2-1-2012

Schedulability-driven scratchpad memory swapping for resource-constrained real-time embedded systems

Michael De Francis

Follow this and additional works at: http://scholarworks.rit.edu/theses

Recommended Citation

This Thesis is brought to you for free and open access by the Thesis/Dissertation Collections at RIT Scholar Works. It has been accepted for inclusion in Theses by an authorized administrator of RIT Scholar Works. For more information, please contact ritscholarworks@rit.edu.
Schedulability-Driven Scratchpad Memory Swapping for Resource-Constrained Real-Time Embedded Systems

by

Michael P. De Francis

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of Master of Science in Computer Engineering

Supervised by

Dr. Roy Melton
Department of Computer Engineering
Kate Gleason College of Engineering
Rochester Institute of Technology
Rochester, NY
February, 2012

Approved By:

_____________________________________________
Dr. Roy Melton
Primary Advisor – R.I.T. Dept. of Computer Engineering

_____________________________________________
Dr. Jim Vallino
Secondary Advisor – R.I.T. Dept. of Software Engineering

_____________________________________________
Dr. Muhammad Shaaban
Secondary Advisor – R.I.T. Dept. of Computer Engineering
Thesis Release Permission Form

Rochester Institute of Technology
Kate Gleason College of Engineering

Title: Schedulability-Driven Scratchpad Memory Swapping
for Resource-Constrained Real-Time Embedded Systems

I, Michael P. De Francis, hereby grant permission to the Wallace Memorial
Library to reproduce my thesis in whole or part.

_________________________________
Michael P. De Francis

_________________________________
Date
Dedication

To my loving family for all of the support they have provided me

and

To Jennifer Ferrin for her revitalizing words of encouragement
Many thanks to all of my advisors for their guidance and their patience. Their many hours assisting me in the development of ideas has made this research possible. Particular thanks are due to Dr. Melton for his support at every turn.
Abstract

In resource-constrained real-time embedded systems, scratchpad memory (SPM) is utilized in place of cache to increase performance and enforce consistent behavior of both hard and soft real-time tasks via software-controlled SPM management techniques (SPMMTs). Real-time systems depend on time critical (hard) tasks to complete execution before their deadline time. Many real-time systems also depend on the execution of soft tasks that do not have to complete by hard deadlines. This thesis evaluates a new SPMMT that increases both worst-case task slack time (TST) and soft task processing capabilities, by combining two existing SPMMTs.

The schedulability-driven ACETRB / WCETRB swapping (SDAWS) SPMMT of this thesis uses task schedulability characteristics to control the selection of either the average-case execution time reduction based (ACETRB) SPMMT or the worst-case execution time reduction based (WCETRB) SPMMT. While the literature contains examples of combined management techniques, until now there have been none that combine both WCETRB and ACETRB SPMMTs. The advantage of combining them is to achieve WCET reduction comparable to what can be achieved with the WCETRB SPMMT, while achieving significantly reduced ACET relative to the WCETRB SPMMT.

Using a stripped-down RTOS and an SPMMT simulator implemented for this work, evaluated resource-constrained scenarios show a reduction in task slack time from the SDAWS SPMMT relative to the WCETRB SPMMT between 20% and 45%. However, the evaluated scenarios also conservatively show that SDAWS can reduce ACET relative to the WCETRB SPMMT by up to 60%. For the evaluated scenarios, the
smallest task slack time and largest ACET reduction are seen when the SPM swap time (SPMST) to task WCET ratio is minimized. Though a SPMST/WCET ratio of 1:5 or greater reduced slack times under SDAWS versus the ACETRB SPMMT, a SPMST/WCET ratio of 1:100 resulted in slack times up to 200% larger than those under the ACETRB SPMMT. Thus, for systems that can provide small SPMST/WCET ratios, SDAWS can provide significant ACET reduction while maintaining the majority of slack-time assurance provided under WCETRB SPM management.
Contents

List of Figures ix
List of Tables xi
Definition of Terms xiv

CHAPTER 1 – INTRODUCTION
1.1 – WHY SPM INSTEAD OF cache 1
1.2 – EXISTING SPM MANAGEMENT TECHNIQUES 2
1.2.1 The WCET and WCET Reduction-Based SPM Management Techniques 2
1.3 – SCHEDULABILITY-DRIVEN ACETRB / WCETRB SWAP SPMMT 3
1.4 – TESTING SPM MANAGEMENT TECHNIQUE PERFORMANCE 4
1.5 – SPM MANAGEMENT TECHNIQUE RESULTS 5

CHAPTER 2 – PROBLEM STATEMENT
2.1 – PROBLEMS WITH THE WCETRB SPMMT 8
2.2 – THE SOLUTION: ADVANTAGES TO THE SDAWS SPMMT 10
2.3 – PROBLEMS OF SPMMT ANALYSIS 10

CHAPTER 3 – SUPPORTING WORK
3.1 – THE ACET REDUCTION-BASED SPMMT 12
3.1.1 – ACET Reduction-Based SPMMT: Allocation Methods 13
3.1.2 – Some ACET Reduction-Based SPMMT Results 16
3.2 – THE WCET REDUCTION-BASED SPMMT 19
3.2.1 – WCET Reduction-Based SPMMT: Allocation Methods 19
3.2.2 – Some WCET Reduction-Based SPMMT Results 22
3.3 – SCRATCHPAD MEMORY VERSUS cache 25
3.3.1 – Cache-Only Solutions Versus Static SPM Allocation 26
3.3.2 – SPM-Only Solutions and Dynamic SPM Management 28
3.3.3 – A Mixture of SPM and Cache 29
3.4 – RTOS / SYSTEM-DEPENDENT SPM ALLOCATION 30
3.4.1 – Stack Versus Heap SPM Allocation 30
3.4.2 – Preemption Versus Non-Preemption 36
3.4.3 – Static Versus Dynamic SPM Allocation 38
3.5 – REAL-TIME SCHEDULING AND SCHEDULABILITY ANALYSIS 39
3.5.1 – Rate Monotonic Analysis 39
3.5.2 – Fixed-Priority Scheduling 43
3.5.3 – Earliest Deadline First Scheduling 44

CHAPTER 4 – METHODOLOGY
4.1 – DESIGN METHODOLOGY 47
4.1.1 – Development of the SDAWS Threshold Detection Algorithm 47
4.1.2 – Design of the RTOS 51
4.1.3 – Design of the SPMMT Simulator 61
4.2 – VALIDATION METHODOLOGY 71
4.2.1 – RTOS Validation 72
4.2.2 – SPMMT Simulator Validation 76
4.3 – EXPERIMENT METHODOLOGY 81
4.3.1 – The One-Task Experiment Methodology 82
4.3.2 – The Multi-Task Experiment Methodology 85
CHAPTER 5 – RESULTS AND ANALYSIS 91
  5.1 – INTRODUCTION 92
    5.1.1 – Metric Definitions 93
    5.1.2 – SPMMT-Independent Simulation Parameters 97
    5.1.3 – Experiment List and Brief Summary of Findings 102
  5.2 – COMPARISON OF THE ACETRB AND WCETRB SPMMTS 104
    5.2.1 – Analysis of Non-Avertable and Avertable Cache Block Loads 105
    5.2.2 – Analysis of ACETRB Average Execution Time Reduction 111
    5.2.3 – Analysis of Worst-Case Processor Utilization 116
  5.3 – THE EFFECT OF OVERHEAD ON SDAWS PERFORMANCE 123
    5.3.1 – SDAWS Results Summary 124
    5.3.2 – Comparison of RTOS Overhead for Different SPMMTs 128
    5.3.3 – The Effect of The SPMST to WCET Ratio on SDAWS 131
    5.3.4 – The Effect of Desired Deadlines on SDAWS 142
CHAPTER 6 – CONCLUSION 147
CHAPTER 7 – CONSIDERATIONS FOR FUTURE WORK 151
  6.1 – ADDITIONAL SDAWS SPMMT TESTING 151
    6.1.1 – Additional Simulation Testing with The RTOS and SPMMT Simulator 151
    6.1.2 – Comparison of SDAWS to other SPMMTs 152
    6.1.3 – Application Testing of SDAWS 152
  6.2 – SDAWS IMPLEMENTATION OPTIMIZATION 153
    6.2.1 – Schedulability Analysis Optimization 153
    6.2.2 – SPM Swap Optimization 155
    6.2.3 – Alternate RTOS Configurations 156
  6.3 – DEVELOPMENT OF A BLACK-BOX SOLUTION 157
    6.3.1 – Compile-Time SDAWS SPM Allocator 158
    6.3.2 – Debugger-Based Automated Profiling Tool For SDAWS Allocation 158
    6.3.3 – Debugger-Based Cache Block Load Analyzer 159
BIBLIOGRAPHY 160
Appendix A 162
Appendix B 173
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ACET Reduction-Based SPM Allocation: Method Flow</td>
</tr>
<tr>
<td>2</td>
<td>SPM Allocation Via Profiling [19]</td>
</tr>
<tr>
<td>3</td>
<td>Multiple Benchmarks – Normalized Processor Cycles [3]</td>
</tr>
<tr>
<td>4</td>
<td>Beamformer Benchmark – Normalized Processor Cycles [3]</td>
</tr>
<tr>
<td>5</td>
<td>WCET Reduction-Based SPM Allocation: Method Flow</td>
</tr>
<tr>
<td>6</td>
<td>WCET Benchmarks and Allocation Method Performance for [8]</td>
</tr>
<tr>
<td>7</td>
<td>Idle Runtime in Seconds over a Five Minute Period</td>
</tr>
<tr>
<td>8</td>
<td>Core RTOS Event Handling Task WCET in microseconds</td>
</tr>
<tr>
<td>9</td>
<td>Example Task ACET – Task Stack Moved into SPM in Rev 2</td>
</tr>
<tr>
<td>10</td>
<td>ACET by Computations Per Memory Access (10,000 Memory Accesses)</td>
</tr>
<tr>
<td>11</td>
<td>ACET by SRAM to SDRAM Ratio (1k MAs, 2 Average-Case CBLs)</td>
</tr>
<tr>
<td>12</td>
<td>WCET by SRAM to SDRAM Ratio (1k MAs, 4 Worst-Case CBLs)</td>
</tr>
<tr>
<td>13</td>
<td>ACET and WCET Reduction for Corresponding SPMMTs (average/worst-case non-avertable CBLs)</td>
</tr>
<tr>
<td>14</td>
<td>ACET and WCET Decrease for Corresponding SPMMTs (via average/worst-case avertable CBLs)</td>
</tr>
<tr>
<td>15</td>
<td>ACET (average/worst-case avertable CBLs)</td>
</tr>
<tr>
<td>16</td>
<td>WCET (average/worst-case avertable CBLs)</td>
</tr>
<tr>
<td>17</td>
<td>Average Task Execution Time According to WCET to ACET Ratio</td>
</tr>
<tr>
<td>18</td>
<td>Percent Average Execution Time for the ACETRB SPMMT over the WCETRB SPMMT for Different WCET to ACET Ratios</td>
</tr>
<tr>
<td>19</td>
<td>Average Task Execution Time According to Average Cases per Worst-Case</td>
</tr>
<tr>
<td>20</td>
<td>Percent Average Execution Time for the ACETRB SPMMT over the WCETRB SPMMT for Different Average Cases per Worst-Case</td>
</tr>
<tr>
<td>21</td>
<td>Percent Slack Time Increase for the WCETRB SPMMT over the ACETRB SPMMT for Different Worst-Case Processor Utilizations (via Contrast Periods and Task Runtimes)</td>
</tr>
<tr>
<td>22</td>
<td>Percent Slack Time Increase for the WCETRB SPMMT over the ACETRB SPMMT for Different Worst-Case Processor Utilizations (via Numbers of Identical Tasks)</td>
</tr>
<tr>
<td>23</td>
<td>RTOS % Overhead for Aperiodic Disturbance Tests: Task Periodicity = “T1 Repetition Period”</td>
</tr>
<tr>
<td>24</td>
<td>SDAWS % Slack Time Increase Versus ACETRB for SPMST:WCET Equal To 1:3</td>
</tr>
<tr>
<td>25</td>
<td>Total WCET (Including SPM Swap Time) for SPMST:WCET Equal To 1:3</td>
</tr>
<tr>
<td>26</td>
<td>Aperiodic Distortion – SDAWS % Slack Time Improvement Over ACETRB for Memory Swap Times</td>
</tr>
<tr>
<td>27</td>
<td>Aperiodic Distortion – Total WCETs (Including SPM Swap Time) for Memory Swap Times</td>
</tr>
<tr>
<td>28</td>
<td>Aperiodic Distortion — Worst-Case Slack Times for Memory Swap Times</td>
</tr>
<tr>
<td>29</td>
<td>Aperiodic Distortion – Worst-Case Processor Utilization for Memory Swap Times</td>
</tr>
</tbody>
</table>
Figure 30: Identical Tasks w/ Different Periodicity – SDAWS Slack Time Improvement for Desired Deadlines ..................................................................................................................................... 146
Figure 31 – ACET for Computations per Memory Access (1,000 Memory Accesses) ................. 162
Figure 32 – ACET (average/worst-case non-avertable CBLs) .......................................................... 165
Figure 33 – WCET (average/worst-case non-avertable CBLs) .......................................................... 165
List of Tables

Table 1: Summary of Findings ......................................................................................................... 5
Table 2 – WCET Benchmark Parameters for [8] ........................................................................... 23
Table 3: Experiment Parameters: Memory Accesses ..................................................................... 97
Table 4 – Task Execution Time due to Memory Accesses (All variables in SPM) ......................... 98
Table 4: Experiment Parameters: Computations Per Memory Access ........................................ 98
Table 5 – Computations per Memory Access (10,000 Memory Accesses) ................................. 99
Table 6: Experiment Parameters: Memory Accesses .................................................................... 97
Table 7: Experiment Parameters: SRAM to SDRAM Memory Access Ratio ............................. 100
Table 8 – ACETRB SPMMT (2 Avertable CBLs for Average-Case, 4 Avertable CBLs for Worst-Case) with Average-Case Accesses: Worst-Case Accesses = 1000:2000 .............................................. 106
Table 9 – WCETRB SPMMT (2 Avertable CBLs for Average-Case, 4 Avertable CBLs for Worst-Case) with Average-Case Accesses: Worst-Case Accesses = 1000:2000 .............................................. 106
Table 10: Experiment Parameters: Avertable CBLs ..................................................................... 108
Table 11 – ACETRB SPMMT (2 Non-Avertable CBLs for Average-Case, 4 Non-Avertable CBLs for Worst-Case) with Average-Case Accesses: Worst-Case Accesses = 1000:2000 ......................... 108
Table 12 – WCETRB SPMMT (2 Non-Avertable CBLs for Average-Case, 4 Non-Avertable CBLs for Worst-Case) with Average-Case Accesses: Worst-Case Accesses = 1000:2000 ......................... 108
Table 13: Experiment Parameters: WCET:ACET Ratio ................................................................. 112
Table 14: Experiment Parameters: Average-Cases Per Worst-Case ........................................... 114
Table 15: Experiment Parameters: Worst-Case Processor Utilization Via Task WCETs ............... 117
Table 16: ACETRB SPMMT – Worst-Case Processor Utilization via Varied Task Repetition Periods ..................................................................................................................................................... 118
Table 17: WCETRB SPMMT – Worst-Case Processor Utilization via Varied Task Repetition Periods ..................................................................................................................................................... 118
Table 18: Experiment Parameters: Worst-Case Processor Utilization Via Task WCETs ............... 118
Table 19: ACETRB SPMMT – Worst-Case Processor Utilization via Varied Task Execution Times ..................................................................................................................................................... 119
Table 20: WCETRB SPMMT – Worst-Case Processor Utilization via Varied Task Execution Times ..................................................................................................................................................... 119
Table 21: Experiment Parameters: Worst-Case Processor Utilization Via Varied Numbers of Tasks ..................................................................................................................................................... 121
Table 22: ACETRB SPMMT – Worst-Case Processor Utilization via Number of Tasks ................. 122
Table 23: WCETRB SPMMT – Worst-Case Processor Utilization via Number of Tasks .......... 122
Table 24: Experiment Parameters: Aperiodic Disturbance Test ................................................. 128
Table 25 – ACETRB SPMMT – RTOS Overhead Percentage For Aperiodic Disturbance Tests .... 129
Table 26 – WCETRB SPMMT – RTOS Overhead Percentage For Aperiodic Disturbance Tests ... 129
Table 27 – SDAWS SPMMT – RTOS Overhead Percentage For Aperiodic Disturbance Tests ....... 130
Table 28: Experiment Parameters: Worst-Case Processor Utilization For Fixed Desired Deadline ..................................................................................................................................................... 132
Table 29: Two Identical Tasks: SDAWS Performance For SPMST:WCET Equal To 1:6 ............... 133
Table 30: Experiment Parameters: Aperiodic Disturbance Test with Varied SPMST:WCET Percentage

Table 31: Aperiodic Distortion Results – Different Memory Swap Times

Table 32: Aperiodic Distortion – SRAM Priority To Aperiodic Tasks

Table 33: Aperiodic Distortion – SRAM Priority To Periodic Tasks

Table 34: Experiment Parameters: Task WCET For Fixed Desired Deadline

Table 35: Two Identical Tasks – Desired Deadlines Equal To Task Deadlines (Periodicity)

Table 36: Desired Deadline Experiment Parameters

Table 37: Five Identical Tasks w/ Different Periodicity – Desired Deadlines

Table 38: Summary of Findings

Table 40 – Computations per Memory Access (1,000 Memory Accesses)

Table 41 – SRAM to SDRAM Access Ratio (1,000 Memory Accesses)

The following tables show the results of CBL experiments in which average-case CBLs are equal to worst-case CBLs, and there are an identical number of worst-case and average-case memory accesses. Non-avertable and avertable CBLs were varied. The WCETRB SPMMT was not tested because the worst-case and average-case execution paths are identical; therefore the WCETRB SPMMT performs identically to the ACETRB SPMMT.

