1))} \quad (Eq.7)$$

Separating the equation according to like operations highlights the major divisions of work required by the algorithm. For set sizes of 10 and 20 points, Eq.4, Eq.5, and Eq.6 are iterated 45 and 190 times and Eq.7 only iterates once at the end for both set sizes. This shows that the computational cost of Eq.4, Eq.5, and Eq.6 is dynamic and depends on set size. These three equations are represented in the growth equation (Eq.2) with the variable ‘d’ for dynamic cost and represent the cost per comparison. The number of iterations of Eq.7 remains the same regardless of set size and is represented by the variable ‘s’ for static cost. Figure 5 shows a graph of the growth equation when $d = 1$ and $s = 0$ showing the dynamic cost grows exponentially as the set size increases.

Figure 5: A graph showing the Number of calculations vs Set size to complete the Kendall Tau Algorithm

Eq.4, Eq.5 and Eq.6 are the most computationally expensive for large sets, making these equations ideal candidates for acceleration or off-loading to a hardware device for co-processing.[20] These three equations represent the dynamic cost, therefore if we assume each equation takes one instruction cycle to execute $d = 3$. In the Chapter 4 we modify the growth equation (Eq.2) to match the parallelization of each implementation offers. Eq.7 executes only once, therefore $s = 1$. Eq.8 shows the updated growth equation.

$$\left(\frac{(n-1)^2 + (n-1)}{2}\right)3 + 1 \quad (Eq.8)$$

This thesis uses a set of 20 data points per set, to generate the Kendall Tau coefficient which is the typical set size used to gain enough confidence for a pacemaker's recommended capture threshold. [2] Random data is used to fill in sets, due to actual patient data being confidential and thus unavailable. Using random sets of data does not affect the calculation time of the Kendall Tau coefficient because the algorithm requires a comparison of each point to another in their respective sets and does not branch based on data. Therefore, all 190 comparisons between the measured heart signal and saved templates execute regardless of the data content, and does not affect the calculation time of the algorithm.

Chapter 4: Implementation and Theory

Chapter 4 includes a description of the hardware configurations applied in this thesis as well as a justification for their use. In addition, this chapter includes the measurement methodology used for calculation time and power consumption.

4.1 Hardware

For the software component, we utilize an Atmega 328p MCU on an Arduino Uno development board. The Arduino Uno is an 8-bit MCU operating at 16 MHz. We chose this MCU because pacemakers also utilize 8-bit MCUs for low power consumption. We developed the MCU code using the C programming language and the AVR Studio 4 IDE. [17] Figure 6 shows MCU used.

Figure 6: The atmega328p MCU on the Arduino Uno development board.

For the hardware component of this experiment, we utilize the DS312 Spartan 3e mounted on a FPGA Digilent Nexys2 development board. Nexys2 board operates at 50 MHz; this faster processing speed is taken into account when we analyze the

results. The FPGA contains five types of hardware blocks: configurable Logic Blocks, input/output blocks, block ram, multiplier blocks and digital clock manager. These blocks form four quadrants and are connected using four 8-bit bus lines. [16] We model all co-processing hardware implemented on the FPGA using VHDL in the Xilinx ISE Design Suite 13.3. Figure 7 shows MCU used.

Figure 7: The Spartan 3e FPGA mounted on the Nexys2 development board.

Actual pacemaker hardware was unavailable at the time of this thesis due to licensing and logistical issues. Further testing on pacemaker hardware would be required.

4.2 Implementation of Kendall Tau on a MCU

Figure 8 shows the black box diagram of the Kendall Tau implementation on the MCU. The MCU uses a UART connection to deliver the Kendall Tau coefficient to the computer via the “UART TX” and “UART RX” pins. The “pulse” pin raises to signify the start of the calculation; when the “pulse2” pin raises it

signifies the end of the calculation. An oscilloscope measures the calculation time by finding the difference in time between the two rising edges of “pulse” and “pulse2.” The “pulse” and “pulse2” pin change code encapsulates only the code pertaining to the Kendall Tau calculation.

