**University of Calgary**

**PRISM Repository** **https://prism.ucalgary.ca**

The Vault Open Theses and Dissertations

2014-01-20

# Reliable Embedded Systems Development

Deng, Dongcheng

Deng, D. (2014). Reliable Embedded Systems Development (Master's thesis, University of Calgary, Calgary, Canada). Retrieved from https://prism.ucalgary.ca. doi:10.11575/PRISM/27616
http://hdl.handle.net/11023/1274
*Downloaded from PRISM Repository, University of Calgary*

UNIVERSITY OF CALGARY

# Reliable Embedded Systems Development

by

Dongcheng Deng

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES

IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE

DEGREE OF MASTER OF SCIENCE

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

CALGARY, ALBERTA

JANUARY, 2014

© Dongcheng Deng 2014

**Abstract**

With software defects reducing profits in many fields, it is worthwhile to consider moving key defect reducing strategies from one development area into another. A considerable amount of effort has been contributed to adapt and adopt the success of Agile methods into the embedded domain as defect-reduction methods. However, limited efficient and effective tools constrain the applicability of the Agile philosophy. Moreover, little studies have been reported to summarize the current state of the Agile-inspired embedded software development (AIESD) domain. The implications of an overview of AIESD generated from a systematic mapping study are discussed. An Agile test support (ATS) co-processor was proposed to provide low-overhead test insertion capability to embedded processors. The performance of the ATS was compared to existing hardware-assisted test insertion techniques. A demonstration application shows the use of ATS co-processor in multi-threaded environment.

**Acknowledgements**

I would like to take the chance to express my sincere gratitude to many people that have helped and supported me along the journey of my M.Sc. study.

First, my deepest gratefulness belongs to Dr. Michael Smith, for his patience, advice, guidance and thoughtful consideration on every aspect of my life as an international student. Without his support, this thesis would be a mission impossible.

I thank Dr. James Miller (University of Alberta) for his valuable advice and feedback on my research work. I thank Dr. Frank Maurer (University of Calgary) for his suggestions on performing a systematic mapping study as part of my research.

Next, I send my thank you to all my colleagues in the Small Microsystems for Increasing Life Expectancy (S.M.I.L.E) lab at ICT 520A, University of Calgary – special thanks to Samiul Choudhury, Emily Marasco, Jordan Woehr and Mila Gorobets.

Last but not the least, my thank you goes to my lovely friends in Calgary – Hai Wang, Lingyun Lei, Longchen Liu and Michael Wasson. Without you guys, my life would be miserably boring. And Xuxiao Mao, nice to meet you in the summer of 2013.

This work was financed through an industrial collaborative research and development grant co-sponsored by Analog Devices, Natural Sciences and Engineering Research Council (NSERC) of Canada and University of Calgary.

## Dedication

Dedicated to my beloved mom. For she taught me how to be a man.

献给我挚爱的妈妈。因为她教会了我如何成为一个男人。

## Table of Contents

Abstract ..... ii
Acknowledgements ..... iii
Dedication ..... iv
Table of Contents ..... v
List of Figures and Illustrations ..... ix

CHAPTER ONE: INTRODUCTION ..... 1
1.1 Research Motivation ..... 1
1.2 Thesis Organization ..... 4
1.3 Thesis Contributions ..... 5

CHAPTER TWO: TWO WORLDS – AGILE AND EMBEDDED ..... 8
2.1 Embedded Systems ..... 8
2.2 Agile Methodologies ..... 11
2.2.1 Extreme Programming ..... 11
2.2.2 Scrum ..... 13
2.2.3 Test Driven Development ..... 14
2.3 Evolution of Embedded Software Development Lifecycles ..... 15
2.3.1 Chaos ..... 15
2.3.2 Waterfall lifecycle model ..... 16
2.3.3 V-Shaped ..... 17
2.3.4 Agile-Inspired Embedded Software Development (AIESD) ..... 18
2.4 Chapter Summary ..... 19

CHAPTER THREE: SYSTEMATIC MAPPING OF AGILE-INSPIRED EMBEDDED SOFTWARE DEVELOPMENT ..... 20
3.1 Introduction to Systematic Mapping Methodology ..... 20
3.1.1 What is a Systematic Mapping Study? ..... 21
3.1.2 Related Secondary Studies on AIESD ..... 23
3.1.3 Modified Systematic Mapping Process ..... 24
3.2 Conducting Modified Process for AIESD Systematic Mapping ..... 25
3.2.1 Research Questions ..... 25
3.2.2 Search Approach ..... 29
3.2.3 Screening of Papers and Key-wording of Abstracts ..... 31
3.2.4 Data Extraction and Mapping ..... 32
3.3 Bibliometric and Demographic Trends of AIESD (RQ1) ..... 32
3.3.1 RQ1.1 – Year of Publication ..... 33
3.3.2 RQ1.2 – Article Type Classification ..... 36
3.3.3 RQ1.3 – Venue Counts ..... 37
3.3.4 RQ1.4 – Most Cited (Popular) Paper ..... 38
3.3.5 RQ1.5 – Active Researchers ..... 41
3.3.6 RQ1.6 – Observed Challenges to Validity ..... 42
3.4 Discussion ..... 44
3.5 Chapter Summary ..... 46

CHAPTER FOUR: SYSTEMATIC MAPPING REUSLTS AND ADAPTED AND ADOPTED AGILE PRACTICES AND TOOLSETS....47
4.1 Systematic Mapping Results of AIESD (RQ2)....47
4.1.1 RQ2.1 – Research Type Facet....48
4.1.2 RQ2.2 – Contribution Facet....49
4.1.3 RQ2.3 – Adapted Agile Methods / TDD Used....53
4.1.4 RQ2.4 – Tool Functionality Facet....55
4.1.5 RQ2.5 – Domain of Study Facet....56
4.1.6 RQ2.6 – Case Study Included....57
4.2 Adapted and adopted Agile Practices and Toolsets....58
4.2.1 Testing Frameworks....58
4.2.2 Embedded-Agile Supporting Tools....59
4.2.3 Proposed Embedded-Agile Lifecycles....60
4.3 Discussion....63
4.4 Chapter Summary....66

CHAPTER FIVE: AGILE TEST SUPPORT COPROCESSOR DESIGN....67
5.1 Introduction....67
5.2 Existed Hardware-Assisted Testing Techniques....70
5.2.1 Hardware-Assisted Test-Vector Insertion....70
5.2.2 Hardware-Assisted Code Coverage....70
5.2.3 Hardware-Assisted Test Insertion....72
5.3 ATS Co-processor's Advantages and Hardware Concepts....74
5.4 Expression of Hardware Design Requirement....77
5.4.1 Software Concepts....77
5.4.2 Monitoring Bus Activity....80
5.4.3 Anticipated Agile Test Support Software Overhead Per Test....81
5.4.4 Developing "Just Enough" Hardware to Satisfy the Co-Design Test....83
5.5 Chapter Summary....85

CHAPTER SIX: AGILE TEST CO-PROCESSOR PROTOTYPE AND PERFORMANCE ANALYSIS....86
6.1 ATS Hardware Design Decisions....86
6.2 Identifying Data and Instruction Watch Activity....91
6.3 Performance Comparison of Existing and Proposed Hardware Test Insertion Mechanism....95
6.4 Implementing ATS Co-processor on a Dual Core Embedded Processor....100
6.5 Chapter Summary....101

CHAPTER SEVEN: DEMONSTRATION OF USING AGILE TEST CO-PROCESSOR TO DETECT RACE CONDITION....102
7.1 Introduction to the Case Study System....102
7.2 Race Condition Detection Method Used....103
7.3 ATS Co-processor Running Results....104
7.4 Performance Analysis....107
7.5 Chapter Summary....110

CHAPTER EIGHT: CONCLUSION AND FUTURE WORK .....111
8.1 Summary and Conclusion .....111
8.2 Future Work .....113

REFERENCES .....115

APPENDIX A: DISCOVERED STUDIES FOR SYSTEMATIC MAPPING STUDY .128

APPENDIX B: DISCOVERED STUDIES ASSOCIATED WITH EACH FIGURE ....139

## List of Tables

Table 3.1 – Inclusion and exclusion criteria for screening of papers ..... 31

Table 3.2 – Venues with at least 2 publications (ranked by number of publications)..... 38

Table 3.3 – The top 20 studies (ranked by absolute citation number)..... 38

Table 3.4 – Active researchers (ranked by number of publications) ..... 41

Table 4.1 – Comparison of proposed Embedded-Agile Lifecycles..... 62

Table 6.1 – ATS co-processor memory mapped registers (Specific to ADSP-BF533 EZ-KIT Lite)..... 94

Table 6.2 – Theoretical and Actual Performance Comparison of ATS Co-processor and Existing Hardware-assisted Test Insertion Methods..... 100

## List of Figures and Illustrations

Figure 2.1 – Planning and sprint loop in Extreme Programming and Scrum (Adapted from (Beck 2004)). ..... 13

Figure 2.2 – Tests are written to before the code is developed in the Test Driven Development approach. This customer oriented approach is repeated in the unit tests written by developer (Adapted from (Beck 2004)). ..... 15

Figure 2.3 – The Waterfall lifecycle with overlapping stages (Adapted from (Boehm 1988)). ... 17

Figure 2.4 – The V-Shaped software development lifecycle (Adapted from (Rook 1986)). ..... 18

Figure 3.1 – Year of publication, showing the number of studies published in each year. ..... 33

Figure 3.2 – Systematic mapping of year facet against the Article Type facet. ..... 33

Figure 3.3 – Systematic mapping of year facet against the research type facet. ..... 34

Figure 3.4 – Systematic mapping of year facet against the contribution facet. ..... 34

Figure 3.5 – Systematic mapping of research type facet against the article type classification facet. ..... 36

Figure 3.6 – Systematic mapping of contribution facet against the article type classification facet. ..... 36

Figure 3.7 – Histogram of number of citations for all studies in our study. ..... 39

Figure 3.8 – Histogram of normalized number of citations for all studies in our study. ..... 39

Figure 4.1 – Research type facet. ..... 48

Figure 4.2 – Contribution facet. ..... 49

Figure 4.3 – Number of contributions in one study. ..... 50

Figure 4.4 – Research type facet mapped against the contribution facet. ..... 50

Figure 4.5 – Systematic mapping of research type facet and adapted methods / TDD used facet. ..... 53

Figure 4.6 – Contribution facet maps against adapted methods / TDD used facet. ..... 53

Figure 4.7 – Systematic mapping of adapted methods / TDD used facet and tool functionality facet. ..... 55

Figure 4.8 – Systematic mapping of research type facet and domain facet. ..... 56

Figure 4.9 – Systematic mapping of contribution facet and domain facet. ..... 56

Figure 4.10 – Research type of studies that included case studies. ..... 57

Figure 4.11 – Contribution facet of papers with case studies ..... 57

Figure 5.1 – Schematic of the levels of a FIT test framework necessary within an embedded environment for validating proposed customer acceptance tests (XPI-level 1) through XPI-stages 2 to 5 (Smith et al. 2009a) and a ASIC Stage 6 proposed by Johnston (Johnson 2011). Equivalent layers of unit tests are needed by the developer at later stages of development. Modified from (Smith et al. 2009a). ..... 68

Figure 5.2 – E-RACE's hardware approach of data race detection avoids extensive software instrumentation. Modified from (Huang et al. 2008). ..... 72

Figure 5.3 – The ATS co-processor monitors activity on the processor's PM and DM address buses. The co-processor logic activates the processor's NMI line upon recognizing a designated activity. ..... 76

Figure 5.4 – The code in this screen capture is a simplified version of a test that uses the ATS co-processor to verify whether a thread holds the correct locks when accessing a specified memory location. The test in red is expressed in the syntax of the EmbeddedUnit testing framework (Smith et al. 2009a; Smith et al. 2009b; Smith et al. 2005). The code highlighted in yellow are the ATS co-processor related function calls. .... 79

Figure 5.5 – A screen capture of the code supporting mocked read and write memory operations monitored by the ATS co-processor and the format of the non-maskable interrupt code activated via software interrupt (*whichATSmode = MOCK_ALL*) or hardware (*whichATSmode = MOCK_BUSSES, TRUE_ATS*). ..... 80

Figure 5.6 – A screen shot of ATS co-processor reducing testing overhead by deactivating itself following the first access of shared memory within a tight loop. ..... 83

Figure 5.7 – This schematic shows the development of just enough FPGA logic to satisfy the ATS co-processor tests shown in Figures 5.4 to 5.6. ..... 85

Figure 6.1 – This schematic illustrates the configuration of the FPGA ATS co-processor relative to Blackfin BF533 core. The external coprocessor can only monitor the SDRAM bus activity and not the core L1 memory activity, in contract to Blackfin's instruction and data watch unit capabilities. ..... 89

Figure 6.2 – Bus capture blocks logic. On the raising edge of SCLK (System Clock), the four control signals: CS (Chip Select), RAS (Row Access Strobe), CAS (Column Access Strobe) and WE (Write Enable) are used to determine when to latch the bus information, provided the ATS_EN (ATS co-processor Enable) line is high. Since the address bus is multiplexed, row address, column address for instruction watch (memory read operation) and column address for data watch (memory write operation) are distinguished by different patterns of the four control signals. Data bus's value is latched only when

column address of data watch appears on the address bus. The data value is used to configure the ATS coprocessor's registers ..... 90

Figure 6.3 – ATS co-processor SPI configuration state machine ..... 91

Figure 6.4 – Program flow of the FPGA-based test insertion system. (1) The multi-thread program running on the embedded processor first initializes the co-processor via an SPI or standard memory write instructions. (2) Then the original threads are run with (3) co-processor monitoring the processor's data and address bus activities. On finding a watched activity (4), an NMI is generated (5) causing the processor to (6) run an NMI handler which (7) inserts a test before (8) allowing the watched activity to execute without further interruptions. ..... 92

Figure 7.1 – ATS co-processor configuration. Demonstrated in CrossCore Embedded Studio environment ..... 104

Figure 7.2 – Code for starting the ATS co-processor data watch and the EmbeddedGear application as well as reporting the race condition ..... 105

Figure 7.3 – UART console output, reporting the EmbeddedGear running information and race condition information. ..... 106

Figure 7.4 – Disassembly window shows which instruction made the race condition occur..... 106

Figure 7.5 – Flow chart of ATS co-processor detecting race condition in a multi-threaded application on recognizing a shared memory access ..... 108

## List of Symbols, Abbreviations and Nomenclature

| Symbol | Definition |
|---|---|
| AIESD | Agile-Inspired Embedded Software Development |
| ATS | Agile Test Support |
| ATS_EN | Agile Test Support co-processor Enable |
| CAS | Column Access Strobe |
| CAMUG | Calgary Agile Methodologies User Group |
| CCES | CrossCore Embedded Studio |
| CPLB | Catch Protection Logic Buffer |
| CS | Chip Select |
| DM | Data Memory |
| ETDD | Embedded Test Driven Development |
| FPGA | Field-Programmable Gate Array |
| GPIO | General Purpose Input/Output |
| IDDE | Integrated Design and Development Environment |
| ISR | Interrupt Service Routine |
| JTAG | Joint Test Action Group |
| KIS | Keep It Simple |
| MCH | Model, Conductor and Hardware |
| MMF | Minimum Marketable Feature |
| MVP | Minimal Viable Product |
| NMI | Non-Maskable Interrupt |
| PM | Program Memory |
| RAM | Random-Access Memory |
| RAS | Row Access Strobe |
| RQ | Research Question |
| SCLK | System Clock |
| SDRAM | Synchronous Dynamic Random-Access Memory |
| SoC | System on Chip |
| SPI | Serial Peripheral Interface |
| TDD | Test Driven Development |
| VDK | VisualDSP Kernel |
| VDSP | VisualDSP++ |
| WE | Write Enable |
| XP | eXtreme Programming |
| XPI | eXtreme Programming Inspired |
| XSR | eXception Service Routine |

# CHAPTER ONE: INTRODUCTION[1]

## 1.1 Research Motivation

Embedded systems are ubiquitous in today's society with applications ranging from personal and public health to business, public transportation, industry and education. Embedded systems products are characterized as task-specific, highly customized, price and size sensitive in order to provide satisfactory effectiveness and efficiency to customers. Many embedded systems products are required to fulfill real-time requirements and work using processors with limited computational power and stringent memory requirements. These systems must often work reliably in harsh conditions for relatively long periods of time with minimum or zero maintenance. Some mission-critical embedded products, e.g. a central control hub controlling operations in a nuclear power plant, must function correctly all the time. Malfunction of these embedded products would cause catastrophic consequences with massive monetary losses, unrecoverable environment destructions or most importantly, human health endangerment. Thus embedded systems with mission-critical applications demand the utmost reliability.

An often cited document reported software defects in business enterprise systems caused an estimated $59.5 billion annual loss of profits in US in 2002, with more than one third of that loss, 22 billion / year, considered as reducible with existing best development practices (Tassey 2002). It is reasonable to anticipate similar losses and possible profit recovery can be associated with

---

[1]This chapter is derived from the following paper:
**Deng, D.**, Smith, M., & Miller, J. A Systematic Mapping Study of Approaches for Agile-Inspired Embedded Software Development. To be submitted to *Journal of Information and Software Technology*.

embedded system software development. Given this level of reducible cost, considerable effort is being placed on attempts to identify whether key defect-reducing strategies from one software development area can be moved, successfully, into another.

*Agile methodologies* (Beck and Andres 2004) offer an iterative development process capable to react to changing requirements during the course of enterprise system software development, comparing to linear software development process, for example, *Waterfall* (Boehm 1988) and *V-Shaped* (Rook 1986) lifecycles. Agile methodologies emphasize the ability to respond to requirement changes, software quality, communication between developers and customers and collaboration among developers. *Test driven development (TDD)* (Beck 2003), as one of the core practices of *eXtreme Programming* (*XP)* (Beck and Andres 2004), encourages developers to write test cases before the product code is actually developed. Test cases are expressions of the customer's requirement with all of the test cases developed have to be passed during each stage of the whole development cycle. In this way, the software development is '*infected*' with tests (Beck 2003), which can therefore lead to an improvement in software quality (Maximilien and Williams 2003; Beck and Andres 2004; Bhat and Nagappan 2006; Madeyski and Szała 2007; Nagappan et al. 2008). *Test-first* practices also enable defect removal at a lower cost as the expense on fixing hidden defects discovered late in a project using test last approaches is avoided. Further the early formalized expression of the customer's requirement means that the cost of 'gold-plating', the development of unnecessary features, is avoided.

In the past decade, the Agile-inspired Embedded Software Development (AIESD) research field has grown richer and more mature since Grenning (Grenning 2002), Greene (Greene 2004) and Dahlby (Dahlby 2004) first considered applying Beck's Agile methodologies (Beck and

Andres 2004) as a pro-active defect reduction process onto embedded system development. To become Agile, guidance on adapting and adopting Agile methods for use with embedded software development and efficient and effective automated tool support are necessary (Dahlby 2004; Greening 2007; Smith et al. 2009a; Smith et al. 2013). Two systematic literature review studies (Shen et al. 2012; Xie et al. 2012) targeted characteristics of embedded software development and Agile methods used in embedded domain respectively, but did not present a panoramic view of AIESD (research types, contributions of the papers and etc.) or bibliometrics statistics (trend of the research field, popular papers, researchers and venue of publication). Agile practitioners (Dahlby 2004; Greene 2004; Smith et al. 2009a; Smith et al. 2011; Smith et al. 2013) reported that limited tool support constrains the adoption of Agile methods. Greene (Greene 2004) and his team further elaborated this toolset deficiencies as they are forced to create their customized tools to foster the unit testing and test code setup.

To address these problems, this thesis focuses on (1) providing an overview of state-of-art AIESD research field by conducting a systematic mapping study to answer the observation by Smith et al (Smith et al. 2009a); and (2) proposing and developing an 'Agile test support' (ATS) co-processor to provide / improve the test insertion capability of current embedded processors.

## 1.2 Thesis Organization

The remainder of this thesis is set as follows:

The two different worlds of embedded systems and Agile methodologies are discussed in Chapter Two. Both embedded systems characteristics and core Agile practices are presented. The evolution of embedded software development lifecycles is explored.

In order to obtain the overview of current AIESD research field, the systematic mapping methodology is discussed in Chapter Three. Using the modified systematic mapping process, we present how we conducted the systematic mapping study on AIESD and the results of each step. The bibliometric and demographic trends of AIESD research community are first examined.

The research questions from the systematic mapping are answered in Chapter Four. By going into the details of the discovered papers in our systematic mapping study, we examined the Agile-inspired adapted and adopted methods and toolsets which are proposed to enable the embedded software development to become more agile. Discussions on these results are made.

In Chapter Five, we propose an 'Agile test support' (ATS) FPGA-based co-processor which is able to provide low-overhead test insertion support for embedded testing frameworks. The capability and advantages of this co-processor are outlined. The initial development is presented in detail.

Further hardware design decision details and the working mechanism of a practical ATS co-processor are presented in Chapter Six. The performance of ATS test insertion co-processor is compared to the existing hardware-assisted test insertion techniques.

Chapter Seven demonstrates the use of ATS test insertion co-processor in a real multi-threaded application at length. The performance analysis on number of cycles that the ATS co-processor takes to perform the data race detection is discussed.

Finally, conclusions and future work are presented in Chapter Eight.

### 1.3 Thesis Contributions

Three papers (Deng et al. 2013; Smith et al. 2013; Deng et al. 2014) were derived from this thesis work. I am the first author of the Irish Signals and Systems Conference (ISSC 2013) paper (Deng et al. 2013) and I presented that paper in Letterkenny, Ireland. I was the second author of the Journal of Signals Processing and Systems (JSPS) paper (Smith et al. 2013) where I was responsible for designing, implementing and performing the performance analysis and comparison of the ATS test insertion co-processor and a major rewrite of this paper. I am the first author of the systematic mapping study paper (Deng et al. 2014) which is in preparation for a January 2014 submission to Journal of Information and Software Technology (JIST). This systematic mapping paper was started from a graduate course project for Agile Software Engineering (SENG 615) at University of Calgary and I updated the systematic mapping results in October 2013.

The major goal of this thesis is to provide a big picture of current Agile-inspired embedded software development research field to summarize the fruit of existing researches and propose tool support to make the primitive embedded software development become as defect-free as possible. As such, I conducted a systematic mapping study (Petersen et al. 2008) on AIESD in order to identify, synchronize, structure and summarize the current existing researches on

AIESD. Furthermore, I developed an ATS FPGA-based co-processor which is prototyped to provide test insertion capability to embedded processors with low overhead.

Following a previously suggested approach (Petersen et al. 2008), I proposed sets of AIESD research questions and developed search strings to be applied to major research databases. This systematic mapping study included 78 discovered studies within 171 retrieved articles published between 2002 and 2013. Twelve AIESD research questions are presented with associated mapping results. Biblometric results were extracted from the included papers as well as the risks to the validity of the study are evaluated. Research type classification, contribution made, adapted Agile methods / test driven development method used, proposed tool functionalities, domain of the studies and cases study included are discussed. To obtain a detailed review of the proposed methods and toolsets, significant papers are located, and selected for investigation to obtain a picture of best practice. The results of this systematic mapping study offer an overview of research into AIESD and can act as a baseline paper for follow-up research on AIESD ideas. This is the first systematic mapping study on AIESD research field.

To offer the AIESD research community with more effective and efficient tool supports, Smith (Smith et al. 2010; Smith et al. 2013) proposed a FPGA-based co-processor which is able to provide test insertion capability universally to different families of embedded processors. The advantages, software and hardware design decisions of ATS co-processor are outlined. A performance comparison was made between hardware-assisted test insertion methods using the proposed ATS co-processor and the existing instruction and data watch debugging units of the Analog Devices ADSP-BF533 Blackfin processor (Analog Devices 2011a). The co-processors' overhead was orders of magnitudes lower than that for the Blackfin's existing data watch unit;

and better than the existing instruction watch unit and other suggested approaches that involved combinations of this processor's data watch, instruction watch and cache-miss detection units. The concept of this work was initially accepted as a journal paper by Journal of Signals Processing Systems (JSPS) (Smith et al. 2013). I further detailed the implementation and performance analysis of ATS co-processor and published and presented this work in Irish Signals and Systems Conference as the first author in June 2013 (ISSC 2013) (Deng et al. 2013). In August 2013, we published this work (Smith et al. 2013) in JSPS with a major rewrite and most updated results where I was the second author.

# CHAPTER TWO: TWO WORLDS – AGILE AND EMBEDDED[1]

In Chapter One, the need for reliable embedded systems development has been briefly introduced. Agile methodologies, as a defect-reduction software development method, have been transferred from the enterprise world to the embedded software development. However, inadequate development tools and processes are major obstacles. In this chapter, the characteristics of embedded systems are presented and general Agile methodologies are described. Some of the core embedded development practices are explored. This chapter provides background knowledge for further discussion on Agile-Inspired Embedded Software Development (AIESD) in Chapters Three to Seven.

## 2.1 Embedded Systems

The term of embedded systems is indistinct to many people despite the fact that they use hand-held computers and many electronic devices in their daily lives. Dahlby (Dahlby 2004) has loosely defined embedded systems as "*a system that is not primarily a computer but contains a processor*"; a definition that probably remains still obscure to the general public. Peckol's embedded system's definition (Peckol 2007) is "*An embedded system is a computer system that targets dedicated tasks with mechanical or electric peripherals often with real-time constraints*".

---

[1]This chapter is derived from the following paper:
**Deng, D.**, Smith, M., & Miller, J. A Systematic Mapping Study of Approaches for Agile-Inspired Embedded Software Development. To be submitted to *Journal of Information and Software Technology*.

A clearer understanding of the characteristics of embedded systems is explored in the following paragraphs:

- *Limited Computational Power and Memory:* Compared to desktop computers, embedded systems often contain far less computational power and memory. The available processing capabilities, memory and peripherals are just enough for the tasks that need to be performed; the extra resources would increase unnecessary costs and size of the embedded products and thus lower the profits and market popularity.

- *Efficient Power Usage:* Battery life is one of the major concerns for electronic devices, e.g. cell phones, and determines how long the system is able to run. Therefore, these embedded systems products must be engineered as power efficient as possible.

- *Dedicated Tasks with Customized Hardware:* Embedded systems are often designed to perform a small set of dedicated tasks continuously and repeatedly. The hardware is customized specifically to fulfill the requirement of targeted tasks. Thus, the development of embedded systems usually involves a software and hardware co-design pattern which requires developers to possess detailed knowledge on both software and hardware of the system as a whole.

- *Sensitive to Price and Size:* To offer an appealing price point and portability to end users, embedded systems are engineered to possess just enough resources, such as processor power, size of on-chip / off-chip memory and number of peripherals. Many embedded systems products also have physical size constraints, size and weight for example, to be as portable as possible.

- *Constrained to meet Real-time Requirements: Many* embedded systems have to respond / handle events in a fixed time provided. For example, an audio signal is passing through the systems at 48 kHz then the embedded systems must be able to process the audio sample within

1/48 millisecond. This often acts a baseline to determine how much computational power the embedded systems need.

- *Limited Development Tools:* Unlike enterprise system software development environments that enjoy rich development tools, embedded counterparts often use basic editors and compilers in many development environments because embedded systems use custom hardware from different vendors, which have limited tool support. Low-level software and hardware interactive debugging tools, for example, JTAG (Joint Testing Action Group) (Whetsel 1997), are available for debugging purpose (Karlesky et al. 2006).

- *Extensive Reliable Uptime under Harsh Environments:* Some embedded systems are often used in harsh environment, e.g. extremely low or high temperatures, intense radiation or magnetic fields, dusty or high humidity (Liggesmeyer and Trapp 2009). They are also used for mission-critical purpose, such as military and medical purposes. Thus, the requirements for reliability and exception handling are more rigorous than for many other types of software development. They are frequently working behind the scene and are supposed to have extensive reliable uptime with little or no human intervention. Furthermore, for some embedded systems, it is very difficult or even impossible to update the software. Therefore, having the code as defect-free as possible is often a requirement for even the initial commercial release of embedded systems products.

These embedded system characteristics lead to three important facts of embedded system development: (1) Embedded systems have to be engineered reliably; (2) Hardware may be delivered in the late stage of development lifecycle and thus limiting the available software development and testing time; (3) Defects would be caused by software, hardware or the

combination of these two, which is a nightmare for embedded developers. Because of these facts, embedded developers are eager for efficient and effective testing tools and development methods to overcome the crude embedded world.

## 2.2 Agile Methodologies

Agile methodologies (Beck and Andres 2004) were initially proposed to handle the changes of customer requirements during development. Over time, Agile methodologies have been developed and recognized as a lightweight software development process with emphasis on people, communication and software quality. According to the Agile Manifesto (Fowler and Highsmith 2001), the key Agile values are:

- ***Individuals and interactions over processes***
- ***Working software over comprehensive documentation***
- ***Customer collaboration over contract negotiation***
- ***Responding to change over following a plan***

To further explain the concepts above, the three most well-known Agile methods, Extreme Programming (XP), Scrum and Test Driven Development (TDD) are discussed:

### *2.2.1 Extreme Programming*

Extreme Programming is intended to provide frequent deliverables in short development cycles (iterations) in order to improve software quality, productivity and response to new or changing customer requirements. Some core XP practices are (Beck and Andres 2004):

- ***Planning game*** – Customers and programmers together plan what will be released for this release or iteration. Customers select features to be implemented while programmers estimate features costs in the planning meeting.
- ***Small and quick deliverable*** – Executable artifacts are delivered at the end of each iteration cycle. Customers provide feedbacks based on deliverables to developers and new changes can be adopted and worked on in the next iteration.
- ***Keep it simple (KIS) design*** – Build just enough code to pass the tests that express the requirements. Extra code is considered as unnecessary and should not be written unless tests need it.
- ***Pair programming*** – Each task is assigned to two developers. One developer writes the code while the other developer reviews the code and come up with possible improvement methods and test cases to the code. The knowledge of both developers can be shared.
- ***Refactoring*** – Improve the old code / architecture continually. This includes renaming the class name, method name and variable name for readability purposes and updating the architecture for maintainability reasons.

### *2.2.2 Scrum*

Scrum is a project management process which divides a project into short iterations, or called *sprints*. The duration of sprint is typically defined as two weeks. Before each sprint, sprint planning meeting is held to select tasks that are going to be done, decide how much time required to finishing each selected tasks and how many tasks can be done in this sprint cycle. Then the tasks are prioritized and added to backlog. Daily scrum, or daily stand-up, is held on each day of sprint in order to enable project team communication.

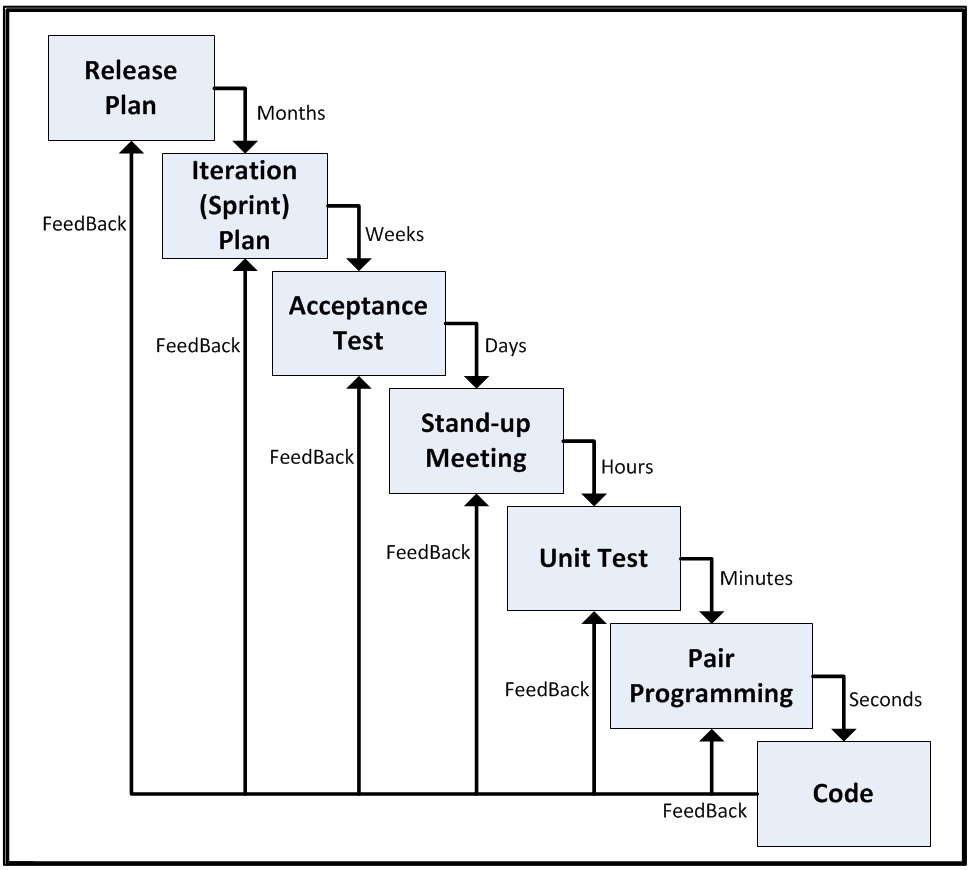


**Figure 2.1 – Planning and sprint loop in Extreme Programming and Scrum (Adapted from (Beck 2004)).**

### *2.2.3 Test Driven Development*

Test Driven Development is a core Agile practice which is intended to improve the software quality by '*infecting*' the software development with tests (Beck 2004). Customer requirements are expressed as test cases (executable documents) which are considered as an effective software design and communication approach between customers and developers (Beck 2003; Maurer and Hellmann 2013). Test cases developed in TDD are unit test oriented, though they also build up a test suite repository for use during regression testing. As the system grows, new test cases will be incrementally added to the test suite with the test suite continuously re-run to make sure the newly developed code did not introduce new defects which break the code that was working previously.