Table 42 – Non-avertable CBLs for the ACETRB SPMMT (2 AC:WC CBLs = 1:1) with Average-Case Accesses: Worst-Case Accesses = 1000:1000

Table 43 – Avertable CBLs for the ACETRB SPMMT (2 AC:WC CBLs = 1:1) with Average-Case Accesses: Worst-Case Accesses = 1000:1000

The following tables show the results of CBL experiments in which avertable average-case and avertable worst-case CBLs were modified, but there are an identical number of worst-case and average-case memory accesses.

Table 44 – AC Avertable CBLs for the ACETRB SPMMT with Average-Case Accesses: Worst-Case Accesses = 1000:1000

Table 45 – WC Avertable CBLs for the ACETRB SPMMT with Average-Case Accesses: Worst-Case Accesses = 1000:1000

Table 46 – ACET:WCET Ratio for SDAWS

Table 47: ACETRB SPMMT – Varied WCET to ACET Ratio

Table 48: WCETRB SPMMT – Varied WCET to ACET Ratio

Table 49: ACETRB SPMMT – Varied Average Cases Per Worst-Case

Table 50: WCETRB SPMMT – Varied Average Cases Per Worst-Case

Table 51 – Average-Cases Per Worst-Case for SDAWS

Table 52 – ACETRB SPMMT: Average and Worst-Case Task Slack Times for the Processor Utilization by Task Numbers Experiment

Table 53 – WCETRB SPMMT: Average and Worst-Case Task Slack Times for the Processor Utilization by Task Numbers Experiment

Table 54 – ACETRB and WCETRB Comparison for Processor Utilizations, Task WCETs, and Task Periodicities for Aperiodic Disturbance (No Clear Trends – Only General Statements About Relative Performance Can Be Made)

Table 55: Aperiodic Distortion Results – Different Task Sizes
Table 56 – SDAWS SPMMT Task Data Set: SPM Swap Times for 97% WC Processor Utilization via Aperiodic Disturbance .......................................................................................................................... 171
Table 57 – ACETRB, WCETRB, SDAWS SPMMTs – Task Data Set: Desired Deadline Comparison For 97% WC Processor Utilization via the Aperiodic Disturbance Test (1:12 SPMST:WCET Ratio) .................................................................................................................................................. 172
**Definition of Terms**

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMCRTOS</td>
<td>Incredibly Minimalist C Real-Time Operating System</td>
</tr>
<tr>
<td>RTOS</td>
<td>Real-Time Operating System</td>
</tr>
<tr>
<td>SPM</td>
<td>Scratch-Pad Memory</td>
</tr>
<tr>
<td>SPMA</td>
<td>SPM Allocation</td>
</tr>
<tr>
<td>SPM Management</td>
<td></td>
</tr>
<tr>
<td>SPM MM</td>
<td>SPM Management Technique</td>
</tr>
<tr>
<td>SPM MM TS</td>
<td>SPM MM Simulator</td>
</tr>
<tr>
<td>SPM ST</td>
<td>SPM Swap Time</td>
</tr>
<tr>
<td>RAM</td>
<td>Random Access Memory</td>
</tr>
<tr>
<td>SRAM</td>
<td>Static RAM</td>
</tr>
<tr>
<td>SDRAM</td>
<td>Synchronous Dynamic RAM</td>
</tr>
<tr>
<td>DDR SDRAM</td>
<td>Dual Data Rate SDRAM</td>
</tr>
<tr>
<td>CPU</td>
<td>Central Processing Unit</td>
</tr>
<tr>
<td>RT</td>
<td>Real-Time</td>
</tr>
<tr>
<td>ACET</td>
<td>Average-Case Execution Time</td>
</tr>
<tr>
<td>WCET</td>
<td>Worst-Case Execution Time</td>
</tr>
<tr>
<td>Schedulability</td>
<td>The ability of tasks in an RT system to meet their desired deadlines</td>
</tr>
<tr>
<td>DCT</td>
<td>Desired Completion Time</td>
</tr>
<tr>
<td>EDF</td>
<td>Earliest Deadline First</td>
</tr>
<tr>
<td>FP</td>
<td>Fixed Priority</td>
</tr>
<tr>
<td>RMA</td>
<td>Rate Monotonic Analysis</td>
</tr>
<tr>
<td>DMA</td>
<td>Direct Memory Access</td>
</tr>
<tr>
<td>TRT</td>
<td>Task Response Time</td>
</tr>
<tr>
<td>PSA</td>
<td>Point-Schedulability Analysis</td>
</tr>
<tr>
<td>PSAT</td>
<td>Point-Schedulability Analysis Time</td>
</tr>
<tr>
<td>WCET RB</td>
<td>WCET Reduction-Based</td>
</tr>
<tr>
<td>ACET RB</td>
<td>ACET Reduction-Based</td>
</tr>
<tr>
<td>SDAWS</td>
<td>Schedulability-Driven ACET RB/WCET RB Swapping</td>
</tr>
<tr>
<td>SDAWS TDA</td>
<td>SDAWS Threshold Detection Algorithm</td>
</tr>
<tr>
<td>ILP</td>
<td>Integer Linear Programming</td>
</tr>
<tr>
<td>C per MA</td>
<td>Computations per Memory Access</td>
</tr>
<tr>
<td>WC</td>
<td>Worst-case</td>
</tr>
<tr>
<td>AC</td>
<td>Average-case</td>
</tr>
<tr>
<td>MAR</td>
<td>Memory Access Ratio</td>
</tr>
<tr>
<td>uS</td>
<td>Microseconds</td>
</tr>
</tbody>
</table>
CHAPTER 1 – INTRODUCTION

Embedded processors are often packaged with scratchpad memory (SPM) integrated onto the chip. SPM can reduce execution times and provide more visibility than cache under various scenarios, as using SPM allows engineers to know for sure what variables are loaded in fast memory, (though this depends on the software management strategy employed). Further, SPM has a notably smaller size footprint than cache, faster access times, and significantly reduced power consumption [7, 15]. The trade-off is that SPM must be controlled by software.

1.1 – WHY SPM INSTEAD OF cache?

In general purpose computing, caches are the primary form of on-chip memory. The main advantage of cache is its automatic management by hardware. However, in embedded systems, engineers must be able to determine the worst-case execution time (WCET) for any hard RT tasks. WCET is defined as the maximum time a task may take to run for a given system, and depends on such factors as the hardware platform on which is it running, the worst-case amount of data that the task will have to process, and the number of tasks or interrupt routines that may preempt its execution. The non-deterministic nature of cache leads to difficulties determining task WCET, because at any given point in time, it is exceedingly difficult to guarantee that particular variables will be in cache.

SPM in the form of on-chip static RAM (SRAM) is popular because of a smaller size footprint on the chip, faster memory access times, and reduced power consumption versus cache [7, 15]. A primary disadvantage of SPM is that in order to use it efficiently, the engineer must choose an effective SPM management technique (SPMMT), and develop software to implement the SPMMT, which may take a lot of time. However, a primary advantage of software control is
that, because one may specify exactly which variables will go into SPM, variables in SPM can achieve a greater hit rate than had they been loaded into main memory / cache. For these reasons, much research has targeted determining an optimal SPMMT for allocating stack and heap variables into SPM.

1.2 – EXISTING SPM MANAGEMENT TECHNIQUES

Research has identified several SPM management techniques (SPMMTs) that have been tested in industry and in academia. All of the SPMMTs investigated by this thesis assume that there exists an on chip memory architecture incorporating 50% cache and 50% SPM. The following SPMMTs were researched for this thesis:

1. Static and dynamic ACET reduction-based allocation [1,2,3,4,5]
2. Static and dynamic WCET reduction-based allocation [7,8]
3. Static and dynamic stack and heap allocation of data [17,20]
4. Allocating of code and data for reduced power consumption [3]
5. Static allocation of code [21]
6. Static allocation of RTOS parameters [29]

This thesis utilized two of these techniques to develop the schedulability-driven SPMMT. This Chapter briefly mentions the two SPMMTs that are relevant to this thesis, when they are valuable, and why they are necessary. Some qualities of these and other SPMMTs include static versus dynamic SPM allocation, RTOS / ISR / message passing parameter SPM allocation, stack versus heap SPM allocation, and ACET versus WCET reduction. These qualities of SPMMTs will be covered in more detail in the Supporting Work Chapter.

1.2.1 –The WCET and ACET Reduction Based SPM Management Techniques
For real-time (RT) systems, worst-case execution time (WCET) reduction is desirable to aid tasks in meeting their deadlines in the worst-case; however, for RT systems, it is also desirable to reduce average-case execution time (ACET) in order to achieve better task throughput. The WCET reduction based (WCETRB) SPMMT and ACET reduction based (ACETRB) SPMMT address conflicting needs [3,8,9]. While WCET reduction is desirable in order to prevent tasks from missing deadlines, ACET reduction is desirable in order to increase overall processing power.

A majority of SPMMT research has focused on using SPM to reduce ACET, either statically (at compile-time) or dynamically (on-the-fly) [3,5,6,7,10,11]. This can be done either by using the access frequencies of variables obtained through profiling, or through analyses performed at compile time. Though the same profiling or compile time analysis can be used for WCET reduction SPMMTs, reducing WCET is more complex. This is because it involves looking at those variables that are accessed only along the worst-case path (WCP). Research into WCETRB SPMMTs includes [2,8,9,12].

1.3 – SCHEDULABILITY-DRIVEN ACETRB / WCETRB SWAP SPMMT

The new SPMMT developed for this thesis is named “the schedulability-driven ACETRB / WCETRB swap SPMMT (SDAWS SPMMT)”. Until now, no method has been proposed that takes advantage of both WCETRB and ACETRB SPM allocation (SPMA). This thesis certifies that, given certain RT task parameters (such as task memory usage versus total SPM size, AC path/WC path access overlap, etc.), it is possible for a single program to switch between static ACETRB and static WCETRB SPMA according to real-time scheduling conditions.
This thesis performs ACETRB / WCETRB swapping in such a way that reduces overall execution time of tasks versus static WCET SPM allocation, while maintaining similar worst-case task slack-time as static WCET SPM allocation. This is possible through task completion point and task scheduling point analysis. Task desired deadlines (% of task deadline time) and the SPMA swap time are incorporated into the algorithm that detects if an SPMA swap should be made. This work assumes that a 50% cache, 50% SPM memory architecture is utilized by the MCU.

1.4 – TESTING SPM MANAGEMENT TECHNIQUE PERFORMANCE

A very small (minimalist) RTOS was developed for this thesis. This RTOS incorporates fixed priority task switching and automated timing analysis. In addition an SPMMT simulator was developed capable of testing the WCETRB, ACETRB, and SDAWS SPMMTs. The SPMMT simulator incorporates a task simulator that allows for worst-case and average-case cache block loads (CBLs), memory accesses, computations per memory access, and other task parameters to be specified. Notably, the simulator allows task periodicity and average-case path per worst-case path to be specified.

The simulator was then run for each of a multitude of specified task scenarios, for each SPMMT. Task slack time (TST) and task execution time are used to measure SPMMT performance. Task slack time is the amount of time between when a task has completed execution and that task’s deadline time. Specifically, differences in ACET, WCET, worst-case TST, and average TST between the different SPMMTs are measured for each test scenario. These results are compared to published results for the ACETRB and WCETRB SPMMTs.
Differences between SDAWS, ACETRB and WCETRB SPMMT performance are highly dependent upon system conditions. However, because SDAWS requires additional RTOS overhead in order to operate, some system conditions affect SDAWS performance but not ACETRB or WCETRB performance. Table 1 (below) summarizes the findings obtained for the SPMMTs under different simulated system conditions.

Table 1: Summary of Findings

<table>
<thead>
<tr>
<th>System Conditions</th>
<th>SPMMT</th>
<th>SPMMT Performance Characterization</th>
</tr>
</thead>
<tbody>
<tr>
<td>High task frequency</td>
<td>SDAWSs</td>
<td>Weak performance due to high overhead</td>
</tr>
<tr>
<td>Low task frequency</td>
<td>SDAWS</td>
<td>Strong performance due to low overhead</td>
</tr>
<tr>
<td>High SPM Swap Time : WCET Ratio *</td>
<td>SDAWS</td>
<td>Weak performance due to high overhead</td>
</tr>
<tr>
<td>Low SPM Swap Time : WCET Ratio *</td>
<td>SDAWS</td>
<td>Strong performance due to low overhead</td>
</tr>
<tr>
<td>SPMST + PSAT *** &lt; SDAWS’ WCET Improvement</td>
<td>SDAWS</td>
<td>SDAWS slack times &gt; ACETRB slack times. *</td>
</tr>
<tr>
<td>SPMST + PSAT *** &gt; SDAWS’ WCET Improvement</td>
<td>SDAWS</td>
<td>SDAWS slack times &lt; ACETRB slack times. SDAWS cannot be used for the RT system. *</td>
</tr>
<tr>
<td>High Worst-Case Processor Utilization Percentage *</td>
<td>SDAWS, WCETRB</td>
<td>Strong slack time improvements versus ACETRB*</td>
</tr>
<tr>
<td>Low Worst-Case Processor Utilization Percentage *</td>
<td>SDAWS, WCETRB</td>
<td>Weak slack time improvements versus ACETRB *</td>
</tr>
<tr>
<td>Small / infrequent worst-case path execution</td>
<td>ACETRB</td>
<td>Strong average execution time reduction versus WCETRB*</td>
</tr>
<tr>
<td>Large / frequent worst-case path execution</td>
<td>ACETRB</td>
<td>Weak average execution time reduction versus WCETRB*</td>
</tr>
<tr>
<td>Large number of avertable cache block loads per execution path **</td>
<td>ACETRB, WCETRB, SDAWS</td>
<td>Significant execution path reduction due to SPM allocation</td>
</tr>
<tr>
<td>Small number of avertable cache block loads per execution path **</td>
<td>ACETRB, WCETRB, SDAWS</td>
<td>Insignificant execution path reduction due to SPM allocation</td>
</tr>
</tbody>
</table>

* See Chapter 5, section 5.1.1
** See Chapter 5, section 5.3.2
*** See Chapter 5, section 5.3.3
This thesis shows that ACET reductions under the ACETRB SPMMT and WCET reductions under the WCETRB SPMMT range between 10% and 60% for the tested scenarios, where the amount of reduction is dependent upon the cache block load to execution time ratio. Due to WCET reductions, use of the WCETRB SPMMT results in a minimum of 10% reduction in worst-case processor utilization under tested heavily resource-constrained conditions.