Figure 8: (a) A black box diagram showing the inputs and outputs of the MCU implementation, and (b) the MCU implementation showing the “pulse” and “pulse2” lines connected to a scope for read out and UART connection through USB.

This implementation calculates the Kendall Tau coefficient using only the MCU. The calculation is relatively simple to implement using C programming and

entails only a few lines of code. This simplicity is due to having a compiler interpret the C code then translating it into the MCU's native assembly language. The assembly language can then tell the ALU how to cycle data between arithmetic, logic and memory elements to produce the Kendall Tau Coefficient. A FPGA solution can optimize this cycling, but the designer must manually specify the arithmetic, logic and storage elements.

4.2.1 MCU Implementation Design Theory

This implementation on the Atmega328p has no parallel optimization which causes $d = 3$ and $s = 1$. That means Eq.8 remains the same. Figure 9 shows the growth of Eq.8 as a function of set size.

$$\left(\frac{(n-1)^2 + (n-1)}{2}\right)3 + 1 \quad (Eq.8)$$

Figure 9: A plot of the Number of Executed Instructions vs Set Size showing the growth function for the MCU implementation.

4.3 Implementation of Kendall Tau on a FPGA

Figure 10(a) shows the black box diagram of the Kendall Tau implementation on the FPGA. In Figure 10(b) MCU is on the left and the FPGA on the right. For this implementation, the MCU sends randomly generated data points to the FPGA and receives the finished Kendall Tau Coefficient; the MCU passes the Kendall Tau Coefficient to a terminal program via the MCU's UART peripheral to a desktop computer for verification. This implementation calculates the Kendall Tau coefficient using only the FPGA.

Figure 10: (a) A black box diagram showing the inputs and outputs of the FPGA implementation, and (b) a photograph of the FPGA implementation showing the parallel data, and control lines.

Figure 11 shows the architecture of the hardware implementation on the FPGA. This implementation is comprised of four main components: KT_top, Interface, Mem and the KT_Module. KT_top act as a wrapper module that handles the port mapping between the interface module, memory module and KT module. When the calculation of the Kendall Tau coefficient completes, the hardware outputs it across the data bus to the MCU.

Figure 11: A black box diagram for the internal FPGA modules (FPGA implementation).

4.3.1 Interface Module

The interface module provides a means for organizing communication between MCU and FPGA. The interface block uses a four-stage communication handshake consisting of *request*, *transfer*, *acknowledge* and *reset*. For this configuration, the MCU acts as the master while the FPGA acts as the slave. Figure 12 shows a simulation timing diagram of the four phases of the handshake. The master initiates the *request* phase of the handshake by placing valid data on the 8-bit bus and then raises the REQ line. The FPGA acknowledges by recording the eight one-bit data lines and raises the ACK line to complete the *transfer* stage. During the *acknowledge* phase, the MCU senses the raising of the ACK line and it drops the REQ to signal the FPGA to end the *reset* phase. Now the system is ready for the next transmission. When the FPGA senses the dropping of the REQ, it responds by dropping its ACK line to signify it can receive next send, thus completing the reset phase.

Figure 12: A timing diagram showing the 4-stage handshake for REQ and ACK lines

4.3.2 Memory Module

The memory module populates two sets of 20 points to represent the sensor and the comparison points. The memory model receives these points from the interface module. When fully populated the memory module sends a ready signal and then send the points to the Kendall Tau module over a 360 bit bus (40 points x 8 bits). The memory module does not save any more points from the interface block until it receives a reset signal.