Figure 2.2 depicts the TDD approach. The first step of TDD is to write new tests according to a prioritized list of the customer requirements. The tests are run, with a basic stub written for the required code, to ensure that the tests fail if the incorrect functionality is present – the test of the test (Beck 2003). Then just enough code (Keep It Simple principle) is developed to pass the tests that were just written. New tests based on further customer requirement would not normally be written until all the existing test cases have been passed. However new tests may be added to better explore the system to understand why previous tests have failed. On completion, new test cases based on the next highest priority in the customer requirements are developed and then new code is written to fulfill the tests. This loop is repeated until all the requirements are finished. Figure 2.2 depicts the TDD approach explained above.

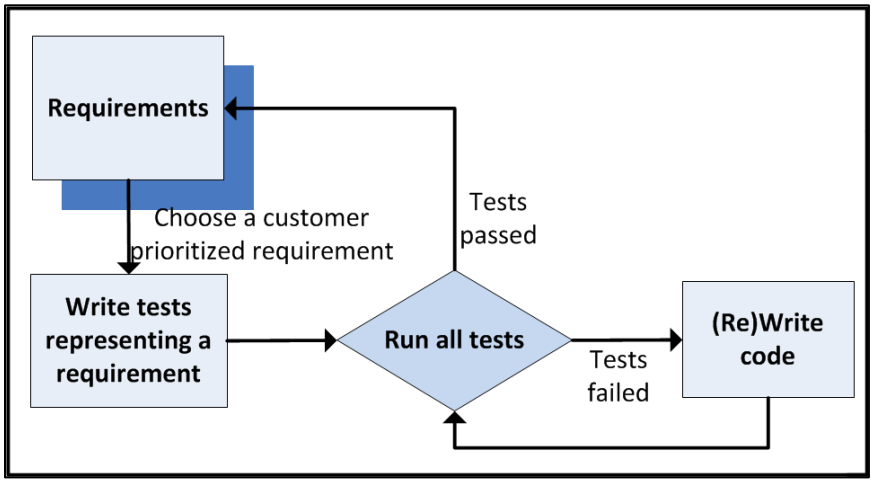


**Figure 2.2 – Tests are written to before the code is developed in the Test Driven Development approach. This customer oriented approach is repeated in the unit tests written by developer (Adapted from (Beck 2004)).**

## 2.3 Evolution of Embedded Software Development Lifecycles

Before the existence of Agile methodologies, software development lifecycles were used to ensure successful analysis, planning, development, testing and maintenance for both desktop and embedded software development. Evolution of embedded software development lifecycles are introduced in this section.

### *2.3.1 Chaos*

Early embedded software development, in both embedded and enterprise worlds, can described by a *Chaos* model with developers simply starting to tackle with the coding without any detailed requirement, design, testing or maintenance analysis or planning (Dahlby 2004). This software development model only works for relatively small and simple embedded systems development with small amount of source code and only a few hardware modules within a whole system

Furthermore, the Chaos model fails to enable the software development to evolve gracefully to match the changing or accumulating needs and requirements (Lehman et al. 1997; Hassan and Holt 2003).

### *2.3.2 Waterfall lifecycle model*

The *Waterfall* lifecycle model, on the other hand, is a software development lifecycle that attempts to address the issues associated with chaos model. By decomposing the software development into interrelated development stages, the Waterfall lifecycle serves better to software development and has been, and continues to be, the basis for software engineering for a long period of time (Boehm, 1988). However, this is a risky software development model since it attempts to develop the fully requirement and design analysis and planning at the beginning of the software development stage. The preliminary planning and estimations may prove to be miscalculated (Boehm 2000). Performing software testing and verification at the late stage of lifecycle (*test-last*) may induce extremely expensive costs to fix the uncaught defects occurred in early design stage (Sommerville 1996). Changes in requirements, especially at the late stage of the project, are costly to be addressed (Pressman 1997). This model is illustrated in Figure 2.3.

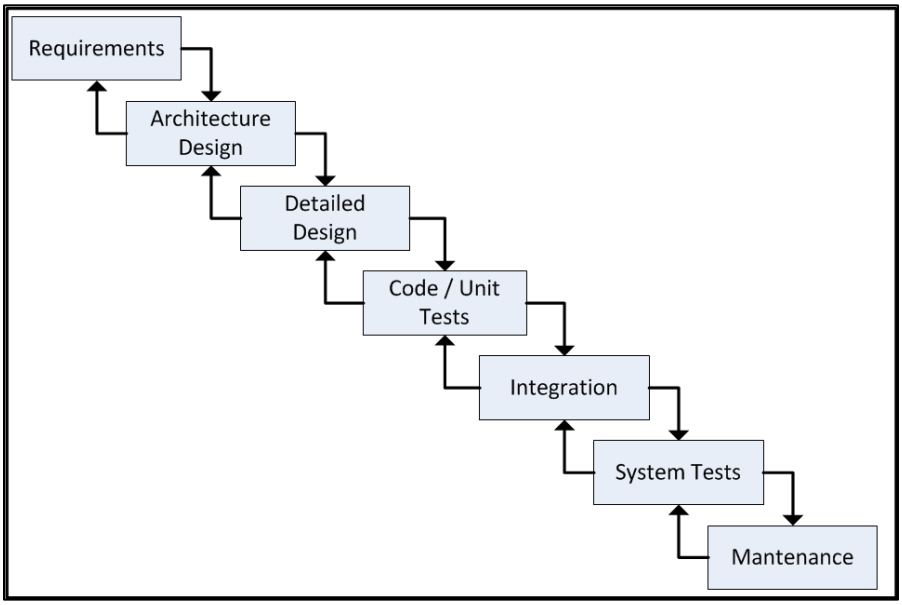


**Figure 2.3 – The Waterfall lifecycle with overlapping stages (Adapted from (Boehm 1988)).**

### *2.3.3 V-Shaped*

Instead of conducting the linear development approach associated with the Waterfall lifecycle mode, the V-Shaped lifecycle 'bends' the development stages upwards after the coding stage in order to associate related stages with testing (Rook 1986). Figure 2.4 presents the V-Shaped lifecycle. However, both Waterfall and V-Shaped software development lifecycles fail to address the fact that requirements from customers are evolving as the development progresses. Furthermore, since most of embedded system development consists of both hardware development and software development, the hardware is very likely ready at the late stage of lifecycle. This would extensively limit the available software development and software testing time and therefore, greatly reducing the software quality (Dahlby 2004; Grenning 2007).

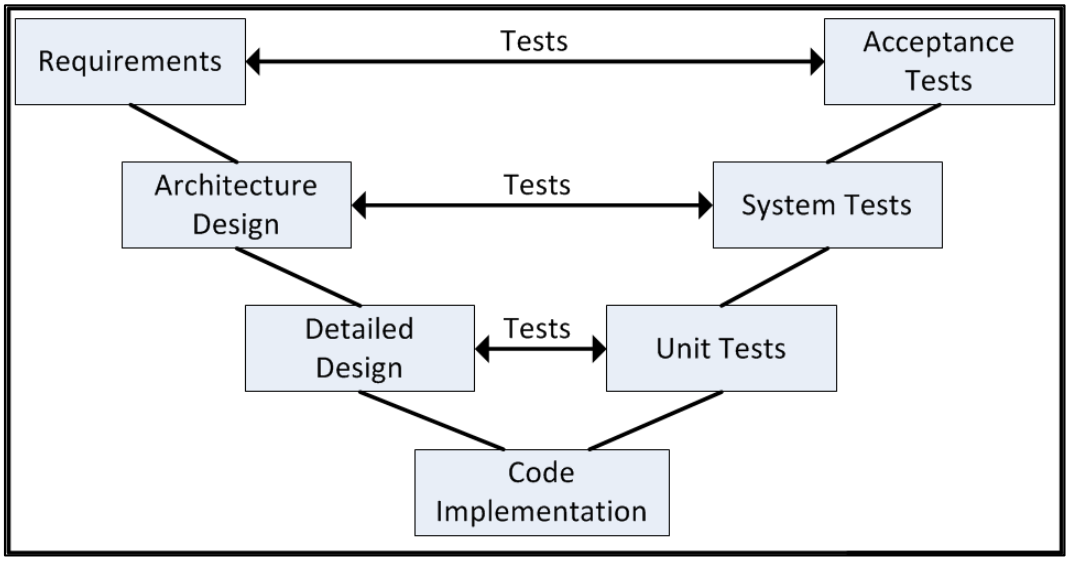


**Figure 2.4 – The V-Shaped software development lifecycle (Adapted from (Rook 1986)).**

### *2.3.4 Agile-Inspired Embedded Software Development (AIESD)*

Embedded developers began to transfer the successfulness of Agile method as a defect-reduction development method in the emerging stage of Agile methodologies (Grenning 2002). As a fact, many embedded developers exploring Agile for the first time get the feeling of '*deja vu*'. For example 'Scrum' is an Agile management process which divides a project into short iterations, or '*sprints*'. What is the difference between '*Scrum*' and developing a product using an iterative or spiral lifecycle? We have begun to see these processes as similar in concept, hence the '*deja vu*', but differing in some important details.

Inspired by Agile methodologies, embedded Agile lifecycles were proposed. *Extreme Programming Inspired (XPI)* was proposed by (Smith et al. 2009a; Smith et al. 2011; Smith et al. 2013). This approach was designed to *infect* the typical embedded development stages with re-useable tests to ensure the embedded software development evolves gracefully and reliably. As

will discussed later in greater detail later, test are generated, and later reused, during requirement's gathering (XPI-stage 1), high-level prototyping (XPI-stage 2), initial production (XPI-stage 3), optimization with architecture features (XPI-stage 4), optimization with co-processor (XPI-stage 5) and possible associated with application-specific integrated circuit (ASIC) (XPI-stage 6) (Johnson 2011). Grenning (Grenning 2007) proposed an Agile-related lifecycle that captured more details of XPI stage 3 and 4. His *Embedded-TDD (ETDD)* was aimed at isolating the software and hardware development in order to make progress before the hardware availability and separating software defects from hardware. Chapters Three and Four further investigate these and other AIESD lifecycles.

### 2.4 Chapter Summary

Embedded systems development is featured with several special characteristics and thus encounters a set of specialized difficulties that needs extra cares. As with enterprise systems, traditional embedded software development lifecycles have evolved from Chaos model to Waterfall and then to V-Shaped lifecycles. However, these basic lifecycles fail to tackle the changing requirement issues and their linear development manner constrains the embedded software development and testing issues. The core practices of Agile methods have been presented as possible defect-reduction software development approaches adaptable for embedded system processes. The tailored Agile methods – XPI and Embedded TDD, are introduced specifically to handle issues where traditional embedded software development lifecycles fail. A systematic mapping study is presented in next chapter to provide an overview of current Agile embedded software development research field.

# CHAPTER THREE: SYSTEMATIC MAPPING OF AGILE-INSPIRED EMBEDDED SOFTWARE DEVELOPMENT[1]

Chapter Two discussed the backgrounds of the two different worlds – embedded systems and Agile methodologies. As discussed in Chapter One and Two, Agile methodologies are defect-reduction software development methods intended to make embedded software development more reliable. Therefore, the current available researches on adapted and adopted Agile methods applied to embedded software development are extremely important in this thesis. However, there is no available flagship research to summarize this research area. To render an overview of state-of-art researches on Agile-inspired embedded software development (AIESD), we conducted a systematic mapping study and present the results of it in this chapter. A brief introduction on systematic mapping methodology is first discussed and then followed by the adapted process of systematic mapping study for AIESD. The bibliometric and demographic trends of included studies as well as discussions on these results are presented. The systematic mapping results of AIESD research field are offered in next chapter.

## 3.1 Introduction to Systematic Mapping Methodology

This section introduces what as systematic mapping study is. The differences of systematic mapping study and systematic literature review study are presented. Original systematic mapping process is tailored for the use of AIESD research field in this thesis.

---

[1]This chapter is derived from the following paper:
**Deng, D.**, Smith, M., & Miller, J. A Systematic Mapping Study of Approaches for Agile-Inspired Embedded Software Development. To be submitted to *Journal of Information and Software Technology*.

### *3.1.1 What is a Systematic Mapping Study?*

A systematic literature review study, which is often conducted, selects and combines research methods, techniques and evidence from available high quality research papers in order to find out answers to a specific research question. In contrast, a systematic mapping study is a method appropriate to review, analyze and structure papers in a specific research field in order to provide a general overview of the undertaken research as Petersen et al. suggested (Petersen et al. 2008). A systematic mapping study is recommended to be used in research areas, such as AIESD, which lack both primary high-quality studies and cross references between studies (Kitchenham and Charters 2007). Thus a systematic mapping study should be considered as an effective method for researchers who are interested in gaining a baseline, panoramic view of a certain research field; potentially prior to undertaking a more comprehensive systematic literature review to obtain a more detailed view (Petersen et al. 2008; Kitchenham et al. 2011).

Systematic mapping studies and systematic literature review studies are categorised as two major underpinnings of literature reviews which is a type of secondary research. However, Petersen et al. (Petersen et al. 2008) and Kitchenham et al. (Kitchenham et al. 2011) indicated that systematic mapping studies and systematic literature reviews are different in many traits:

- **Goal:** Systematic literature review inspects comparative papers to answer specific research questions whereas systematic mapping studies investigate almost all research papers in one certain research area to obtain a summarized state-of-art of that research field. The results of systematic mapping studies are mostly statistics on various dimensions of the interested facets of the research field, whereas systematic literature review results are aggregated evidence on a certain method / technique.

- **Research Questions:** Systematic mapping studies look at broad generic questions, e.g. how many papers proposed a new method / tool, whereas systematic literature reviews ask specific questions based on empirical evidence and/or experimental results, e.g. is method A better than B?
- **Depth, breadth and effort required:** A larger body of papers can covered via a systematic mapping study which does not require papers to be reviewed with the same detail as within a systematic literature review.
- **Searching and inclusion / exclusion strategy:** Systematic mapping studies use more general search strings to provide a broader range of results than systematic literature reviews and do not usually include / exclude papers based on the quality of the paper. To identify a larger quantity of papers in a newly emerging field, systematic mapping studies use the primary (general) keywords of the research area and include all the papers discussing the subject. Systematic literature reviews, however, exclude papers that are low quality or not highly related to a specific method / technique of a certain research field to make sure that the papers are evaluating the best quality results.
- **Threats to validity:** As Mendes (Mendes 2005) and Jorgensen and Shepperd (Jorgensen and Shepperd 2007) pointed out, 73% of research papers are designed in a wrong way and are not performing controlled experiments. As a consequence, systematic mapping studies may result in the incorrect inclusion or wrong classification of papers. This threat may be avoided or minimized in systematic literature review which investigates papers in a fine-grained way that removes low quality papers. The very newness of the AIESD field must also be considered as a possible threat to the validity of a systematic mapping study, e.g. inconsistent use of agile terms combined with the author's use of non-AIESD keywords in the paper title and abstract when

submitting to established journals and conferences that do not normally cover the combined area of Agile and embedded system development.

While systematic mapping studies and systematic literature reviews traits are different as mentioned above, they are actually complementary and should be combined to provide both a panoramic view and refined detail on a specific research field as suggested in (Mujtaba et al. 2008; Petersen et al. 2008; Kitchenham et al. 2011). However, Kitchenham (Kitchenham et al. 2011) reported there is no follow-on research (including systematic literature review) making use of systematic mapping studies study.

### *3.1.2 Related Secondary Studies on AIESD*

We have investigated the current available secondary studies on AIESD. However, limited secondary studies have been reported for the embedded software development field. Antonio et al. (Antonio et al. 2012) resented a systematic mapping study on embedded software architectures, which is unrelated to Agile methodologies. Xie et al. (Xie et al. 2012) and Shen et al. (Shen et al. 2012) generated systematic literature review studies that respectively targeted characteristics of embedded software development and Agile methods used in embedded domain but did not present a panoramic view of AIESD (research types, contributions of the papers and etc.) or bibliometrics statistics (trend of the research field, popular papers, researchers and venue of publication). In particular these studies do not attempt to determine to what extent embedded system developers and researchers have responded to an observation by Smith et al. (Smith et al. 2009) which can be paraphrased as: "*It is one thing to adapt an Agile process (such as method or tool) for embedded system development; but an entirely different thing for the concept (the philosophy of Agile) to be adopted by developers*".

Therefore, we decided to conduct a systematic mapping study on AIESD research field to address the comment from Smith et al. and provide a panoramic overview of AIESD. This is the first systematic mapping studies reported on AIESD research field.

### *3.1.3 Modified Systematic Mapping Process*

We have modified the five-step systematic mapping methodology for software engineering topics proposed by Petersen et al. (Petersen et al. 2008).

1. The research questions are used to define the research scope of this study.
2. Search for all possible studies before screening. The proposed research questions and field of study should guide the search strings developed.
3. Develop and apply an established set of inclusion and exclusion criteria to better identify those papers properly within the research scope.
4. Key-wording is a systematic approach to develop a classification scheme which takes all existing studies into account (Petersen et al. 2008). In this step, the reviewers read the abstract to identify keywords and concepts that characterize the paper's contribution, and identify the context of the research. The final set of keywords is then clustered to form categories (facets), and groups of categories.
5. With the classifications established, the included papers are sorted into a scheme and a table of frequencies for each keyword discussed. This permits identification of which categories and keywords have been emphasized by past researchers with the gaps indicating possibilities for future research (Petersen et al. 2008).

In the next section, we present results for each of the steps of the systematic mapping process. The details of conducting the systematic mapping study are presented. We present the

research questions and answer them with the results of systematic mapping study. Bibliometric analysis and demographic trends are also presented in addition to traditional systematic mapping study results.

## 3.2 Conducting Modified Process for AIESD Systematic Mapping

In this section, we propose a process of conducting an AIESD systematic mapping. Findings, trend and implications of AIESD research questions proposed are provided in Sections 3 and 4. To obtain a finer result of systematic mapping study, we propose the mapping study should be performed in an iterative manner to give feedback on the previous processes. The results and experience gained from later processes should be used to refine the earlier processes, for example, we can update the research questions and the search approach while we are screening and key-wording the papers.

### *3.2.1 Research Questions*

We have generated two main categories of AIESD research questions into: bibilometric analysis and demographic trends (RQ1) inspired by (Garousi et al. 2013) to understand the depth of study in this new field and systematic mapping research questions (RQ2) inspired by (Petersen et al. 2008). To have superior fine-grained views on these two research questions, we further divide them into sub-questions:

**RQ1 – Bibliometric and demographic trend:** By asking the following research questions, a bibliometric and demographic analysis of the research field is presented. Assessing the

bibliometric and demographic trend can assist in identifying the best practice, papers, researchers and venues for follow-up researches (Garousi et al. 2010; Garousi et al. 2013).

- **RQ1.1 – Year of publication: What is the publication number across each year in last decade?** We map this research question with some sub-questions of RQ2 in order to see the trend of AIESD.
- **RQ1.2 – Article type classification: Is the article a conference paper, journal paper, technical report, thesis, book or magazine article?** Since the research community of AIESD is a relatively new research area, we want to know what article type has been favored by the researchers and developers. One paper only belongs to one article type.
- **RQ1.3 – Venue counts: Which venues (conference, journals and etc.) are popular for papers from AIESD?** It provides indications for searching or publishing AIESD papers.
- **RQ1.4 – Most cited papers: Which papers are cited the most frequently in the field of AIESD?** Absolute citation number and normalized citation number (average citation count since the paper is published) are presented and ranked.
- **RQ1.5 – Active researchers: According to the number of publication in the field and the total citation number of the publication the author owns, who are the active researchers?** We only count the first author in this research question due to the complexity of considering the contribution of the second and other authors.
- **RQ1.6 – Observed challenges to validity: Is there early evidence of the challenges to the validity of this study?** According to the systematic mapping results presented, we want to discuss the validity threats observed in the early stage. This research question would be helpful to future systematic mapping studies.

**RQ2 – Systematic mapping:** Based on our personal research and industry experiences, we found the key questions on AIESD field are: (1) Which of the Agile methods have been transferred from the enterprise world to embedded systems software development; (2) Which Agile methods have been tailored for AIESD use; (3) What tools are available to developers who want to adopt an Agile approach to embedded software development; and (4) Are facets of Agile being ignored, or under-utilized by developers and researchers seeking improved development approaches in a safety critical field? These questions are translated into research questions as following (Answers to these results are made in next chapter):

- **RQ2.1 – Research type: Is the paper a validation, evaluation, solution, philosophical, opinion or experience paper** (Wieringa et al. 2006)**?** This research question is general and independent of the specific AIESD focus area. The rational for this question is to classify papers into different research methods and identify which research methods have been focused upon or neglected. The research types were treated as mutually exclusive, meaning a paper can only belong to one research type. No AIESD validation research types were found. The solution research type covers all studies proposing a new solution.
- **RQ2.2 – Research contribution: What type of contribution does the paper offer to the research community** (Petersen et al. 2008)**?** As suggested by Petersen et al. (Petersen et al. 2008), it is important to know whether the study proposes a new tool, a process to implement a method, a method to solve a problem, a model (example) of the method, or provides a metric to measure results / techniques or evaluation results of a practise. A given paper can make multiple contributions.
- **RQ2.3 – Adapted Agile methods / TDD used: What are the papers that specifically discuss TDD or adapted methods for use in embedded software development context?**

We found that papers in AIESD are more TDD oriented from the review of discovered studies. Since TDD is an actual development practice, researchers tend to move it or tailor it for the use of embedded software development. Therefore, we proposed the papers that discuss TDD or adapted Agile methods to be selected separately in this research question and mapped with various research questions so as to reflect the trend, research type and contribution of those papers. The selected papers from the discovered studies can only belong to either TDD or adapted Agile method category.

- **RQ2.4 – Domain of study: What embedded system domains are the Agile methods applied onto?** Is the study, discussing on medical embedded products, embedded system education, control system or general embedded software? By presenting this research question, a general idea of which domain of study has been focused and readers can look for papers in a specific field which they are interested in. A paper only belongs to one domain of study.
- **RQ2.5 – Case study included: Does the paper include a case study section?** Case studies are an important empirical evidence for researchers and developers to evaluate the results of a certain method or technique. Therefore, studies with case study can provide more insights to the research community.
- **RQ2.6 – Tool functionality: What functionality exists within a proposed AIESD tool?** Tools can enable or provide better features on: unit test, regression test, acceptance test, mock generation (using software to mimic the behavior of existing or planned hardware and/or software; e.g. simulate a return value of a proposed sensor), build utility (e.g. automatically link the required objects), model-based testing, refactoring, integration testing, version control and image processing for medical devices software development. In this sub-

research question, we went beyond the systematic mapping study requirement suggested in (Petersen et al. 2008) of only reviewing abstracts. As there was not necessary agreement on AIESD related terms used in the abstract, we found it sometimes is necessary to investigate the introduction and conclusion if the abstract was too vague to classify a paper. Given the newness of the AIESD area, the body of the papers were examined to identify what tool functionalities they proposed. One paper can propose one or more tools with different functionalities. We classify the tool functionality purely based on what the paper said; evaluation of the tool functionality was considered out of this study's scope.

### *3.2.2 Search Approach*

As a relatively new area of study, we recognize that AIESD embedded aspects are not being generally accepted by main stream Agile developers, while AIESD's software engineering aspects are not generally accepted by main stream Embedded Developers. We therefore considered it appropriate to investigate possible matches from small industrial conferences, personal blogs, Agile-related websites in addition to those from the main stream scientific literature. To meet this requirement, the search strings associated with Embedded-Agile concepts and application areas were presented to the following data bases and search engines: (A) IEEE Xplore, ACM Digital Library, Springer Link, Elsevier; (B) Google Scholar and Microsoft Academic Search; and (C) Google.

The search strings used for this initial overview of the two worlds of Agile and embedded systems were related to key Agile concepts (Agile, XP, scrum, TDD and Lean) and embedded systems related terms (embedded, embedded software, biomedical, military). Biomedical and military devices are embedded products and developers of them might be interested in

investigating alternative development techniques to reduce the number of introduced defects and/or prevent introducing new defects into these products which require the utmost reliability.

The search strings used in this AIESD systematic mapping are listed below:

| |
|---|
| (Agile OR “extreme programming” OR XP OR scrum OR “test driven development” OR TDD OR lean) AND (embedded OR “embedded software” OR biomedical OR military OR army) |

We conducted an initial systematic mapping study during the period September 2011 to December 2011, and updated the results in October 2013. The use of these strings with the Google database to identify possible ‘non-academic’ articles, such as blogs, generated 100,000 hits. As this was an overview using a systematic mapping study approach, rather than a detailed literature, we made a decision to only review the first 500 Google hits in details. By investigating the paper title and abstract, we narrow search results down to 171 studies. These studies, starting in 2002, are in form of books, papers, blog articles, theses, technical reports and commercial magazine articles.

### *3.2.3 Screening of Papers and Key-wording of Abstracts*

**Table 3.1 – Inclusion and exclusion criteria for screening of papers**

| | |
|---|---|
| Inclusion | Studies in any form (books, conference / journal papers, blog articles and etc.) concerning the application of Agile methods onto embedded system software development. Specially, studies about biomedical and military engineering using Agile methods are also included. |
| Exclusion | Studies that are out of the scientific and engineering domain or did not focus on the discussion of adapting or adopting Agile methodologies to embedded system software development. Studies that are not written in English studies are excluded. If two or more studies contain too many similarities, only the latest study is included. |

As a standard approach to include and exclude the studies from the systematic mapping study pool (Petersen et al. 2008), only the abstract of the paper is evaluated, which is a major difference from systematic literature review studies. The criteria of inclusion and exclusion applied onto the studies' abstract are summarized in Table 1. As a result, 78 studies are included into this systematic mapping study. We used the term *included*, *selected* or *discovered* studies to refer to these studies interchangeably in the rest of this thesis. The S-number references that appear in the rest of this chapter and Chapter 4 are the discovered studies with details presented in Appendix A.

Key-wording is a systematic approach to extract the primary traits from all existing studies in order to classify and structure them (Petersen et al. 2008; Kitchenham et al. 2011). We develop the classification scheme by mostly investigating the abstract of the studies so as to reduce the time and effort required as suggested in (Petersen et al. 2008). When the abstract of a

study was not well-formed for keywords to be recognized, we reviewed the introduction, conclusion and sometimes the body of the study to gain a better understanding of the study. The classification scheme has been presented in the research questions (section 4.1).

#### *3.2.4 Data Extraction and Mapping*

All included studies were classified to present their characteristics and contributions to a "big picture" of the AIESD research field once the classification schemes, sets of keywords, had been prepared. A spreadsheet was used to collect and collate the data. The systematic mappings and associated discussions for research questions RQ1 and RQ2 are presented in Sections 3 and 4 respectively.

### 3.3 Bibliometric and Demographic Trends of AIESD (RQ1)

In this section, we present the bibliometric and demographic trends of discovered AIESD studies. The statistical overview of AIESD research field can be easily identified with these research questions.

### *3.3.1 RQ1.1 – Year of Publication*

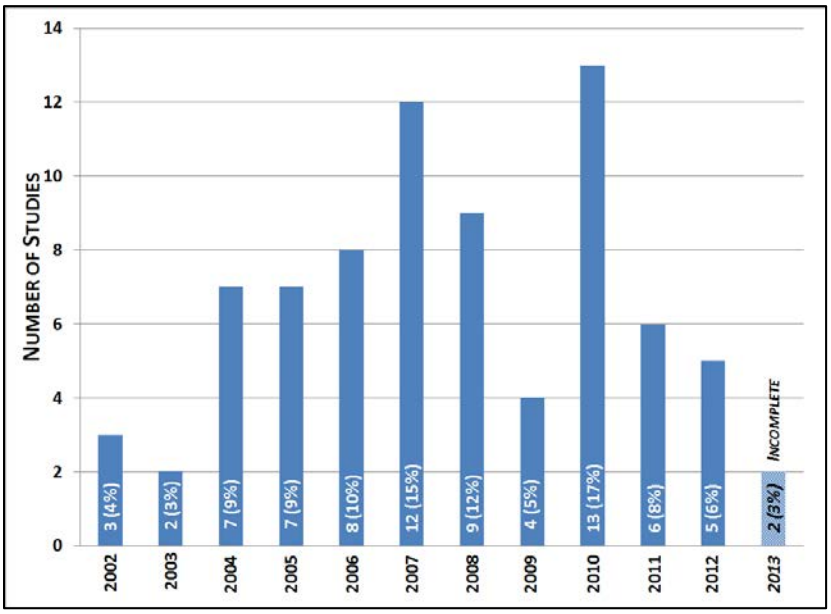


**Figure 3.1 – Year of publication, showing the number of studies published in each year.**

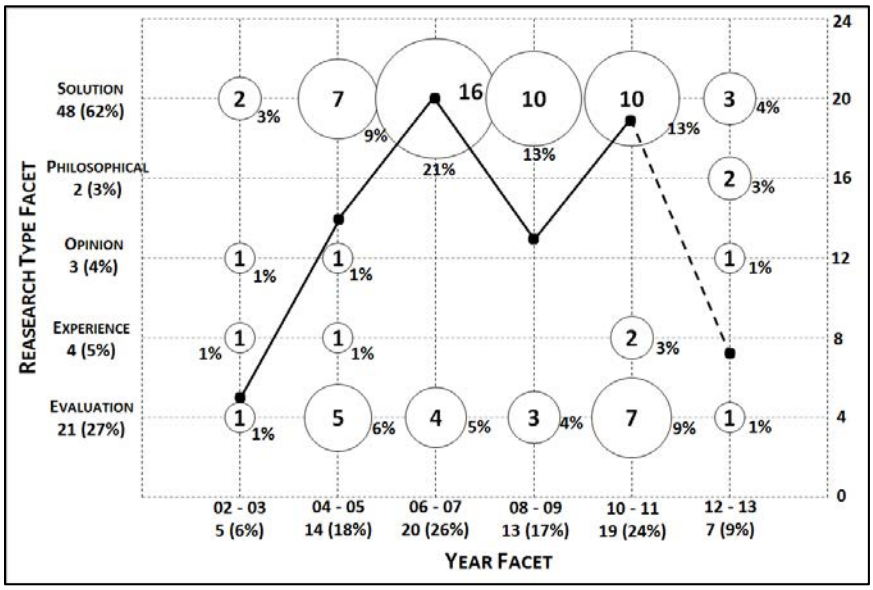


**Figure 3.2 – Systematic mapping of year facet against the Article Type facet.**

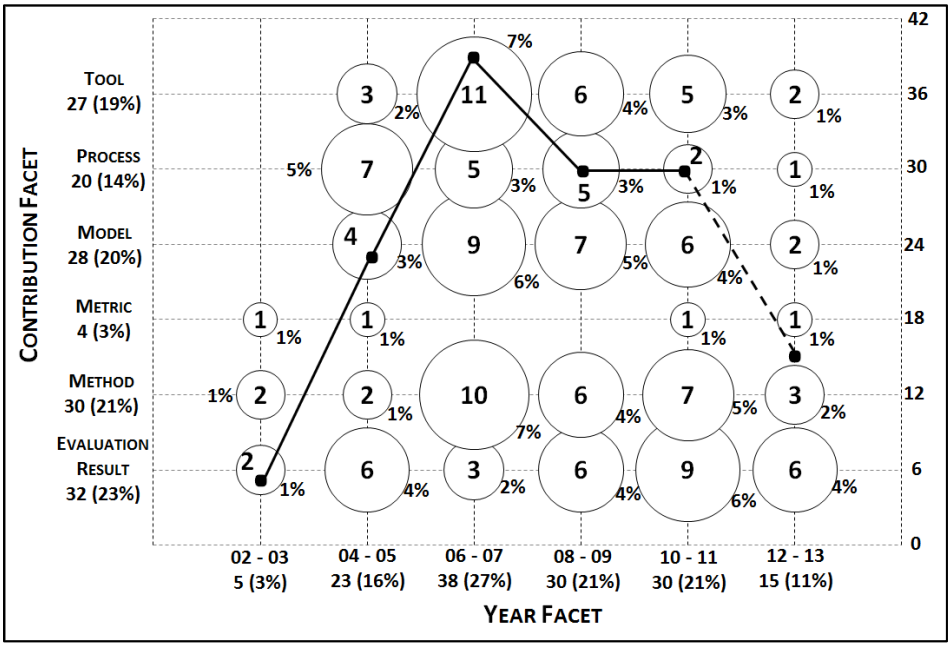


**Figure 3.3 – Systematic mapping of year facet against the research type facet.**

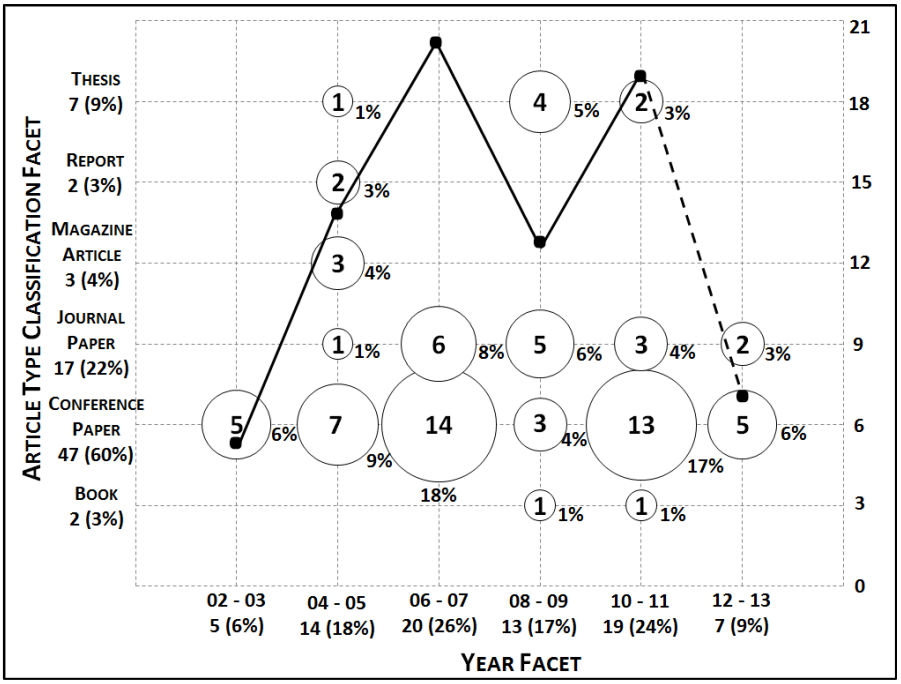


**Figure 3.4 – Systematic mapping of year facet against the contribution facet.**

In this research question, we mapped the year of publication facet to RQ1.2, RQ2.1 and RQ2.2 in order to provide an overview of the trend of AIESD research community. Because of the size of the figures, we decided to group two consecutive years as one category in Figures 3.2 to 3.4. We are not able to include all the papers published in 2013 as we conducted our search in October 2013. Thus we use shaded column for the year of 2013 in Figure 3.1 and dotted line to present the publication count in range 2012 to 2013 in Figures 3.2 to 3.4.