This thesis demonstrates that the SDAWS SPMMT performs marginally worse than the WCETRB SPMMT under aperiodic disturbance scenarios unless the SDAWS SPMA swap drastically affects performance. However, SDAWS can be used in conjunction with task scheduling point and completion point analysis, which requires additional overhead. For tasks with extremely small execution times relative to the completion point / scheduling point scan times, SDAWS cannot be used. Furthermore, for tasks with very large memory spaces relative to their execution times (in which SPM swap time is large relative to task execution times), SDAWS achieves worse performance than both WCETRB and ACETRB SPMA. However, for tasks with relatively small memory spaces (in which SPMA swap time is insignificant relative to task execution time) and for tasks with large execution times, SDAWS achieves roughly equivalent WCETs as WCETRB SPMA, with greatly improved ACETs. For systems dominated by large aperiodic disturbances, SDAWS shows reduced WCETs versus WCETRB SPMA; while for systems dominated by very small aperiodic disturbances, WCETRB SPMA shows reduced WCETs.
CHAPTER 2 – PROBLEM STATEMENT

Despite often having strict (hard) deadline requirements, real-time systems often run tasks with scaling (soft) deadline requirements. The WCETRB SPMMT increases a task’s ability to meet hard deadlines in the worst-case [7], while the ACETRB SPMMT increases overall system performance, granting tasks with soft deadline requirements additional processor time [2]. For systems with both hard and soft task types, the problem remains: how does an engineer choose between ACET or WCET reduction?

It is often desirable to complete such soft tasks as soon as possible, though a later completion time is acceptable [2]. One example of this scenario might be data transfer over Ethernet for an online console game. While there is an acceptable lag tolerance, it is desirable to handle network communication tasks as quickly as possible in order to provide the user with an optimal experience. In this scenario, WCET is important, because there is a hard real-time deadline associated with tasks; but, ACET is important as well, because reducing the ACET of network tasks will result in a better user experience. For applications such as this one, it is desirable to obtain the advantages of both the ACETRB and the WCETRB SPMMT. Another example of a scenario in which it is advantageous to reduce both ACET and WCET would be a system that has tasks with hard deadlines, but in which idle time is utilized to perform system maintenance and diagnostics. Reduced ACET results in increased idle time, while reduced WCET leads to increased ability to meet task deadlines.

A large number of embedded applications are data-driven [2], so it is valuable to investigate the effects of SPM allocation on these kinds of systems. However, for these systems, in order to determine the optimal SPM allocation, all data must be considered, including those
required by RTOS overhead (interrupt handlers, task switching, scheduling, memory management, error checking, etc). Furthermore, for RTOSes that run aperiodic tasks that share heap variables with periodic tasks, the lifetime of heap variables depends on what tasks are running. The RTOS itself may introduce considerable overhead due to task scheduling, context switching, memory management between tasks, communication between tasks, etc. For these systems, any aperiodic events will upset the performance of the system if SPM does not have allocated to it variables associated with aperiodic events that may occur. Aperiodic events may originate from the RTOS, (in the form of garbage collection routines, status update routines, etc). Furthermore, aperiodic events may also be caused by aperiodic tasks. Any complete SPM allocation strategy for data-driven real-time systems should include analysis of all aperiodic events, and the worst-case combinations of aperiodic events that can occur.

The simplest answer – to the question of WCET reduction versus ACET reduction – is to choose WCET reduction only when it is necessary in order for tasks to meet their deadlines in the worst-case; however, it is not the best answer. This is because, for systems that are barely schedulable under the ACETRB SPMMT in the worst-case, it may be desirable to use the WCETRB SPMMT in order to ensure schedulability, especially as development continues and task execution times are expanded. Furthermore, for systems that need WCETRB SPMA in order to meet task deadlines, this may only be true in a worst-case task stack-up scenario. In many cases, the worst-case task stack-up may rarely occur. For systems like these, overall performance could be increased either by selective ACETRB or WCETRB SPMA on a task by task basis, or by dynamic reallocation of SPM according to task schedulability. This thesis takes the latter approach to solving the problem of ACETRB versus WCETRB SPMMT selection, through the use of a schedulability-driven ACETRB / WCETRB swap (SDAWS).
2.1 – PROBLEMS WITH THE WCETRB SPMMT

The WCET for tasks often occurs during the worst-case task stack-up for a real-time system [7]. Here one may envision the scenario that an RT system is running normally, but then suddenly receives 50 Ethernet packets that all need to be processed as soon as possible. In this scenario, the execution time for the normal priority tasks is greatly increased because they will have to wait for some Ethernet packets to be handled before they get to run, (regardless of the scheduling method or SPMMT used). In this scenario, confidence that the normal tasks will meet their hard deadlines is diminished.

Intuitively, use of a WCETRB SPMMT seems necessary to ensure that tasks meet their deadlines in a worst-case scenario, while, in the general case, tasks may be able to meet their deadlines under the ACETRB SPMMT. However, the WCETRB SPMMT may actually increase the WCET of tasks versus the ACETRB SPMMT in this scenario [8]. This is because increased task ACET (under WCETRB SPMA) may lead to a worst-case task stack-up that is significantly worse than the worst-case task stack-up under ACETRB SPMA. If ACET is increased, aperiodic task dominance (ex: 50 Ethernet packets to process) will cause greater disturbance for WCETRB SPMA than ACETRB SPMA if a majority of the execution time spent running the aperiodic tasks is spent running AC paths and not WC paths (which is generally the case) [29]. If all of the aperiodic tasks could execute according to WC paths, WCETRB may be justifiable, but this is generally not the case in a real scenario. If a scenario with a very large number of aperiodic tasks occurring simultaneously such as this one can occur on a system, WCETRB SPMA will worsen
worst-case task stack-up versus ACETRB SPMA, which means that WCETRB SPMA may fail to achieve its goal of reducing WCET.

2.2 – THE SOLUTION: ADVANTAGES OF THE SDAWS SPMMT

In order to mitigate the damage that may be caused by a worst-case task stack-up due to aperiodic disturbances or otherwise under the WCETRB SPMMT, this thesis employs the schedulability-driven ACETRB / WCETRB swapping (SDAWS) SPMMT. Schedulability-driven SPMA swapping allows the RTOS to determine when WCETRB SPMA is necessary in order to ensure that takes meet their deadlines (or a percentage of the deadlines). Furthermore, SDAWS reduces worst-case task stack-up through employing ACETRB in the general case. In the previous Ethernet packet example, had SDAWS been employed, fewer tasks would have been running on the processor when the Ethernet packets came in. Furthermore, so long as the Ethernet packets operate according to high priorities, they will block any other real-time tasks from executing. Under SDAWS, the Ethernet packets could be processed according to ACETRB SPMA until a new task is scheduled that would not be able to meet its desired deadline.

2.3 – PROBLEMS OF SPMMT ANALYSIS

Most prior research has approached the SPM allocation problem from the angle of a non-preemptive, single-threaded application [3,6,7,8,9]. This prior research assumes that only one task is running at a time, and that the overhead incurred by the RTOS is insignificant. However, in data-driven systems, where the flow of data is often non-deterministic, these assumptions
cannot be made. No prior research has tested the performance of the WCETRB SPMMT under aperiodic disturbance scenarios (prior research has only run individual task instance simulations [7]). Furthermore, while prior research has tested the ACETRB SPMMT under aperiodic disturbance scenarios [2], no research has directly compared the performance of the ACETRB SPMMT and the WCETRB SPMMT under either individual task or multitask benchmarks. Additionally, because it must be managed by software, reallocation of variables into and out of SPM at runtime is generally ineffective, as it eats up processor time [2]. Therefore, dynamic SPMA has not been tested as much as static SPMA. Though dynamic SPMA has been investigated [4, 23], dynamic SPMA via mode swapping – the dynamic method employed by this thesis – has not been researched thoroughly prior to this thesis. This thesis solves these problems by testing WCETRB SPMA under multitask scenarios, comparing WCETRB SPMA and ACETRB SPMA for identical benchmarks, and providing dynamic SPMMT results for the SDAWS SPMMT.
CHAPTER 3 – SUPPORTING WORK

Scratchpad memory (SPM) has been around for a long time, and researchers have approached the problem of finding an optimal use of it in different ways from several angles. Some of the problems addressed include the tradeoff between cache and SPM [7,15,16], code versus data in SPM [7,16], static versus dynamic allocation of SPM [1,2,3,4], stack versus heap variables in SPM [7,16], and particular metrics versus overall speedup [1,2,3,8,9]. Moreover, SPM is often used in addition to cache in embedded systems because SPM occupies less area by 34%, and consumes less power by 40% versus cache [15].

This Chapter reviews existing SPM management techniques (SPMMTs) and analyses that are relevant to the development of a schedulability-driven SPMMT. In particular, this Chapter reviews average-case execution-time (ACET) driven and worst-case execution-time (WCET) driven SPMMTs, including the advantages and disadvantages of each. Furthermore, this Chapter reviews differences between types of real-time (RT) systems that must be taken into account when determining how to allocate SPM including pre-emption, system size, processor utilization, aperiodic disturbance event frequency, task schedulability, and other metrics. In particular, research by various authors into schedulability analysis and cache analysis is analyzed. To support this end, this Chapter reviews SPM management techniques employed by Lutron Electronics Co., Inc. as a real system example.

3.1 – THE ACET REDUCTION-BASED SPMMT

A large amount of the research that has been done has focused on determining an ideal mathematical procedure for allocating stack and heap variables into SPM in order to reduce the
average-case execution-time of individual tasks (ACET) [3,6]). Other research has focused on reducing ACET for real systems with multiple tasks running [1,2,7]. Experiments performed on individual tasks provide only a partial picture of the SPM management problem, as it needs also to account for task priority and length. For instance, a system with five tasks of varying lengths, sizes, and periodicities should not allocate an equal amount of SPM to each task. Rather, in order to maximize overall system ACET reduction, each task should have allocated to it an amount of SPM relative to the average percentage of processor utilization time that it receives (before any SPMMT has been applied).

3.1.1 – ACET Reduction-Based SPMMT: Allocation Methods

Following other trends, the majority of research into SPM allocation over the past ten years has focused on techniques that can be employed by the compiler [1,2,3,5,7,8,11,12]. Integer Linear Programming (ILP) is the most common approach that could be employed by a compiler [1,2,3,7,11,12]. However, in industry SPM allocation is generally performed either through code inspection, which is highly discouraged by researchers, or through hardware-based or software-based memory access profiling. Figure 1 shows the process flow for hardware and software-based profiling of data for ACET reduction-based SPM allocation generation.

![Figure 1: ACET Reduction-Based SPM Allocation: Method Flow](image-url)
3.1.1.1 – ACET Reduction-Based Allocation Method: Integer Linear Programming

Integer Linear Programming (ILP) is a mathematical technique that may be employed to measure data flow through a program. Using ILP, the number of memory accesses for each variable in a program is recorded, and those variables with the largest number of accesses are allocated into SPM. Many studies have used ILP to allocate data into SPM statically at compile time, and have achieved measurable improvements over cache-only solutions in several instances [1,2,3,7].

3.1.1.1 – ACET Reduction-Based Allocation Method: Anticipated Average-Case Profiling

A large number of studies have used simulations [1,2,3,6] rather than actual runtime results to show performance gains, or have investigated only non-preemptive systems where the number of possible flows through the program is easy to determine. However, for pre-emptive systems, a combination of ILP and explicit examination of the average number of variable accesses through profiling allow the engineer to determine which variables ought to be placed in SPM to maximize ACET reduction for a given task running according to a fixed set of anticipated runtime scenarios.

Moreover, profiling is solely sufficient for determining any SPM allocation [19,23]. Though a large amount of research has sought to develop a compiler that automatically allocates variables into SPM, this approach is inherently flawed, because, for a program with multiple conditional branches, input conditions, and paths through the code, the compiler cannot predict which path is the most common, the least common, etc. These compilers need to “guess” what the average path through the code is.
Figure 2 describes the profiling architecture used in [19]. In this architecture, simulations are run while profiling is performed. Filters are applied to count relevant data accesses, and partitioning algorithms are applied on the relevant profiled data sets (e.g. ACETRB or WCETRB). [19] discusses use of hardware to perform profiling automatically, though explicit (manual) profiling could be used as well through debugger tools suites. Profiling measures variable accesses at run-time rather than at compile time and therefore provides the most accurate average-case memory access counts. Simulations may be used by development tools to measure access frequencies at run time. Running a program under expected conditions, while performing profiling, will provide the programmer with a list of variables that were accessed the most frequently under the tested conditions. These conditions can be real or simulated.
3.1.2 – Some ACET Reduction-Based SPMMT Results

The results displayed in this section represent only a small portion of previous research into ACET reduction-based SPMMTs. P. R. Panda [3] demonstrates the performance in terms of number of accesses and number of cycles (time) of various allocation methods, including Panda’s approach is an ACET reduction-based approach that uses ILP to determine which variables to place into SPM (SRAM). In particular, type A incorporates only SRAM and no cache (using Panda’s ILP ACET reduction-based approach). Type B incorporates cache only. Type C places random variables in SPM (SRAM) and uses half cache and half SPM. Type D uses half cache and half SRAM but allocates variables into SRAM according to Panda’s method.

It is important to note that the use of different processor benchmarks results in widely varying results between experiments. It should also be noted that performance is inversely proportional to the number of cycles. Each benchmark program runs only a single task. As the source code for these benchmarks is unavailable, interpretation of performance depends upon text descriptions of each benchmark. Therefore, a brief description of each benchmark is provided below.

1. **SOR and DHRC Benchmarks**
   a. The smallest number of variables / memory size
   b. A large number of common variables with much higher access frequency than other variables (in SPM)
2. **Matrix Multiplication Benchmark**
   a. A very large amount of memory
   b. Shared memory workspace in SPM
3. **Dequant and IDCT Benchmarks**
   a. A small amount of memory
   b. A widely distributed memory-space causing additional cache block loads
   c. A large number of common variables with much higher access frequency than other variables (in SPM)
4. **Fast Fourier Transform and Beamformer Benchmarks**
   a. A streamlined memory-space reducing additional cache block loads
b. A large number of common variables with much higher access frequency than other variables (in SPM)

Figure 3 – Multiple Benchmarks – Normalized Processor Cycles [3]

Figure 4 – Beamformer Benchmark – Accesses and Cycles [3]

Figure 3 shows the results of several benchmark tests. The first thing one notices from examining figure 3 is that the ACET reduction-based SPMMT performs better than any other
memory type tested for the given benchmarks. The reason for this is clear. These benchmarks all perform streamlined mathematical operations for which there exists an operating memory space that is continuously in use by the processor. When the memory in continuous use is loaded into SPM, speedup is achieved through averted cache block loads. Thus, these benchmarks are tailored to indicate cases where SPM use is valuable. Note that the SRAM-only case performs better than the cache-only case for four out of seven benchmarks (SOR, DHRC, Dequant, and IDCT). This is because, for these benchmarks, the vast majority of memory needed can be fit into SPM (the benchmarks contain fewer variables). Furthermore, for two of the remaining four examples (Matrix Mult and FFT), the random SPM allocation strategy combined with some cache performs as well or nearly as well as the cache-only solution. This is because, for these examples, the access frequency for variables is relatively even, (i.e.: there is little advantage to allocating one variable into SPM over another).