4.3.3 Single Kendall Tau Module

This module does the bulk of the Kendall Tau algorithm. It executes two concordant test comparisons per cycle for τ_{p1} and τ_{p2} , and then tabulates τ_p . It uses two counter/comparator sets (i and j shown in Eq.2), the counters keep track of what array index to pass to the comparator to find concordant pairs. After the

comparison is finished, the Kendall Tau module reports the summation of concordant pairs and discordant pairs to the KT_top.

$$z = (x[i] - x[j]) * (y[i] - y[j]) \quad (Eq.2)$$

Figure 13: The data points used per cycle of the Kendall Tau comparison for a single hardware set.

Figure 13 shows how counter ‘i’ iterates one index and shows what array elements are used per round. On the first iteration only to index ‘1’ of both cap and loc sets is used. However on the fifth iteration we see that indices five down to one is used and takes five cycles to complete. This shows the quadratic growth of the Kendall Tau algorithm.

4.3.4 Double Kendall Tau Module

This module executes similarly to the single Kendall Tau module; however it adds another layer of parallelism by simultaneously comparing four concordant test comparisons instead of two. This implementation contains four counters to keep track of which comparisons are complete. The module achieves this by allowing ‘i’ and ‘j’ counters to handle even index comparisons, and ‘k’ and ‘l’ to handle odd index comparisons. This roughly provides a theoretical 50% reduction in calculation time by dividing the work between two sets of hardware. We can further parallelize this module by adding another counter/comparison set for a triple Kendall Tau module or even a twenty Kendall Tau module. However, the additional hardware, increases resource utilization and power consumption. This module provides results to show the power consumption and speed trends of adding parallel hardware.

Figure 14 shows how counters ‘i’ and ‘k’ iterate over two indices and what array elements are used per round. By comparing the single implementation (Figure 13) and the double implementation (Figure 14), we can see the benefits of parallelization. By the fourth iteration the single implementation only reaches array index four while the double implementation reaches array index eight. This gives the double implementation a theoretical 50% decreased processing time.

Figure 14: The data points used per cycle of the Kendall Tau comparison for a double hardware set.

4.3.5 FPGA Implementation Design Theory

This implementation utilizes the FPGA parallelization capabilities to complete the Kendall Tau Algorithm, which in turn modifies the growth equation. The single FPGA implementation executes the eq.4, eq.5 and eq.6 in one cycle causing $d = 1$ and $s = 1$ resulting in the growth equation shown by Eq.9. The double implementation executes two sets (of eq.4, eq.5 and eq.6) in one cycle therefore causing $d = .5$ and $s = 1$ giving Eq.10. Figure 15 plots the resulting growth function and shows the theoretical impact of adding parallelization. The MCU implementation for a set size of 20 requires 570 instructions while the FPGA

single that exploits parallel processing requires 190 cycles giving a theoretical 200% speed up if they have the same clock speed.

$$\left(\frac{(n-1)^2 + (n-1)}{2}\right)1 + 1 \quad (\text{Eq.9})$$

$$\left(\frac{(n-1)^2 + (n-1)}{2}\right)0.5 + 1 \quad (\text{Eq.10})$$

Figure 15: A graph showing the Number of Executed Instructions vs Set Size in order to compare MCU, FPGA Single, FPGA Double implementations

4.4 Implementation of Kendall Tau on a MCU and FPGA

The black box diagram in Figure 16 shows the Kendall Tau implementation on the FPGA and MCU. An oscilloscope solves the time difference between the two rising edges of “pulse” and “pulse2” to obtain the processing time. This

implementation distributes the Kendall Tau calculation between the MCU and FPGA and includes the transfer time between the two devices.

(a)

(b)

Figure 16: a) A black box diagram showing the inputs and outputs of the MCU/FPGA implementation, and b) the MCU/FPGA implementation showing the parallel data, control lines and “pulse” and “pulse2” lines.

The MCU transfers two points and the FPGA compares and tabulates the points. After the MCU transfers the 190 points, the FPGA returns the Kendall Tau Coefficient value of comparisons to the MCU. This implementation only uses a single set implementation of the Kendall Tau module. Figure 17 shows the architecture for the hardware implemented on the FPGA.