In Figure 3.1, there are two publication peaks in year of 2007 and 2010. Figure 3.2 mapped the year of publication facet and article type facet to present the trend of different publication type. Figure 3.2 further shows that the evaluation studies have a relatively flat publication count across the years whereas the solution studies have a peak in year of 2006 to 2007 and remain in a high number of publications from year of 2004 to 2011. We expect the number of philosophical studies will continue to grow since we believe it is time to identify, structure and summarize the current existing fruit of research of AIESD field. Figure 3.4 systematic mapped the year of publication facet and contribution facet to show the changes of contribution made by AIESD papers across the years.

### *3.3.2 RQ1.2 – Article Type Classification*

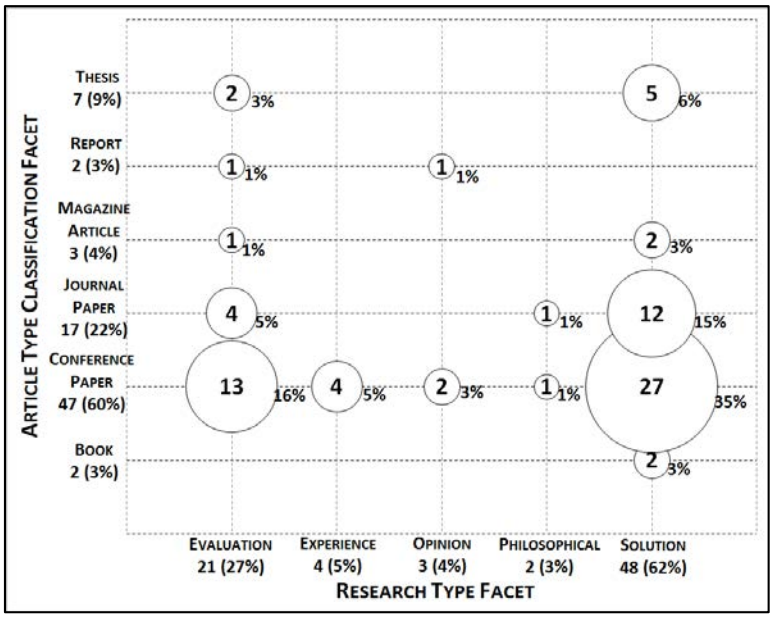


**Figure 3.5 – Systematic mapping of research type facet against the article type classification facet.**

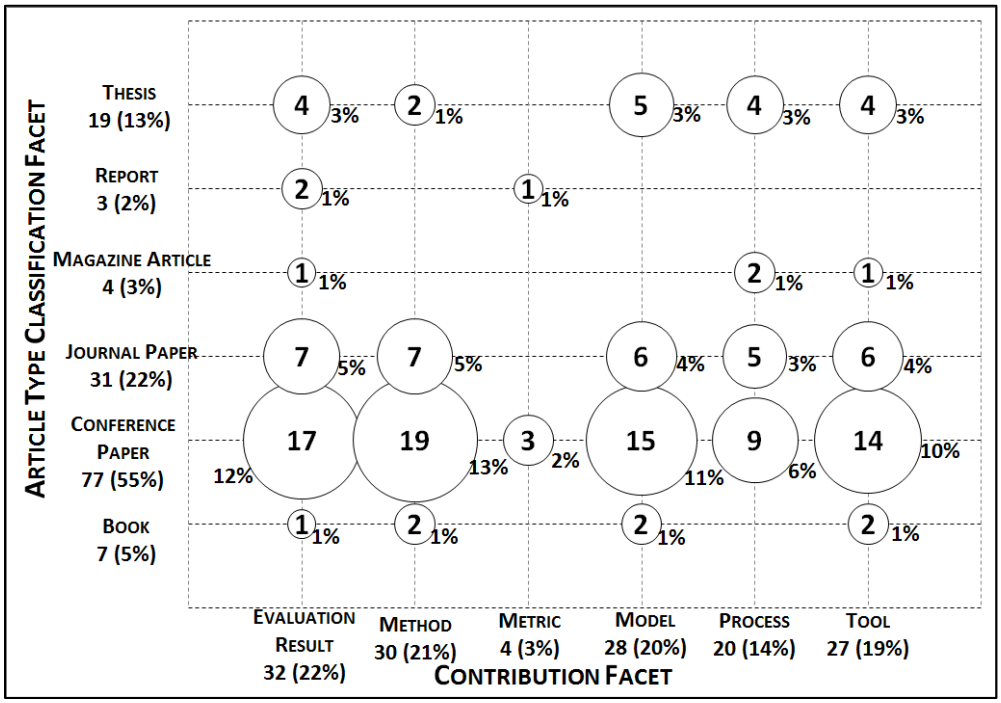


**Figure 3.6 – Systematic mapping of contribution facet against the article type classification facet.**

Since AIESD is a relatively new research area, we category the pool of studies into different article types in order to see the structure of current available studies. Figure 3.5 and 3.6 show the systematic mapping of research type facet and contribution facet map against the article type classification facet, respectively. Conference papers and journal papers contribute the most number of papers (60% and 22% respectively in Figure 3.5) and most contributions (55% and 22% respectively in Figure 3.6) to research community.

### *3.3.3 RQ1.3 – Venue Counts*

This RQ reveals the popularity of venues in AIESD research field. There are 50 different venues / publication sources among 78 studies in our pool (1.56 studies / venue on average), indicating the richness in different domains (e.g. biomedical, control systems, education and etc.) divides the articles into various venues and publication sources. Particularly, we cluster all the theses into thesis publication source. However, we do not think the scattering of AIESD papers across a large number of different venues was a good phenomenon since it reduces the ease with which collaboration and knowledge distribution can occur between members of the research community as will be further discussed in *RQ1.6* which discusses the challenges to the validity of this systematic mapping. From this research question, researchers would be able to choose a suitable venue to publish or search their own research work. Table 3.2 listed the venues that have at least 2 publications with the total citation counts of all papers in each venue. However, we observed that various conferences and journals submission requirement also distracts researchers from consistently using AIESD related keywords, resulting in failure of including all the AIESD paper into this study.

**Table 3.2 – Venues with at least 2 publications (ranked by number of publications)**

| No. | Venue Title | Type | Publication Count | Citation Count |
|---|---|---|---|---|
| 1 | Thesis | Thesis | 7 | 3 |
| 1 | XP Int. Conf. | Conference | 6 | 63 |
| 3 | Embedded Systems Conf. | Conference | 6 | 40 |
| 4 | Agile Conf. | Conference | 4 | 80 |
| 5 | Int. Conf. on Computer Software and Applications | Conference | 3 | 14 |
| 6 | Circuit Cellar Magazine | Magazine | 3 | 3 |
| 7 | Software, IEEE | Journal | 2 | 19 |
| 8 | Journal of Signal Processing Systems | Journal | 2 | 4 |
| 9 | Proceedings of European Conf. on the Use of Modern Information and Communication Technologies | Conference | 2 | 2 |

### *3.3.4 RQ1.4 – Most Cited (Popular) Paper*

**Table 3.3 – The top 20 studies (ranked by absolute citation number)**

| Study No. | First Author | Year | Citation # | Normalized # |
|---|---|---|---|---|
| [S57] | O. Salo | 2008 | 87 | 14.50 |
| [S43] | P. Manhart | 2004 | 52 | 5.20 |
| [S19] | B. Greene | 2004 | 39 | 3.90 |
| [S38] | P. Kettunen | 2008 | 30 | 5.00 |
| [S56] | J. Ronkainen | 2003 | 30 | 2.73 |
| [S74] | N. Schooenderwoert | 2004 | 28 | 2.80 |
| [S34] | D. Kane | 2006 | 25 | 3.13 |
| [S20] | J. Grenning | 2002 | 22 | 1.83 |
| [S37] | P. Kettunen | 2005 | 20 | 2.22 |
| [S64] | M. Smith | 2005 | 17 | 1.89 |
| [S15] | B. Douglas | 2009 | 15 | 3.00 |
| [S38] | M. Karlesky | 2007 | 15 | 2.14 |
| [S35] | M. Karlesky | 2006 | 15 | 1.88 |
| [S58] | J. Savolain | 2010 | 14 | 3.50 |
| [S21] | J. Grenning | 2007 | 13 | 1.84 |
| [S71] | J. Srinivasan | 2009 | 12 | 2.40 |
| [S6] | O. Cawley | 2010 | 11 | 2.75 |
| [S14 | T. Dohmke | 2008 | 10 | 1.67 |
| [S48] | J. Miller | 2007 | 10 | 1.43 |
| [S24] | J. Grenning | 2011 | 9 | 3.00 |

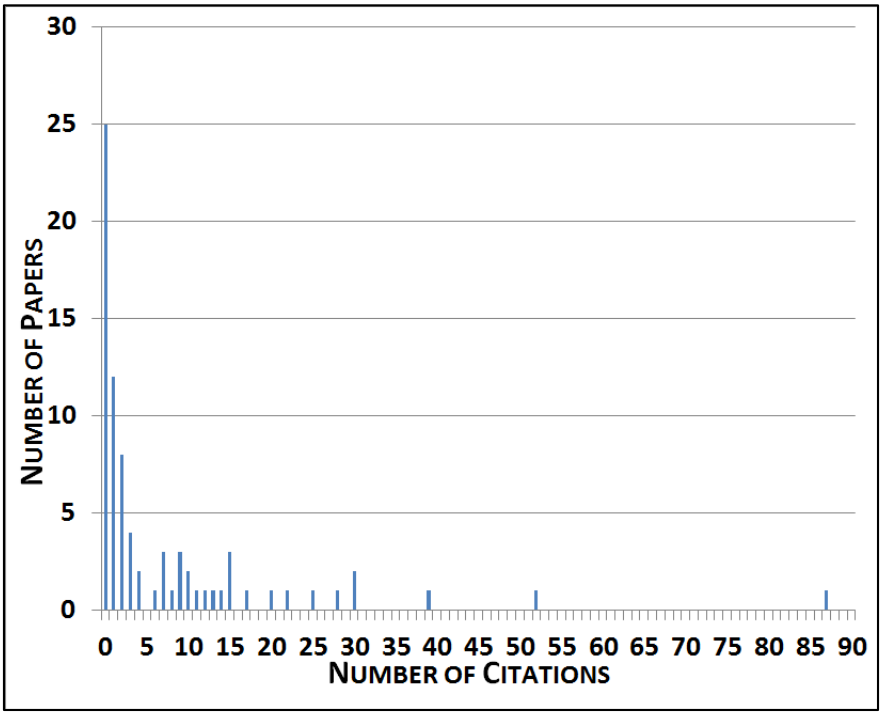


**Figure 3.7 – Histogram of number of citations for all studies in our study.**

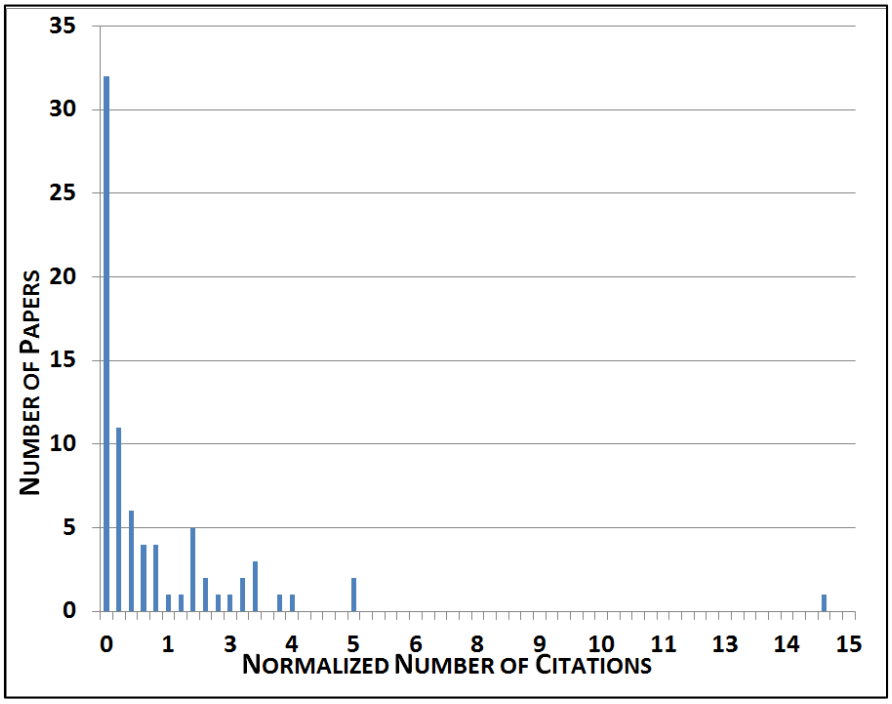


**Figure 3.8 – Histogram of normalized number of citations for all studies in our study.**

We retrieve the citation count of each paper under this study from Google Scholar on October 2013. We use two metrics from Garousi et al. (Garousi et al. 2013) to measure and identify the most cited (popular) paper in the pool: (1) the absolute (total) number of citations of the paper, and (2) the normalized number of citations of the paper because we want to see the average citation count of a paper per year since its publication year, which is defined as follows:

$$NormalizedCitations = \frac{AbsoluteCitation}{2013 - PublicationYear + 1} \qquad (Eqn. 3.1)$$

where 2013 is the year we conducted the literature search and plus 1 to avoid zero divide problem. For example, [S57] has 87 absolute citations and it was published in 2008. Thus its normalized citation is:

$$NormalizedCitations = \frac{87}{2013 - 2008 + 1} = 14.50$$

According to Table 3.3, we found that the most cited papers are [S57], [S43] and [S19] based on both total citation number and normalized citation number. Salo et al. [S57] was among the first studies to conduct a survey based on 13 industrial companies in 8 European countries and 35 individual embedded software development projects which are using Extreme Programming and Scrum. It provides the real data and feedback from the industry. The data and results can be used across all disciplines, not limited to AIESD. Therefore, it gained a large number of citations from both inside and outside the AIESD field in a relatively short period of time. Instead of simply moving the full Agile techniques onto the embedded software development, Manhart et al. [S43] specified that integrating TDD into some classical software engineering processes, fostering the acceptance of Agile techniques. [S43] is among the first papers (published in 2004) which promoting the adoption and adaption of TDD and it indicates the large number of studies on TDD followed while Greene [S19] is delivered from the *Intel*

*Corporation* when they were developing the firmware for Intel® Itanium® processor family in 2004. Figure 3.7 and 3.8 illustrate the Histogram of number of citations and normalized number of citations for all selected studies.

### *3.3.5 RQ1.5 – Active Researchers*

**Table 3.4 – Active researchers (ranked by number of publications)**

| No. | Author Name | Publication Count | Citation Count | Study No. |
|---|---|---|---|---|
| 1 | M. Smith | 10 | 35 | [S61, S62, S63, S64, S65, S66, S67, S68, S69, S70] |
| 2 | J. Grenning | 5 | 44 | [S20, S21, S22, S23, S24] |
| 3 | N. Schooenderwoert | 3 | 37 | [S73, S74, S75] |
| 4 | P. Kettunen | 2 | 50 | [S37, S38] |
| 5 | M. Karlesky | 2 | 30 | [S35, S36] |
| 6 | J. Miller | 2 | 12 | [S47, S48] |
| 7 | T. Dohmke | 2 | 10 | [S13, S14] |
| 8 | J. Chen | 2 | 8 | [S6, S7] |
| 9 | H. Gustavsson | 2 | 1 | [S25, S26] |
| 10 | C. Matthews | 2 | 1 | [S45, S46] |
| 11 | D. Wilking | 2 | 1 | [S77, S78] |
| 12 | T. Punkka | 2 | 0 | [S53, S54] |
| 13 | O. Salo | 1 | 87 | [S57] |
| 14 | P. Manhart | 1 | 52 | [S43] |
| 15 | B. Greene | 1 | 39 | [S19] |
| 16 | J. Ronkainen | 1 | 30 | [S56] |
| 17 | D. Kane | 1 | 25 | [S34] |
| 18 | B. Douglas | 1 | 15 | [S15] |
| 19 | J. Savolain | 1 | 14 | [S58] |
| 20 | J. Srinivasan | 1 | 12 | [S71] |
| 21 | O. Cawley | 1 | 11 | [S5] |

This RQ investigates the which researchers in the field of AIESD contribute to the research community most by their number of publications and total number of citations of all of his / her published papers. Only the first authors of each of the studies are inspected. Table 3.4 shows the

authors who have at least two papers published or have more than 10 citations in his / her publications.

### *3.3.6 RQ1.6 – Observed Challenges to Validity*

Systematic mappings are a contradiction in terms: They are intended for a broad overview of a newly emerging field but require a domain expert for the proper identification of appropriate search strings, inclusion and exclusion criteria formed, the data / keywords exacted, and there will be changes in the field between the last search and publications. We attempted to mitigate these issues by these approaches:

- Reviewing systematic mapping papers on other software engineering field (Engström and Runeson 2011; Kitchenham et al. 2011; Garousi et al. 2013);
- Presenting our findings to an experienced industrial practitioner in the area of practical application of Agile development processes in a number of fields (Geras 2012);
- Making use of the collective experience of the senior researchers in the area of software engineering (James Miller) and embedded systems (Michael Smith). More complicated search strings can be developed with the light of this thesis.

In this thesis, we deliberately used the most common Agile terms (Agile, extreme programming, scrum and test-driven development) to develop the systematic mapping search strings for this initial overview of AESD concepts. According to guidelines of conducting systematic mapping in software engineering domain (Petersen et al. 2008), only the abstract of the paper needs to be reviewed to decide whether to include or exclude the paper and extract the keywords for use in the classification scheme. It is considered acceptable to occasionally use the introduction and conclusion if the abstract is too vague. While this approach is commonly used

to reduce the time and effort to generate a systematic mapping, it is obvious that such studies could be biased by including / excluding a paper incorrectly or biased by not correctly identifying all possible variations of a set of keywords.

An obvious cross-check on the unavoidable extent of this problem is to (1) identify whether this systematic study failed to discover a number of AIESD papers that we know have been published, and (2) identify why these papers remained undiscovered.

The undiscovered Johnson's paper (Johnson 2011), and related blogs, focus more on even more platform specific level than embedded system software development; Agile-inspired integrated circuit development (AIICD) of hardware related to system on a chip design. The undiscovered paper's by Huang et al (Huang et al. 2008) and Tran et al (Tran et al. 2008) covered concepts in test driven development and code coverage tools whose implementation was made possible by custom use of hardware features built into existing embedded system processors. These hardware units designed into the processor core are not regularly available to enterprise system developers, but are a fundamental tool in the embedded developer's arsenal. However these papers had abstracts intentionally targeted towards acceptance by a main stream software engineering conference not normally identified as covering embedded system issues. The title and the abstract of the undiscovered Wiederseiner et al.'s paper (Wiederseiner et al. 2011) was also tailored to emphasize that paper's general software engineering aspects rather than stressing Agile embedded concepts. We consider that a key reason for these studies remaining undiscovered in this systematic mapping as a direct consequence of the current difficulty in publishing AESD ideas in the main stream literature as discussed in the early section on venues (*RQ1.3*).

We decided to exclude these undiscovered papers from the current systematic mapping study for consistency reasons. However we can use the knowledge that the search strings only recognized 10 of 13 of our own papers to suggest that this initial systematic mapping has provided a reasonable first AIESD overview by identifying approximately 77% of the papers that overlap the embedded and Agile worlds. Given the confusion of key-words used in the AIESD study, it may be necessary to examine the field in more detail via a more detailed literature research to improve the coverage rather than performing a further systematic mapping using new search terms. To better permit future researchers to validate their own studies we provide a list of our discovered studies in Appendix A.

### 3.4 Discussion

First of all, to our best knowledge, this study is the first systematic mapping in the field of Agile-inspired embedded software development. Based on the searching string (mostly the combination of Agile-relevant words and embedded-relevant words) applied, 171 studies were found and 78 studies of them are included to perform this systematic mapping study. Comparing to other recent systematic mapping studies in software engineering domain (e.g. Garousi et al. (Garousi et al. 2013) 79 included papers, da Mota Silveira Neto, Paulo Anselmo, et al. (da Mota Silveira Neto et al. 2011) 45 included papers, Engström et al. (Engström and Runeson 2011) 64 included papers and Palacios et al. (Palacios et al. 2011) 33 included papers), the number of 78 included studies in this study is not a small number, which also indicates the eager and discussion of transferring the Agile success to embedded software development world are noticeable.

RQ1.1 presents the trend from the beginning the AIESD research field, 2002, to the most recent paper, 2013, we found. Figure 3.1 reveals that a general increase of number of papers

from 2002 to 2007 and there are two publication peaks in 2007 and 2010. We should be aware that some of the newly published / accepted papers in 2013 are probably not public available by the time we conducted the last search (October 2013). Regarding to Figure 3.2, conference paper contribute to the AIESD research community the most with two publication peaks in year ranges of 2006 to 2007 and 2010 to 2011. No magazine articles and report types are found after 2005. According to Figure 3.3 and 3.4, even though the numbers of papers are increasing comparing in year range 10-11 (19 papers) to 08- 09 (13 papers), the numbers of contributions stay the same (30 contributions for both year ranges). Evaluation results, in Figure 3.4, have two steady increases: from 2002 to 2005 and from 2006 to 2011. We believe the first increase is because the flagship people want to popularize the use of Agile and the second increase is because with a large amount of papers in AIESD published in 2006 to 2007, as more and more people become to recognize the benefits and advantages of Agile and thus want to evaluate the adoption and adaptation of Agile onto the embedded software development domain. RQ1.2 maps the article type classification facet to research type facet and contribution facet respectively. Combining the mapping result of Figure 3.5 and 3.6, expect for the book and thesis type of studies (3% and 9% of publication amount contribute 5% and 13% of contributions), other types of studies contribute approximately the same proportion of their publication amount.

We presented the top venues in Table 3.2 in RQ1.3 and the citation count and normalized citation count for each paper are shown in Table 3.3 in RQ1.4. The most popular papers can be extracted from Table 3.3 easily and the distribution of number of citations and normalized citations are revealed in Figure 3.7 and 3.8, respectively. When comparing the absolute citation number and the normalized citation number of our study to (Garousi et al. 2013), we found that both the absolute and normalized numbers of citations in our study are less. We believe that this

is an indication of the research work of AIESD is more isolated to other research fields and have less cross reference within the AIESD research area. Thus we highly encourage researchers and industrial developers to synchronize the existing research fruit into their current research and work. RQ1.5 presented the active researchers in Table 3.4. RQ1.6 investigated the observed threats to validity of our study.

### 3.5 Chapter Summary

In this chapter, the systematic mapping methodology was introduced. We modified the original systematic mapping study process to make it more suitable for AIESD systematic mapping study used. We have presented a systematic mapping study on papers that investigated adopting or adapting Agile methods onto embedded software domain in order to offer an overview this research field. There were 171 studies found and after exclusion, 78 studies were included which are published between 2002 and 2013. To identify, categorize and summarize the current status of AIESD research area, a set of classification schemes has been generated. Papers were then sorted into the schemes and data are extracted from the pool of studies. Data were presented in the form of mapping to answer the research questions. The bibliometric and demographic trend, year of publication, article type classification, top venues, most cited papers and active researchers were presented. The next chapter presents the systematic mapping results to answer the research questions in RQ2.

# CHAPTER FOUR: SYSTEMATIC MAPPING REUSLTS AND ADAPTED AND ADOPTED AGILE PRACTICES AND TOOLSETS[1]

Chapter Three has presented the bibliometric and demographic trends (RQ1) of the AIESD research field. In this chapter, the AIESD specific research questions are answered with the systematic mapping results. To obtain a detailed review of the proposed methods and toolsets, significant papers are located, and selected for investigation to obtain a picture of best practice.

## 4.1 Systematic Mapping Results of AIESD (RQ2)

In this section, we present various mapping results regarding to RQ2 proposed in Chapter Three. RQ2.1 and RQ 2.2 were proposed by (Wieringa et al. 2006; Petersen et al. 2008) respectively while RQ2.3 to RQ2.6 are new questions.

---

[1]This chapter is derived from the following paper:
**Deng, D.**, Smith, M., & Miller, J. A Systematic Mapping Study of Approaches for Agile-Inspired Embedded Software Development. To be submitted to *Journal of Information and Software Technology*.

### *4.1.1 RQ2.1 – Research Type Facet*

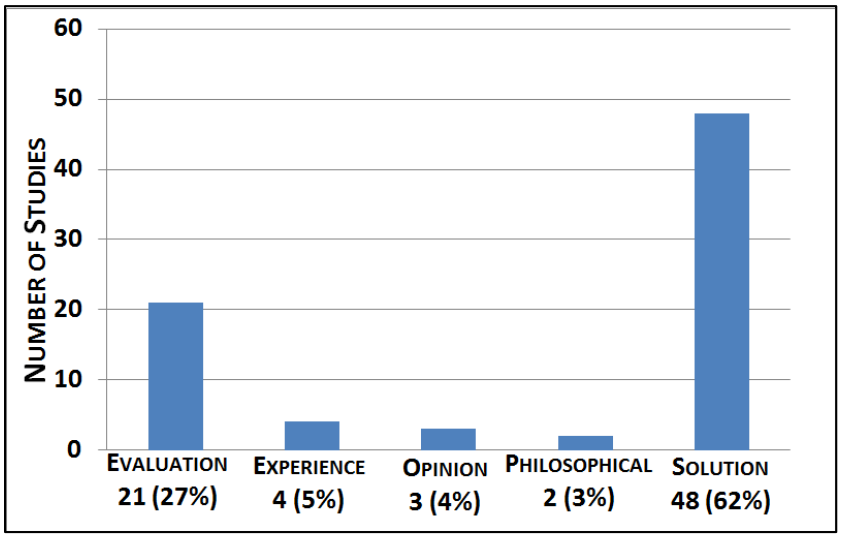


**Figure 4.1 – Research type facet.**

Figure 4.1 shows the distribution of what the research method used in the 78 studies examined. One study can only belong to one research type. This biases the solution type towards having the higher occurrence as we classify a paper to be of this research facet if it proposes a solution, regardless of whether it also evaluates, gives opinions and provides experience to the solution.

There are 21 studies (27%) that evaluate the applicability of well-established Agile methods in the context of embedded software and 48 studies (62%) are proposing new methods / solutions to tailor the Agile methods for use within embedded software development. We believe that the dominance of these two research types (69 papers, 89% in total) is an indication that Agile methods are appealing to researchers and industry people in embedded software development domain as they have shown their desire to assess or tailor Agile for use. However, this also indicates that the desktop / web application oriented Agile methods may still needs special cares when they are applied onto the embedded content. Because evaluation and solution type papers

are evidences that researchers are making effort to adapt and adopt Agile methods. Few studies discuss or summarize the existing proposed new solutions and provide evaluation results of the AIESD field, e.g. Cordemans et al. [S9], Shen et al. [S59] and Albuquerque et al. [S2]. We suggest that a growth in flagship studies and high-quality secondary studies would increase the acceptance of AIESD ideas amongst both software engineers and embedded developers.

### *4.1.2 RQ2.2 – Contribution Facet*

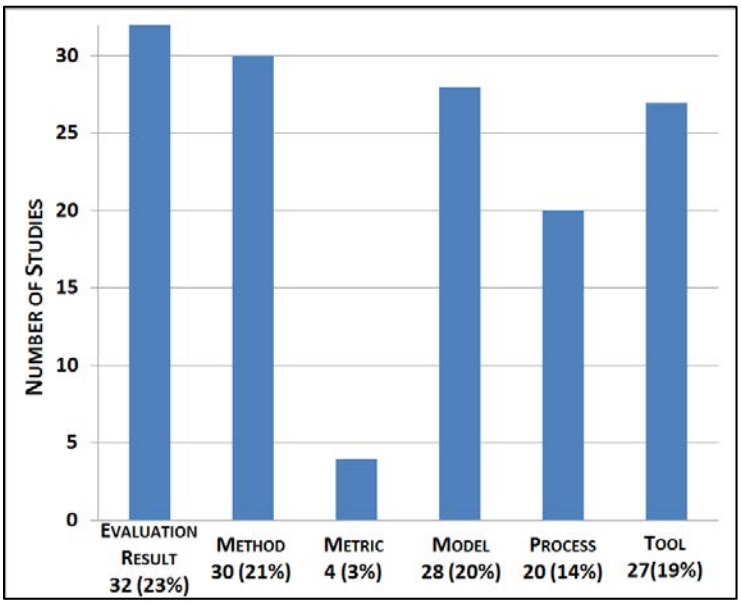


**Figure 4.2 – Contribution facet.**

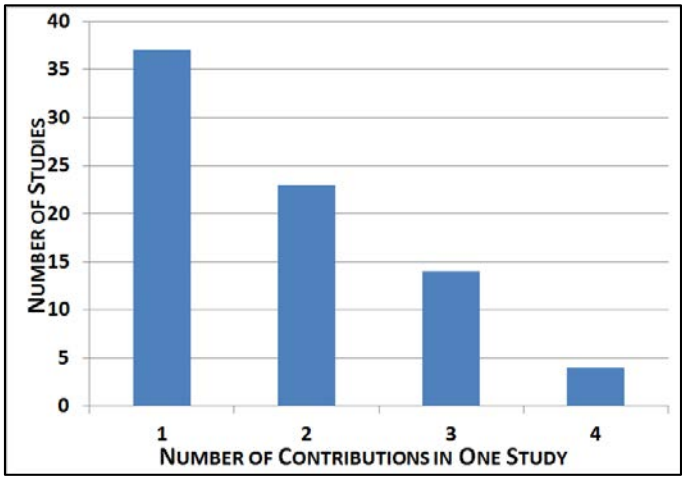


**Figure 4.3 – Number of contributions in one study.**

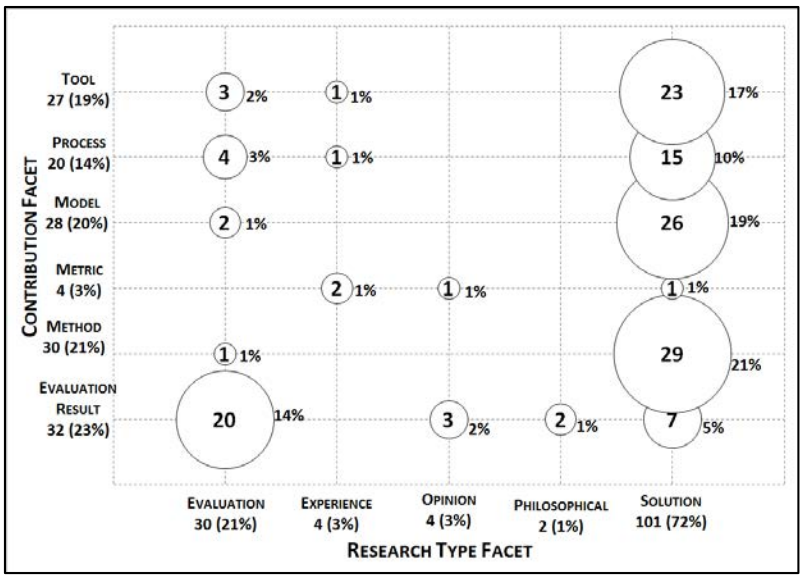


**Figure 4.4 – Research type facet mapped against the contribution facet.**

Figure 4.2 shows the distributions of contributions that are in the pool of studies have offered. One study can offer one or more contributions and therefore the total amount of the contributions is larger than the number of studies. For example, [S67] made 3 contributions: (1) proposed tools

for unit test and acceptance test; (2) a new solution (method) on how to tailor the Agile methods into the embedded software development cycle; and (3) detailed steps (model) on instructing people how to apply the new method. The number of contributions per study is shown in Figure 4.3 so that the papers with most contributions can be recognized (See Appendix A for reference number).