Therefore, the Beamformer benchmark (figure 4) best indicates the performance of P. R. Panda’s ACET reduction-based SPMMT, because the operation is large enough to prevent all of the variables from being put into SPM, and distributed enough to eliminate the possibility of a single operational space that could continuously exist in SPM. In figure 4, the difference in SRAM accesses between the random distribution and the ACET reduction-based SPMMT distribution indicates the effectiveness of Panda’s technique [3]. For this benchmark, the ACET reduction-based SPMMT provides significant benefit over a random SPM distribution because the memory size is large enough and the variable access frequency varies enough to give a significant advantage to the SPMMT that selectively chooses variable to place in SPM according to access frequency.
3.2 – THE WCET REDUCTION-BASED SPMMT

Nearly all of the techniques applied to allocating data into SPM, including those that allocate SPM dynamically at run-time, and those that use procedures other than ILP to allocate data, have attempted to reduce only the ACET of tasks [2,3,5,6,7,10,11]; however, in hard real-time systems, improving the worst-case execution-time (WCET) of tasks is generally equally or more important [7,8], as these systems must ensure that all tasks will meet their deadlines in the worst-case. Because of data dependencies, determining the worst-case execution path through a section of code can occasionally be unfeasible; therefore, these techniques employ strategies to estimate the WCET of the unfeasible paths. Several studies have investigated WCET-based SPM allocation [2,8,9,12].

3.3.1 – WCET Reduction-Based SPMMT: Allocation Methods

The primary difficulty with reducing the WCET is that, for two possible worst-case paths with similar execution times, allocation of the worst path’s variables into SPM may cause the other path to dominate the WCET, and so the WCET reduction is only equal to the old WCET minus the new WCET, rather than the reduction in execution time of the original worst-case path [12]. Therefore, all WCET reduction-based SPM allocation methods must be iterative. That is, they must allocate SPM slices, see which path is the worst after the slices have been allocated, and then allocate new SPM slices (as illustrated in figure 5). Each SPM slice consists of the memory elements in the worst-case path that have the highest access frequencies.
[8] has identified static memory allocation techniques for allocating data into SPM according to worst-case execution time (WCET). The authors used three methods of compiler-based allocation: using ILP, using a “Greedy” approach, and using a branch and bound approach. Execution path prediction and memory access counting vary depending on which of these compiler-based methods are used.

3.3.1.1 – WCET Reduction-Based Allocation Method: Integer Linear Programming

The ILP approach utilizes the same ILP method described in 3.1.1, with the exception that only those variables that reside along the worst-case execution path are considered [8]. However, the ILP approach always overestimates the length of the worst-case path, because it does not take into account combinations of conditional branches that will never occur at the same time. This is because the longest possible path through the code from the compiler’s perspective is not necessarily the longest possible path that may occur in the code. Moreover, the longest possible path through the code may never actually occur in a data-driven system that processes one input at a time. These systems often contain failsafe code to consider a case that may never occur, and in this instance the ILP approach would allocate variables into SPM according to this case that may never occur.
3.3.1.2 – WCET Reduction-Based Allocation Method: Branch and Bound

To achieve a more ideal allocation than the ILP approach, the branch and bound approach is employed to prevent overestimation because of impossible path combinations [8]. This technique employs a lot of computational power to determine, for each variable, the maximum contribution of that variable along any path of execution. Variables with the largest maximum contributions are moved into SPM. The branch and bound method is shown to achieve optimal SPM allocation for WCET reduction for the benchmark programs on which it is run.

3.3.1.3 – WCET Reduction-Based Allocation Method: “Greedy”

The third method (“Greedy”) is a compromise approach, aimed at reducing the computation time required to determine SPM allocation. It does not guarantee optimal WCET reduction, but is shown in [8] to achieve very similar results to the branch and bound approach. The “greedy” method is applied by determining the second-worst execution path, and allocating the variable in it that has the most calls into SPM. Then, the new second-worst execution path is determined, and the process is repeated until SPM is full.

3.3.1.4 – WCET Reduction-Based Allocation Method: Anticipated Worst-Case Profiling

Though the branch and bound and “greedy” compiler-based approaches to WCET reduction-based allocation are able to determine the longest possible paths through code, and the paths which would take the longest time to execute, they are inherently limited. This is because compilers are unable to know the expected use-cases for tasks or combinations of tasks that may be run simultaneously for any system [7].
As is the case with ACET reduction-based SPM allocation, profiling can be used to determine the optimal set of parameters to place in SPM for any testable scenario or set of scenarios [19,23]. The worst-case execution paths for tested tasks are evaluated either by running the tasks under real conditions and evaluating only task instances that take the longest time, or by simulating worst-case input combinations.

In RT and embedded systems, it is frequently the case that if all tasks were scheduled to run simultaneously according to their WCETs, the system would be unschedulable. Therefore, worst-case profiling that measures the number of parameter accesses under a defined set of anticipated worst-case conditions provides a more accurate WCET-based SPM allocation than any compiler-based approach [19,23]. As stated earlier, however, explicit profiling does require a significant time investment, and will need to be performed repeatedly as modifications are made to existing code. Hardware-based profiling such as that described in [19] requires specific hardware to be developed and used for profiling and therefore cannot always be performed.

3.3.2 – Some WCET Reduction-Based SPMMT Results

The results displayed in this section represent only a small portion of prior research into WCET reduction-based SPMMTs. In [8], the authors compare ILP, branch and bound, and greedy WCET allocation methods according to a WCET estimator that was developed for the purpose of evaluating these methods. The WCET estimator evaluates SPM allocations according to different benchmark programs. It should be noted that each benchmark program runs only a single task.

However, unlike Panda’s benchmarks, the benchmark programs used in [8] differ dramatically from one another in terms of the amount of memory used by each program. This is
because the cases in which WCET-based SPMMTs grant reduced WCET over cache-only solutions is greater than the number of cases in which ACET-based SPMMTs grant reduced ACET over cache-only solutions. Fundamentally, the purpose of cache is to reduce ACET, and not WCET. Therefore, [8] does not test each WCET-based SPM allocation method against other SPM / cache combinations or other SPMMTs. Instead, the WCET estimator assumes that a 1:1 ratio of SPM and cache is utilized, as this is the most common and generally the most effective mixture of memory types employed on embedded chips [2]. Because the source code for these benchmark programs is not provided, table 2 that follows is used to indicate differences between benchmark programs.

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Data Memory (Bytes)</th>
<th>Scalars (Bytes)</th>
<th>Arrays (Bytes)</th>
<th>WCET considering infeasibility (cycles)</th>
<th>WCET w/o considering infeasibility (cycles)</th>
</tr>
</thead>
<tbody>
<tr>
<td>lingua</td>
<td>481</td>
<td>141</td>
<td>340</td>
<td>823,305</td>
<td>825,227</td>
</tr>
<tr>
<td>statemate</td>
<td>227</td>
<td>163</td>
<td>64</td>
<td>41,578</td>
<td>44,938</td>
</tr>
<tr>
<td>susan</td>
<td>36,232</td>
<td>96</td>
<td>36,136</td>
<td>293,989,241</td>
<td>485,328,185</td>
</tr>
<tr>
<td>compress</td>
<td>264,006</td>
<td>157</td>
<td>263,849</td>
<td>319,075</td>
<td>390,937</td>
</tr>
<tr>
<td>des</td>
<td>1,361</td>
<td>208</td>
<td>1,153</td>
<td>643,270</td>
<td>643,894</td>
</tr>
<tr>
<td>fresnel</td>
<td>536</td>
<td>536</td>
<td>0</td>
<td>256,172</td>
<td>256,172</td>
</tr>
</tbody>
</table>

Table 2 – WCET Benchmark Parameters for [8]

The difficulty with table 2 is that it does not indicate the number of accesses per scalar, per array, or per data memory element in general. Consequently, it is impossible for the reader to know how many accesses are occurring in SPM, and how many cache block loads (CBLs) are avoided according to each WCET-based SPM allocation method. Regardless, it can be assumed that differences in WCET between each allocation method result from reductions in worst-case CBLs according to variable access frequency.
Figure 6 – WCET Benchmarks and Allocation Method Performance for [8]

Figure 6 shows the results of each of the WCET reduction-base compiler SPM allocation methods according to six different benchmarks. In figure 6, the black bars indicate estimated WCET according to a cache-only solution. The three sets of bars indicate estimates taking into account SPM that is 5%, 10%, and 20% of the size of the total amount of memory needed by the
program. Note that in every case the cache-only WCET exceeds the WCET-based SPM WCET by a large margin. This is expected due to the ACET-reducing nature of cache.

In each of these benchmark experiments, the branch and bound technique performs just as well as or better than the “greedy” and ILP approaches. The ILP approach performs roughly equivalently to the other two for the benchmarks where there is not a significant difference between the WCET estimate with infeasibility paths included and the WCET estimate without infeasibility paths included. This is because the ILP approach includes infeasible paths in its SPM allocation. The ILP approach performs significantly worse than the other two methods for the “Susan” benchmark, in which there is a significant difference between including and excluding infeasibility paths. In the cases where the greedy approach performs worse than the branch and bound approach, this is because there is a significant enough difference between the second worst and worst execution paths such that the worst-case path does not receive as much SPM allocation as it would under branch and bound [8]. This occurs because allocation of variables in the second worst path into SPM results in new worst-case paths. In the cases where the greedy approach performs as well as the branch and bound approach, this occurs because there exists very little difference between the worst-case and second worst execution paths, such that nearly every time the WCET is estimated, a new second worst execution path is determined and optimized by the greedy algorithm.

3.3 – SCRATCHPAD MEMORY VERSUS cache

Other research has investigated the tradeoff between using cache only, SPM only, and a mixture of SPM and cache, using ILP type allocation of data into SPM [1, 2, 12, 15, 17, 18, 20,
A large amount of research has focused on hardware-controlled caches versus software-controlled caches via SPM [24, 25, 26, 27], on optimal combinations of cache and SPM [7, 15] and on determining the conditions under which each of these types of on-chip memory components provides greater benefit over the others.

Cache is widely used in a majority of micro-controllers today. While many of these micro-controllers incorporate SPM, many more do not. This section touches on each area of study without going into a great amount of detail for any one of them. This is because the intent of this section is merely to illuminate the strengths and weaknesses of each type of on-chip memory in order to identify the types of RT systems that benefit from SPM use. The effectiveness of SPM versus cache depends heavily upon whether SPM is allocated at compile time (statically) or at run time (dynamically).

3.3.1 – Cache-Only Solutions Versus Static SPM Allocation

For general purpose applications in which programmers do not have additional time to invest in optimally allocating available SPM, cache-only solutions are widely preferred. This is because it can take a great deal of development time to determine which SPMMT to use, how to partition SPM, etc. There exist many circumstances in which cache-only solutions out-perform any SPM-only or SPM / cache combinations for any static SPMMT, in terms of ACET reduction. These circumstances occur whenever two conditions are met. The first condition is that system variables or shared task variables are not accessed frequently enough such that they would reside in cache 100% of the time. The second condition is that a small subset of repeating tasks requires the use of all of the system cache. In this type of scenario, statically allocated SPM provides very little benefit because, if the same data were allocated into cache in the form of
cache blocks, it would be replaced by cache blocks with greater access frequencies a majority of the time.

Cache-only solutions are limited according to cache block sizes, and therefore do not perform as well as cache / SPM combinations in many common circumstances [3, 8]. In cases where global variables used by common functions are spread over a large area of code space, cache block loads (CBLs) may result in a large amount of unnecessary data being pulled into cache. This occurs most often when, for a single task, a large number of function calls are made which cause the PC to jump between many different disjointed code segments (spaghetti code).

Though compilers generally attempt to fit variables in memory such that CBLs are minimized, it is generally impossible to minimize CBLs for every scenario. Furthermore, compilers generally cannot know or anticipate runtime scenarios or flows through code, and have to guess. Guessing may result in a variable structure optimized to reduce CBLs for uncommon scenarios, particularly in the case of data-driven systems where many tasks may run aperiodically, infrequently, or not at all under standard conditions.

A cache-only system would perform better than \textit{statically allocated} SPM / cache combination in this scenario – when the optimal on-chip memory solution is highly dynamic (i.e.: when data must be swapped into and out of on-chip memory at a high frequency relative to the periodicity of system tasks). This is because studies have shown that hardware-controlled caches perform equal to or better than \textit{static} SPM allocation under a vast majority of cases [27]. This can be mitigated if dynamic memory allocation (DMA) is used (see Chapter 6), but only under certain circumstances.
3.3.2 – SPM-Only Solutions and Dynamic SPM Management

SPM can be found in the majority of small and medium size microcontrollers. In many very small applications SPM serves as main memory; however, for a majority of large RT systems, main memory must be off-chip. SPM-only solutions are common on very small microcontrollers in which the intent is for all of main memory to be located in SPM. Further, SPM consumes less power than cache and is often used in place of cache for this reason [15].

SPM-only solutions using dynamic SPM allocation are difficult to compare to cache-only solutions. Dynamic SPMMs are required, and are difficult to implement, making dynamic SPM management less common than static management. Dynamic SPM management can involve either paging-based SPM management in which large memory blocks are occasionally moved into and out of SPM via DMA, mode-based SPM management in which memory blocks are moved into and out of SPM according to system-defined transition parameters (such as when tasks are complete or are scheduled to run on the processor), or software-controlled caches [27]. The first two of these types of Dynamic SPM management involve mixed cache/SPM solutions, and are able to achieve better performance than non-avertable cache-only solutions [15].

Software-controlled cache solutions using SPM can involve either SPM-only hardware or a mixture of SPM and cache. A large amount of research over the past ten years has targeted software-based caching versus hardware-based cache [24, 25, 26, 27]. In general, hardware-based cache grants more consistent performance. However, SPM-only solutions can benefit from static and certain dynamic SPM allocation used conjointly with software-based caching. Further, software-based caching methods using SPM can target factors that cannot be predicted by hardware (such as task repetition frequency, algorithm characteristics, RTOS features, etc).
Existing work identifies software-caching techniques that perform better than hardware-based cache under a variety of scenarios [24, 25, 26, 27]. This is due to the limitation of CBL sizes (loading unnecessary data) and the fact that hardware caches generally wait until memory is accessed before performing CBLs. Though pre-fetching can be used to mitigate the latter disadvantage, algorithms have been developed that can grant software caching an advantage in this area [27].

### 3.3.3 – A Mixture of SPM and Cache

Ignoring software-controlled caches, studies have shown [7, 15] for most benchmark programs that a mixture of cache and SPM is preferable to an SPM-only solution. This is because of the fact that variables accessed along rare or uncommon paths will never be moved into fast memory for the SPM-only solutions, leading to repeated large memory access times for worst-case paths. Research has shown that cache block size plays a large role in the effectiveness of cache. The fact that ideal cache block sizes may vary dramatically for different real-time programs is another reason why some amount of SPM is often preferred over cache-only solutions.