Figure 17: A black box diagram for the internal FPGA modules (MCU/FPGA implementation).

4.4.1 FPGA Implementation Design Theory

This implementation utilizes the FPGA parallelization to compare one set, tabulate the τ_p value and after 190 iterations will return the τ_p to the MCU. The FPGA executes the Eq.4, Eq.5 and Eq.6 in one cycle causing $d = 1$ and $s = 1$ giving Eq.11. Figure 18 plots the resulting growth function. The resulting growth equation is identical to the FPGA single implementation, because the

MCU/FPGA and FPGA Single implementation have an equivalent amount of parallelization.

$$\left(\frac{(n-1)^2 + (n-1)}{2}\right)1 + 1 \quad (\text{Eq.11})$$

Figure 18: A graph showing the Number of Executed Instructions vs Set Size comparing MCU, FPGA Single, FPGA Double, MCU/FPGA implementations. The MCU/FPGA implementation takes the amount of executed instructions as the FPGA single implementation.

4.5 Measurement Methodology

This section outlines the methodology used to measure the calculations time and power consumption. This section also includes our reasoning for implementing this particular measurement methodology.

4.5.1 Calculation Time Measurement Method

We obtain the time required to calculate the Kendall Tau Coefficient by measuring the time difference between two pins (pulse and pulse2 pins shown in Figure 19 and Figure 20) with an oscilloscope. This measurement method isolates the time to calculate the Kendall Tau Coefficient because the systems are kept free from interrupt service routines (ISRs) and other hardware that add additional overhead time to the results.

Figure 19: A photograph showing the pulse and pulse2 pins from the FPGA to oscilloscope probes

Figure 20: A photograph showing the pulse and pulse2 pins from the MCU to oscilloscope probes

We initially attempted to obtain the time measurements using the MCU's timer peripheral and external interrupts. However this approach was not an accurate method of measurement. This method was suitable for the MCU only implementation, but lost accuracy when measuring the FPGA's calculation time. The FPGA clock at 50MHz is significantly faster than the MCU's 16 MHz clock speed and results in the execution of multiple FPGA clock cycles before the

MCU's timer would interrupt and record the time. Therefore, we can obtain a more accurate result if we measure directly from the FPGA when possible.

We acknowledge that the MCU time may contain a slight inaccuracy between measured time and actual calculation time due to the time required to update a pin. The Arduino Uno requires a minimum of 250 ns to change a pin state, however only the full MCU implementation suffers from this delay. Timing measurements in the other implementations have the timing measurements taken from the FPGA which has no delay due to its parallel processing capabilities. Since the results the MCU implementation is in the millisecond range, a delay of 250ns is not significant.

4.5.2 Power Measurement Method

The method to measure the power consists of splicing a USB wire and measuring the current draw of the power line. We connect the remainder of the lines to their respective wire via a breadboard. Figure 21 shows this setup. A USB port supplies a constant voltage of 5V; the measured current allows us to calculate the instantaneous power consumed. To obtain the power consumed over time we multiply the instantaneous power with the calculation time for each implementation.

Figure 21: A photograph showing the USB wires to MCU and FPGA spliced with ammeter in series.

Chapter 5: Results

This chapter includes the measured calculation time, power results and discussion of the results. The discussion gives special focus to how the experimental results of parallelizing the Kendall Tau Algorithm compares to the theoretical results from Chapter 4.

5.1 Time to Calculate Kendall Tau Coefficient

Table 1 shows the run time speeds of the four different implementations described in Chapter 4. The MCU implementation executes the slowest primarily due to the slower clock frequency in addition to the fact that it has to iterate each calculation through the ALU. However, even if we increase the MCU clock frequency to 50MHz (equivalent to the clock frequency of the FPGA) and the execution time was scaled equivalent to the MCU clock increase, it would give an execution time of 1664us. The modified execution time still cannot match the FPGA execution time of 6us. This overall result shows the power of parallelization of operations.