New tools or the new use of old tools were proposed in 27 out of 78 studies (35%, 19% out of 141 contributions made by 78 studies). This implies: (1) with the help of tools, developers can actually take advantages of Agile more easily, in other words, tools are required to support the use of Agile methods within Embedded Software Development; while (2) inadequate tool support remains an obstacle that limits the applicability of Agile methods in this new environment [S12] [S70]. We encourage researchers and industrial developers to continue developing / proposing new tools or new use of tools to enable us to become more Agile.

In Figure 4.4, there are 78 (55%) contributions that are associated with new processes, methods and models. They are the new solutions of applying Agile methods. We suggest more model studies (examples on conducting a process or method) to be presented so that the AIESD community can gain more benefits from the large number of new processes and methods studies. There are 32 out of 78 (41%) studies offer the evaluation results which can certainly provide the guideline, opinion and experience of utilizing Agile methods on embedded content. Studies with a new metric proposed to evaluate the Agile methods in the embedded content seem to have less attention. The systematic study indicated that there are no controlled experiments that compared the cost, development speed, defect rate or developers / customers' satisfaction, etc. between the Agile and the traditional development lifecycles (water-fall and V-shaped).

The systematic mapping of Figure 4.4 reveals that the majority of the contributions are made by the solution studies. However, the evaluation results of the new solutions should be reported more in order to enable the research community to assess and the new solutions. As we mentioned in the introduction, we found no study that actually synchronizes, compares or combines the proposed new methods. Evaluation studies offer second largest number of contributions with many evaluation results. The philosophical studies are systematic literature reviews published in 2012.

### *4.1.3 RQ2.3 – Adapted Agile Methods / TDD Used*

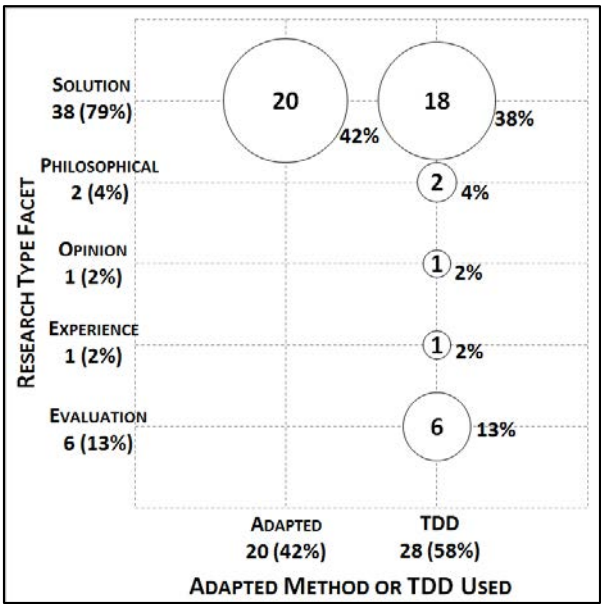


**Figure 4.5 – Systematic mapping of research type facet and adapted methods / TDD used facet.**

**Figure 4.6 – Contribution facet maps against adapted methods / TDD used facet.**

In this research question, we investigated which adapted Agile methods and TDD methods have been discussed in our pool of studies as proposed in Section 2.1. Figure 4.5 mapped the adapted method and TDD used facet against the research type facet. According to Figure 4.5, there are 48 papers (62% out of 78 discovered papers) that discussed adapted methods and TDD. To tailor Agile methods to become more suitable to embedded software development, 20 out of 78 studies (26%) proposed the adapted methods. Papers that proposed new adapted methods are classified as solution papers while papers that discussed TDD have varieties on research types. Solution research type (79%) has the most number of adapted methods / TDD papers. Figure 4.6 mapped the contribution facet against the adapted methods / TDD used facet. According to Figure 4.6 and Figure 4.4, there are 48 papers that discussed adapted methods and TDD contribute 99 contributions (70%) out of 141 contributions in 78 papers. Metrics on how to evaluate the effectiveness and efficiency of TDD and adapted Agile methods are needed.

As can be seen in the Figure 4.6, studies discussing TDD proposed more processes (concrete steps that can be followed) on how to conduct TDD on the embedded content while the studies discussing adapted Agile methodologies proposed more methods on how to apply Agile methods in an innovative way. Only 4 evaluation results are provided for the adapted Agile methods, indicating the lack of investigation on new adapted methods. Adapted Agile methods, on the other hand, proposed more method contributions (how to solve a particular problem) than process contribution (concrete steps on how to implement a method). More than 50% (25 out of 48 papers) of the adapted Agile methods and TDD papers propose new tools or old tools used in a new way.

### *4.1.4 RQ2.4 – Tool Functionality Facet*

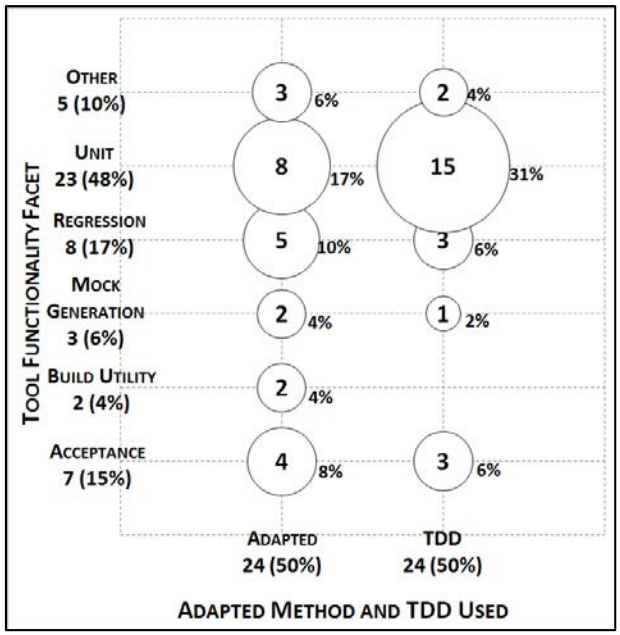


**Figure 4.7 – Systematic mapping of adapted methods / TDD used facet and tool functionality facet.**

If a study made a contribution of proposing tools, we dug into the study and figure out what the type of tools it has proposed. Figure 4.7 presents the systematic map of adapted methods / TDD used facet and tool functionality facet. Because of the limit of the figure size, we group the tool functionalities of model-based testing, refactoring, integration testing, static code analysis, test insertion, image processing and version control as other in Figure 4.7. We found that there is no study proposing a tool to help refactoring.

According to Figure 4.6 and Figure 4.7, 25 adapted Agile method and TDD papers that proposed tools offer 48 tool functionalities. Even though the number of adapted Agile method papers is less than TDD papers (20 to 28, in Figure 4.5), the number of tool functionalities proposed are the same for both categories.

### *4.1.5 RQ2.5 – Domain of Study Facet*

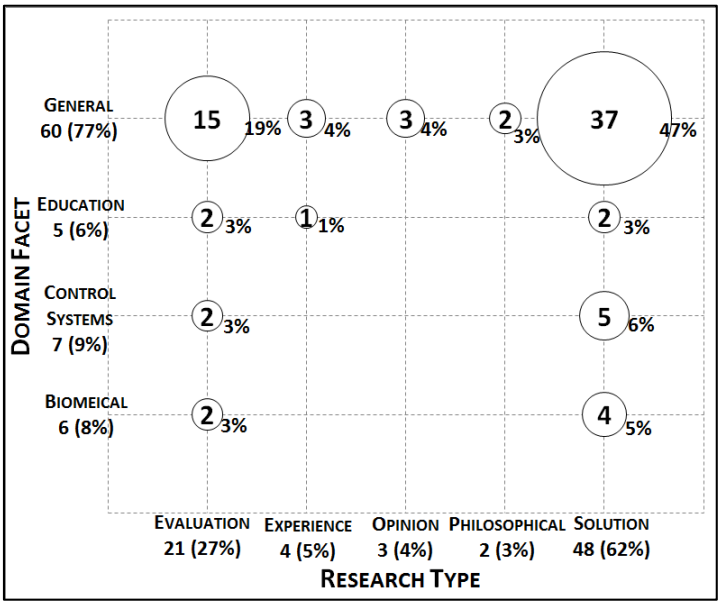


**Figure 4.8 – Systematic mapping of research type facet and domain facet.**

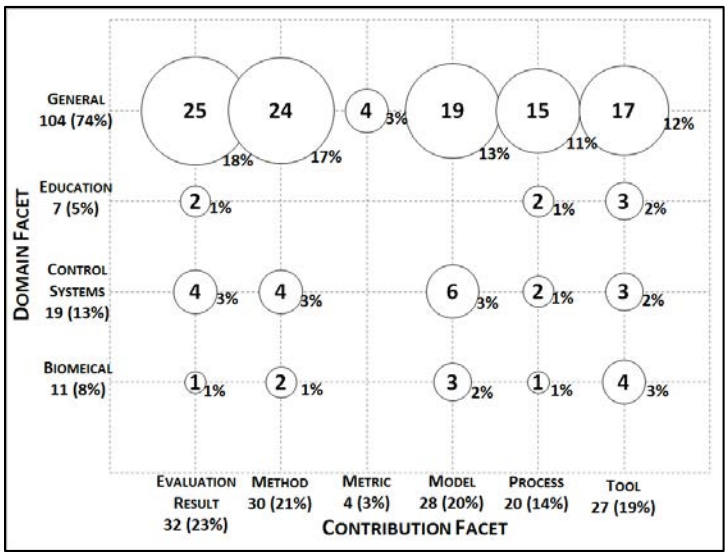


**Figure 4.9 – Systematic mapping of contribution facet and domain facet.**

Figure 4.8 maps the domain of study against the research type. Studies are categorized to a specific domain only if they mention the domain they investigated in the keywords or in the abstract. Otherwise we categorize the study into the general domain. Figure 4.9 presents the systematic mapping of contribution facet against the domain facet.

### *4.1.6 RQ2.6 – Case Study Included*

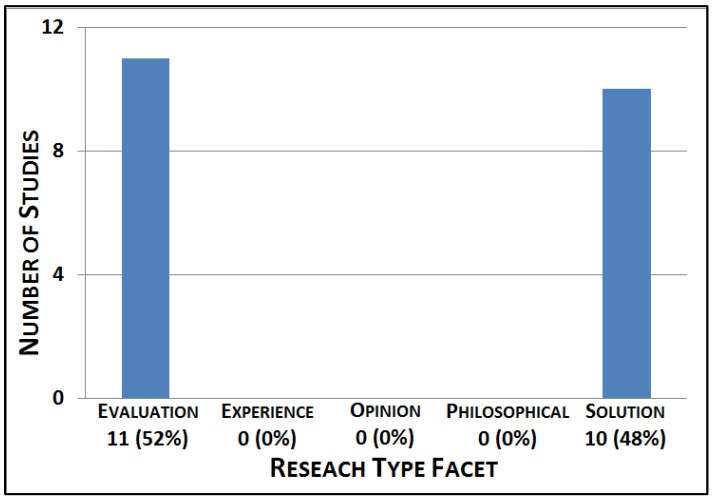


**Figure 4.10 – Research type of studies that included case studies.**

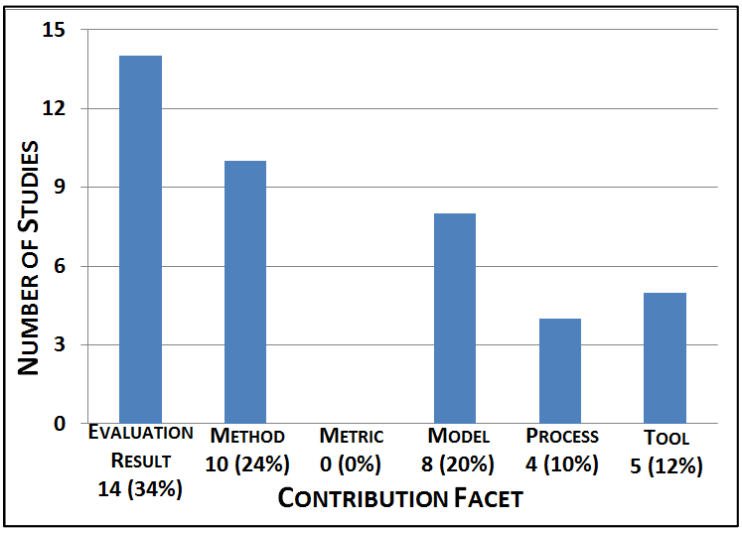


**Figure 4.11 – Contribution facet of papers with case studies**

We use the histogram to present the systematic mapping of case study included facet against the research type facet (Figure 4.10). There are 21 out of 78 studies (27%) offer case studies for the research community to have a better review / understanding of the results or conclusions they provided. In Figure 4.10, case studies appear in evaluation and solution studies in order to support their evaluation result and the solutions they proposed. It is reasonable that the opinion studies and philosophical studies do not have case study included since the opinion studies are studies that express the personal opinion which do not rely on any work of others and the philosophical studies are the "big picture" providers which the case studies are too fine-grained to be discussed. Figure 4.11 presents the contributions made by the papers that included case studies.

## 4.2 Adapted and adopted Agile Practices and Toolsets

In the previous section and in Chapter Three, the systematic mapping results indicate an isolated status of AIESD research field – little cross references among researchers and new adapted methods, small number of studies which evaluate the new solutions and small average number of publications in one venue. Thus we believe it is time to summarize and go beyond the concepts of a systematic mapping study to provide details of proposed testing frameworks, tools and AIESD lifecycles.

### *4.2.1 Testing Frameworks*

- **EMBEDDED-UNIT** – Unit Test Tool for Embedded Software Development. Embedded-Unit is a unit testing tool based on CppUnitLite [28] and UnitTest++ [29] testing frameworks. Modifications have been applied in order to meet the memory and speed

constraints for embedded systems; i.e. this tool can run directly on evaluation boards and product target in addition to simulators. Processor on-chip timers are used to provide assistance for the non-functional testing needed for real-time code. A recovery mechanism allows testing to continue when prototype software / hardware interfaces fail [S10, S48, S55, S63, S64, S67, S69].

- **SYSTIR** – System Test Framework. Systir helps introduce input to a system and compare the output to which is expected in system tests. Systir can bind to libraries of other language providing any features that are needed in system test, and allows definition of a Domain Specific Language to express system tests more easily [S36].
- **UNITY** – Unit Test Framework for C. Unity is a lightweight testing framework for C. It can output test results through standard I/O (stdio), a serial port, or via simulator output [S36].

#### *4.2.2 Embedded-Agile Supporting Tools*

- **C-MOCK** – Mock Function Library for C. C-MOCK is a Ruby-based tool to automate the creation and maintenance of mock functions for unit testing in C language [S36].
- **E-COVER** – Code Coverage Analysis Tool. TRACE Unit is a full code coverage analysis tool. Speed is improved over software code-coverage tools by following the program flow between the tests and code by utilizing the hardware present in current modern processors [17]. This approach has recently been modified for automated requirements traceability [27].
- **E-FIT / E-FITNESSE** – Acceptance Test Cases Generation Tool. This tool is intended to assist customers express their requirements in terms of acceptance tests, and follow successful product development through all five stages of the eXtreme Programming Inspired Embedded-Agile lifecycle. [S7, S67].

- **E-RACE** – Data race identification tool in multi-threaded systems using shared memory. To reduce the chances of false positive and false negatives, E-RACE avoids the overhead of software data race tools by employing the processor's data watch and instruction watch units to monitor memory activity [S68] [26]. Recently an FPGA-based Agile test support (ATS) co-processor has been suggested to provide equivalent support across all families of processors [S12, S70].
- **FUXI** – is described as an Agile Development environment for embedded systems [S76].
- **XEST** – is described as an automated framework for regression testing of embedded software [S51].

#### *4.2.3 Proposed Embedded-Agile Lifecycles*

As can be seen from Table 4.1, Grenning's Embedded TDD lifecycle [S21] provides a detailed overview of moving embedded code from simulation, through prototyping on an evaluation board onto testing the production model. The eXtreme Programming Inspired (XPI) embedded cycle [S67] covers an Embedded-Agile approach to (1) customer acceptance tests [S7], (2) research into, and development of new algorithms using MATLAB [S55] that are moved into (3) simulation before (4) being optimized for real-time operation on the target [S10] [S64] [S67]. Recent extensions have been suggested to allow an Agile approach (5) to the development of FPGA co-processors to handle critical real-time issues or to identify data-races in multi-threaded systems [S12, S70].

A XPI-stage 6 may be needed! Johnson discussed his attempts to apply Agile concepts to IC development [25] at a Calgary Agile Methodologies User Group (CAMUG) informal monthly meeting. It was modifications of XP processes to handle chip development and simulation that

formed the majority of that talk since, as the speaker pointed out, a TDD approach was difficult to justify when the next minimum viable product needed for the next Scrum meeting with the customer would cost millions of dollars!

A key element in using Agile at all stages during embedded system development is the ability to write tests that validate the hardware, even when the hardware does not yet exist. A testable design can be achieved by using a Model, Conductor and Hardware (MCH) design pattern [S35, S36] which isolates the functional logic from hardware.

The next Section, 6.2, is a list of testing frameworks that may assist the developer to take advantage of Embedded-Agile ideas. Section 6.3 provides references to other tools suggested for support of Embedded-Agile development.

**Table 4.1 – Comparison of proposed Embedded-Agile Lifecycles**

| **Embedded TDD (ETDD) lifecycle [S21]** | | | | | **eXtreme Programming Inspired (XPI) Embedded-Agile lifecycle [S7, S10, S12, S55, S64, S66, S67, S70]** | | |
|---|---|---|---|---|---|---|---|
| Stage | USES | PRODUCES | | | Stage | USES | PRODUCES |
| | | | | | XPI Stage 1 Initial interaction with customer | User stories and wish lists | Acceptance tests expressed via FIT and FitNesse tools; Excel spreadsheet |
| | | | | | XPI Stage 2 Discussion with research team | XP1–tests; new algorithm concepts | Unit tests expressed in MATLAB language |
| ETDD-Stage 1 C simulation | Customer wish list | Unit tests running in PC development environment. | LESS FREQUENT - TESTING - MORE FREQUENT | | XPI Stage 3 C Simulation | XP1-tests, XP2-tests | Unit tests running using target's compiler using a target simulator or on evaluation board |
| Stage 2 Compile tests for target | ETDD - tests | Unit tests running using target's compiler on target simulator | | | | | |
| Stage 3 Run test on evaluation board | ETDD1-tests, hardware interface | Additional unit tests associated with hardware interaction | | | XPI Stage 4 Optimization to meet real-time requirements | XP1-tests, XP2-tests, XP3-tests, real time requirements, hardware interface | Unit tests running on evaluation board or target. Optimized C / C++ code or custom assembly code |
| Stage 4 Run tests on target hardware | ETDD1-tests, ETDD3-tests | Tests that run on target hardware | | | | | |
| Stage 5 Manual testing | ETDD1-tests, ETDD3-tests | Tests that run on target hardware with manual intervention | | | | | |
| THE REST OF THIS TABLE IS PRESENTED ON NEXT PAGE | | | | | | | |

| | | | |
|---|---|---|---|
| CONTINUE TO TABLE 4.1 | | | |
| | XPI Stage 5 Movement to co-processor to meet real-time requirements | XPI 1 - tests, XPI 2 - tests, XPI 3 - tests, XPI 4 - tests | Unit tests mocking the co-processor in software. Unit tests with actual co-processor implemented with mocked and actual busses [S12, S70] |
| | XPI Stage 6 IC development | Proposed in [25] | |

### 4.3 Discussion

Regarding to our RQ2.1, there are a large number / percentage of solution papers proposed, according to Figure 4.1. This implies two things: (1) researchers are very enthusiastic to propose new method / solutions to better apply Agile methods onto embedded software development; (2) some Agile methods should be used with special care, modifications or new methods and tools. We expect the solution papers will keep growing in the future but conducting evaluation, opinion or experience researches on the new solutions is a research area that still requires more attention in order to synchronize the current research results. Philosophical studies, as a type of secondary study, are highly encouraged since it reviews and summarizes a number of papers in the field to provide the "big picture".

For contribution facet in RQ2.2, many solutions-related attributes (e.g. tools, processes, methods and models) have high percentage contribution according to Figure 4.2. However, we again feel the need to summarize them so as to render the AIESD community better toolsets and

practices. New metric methods will encourage evaluations on new solutions. According to Figure 4.4, solution papers contribute the most (72%) to the community, and therefore, solution papers are most welcomed and suggested to be reviewed for new ways of applying Agile onto embedded software development.

Adapted Agile methods / TDD used papers are discussed in RQ2.3. Again, solution papers contribute the most in these two categories. We believe the reason that TDD is favored for embedded software development is because: (1) TDD is supposed or proven to be a defect-reduction practice (Maximilien and Williams 2003; Beck and Andres 2004; Bhat and Nagappan 2006; Madeyski and Szała 2007; Nagappan et al. 2008); (2) TDD is a practical programming practice that can be actually followed and performed. However, studies reported (Huang et al. 2008; Smith et al. 2013) that the lack of efficient tools impedes the embedded software development to become agile. Adapted Agile methods, on the other hand, are proposed to tailor the Agile methods for better use. Evaluation papers are needed on the adapted methods, according to Figure 4.6.

RQ2.4, which investigates the tool functionalities presented in the included studies. According to Figure 4.7, tools on unit testing have the largest percentage. In this research question, we found that there is no tool on refactoring, which is a main practice of TDD. We believe it is because all the major development environments (CrossCore® Embedded Studio (Analog Devices 2013a), Code Composer Studio (Texas Instruments 2013a), Atmel Studio (Atmel 2013) and VisualDSP++ (Analog Devices 2013b) have already possessed a refactor feature. However, according to Smith et al. (Smith et al. 2009; Smith et al. 2013), refactoring in AIESD life-cycle has a different meaning – transferring verified, high-level (platform independent) and sometimes slow code to a faster or cheaper version on a specific processor.

RQ2.5 categorizes the pool of studies into various domains. The missing bubbles in Figure 4.8 and 4.9 (e.g., opinion papers for control system, biomedical and education domain in Figure 4.8) indicate the needed researches for the uses of research / industrial purpose. This research question groups studies in the same domain for the use of people who interested in one specific AIESD domain.

RQ2.6 presents the statistical results of papers that included case studies on research type facet and contribution facet. One interesting finding from Figure 4.11 is there is no paper proposes metric in these case study included papers. New metric methods seem to be needed with a case study to validate its metric method.

By going beyond the scope of a standard systematic mapping study, testing frameworks and supporting tools from the pool of studies are presented in Section 7.1 and 7.2. Section 7.3 presents the comparison of two adapted Embedded-Agile lifecycle from two authors, Smith and Grenning who have the most number of publications in the field of AIESD, according to RQ1.5. Smith's XPI embedded life cycle captures the whole embedded software development from developing requirements with customers in the form of acceptance tests to transferring memory or time consuming C/C++ code from embedded processors to task-specific co-processor or ASIC chips for speed or lower cost. Whereas, Grenning's embedded TDD lifecycle provides more detail in handling XPI stage 3 and 4, which are product development stages with emulator / actual hardware. By extracting the elites that can be applied from the pool of studies, we believe that AIESD community can gain benefits of the current available research.

This systematic mapping study with results presented in Chapter Three and this chapter indicates that the AIESD research area is a hot topic but with little synchronization and summarization. By classifying the pool of studies into different categories, the characteristics of

AIESD field can be identified. Solution-proposing paper are favored and making more contributions. However, evaluation, opinion and experience papers on these new solutions are needed. Such areas are examples of a guideline to assist researchers in conducting future research in this area, which is one of the main objectives of this study.

### 4.4 Chapter Summary

To answer the research questions in RQ2 presented in last chapter, the research type, contribution, adapted Agile methods / TDD used, tool functionalities, domain of study, whether case studies are included and article type of the paper were shown via systematic mapping results. Furthermore, by going beyond the scope of systematic mapping, the adapted and adopted Agile methods are presented and discussed with three aspects: testing framework, support tools proposed and embedded Agile lifecycle. Discussions and possible future research guidelines were offered at the end of this chapter. The Agile test support co-processor which is designed to improve testing practices is introduced in next chapter.

# CHAPTER FIVE: AGILE TEST SUPPORT COPROCESSOR DESIGN[1]

Details of testing frameworks, Agile embedded support tools as well as Agile embedded lifecycles which are tailored for the use in context of Agile embedded were introduced in the last chapter. In this chapter, the concepts of hardware-assisted test support tools that make use of hardware features present on embedded processors are first explored. An Agile Test Support (ATS) FPGA co-processor is proposed to provide low overhead hardware-assisted test insertion capabilities for testing frameworks used with test-first (Agile) and test-last hardware-software co-design philosophies. The detailed ATS FPGA co-processor hardware decisions, features, performance analysis and a demonstration of the use of ATS co-processor are presented in Chapter Six.

## 5.1 Introduction

A key Agile element to improve software quality is the repeated use of tests as the project moves through several development stages as suggested in (Beck 2003; Smith et al. 2009a). This concept is demonstrated in Figure 5.1 (Smith et al. 2009a) with the generation of appropriate FIT (Cunningham 2002) or FitNesse drivers (FitNesse 2003) within the 6 stages eXtreme

---

[1]This chapter is derived from two following papers:

**Deng, D**., Smith, M., Islam, S., & Miller, J. (2013, June). Prototype test insertion co-processor for agile development in multi-threaded embedded environments. In *Signals and Systems Conference (ISSC 2013), 24th IET Irish* (pp. 1-8). IET

Smith, M., **Deng, D**., Islam, S., & Miller, J. (2013). Enhancing Hardware Assisted Test Insertion Capabilities on Embedded Processors using an FPGA-based Agile Test Support Co-processor. *Journal of Signal Processing Systems*, 1-14.

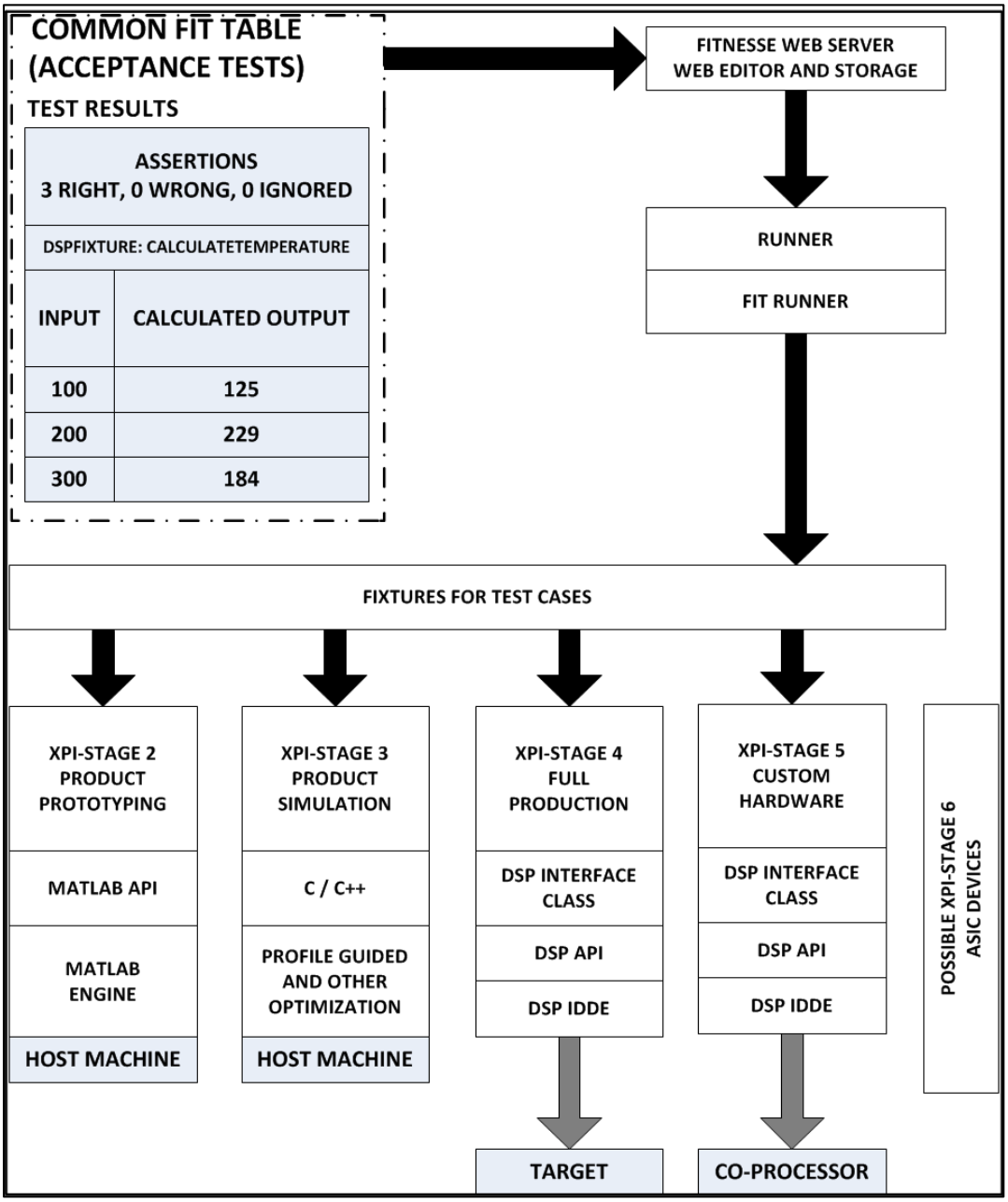


**Figure 5.1 – Schematic of the levels of a FIT test framework necessary within an embedded environment for validating proposed customer acceptance tests (XPI-level 1) through XPI-stages 2 to 5 (Smith et al. 2009a) and a ASIC Stage 6 proposed by Johnston (Johnson 2011). Equivalent layers of unit tests are needed by the developer at later stages of development. Modified from (Smith et al. 2009a).**

Programming Inspired (XPI) lifecycle used within our group. Using this approach it is possible to demonstrate to customers that their initial acceptance tests developed in XPI stage 1 are satisfied by all lifecycle stages as the code moves across a variety of development environments. A similar approach is taken to ensure that new unit tests developed at any one XPI-Stage can be re-used for regression testing during later stages.

This requires automated testing frameworks to effectively and efficiently support the developer in executing test cases. Unlike enterprise development environments, embedded integrated design and development environments (IDDE) provide easy access to a processor's on-chip low level specialized hardware features. Such features can be co-opted to provide effective and efficient support for testing. Moreover, these features can result in a lower overhead when compared to the equivalent software instrumented approaches to testing (Huang et al. 2008; Tran et al. 2008; Smith et al. 2010). Unfortunately, not all embedded processors have the desirable on-chip features to support these hardware-supported testing framework advances.

To provide effective and efficient testing support, we propose a FPGA-based, 'Agile Test Support' (ATS) co-processor to provide test insertion capabilities in this chapter. As a hardware-assisted testing method, the ATS co-processor is intended to be universally applicable to embedded processors and impose less performance loss. In this thesis, we imagine the ATS test insertion co-processor is being used in a multi-threaded environment to provide low-overhead dynamic race condition detection capability.

This chapter is organized as follows. Section 2 presents the existed hardware-assisted testing support methods. Section 3 outlines the advantages of an ATS co-processor while the design of the ATS is presented in Section 4. Section 5 summarizes this chapter.

## 5.2 Existed Hardware-Assisted Testing Techniques

In this section, the existed hardware-assisted testing methods are examined. We outline (A) how low-level debug capabilities could be co-opted to work in a dynamic fashion to support both embedded system test-first (Agile) and test-last development together with (B) examples of how current processor architectures limit the actual implementation of these ideas.

### *5.2.1 Hardware-Assisted Test-Vector Insertion*

Test vector (necessary variable configuration) are used to reach and validate an infrequently activated, but critical, software conditional branch. But it may take an enterprise system developer a considerable amount of time to set up the test vectors. By comparison, test-vectors can be inserted directly by the embedded developer into required variables stored in registers or memory locations using the boundary scan architecture (IEEE 1149.1 (JTAG) standard) (Whetsel 1997) present on most modern embedded processors (Rajsuman 1999).

### *5.2.2 Hardware-Assisted Code Coverage*

Code coverage can (A) identify whether current tests adequately exercise the code associated with all branches of a conditional statement (Chen et al. 2001; Graaf et al. 2003) or (B) generate a traceability mapping to match code and tests with system requirements (Murray et al. 2002). While code coverage can be tackled using software approaches, Shey et al. (Shye et al. 2005) and other authors have suggested that there are advantages in using for hardware-assisted code coverage in an embedded environment.