In general, embedded developers prefer the use of some SPM in place of additional cache due to the flexibility SPM provides [7]. Specifically, RT systems benefit from SPM insofar as it allows task WCETs to be either partially or completely guaranteed not to exceed a given execution time. Cache adds uncertainty, as it is extremely difficult to predict what will and will not be loaded into cache at any given point in time. However, some cache is generally required in order to ensure fast handling of uncommon data paths or ones which require a majority of system resources.
3.4 – RTOS / SYSTEM-DEPENDENT SPM ALLOCATION

SPM allocation may be employed for RTOS parameters for several reasons. It may be employed to guarantee minimal RTOS overhead, to guarantee fast communication times between tasks, to guarantee fast external data latching (such as Ethernet or serial data), etc. This section reviews only the three RTOS-dependent SPM allocation criteria that are relevant to this thesis.

Specifically, this section covers stack versus heap SPM allocation because this thesis targets heap SPM allocation. This section covers preemption versus non-preemption because this thesis targets non-preemptive SPM allocation applications. Further, this section covers static versus dynamic SPM allocation because this thesis proposes a method of dynamic SPM allocation (the schedulability-based approach) and contrasts it against static SPM allocation approaches (ACET reduction-based and WCET reduction-based).

3.4.1 – Stack Versus Heap SPM Allocation

Some research over the past ten years has targeted differences between allocation of task / system / interrupt stacks in SPM and allocation of global (heap) variables in SPM [17, 20]. In large part, these allocation approaches are case-specific. That is, some systems benefit more from stack allocation in SPM than from heap allocation in SPM, while others benefit more from heap allocation.

Research done into stack versus heap SPM allocation assumes that a mixed SPM and cache (50/50) memory structure is used [17, 20]. Therefore, whether stack or heap data ought to be placed in SPM in order to reduce task / system / RTOS ACETs or WCETs depends upon two
things: the number of cache block loads (CBLs) that are avoided via SPM, and the amount of wasteful data per CBL that is avoided via SPM. If task stacks are continuously being loaded / reloaded into cache (as is often the case in preemptive RTOSes), and much of the stacks that are loaded into cache is not used (i.e.: stacks never exceed more than a certain percentage of their maximum size), then it is generally more beneficial to place task stacks in SPM [20]. Alternatively, if there is only one task / system stack (as in the case of most non-preemptive RTOSes), or if the system relies on shared accesses to global memory that would most likely result in several cache blocks never being replaced, then it is better to store heap data in SPM [17].

Some researchers and engineers in the industry have concluded that the best solution is a mixed one in which either part or most of the task / system stacks are located in SPM along with global variables according to an ACET-based or WCET-based SPMMT [16]. The size / number of system or task stacks that are located in SPM depends upon the speedup desired. If ACET reduction is desired, average-case stack sizes may be placed in SPM in order to free space for additional ACET reduction-based SPMMT global variable allocation. If WCET reduction is desired, the entire task stack / interrupt stack / system stack may be loaded into SPM in order to eliminate additional stack-based CBLs for worst-case data paths.

3.4.1.1 – Mixed Stack and Heap SPMMT: Real System Example

Lutron Electronics Co., Inc. is an independently owned and operated electronics company located in Coopersburg, PA. It designs and manufactures light dimmers, home and commercial lighting control systems, and automatically controlled shades. Lutron’s commercial office building systems need to be designed in order to accommodate worst-case (simultaneous) use of
lighting control devices. Therefore SPM management is employed to improve worst-case handling of communication processing tasks.

Lutron’s largest commercial lighting system involves a multitude of lighting and control devices (switches, dimmers, light-level sensors, etc.) controlled by a central processor. The central processor contains a database of light levels, shade positions, occupancy levels, etc. for different spaces within the building in which the system is installed. The central processor receives command signals from different types of control devices, uses the database to determine the appropriate response, and then sends command signals to the lighting or shade devices that must respond, all via wired communications.

The central processor incorporates a 32-bit microcontroller that runs the database, communication, and data-processing software. This 32-bit microcontroller contains both cache and SPM in the form of on-chip SRAM. The custom RTOS that runs on this central processor is preemptive; therefore, each task that runs on the central processor possesses its own stack. Furthermore, each stack is large enough to accommodate the worst-case data path for its corresponding task.

As a consumer electronics company, Lutron has an interest in improving the efficiency of the software running on this central processor so that new hardware does not need to be designed as code size / data processing needs expand. Therefore, the author of this thesis performed SPM management research and optimizations for the central processor. A large part of this research involved investigating how SPM was utilized at the time and determining ways in which SPM could be used more effectively. In particular, testing was performed in order to determine whether placing task stacks into SPM would result in greater task WCET / ACET performance than placing global variables into SPM.
Testing was performed using a system incorporating a few central processors connected to a large number of switch-type devices that would simulate a system stressed to maximum command-handling capacity, dubbed the “clickety-clackety wall” due to the noise it made while operating. Using this test system, it was determined that due to the nature of the software running on the RTOS (large number of periodic and aperiodic tasks, large number of context switches per task instance, very large amount of global memory for communications, etc), a far greater benefit would be achieved through the allocation of task stacks into SPM than through the allocation of global memory alone.

After meetings with fellow engineers, it was agreed that several task stacks ought to be moved entirely into SPM in order to reduce the WCET of those tasks. The decision to move stacks into SPM was made in order to ensure that those tasks would not slow the system too much in the event of a worst-case execution. Further, the tasks that had their stacks moved entirely into SPM had a fairly large worst-case execution to average-case frequency. That is, these tasks ran according to their worst-case data path often due to communications handling. Additionally, the interrupt stack was moved into SPM in order to reduce interrupt service times (resulting in a large amount of speed-up due to the frequency of interrupt service routines).

Furthermore, it was decided that several RTOS global (heap) parameters would be allocated in SPM in order to reduce context switch time (ACET reduction). Additionally, a small amount of SPM was allocated for message passing parameters ensuring fast communication between interrupts and tasks. The message passing parameters were placed in SPM in order to ensure safe operation in the event of a worst-case interrupt stack-up or worst-case data handling scenario.
Moreover, after a large amount of trial and error and a significant amount of research, the memory placed in SPM resulted in a reduced ACET for tasks with non-critical deadlines, and a significantly reduced WCET for tasks with critical deadlines. Though it was impossible to know for certain whether the best SPM allocation was employed, the project was a definite success. After SPM allocation was completed, very thorough testing was performed in order to ensure the safe operation of all system structures that had parameters taken from SPM and placed in main memory. Further testing was performed on other test systems in order to ensure robust operation of software affected by the new SPM allocation.

3.4.1.2 – Mixed Stack and Heap SPMMT: Results

Results of the SPM allocation were measured using a set of timing analysis tools that were built into the RTOS in order to measure the average and worst-case execution times for tasks in the system. Timing was measured both before and after each of the stages of the SPM reallocation. The four stages timed include the pre-reallocation stage (Original), the initial reallocation stage (Rev 1), interrupt-stack and RTOS parameter reallocation stage (Rev 2), and post-project point in time after several new features had been added to the system (6/23/2010). Additionally, idle time was measured at each stage in order to measure overall system speedup. Figures 7, 8, and 9 show the WCET of tasks before and after various stages of static stack allocation into SPM. The last bar for each graph (quantum trunk) shows WCET performance after additional code was added to the system, while the SPM allocation remained identical to that of bar three (dedicated interrupt stack).
Figure 7 – Idle Runtime in Seconds over a Five Minute Period

Figure 8 – Core RTOS Event Handling Task WCET in microseconds
Figure 9 – Example Task ACET – Task Stack Moved into SPM in Rev 2

Figure 7 shows idle time increase due to the SPMMT employed. Figures 8 and 9 show task WCET reduction for two key communication handling tasks. The SPMMT employed for the software running on the central processor of the lighting control system resulted in an approximate 50% ACET reduction and an approximate 60% WCET reduction for tasks that had their stacks moved into SPM (figures 8 and 9). For tasks that did not have their stacks moved into SPM, a 10% ACET / WCET reduction was achieved through the relocation of RTOS parameters into SPM. The maximum idle time increase that was achieved was 20% (due to an idle time gain of approximately 60 seconds for a five-minute test).
3.4.2 – Preemption Versus Non-Preemption

Preemption offers several performance benefits, the largest of which being immediate handling of high priority, critical tasks [28]. One difficulty with preemption is that system lock-up can occur when low priority tasks gain control of shared resources and thus do not allow high priority tasks to run when they need to. SPM can be used to speed-up shared memory accesses in order to lessen the effect of lock-up [29]. In general, however, preemption affects SPM allocation in other ways [29].

Whether an RTOS is preemptive or non-preemptive greatly affects a developer’s ability to determine which task will be running at any given time, what variables will be loaded into cache, what the stack will look like, etc [14, 28, 29]. Preemptive RTOSes are much more difficult to test, because preemption makes these things more difficult to determine. Consequently, SPM is often used in pre-emptive RTOSes to ensure that variables are located in fast on-chip memory. Furthermore, it is generally the case that preemptive RTOSes are larger in terms of amount of code space than non-preemptive RTOSes, and introduce much more overhead into the system than non-preemptive RTOSes. Therefore, preemptive RTOSes benefit from having task headers and context-switching parameter data loaded into SPM.

In addition, preemptive RTOSes that incorporate separate task stacks for each task require much more total stack space than a similarly sized non-preemptive system [29]. This means that preemptive RTOSes do not necessarily outperform non-preemptive RTOSes, but, rather, that non-preemptive RTOSes can perform better than preemptive RTOSes for very limited on-chip memory spaces [14]. Moreover, for any SPM space, non-preemptive RTOSes have more leeway, because SPM does not need to be occupied primarily by overhead or task
stacks. For this reason, ACET reduction-based and WCET reduction-based SPMMTs for heap allocation of data into SPM should be considered particularly for non-preemptive RT systems.

### 3.4.3 – Static Versus Dynamic SPM Allocation

Static SPM allocation refers to SPM allocation that occurs at compile time. Dynamic SPM allocation refers to SPM allocation that occurs at run time. In industry, static SPM allocation is more common because it is simpler, and ensures that certain parameters are always in SPM. Dynamic SPMMTs do not provide the same kinds of assurances that static SPMMTs do; however, studies have shown that dynamic SPMMTs can achieve much more ACET reduction than static SPMMTs under most circumstances [23].

There are many ways of allocation SPM dynamically. On the extreme-end, it is possible to create a software-controlled cache using SPM, though this type of cache is not as effective as hardware-controlled cache in a majority of scenarios as has already been mentioned. On the other end of the spectrum, it is possible to allocate cache manually dynamically via a user interface and commands sent to the RTOS. In-between these two extremes, it is possible to switch SPM modes according to runtime conditions that are automatically detected by the RTOS. This is dynamic SPM mode switching (DSPMMS).

The dynamic SPMMTs examined by [4] and [23] incorporate a form of DSPMMS in which SPM usage is examined in a similar manner to paging. Though this research is closely related to software caching, the techniques and mathematics employed are sufficiently different such that the number of SPM page loads do not impact performance as negatively as block loads in software caching. Further research should be done in this area to determine whether such techniques can be employed at the RTOS level generically.
DSPMMS is entirely dependent upon the application / RT system at hand. Therefore, this is an open area of research. Applications of DSPMMS may include such things as optimized safe-mode operation, acceleration of data-driven event handling, increased performance of user-specified tasks, etc. The schedulability-driven SPMMT proposed in this thesis is a form of DSPMMS.

3.5 – REAL-TIME SCHEDULING AND SCHEDULABILITY ANALYSIS

Because this thesis proposes the use of schedulability analysis as a means of determining when to switch between ACET and WCET SPM allocation, it is important to study the techniques others have used to determine the schedulability of tasks when using different scheduling algorithms. In particular, this section covers fixed priority and earliest deadline first (EDF) scheduling techniques.

Schedulability analysis involves determining whether or not tasks will meet their deadlines. This thesis specifies a task’s deadline as the amount of time between when a task is first scheduled to run by the RTOS until that task must have completed its execution cycle. Factors affecting whether or not a task will finish executing by its deadline time include the amount of time between when the task is scheduled until when it begins running on the RTOS.

3.5.1 – Rate Monotonic Analysis

Rate monotonic analysis (RMA) is a mathematical algorithm uses to determine whether a set of tasks is schedulable. RMA determines the schedulability of tasks in a system according to the periodicity of tasks, the worst-case execution times of tasks, and the number of tasks
The process that the algorithm uses to determine schedulability is to assign the highest priorities to tasks with the highest periodicity, and to assume that each task will take its worst-case execution time (WCET) to execute. In general the assumption is also made that task deadline time is equal to task periodicity. If task deadline time is smaller than task periodicity, deadline-monotonic analysis is used. Deadline monotonic analysis is identical to rate monotonic analysis except that task deadline time is a parameter in the equation as well. The rate monotonic algorithm (shown in (1) below) takes the number of tasks, task periodicities, and WCETs as inputs returns processor utilization $U$.

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq n(2^{1/n} - 1)$$

In (1), $C_i$ is the task WCET, $T_i$ is the task period, and $n$ is the number of tasks to be scheduled. When the number of tasks approaches infinity, this expression will tend towards the natural log of two, or 0.693. Using the RMA algorithm, tasks can meet all the deadlines if CPU utilization is capped at 69.3% or less [30]. However, RMA does not verify for certain that a set of tasks is not schedulable. Rather, if a task fails the RMA test for schedulability, it means that the task may not be schedulable. Other analysis have shown that randomly generated task sets can be scheduled with 85% processor utilization. Furthermore, it is possible to schedule all task to meet deadlines with close to 100% processor utilization, depending on the periods and WCETs of all of the tasks involved.

The RMA schedulability algorithm can be used at compile time to determine whether a set of tasks is schedulable. This has several advantages. The primary advantage is that schedulability analysis does not need to be performed at run-time, freeing up processor time for
other things. It also reduces RTOS kernel complexity. However, the RMA schedulability algorithm is not optimal as it does not work well for all RT task scenarios. Despite any shortcomings, the manner in which rate-monotonic analysis determines schedulability is robust and applicable to this thesis.

The main problem with RMA is that the precise periodicity of tasks must be known. While WCETs of tasks may be approximated or determined through testing, periodicity cannot always be determined, because many real-time systems run aperiodic tasks. These aperiodic tasks must be handled using special cases – instances of rate-monotonic analysis that account for worst-case periodic task stack up for a given instance of an aperiodic task being scheduled. For systems with many aperiodic tasks, task priorities may need to change frequently as these special cases occur over and over again.

3.5.1.1 – Non-Preemptive Schedulability Analysis

The standard approach to determining task schedulability for non-preemptive RTOSes is to set task priorities by assigning tasks with the shortest deadlines the highest priority and continuing until the tasks with the longest deadlines have the lowest priority [14, 30]. RMA is then employed on the task set. Aperiodic disturbances are handled according to different disturbance scenarios that may occur. If all aperiodic tasks may run at the same time, RMA is run to account for all of them. If only one aperiodic task may run at a time in the worst case, then each aperiodic task scenario is tested individually.

For non-preemptive systems where the number of possible task / priority combinations is manageable, trial and error type approaches are sometimes used. Tasks are assigned different priorities, and runtime testing is performed on different sets of prioritized tasks to determine
schedulability. If tasks are still unable to meet their deadlines after all possible priorities have been tested for task sets, task deadlines may be modified in order to determine what task deadlines would need to be in order to create a schedulable system.

3.5.1.1 – Preemptive Schedulability Analysis

Preemption dramatically affects whether or not tasks can complete their deadlines [28]. This is because non-preemptive RTOSes must wait for the current task to complete execution before any other tasks can run. Therefore, any newly scheduled high priority task must wait for a large low priority task that is running to complete. In the worst-case task stack-up scenario, high priority tasks are blocked from running such that it is impossible for them to meet their deadlines. For this reason, preemptive RTOSes are generally preferred for large RT projects.

However, preemption drastically complicates analysis. Though RMA can be performed for preemptive systems, the worst-case number of task divisions for each task and the worst-case context switch time for each task switch must be accounted in order to use RMA. Because these types of systems tend to be large, the amount of work required to determine each of these worst-case parameters can be staggering. In the case of an unfinished project, determining worst-case RMA parameters can be impossible. Therefore, engineers working on preemptive systems often employ metrics to determine fixed priorities for tasks. The following are only a few prioritization metrics that are used in industry [3].