	Operating Time (us)
MCU implementation	5200.0
FPGA implementation	6.0
FPGA implementation double	4.8
MCU/FPGA implementation	2280.0

Table 1: This table shows the processing times for the various implementations.

The single and double FPGA implementations show a difference in processing time of 1.2 μ s or 20%. This time difference is due to the additional counter/comparator hardware in the double implementation. However, the growth equation in section 4.3.5 expected a theoretical 50% decrease in processing time from doubling the number of comparators. This difference between experimental and theoretical result is due to the extra clock cycles needed to combine the odd and even accumulators in the double implementation. This accumulation time emphasizes an important aspect of parallelization where the aggregation of parallel data creates overhead. Therefore, it is key for a designer to find where the overhead time does not exceed the time saved by using parallelization.

The FPGA implementation calculation time may contain a bias because, it excludes the communication time to and from MCU, while the MCU/FPGA implementation includes the transfer time. In most pacemakers, a MCU is the central control unit and the data would have to be passed to it. In an actual pacemaker implementation, this additional delay cannot be avoided. The implemented interface module requires 4 μ s to transfer 8-bits between MCU and FPGA. However, this thesis solely focuses on processing time and does not include the transfer time in the FPGA implementation.

In the MCU/FPGA implementation, we attempted to leverage the decision making capabilities of the MCU and the parallelism of the FPGA. However, the

MCU/FPGA implementation is considerably slower than the full FPGA implementation. It is interesting to note that, the MCU/FPGA implementation could have been faster because there was an added delay from transferring data back and forth between the MCU and FPGA. Since the data transfers happen in between calculation, it difficult to accurately isolate only calculation time, therefore the MCU/FPGA implementation results contain data transfer time as well. The measured time to transfer an 8-bit number with the four-stage protocol described in section 4.3.1 is approximately 4us. Since the MCU/FPGA implementation has to transfer data between the MCU and FPGA and the transfer time alone is more than the FPGA implementation's calculation time, the MCU/FPGA implementation runs significantly slower. One possibility that would make this comparison fairer, would be to re-implement the MCU/FPGA implementation on a true shared memory platform. This would eliminate the interface time and would more accurately show the processing time of the MCU/FPGA implementation. A shared memory implementation would mitigate the transfer time by allowing the MCU and FPGA to write and read from a common memory source.

In actual applications, the MCU/FPGA implementation would be more flexible and allow firmware updates to change its behavior. This updating would not be possible on the FPGA implementation. Pacemakers primarily utilize an ASICs and not an FPGAs; ASICs effectively lock the hardware architecture and do not allow updates. Therefore, if a doctor would like to run the Kendall Tau algorithm

with a different sampling size, or other parameters, the doctor would only be able to do so, on the MCU and MCU/FPGA implementation because the implemented ASIC version is not configurable.

5.2 Power Consumption

Table 2 shows the power consumption of the four implementations described Chapter 4. Table 2 shows that the lowest power consumption is from the MCU and the highest is from the MCU/FPGA implementation. In the two FPGA implementations, there is a small increase between the single and double hardware implementation. We attribute this increase to the area used to implement the additional hardware routing on the FPGA, thus causing a slight increase in power consumption. We opted to make no attempt at hand routing the FPGA hardware for optimal placement, because a real world pacemaker implementation would be routed on an ASIC which is not comparable to FPGA routing. Section 5.3 further discusses the FPGA utilization between the single and double set of hardware.