Most processors would support a timer interrupt to record changes in the processor's program counter as tests access the developer's code, which would can be used to obtain the

code coverage report. To avoid degrading the hardware-software development environment, post-run analysis would be used to map the program flow information back into specific code lines within the tests. However, only sparse coverage is possible with this approach since frequent timer interrupts would place a heavy overhead on any real-time aspects under test (Shye et al. 2005).

Some processors possess a trace unit designed to re-create the program sequencer's recent path as a debug break point set in the code by the embedded system developer, e.g. last N ≈8 program jumps. Shey et al. (Shye et al. 2005) proposed sampling the trace unit's buffer rather than the program counter to lower overhead, but found that adequate coverage was only obtained by aggregating coverage across multiple runs. The increased overall downtime incurred by the developer makes this approach inappropriate for the frequent testing required for Agile development.

The trace units for the Analog Devices Blackfin (ADSPBF5XX) (Tran et al. 2008) family of processors offer another approach to providing hardware assisted code coverage. This trace unit invokes an exception handler to spill the processor's internal program-flow buffer to external memory each time the buffer overflows. This trace unit's loop compression options also allow repeated program flow changes to be ignored. Such a combined approach allows a more accurate program flow analysis with a lower overhead than the sparse sampling approach possible with an external timer interrupt (Tran et al. 2008). This trace buffer capability has been incorporated into *AutoETF*, an automated embedded traceability framework (Wiederseiner et al. 2011).

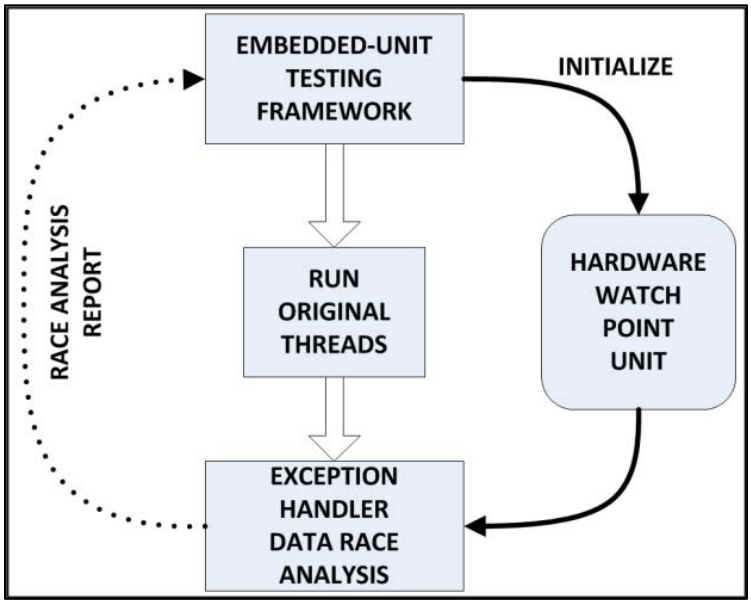


**Figure 5.2 – E-RACE's hardware approach of data race detection avoids extensive software instrumentation. Modified from (Huang et al. 2008).**

### *5.2.3 Hardware-Assisted Test Insertion*

When new threads are introduced into an existing embedded system or integrating the legacy, off-the-shelf code into the multi-threaded system, there is a potential for data races to be introduced if developers fail to introduce the proper synchronization to allow multiple threads to concurrently attempt to read and write to a shared memory location.

The Blackfin embedded processor family debug instruction and data watch units were designed to provide hardware breakpoint support during code debugging (Huang et al. 2008; Smith et al. 2010). Huang et al. (Huang et al. 2008) proposed the E-RACE tool, Figure 5.2, which co-opted the instruction and data watch units to respectively support time stamping of specific program activity to identify threads not meeting hard-time constraints or improperly accessing shared memory locations. When compared to software test insertion approaches, e.g. (Flanagan and Freund 2001), this approach offers two advantages:

(A) There is no need to change / recompile existing legacy or third-party delivered software object files to insert tests; and

(B) The instruction watch unit incurs zero-overhead in absence of the specified instruction data bus activity, and only a 19 cycle overhead (40 ns) when the unit's exception handler is activated to insert a test (Huang et al. 2008; Smith et al. 2010).

Given the low-overhead of test insertion found with the Blackfin's instruction watch unit, Huang et al. (Huang et al. 2008) anticipated a similar performance with the debug data watch unit when used to identify possible data races. However, the testing would become inherently less efficient as this data watch unit could only monitor single memory location or contiguous block in contrast with the Blackfin instruction watch unit's capability of monitoring activities occurring at several instruction locations or within several contiguous code blocks.

Compounding the test-insertion issues, the data watch unit switches the processor into an emulation mode upon recognizing the specified memory access. This activates the IDDE running on an external PC and forces the processor to stop. Using scripts to automate this manual mode of debugging led to individual test insertion times of 200,000+ cycles, approximately 1/ 30 s, inappropriate for the low overhead testing framework needed during Agile-guided embedded development (Smith et al. 2010).

We have indicated how specialized processor features can be co-opted to simplify Agile or test-last oriented embedded system development. However, the implementation of these features is inconsistent across processors and manufacturers. In the next section we detail the advantages and hardware concepts of an automated test insertion ATS co-processor to overcome these problems.

### 5.3 ATS Co-processor's Advantages and Hardware Concepts

Although we have outlined the advantages of utilizing the hardware features of embedded processors to insert tests, these features are not present across various families of processors or the overhead of using these existed hardware capabilities are unacceptable. We propose an Agile test support (ATS) co-processor to provide a low-overhead hardware-assisted test insertion mechanism upon recognizing specified data bus activity, which is universally adaptable to different families of processors. This ATS co-processor is intended to add to, or improve, an embedded processor's ability to support either (1) an automated testing framework suitable for an Agile test driven (test-first) development; or, (2) a framework for more conventional Waterfall or V-shaped (test-last) lifecycles. Even though the concepts of hardware-assisted testing support are general, we specifically discuss an ATS system to enable a developer to take a pro-active approach to software testing in multithreaded environments in order to identify possible race conditions.

A race condition is said to occur if two or more threads simultaneously access a shared resource without having proper synchronization techniques, where at least one of the threads is writing to the shared resource (Netzer and Miller 1992). Software race condition detection includes two main categories in general: static race condition detection and dynamic race condition detection. Static race condition detection methods analyze the whole program and examine about all the possible scheduling among threads, but they are costly in time and space and false alarm would be reported (Flanagan and Freund 2009; Engler and Ashcraft 2003; O'Callahan and Choi 2003), which limits the static race condition detection methods' applicability. Dynamic race condition detection methods (Pozniansky and Schuster 2003; Pozniansky and Schuster 2007; Yu, Rodeheffer, and Chen 2005), based on Lamport's *happens-*

*before* algorithm (Lamport 1978). These dynamic methods are limited by performance issues because they have to record information of each thread in a system and can only check a set of specific inputs according to (Flanagan and Freund 2009). Static race condition detection methods and dynamic race condition detection methods can be mutually complementary to each other in order to generate a more accurate race detection report.

Dynamic race condition detection methods, however, would be probably not suitable to be applied in embedded context because they have to instrument the C++ source in order to monitor all accesses to each of the shared resources (Pozniansky and Schuster 2003; Pozniansky and Schuster 2007). The instrumented C++ code would possibly impair the real-time characteristics and increased-size of code would exceed the available memory possessed in the embedded system. ATS test insertion co-processor, on the other hand, can be incorporated with these dynamic race detection methods since it provides the embedded processor low overhead test insertion capabilities to monitor the accesses of the shared resources.

We propose that incorporating such a co-processor within an Agile framework (1) could prevent the introduction of data races into newly developed multithreaded systems; and, (2) more easily identify possible data races in legacy or third-party systems formed by combining a number of existing threads or multi-threaded systems together. Test insertion methods provided by the system should not change the result of data race analysis; i.e. neither introducing new data races (false positives) nor masking existing data races (false negatives).

The intention behind the use of an ATS is to provide the developer the ability to insert a test following a specified activity, e.g. access to a shared memory location (specified by address bus activity) or execution of an over-delayed thread that is subject to a hard-time constraint (specified instruction address bus activity). Extending the ideas of Huang et al. (Huang et al.

2008), the co-processor should be able to insert a test without the need for the developers / race condition detection tools to modify the original code. We suggest that this requirement can be met by giving the co-processor the capability of monitoring and capturing instruction, data and control bus activities from the processor. To achieve best performance, test insertion mechanisms should work concurrently with, and require minimal participation from, the processor. This means there is no cost until a specified data / instruction activity is recognized and the embedded processor has to respond to the activity.

Figure 5.3 schematically demonstrates how an ATS coprocessor could be used monitor a processor's data memory (DM) and the program memory (PM) address busses. On recognizing the activity, the ATS will either use the processor's non-maskable interrupt (NMI) line to activate an interrupt service routine (ISR) allowing the testing framework to insert a test (assert) immediately or ignore this particular bus activity and insert a test at a later time.

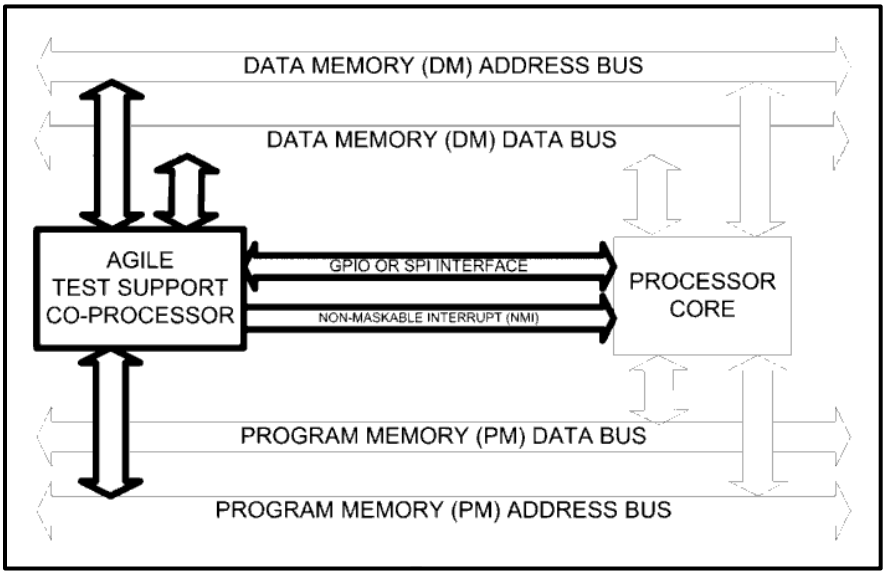


**Figure 5.3 – The ATS co-processor monitors activity on the processor's PM and DM address buses. The co-processor logic activates the processor's NMI line upon recognizing a designated activity.**

## 5.4 Expression of Hardware Design Requirement

In this section, we detail an extension of the XPI embedded Agile lifecycle (Miller et al. 2006; Smith et al. 2009a; Smith et al. 2009b) to guide the design and the implementation of an 'Agile test support' (ATS) co-processor. In the following sub-sections, explanation and demonstration of the design and implementation of the ATS co-processor utilizing an extension of the Agile XPI embedded lifecycle are presented.

### *5.4.1 Software Concepts*

Figure 5.4 is the screen capture of a test designed to express the hardware design requirements of the ATS co-processor. The test is expressed in the syntax of the EmbeddedUnit testing framework (Smith et al. 2005; Smith et al. 2009a; Smith et al. 2009b). The Figure 5.5 screen capture shows how the test in Figure 5.4 may be utilized to validate the ATS coprocessor operation during later stages of its development from software mockup to final hardware implementation. A simplified version of an inserted test is shown in Figure 5.5, Lines 141–143. The full version of the inserted test would be designed to check, *"Does the currently running thread hold the correct locks when accessing a specified memory location?"*

This stage corresponds to development associated with stages 3 and 4 of the standard embedded XPI lifecycle extended to handle hardware-software co-design. Setting the global test parameter *MODE=MOCK_ALL* causes the testing framework to support the initial mocking (simulating) of the ATS co-processor functionality. This implies that all features of this hardware-software co-design are implemented in software running on the embedded processor. By mocking functionality of the ATS co-processor, we can verify the correctness of the

requirements of the ATS co-processor (functionality) together with developing tests that can be used in refactoring the simulated ATS co-processor into hardware.

In the EmbeddedUnit TEST illustrated in Figure 5.4, the ATS is first initialized (Figure 5.4, Line 45) to recognize data memory write, but not read, activities. The ATS data watch unit is then configured (Line 46) to trigger on the second write access of the 6th element in the *testArray[ ]*. This will allow a test of whether the ATS is able to respond to, or ignore, specific data memory operations. With initialization complete, the ATS is activated (Line 47).

The ATS unit must both allow the memory operation to proceed normally and respond to a request for test insertion on a particular memory access. This capability is demonstrated when the *testArray[index]* value is read (Line 53), then modified and finally written back (Line 56). Lines 54 to 58 include EmbeddedUnit test framework assertions (*CHECK*) used to determine if the ATS logic is incorrectly inserting tests during memory (read and write) operations.

Lines 56–61 check that test insertion occurred only during the second write access to the element *testArray[6]*. The ATS co-processor is de-activated (Line 64) to allow EmbeddedUnit assertions (Lines 65–66) to verify that the required write operation to *testArray[6]* occurred without the ATS unintentionally inserting tests as this monitored location is accessed.

```
20 #define EMBEDDEDUNIT_LITE
21 #include <EmbeddedUnit/EmbeddedUnit.h>
22
23 #define FIND_SECOND_ACCESS 2
24
25 enum mode_type {UNKNOWN, MOCK_ALL, MOCK_BUSSES, TRUE_ATS};
26 volatile mode_type MODE = UNKNOWN;   // Check ATS variants by modifying MODE during TEST( )
27
28 TEST_CONTROL (ATScoprocessor_Test_cpp);
29
30 volatile int indicateTestInserted = 0;
31
32 TEST (ATScoprocessor_Monitoring_WriteAccesses)
33 {
34     // Test ATS prototype with mocked processor busses
35     MODE = MOCK_BUSSES;
36     int testArray[N] = {0};
37     int expectedFinal[N] = {0};
38
39     // Set testArray and expected values of result
40     for (int j = 0; j < N; j++) {
41         testArray[j] = 2 * j;
42         expectedFinal[j] = 2 * j + 2;
43     }
44
45     ATS_Reset(MODE, IDENTIFY_WRITES);
46     ATS_WatchedAddress(MODE, testArray, 6 /* arrayIndex */, FIND_SECOND_ACCESS);
47     ATS_Activate(MODE);
48
49     indicateTestInserted = 0;
50
51     for (int memoryAccessed = 1; memoryAccessed < 5; memoryAccessed++) {
52         for (int index = 0; index < N; index++) {
53             int temp = Mock_ReadMemory(MODE, testArray, index);
54             CHECK(indicateTestInserted == 0);   /* Ignore reads */
55
56             Mock_WriteMemory(MODE, testArray, index, temp + 2); /* Write new result */
57             if (memoryAccessed != 2)
58                 CHECK (indicateTestInserted == 0);
59             else if (index != 6)    /* Memory accesses not watched */
60                 CHECK (indicateTestInserted == 0);
61             else CHECK (indicateTestInserted == 1); /* testArray[6] access watched */
62
63             indicateTestInserted = 0;
64             ATS_Deactivate(MODE);
65             CHECK (expectedFinal[index] == testArray[index]);
66             CHECK (indicateTestInserted == 0);
67             ATS_Activate(MODE);
68         }
69     }
70 }
71
72 TEST_FILE_RUN_NOTIFICATION(ATScoprocessor_Test_cpp);
```

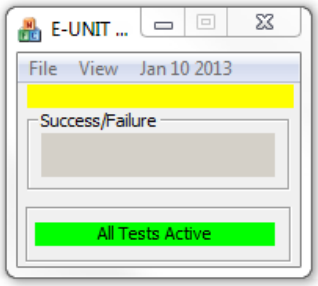


**Figure 5.4 – The code in this screen capture is a simplified version of a test that uses the ATS co-processor to verify whether a thread holds the correct locks when accessing a specified memory location. The test in red is expressed in the syntax of the EmbeddedUnit testing framework (Smith et al. 2009a; Smith et al. 2009b; Smith et al. 2005). The code highlighted in yellow are the ATS co-processor related function calls.**

```
118
119 // ATS co-processor mocking
120 int Mock_ReadMemory(int whichATSmode, int array[], int index) {
121     int temp = array[index];     // Perform memory read on monitor address
122
123     switch (whichATSmode) {
124         case MOCK_ALL:  // Send data memory address to mock ATS
125             Mock_MemoryAddressBus_Software(&array[index]);
126         break;
127
128         case MOCK_BUSSES:   // Send data memory address to actual ATS via GPIO lines
129             MOCK_MemoryAddressBus_GPIO(&array[index]);
130         break;
131
132         case TRUE_ATS;  // True ATS monitored memory access directly (Line 121)
133         break;
134     }
135     return temp;
136 }
137
138 #include <sys/exception.h>
139
140 #pragma always_inline
141 inline void InsertedTest(void) {
142     indicateTestInserted++;
143 }
144
145 // Cause NMI via software or hardware interrupt
146 EX_NMI_HANDLER(ATS_coprocessor_NMISR) {
147     // ATS co-processor reset the watch counter automatically
148     ATS_Deactivate(MODE);
149     InsertedTest();
150     ATS_Activate(MODE);
151 }
```

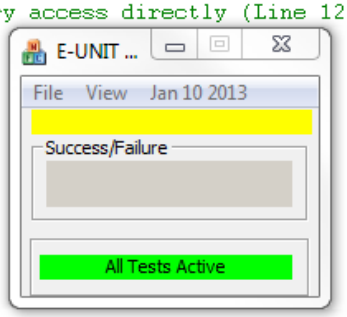


**Figure 5.5 – A screen capture of the code supporting mocked read and write memory operations monitored by the ATS co-processor and the format of the non-maskable interrupt code activated via software interrupt (*whichATSmode* = *MOCK_ALL*) or hardware (*whichATSmode* = *MOCK_BUSSES, TRUE_ATS*).**

### *5.4.2 Monitoring Bus Activity*

Figure 5.5 illustrates the code necessary to support mocking of read and write memory operations. The ATS co-processor monitors the memory access operation associated with element *array[index]*, Line 121. The code of the non-maskable interrupt service routine (ISR) to insert tests is presented in (Lines 146 to 151). The ISR can be activated within the simulated ATS by software interrupt within the function being tested, and through a true NMI during the hardware mocking stage and final testing.

By setting the global test parameter *MODE* = *MOCK_BUSSES* in Figure 5.4, the tests in Figure 5.4 can now be used as a starting point to refactor the verified action of the simulated

version of the ATS co-processor into FPGA hardware, which is the new XPI-stage 5. Ideally, each FPGA component would be implemented in hardware and the equivalent simulated ATS components are removed. This hardware refactoring would be verified by running the test in Figure 5.4.

With the test *MODE* set to *TRUE_ATS*, the memory read in Line 121 will actually be monitored by the ATS. For the other test modes, information about the read address must be sent to an ATS mocked in software (Line 125) or sent to the actual ATS logic over the processor's GPIO interface (line 129).

### *5.4.3 Anticipated Agile Test Support Software Overhead Per Test*

As shown in Figure 5.5, the probable overhead of servicing an implemented ATS co-processor's data watch unit is around 60–80 cycles / test insertion, since each of the three function calls takes approximately 20 cycles and the disruption of processor's pipeline by the interrupt. Although it is lower than the 200,000 cycles / test insertion for the current Blackfin data watch unit (Smith et al. 2010), the overhead is still three to four times that of the Blackfin instruction watch unit's 19 cycles / test (Huang et al. 2008; Smith et al. 2010). Setting the compiler to optimize the code, and in-lining the function calls as demonstrated in lines 140–143 (Figure 5.5), removes these pipeline disruptions and implements the ATS de-activate function (Line 148) and ATS activate function (Line 150) in the ATS co-processor, giving an ATS anticipated overhead (23 cycles) closer to the 'ideal' performance of the Blackfin instruction watch unit.

It can be anticipated that the inserted test may involve accessing the monitored array element (simulated shared memory). This would lead to possible nested NMI interrupts which must be prevented by deactivating (Line 148) and later reactivating (Line 150) the ATS. This potential

additional overhead can be avoided by implementing this functionality as part of our proposed FPGA based custom, ATS architecture.

An approach for the further reduction in test insertion overhead is mocked in Figure 5.6. This incorporates the loop-overhead reduction capability present in the Analog Devices BF533 processor's trace unit discussed in Section 2.2 of this chapter into the ATS watch unit design. Lines 201 to 204 demonstrate the continued access of a shared memory location in a loop of size *LOOPTIMES*. The presence or absence of a valid memory lock will be known after the first shared memory access; the other (*LOOPTIMES – 1*) inserted tests are unnecessary overhead.

In this mocked ATS functionality, we register (Line 196) a new NMI ISR handler (Lines 207–211) to replace the standard NMI handler (Figure 5.5, Lines 146–151). The first time around the loop accessing the shared memory (Figure 5.6, Line 203) would cause the ATS co-processor to force the embedded processor to execute this new ISR handler (Line 207–211). Since the co-processor is disabled (Line 208) in the NMI handler, only one data race detection test is run for the whole loop.

```
191 #define FIND_FIRST_ACCESS 1
192
193 // ATS coprocessor activated for low overhead test-insertion during tight loops
194 void TightLoopAccess() {
195     // register the tight loop interrupt handler
196     ATS_Reset(MODE, IDENTIFY_WRITES_TIGHT_LOOP);
197     ATS_WatchedAddress(MODE, array, 6 /* arrayIndex */, FIND_FIRST_ACCESS);
198     ATS_Activate(MODE);
199
200     // Loop involving multiple accesses to same shared memory location
201     for (int i=0; i<LOOPTIMES; i++){
202         // Accesses array[6] many times
203         AccessAllElementsOfArray(array);
204     }
205 }
206
207 EX_NMI_HANDLER(ATS_coprocessor_TIGHT_LOOP) {
208     ATS_Deactivate(MODE);
209     InsertedTest();
210     // No need to re-activate at this point
211 }
212
213
214
```

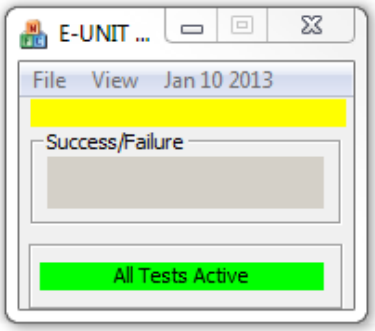


**Figure 5.6 – A screen shot of ATS co-processor reducing testing overhead by deactivating itself following the first access of shared memory within a tight loop.**

### *5.4.4 Developing "Just Enough" Hardware to Satisfy the Co-Design Test*

A standard approach with Agile software development is to choose an item (requirement) from the customer's wish list, write a test to demonstrate when that requirement has been met, and then write just enough software to satisfy the test. We proposed taking an equivalent approach during Agile embedded hardware-software co-design (XPI-Stage 5) by developing just enough FPGA logic to provide the needed functionality to satisfy the tests of Figures 5.4 to 5.6.

Figure 5.7 captures the essence of the logic of the ATS coprocessor. The requirements for the proposed hardware interface operations indicated by the ATS functions are described through the tests and code shown in Figures 5.4 to 5.6. The ATS *CONTROL* register requires the capability of resetting the co-processor, enabling / disabling the program and memory watch activity. This FPGA block must also have functionality to enable / disable the sending of the

NMI signal to the core; and to be able to watch a single instruction / data location or watch contiguous instruction / data block.

Multiple ATS data watch units are required in order to monitor the shared memory accesses of multiple threads. Three additional registers will be required to implement each unit with the following functionality. With the ATS activated, the *WATCH_COUNTER* decrements each time the processor address bus value matches the *WATCH_ADDRESS* value. On reaching a zero value, an NMI signal is generated and the *WATCH_COUNTER* is automatically re-loaded with the value of the *WATCH_PERIOD*. This reduces the ATS testing insertion overhead below that of the existing Blackfin instruction watch units which must be reset by the processor to permit further watch operations. Tran and Wiederseiner (Tran et al. 2008; Wiederseiner et al. 2011) further suggested that the program memory (PM) address bus values are also fed into the *PROGRAM_FLOW* unit where changes in the program flow are recognized and stored in a trace buffer for later use.

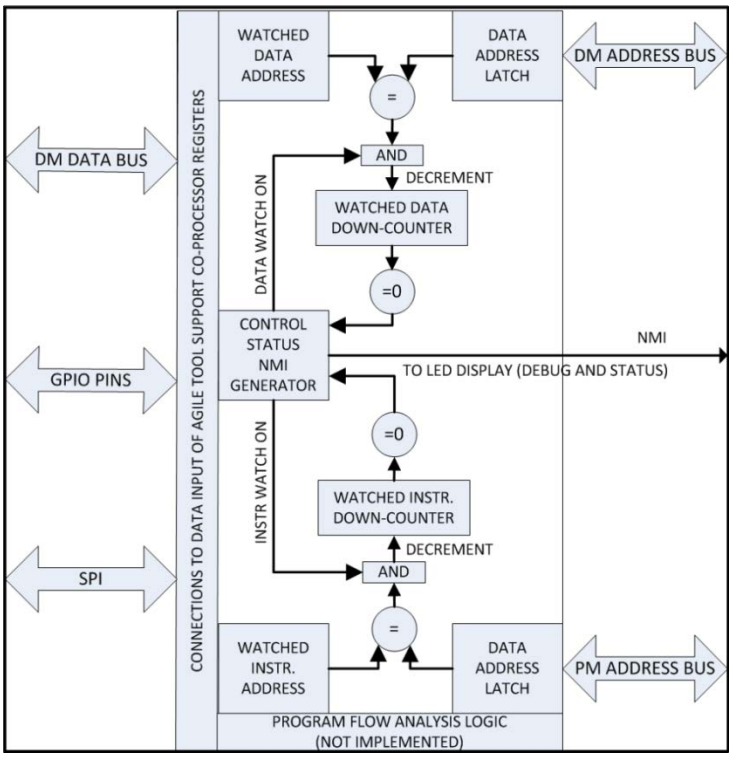


**Figure 5.7 – This schematic shows the development of just enough FPGA logic to satisfy the ATS co-processor tests shown in Figures 5.4 to 5.6.**

### 5.5 Chapter Summary

In this chapter, we first outlined advantages and limitations of utilizing hardware features existed in modern embedded processors. The ATS co-processor was then proposed to provide low-overhead hardware assisted test insertion mechanism upon recognizing specified bus activity. Guided by XPI embedded lifecycle, we demonstrated the process and the details of developing "just enough" ATS co-processor hardware logic along with satisfying tests developed in earlier stages. The performance was expected to be orders of magnitudes lower than that for the Blackfin's existing data watch unit and other possible performance improvement approaches are discussed. Next chapter details the ATS co-processor prototype's implementation and performance analysis.

# CHAPTER SIX: AGILE TEST CO-PROCESSOR PROTOTYPE AND PERFORMANCE ANALYSIS[1]

In this chapter we provide the specifics of the hardware implementation of the ATS co-processor discussed in Chapter Five. Performance analysis and comparison with existing hardware-assisted test insertion methods is provided. Possible ATS implementation on a dual-core embedded processor is also discussed.

## 6.1 ATS Hardware Design Decisions

The test environment for the ATS prototype was a Blackfin FPGA EZ-Extender (Analog Devices 2012) with a Xilinx Spartan 3 FPGA (Xilinx 2013a) mounted on an ADSP-BF533 EZ-KIT Lite Evaluation System (Analog Devices 2004) with direct access to the address bus, data bus, and interface signals of an Analog Devices ADSP-BF533 embedded processor (Analog Devices 2011a). The ADSP-BF533 was chosen as this processor already possesses hardware to support both high speed program flow logic (trace-buffer) and instruction watch capability; but lacks a suitable data watch capability. Thus the environment allows for a straight-forward performance comparison with the ATS co-processor. Original code development and testing was performed using the Analog Devices Visual DSP+V5.0 (VDSP) integrated designed and

---

[1]This chapter is derived from two following papers:

**Deng, D**., Smith, M., Islam, S., & Miller, J. (2013, June). Prototype test insertion co-processor for agile development in multi-threaded embedded environments. In *Signals and Systems Conference (ISSC 2013), 24th IET Irish* (pp. 1-8). IET

Smith, M., **Deng, D**., Islam, S., & Miller, J. (2013). Enhancing Hardware Assisted Test Insertion Capabilities on Embedded Processors using an FPGA-based Agile Test Support Co-processor. *Journal of Signal Processing Systems*, 1-14.

development environment (IDDE) (Analog Devices 2013b) with Visual DSP++ Kernel (VDK) (Analog Devices 2006), ISE WebPACK Design Software (Xilinx 2013b) and EmbeddedUnit automated testing framework (Smith et al. 2005; Smith et al. 2009a; Smith et al. 2009b). We have also demonstrated an ATS co-processor application on the new IDDE CrossCore Emebdded Studio (CCES) (Analog Devices 2013a) with µC/OS-III (Micriµm 2010) operating system using EmbeddedUnit automated testing framework.

While details may change across families of processors, the goals, design decisions and limitations surrounding the development of an external ATS FPGA system are generally applicable. The majority of the ATS logic shown schematically in Figure 5.7 can be implemented in a generic fashion within an FPGA using standard registers, counters and comparators.

Figure 6.1 demonstrates how the ATS co-processor, Figure 5.7, was interfaced to the Blackfin ADSP-BF5XX processor system's bus interface. Unfortunately this limits the ATS to monitoring only off-chip memory accesses in contrast to the on-chip memory (L1) and program memory monitoring capability of the Blackfin's instruction and data watch unit. This is not a serious limitation since the Blackfin internal core memory is relatively small in common with many embedded processors. This means that the user code and data will typically reside in the SDRAM and only the operating system code will be placed in L1 memory. In addition, some processors will put a dummy address request to external memory in parallel with the cache access to speed-up the system's response if a data or code cache miss occurs. The associated address bus signals will be very short in duration if a cache hit occurs. It would be processor dependent whether this transient signal would pass beyond the SDRAM controller shown in Fig. 8 and become recognized by the ATS co-processor logic.

With this configuration, we were able to synchronize the ATS instruction and data address capture blocks, Figure 6.2, with the Blackfin SDRAM activities via a common clock, based upon the processor's system clock (SCLK). The micron SDRAM external memory used in many evaluation and production boards is an array of cells organized into row and columns. The cell address is time multiplexed, with the row (RAS) and column (CAS) access strobes transmitted sequentially. Figure 6.2 shows the schematic of the data watch capture block. Combinations of the Chip Select (CS) and Write Enable (WE) signals, together with the RAS and CAS signals, trigger the capture of the row or column address. The row address appears first on the address bus with the column address appearing on the SDRAM address bus in parallel with the data value on the data bus. This address bus logic is duplicated within the instruction watch units which can capture data (instructions) from the instruction data bus for advanced, context-aware, ATS features.

Non-time-critical information, e.g. configuration commands, can be sent via the SPI between embedded processor and ATS co-processor. Since one SPI word is 16-bit in length, the watched address (32-bit) is separated into 2 SPI words as row address and column. Figure 6.3 presents the state machines of using the SPI to read or write to ATS co-processor registers. Although the state machines are specific to the configuration commands, the idea is generally applicable to other purposes, for example, transferring rich initial data race analysis report from ATS co-processor to embedded processor.