1. Task Deadline Time (shortest deadlines have highest priority) – For systems that have their schedulability strained by tasks with deadline times smaller than task repetition periods (i.e.: aperiodic tasks with large WCETs and short deadlines, or periodic tasks with short deadlines relative to their periodicity).
2. Task Repetition Period (shortest periods have highest priority) – For systems whose periodic tasks place the largest strain on task schedulability (where task deadlines are determined through task repetition period).

3. Soft Deadline Maximization (data processing tasks are prioritized) – For data driven or input / output systems where data throughput and processing is prioritized over meeting any hard deadlines. This metric is commonly used when all tasks are guaranteed to meet their deadlines under the conditions in which the high priority data processing tasks will run.

These metrics work well for preemptive operating systems in which new tasks with high priority will be executed by the RTOS kernel immediately. For non-preemptive RTOSes, these metrics do not work as well. This is because, for non-preemptive RTOSes, large low priority tasks will block high priority tasks until they complete execution, as the worst-case task blockage time must be included either in the schedulability analysis as a constant or in the task deadline time.

3.5.2 – Fixed-Priority Scheduling

The most common type of scheduling algorithm is fixed priority (FP). This type of algorithm assigns a static priority to each task in the system [14, 30]. This scheduler is the most common because it is easy to implement. Fixed priority schedulers are easy to implement because they always switch to the task with the highest priority, regardless of other conditions. Unlike earliest deadline first (EDF) scheduling, fixed priority does not need to resort the priorities of tasks when a task completes or a new task is scheduled.
The problem with fixed priority scheduling is that it requires analysis to be performed on all of the tasks in the system so that appropriate priorities for tasks can be determined. However, RMA does not indicate what the priorities of each task should be. Rather, rate monotonic analysis only verifies whether a set of tasks is schedulable. Furthermore, aperiodic disturbances require additional analysis to be performed under each possible disturbance scenario. RMA can be performed ahead of time for each potential type of aperiodic task that can occur. However, RMA only works for a system where the worst-case aperiodic task stack-up is known. Many real-time systems do not have to deal with many aperiodic tasks, or deal only with aperiodic tasks that do not heavily affect the run-time of the system. For these systems, FP scheduling is acceptable. Therefore, in industry, as tasks are expanded and new tasks are added to a system, worst-case aperiodic data processing requirements under different scenarios are used as worst-case task periods. A clear example is a system that uses a task to process one Ethernet packet at a time. That task’s period would be specified as the worst-case Ethernet packet entry rate for the purposes of performing RMA.

3.5.3 – Earliest Deadline First Scheduling

While fixed priority scheduling provides a simple solution to task scheduling, it often has difficulty accounting for aperiodic disturbance scenarios. This is because under each of these scenarios, the optimal task prioritization may be very different than in the general case. Consequently, and for other reasons, other scheduling algorithms are often employed.

Earliest deadline first (EDF) scheduling is a scheduling method that is used in place of fixed-priority scheduling. Unlike rate-monotonic scheduling, earliest deadline first (EDF) scheduling requires schedulability analysis to be performed at runtime [13]. EDF scheduling
grants the RTOS running it an increased ability to meet task deadlines. Though this thesis does not employ EDF scheduling, this thesis does propose reallocation of SPM according to point-schedulability checks.

[31] shows that EDF scheduling results in better results regarding task completion times versus fixed priority in the case where aperiodic tasks play a large role in system execution. Prior research establishes clear benchmarks for demonstrating scheduling performance such as measuring how soon before their deadlines tasks are able to complete (TST), how many are able to complete in an unschedulable scenario, etc. Though research has shown that EDF scheduling is not ideal, alternatives all involve incorporating the amount of time the tasks will take to complete into the scheduling calculation. This is not realistic in most modern applications, because task completion time is very non-deterministic. It is often impossible to guess how often software will take the shortest route versus the longest route through a particular series of conditional statements. However, task deadline time can be determined prior to run-time. This is because task deadline time is determined through protocol adherence, user requirements, and detailed design specifications. Consequently, EDF scheduling is suitable for industry adoption, and it has been in some instances (e.g. NASA, Aerospace Industry, etc).

The EDF scheduling algorithm recalculates task priorities according to the order in which tasks must finish. Task reprioritization provides increased performance and allows for handling of systems dominated by aperiodic tasks, but reprioritization also increases overhead because task priorities must be resorted at each task completion point [13]. This work proposes a desired deadline detection algorithm that performs point-schedulability checks in the same manner as EDF schedulers, except that the deadlines used are not hard deadlines but "desired" ones. This difference is be discussed in the next section.
3.5.1.1 – Point-Schedulability Check

EDF scheduling employs point-schedulability checks (PSCs) at task scheduling points and at task completion points [13]. The PSCs are used in order to verify task schedulability at runtime. The point-schedulability check determines if the tasks currently scheduled on the processor can meet their deadlines. A PSC is performed by running the response time equation for each task in the system.

\[ a_{n+1} = C_{n+1} + \sum_{i=1}^{n} (C_i) \]  

(2)

In (2), \( a_{n+1} \) is the response time of task \( n+1 \), (the amount of time expected to elapse before task \( n+1 \) completes), \( C_{n+1} \) is the WCET of task \( n+1 \), and \( C_i \) is the WCET of task \( i \). (2) implies the following: for a task, if the task’s WCET plus the sum of WCETs of tasks with higher periodicity (the task’s response time) is less than the task’s deadline time, then the task is schedulable.
CHAPTER 4 - METHODOLOGY

The methodology Chapter of this thesis is split into three parts:

- Part 1 – Design Methodology. This part describes in detail the Schedulability-Driven SPM Swapping Algorithm, RTOS, and SPMMT simulator developed for this thesis.
- Part 2 – Validation Methodology. This part lays out how the RTOS and SPMMT simulator are validated to ensure correct operation and accurate simulation results.
- Part 3 – Experiment Methodology. This part defines the SPMMT simulator test cases which form the bases for comparison between the different SPM management strategies. This section also describes the rationale behind each set of test cases.

4.1 – DESIGN METHODOLOGY

The design methodology portion of this Chapter is divided into three sections.

- Section 1 – Development and classification of the Schedulability-Driven ACETRB / WCETRB Swapping (SDAWS) Threshold Detection Algorithm
- Section 2 – Design and parameters of the RTOS developed for this thesis.
- Section 3 – Design of the SPM Management Technique Simulator (SPMMT simulator).

4.1.1 – Development of the SDAWS Threshold Detection Algorithm

SDAWS requires memory blocks to be swapped into and out of SPM according to task schedulability. However, if ACETRB to WCETRB swapping only occurred whenever tasks
could not meet their deadlines, than there would be little benefit to using SDAWS versus ACETRB SPMMT. Therefore, task deadline thresholds are used by this thesis to determine when to swap SPM allocations. The deadline threshold is a percentage of task deadline time. A threshold detection algorithm is developed by this thesis in order to determine when to swap SPM allocations. This section describes the equations used by the threshold detection algorithm, and how they are used. The following key terms are used by the equations of the SDAWS threshold detection algorithm:

4.1.1.1 List of Key Terms Utilized by the SDAWS Threshold Detection Algorithm

- \( t_n \): The task being examined by the scheduler.
- \( n \): The priority of task \( t_n \). Lower numbers have higher priorities, with 0 having the highest priority.
- \( \text{Threshold}(t_n) \): Value between 0 and 1 that represents the percentage of deadline time by which it is desired for \( t_n \) to have completed.
- \( T(\text{Realloc}) \): The amount of time the system takes to reallocate SPM from one allocation scheme to the other.
- \( \text{Scheduled}(t_n) \): The number of scheduled instances of task \( t_n \).
- \( \text{Running}(t_n) \): The number of running instances of task \( t_n \).
- \( \text{Runtime}(t_n) \): The amount of runtime all running instances of task \( t_n \) have received.
- \( \text{WCET}(A,t_n) \): The WCET of task \( t_n \) under the SPM management technique \( A \).
- \( \text{DCT}(t_n) \): The desired completion time of task \( t_n \).
- \( \text{SWCETS}(t_n) \): Scheduled WCET Sum. The total WCET of tasks scheduled to run.
- \( \text{RWCETS}(t_n) \): Remaining WCET Sum. The total remaining WCET of tasks that are currently running.
- \( \text{TRT}(t_n) \): The task response time of task \( t_n \).
4.1.1.2 Desired Completion Time

This work defines a deadline threshold as the fraction of a task’s hard deadline by which designers desire for the task to have completed. In addition, this work defines desired completion time (DCT) as a simple function of deadline threshold and deadline time (3).

\[ DCT(t_n) = \text{Threshold}(t_n) \times \text{Deadline}(t_n) \]  

(3)

SPM reallocation is performed depending on the ability of tasks to complete before their desired completion times (DCTs). The algorithm used to determine when to swap variables into and out of SPM that is proposed by this work is called by the real-time scheduler at scheduling points and task completion points.

4.1.1.3 Task Response Time

In order to determine schedulability, the algorithm determines the response time of each task (the expected amount of time that will elapse before the task under examination will complete). The task response time (TRT) (6) is calculated according to the WCET of the task under examination, the remaining WCETs of the tasks currently running on the processor with higher priority (4), and the WCETs of the tasks scheduled to run on the processor with higher priority (5). The algorithm then compares this time to the desired completion time.

\[ \text{RWCETS}(t_n) = \sum_{r=0}^{n-1} [\text{Running}(t_r) \times \text{WCET}(\text{ACETRB}, t_r) - \text{Runtime}(t_r)] \]  

(4)

\[ \text{SWCETS}(t_n) = \sum_{k=0}^{n-1} [\text{Scheduled}(t_k) \times \text{WCET}(\text{ACETRB}, t_k)] \]  

(5)

\[ \text{TRT}(t_n) = \text{WCET}(\text{ACETRB}, t_n) + \text{RWCETS}(t_n) + \text{SWCETS}(t_n) \]  

(6)
Furthermore, (4) assumes that, for each priority level, only one task may be running at the same time. The assumption is made that incoming tasks with the same priority as tasks that are currently running must wait until the running tasks with the same priority complete before they are given run time.

4.1.1.4 SDAWS Scheduling-Point Detection

The scheduler swaps SPM variables from the ACET to the WCET allocation at system scheduling points. At each system scheduling point, the ability of every task in the system to meet its DCT is examined according to the ACET to WCET swap criterion (7). The swap is performed when the operating system determines that any one of the tasks will not be able to complete by its desired completion time (DCT). This calculation assumes that all tasks in the system with higher priority than the task under examination will take their WCET to run.

\[ \text{DCT}(t_n) + T(\text{Realloc}) \leq \text{TRT}(t_n) \]  

4.1.1.4 SDAWS Completion-Point Detection

The scheduler swaps SPM variables from the WCET to the ACET allocation at task completion points. At each task completion point, the ability of every task in the system to meet its DCT is reexamined according to the WCET to ACET swap criterion (8). WCET of all tasks is assumed. If all of the tasks are able to meet their DCTs after the reexamination, then the swap from WCET to ACET allocation is performed.

\[ \text{DCT}(t_n) + T(\text{Realloc}) \geq \text{TRT}(t_n) \]
Both the scheduling point examination and the task completion examination use the ACETRB allocation estimation of WCET for all tasks in the system, to avoid oscillatory SPM swapping. If the task completion examination used the WCETRB allocation to estimate task WCETs, then it is possible that tasks would not be able to meet their DCTs after switching to the ACETRB allocation, because the WCET under ACETRB allocation will always be larger than WCET under WCETRB allocation.

4.1.2 – Design of the RTOS

The RTOS developed for the purpose of running the SPMMT simulator will be referred to by this work as the Incredibly Minimalist C RTOS (IMCRTOS). This name reflects the language in which the RTOS is written as well as its limited functionality. This section uses “RTOS” and “IMCRTOS” interchangeably to refer to the IMCRTOS.

The IMCRTOS is developed for exclusive use on the Freescale MCF52259 microcontroller, though a majority of the code could easily be transported to any other 32-bit platform. Specifically, the IMCRTOS takes advantage of timing and I/O functionality on the MCF52259. However, the IMCRTOS is written to contain only those RTOS elements which are required by the SPMMT simulator. Thus, the IMCRTOS possesses the following components:

A. A non-preemptive RTOS kernel with fixed-priority task management, idle mode, and automatic task scheduling functionality.

B. A simple I/O Interface which allows one-way serial communication.

C. Simple timing functionality which allows the current time relative to the most recent system restart to be measured in micro-seconds.

D. Automatic capture of task R-T timing characteristics.
E. A system schedulability analyzer with:

a. Task completion-point scanning functionality.
b. Task scheduling-point scanning functionality.

4.1.2.1 – RTOS Kernel, Idle Mode, and Task Scheduling

4.1.2.1.1 RTOS Kernel:

The RTOS allows tasks to be defined and parameters for those tasks to be specified. A majority of task parameters specified are used by the schedulability analyzer and the SPMMT simulator; however, the kernel accesses uses three task parameters to find and run scheduled tasks:

Task Struct{

…

1. Task.Enabled – Boolean indicating whether this task has been enabled.

2. Task.Scheduled – Boolean indicating whether this task has been scheduled to run.

3. Task.Function – Function pointer indicating which function is to be run by the RTOS every time the kernel executes the task.

…}

The RTOS kernel organizes tasks within a prioritized list. Tasks are inserted into this list according to priority. Up to eight tasks may be inserted at each of eight priority levels. The RTOS does not support multiple tasks of equal priority. The number of priority levels / maximum number of tasks may be increased by the user. Each time the RTOS kernel executes, it
searches the prioritized task list starting with the highest priority task slots until it finds a scheduled task. The RTOS kernel then executes the function assigned to this task via accessing the task’s function pointer.

When any task completes execution, task completion point scanning is performed. After the completion point scanning (discussed later) has finished, the IMCRTOS kernel continues to scan through the prioritized task list executing lower priority tasks that have been scheduled. Upon completion of prioritized task list scanning, the RTOS exits the kernel and enters Idle Mode.

4.1.2.1.2 RTOS Idle Mode:

While idling, the IMCRTOS runs any idle task that has been assigned by the user via a function pointer. Furthermore, when idle the IMCRTOS runs an automatic task scheduling routine. The RTOS continuously performs both of these operations while it remains in Idle Mode. Idle mode is aborted when the automatic task scheduler schedules one or more tasks to be run.

4.1.2.1.3 Automatic Task Scheduling:

In order to determine if tasks need to be scheduled the automatic task scheduler examines all enabled tasks in the system. In particular, the automatic task scheduler looks at these parameters:

Task Struct{

...
1. REPETITION_PERIOD – The user-specified amount of time to elapse between when a task is scheduled to when that task is to be rescheduled.

2. SCHED_TIME – The time at which the task was most recently scheduled.

Each time the task scheduler executes, it grabs a copy of the current time. Further, the scheduler determines, for each task, the next time that the task needs to be scheduled by adding the task’s most recent scheduling time and repetition period. If the sum of these values exceeds the current time value, then the task is scheduled. This process is repeated for all tasks in the prioritized task list.

As each task is scheduled, its task parameters are updated to reflect the new most recent scheduling time. Flags indicating that tasks are scheduled are also updated as the tasks are scheduled. In addition, the following parameters are checked/updated for use by the schedulability analyzer as tasks are scheduled:

Task Struct{

... 

1. DCT_INTERVAL – The amount of time from when a task is scheduled to when the task is supposed to have completed.

2. DCT – The time by which the task is supposed to have completed.

...}

The scheduler updates each task’s desired completion time as it is scheduled according to its DCT interval. The DCT is set to the task’s new scheduling time (the current time) plus its
DCT interval. Furthermore, after any task is scheduled, task scheduling point scanning is performed (discussed later).