	Operating Current (mA)	Operating Voltage (V)	Instantaneous Power (mW)
MCU implementation	52.1	5.0	260.5
FPGA implementation	106.7	5.0	533.5
FPGA implementation double	108.5	5.0	542.5
MCU/FPGA implementation	158.8	5.0	794.0

Table 2: Shows the measured Current/Voltage/Instantaneous power for each implementation

An important caveat to these results is that we obtained the measurements from development boards and not directly from the MCU or the FPGA. This means that there are peripheral components that affect the power consumption metrics. Each implementation powered down all accessible unused components such as displays, LEDs, etc. However, there are still inaccessible components on the development boards that still consume power in addition to the MCU and FPGA. The Arduino Uno development board's low power mode uses approximately 10mA less and the Nexys 2 board does not contain a low power mode. If the MCU is broken out of their development board the MCU would only consume 0.2mA while active and .75uA in power-down mode. The FPGA did not have a listed active and low power rating. [16][17]

The FPGA implementation power consumption does not include the MCU, which generates the points and delivers the result to a computer terminal program. The MCU in the FPGA implementation does not calculate the Kendal Tau coefficient and therefore we exclude its power consumption metrics.

Initially the high power consumption of an FPGA seems unacceptable for use in the medical device industry. However we are interested the direct power necessary to produce the Kendall Tau Coefficient.

	Instantaneous Power (mW)	Operating Time (us)	Inst. Power x Opr. Time (uJ)
MCU implementation	260.5	5200.0	1354.6
FPGA implementation	533.5	6.0	3.2
FPGA implementation double	542.5	4.8	2.6
MCU/FPGA implementation	794.0	2280.0	1810.3

Table 3: The power cost to calculate the Kendall Tau Algorithm.

We can calculate the direct power combining the runtime with the power consumption data in Table 3. The FPGA may consume the most instantaneous power of all the implementations, but over time the MCU consumes more power, because the processing time for the two FPGA implementations are relatively short. Implementing an idle/sleep state where the FPGA would consume little to no power while not calculating the Kendall Tau coefficient is critical to exploit the power savings of a fast calculation.

Additionally, the results show that the MCU/FPGA implementation consumes more than the MCU despite having a faster execute time. This means that the cost of powering both FPGA and MCU running outweighs the reduction in processing time.

5.3 Difference in FPGA area consumption Single vs. Double

Table 4 shows the resource summary of the two FPGA implementations. The effects of doubling the hardware significantly increases the amount of hardware required to implement the algorithm. If we compare the number of look up-tables (LUTs) used for a single implementation and double implementation, the utilization is 692 LUTs for the single and 1988 LUTs for the double. The increase in parallelism has significantly increased the amount of hardware required to process the algorithm which in turn increases the power consumption.

	Framework	Single Set	Double Set	Double - Single
Number of Slice Flip Flops	12	371	384	13
Number of 4 input LUTs	12	692	1,988	1296
Number of occupied Slices	12	554	1,300	746
Number of Slices (related logic)	0	554	1,300	746
Number of Slices (unrelated logic)	12	0	0	0
Number of bonded IOBs	35	35	35	0
Number of BUFGMUXs	1	1	1	0
Number of MULT18X18SIOs	2	1	2	1
Average Fan-out of Non-Clock Nets	0.00	3.38	4.24	0.86

Table 4: Resource usage for the Single VS Double FPGA implementations.

5.4 Difference in FPGA area consumption FPGA only vs. MCU/FPGA

Table 5 shows the resource summary of the FPGA and MCU/FPGA implementations. The FPGA single and the MCU/FPGA implementations show a difference of 1509% in 4-input LUTs and a difference of 1483% in occupied slices. The resource utilization of a MCU/FPGA implementation is significantly less, because the FPGA routing does not have to route 40 points for individual

comparisons. The MCU/FPGA compares four incoming points, tabulates the results for 190 iterations, and then outputs the result. The reduced utilization on the FPGA usually would lead to reduced power consumption, however we do not see a considerable drop in the FPGA power utilization. This is probably due to the fact that a development board was used to implement this approach. This power and FPGA utilization issue requires further testing on a hardware specific to St Jude Medical in order to acquire real world results.