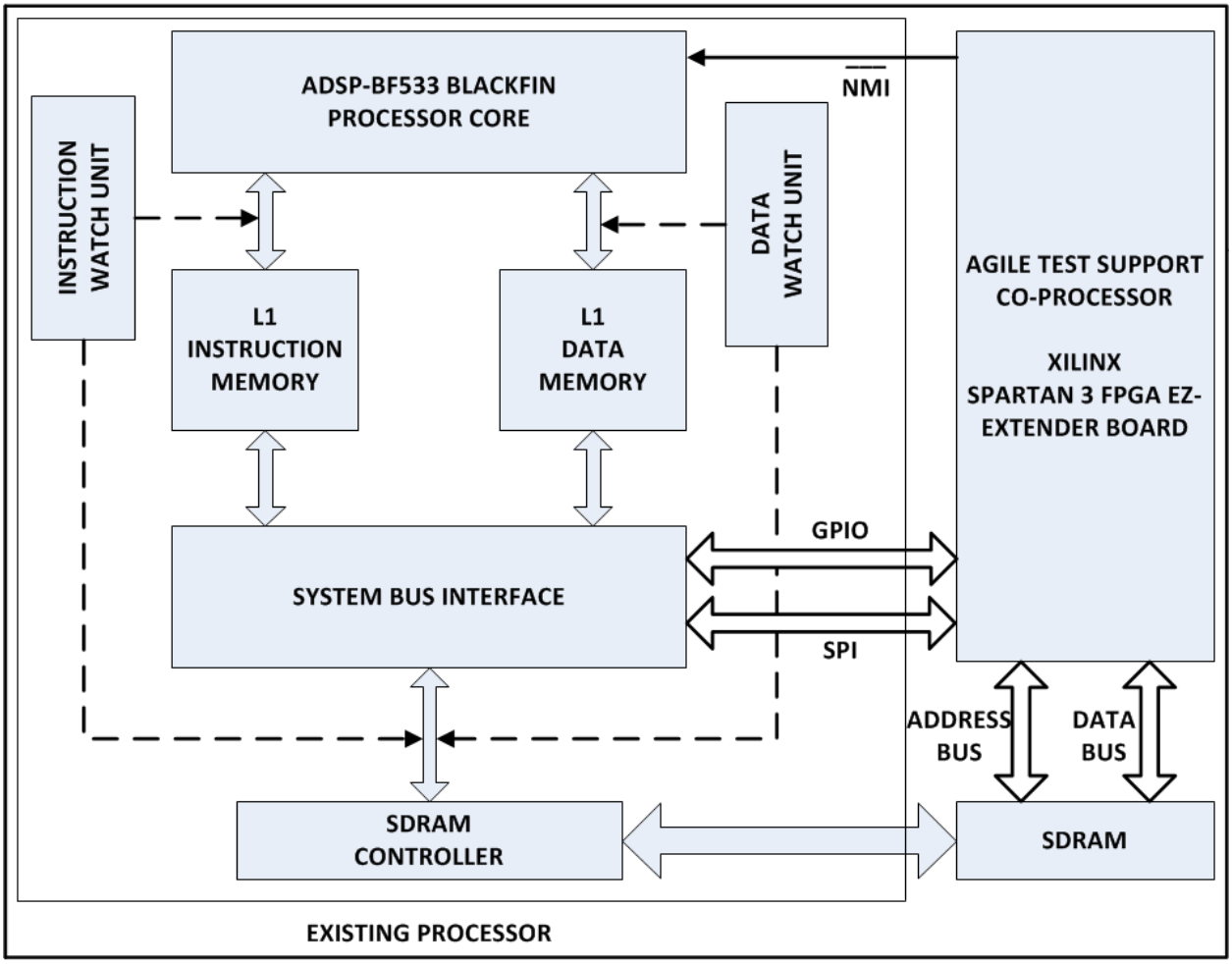


**Figure 6.1 – This schematic illustrates the configuration of the FPGA ATS co-processor relative to Blackfin BF533 core. The external coprocessor can only monitor the SDRAM bus activity and not the core L1 memory activity, in contract to Blackfin's instruction and data watch unit capabilities.**

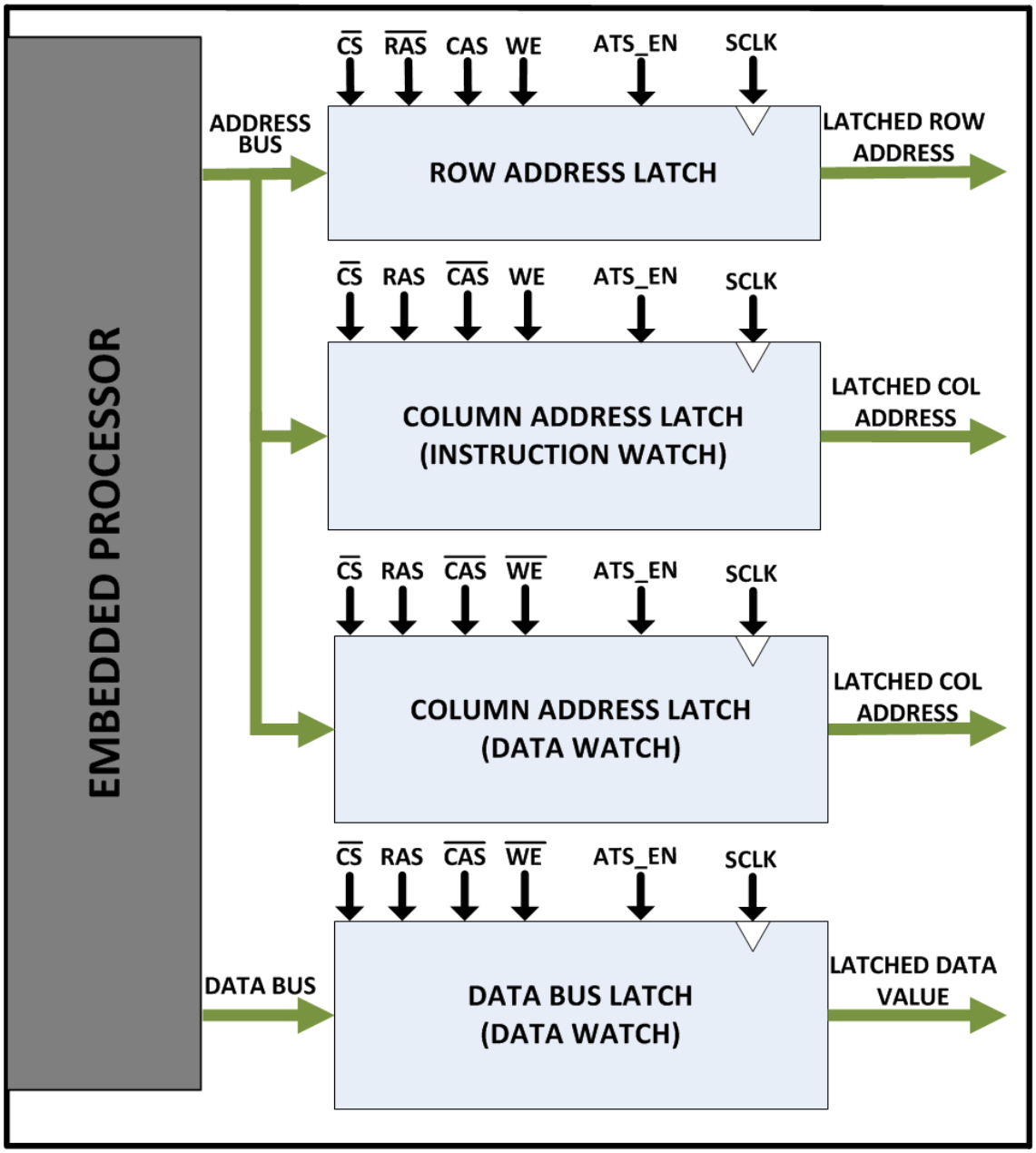


**Figure 6.2 – Bus capture blocks logic. On the raising edge of SCLK (System Clock), the four control signals: CS (Chip Select), RAS (Row Access Strobe), CAS (Column Access Strobe) and WE (Write Enable) are used to determine when to latch the bus information, provided the ATS_EN (ATS co-processor Enable) line is high. Since the address bus is multiplexed, row address, column address for instruction watch (memory read operation) and column address for data watch (memory write operation) are distinguished by different patterns of the four control signals. Data bus's value is latched only when column address of data watch appears on the address bus. The data value is used to configure the ATS coprocessor's registers**

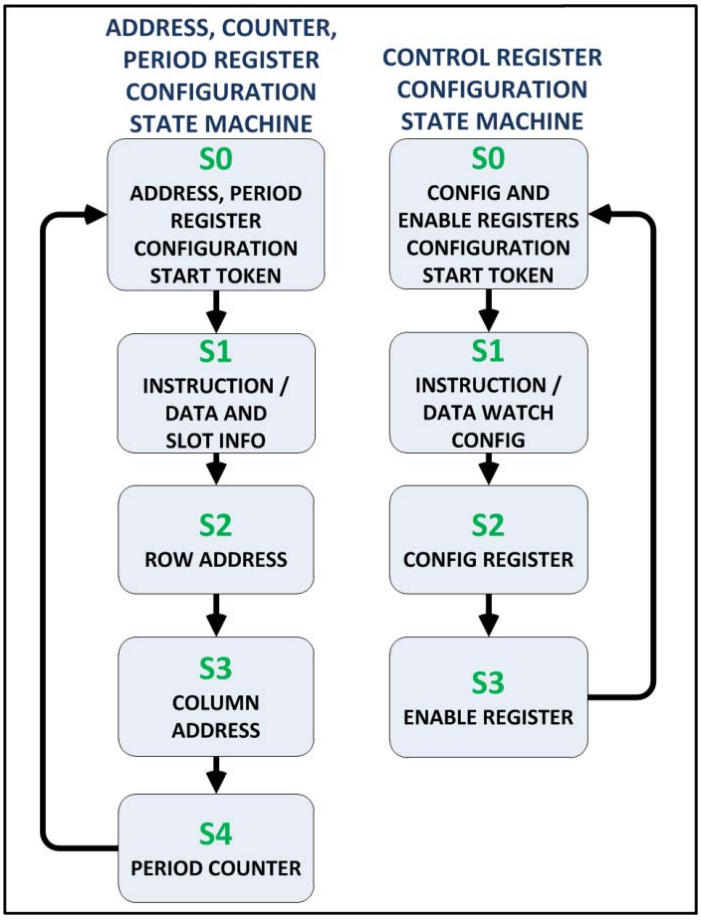


**Figure 6.3 – ATS co-processor SPI configuration state machine**

### 6.2 Identifying Data and Instruction Watch Activity

Figure 6.4 illustrates the proposed use of our ATS test insertion co-processor for inserting tests in a multi-threaded test environment. In Step 1, a run-once function is used to configure the ATS test insertion co-processor's Memory-Mapped Watch Unit Registers to watch instructions within a specific thread component or any access to a shared memory location via SPI or standard memory write instructions. The specific locations of instructions or data can be specified as part of the project's loader description file (.ldf) or determined from the linker map. Step 2 is to allow

the original, unmodified, threads to execute while (Step 3) the processor's instruction and data address busses are monitored by the co-processor.

No processor overhead is incurred (Step 4) until the co-processor recognizes a defined watched activity. At this time (Step 5) the co-processor pulls the processor's NMI line high. On receiving the NMI signal (Step 6), the processor switches to a NMI handler which (Step 7) inserts the test which can be designed to immediately completely analyse the watched condition when that incurs a lower overhead than storing test information for later analysis. Finally (Step 8) the interrupted watched activity is allowed to complete.

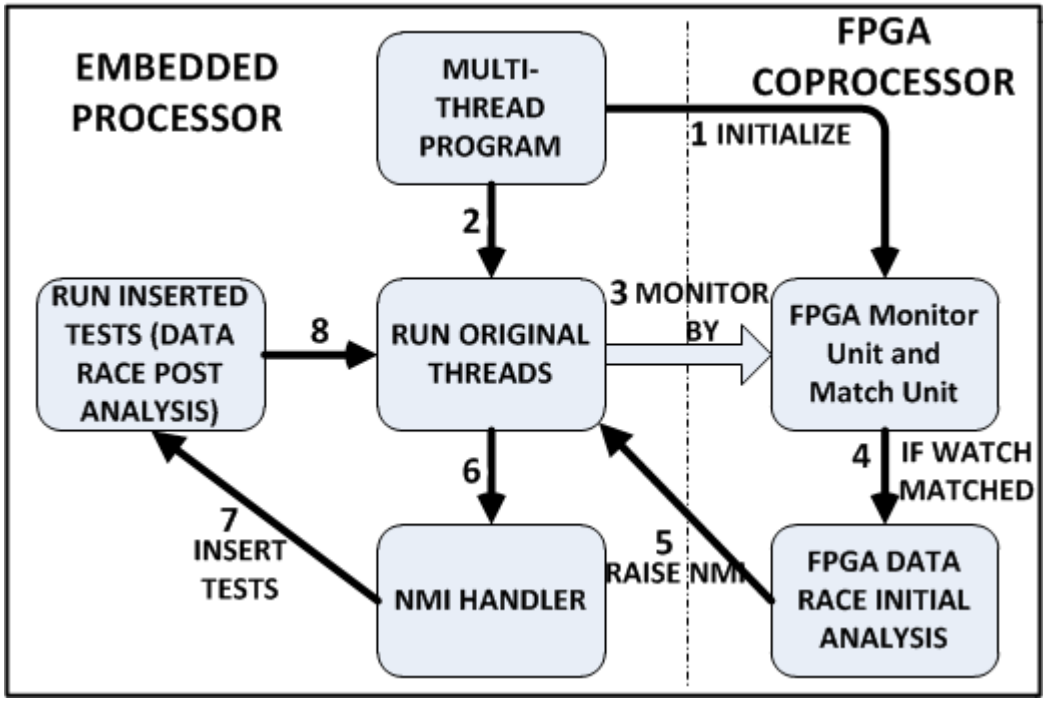


**Figure 6.4 – Program flow of the FPGA-based test insertion system. (1) The multi-thread program running on the embedded processor first initializes the co-processor via an SPI or standard memory write instructions. (2) Then the original threads are run with (3) co-processor monitoring the processor's data and address bus activities. On finding a watched activity (4), an NMI is generated (5) causing the processor to (6) run an NMI handler which (7) inserts a test before (8) allowing the watched activity to execute without further interruptions.**

We made three design decisions intended to minimize the detailed knowledge of memory interfacing required to move the use of the ATS co-processor across a range of processor families.

- ATS instruction and data watch units would be configured as write-only memory mapped registers. On the Blackfin processor this was done by using the same addresses at the bottom 1 K of the evaluation board's external SDRAM memory space.
- The system's loader description file would be modified so that the linker did not place non ATS related program code or data into those 1 K locations.
- The ATS co-processor registers were also configured to be accessed for read and write operations over simply programmed SPI or general purpose I/O interfaces common on embedded chips.

The first two decisions allow the existing SDRAM logic to handle all hand-shaking timing issues when the processor configures the FPGA registers over the processor's address and data buses. The third allows read operations on the ATS registers to confirm their configuration during testing, which is a non-time critical situation. Table 6.1 illustrates the memory mapped ATS co-processor registers at the bottom (last) 1 K of Blackfin's external memory. The full address is computed with the row and column addresses and compared to *WATCH_ADDRESSES* registers. If matched, the value of an associated *WATCH_COUNTER* register will be decremented by 1. When *WATCH_COUNTER* reaches 0, the *WATCH_COUNTER* will be automatically reloaded with the value of *WATCH_PERIOD*, allowing further, possibly delayed, watch operations.

At the same time, the NMI output pin is raised (high) forcing the embedded processor to run the NMI handler – inserting a test, see Figure 5.3. Line 108–113. Additional information about which watched location caused the interrupt, thread identification, can be sent to the embedded processor via GPIO (parallel) or SPI (serial) lines. During prototype testing, we achieved no speed advantages with either approach given that the ATS initiated transmission is executed in parallel with initial NMI service routine instructions. However, the ATS SPI interface is easily multiplexed with other devices to free up the GPIO lines from more traditional interfacing purposes.

**Table 6.1 – ATS co-processor memory mapped registers (Specific to ADSP-BF533 EZ-KIT Lite)**

| ATS Register Name | Note |
|---|---|
| Data Watch Enable Register | To enable Data Watch activity |
| Data Watch Config Register | To configure watch for read/write operations |
| Instruction Watch Enable Register | To enable Instruction Watch activity |
| Instruction Watch Config Register | To configure watch for read/write operations |
| Data Watch Address Register | To store the watched address |
| Data Watch Period Register | To reload the counter number when NMI signal is sent |
| Instruction Watch Address Register | To store the watched address |
| Instruction Watch Period Register | To reload the counter number when NMI signal is sent |

## 6.3 Performance Comparison of Existing and Proposed Hardware Test Insertion Mechanism

In this section, we provide a theoretical analysis of the expected performance hit associated with using a number of existing hardware test insertion mechanisms and compare these predictions to the actual overheads reported in (Smith et al. 2010). We then compare these results with predicted and actual performance hits associated with the proposed FPGA based ATS co-processor.

The performance ratio, PR, associated with inserting tests can be estimated by considering an application containing N instructions implemented on an 'ideal' embedded processor capable of executing 1 instruction / system clock cycle. Let a fraction k of those instructions require instrumentation in order to perform a test that executes in $C_{TEST}$ cycles. The performance ratio is:

$$\begin{aligned} PR &= \frac{\text{Execution Time With Inserted Tests}}{\text{Original Execution Time}} \\ &= 1 + k\left(C_{TEST} + C_{RECOGNIZE} - 1\right) \end{aligned} \qquad \text{(Equation 6.1)}$$

when the test insertion mechanism requires $C_{RECOGNIZE}$ clock cycles to recognize that a test insertion operation must be performed (Smith et al. 2010). Since recognizing the need of test insertion is a background hardware task, no overhead costs are incurred unless the specified data or instruction activity is recognized. At that time, $C_{PIPELINE}$ cycles are lost to re-establishing the system pipelines as the processor jumps into and out of the service routine. A further $C_{WRITE}$ cycles are required to re-configure (write) the watch unit's down counter to (1) allow the current watched event to complete normally without triggering unwanted interrupts and (2) permit future watched events to be recognized.

- ***Theoretical performance comparison***

The theoretical performance of the proposed ATS coprocessor on recognizing specific data and instruction activity can be compared to that for the existing Analog Devices Blackfin (BF5XX) processor family (Huang et al. 2008; Smith et al. 2010). On recognizing the desired instruction activity, the Blackfin processor triggers an exception, breaking the 10-stage processor pipeline. Modifying the instruction watch counter debug register to permit further test insertions takes another 4 cycles on this class of 32-bit RISC processor. Four additional cycles are required to re-establish the pipeline on exiting the exception handler (Analog Devices 2003). To avoid double counting, Smith et al. (Smith et al. 2010) suggested that the costs of saving and recovering the registers used within the exception handler be considered to form part of inline inserted test. Thus $C_{RECOGNIZE}$=18 cycles under ideal circumstances for the Blackfin instruction watch unit, roughly twice the number of stages in this processor's pipeline.

As discussed early, on sending a NMI signal to embedded processor, the FPGA-based ATS co-processor can be designed to reconfigure itself without processor intervention, resulting in $C_{WRITE}$=0. Therefore we anticipate $C_{RECOGNIZE}$=14 for ATS co-processor on both instruction and data watch functionalities. The fact that theoretical performance result of ATS system outperforms the ideal circumstances of watch units present on Blackfin processors is meaningful, because it permits us to claim ATS co-processor still is useful to those embedded processors with 'ideal' data watch capabilities. If we assume that 1% of the data or instruction activity requires watching, $k = 0.01$, then the performance hit of inserting a test is *1.14*.

- ***Performance analysis of dual instruction watch and data watch method***

Huang et al. (Huang et al. 2008) determined that the Blackfin data watch debugging hardware triggered less-desirable emulation events. These connect the live system-under-test to an external

platform through a boundary scan (JTAG) mechanism, permitting debugging either via "human intervention" or scripted tests. Rather than activating one slow (200,000 cycle) emulation event every 1 / k instructions, they proposed a faster solution which combined the Blackfin's data and instruction watch capability to insert tests with lower overhead. They programmed the instruction watch unit to trigger every *m* instructions during a test's execution to monitor whether the data watch unit (configured to passively watch data activity) has recognized a desired event.

This dual-purpose exception service routine (XSR) has an execution time of:

$$C_{RECOGNIZE} = C_{PIPELINE} + \left(1 + \frac{k}{m}\right) C_{WRITE} + C_{CONDITION-CHECK} + 1 \qquad \text{(Equation 6.2)}$$

Given that high-level language memory operations will be interspersed with many low level, non-memory modifying instructions, Huang et al. (Huang et al. 2008) identified that it was only necessary to cause an exception at every 6th instruction leading to 350 cycles / test insertion (average 7x fold performance hit). They considered this as offering borderline performance when evaluating whether newly developed threads had appropriate locks when accessing shared resources, but inadequate for real-time system oriented (acceptance) testing with multiple threads operating.

- ***Performance analysis of dual data cache and instruction watch methods***

Smith et al. (Smith et al. 2010) showed how a processor's cache protection logic buffer (CPLB) could be configured to deliberately cause a "cache-miss" when certain "watched" data locations in memory were accessed. However, rather than using the data cache exception handler for reloading the data cache, a data watch test was inserted. Further memory accesses to the same

location during the inserted test will find a valid CPLB descriptor for the requested data address, avoiding nested exceptions. On completion of the inserted test, the data-cache exception handler is exited and the original memory operation interrupted by the "cache-miss" exception can be completed.

In principle, such a test insertion mechanism would be possible on any embedded processor supporting a cache and with a performance only a few cycles slower than the ideal direct watch operation. Unfortunately, the data cache CPLB descriptors are left configured to cause no further exceptions, i.e. only one data activated test can be inserted. This problem can be overcome by using the processor's instruction watch capability to trigger a second exception that re-enables the data-cache watch test insertion mechanism (Smith et al. 2010). The total overhead for this dual cache / instruction watch approach was calculated as 60 cycles / test insertion (Smith et al. 2010).

- ***Actual performance comparison***

The actual overheads on the Blackfin ADSP-BF5XX family for the ideal instruction watch, combined instruction / data watch and combined cache miss / instruction watch approaches were measured as 22, 480 and 84 cycles / test insertion respectively, (Smith et al. 2010). The higher than "ideal" values were associated with pipeline delays that could not be executed concurrently with other operations. Unanticipated additional test insertion overhead arose from the need to insert a software patch to recover from the hardware anomaly (silicon bugs) for exception and interrupt handlers in the Blackfin chips silicon version 0.5 or lower (Analog Devices 2011b) is taken into account, the actual performance of the ideal instruction watch is 37 cycles / test insertion. The actual performance of ATS co-processor for both instruction watch and data watch is 27 cycles / test insertion. Therefore, the ATS co-processor was found to have lower overhead

than any of the instruction watch (Smith et al. 2010), combined instruction / data watch (Huang et al. 2008) or combined cache miss / instruction watch (Smith et al. 2010) test insertion approaches. We anticipate the overheads will be closer to the ideal performance calculated for processors in silicon version 0.6 because the cycles used to recover from hardware anomalies required for interrupt and exception handlers will not be required. While these actual results are specific to the Analog Devices Blackfin family of processors, the performance and implementation issues are indicative of those to be expected from other embedded processors with hardware debug capabilities. Table 6.2 summarizes the theoretical and actual performance comparison and performance hit (when 1% of the instruction / data requires test insertion, $k = 0.01$) comparison associated with ATS co-processor and current existing hardware-assisted test insertion methods.

**Table 6.2 – Theoretical and Actual Performance Comparison of ATS Co-processor and Existing Hardware-assisted Test Insertion Methods**

| Hardware Test Insertion Methods | Theoretical Cycles | Actual Cycles | Performance Hit (*k = 0.01)* |
|---|---|---|---|
| | (Smaller the better) | (Smaller the better) | (Smaller the better) |
| Existing Instruction Watch on Blackfin Driven by Exception Handler | 18 | 23 | 1.23 times |
| Existing Data Watch on Blackfin Driven by Emulation Handler | 18 | 200,000+ | 2,000+ times |
| Dual Instruction Watch and Data Watch Driven by Exception Handler | 350 | 480 | 11.8 times (7 times performance hit without inserting any test) |
| Dual Cache Protection Logic Buffer (CPLB) and Instruction Watch Driven by Exception Handler | 60 | 84 | 1.84 times |
| Instruction and data Watch on ATS co-processor Driven by NMI Handler | 14 | 15 | 1.15 times |

## 6.4 Implementing ATS Co-processor on a Dual Core Embedded Processor

To further investigate the capability of monitoring the internal memory of the embedded processor, the dual core in a same chip implementation is proposed. The new SoC (system on chip) ALTERA's Cyclone V Device (Altera 2012) solution places a hard-core (an ARM core) and a soft-core (FPGA) on a single chip. In principle, with both the soft-core and hard-core placed on the same chip, an ATS co-processor implemented on the soft-core should be able to monitor both internal (processor to on-chip RAM) and external (processor to SDRAM) memory activities. However, because of the connection limitations between the soft-core and hard-core of

the Cyclone V Device, neither the external memory activities nor the internal memory activities are visible to the FPGA soft-core. This prevents the direct on-chip implementation of an ATS co-processor.

A possible workaround to this problem, although beyond the scope of the current thesis, would be to have the hard-core send memory access monitoring information to the soft-core via the high-speed bridge between them. This method, however, might impose a large overhead onto the hard-core since the hard-core is responsible for sending the memory access information to the soft-core. Another consideration is the high-speed bridge is shared with other purposes, probably slowing down the traffic of all the transmissions.

### 6.5 Chapter Summary

In this chapter, detailed ATS co-processor hardware design and implementation decisions are presented. A performance comparison was made between hardware-assisted test insertion methods using the proposed ATS co-processor and the existing instruction and data watch debugging units of the Analog Devices ADSP-BF5XX. The ATS co-processor's overhead was orders of magnitudes lower than that for the Blackfin's existing data watch unit; and better than the existing instruction watch unit and other suggested approaches that involved combinations of this processor's data watch, instruction watch and cache-miss detection units. Possible implementation of ATS co-processor on a dual core embedded processor is discussed. Next chapter presents a demonstration of ATS co-processor detecting race condition of multi-threaded program running at an embedded processor.

# CHAPTER SEVEN: DEMONSTRATION OF USING AGILE TEST CO-PROCESSOR TO DETECT RACE CONDITION

In this chapter, the use of ATS test insertion co-processor in a multi-threaded application running in embedded system environment is demonstrated. The case study system includes an embedded system platform, an operating system, an ATS test insertion co-processor and a multi-threaded application. The details of the case study system are present in detail. Various race condition detection methods are discussed and because of the limitation of the operating system using, I tailor the operating system and the multi-threaded application to enable the demonstration in Section 2 while the running results are presented in Section 3.

## 7.1 Introduction to the Case Study System

As mentioned in Chapter Six, the embedded system platform used for this case study system was a Blackfin FPGA EZ-Extender (Analog Devices 2012) and a Blackfin FPGA EZ-KIT Lite board (Analog Devices 2004). The Integrated Software Development and Debugging Environment (IDDE) used is the CrossCore Embedded Studio (CCES) (Analog Devices 2013a) which is released in April 2013 by Analog Devices instead of the previous IDDE Visual DSP++ Development Software (VDSP). A user-friendly µC/OS-III (Micriµm 2010) real time operating system (RTOS) is integrated into CCES to provide seamless support for configuring RTOS and developing multi-threaded applications in the embedded context. EmbeddedGear (Wiederseiner, 2011), a multi-threaded program running at VDSP with using the Visual DSP++ Kernel (VDK) operating system (Analog Devices 2006), was adapted to be used in CCES with µC/OS-III RTOS. EmbeddedGear is a good demonstration application for ATS test insertion co-processor

to detect data-race condition since it is an easy-understandable application with requirements on global variables synchronization.

### 7.2 Race Condition Detection Method Used

As discussed in Chapter Five, a race condition is said to exist if no proper synchronization was made before an access to the shared resources. Semaphores and mutexes (Courtois et al. 1971) are synchronization mechanisms that ensure the shared resources are accessed properly. In this demonstration application, we decided to implement a simple race detection technique – Does the thread has the proper locks (semaphores / mutexes) when accessing the shared variable? To decide if the access is valid, we have to know:

1. Which thread is accessing the shared variable?
2. What are the semaphores or mutexes that ensure the synchronization of the watched shared variable?
3. Does the thread accessing the shared variable possess those semaphores or mutexes?

As discussed in Chapter Five and Six, the NMI signal is sent to the embedded processor to execute the NMI handler when ATS co-processor recognized an access to the watched shared variable. By reading the trace-buffer stack registers, the interrupted thread by the NMI handler can be determined, which provides the first information required. For the second information, we assume that multi-threaded program (EmbeddedGear) keeps the relationship between global shared variables and semaphores or mutexes that protect them. By maintaining a semaphore owner list for each semaphore, the third required information is supplied.

## 7.3 ATS Co-processor Running Results

InitThread.cpp | ManeuversThread.cpp | CleanThread.cpp | cpu_core.c | Utilities.h | SPI.h | SPI.cpp

```
66      /*
67       * Necessary initialization, including GPIO, SPI, Trace-buffer and UART
68       * */
69
70      unsigned int addrReg;
71      unsigned short counter;
72
73      // Set the WP using SPI
74      EUNIT_printf ("Configuring the ATS Co-processor...\n");
75      SetATS_SPI(WATCH_WRITE, DATA_WATCH, SLOT_1, (unsigned int)&g_Throttle, MATCH_ONCE);
76
77      // Read back the configuration info to check if FPGA has been set correctly
78      ReadATS_SPI(DATA_WATCH, SLOT_1, &addrReg, &counter);
79      EUNIT_printf ("Reading the ATS configuration information...\n");
80      EUNIT_printf ("\nATS is watching variable at 0x%08X.\nPeriod counter is set to: %d\n", addrReg, counter);
81
82      adi_int_result = adi_int_InstallHandler (
83              (uint32_t)                  ADI_CID_NMI,
84              (ADI_INT_HANDLER_PTR)       NMIHandlerASM,
85              (void *)                    0,
86              (bool)                      true
87          );
88      CheckIntResult(adi_int_result);
89
90      /*
91       * uC/OS initialization and threads creation START
92       */
```

**Figure 7.1 – ATS co-processor configuration. Demonstrated in CrossCore Embedded Studio environment**

Figure 7.1 illustrates the Initialization methods that are required by EmbeddedGear application, ATS co-processor and the CCES running environment. The ATS co-processor is configured by calling the method *SetATS_SPI()* (Line 75) where the first specify the ATS co-processor to watch write operation and second parameter configure to perform data watch. The third to the last parameters which data watch slot is used, what the shared variable (*g_Throttle*) address is watched and the number of times matches to generate NMI signal. *ReadATS_SPI()* (Line 78) reads back the configuration values to ensure the configuration made by *SetATS_SPI()* are correct. Code in Line 82 to 87 registers the NMI handler as *NMIHandlerASM* which is used to insert the data race analysis test to embedded processor. *EUNIT_printf()* (Lines 68, 74, 79 and 80) are output information printed via UART.

```
255     EUNIT_printf("\nATS has started watching\n");
256     ATSWatchStart(true);
257
258     g_raceNotFound = true;
259
260     EUNIT_printf("EmbeddedGear has started\n\n");
261     OSStart(&err);
262     CHECK_ERROR(err,"Example failed: Error in OSStart");
263
264     while (g_raceNotFound);
265
266     ATSWatchStart(false);
267
268     EUNIT_printf("\nThere was a data race in your program\n");
269     EUNIT_printf("Data race is found at instruction located at 0x%08x\n", dataRaceAddr);
270     EUNIT_printf("\nATS has stopped watching\n");
271     EUNIT_printf("EmbeddedGear has been terminated\n");
272
273     /*
274      * Delecting all the threads.
275      */
276
277     /*
278      * uC/OS setup and initialization END
279      */
```

**Figure 7.2 – Code for starting the ATS co-processor data watch and the EmbeddedGear application as well as reporting the race condition**

In Figure 7.2, *ATSWatchStart()* (Lines 256 and 266) starts or stops the watch operation on the ATS co-processor. Line 261 starts the EmbeddedGear multi-threaded application. This *InitThread* waits to allow the execution of other threads until the race condition is reported (Line 264). Code in Lines 268 to 271 reports the data race issue with which lines of code are found to be causing race condition.

```
COM1:57600baud - Tera Term VT
File  Edit  Setup  Control  Window  Help
Outputs are redirected to UART.

Configuring the ATS Co-processor...
Reading the ATS configuration information...

ATS is watching variable at: 0x02AAAD56.
Period counter is set to: 1

ATS has started watching
EmbeddedGear has started

Throttle: 80.000000, EngineRPM: 2474.945000, VehicleSpeed: 4.090595, Gear: 1, Time take in s: 0

There was a data race in your program
Data race is found at instruction located at: 0xffa02922

ATS has stopped watching
EmbeddedGear has been terminated
```

**Figure 7.3 – UART console output, reporting the EmbeddedGear running information and race condition information.**

```
Outline  Disassembly                                   0xffa02922
ffa028f4:   CALL _printf ;
ffa028f8:   R0 = 1 ;
ffa028fa:   P1.L = 0xe4c ;
ffa028fe:   P1.H = 0x0 ;
ffa02902:   CALL ( P1 ) ;
45                g_Throttle = maneuvers.getThrottle();
ffa02904:   R4.L = 1840 ;
ffa02908:   R4.H = -128 ;
ffa0290c:   R0 = ROT R4 BY 0 ;
ffa02910:   CALL getThrottle ;
ffa02914:   P1.L = 0xad54 ;
ffa02918:   P1.H = 0x2aa ;
ffa0291c:   [ P1 ] = R0 ;
46                g_BrakeTorque = maneuvers.getBrake();
ffa0291e:   R0 = ROT R4 BY 0 ;
ffa02922:   CALL getBrake ;
ffa02926:   P0.L = 0x3688 ;
ffa0292a:   P0.H = 0xff80 ;
ffa0292e:   [ P0 ] = R0 ;
49                OSSemPend(&g_blockManeuversSem, 0u, OS_OPT_PEND_BLOCKING, NULL, &err);
ffa02930:   [ SP + 0x10 ] = R7 ;
ffa02932:   [ SP + 0xc ] = P5 ;
ffa02934:   R2 = 0 ;
ffa02936:   R1 = 0 ;
ffa02938:   R0.L = 2428 ;
ffa0293c:   R0.H = -128 ;
ffa02940:   CALL OSSemPend ;
50                CHECK_ERROR(err, "Error in sem pending of g_blockManeuversSem in Maneuver
ffa02944:   R0 = [ FP + 0x8 ] ;
ffa02946:   CC = R0 == 0 ;
```

**Figure 7.4 – Disassembly window shows which instruction made the race condition occur**

The UART console output result is shown in Figure 7.3. The address circled by the first red oval is the watched variable address (*g_Throttle,* Line 75 in Figure 7.1). To locate the guilty instruction address which is causing the race condition, second yellow oval circled the address which is close to the guilty instruction. The disassembly window information shown Figure 7.4 shows that the NMI interrupted instruction was located at *0xffa02922,* which means the defect is at a number of lines instruction before. The guilty code is circled in red color in Figure 7.4 and it is located in Line 45 of the C++ source file. The reason of the ATS was pointing to the instruction located after the actual defect is because the ATS takes a number of cycles to compare the values appear on the address bus and generate the NMI signal to the embedded processor. This is not a serious issue since the reported location is very close to the guilty code and the user is aware of what variables are watched. In the next section, we determine the number of cycles that the ATS co-processor takes to perform the data race detection method used in this thesis.