4.1.2.2 – One-Way Serial I/O

There are multiple serial ports accessible on the MCF52259 tower kit (on the 52259 board and the serial module). There is a standard RS232 connector on the serial module. Furthermore, a second (virtual) serial port may be accessed via the debugger USB cable; however, use of the virtual serial port requires use of a third-party program to access and display data sent over the virtual port. Each of these two available utilize different UARTs on the Freescale Coldfire 52259 microcontroller.

Serial I/O functionality in the form of printf() is provided by Freescale Semiconductor. This serial communication is achieved through UART functionality. The functionality provided by Freescale Semiconductor is available through the CodeWarrior debug interface. The IMCRTOS does not extend serial I/O functionality for use outside of the CodeWarrior debugger; however, this functionality can be extended by the user simply by changing the UART that is used by the printf() routine.

4.1.2.3 – Simple Timing Functionality

The timing functionality built-in to the RTOS consists of hardware-based 32-bit timer configuration on the MCF52259. Timer registers are configured such that the clock signal is divided several times to provide the desired timer resolution (one microsecond). Operations are then performed on captured timer ticks in order to return the number of microseconds that have elapsed since system restart.
Further, a timer interrupt occurs whenever the 32-bit timer overflows. The number of
timer overflows that have occurred since system restart is also recorded; however, the RTOS
does not store time in 64-bit values. This is because no test run on the system needs to run for
more than one overflow period on the divided 32-bit timer; however, hooks are in place to allow
for longer time periods to be measured.

4.1.2.4 – Automatic Time-Statistic Capture

Automatic timing functionality is woven into the IMCRTOS kernel such that each
iteration of each task running on the RTOS is timed. Individual times are summed and divided
by the number of task instances in order to determine the average task execution time.
Furthermore, worst-case execution times are calculated by retaining only the longest (or smallest
in the case of slack time) times recorded. This procedure is duplicated in order to time other
functions including OS features, etc.

Task parameters are used to store time data. The following task parameters are used by
the automatic time-statistic capture routine:

Task Struct{

…

1. WORST_CASE_SLACK_TIME – The smallest task slack time recorded.
2. TOTAL_SLACK_TIME – The sum of all task slack time recorded.
3. WORST_CASE_EXECUTION_TIME – The longest task execution time
   recorded.
4. TOTAL_EXECUTION_TIME – The sum of all task execution time recorded.
5. **TOTAL_AC_EXECUTION_TIME** – The sum of all task execution time recorded while executing the average-case path.

6. **TOTAL_WC_EXECUTION_TIME** – The sum of all task execution time recorded while executing the worst-case path.

7. **WORST_CASE_SCHEDULING_DELAY** – The longest amount of time between when the task is supposed to be scheduled to when it is scheduled.

8. **TOTAL_SCHEDULING_DELAY** – The sum of all scheduling delay for this task.

9. **NUM_SCHEDULED_INSTANCES** – The number of scheduled task instances.

10. **WORST_CASE_START_DELAY** – The longest amount of time between when the task is scheduled to run and when it actually is run.

11. **TOTAL_START_DELAY** – The sum of all start delay for this task.

Average task timing statistics are determined by dividing total execution times by the number of scheduled task instances. Other parameters are simply updated at every new task execution instance, and displayed at the end of run time. The following features of the SPMMT simulator and RTOS are measured by the RTOS’ automatic time capture functionality:

**SPMMT simulator Deadline-Based Memory Swapping**

1. Number of Memory Swaps from WCETRB SPMMT to ACETRB SPMMT

2. **WCETRB SPMMT to ACETRB SPMMT time**

3. Number of Memory Swaps from ACETRB SPMMT to WCETRB SPMMT

4. **ACETRB SPMMT to WCETRB SPMMT time**
5. Total Time Spent Executing Completion Point Scans
6. Total Time Spent Executing Scheduling Point Scans

**RTOS Scheduling**

7. Task x Total Number of Scheduled Instances
8. Task x Total Number of Run Iterations
9. Task x Worst-Case Scheduling Delay – Longest time from when task x is supposed to have been scheduled to when it was actually scheduled.
10. Task x Average Scheduling Delay – Average time from when task x is supposed to have been scheduled to when it was actually scheduled.
11. Task x Worst-Case Start Delay – Longest time from when task x has been scheduled to when task x has been run by the RTOS.
12. Task x Average Start Delay – Average time from when task x has been scheduled to when task x has been run by the RTOS.
13. Total System Idle Time – The total amount of time the RTOS has spent in Idle Mode.

**Task RT Runtime Parameters**

14. Task x Worst-Case Slack Time – For all instances of task x, the shortest time between task instance completion and task instance DCT.
15. Task x Average Slack Time -- For all instances of task x, the average time between task instance completion and task instance DCT.
16. Task x Worst-Case Execution Time – For all instances of task x, the longest execution time from task instance execution to task instance completion.

17. Task x Average Execution Time – For all instances of task x, the average execution time from task instance execution to task instance completion.

18. Task x Average Average-Case-Path Execution Time – For all average-case path instances of task x, the average execution time from task instance execution to task instance completion.

19. Task x Average Worst-Case-Path Execution Time – For all worst-case path instances of task x, the average execution time from task instance execution to task instance completion.

4.1.2.5 – Task Schedulability Analyzer

The task schedulability analyzer is run automatically by the IMCRTOS at task completion points and task scheduling points. This is achieved through the use of completion point scans and scheduling point scans, respectively. The SPMMT simulator uses the task schedulability analyzer to determine when to swap SPM allocations between WCETRB SPMMT and ACETRB SPMMT while SPM reallocation is enabled (discussed later). Further, the schedulability analyzer can also be configured to implement EDF scheduling by reprioritizing tasks according to task DCTs (discussed later).

4.1.2.5.1 Task Scheduling Point Scanning:

After any task is scheduled, task scheduling point scanning is performed in order to determine whether all tasks in the system are still able to meet their desired deadlines. This
analysis assumes that all tasks will take their worst-case execution times to execute; however, because worst-case execution times are not known, the following task parameter is used to approximate WCET:

```c
Task Struct{
    ...
    WCET_ESTIMATE – User defined WCET estimate – Determined through timing.
    ...
}
```

The WCET estimate is determined by running the IMCRTOS with memory swapping disabled. The ACETRB SPMMT is used for this test. The automatic time capture functionality records the worst-case task execution time. This time is then input by the user / tester into the WCET_ESTIMATE field before further experiments are run.

The scanner loops through every scheduled task in the system, starting with task with the highest priority, and ending with the task with lowest priority. As each scheduled task is examined, it’s WCET estimate is added to a worst-case-completion-time parameter. If at any point the worst-case-completion-time exceeds the current task’s desired completion time, the scanner concludes that all tasks will not meet their real-time deadlines. However, if the loop completes before the scanner identifies any desired completion times that cannot be met, then the scanner concludes that all tasks in the system will meet their desired deadlines.

Scheduling point scanning is only performed if SPMMT simulator memory swapping is enabled. If memory swapping between WCETRB SPMMT and ACETRB SPMMT is enabled, then the SPMMT simulator performs an ACETRB SPMMT to WCETRB SPMMT swap if and
only if one or more tasks will fail to meet its desired deadline. If all tasks will meet their desired deadlines, then no swap is performed.

4.1.2.5.2 Task Completion Point Scanning:

Task completion point scanning is performed in an identical manner to task scheduling point scanning, with a few exceptions. Completion point scanning is performed if SPMMT simulator memory swapping is enabled, or EDF scheduling is enabled. If EDF scheduling is enabled, the scanner resorts the priority task list according to task desired deadlines. If ACETRB SPMMT to WCETRB SPMMT swapping is enabled, then the SPMMT simulator performs a WCETRB SPMMT to ACETRB SPMMT memory swap if and only if all tasks will meet their desired deadlines. If a single task is expected not to meet its desired deadline, no memory swap is performed.

4.1.3 – Design of the SPMMT Simulator

The Scratch-Pad-Memory Management Technique Simulator (SPMMT simulator) is developed for the purpose of simulating and comparing task performance gains for a large variety of RT tasks under three SPMMTs:

1. ACET Reduction Based (ACETRB)
2. WCET Reduction Based (WCETRB)
3. Schedulability-Driven ACETRB/WCETRB Swapping (SDAWS)

In order to perform this function, the SPMMT simulator incorporates several components into the RTOS developed for this thesis. The components are:
A. A multitude of task parameters which are attached to tasks being run by the RTOS. These task parameters are used to tune the conditions under which the SPMMT simulator simulates experimental scenarios.

B. Global parameters which specify the operation of the SPMMT simulator and further control the performance of all simulated tasks.

C. An RT task simulator. The RT task simulator performs computational operations according to SPMMT simulator task parameters in order to approximate the amount of time required to execute task instances under the different SPM management strategies.

D. SDSPMS memory swapping functions which copy memory blocks into and out of SPM depending on the schedulability of tasks.

E. A test bench which provides functions that grant access to core parameters of the RTOS and allow for quick definition of SPMMT simulator test cases.

4.1.3.1 – SPMMT simulator Task Parameters

The goal of the SPMMT simulator is to incorporate into its RT task simulation all RT task parameters that directly affect task runtime. The principle element types considered are:

1. Computations per memory access
2. Average-case / worst-case memory accesses
3. Average-case / worst-case number of cache block loads

Computations per memory access is considered because the number of operations performed relative to the number of memory accesses can drastically affect the amount of time that a task takes to run. For example, a task that performs image processing or vector operations
has a very large number of computations per memory access; whereas, a task that copies data from one place to another will have an extremely low number of computations per memory access. Therefore, the amount of time taken to load and store data relative to the amount of computation time a task uses affects performance under different SPM management strategies, because tasks whose execution time is dominated by computations rather than memory accesses will not see much of a difference in execution time performance between any SPM management strategy.

Average-case and worst-case memory accesses are considered because they directly correlate to average-case and worst-case execution paths through a given task, and longer execution paths directly correlate to longer execution times. In the real world, there may be hundreds of different paths through a task; however, this thesis assumes that the ACETRB SPMMT considers an ILP approach including all execution paths. Therefore, the number of average-case memory accesses is equal to the total number of memory accesses through all execution paths over a use-case period of time, divided by the number of task instances that have occurred over that period.

Average-case and worst-case cache block loads are considered because CAHCE block loads add a considerable amount of time to task execution. The SPMMT simulator task simulator simulates cache block loads (discussed later). Furthermore, average and worst-case loads are considered separately because the number of variables / function calls / etc. may vary drastically between task execution paths. Cache block loads add considerable execution time at the start of tasks; however, depending on the number of cache block loads relative to the length of execution paths, cache block loads may or may not have a large effect on execution time.
The SPMMT simulator uses several different varieties of the parameters described above. The following task parameters reflect the SPMMT simulator’ need to distinguish between floating (placed into / out of SPM depending on the SPM allocation strategy employed) and static (always placed in main memory) accesses. Avertable cache block loads correspond to loads that only occur under certain SPM management strategies and not under others. Either floating / non-avertable cache block loads or floating / DRAM / SDRAM memory accesses are simulated, depending upon whether or not cache simulation is enabled.

Task Struct

{…

1. COMPUTATIONS_PER_MA – The number of arithmetic operations performed per 32bit load / 32bit store combination.

2. ACs_PER_WC – The number of times that the task executes it’s worst-case execution path.

3. STATIC_WC_CBLs – The number of cache block loads that occur every time the simulated worst-case execution path is run.

4. STATIC_AC_CBLs – The number of cache block loads that occur every time the simulated average-case execution path is run.

5. FLOATING_WC_CBLs – The number of cache block loads that occur each time the simulated worst-case execution path is run and the ACETRB SPMMT is loaded.

   Note: These loads do not occur when the WCETRB SPMMT is loaded.

6. FLOATING_AC_CBLs – The number of cache block loads that occur each time the simulated average-case execution path is run and the WCETRB SPMMT is loaded.

   Note: These loads do not occur when the ACETRB SPMMT is loaded.
7. AC_DRAM_ACCESSSES – If cache simulation is not enabled, the number of memory accesses that occur out of SDRAM when the simulated average-case execution path is run. If cache simulation is enabled, all average-case memory accesses are summed together.

8. AC_FLOATING_ACCESSSES – If cache simulation is not enabled, the number of memory accesses that occur out of either SDRAM or SPM depending on which SPM strategy is loaded when the simulated average-case execution path is run.

9. WC_DRAM_ACCESSSES – If cache simulation is not enabled, the number of memory accesses that occur out of SDRAM when the simulated worst-case execution path is run. If cache simulation is enabled, all worst-case memory accesses are summed together.

10. WC_FLOATING_ACCESSSES – If cache simulation is not enabled, the number of memory accesses that occur out of either SDRAM or SPM depending on which SPM strategy is loaded when the simulated worst-case execution path is run.

11. OVERLAP_DRAM_ACCESSSES – If cache simulation is not enabled, the number of accesses that occur out of SDRAM when any execution path is run.

12. OVERLAP_SPM_ACCESSSES – If cache simulation is not enabled, the number of accesses that occur out of SPM when any execution path is run.

…}

4.1.3.2 – Global Parameters

In addition to task parameters, global parameters are also used to tune the operation of all tasks in the system and control memory swapping. These parameters are:
1. **SPM_TO_SDRAM_MAR** – The memory access ratio between SPM and SDRAM. Because all memory accesses occur in SPM on the MCF52259, SDRAM loads and stores are simulated through repeated SPM loads and stores, according to this ratio. If cache simulation is enabled, this ratio is used to increase/decrease CAHCE block load time; otherwise, this ratio is used to control individual SDRAM access time.

2. **SPMA_MODE** – The SPMMTA uses this variable to keep track of the current SPM allocation strategy, either WCETRB SPMMT or ACETRB SPMMT.

3. **SPM_SWAP_TIME** – The simulated amount of time required to perform an ACETRB SPMMT to WCETRB SPMMT or WCETRB SPMMT to ACETRB SPMMT memory swap for a single task.

### 4.1.3.3 – SPMMT simulator RT Task Simulator

The SPMMT simulator operates by running specially configured tasks. Each of these tasks executes either a worst-case or an average-case path simulation function.

**4.1.4.3.1 Execution Path Determination**

For each instance of these tasks that are run, the SDAWS determines whether to run an average-case path simulation or a worst-case path simulation. This is done by examining the task’s average-case to worst-case ratio discussed in Component A of this section. Statistics are stored corresponding to the number and type of run task instances and used to determine the type of execution path to run.

Task Struct{

...
1. LAST_RUN_TYPE – Either an AC or a WC path execution.
2. AC_EXECUTIONS – The total number of average-case executions.
3. WC_EXECUTIONS – The total number of worst-case executions.
4. AC_EXECUTION_COUNTER – A counter used to determine whether an AC or WC execution path simulation needs to run (compared to the average-case to worst-case path ratio).

4.1.4.3.2 Cache Simulation

If cache simulation is enabled, the SPMMT simulator simulates CBLs as soon as either the average-case or worst-case execution path is run. Depending on which path is run, the simulator uses the average-case or worst-case static and floating task parameters to determine the number of CBLs to simulate. Avertable CBLs are performed only if the SPM strategy (ACETRB SPMMT or WCETRB SPMMT) associated with the current execution path time (average-case or worst-case) is not loaded.

For each CBL that must be simulated, the SPMMT simulator simulates block load and block store operations. What is simulated is not the precise functionality of cache, but merely the amount of time that it would take cache to “replace” the number of blocks indicated by the task parameters. Cache block load time simulation is achieved using loops to perform SPM loads / stores according to the SPM to SDRAM memory access ratio. Note here that the memory access ratio is used only on the first load / store, due to the serial nature of loading and storing blocks of memory. Loops are unrolled in order to minimize overhead.