	Framework	FPGA single	MCU/FPGA
Number of Slice Flip Flops	12	371	41
Number of 4 input LUTs	12	692	43
Number of occupied Slices	12	554	35
Number of Slices (related logic)	0	554	35
Number of Slices (unrelated logic)	12	0	0
Number of bonded IOBs	35	35	35
Number of BUFGMUXs	1	1	1
Number of MULT18X18SIOs	2.41	1	1
Average Fan-out of Non-Clock Nets	-	3.38	2.45

Table 5: Resource usage of the FPGA single and MCU/FPGA .

Chapter 6: Conclusion

This chapter recommends an implementation based on the results gathered by this thesis's implementation. This chapter also outlines possible future work that would provide more insight to hardware/software co-design principles.

6.1 Overall conclusion

Our results show that the MCU/FPGA and the FPGA implementation have a 128% and 86566% increases in speed, respectively, when compared to a MCU implementation. This improvement originates from the customization of hardware to operate in parallel on the Kendal Tau Algorithm. This increase in speed has also increased the power consumption due to the parallelization of the algorithm implementation. By adding an FPGA the MCU/FPGA and FPGA implementation had a maximum instantaneous power increase of 204% and 105%, respectively, when compared to the MCU implementation. Based on the hardware used in this thesis, the FPGA implementation appears to be the best solution. However, in order to maximize the potential of the FPGA implementation, the designer must implement a low power mode, which would mitigate the power consumption of the FPGA when idle.

6.2 Future Work

This section describes sever areas of future work that would provide additional results while complementing the work done in this thesis.

6.2.1 Shared Memory Implementation

The implementations in this thesis did not use a shared memory architecture. By not doing so, this resulted in additional communication time to the MCU/FPGA implementation. If this experiment were run on a shared memory implementation the communication time would be mitigated, because the MCU could write values to preset locations in memory where the FPGA at the same time would also be able to read and begin the calculation. This approach would remove the four-stage communication protocol, which requires approximately 4 μ s per point to run, and possibly improve the different FPGA size/power consumption and processing time.

6.2.2 St. Jude Medical Hardware Specific Implementation

The biomedical device industry configures their hardware for reliability and power longevity. This experiment utilized typical off-the-shelf hardware, which typically focuses on ease of use and performance. That means there are additional components (LED, 7seg displays, power regulation, and external memory) that consume extra power and possibly improving performance. Compiling the firmware and running it on an actual medical device's hardware will give different results. This experiment, however provides a good indication of which implementation will provide the best results in future development.

6.2.3 ASICs Implementation

The Kendal Tau algorithm would benefit from an ASIC implementation. In general ASICs provide better physical size, lower power consumption and higher

performance characteristics over an FPGA. The smaller physical size would allow for a less obtrusive form factor which would increase comfort when implanted in a patient. The lower power consumption would allow for longer battery life and subsequently reduce the amount of surgeries. The increase performance would further increase the response window for other processes to execute.

6.2.4 Further FPGA Parallelization

In this thesis the FPGA implementation used a single and double set of hardware to parallelize the Kendall Tau Algorithm. However, many more levels of parallelization can be implemented and require testing to give a better estimation of calculation time and power consumption. Further testing would find the optimal parallelization level, where the calculation time is minimized while still maintaining an acceptable rate of power consumption. We can use our results to extrapolate the notion that, when the level of parallelization increases the savings gained will eventually be outweighed by the overhead to compile the answer from each parallel comparator.