### 7.4 Performance Analysis

The ATS co-processor requires the embedded processor to insert a test to detect data race condition whenever shared memory is accessed. Then the embedded processor is responsible to collect the necessary information to determine whether a race condition has occurred or not. Figure 7.5 demonstrates the relative steps required to obtain the needed information and performing the race condition analysis. Since this ATS co-processor implementation processes 6 Data Watch units, 6 threads and 6 mutexes associated with these threads (one mutex for one thread) are assumed to be used in this application.

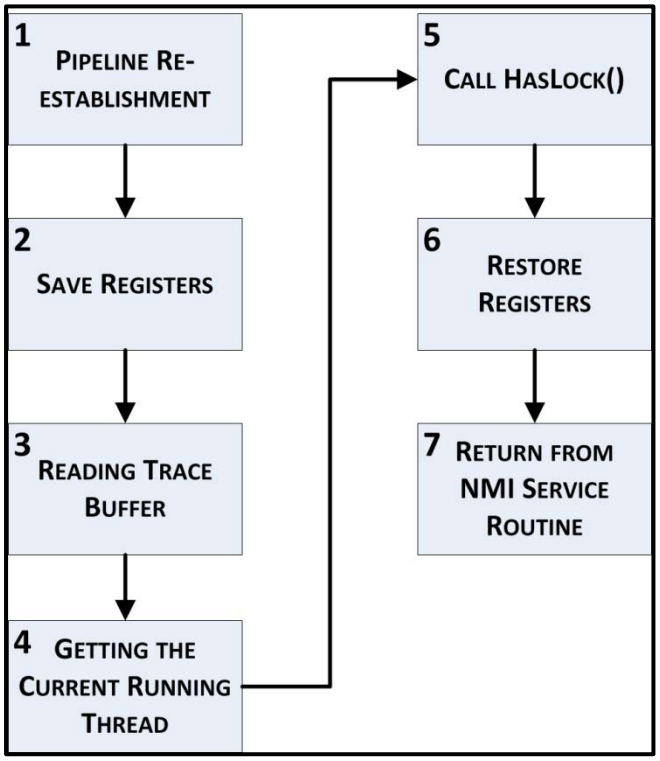


**Figure 7.5 – Flow chart of ATS co-processor detecting race condition in a multi-threaded application on recognizing a shared memory access**

In Figure 7.5, stage 1 takes 10 cycles to re-establish the pipeline and 4 cycles each to save and restore registers needed in stages 2 and 6. Pointing to the *Trace Buffer (TB)* and reading it (stage 3) takes 12 cycles since it takes 2 cycles to point to TB and 10 cycles to read the interrupted instruction's address from the TB (calling stack). In stage 4, with the interrupted instruction address at hand, determining which thread is currently running takes 3 * (4 + 2 + 5 + 4) = 45 cycles on average. Since 6 threads are assumed being used / watched in this application, 3 comparisons are required to determine which thread is running on average. 4 cycles are used to setup the thread function's head and end and 2 cycles are used to do the comparison. 5 cycles are taken to perform the if-else statement jump and another 4 cycles wasted by the loop overhead. Stage 5 calls *HasLock()* function to check if the currently running thread has the correct mutex to

access the shared variable. Since the *HasLock()* function is an inline function and the parameters required are setup by stage 4, no overhead is occurred when calling it. In *HasLock()* function, it takes 3 * (5 + 2 + 4) + 2 * 4 = 41 cycles to check if the thread has the correct mutex. Again, 3 is the average number of comparisons with 6 threads and 5 cycles is used for if-else jump and 2 cycles for comparison and 4 cycles for loop overhead. Reporting the data race detection results (writing to a global boolean flag showing if the race condition has occurred and instruction address that caused the race condition) take 8 cycles (2 * 4, 4 cycles for writing to each memory location). In stage 7, it takes 5 cycles to return from non-maskable interrupt service routine (Analog Devices 2003). Therefore, theoretically it takes 121 cycles averagely in total to perform race condition detection for each watched shared memory access with ATS co-processor. The best case (finds the associated thread or mutex on the first comparison) takes 66 cycles and the worst case (finds the associated thread or mutex on the last comparison) takes 176 cycles. With 1% of the instructions / data accesses that need to be analyzed, the performance loss is 1.21 to the original program on average. If the ATS co-processor is used in checking the race condition of shared variables used in a loop, only the first access of the shared variables it needed to be checked for race condition. It is because once the loop acquired the locks, it would not release the lock until the loop is finished. Therefore, it takes 121 + 8 = 129 cycles to run a race condition detection and turn off and on watch operation of ATS co-processor after the first access of the variables and after the loop, regardless how many times of shared variable accesses within the loop.

The ATS co-processor can be set to perform off-line race condition analysis, which means that on each NMI handler, the embedded processor only collect and record the needed information for race condition analysis without actually performing the analysis so as to reduce

the real time performance impact caused by the NMI handler. The actual race condition analysis will be run with the information recorded against all the shared variable accesses when the recording buffer is full or the program is in the a non-time-critical section or the multi-threaded program ends. Thus the NMI handler is responsible for recording the interrupted instruction (stage 3 on Figure 7.5) and saving the states of all mutexes (stage 5 on Figure 7.5) with stage 4 can be done off-line. In this case, stage 5 takes 6 * (4 + 4) = 48 cycles as we have to save 6 mutex's states and it takes 4 cycles for writing memory and 4 cycles wasted in the loop overhead. Therefore, NMI handler takes 83 cycles to save required information for off-line analysis. It is around 50% better than the average performance loss with performing race condition analysis (121 cycle's losses) on each shared variable access.

### 7.5 Chapter Summary

In this chapter, the use of ATS co-processor in demonstrated in a real multi-threaded application running on an embedded processor. The case study system and the race condition detection method are briefly introduced. The running results and performance analysis are presented and discussed. To perform a race condition detection, the ATS co-processor would incur 121 cycles overhead on each race detection averagely and incur 1.21 times performance losses to the original program provided that 1% the instructions / data accesses that needs to be analyzed. Comparing to other software instrumentation methods and hardware-assisted test insertion methods, this is a great improvement.

# CHAPTER EIGHT: CONCLUSION AND FUTURE WORK

## 8.1 Summary and Conclusion

Agile methodologies' successes have been seen in enterprise systems and efforts from early Agile practitioners were made to translate the achievement of Agile onto embedded systems development. However, the Agile-inspired embedded software development (AIESD) research field still remains obscure to researchers and industrial developers in this community. In this thesis, two major contributions have been made: (1) We have presented a panoramic overview of state-of-art AIESD with bibliometric results and answers to systematic mapping questions specific to AIESD by conducting a systematic mapping study (Petersen et al. 2008) on AIESD research domain; (2) An Agile test support (ATS) co-processor was proposed and implemented to provide low-overhead test insertion capability to embedded processors.

A systematic mapping study on papers that investigated adopting or adapting Agile methods onto embedded software domain in order to offer an overview in this research field. We have found 171 studies and after exclusion, 78 studies are included which are published between 2002 and 2013. To identify, categorize and summarize the current status of AIESD research area, a set of classification schemes has been generated. Papers are then sorted into the schemes and data are extracted from the pool of studies. Data are presented in the form of mapping to answer the research questions. Not only the bibliometric and demographic trend, year of publication, article type classification, top venues, most cited papers and active researchers are presented, but also the research type, contribution, adapted Agile methods / TDD used, tool functionalities, domain of study, whether case studies are included and article type of the paper are shown via systematic

mapping. Furthermore, by going beyond the scope of systematic mapping, the adapted and adopted Agile methods are presented and discussed with three aspects: testing framework, support tools proposed and embedded Agile lifecycle. This mapping study indicates that the AIESD research realm is a hot topic but with little synchronization and summarization. By classifying the pool of studies into different categories, the characteristics of AIESD field can be identified. This study can also serve as a baseline paper for follow-up researches.

Advantages of utilizing the hardware features of embedded processors to insert tests are outlined. As these features are not present across various families of processors, we proposed an Agile test support (ATS) co-processor to provide a low-overhead hardware assisted test insertion mechanism upon recognizing specified data bus activity. A performance comparison was made between hardware-assisted test insertion using the proposed ATS co-processor and the existing instruction and data watch debugging units of the Analog Devices ADSP-BF5XX (Analog Devices 2011a). The co-processors' overhead was orders of magnitudes lower than that for the Blackfin's existing data watch unit; and better than the existing instruction watch unit and other suggested approaches that involved combinations of this processor's data watch, instruction watch and cache-miss detection units. A demonstration of using the ATS co-processor to detect race condition was also presented.

This thesis is derived from three papers that I have made major contribution to. I started the systematic mapping study on AIESD as a graduate course project for Agile Software Engineering (SENG 615) and this paper (Deng et al. 2014) was evolved and under the preparation for a January 2014 submission to Journal of Information and Software Technology (JIST). The concept of ATS test insertion co-processor was proposed in (Smith et al. 2013) where I was the second author. I did a major rewrite after the paper has been accepted and I was

responsible for the ATS co-processor design, implementation and performance analysis and comparison. The detailed ATS co-processor implementation and initial performance analysis and comparison was presented in (Deng et al. 2013) where I was the first author and I have presented this work in an international conference (Irish Signals and Systems Conference, ISSC 2013) in Letterkenny, Ireland.

## 8.2 Future Work

While the initial systematic mapping results on AIESD was presented, a useful update addressing the observed challenges to validity presented in Chapter Three Section 3.6 is encouraged. The search string applied should be able to represent the domain richness of embedded systems. More research questions can be proposed by investigating the included papers with more depth.

Based on the discovered studies included by this systematic mapping study, systematic literature reviews on AIESD testing frameworks, supporting tools with various functionalities, and embedded Agile lifecycles can be performed. The systematic literature reviews can provide best practices to AIESD research realm (Petersen et al. 2008; Kitchenham et al. 2011).

Incorporating with some useful software-instrumented dynamic race condition detection techniques, e.g. (Pozniansky and Schuster 2003; Pozniansky and Schuster 2007), the performance characteristics (speed and false error rates) of hardware-assisted ATS test insertion co-processor can be compared to software-instrumented race condition detection methods.

It would also be meaningful to evaluate the performance of ATS co-processor on difference embedded processors, e.g. Texas Instruments MSP430 (Texas Instruments 2013b). Another suggestion is to implement ATS co-processor as a component into FPGAs with soft-core

embedded processor, for example, the MicroBlaze (Xilinx 2012), in order to offer internal memory monitor capability.

## REFERENCES


(Altera 2012) Altera. (2012). Cyclone V Device Handbook: Volume 3: Hard Processor System Technical Reference Manual. [Online]. Available: Altera. http://www.altera.com/literature/hb/cyclone-v/cv_5v4.pdf. [Accessed 12 July 2013]

(Analog Devices 2003) Analog Devices. (2003). ADSP-BF531/532/533 Blackfin® Processor Multi-cycle Instruction and Latencies. [Online]. Available: Analog Devices. http://www.analog.com/static/imported-files/application_notes/EE-197.pdf [Accessed 12 April 2013]

(Analog Devices 2004) Analog Devices. (2004, April). ADSP-BF533 EZ-KIT Lite Evaluation System Manual. [Online]. Available: Analog Devices. http://www.analog.com/static/imported-files/eval_kit_manuals_legacy/402257434ADSP_BF533_EZ_KIT_Lite_Manual_Rev._1_3.pdf. [Accessed 19 November 2013]

(Analog Devices 2006) Analog Devices. (2006, April). VisualDSP++ 4.5 Kernel (VDK) User's Guide. [Online]. Available: Analog Devices. http://www.analog.com/static/imported-files/software_manuals_legacy/633835940452254139435088 45_vdk_ug.pdf. [Accessed 19 November 2013]

(Analog Devices 2011a) Analog Devices. (2011). ADSP-5XX Processors. [Online]. Available: http://www.analog.com/en/processors-dsp/blackfin/products/index.html. [Accessed 18 November 2013].

(Analog Devices 2011b) Analog Devices. (2011).ADSP-BF531/BF532/BF533 Blackfin Anomaly List for Revisions 0.3, 0.4, 0.5, 0.6 (RevG, 05-23-2011). [Online]. Available: Analog Devices. http://www.analog.com/static/imported-files/ic_anom/ADSP-BF531_BF532_BF533_anomaly_RevG.pdf . [Accessed 12 July 2013]

(Analog Devices 2012) Analog Devices. (2012). Blackfin FPGA EZ-Extender Manual. [Online]. Available: Analog Devices. http://www.analog.com/static/imported-files/eval_kit_manuals/Blackfin_FPGA_ext_man_rev.2.1.pdf. [Accessed 12 July 2013]

(Analog Devices 2013a) Analog Devices. (2013). CROSSCORE® EMBEDDED STUDIO FOR ANALOG DEVICES PROCESSORS. [Online]. Available: http://www.analog.com/en/evaluation/adswt-cces/eb.html. [Accessed 13 September 2013].

(Analog Devices 2013b) Analog Devices. (2013) "VISUALDSP++ DEVELOPMENT SOFTWARE," [Online]. Available: http://www.analog.com/en/dsp-software/vdsp-bf-sh-ts/sw.html. [Accessed 3 October 2013].

(Antonio et al. 2012) Antonio, E. A., Ferrari, F. C., & Ferraz Fabbri, S. C. P. (2012, May). A Systematic Mapping of Architectures for Embedded Software. *In Critical Embedded Systems (CBSEC), 2012 Second Brazilian Conference on* (pp. 18-23). IEEE.

(Atmel 2013) Atmel. (2013) Atmel Studio 6. [Online]. Available: http://www.atmel.ca/Microsite/atmel_studio6/. [Accessed 13 September 2013].

(Beck 2003) Beck, K. (2003). Test-driven development: by example. Addison-Wesley Professional.

(Beck and Andres 2004) Beck, K., & Andres, C. (2004). Extreme programming explained: embrace change. Addison-Wesley Professional.

(Bhat and Nagappan 2006) Bhat, T., & Nagappan, N. (2006, September). Evaluating the efficacy of test-driven development: industrial case studies. *In Proceedings of the 2006 ACM/IEEE international symposium on Empirical software engineering* (pp. 356-363). ACM.

(Boehm 1988) Boehm, B. W. (1988). A spiral model of software development and enhancement. *Computer*, *21*(5), 61-72.

(Courtois et al. 1971) Courtois, P. J., Heymans, F., & Parnas, D. L. (1971). Concurrent control with "readers" and "writers". *Communications of the ACM*, *14*(10), 667-668.

(Chen et al. 2001) Chen, M. H., Lyu, M. R., & Wong, W. E. (2001). Effect of code coverage on software reliability measurement. *Reliability, IEEE Transactions on*, *50*(2), 165-170.

(Cunningham 2002) Cunningham, W. (2002). FIT: Framework for integrated tests. Framework history. [Online]. Available: http://fit.c2.com/wiki.cgi?%20FrameworkHistory. [Accessed 18 November 2013]

(Dahlby 2004) Dahlby, D. (2004). Applying Agile Methods to Embedded Systems Development. *Embedded Software Design Resources*, 41, 1014123.

(da Mota Silveira Neto et al. 2011) da Mota Silveira Neto, P. A., Carmo Machado, I. D., McGregor, J. D., de Almeida, E. S., & de Lemos Meira, S. R. (2011). A systematic mapping study of software product lines testing. *Information and Software Technology*, 53(5), 407-423.

(Deng et al. 2013) Deng, D., Smith, M., Islam, S., & Miller, J. (2013, June). Prototype test insertion co-processor for agile development in multi-threaded embedded environments. *In Signals and Systems Conference* (ISSC 2013), 24th IET Irish (pp. 1-8). IET.

(Deng et al. 2014) Deng, D., Smith, M., & Miller, J. A Systematic Mapping Study of Approaches for Agile-Inspired Embedded Software Development. To be submitted to *Journal of Information and Software Technology*

(Engler and Ashcraft 2003) Engler, D., & Ashcraft, K. (2003, October). RacerX: effective, static detection of race conditions and deadlocks. In *ACM SIGOPS Operating Systems Review*(Vol. 37, No. 5, pp. 237-252). ACM.

(Engström and Runeson 2011) Engström, E., & Runeson, P. (2011). Software product line testing–a systematic mapping study. *Information and Software Technology*, 53(1), 2-13.

(Feathers 2011) Feathers, M. Cpp Unit Lite. (2011, April). [Online]. Available: http://c2.com/cgi/wiki?CppUnitLite. [Accessed September 2013].

(FitNesse 2003) FitNesse. (2003). FitNesse – a tool to specify and run acceptance tests for software systems. [Online]. Available: FitNesse.org http://fitnesse.org/. [Accessed 18 November 2013]

(Flanagan and Freund 2001) Flanagan, C., & Freund, S. N. (2001, June). Detecting race conditions in large programs. In *PASTE* (Vol. 1, pp. 90-96).

(Flanagan and Freund 2009) Flanagan, C., & Freund, S. N. (2009, June). FastTrack: efficient and precise dynamic race detection. In *ACM Sigplan Notices* (Vol. 44, No. 6, pp. 121-133). ACM.

(Fowler and Highsmith 2001) Fowler, M., & Highsmith, J. (2001). The agile manifesto. *Software Development*,*9*(8), 28-35.

(Garousi et al. 2010) Garousi, V., & Varma, T. (2010). A bibliometric assessment of canadian software engineering scholars and institutions (1996-2006). *Computer and Information Science*, 3(2), P19.

(Garousi et al. 2013) Garousi, V., Mesbah, A., Betin-Can, A., & Mirshokraie, S. (2013). A Systematic Mapping Study of Web Application Testing. *Information and Software Technology.*

(Geras 2012) Geras, A. (2012). Personal communication.

(Graaf et al. 2003) Graaf, B., Lormans, M., & Toetenel, H. (2003). Embedded software engineering: the state of the practice. *Software, IEEE*, *20*(6), 61-69.

(Greene 2004) Greene, B. (2004, June). Agile methods applied to embedded firmware development. *In Agile Development Conference*, 2004 (pp. 71-77). IEEE.

(Grenning 2002) Grenning, J. (2002, March). Extreme programming and embedded software development. *In Embedded Systems Conference* (Vol. 2002).

(Grenning 2007) Grenning, J. (2007). Applying test driven development to embedded software. *Instrumentation & Measurement Magazine, IEEE*, 10(6), 20-25.

(Hassan and Holt 2003) Hassan, A. E., & Holt, R. C. (2003, September). The chaos of software development. In *Software Evolution, 2003. Proceedings. Sixth International Workshop on Principles of* (pp. 84-94). IEEE.

(Huang et al. 2008) Huang, L., Smith, M., Tran, A., & Miller, J. (2008, November). E-RACE, A Hardware-Assisted Approach to Lockset-Based Data Race Detection for Embedded Products. *In Software Reliability Engineering, 2008. ISSRE 2008. 19th International Symposium on* (pp. 277-278). IEEE.

(Johnson 2011) Johnson, N. (2011). Applying Agile to IC Development... We're Not That Different After All. *In CAMUG - Calgary Agile Methods User Group*.

(Jorgensen and Shepperd 2007) Jorgensen, M., & Shepperd, M. (2007). A systematic review of software development cost estimation studies. *Software Engineering, IEEE Transactions on*, 33(1), 33-53.

(Kitchenham and Charters 2007) Kitchenham, B. A., & Charters, S. (2007). Guidelines for performing systematic literature reviews in software engineering.

(Kitchenham et al. 2011) Kitchenham, B. A., Budgen, D., & Pearl Brereton, O. (2011). Using mapping studies as the basis for further research–a participant-observer case study. *Information and Software Technology*, 53(6), 638-651.

(Karlesky et al. 2006) Karlesky, M. J., Bereza, W. I., & Erickson, C. B. (2006, May). Effective test driven development for embedded software. In *Electro/information Technology, 2006 IEEE International Conference on* (pp. 382-387). IEEE.

(Karlesky et al. 2007) Karlesky, M., Williams, G., Bereza, W., & Fletcher, M. (2007, April). Mocking the embedded world: Test-driven development, continuous integration, and design patterns. In *Proc. Emb. Systems Conf, CA, USA*.

(Lamport 1978) Lamport, L. (1978). Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, *21*(7), 558-565.

(Lehman et al. 1997) Lehman, M. M., Ramil, J. F., Wernick, P. D., Perry, D. E., & Turski, W. M. (1997, November). Metrics and laws of software evolution-the nineties view. In *Software Metrics Symposium, 1997. Proceedings., Fourth International* (pp. 20-32). IEEE.

(Liggesmeyer and Trapp 2009) Liggesmeyer, P., & Trapp, M. (2009). Trends in embedded software engineering. *Software, IEEE*, *26*(3), 19-25.

(Llopis and Nicholson 2013) Llopis, N., & Nicholson, C. UnitTest++. [Online]. Available: http://unittest-cpp.sourceforge.net/. [Accessed September 2013].

(Madeyski and Szała 2007) Madeyski, L., & Szała, Ł. (2007). The impact of test-driven development on software development productivity—an empirical study. *In Software Process Improvement* (pp. 200-211). Springer Berlin Heidelberg.

(Maximilien and Williams 2003) Maximilien, E. M., & Williams, L. (2003, May). Assessing test-driven development at IBM. In Software Engineering, 2003. Proceedings. *25th International Conference on* (pp. 564-569). IEEE.

(Mendes 2005) Mendes, E. (2005, November). A systematic review of Web engineering research. *In Empirical Software Engineering, 2005. 2005 International Symposium on* (pp. 10-pp). IEEE.

(Micriμm 2010) Micriμm. (2010). μC/OS-III The Real-Time Kernel User's Manual. [Online]. Available: Micrium-uCOS-III-UsersManual. http://www.analog.com/static/imported-files/software_manuals/Micrium-uCOS-III-UsersManual.pdf. [Accessed 19 November 2013]

(Miller et al. 2006) Miller, J., Smith, M., Daenick, S., Chen, J., Qiao, J., Huang, F., ... & Roper, M. (2006, August). An XP inspired test-oriented life-cycle production strategy for building embedded biomedical applications. *In Testing: Academic and Industrial Conference-Practice And Research Techniques, 2006. TAIC PART 2006. Proceedings* (pp. 95-106). IEEE.

(Maurer and Hellmann 2013) Maurer, F., & Hellmann, T. D. (2013). People-Centered Software Development: An Overview of Agile Methodologies. In *Software Engineering* (pp. 185-215). Springer Berlin Heidelberg.

(Mujtaba et al. 2008) Mujtaba, S., Petersen, K., Feldt, R., & Mattsson, M. (2008). Software product line variability: A systematic mapping study. *School of Engineering, Blekinge Inst. of Technology.*

(Murray et al. 2002) Murray, L., Griffiths, A., Lindsay, P., & Strooper, P. (2002). Requirements Traceability for Embedded Software—an Industry Experience Report. In *Int. Conf. on Software Engineering and Applications* (pp. 374-068).

(Nagappan et al. 2008) Nagappan, N., Maximilien, E. M., Bhat, T., & Williams, L. (2008). Realizing quality improvement through test driven development: results and experiences of four industrial teams. *Empirical Software Engineering*, 13(3), 289-302.

(Netzer and Miller 1992) Netzer, R. H., & Miller, B. P. (1992). What are race conditions?: Some issues and formalizations. *ACM Letters on Programming Languages and Systems (LOPLAS)*, *1*(1), 74-88.

(O'Callahan and Choi 2003) O'Callahan, R., & Choi, J. D. (2003). Hybrid dynamic data race detection. *ACM SIGPLAN Notices*, *38*(10), 167-178.

(Palacios et al. 2011) Palacios, M., García-Fanjul, J., & Tuya, J. (2011). Testing in Service Oriented Architectures with dynamic binding: A mapping study. *Information and Software Technology*, 53(3), 171-189.

(Peckol 2007) Peckol, J. K. (2007). *Embedded systems: a contemporary design tool*. Wiley Publishing.

(Petersen et al. 2008) Petersen, K., Feldt, R., Mujtaba, S., & Mattsson, M. (2008, June). Systematic mapping studies in software engineering. *In 12th International Conference on Evaluation and Assessment in Software Engineering* (Vol. 17, p. 1).

(Pozniansky and Schuster 2003) Pozniansky, E., & Schuster, A. (2003, June). Efficient on-the-fly data race detection in multithreaded C++ programs. In *ACM SIGPLAN Notices* (Vol. 38, No. 10, pp. 179-190). ACM.

(Pozniansky and Schuster 2007) Pozniansky, E., & Schuster, A. (2007). MultiRace: efficient on-the-fly data race detection in multithreaded C++ programs. *Concurrency and Computation: Practice and Experience*, *19*(3), 327-340.

(Rajsuman 1999) Rajsuman, R. (1999). Testing a system-on-a-chip with embedded microprocessor. In *Test Conference, 1999. Proceedings. International* (pp. 499-508). IEEE.

(Rook 1986) Rook, P. (1986). Controlling software projects. *Software Engineering Journal*,*1*(1), 7.

(Sanchez et al. 2007) Sanchez, J. C., Williams, L., & Maximilien, E. M. (2007, August). On the sustained use of a test-driven development practice at ibm. In *Agile Conference (AGILE), 2007* (pp. 5-14). IEEE.

(Shen et al. 2012) Shen, M., Yang, W., Rong, G., & Shao, D. (2012, June). Applying agile methods to embedded software development: A systematic review. *In Software Engineering for Embedded Systems (SEES), 2012 2nd International Workshop on* (pp. 30-36). IEEE.

(Shye et al. 2005) Shye, A., Iyer, M., Reddi, V. J., & Connors, D. A. (2005, September). Code coverage testing using hardware performance monitoring support. In *Proceedings of the sixth international symposium on Automated analysis-driven debugging* (pp. 159-163). ACM.

(Smith et al. 2005) Smith, M., Kwan, A., Martin, A., & Miller, J. (2005). E-TDD–embedded test driven development a tool for hardware-software co-design projects. In *Extreme Programming and Agile Processes in Software Engineering* (pp. 145-153). Springer Berlin Heidelberg.

(Smith et al. 2009a) Smith, M., Miller, J., Huang, L., & Tran, A. (2009). A more agile approach to embedded system development. *Software, IEEE*, 26(3), 50-57.

(Smith et al. 2009b) Smith, M., Miller, J., & Daeninck, S. (2009). A test-oriented embedded system production methodology. *Journal of Signal Processing Systems*, *56*(1), 69-89.

(Smith et al. 2010) Smith, M., Helmi, M., & Miller, J. (2010). Comparison of Approaches to Use Existing Architectural Features in Embedded Processors to Achieve Hardware-Assisted Test Insertion. *Proceedings Work-in-Progress Session*.

(Smith et al. 2011) Smith, M., Beaudin, A., Miller, J., Steinback, C., Islam, S., Poulin, P. (2011). Issues surrounding moving a successful business oriented test driven development process into the environment of real-time embedded systems. *IET Irish Signals and Systems Conference ISSC2011, Dublin, Ireland*. 189–194.

(Smith et al. 2013) Smith, M., Deng, D., Islam, S., & Miller, J. (2013). Enhancing Hardware Assisted Test Insertion Capabilities on Embedded Processors using an FPGA-based Agile Test Support Co-processor. *Journal of Signal Processing Systems*, 1-14.

(Tassey 2002) Tassey, G. (2002). The economic impacts of inadequate infrastructure for software testing. *National Institute of Standards and Technology, RTI Project*, 7007.

(Texas Instruments 2013a) Texas Instruments. (2013). Code Composer Studio (CCStudio) Integrated Development Environment (IDE) v5. [Online]. Available: http://www.ti.com/tool/ccstudio. [Accessed 13 September 2013].

(Texas Instruments 2013b) Texas Instruments. (2013). MSP430™ Ultra-Low-Power Microcontrollers. [Online]. Available: http://www.ti.com/lit/sg/slab034w/slab034w.pdf. [Accessed 24 September 2013].

(Toyota 2010) Is Toyota's Accelerator Problem Caused by Embedded Software Bugs? [Online]. Available: http://embeddedgurus.com/barr-code/2010/01/is-toyotas-accelerator-problem-caused-by-embedded-software-bugs/. [Accessed 20 November 2013].

(Tran et al. 2008) Tran, A., Smith, M., & Miller, J. (2008, November). A hardware-assisted tool for fast, full code coverage analysis. *In Software Reliability Engineering, 2008. ISSRE 2008. 19th International Symposium on* (pp. 321-322). IEEE.

(Whetsel 1997) Whetsel, L. (1997, November). An IEEE 1149.1-based test access architecture for ICs with embedded cores. In *ITC* (pp. 69-78).

(Wieringa et al. 2006) Wieringa, R., Maiden, N., Mead, N., & Rolland, C. (2006). Requirements engineering paper classification and evaluation criteria: a proposal and a discussion. *Requirements Engineering*, 11(1), 102-107.

(Wiederseiner et al. 2011) Wiederseiner, C., Garousi, V., & Smith, M. (2011, November). Tool Support for Automated Traceability of Test/Code Artifacts in Embedded Software Systems. *In Trust, Security and Privacy in Computing and Communications (TrustCom), 2011 IEEE 10th International Conference on* (pp. 1109-1117). IEEE.

(Xie et al. 2012) Xie, M., Shen, M., Rong, G., & Shao, D. (2012, September). Empirical studies of embedded software development using agile methods: a systematic review. *In Proceedings*

*of the 2nd international workshop on Evidential assessment of software technologies* (pp. 21-26). ACM.

(Xilinx 2012) Xilinx. (2012, April). Xilinx. LogiCORE IP MicroBlaze Micro Controller System (v1.1). [Online]. Available: http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_1/ds865_microblaze_mcs.pdf. . [Accessed 24 November 2013].

(Xilinx 2013a) Xilinx. (2013, June). Spartan-3 FPGA Family Data Sheet. Xilinx. [Online]. Available: http://www.xilinx.com/support/documentation/data_sheets/ds099.pdf. [Accessed 19 November 2013].

(Xilinx 2013b) Xilinx. (2013, October). ISE Design Tools. Xilinx. [Online]. Available: http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_7/irn.pdf. [Accessed 19 November 2013]

(Yu et al. 2005) Yu, Y., Rodeheffer, T., & Chen, W. (2005, October). Racetrack: efficient detection of data race conditions via adaptive tracking. In *ACM SIGOPS Operating Systems Review* (Vol. 39, No. 5, pp. 221-234). ACM.

## APPENDIX A: DISCOVERED STUDIES FOR SYSTEMATIC MAPPING STUDY


[S1] Aiello, G., Alessi, M., Bruccoleri, M., D'Onofrio, C., & Vella, G. (2007, June). An agile methodology for manufacturing control systems development. In *Industrial Informatics, 2007 5th IEEE International Conference on* (Vol. 2, pp. 817-822). IEEE.

[S2] Albuquerque, C. O., Antonino, P. O., & Nakagawa, E. Y. (2012). An investigation into agile methods in embedded systems development. In *Computational Science and Its Applications–ICCSA 2012* (pp. 576-591). Springer Berlin Heidelberg.

[S3] Berger, C., & Rumpe, B. (2010). Supporting agile change management by scenario-based regression simulation. *Intelligent Transportation Systems, IEEE Transactions on*, *11*(2), 504-509.

[S4] Boydens, J., Cordemans, P., & Steegmans, E. (2010, May). Test-Driven Development of Embedded Software. In *Proceedings of the Fourth European Conference on the Use of Modern Information and Communication Technologies* (pp. 117-128).

[S5] Cates, J., Weinstein, D., & Davis, M. (2006, April). The Center for Integrative Biomedical Computing: advancing biomedical science with open source. In *Biomedical Imaging: Nano to Macro, 2006. 3rd IEEE International Symposium on* (pp. 694-697). IEEE.

[S6] Cawley, O., Wang, X., & Richardson, I. (2010). Lean/agile software development methodologies in regulated environments–state of the art. In *Lean Enterprise Software and Systems* (pp. 31-36). Springer Berlin Heidelberg.

[S7] Chen, J., Smith, M., Geras, A., & Miller, J. (2006). Making fit/fitnesse appropriate for biomedical engineering research. In *Extreme Programming and Agile Processes in Software Engineering* (pp. 186-190). Springer Berlin Heidelberg.