References

- [1] M. Baleani, F. Gennari, Y. Jiang, Y. Patel, R. Brayton, A Sangiovanni-Vincentelli, “HW\SW Partitioning and Code Generation of Embedded Control Application on a Reconfigurable Architecture Platform” from Proceedings of the tenth international symposium on Hardware/software codesign Pages 151-156, 2002
- [2] J. Snell, N. Korsum, S. Fatemi, R. Rooke, “Whitepaper: New Atrial Threshold Search Algorithm Based on Waveform Correlation” SJM internal document, 2007
- [3] P. Schaumont “A Practical Introduction to Hardware/Software Codesign” Springer; 1st Edition. edition (September 14, 2010)
- [4] X. Hu, R. Murphy, S. Dosanjh, K. Olukotun, S. Poole “Hardware/Software Co-design for High Performance Computing: Challenges and Opportunities” from Hardware/Software Codesign and System Synthesis (CODES+ISSS), 2010 IEEE/ACM/IFIP International Conference, 2010
- [5] <http://www.nhlbi.nih.gov/health/health-topics/topics/pace/> web, accessed May, 9 2013.

- [6] H. Beck, W. Boden, S. Patibandla, D. Kireyev, V. Gutpa, F. Campagna, M. Cain, J. Marine, 50th Anniversary of the first successful permanent pacemaker implantation in the United States: historical review and future directions.
<http://www.ncbi.nlm.nih.gov/pubmed/21391322> web, accessed May, 9 2013.
- [7] M. Abdelfattah and V. Betz, “Design Tradeoffs for Hard and Soft FPGA-based Networks-on-Chip” from Field-Programmable Technology (FPT), 2012 International Conference, 2012
- [8] V. Shrimal, M. Sharma, S. Singh, R. Gupta, “Specific Criteria for System Design and Role and Importance of Hardware Software Co-design Process”, World Applied Programming, Vol (1), No (3), 2011
- [9] Y. Manai, J. Haggège, M. Benrejeb, “New Approach for Hardware/Software Embedded System Conception Based on the Use of Design Patterns”,
(<http://www.SciRP.org/journal/jsea>), 2010
- [10] P. Brunmayr, J. Haase, C. Grimm, “A Hardware/Software Codesign Template Library for Design Space Exploration” Proceedings of the 2011 Electronic System Level Synthesis Conference, 2011
- [11] R. Newson “Parameters behind “nonparametric” statistics: Kendall’s tau, Somers’ D and median differences” The Stata Journal, 2002

- [12] F. Abi-Samra, N. Singh, B. Rosin, J. Dwyer, C. “Miller Effect of rate-adaptive pacing on performance and physiological parameters during activities of daily living in the elderly: results from the CLEAR (Cylos Responds with Physiologic Rate Changes during Daily Activities) study
<http://europace.oxfordjournals.org/content/early/2013/02/16/europace.eus425.abstract> web, accessed May, 9 2010.
- [13] S McBader, P. Lee, “An FPGA Implementation of a Flexible, Parallel Image Processing Architecture Suitable for Embedded Vision Systems“ Parallel and Distributed Processing Symposium, 2003. Proceedings. International, 2003
- [14] M. Santambrogio, D. Sciuto, “Design methodology for partial dynamic reconfiguration: anew degree of freedom in the HW/SW codesign“ , Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium, 2008
- [15] I. Khatib, D. Bertozzi, A.Jantsch, L. Benini, “Performance Analysis and Design Space Exploration for High-End Biomedical Applications:Challenges and Solutions” 5th International Conference on Hardware/Software Codesign and System Synthesis, 2007
- [16] www.xilinx.com/support/documentation/data_sheets/ds312.pdf web, accessed May, 9 2013.

- [17] <http://www.atmel.com/Images/doc8161.pdf>, web, accessed May, 9 2013.
- [18] <http://www.aspdac.com/aspdac2011/archive/pdf/6D-2.pdf> web, accessed June, 10 2013
- [19] A. Vincentelli, “Quo Vadis, SLD? Reasoning About the Trends and Challenges of System Level Design” Proceedings of the IEEE (Volume:95 , [Issue: 3](#)), 2007
- [20] L. Bleris, P. Vouzis, M. Arnold and M. Kothar, “A Co-Processor FPGA Platform for the Implementation of Real-Time Model Predictive Control” Proceedings of the 2006 American Control Conference Minneapolis, Minnesota, 2006