[S8] Cordeiro, L., Mar, C., Valentin, E., Cruz, F., Patrick, D., Barreto, R., & Lucena, V. (2008, March). A platform-based software design methodology for embedded control systems: An agile toolkit. In *Engineering of Computer Based Systems, 2008. ECBS 2008. 15th Annual IEEE International Conference and Workshop on the* (pp. 408-417). IEEE.

[S9] Cordemans, P., Boydens, J., Van Landschoot, S., & Steegmans, E. (2012). Test-driven development strategies applied to embedded software. In *Proceedings of the fifth european conference on the use of modern information and communication technologies* (pp. 31-42).

[S10] Daeninck, S., Smith, M., Miller, J., & Ko, L. (2006). Extending the embedded system E-TDDunit test driven development tool for development of a real time video security system prototype. In *Extreme Programming and Agile Processes in Software Engineering* (pp. 200-201). Springer Berlin Heidelberg.

[S11] Dahlby, D. (2004). Applying Agile Methods to Embedded Systems Development. *Embedded Software Design Resources*, *41*, 1014123.

[S12] Deng, D., Smith, M., Islam, S., & Miller, J. (2013, June). Prototype test insertion co-processor for agile development in multi-threaded embedded environments. In *Signals and Systems Conference (ISSC 2013), 24th IET Irish*(pp. 1-8). IET.

[S13] Dohmke, T., & Gollee, H. (2007). Test-driven development of a PID controller. *Software, IEEE*, *24*(3), 44-50.

[S14] Dohmke, T. (2008). *Test-driven development of embedded control systems: application in an automotive collision prevention system* (Doctoral dissertation, University of Glasgow).

[S15] Douglass, B. P. (2009). *Real-Time Agility: The Harmony/ESW Method for Real-Time and Embedded Systems Development*. Pearson Education.

[S16] Dowty, M. (2004). Test driven development of embedded systems using existing software test infrastructure. *University of Colorado at Boulder, Tech. Rep., Mar*.

[S17] Eklund, U., & Bosch, J. (2012). Applying Agile Development in Mass-Produced Embedded Systems. In *Agile Processes in Software Engineering and Extreme Programming* (pp. 31-46). Springer Berlin Heidelberg.

[S18] Fletcher, M., Bereza, W., Karlesky, M., & Williams, G. (2007, August). Evolving into Embedded Develop. In *Agile Conference (AGILE), 2007* (pp. 150-155). IEEE.

[S19] Greene, B. (2004, June). Agile methods applied to embedded firmware development. In *Agile Development Conference, 2004* (pp. 71-77). IEEE.

[S20] Grenning, J. (2002, March). Extreme programming and embedded software development. In *Embedded Systems Conference* (Vol. 2002).

[S21] Grenning, J. (2007). Applying test driven development to embedded software. *Instrumentation & Measurement Magazine, IEEE*, *10*(6), 20-25.

[S22] Grenning, J., & Class, E. S. C. (2007). Test driven development for embedded software. *ESC*, *241*, 2.

[S23] Grenning, J. W. (2011). SOLID Design for Embedded C.

[S24] Grenning, J. W., & Carter, J. (2011). *Test-driven development for embedded C*. Pragmatic Bookshelf.

[S25] Gustavsson, H., & Axelsson, J. (2010, July). Improving the system architecting process through the use of Lean tools. In *Technology Management for Global Economic Growth (PICMET), 2010 Proceedings of PICMET'10:* (pp. 1-7). IEEE.

[S26] Gustavsson, H. (2011). *Lean Thinking Applied to System Architecting* (Doctoral dissertation, Mälardalen University).

[S27] Heidenberg, J., Matinlassi, M., Pikkarainen, M., Hirkman, P., & Partanen, J. (2010). Systematic piloting of agile methods in the large: two cases in embedded systems development. In *Product-Focused Software Process Improvement* (pp. 47-61). Springer Berlin Heidelberg.

[S28] Hill, J. H. (2009). *Agile Techniques for Developing and Evaluating Large-scale Component-based Distributed Real-time and Embedded Systems* (Doctoral dissertation, Vanderbilt University).

[S29] Huang, F. (2008). *Test Driven Development for Multithreaded Embedded Systems*. University of Calgary (Canada).

[S30] Ispir, M. (2004). *Test Driven Development of Embedded Systems* (Doctoral dissertation, MIDDLE EAST TECHNICAL UNIVERSITY).

[S31] Ivanov, P., & Ilieva, S. (2007). Agile Case Study Evaluation In Middle Size Project. *Serdica Journal of Computing*, *1*(3), 241p-254p.

[S32] John, B. K., & Roshni, T. Agile Testing In Embedded Systems to Improve Product Quality.

[S33] Juhola, T. (2010). Customized agile development process for embedded software development.

[S34] Kane, D. W., Hohman, M. M., Cerami, E. G., McCormick, M. W., Kuhlmman, K. F., & Byrd, J. A. (2006). Agile methods in biomedical software development: a multi-site experience report. *Bmc Bioinformatics*, *7*(1), 273.

[S35] Karlesky, M. J., Bereza, W. I., & Erickson, C. B. (2006, May). Effective test driven development for embedded software. In *Electro/information Technology, 2006 IEEE International Conference on* (pp. 382-387). IEEE.

[S36] Karlesky, M., Williams, G., Bereza, W., & Fletcher, M. (2007, April). Mocking the embedded world: Test-driven development, continuous integration, and design patterns. In *Proc. Emb. Systems Conf, CA, USA*.

[S37] Kettunen, P., & Laanti, M. (2005). How to steer an embedded software project: tactics for selecting the software process model. *Information and Software Technology*, *47*(9), 587-608.

[S38] Kettunen, P., & Laanti, M. (2008). Combining agile software projects and large-scale organizational agility. *Software Process: Improvement and Practice*,*13*(2), 183-193.

[S39] Kim, H. K. (2008). Aspect oriented testing frameworks for embedded software. In *Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing* (pp. 75-88). Springer Berlin Heidelberg.

[S40] Kumar, M. U., Kumar, D. S., Rao, K. V., MAdhuri, A., Prasad, A. K., & Shravani, D. (2010). Agile Modeling for Security Requirements–Embedded Application Case Study. *International Journal of Engineering Science*, *2*(6), 2014-2019.

[S41] Liu, D., Su, Q., & Xie, Y. (2010, December). A scalable autotest platform for embedded system. In *Apperceiving Computing and Intelligence Analysis (ICACIA), 2010 International Conference on* (pp. 398-401). IEEE.

[S42] Zhang, L. (2003). *Documentation Driven Agile Development for Systems of Embedded Systems*. NAVAL POSTGRADUATE SCHOOL MONTEREY CA SOFTWARE ENGINEERING AUTOMATION CENTER.

[S43] Manhart, P., & Schneider, K. (2004, May). Breaking the ice for agile development of embedded software: An industry experience report. In *Proceedings of the 26th international Conference on Software Engineering* (pp. 378-386). IEEE Computer Society.

[S44] Marchenko, A., & Abrahamsson, P. (2007). Predicting software defect density: a case study on automated static code analysis. In *Agile Processes in Software Engineering and Extreme Programming* (pp. 137-140). Springer Berlin Heidelberg.

[S45] Matthews, C. E. (2011, October). Agile practices in embedded systems. In *Proceedings of the compilation of the co-located workshops on DSM'11, TMC'11, AGERE!'11, AOOPES'11, NEAT'11, & VMIL'11* (pp. 249-250). ACM.

[S46] Matthews, C. E., Douglass, B. P., & Kiekbusch, J. (2011, October). Agile and object oriented practices in embedded systems. In *Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion* (pp. 327-328). ACM.

[S47] Miller, J., Smith, M., Daenick, S., Chen, J., Qiao, J., Huang, F., ... & Roper, M. (2006, August). An XP inspired test-oriented life-cycle production strategy for building embedded biomedical applications. In *Testing: Academic and Industrial Conference-Practice And Research Techniques, 2006. TAIC PART 2006. Proceedings* (pp. 95-106). IEEE.

[S48] Miller, J., & Smith, M. (2007, June). A TDD approach to introducing students to embedded programming. In *ACM SIGCSE Bulletin* (Vol. 39, No. 3, pp. 33-37). ACM.

[S49] Mondragon-Torres, A. F., Kozitsky, A., Bundick, C., Kenna, E. M., Alley, E., Lloyd, M., ... & Lane, R. (2011, October). Work in progress—An agile embedded systems design capstone course. In *Frontiers in Education Conference (FIE), 2011* (pp. F4F-1). IEEE.

[S50] Mueller, G., & Borzuchowski, J. (2002, November). Extreme embedded a report from the front line. In *OOPSLA 2002 Practitioners Reports* (pp. 1-ff). ACM.

[S51] Netkow, M. H., & Brylow, D. (2010, October). Xest: an automated framework for regression testing of embedded software. In *Proceedings of the 2010 Workshop on Embedded Systems Education* (p. 7). ACM.

[S52] Pont, M. J. (2001). Patterns for time-triggered embedded systems. *SMALL,8051*, 41.

[S53] Punkka, T. (2005). Agile Methods and Firmware Development. Available: http://users.tkk.fi/tpunkka/papers/agile_firmware_punkka_V103.pdf

[S54] Punkka, T. (2010). Embedded Agile. In *Embedded Systems Conference 2010*. pp. 1–15.

[S55] Qiao, J., Smith, M., & Miller, J. (2007, April). On Moving Test-Driven Development from the Business World into a Biomedical Engineering Environment. In *Electrical and Computer Engineering, 2007. CCECE 2007. Canadian Conference on* (pp. 1554-1557). IEEE.

[S56] Ronkainen, J., & Abrahamsson, P. (2003). Software development under stringent hardware constraints: do agile methods have a chance?. In *Extreme Programming and Agile Processes in Software Engineering* (pp. 73-79). Springer Berlin Heidelberg.

[S57] Salo, O., & Abrahamsson, P. (2008). Agile methods in European embedded software development organisations: a survey on the actual use and usefulness of Extreme Programming and Scrum. *Software, IET*, *2*(1), 58-64.

[S58] Savolainen, J., Kuusela, J., & Vilavaara, A. (2010, September). Transition to agile development-rediscovery of important requirements engineering practices.

In *Requirements Engineering Conference (RE), 2010 18th IEEE International* (pp. 289-294). IEEE.

[S59] Shen, M., Yang, W., Rong, G., & Shao, D. (2012, June). Applying agile methods to embedded software development: A systematic review. In *Software Engineering for Embedded Systems (SEES), 2012 2nd International Workshop on* (pp. 30-36). IEEE.

[S60] Singh, A. (2012, February). Agile: Analysis of its Problems and their Solutions. In *IJCA Proceedings on International Conference on Recent Advances and Future Trends in Information Technology*. (pp. 32–35).

[S62] Smith, M., Bariffi, M., Flaman, W., Geras, A., Huang, L., Kwan, A., ... & Miller, J. (2005). Practical application for TDD. Part 1. Write software in a test-driven development environment. *Circuit Cellar*, (176), 34-9.

[S61] Smith, M., Banffi, M., Flaman, W., Geras, A., Huang, L., Kwan, A., ... & Miller, J. (2005). Practical application for TDD. Part 2. Automated test-driven environment. *Circuit Cellar*, (177), 60-7.

[S63] Smith, M. R., Miller, J., & Geras, A. (2005, April). A look at test driven development (TDD) in the embedded environment. *Circuit Cellar magazine*. (pp. 34–39).

[S64] Smith, M., Kwan, A., Martin, A., & Miller, J. (2005). E-TDD–embedded test driven development a tool for hardware-software co-design projects. In *Extreme Programming and Agile Processes in Software Engineering* (pp. 145-153). Springer Berlin Heidelberg.

[S65] Smith, M. R., Miller, J., Ko, L., Chen, J., Geras, A., & Frayne, R. (2006). Approaches to validating the" quantity" in quantitative MR cerebral perfusion studies.

[S66] Smith, M., Miller, J., & Daeninck, S. (2008). A test-oriented embedded system production methodology. *Journal of Signal Processing Systems*, *56*(1), 69-89.

[S67] Smith, M., Miller, J., Huang, L., & Tran, A. (2009). A more agile approach to embedded system development. *Software, IEEE*, *26*(3), 50-57.

[S68] Smith, M., Helmi, M., & Miller, J. (2010). Comparison of Approaches to Use Existing Architectural Features in Embedded Processors to Achieve Hardware-Assisted Test Insertion. *Proceedings Work-in-Progress Session*.

[S69] Smith, M., & Miller, J. (2010, January). Experiences in using TDD as a key component in laboratories when teaching hardware / software co-design of embedded systems. In *9th Workshop on Teaching Software Testing*. (pp. 1–4).

[S70] Smith, M., Deng, D., Islam, S., & Miller, J. (2013). Enhancing Hardware Assisted Test Insertion Capabilities on Embedded Processors using an FPGA-based Agile Test Support Co-processor. *Journal of Signal Processing Systems*, 1-14.

[S71] Srinivasan, J., Dobrin, R., & Lundqvist, K. (2009, July). 'State of the Art'in Using Agile Methods for Embedded Systems Development. In *Computer Software and Applications Conference, 2009. COMPSAC'09. 33rd Annual IEEE International*(Vol. 2, pp. 522-527). IEEE.

[S72] Underseth, M. (2007). A test-driven approach to developing embedded software.*EE: Evaluation Engineering*, *46*(4), 44-50.

[S73] Van Schooenderwoert, N., & Consulting, A. R. (2004). Embedded extreme programming: An experience report. In *Embedded Systems Conference*.

[S74] Van Schooenderwoert, N., & Morsicato, R. (2004, June). Taming the embedded tiger-agile test techniques for embedded software. In *Agile Development Conference, 2004* (pp. 120-126). IEEE.

[S75] Van Schooenderwoert, N. (2006, July). Embedded agile project by the numbers with newbies. In *Agile Conference, 2006* (pp. 13-pp). IEEE.

[S76] Wang, Z. (2007, July). Fuxi: An agile development environment for embedded systems. In *Computer Software and Applications Conference, 2007. COMPSAC 2007. 31st Annual International* (Vol. 1, pp. 631-632). IEEE.

[S77] Wilking, D. (2005). The Effects of an Agile Planning Horizon on Embedded Systems Software Development.

[S78] Wilking, D. (2008). *Empirical studies for the application of agile methods to embedded systems* (Doctoral dissertation, Universitätsbibliothek).

## APPENDIX B: DISCOVERED STUDIES ASSOCIATED WITH EACH FIGURE

**2002:** S20, S50, S52
**2003:** S42, S56
**2004:** S11, S16, S19, S30, S43, S73, S74
**2005:** S37, S53, S61, S62, S63, S64, S77
**2006:** S5, S7, S10, S34, S35, S47, S65, S75
**2007:** S1, S13, S18, S21, S22, S31, S36, S44, S48, S55, S72, S76
**2008:** S8, S14, S29, S32, S38, S39, S57, S66, S78
**2009:** S15, S28, S67, S71
**2010:** S3, S4, S6, S25, S27, S33, S40, S41, S51, S54, S58, S68, S69
**2011:** S23, S24, S26, S45, S46, S49
**2012:** S2, S9, S17, S59, S60
**2013:** S12, S70

**Figure 3.1 – Year of publication, showing the number of studies published in each year.**

**02 – 03 VS. Conference Paper:** S20, S42, S50, S52, S56
**04 – 05 VS. Conference Paper:** S16, S19, S43, S53, S64, S73, S74
**04 – 05 VS. Journal Paper:** S37
**04 – 05 VS. Magazine Article:** S61, S62, S63
**04 – 05 VS. Report:** S11, S77
**04 – 05 VS. Thesis:** S30
**06 – 07 VS. Conference Paper:** S1, S5, S7, S10, S18, S22, S35, S36, S44, S47, S55, S65, S75, S76
**06 – 07 VS. Journal Paper:** S13, S21, S31, S34, S48, S72
**08 – 09 VS. Book:** S15
**08 – 09 VS. Conference Paper:** S8, S32, S57
**08 – 09 VS. Journal Paper:** S38, S39, S57, S66, S67
**08 – 09 VS. Thesis:** S14, S28, S29, S78
**10 – 11 VS. Book:** S24
**10 – 11 VS. Conference Paper:** S4, S23, S25, S27, S41, S45, S46, S49, S51, S54, S58, S68, S69
**10 – 11 VS. Journal Paper:** S3, S6, S40
**10 – 11 VS. Thesis:** S26, S33
**12 – 13 VS. Conference Paper:** S9, S12, S17, S59, S60
**12 – 13 VS. Journal Paper:** S2, S70

**Figure 3.2 – Systematic mapping of year facet against the Article Type facet.**

**02 – 03 VS. Evaluation:** S56
**02 – 03 VS. Experience:** S50
**02 – 03 VS. Opinion:** S52
**02 – 03 VS. Solution:** S20, S42
**04 – 05 VS. Evaluation:** S16, S19, S53, S63, S77
**04 – 05 VS. Experience:** S73
**04 – 05 VS. Opinion:** S11
**04 – 05 VS. Solution:** S30, S37, S43, S61, S62, S64, S74
**06 – 07 VS. Evaluation:** S5, S18, S34, S75
**06 – 07 VS. Solution:** S1, S7, S10, S13, S21, S22, S31, S35, S36, S44, S47, S48, S55, S65, S72, S76
**08 – 09 VS. Evaluation:** S14, S57, S78
**08 – 09 VS. Solution:** S8, S15, S28, S29, S32, S38, S39, S66, S67, S71
**10 – 11 VS. Evaluation:** S3, S4, S6, S45, S46, S49, S58
**10 – 11 VS. Experience:** S54, S69
**10 – 11 VS. Solution:** S23, S24, S25, S26, S27, S33, S40, S41, S51, S68
**12 – 13 VS. Evaluation:** S17
**12 – 13 VS. Opinion:** S60
**12 – 13 VS. Philosophical:** S2, S59
**12 – 13 VS. Solution:** S9, S12, S70

**Figure 3.3 – Systematic mapping of year facet against the research type facet.**

**02 – 03 VS. Evaluation result:** S52, S56
**02 – 03 VS. Method:** S20, S42
**02 – 03 VS. Metric:** S50
**04 – 05 VS. Evaluation result:** S11, S16, S19, S53, S63, S77
**04 – 05 VS. Method:** S37, S43
**04 – 05 VS. Metric:** S11
**04 – 05 VS. Model:** S30, S37, S43, S64
**04 – 05 VS. Process:** S16, S30, S61, S62, S64, S73, S74
**04 – 05 VS. Tool:** S30, S63, S64
**06 – 07 VS. Evaluation result:** S18, S34, S75
**06 – 07 VS. Method:** S1, S22, S31, S35, S36, S44, S47, S55, S72, S76
**06 – 07 VS. Model:** S1, S5, S13, S22, S31, S35, S36, S47, S65
**06 – 07 VS. Process:** S5, S13, S18, S21, S48
**06 – 07 VS. Tool:** S1, S5, S7, S10, S13, S35, S36, S47, S48, S55, S72
**08 – 09 VS. Evaluation result:** S8, S14, S28, S38, S57, S78
**08 – 09 VS. Method:** S8, S15, S28, S39, S67, S71

**08 – 09 VS. Model:** S8, S14, S15, S28, S39, S67, S71
**08 – 09 VS. Process:** S14, S29, S32, S38, S66
**08 – 09 VS. Tool:** S14, S15, S28, S29, S66, S67
**10 – 11 VS. Evaluation result:** S3, S4, S6, S24, S26, S45, S46, S49, S58
**10 – 11 VS. Method:** S23, S24, S25, S26, S27, S40, S68
**10 – 11 VS. Metric:** S54
**10 – 11 VS. Model:** S23, S24, S25, S26, S27, S33
**10 – 11 VS. Process:** S33, S51
**10 – 11 VS. Tool:** S24, S41, S51, S68, S69
**12 – 13 VS. Evaluation result: S2**, S12, S17, S59, S60, S70
**12 – 13 VS. Method:** S12, S17, S70
**12 – 13 VS. Metric:** S9
**12 – 13 VS. Model:** S9, S70
**12 – 13 VS. Process:** S9
**12 – 13 VS. Tool:** S12, S70

**Figure 3.4 – Systematic mapping of year facet against the contribution facet.**

**Evaluation VS. Conference paper:** S4, S5, S16, S17, S18, S19, S45, S46, S49, S53, S56, S58, S75
**Evaluation VS. Journal paper:** S3, S6, S34, S57
**Evaluation VS. Magazine article:** S63
**Evaluation VS. Report:** S77
**Evaluation VS. Thesis:** S14, S78
**Experience VS. Conference paper:** S50, S54, S69, S73
**Opinion VS. Conference paper:** S52, S60
**Opinion VS. Report:** S11
**Philosophical VS. Conference paper:** S59
**Philosophical VS. Journal paper:** S2
**Solution VS. Book:** S15, S24
**Solution VS. Conference paper:** S1, S7, S8, S9, S10, S12, S20, S22, S23, S25, S27, S32, S35, S36, S41, S42, S43, S44, S47, S51, S55, S64, S65, S68, S71, S74, S76
**Solution VS. Journal paper:** S13, S21, S31, S37, S38, S39, S40, S48, S66, S67, S70, S72
**Solution VS. Magazine article:** S61, S62
**Solution VS. Thesis:** S26, S28, S29, S30, S33

**Figure 3.5 – Systematic mapping of research type facet against the article type classification facet.**

**Evaluation result VS. Book:** S24
**Evaluation result VS. Conference paper:** S4, S8, S12, S16, S17, S18, S19, S45, S46, S49, S52, S53, S56, S58, S59, S60, S75
**Evaluation result VS. Journal paper:** S2, S3, S6, S34, S38, S57, S70
**Evaluation result VS. Magazine article:** S63
**Evaluation result VS. Report:** S11, S77
**Evaluation result VS. Thesis:** S14, S26, S28, S78
**Method VS. Book:** S15, S24
**Method VS. Conference paper:** S1, S8, S12, S17, S20, S22, S23, S25, S27, S35, S36, S42, S43, S44, S47, S55, S68, S71, S76
**Method VS. Journal paper:** S31, S37, S39, S40, S67, S70, S72
**Method VS. Thesis:** S26, S28
**Metric VS. Conference paper:** S9, S50, S54
**Metric VS. Report:** S11
**Model VS. Book:** S15, S24
**Model VS. Conference paper:** S1, S5, S8, S9, S22, S23, S25, S27, S35, S36, S43, S47, S64, S65, S71
**Model VS. Journal paper:** S13, S31, S37, S39, S67, S70
**Model VS. Thesis:** S14, S26, S28, S30, S33
**Process VS. Conference paper:** S5, S9, S16, S18, S32, S51, S64, S73, S74
**Process VS. Journal paper:** S13, S21, S38, S48, S66
**Process VS. Magazine article:** S61, S62
**Process VS. Thesis:** S14, S29, S30, S33
**Tool VS. Book:** S15, S24
**Tool VS. Conference paper:** S1, S5, S7, S10, S12, S35, S36, S41, S47, S51, S55, S64, S68, S69
**Tool VS. Journal paper:** S13, S48, S66, S67, S70, S72
**Tool VS. Magazine article:** S63
**Tool VS. Thesis:** S14, S28, S29, S30

**Figure 3.6 – Systematic mapping of contribution facet against the article type classification facet.**

**Evaluation:** S3, S4, S5, S6, S14, S16, S17, S18, S19, S34, S45, S46, S49, S53, S56, S57, S58, S63, S75, S77, S78
**Experience:** S50, S54, S69, S73
**Opinion:** S11, S52, S60
**Philosophical:** S2, S59

**Solution:** S1, S7, S8, S9, S10, S12, S13, S15, S20, S21, S22, S23, S24, S25, S26, S27, S28, S29, S30, S31, S32, S33, S35, S36, S37, S38, S39, S40, S41, S42, S43, S44, S47, S48, S51, S55, S61, S62, S64, S65, S66, S67, S68, S70, S71, S72, S74, S76

**Figure 4.1 – Research type facet.**

**Evaluation Result:** S2, S3, S4, S6, S8, S11, S12, S14, S16, S17, S18, S19, S24, S26, S28, S34, S38, S45, S46, S49, S52, S53, S56, S57, S58, S59, S60, S63, S70, S75, S77, S78
**Method:** S1, S8, S12, S15, S17, S20, S22, S23, S24, S25, S26, S27, S28, S31, S35, S36, S37, S39, S40, S42, S43, S44, S47, S55, S67, S68, S70, S71, S72, S76
**Metric:** S9, S11, S50, S54
**Model:** S1, S5, S8, S9, S13, S14, S15, S22, S23, S24, S25, S26, S27, S28, S30, S31, S33, S35, S36, S37, S39, S43, S47, S64, S65, S67, S70, S71
**Process:** S5, S9, S13, S14, S16, S18, S21, S29, S30, S32, S33, S38, S48, S51, S61, S62, S64, S66, S73, S74
**Tool:** S1, S5, S7, S10, S12, S13, S14, S15, S24, S28, S29, S30, S35, S36, S41, S47, S48, S51, S55, S63, S64, S66, S67, S68, S69, S70, S72

**Figure 4.2 – Contribution facet.**

**Studies that have 1 contribution:** S2, S3, S4, S6, S7, S10, S19, S20, S21, S32, S34, S40, S41, S42, S44, S45, S46, S49, S50, S52, S53, S54, S56, S57, S58, S59, S60, S61, S62, S65, S69, S73, S74, S75, S76, S77, S78
**Studies that have 2 contributions:** S11, S16, S17, S18, S22, S23, S25, S27, S29, S31, S33, S37, S38, S39, S43, S48, S51, S55, S63, S66, S68, S71, S72
**Studies that have 3 contributions:** S1, S5, S8, S9, S12, S13, S15, S26, S30, S35, S36, S47, S64, S67
**Studies that have 4 contributions:** S14, S24, S28, S70

**Figure 4.3 – Number of contributions in one study.**

**Evaluation VS. Evaluation result:** S3, S4, S6, S14, S16, S17, S18, S19, S34, S45, S46, S49, S53, S56, S57, S58, S63, S75, S77, S78
**Evaluation VS. Method:** S17
**Evaluation VS. Model:** S5, S14
**Evaluation VS. Process:** S5, S14, S16, S18
**Evaluation VS. Tool:** S5, S14, S63
**Experience VS. Metric:** S50, S54

**Experience VS. Process:** S73
**Experience VS. Tool:** S69
**Opinion VS. Evaluation result:** S11, S52, S60
**Opinion VS. Metric:** S11
**Philosophical VS. Evaluation result**: S2, S59
**Solution VS. Evaluation result:** S8, S12, S24, S26, S28, S38, S70
**Solution VS. Method:** S1, S8, S12, S15, S20, S22, S23, S24, S25, S26, S27, S28, S31, S35, S36, S37, S39, S40, S42, S43, S44, S47, S55, S67, S68, S70, S71, S72, S76
**Solution VS. Metric:** S9
**Solution VS. Model:** S1, S8, S9, S13, S15, S22, S23, S24, S25, S26, S27, S28, S30, S31, S33, S35, S36, S37, S39, S43, S47, S64, S65, S67, S70, S71
**Solution VS. Process:** S9, S13, S21, S29, S30, S32, S33, S38, S48, S51, S61, S62, S64, S66, S74
**Solution VS. Tool:** S1, S7, S10, S12, S13, S15, S24, S28, S29, S30, S35, S36, S41, S47, S48, S51, S55, S64, S66, S67, S68, S70, S72

**Figure 4.4 – Research type facet mapped against the contribution facet.**

**Adapted VS. Solution:** S1, S8, S9, S15, S22, S23, S24, S27, S28, S36, S39, S40, S42, S43, S65, S66, S67, S68, S70, S76
**TDD VS. Evaluation:** S4, S14, S16, S18, S63, S78
**TDD VS. Experience:** S69
**TDD VS. Opinion:** S11
**TDD VS. Philosophical:** S2, S59
**TDD VS. Solution:** S7, S10, S12, S13, S21, S29, S30, S32, S35, S41, S47, S48, S55, S61, S62, S64, S72, S74

**Figure 4.5 – Systematic mapping of research type facet and adapted methods / TDD used facet.**

**Adapted VS. Evaluation result:** S8, S24, S28, S70
**Adapted VS. Method:** S1, S8, S15, S22, S23, S24, S27, S28, S36, S39, S40, S42, S43, S67, S68, S70, S76
**Adapted VS. Metric:** S9
**Adapted VS. Model:** S1, S8, S9, S15, S22, S23, S24, S27, S28, S36, S39, S43, S65, S67, S70
**Adapted VS. Process:** S9, S66
**Adapted VS. Tool:** S1, S15, S24, S28, S36, S66, S67, S68, S70

**TDD VS. Evaluation result:** S2, S4, S11, S12, S14, S16, S18, S59, S63, S78
**TDD VS. Method:** S12, S35, S47, S55, S72
**TDD VS. Metric:** S11
**TDD VS. Model:** S13, S14, S30, S35, S47, S64
**TDD VS. Process:** S13, S14, S16, S18, S21, S29, S30, S32, S48, S61, S62, S64, S74
**TDD VS. Tool:** S7, S10 S12, S13, S14, S29, S30, S35, S41, S47, S48, S55, S63, S64, S69, S72

**Figure 4.6 – Contribution facet maps against adapted methods / TDD used facet.**

**Adapted VS. Acceptance:** S28, S65, S66, S67
**Adapted VS. Build utility:** S24, S36
**Adapted VS. Mock generation:** S1, S36
**Adapted VS. Regression:** S15, S24, S28, S36, S39
**Adapted VS. Unit:** S15, S24, S28, S36, S39, S66, S67, S68
**Adapted VS. Other:** S36, S36, S70
**TDD VS. Acceptance:** S7, S41, S47
**TDD VS. Mock generation:** S72
**TDD VS. Regression:** S14, S41, S72
**TDD VS. Unit:** S10, S13, S14, S16, S18, S29, S30, S35, S41, S48, S55, S63, S64, S69, S72
**TDD VS. Other:** S12, S13

**Figure 4.7 – Systematic mapping of adapted methods / TDD used facet and tool functionality facet.**

**Evaluation VS. Biomedical:** S5, S34
**Evaluation VS. Control systems:** S3, S14
**Evaluation VS. Education:** S49, S77
**Evaluation VS. General:** S4, S6, S16, S17, S18, S19, S45, S46, S53, S56, S57, S58, S63, S75, S78
**Experience VS. Education:** S69
**Experience VS. General:** S50, S54, S73
**Opinion VS. General:** S11, S52, S60
**Philosophical VS. General:** S2, S59
**Solution VS. Biomedical:** S7, S47, S55, S65
**Solution VS. Control systems:** S1, S8, S13, S25, S26
**Solution VS. Education:** S48, S51

**Solution VS. General:** S9, S10, S12, S15, S20, S21, S22, S23, S24, S27, S28, S29, S30, S31, S32, S33, S35, S36, S37, S38, S39, S40, S41, S42, S43, S44, S61, S62, S64, S66, S67, S68, S70, S71, S72, S74, S76

**Figure 4.8 – Systematic mapping of research type facet and domain facet.**

**Evaluation result VS. Biomedical**: S34
**Evaluation result VS. Control systems**: S3, S8, S14, S26
**Evaluation result VS. Education:** S49, S77
**Evaluation result VS. General:** S2, S4, S6, S11, S12, S16, S17, S18, S19, S24, S28, S38, S45, S46, S52, S53, S56, S57, S58, S59, S60, S63, S70, S75, S78
**Method VS. Biomedical:** S47, S55
**Method VS. Control systems:** S1, S8, S25, S26
**Method VS. General:** S12, S15, S17, S20, S22, S23, S24, S27, S28, S31, S35, S36, S37, S39, S40, S42, S43, S44, S67, S68, S70, S71, S72, S76
**Metric VS. General:** S9, S11, S50, S54
**Model VS. Biomedical:** S5, S47, S65
**Model VS. Control systems:** S1, S8, S13, S14, S25, S26
**Model VS. General:** S9, S15, S22, S23, S24, S27, S28, S30, S31, S33, S35, S36, S37, S39, S43, S64, S67, S70, S71
**Process VS. Biomedical:** S5
**Process VS. Control systems:** S13, S14
**Process VS. Education:** S48, S51
**Process VS. General:** S9, S16, S18, S21, S29, S30, S32, S33, S38, S61, S62, S64, S66, S73, S74
**Tool VS. Biomedical:** S5, S7, S47, S55
**Tool VS. Control systems:** S1, S13, S14
**Tool VS. Education:** S48, S51, S69
**Tool VS. General:** S10, S12, S15, S24, S28, S29, S30, S35, S36, S41, S63, S64, S66, S67, S68, S70, S72

**Figure 4.9 – Systematic mapping of contribution facet and domain facet.**

**Evaluation:** S3, S4, S14, S16, S17, S18, S19, S56, S57, S77, S78
**Solution:** S1, S8, S25, S26, S27, S28, S29, S31, S44, S68

**Figure 4.10 – Research type of studies that included case studies.**

**Evaluation result:** S3, S4, S8, S14, S16, S17, S18, S19, S26, S28, S56, S57, S77, S78
**Method:** S1, S8, S17, S25, S26, S27, S28, S31, S44, S68
**Model:** S1, S8, S14, S25, S26, S27, S28, S31
**Process:** S14, S16, S18, S29
**Tool:** S1, S14, S28, S29, S68

**Figure 4.11 – Contribution facet of papers with case studies**