4.1.4.3.3 Average-case / Worst-case Path Simulation
After cache simulation is performed, the task simulator runs an average-case or worst-case execution path simulation. The execution path simulation is achieved by performing loads, arithmetic operations, and stores on memory locations according to the number of memory accesses: (AC, WC, or Overlap) specified within the task being run. The following parameters are used to access memory:

1. SPM_FLOATING_MEMORY_LOC – A 32-bit pointer to the location in SPM that is assigned to this task’s floating memory.
2. AC_DRAM_FLOATING_MEM_LOC – A 32-bit pointer to the location in SDRAM that is assigned to this task’s floating average-case execution path memory space.
3. WC_DRAM_FLOATING_MEM_LOC – A 32-bit pointer to the location in SDRAM that is assigned to this task’s floating worst-case execution path memory space.
4. AC_FLOATING_MEMORY – A 32-bit pointer to the current location of this task’s floating average-case execution path memory (either in SPM or in SDRAM).
5. WC_FLOATING_MEMORY – A 32-bit pointer to the current location of this task’s floating worst-case execution path memory (either in SPM or in SDRAM).
6. FLOATING_MEM_SIZE – The size of floating memory for this task. This size is the same for both AC and WC floating memory (because the amount of SPM that may be used is the same for both cases).
7. OVERLAP_SPM_MEMORY – A 32-bit pointer to this task’s memory that is always located in SPM.
8. OVERLAP_SPM_SIZE – The amount of memory always kept in SPM for this task.
9. **AC_DRAM_MEMORY** – A 32-bit pointer to this task’s average-case execution path memory that is always located in SDRAM.

10. **AC_DRAM_SIZE** – The amount of average-case execution path memory that is always located in SDRAM.

11. **WC_DRAM_MEMORY** – A 32-bit pointer to this task’s worst-case execution path memory that is always located in SDRAM.

12. **WC_DRAM_SIZE** – The amount of worst-case execution path memory that is always located in SDRAM.

13. **OVERLAP_DRAM_MEMORY** – A 32-bit pointer to this task’s memory that is always located in SDRAM.

14. **OVERLAP_DRAM_SIZE** – The amount of this task’s memory that is always located in SDRAM.

The simulator runs the average-case or worst-case memory accesses by loading a number of 32-bit blocks corresponding to the number of average-case or worst-case accesses plus the number of overlap accesses. If the number of accesses exceeds the size of its corresponding memory space, the simulator loops back to the beginning of the memory space and continues loading from there. As each value is loaded, an arithmetic operation is performed on the memory block. The following operations are performed on every ten 32-bit blocks that are loaded:

1. Four Addition Operations
2. Two Subtraction Operations
3. Three Multiplication Operations
4. One Division Operation
The numbers of each type of operation are chosen according to average programming heuristics; however, the types of operations performed do not affect the task run-time significantly for small C per MA ratios (See results). Finally, the memory block is stored. This process continues for all accesses specified in the task’s parameters.

4.1.3.4 – SDSPMS Memory Swapping Function

The SPMMT simulator performs memory swapping through the use of “memcpy()” function calls. Memory swaps are performed on each task in the RTOS one at a time. Specifically, when an ACETRB SPMMT to WCETRB SPMMT swap is performed, the SPMMT simulator copies data in the SPM floating memory space to the AC SDRAM floating memory space, and then copies WC SDRAM floating memory to the SPM floating memory space. Conversely, when a WCETRB SPMMT to ACETRB SPMMT swap is performed, the SPMMT simulator copies data in the SPM floating memory space to the WC SDRAM floating memory space, and then copies AC SDRAM floating memory to the SPM floating memory space.

The “memcpy()” function is implemented as a loop, which copies bytes of memory one at a time. This is not ideal or optimized for the MCF52259 platform. Further, it is possible to create an optimized “memcpy()” function, or to utilize DMA to swap memory blocks asynchronously. Furthermore, the precise amount of time required to swap memory blocks depends upon a multitude of factors including the size of SPM, the amount of SPM allocated to each task, main memory access time, memory bus use, etc. Therefore, this thesis tests different SPM swap times by changing the amount of SPM allocated to each task. Smaller amounts of SPM result in fewer “memcpy()” calls, which in turn results in reduced swap time.
4.1.3.5 – SPMMT simulator Test Bench

The SPMMT simulator Test Bench consists of a header file, which is modified to change task parameters, and a C file containing access function to global RTOS and simulator parameters. Specifically, task parameters are set using a series of #defines. By modifying the values associated with each of the #defines for a given task, the corresponding task parameters are automatically updated at runtime because the task initializer loads the #defined values into the corresponding task parameters as default values.

Furthermore, the SPMMT simulator Test Bench allows for memory swapping to be enabled or disabled, cache simulation to be enabled or disabled, and the SPM to SDRAM MAR to be modified. In addition, the SPMMT simulator Test Bench provides functions that automatically initialize and run subsets of tasks in the system according to different SPM allocation strategies.

4.2 – VALIDATION METHODOLOGY

The purpose of this portion of the Chapter is to completely lay out how each software component developed for this thesis is tested to verify desired operation. The test methodology portion is split into two sections:

- Section 1: This section defines and describes how the functionality of the RTOS is to be tested, and defines the conditions under which tests are to be run.
- Section 2: This section defines and describes how the functionality of the SPMMT simulator is to be tested and how simulation results are to be verified.
4.2.1 – RTOS Validation

The purpose of this section of the test methodology is to ensure the proper operation of all portions of the Real-Time Operating System developed for this thesis. In particular, each primary element of the RTOS is tested separately to ensure correctness. Element-wise tests of the RTOS are split into test-cases.

A. Validating The RTOS Kernel
B. Validating I/O Functionality
C. Validating Simple Timing Functionality
D. Validating Automatic Time-Capture Functionality
E. Validating The Schedulability Analyzer

4.2.1.1 - Validating the RTOS Kernel

In order to verify that the RTOS is able to run tasks and schedule them, the WCET and ACET of each task are determined by running individual task instances on the RTOS and recording the respective ACET and WCET. Instances of tasks with different ACETs and WCETs are then run on the RTOS simultaneously to verify that the RTOS switches to the correct task (the one with the highest priority), and that the RTOS does not introduce significant and unnecessary overhead. This is tested through automatic capture of timing parameters:

1. Total System Idle Time – The total amount of time that the system was not running tasks or performing other operations. **Important Note:** The RTOS manages task automatic rescheduling while idling; therefore, task rescheduling is
included in all system idle times captured during testing – it is not counted as overhead.

2. **Total Task Execution Time** – The total amount of time that each task has spent executing throughout all of its instances.

By comparing the sum of total task execution times to the total system idle time, the amount of overhead incurred by the RTOS is measured. The RTOS kernel is modified until the overhead introduced by the RTOS is satisfactory. However, because RTOS overhead increases as the number of tasks and the repetition frequency of each task increases, a scenario of five tasks each with a 100 millisecond repetition period is used to benchmark RTOS performance.

Automatic task scheduling functionality is tested by loading tasks with fixed priorities into the RTOS, and running them simultaneously in order to determine the effectiveness of the RTOS in rescheduling the tasks when they need to run, and running them when they are scheduled to run. In particular, timing parameters are automatically recorded:

1. **Average and Worst-Case Scheduling Delay** – The time from which a task should have been scheduled (according to its periodicity), to when the task is actually scheduled by the RTOS.

2. **Average and Worst-Case Start Delay** – The time from which a task has been scheduled to run by the RTOS to when the RTOS actually gets around to running the scheduled task.

Furthermore, the number of scheduled instances and the number of run instances of each task is automatically recorded by the RTOS in order to ensure that the expected number of task
instances (based on task periodicity) has been satisfied. For example, a task that is configured to run once every 100 milliseconds is expected to run 100 times over a ten-second interval.

4.2.1.2 - Validating I/O Functionality

Because basic I/O interface functionality is provided by Freescale Semiconductor via serial communication over USB using a UART, testing of the I/O interface is not extensive. In particular, several strings of characters of varying lengths are transmitted to the CodeWarrior debug terminal.

4.2.1.3 - Validating Simple Timing Functionality

Timing is tested in two ways: Externally via a stopwatch, and internally relative to other captured times. For the stopwatch test, times are displayed via the serial I/O at one-second intervals, and compared to the values on the stopwatch. Timer dividers are checked to ensure that displayed times should and do match stopwatch values (there is additional delay due to the serial I/O display latency).

Internally, the timing functionality is checked by running individual instances of different tasks inside of the main routine (without the RTOS). Times are captured and compared in order to determine total task run time. Task instances are then placed in a loop in which tasks are run continuously (without the RTOS) for a desired period of time (i.e.: 10 seconds, etc.). By comparing the number of task instances that are run over this period to the task execution time, timing functionality is verified.
Further testing of the timing functionality is performed by timing the amount of time that it takes to get the current time in microseconds, and ensuring that this time is less than one-tenth of a microsecond, so as not to impact the results of other testing significantly.

4.2.1.4 - Validating Automatic Time-Capture Functionality

The automatic time-capture functionality of the RTOS is tested by separately testing each of the individual time and instance statistics captured. In particular, C code is inserted into three separate regions of the RTOS in order to capture statistics; therefore, verifying time-capture functionality consists of explicitly capturing simple time data in microseconds and manually computing average times / worst-case execution times in the following locations:

1. The SPMMT simulator Deadline-Based Memory Swapper
   a. Number of Memory Swaps from WCETRB SPMMT to ACETRB SPMMT
   b. WCETRB SPMMT to ACETRB SPMMT time
   c. Number of Memory Swaps from ACETRB SPMMT to WCETRB SPMMT
   d. ACETRB SPMMT to WCETRB SPMMT time
   e. Total Time Spent Executing Completion Point Scans
   f. Total Time Spent Executing Scheduling Point Scans

2. RTOS Scheduler
   a. Task x Total Number of Scheduled Instances
   b. Task x Total Number of Run Iterations
   c. Task x Worst-Case Scheduling Delay
   d. Task x Average Scheduling Delay
3. RTOS Kernel
   a. Task x Worst-Case Slack Time
   b. Task x Average Slack Time
   c. Task x Worst-Case Execution Time
   d. Task x Average Execution Time

4.2.1.5 - Validating the Schedulability Analyzer

The schedulability analyzer is validated through the use of processor breakpoints and timing analyses. Schedulable and un-schedulable task sets are run on the processor, and task slack times are measured to ensure schedulability. Further, breakpoints are inserted within schedulability analyzer code in order to determine if the schedulability analyzer detected schedulable or un-schedulable task instances for each set of tested tasks.

4.2.2 – SPMMT simulator Validation

The purpose of this section of the validation methodology is to ensure the proper operation of all portions of the SPMMT simulator developed for this thesis. In particular, each primary element of the SPMMT simulator is tested separately to ensure correctness. Element-wise tests of the SPMMT simulator are split into test-cases.

A. Validating Task Parameter Operation
4.2.2.1 - Validating Task Parameter Operation

Task parameter validation is performed by individually testing each task parameter to ensure two things (in the following list). Both of these tests are performed using the built-in USB debugger on the MCF52259 within CodeWarrior.

1. That changing the task parameter throughout the desired range of experimental testing does not result in errors at run-time.
2. That changes to task parameters result in changes to task run-time that are within an anticipated range.

4.2.2.1.1 Error-Checking and Parameter Range

For the first test, each task parameter is manually changed in the test bench header file several times, such that all necessary parameter values are checked (via visual inspection). For example, with respect to CBLs, CBL values of between 0 and 100 are checked. Because CBLs of greater than 100 will not be used for experimental purposes, greater values are not tested. For each time that a task parameter is edited, the simulator is run and timing statistics for the task whose parameters have been modified are captured. Results of timing analyses are checked to ensure no anomalies (e.g.: that times did not decrease when they were expected to increase, or decrease when they were expected to decrease). Further, the debugger is used to step-through
execution to ensure that the appropriate number of iterations are performed for changes in
parameter values, and that the appropriate paths are executed / functions are called.

4.2.2.1.2 Result-Checking

For the second test, task parameters are again modified within the test bench header file
several times. However, instead of modifying parameters such that tests span the range of
necessary values to test, parameters are modified by orders of magnitude (x10, x2, etc). For each
increase in parameter magnitude, the task simulator is run. Timing statistic output is examined in
order to determine whether changes to the parameters resulted in a large or small increase or
decrease in task run-time. Changes to different parameters are made and timing results are
compared to one another for each test in order to ensure similar magnitude increases for
parameters which have a similar impact on task run-time.

4.2.2.2 - Validating Global Parameter Operation

Global parameter operation is verified in a very similar fashion as task parameter
validation. However, because there are only three global parameters, each is tested separately:

1. **SPM TO SDRAM MAR Validation** – The operation of the memory access ratio between
SPM and SDRAM is validated by changing the ratio in increasing orders of magnitude and
measuring the effect that these changes have on task execution time while all other task
parameters remain constant (with the number of memory accesses set to 100 for each task).
This is performed with and without cache simulation enabled. In this way, the effect of MAR
on execution time can be directly measured and checked against expected time increases.
2. **SPMA_MODE Validation** – The operation of the SPM mode is validated through the use of the debugger to ensure that, for every task execution, the proper SPM mode is enabled. In addition, the number of SPM swaps is recorded to ensure that the expected number of ACETRB SPMMT to WCETRB SPMMT and WCETRB SPMMT to ACETRB SPMMT memory swaps occurs for certain task sets.

3. **SPM_SWAP_TIME Validation** – The SPM swap time operation does not require complex validation. Timing statistic gathering is inserted into the code, which measures the amount of time that is spent performing memory swaps. This amount of time is compared to the desired memory swap time specified by the user.

4.2.2.3 - **Validating the RT Task Simulator**

Because task parameters are tested separately, the effect of changes to task parameters, and the total task execution time is not considered when validating the RT task simulator, as this was done in test cases A and B. Instead, the validation of the RT task simulator consists of the following tests:

1. **Validation of Execution-path Determination**

2. **Validation of cache Simulation**

Execution-path determination is validated using the CodeWarrior debugger. A single task is loaded into the RTOS, and run. Breakpoints within the debugger are manually used to count the number of times this task runs / reaches the execution-path determination point. Further breakpoints are used to count the number of times that a worst-case or average-case execution path have been run. After execution is complete, the number of average-case and worst-case
execution paths indicated by the debugger breakpoints is compared to the average-case to worst-case ratio specified for the task in the test bench header file.

Validation of proper cache simulation is also done using the debugger. Again, a single task is loaded into the RTOS and run. Debugger breakpoints are used to count the number of times that the task reaches each cache simulation point within the code. Breakpoints are used to verify the following:

1. Cache simulation does not run when it is disabled.
2. Cache simulation properly handles avertable CBLs: That it does not simulate CBLs for matching execution-path / SPM allocation strategy pairs.
3. Appropriate number of loop iterations are performed according to the number of CBLs specified by the task parameters and MAR.

4.2.2.4 - Validating Memory Swapping Functionality

There are two versions of memory swapping utilized by the SPMMT simulator. The first type of memory swapping is simulated using user-specified time. This is tested when the global parameter for memory swap time is tested; therefore, only the non-simulated memory swapping functionality is validated.

Non-simulated memory swapping functionality is validated using the CodeWarrior debugger. Two tasks are loaded into the RTOS and run. The tasks are defined such that memory swapping will be performed. Further, this test is performed with both memory swapping enabled and memory swapping disabled to validate the operation of the memory swap enable / disable feature. For each ACETRB SPMMT to WCETRB SPMMT memory swap, several random
memory addresses are compared before / after the swap to ensure that it completed successfully. This process is repeated for each WCETRB SPMMT to ACETRB SPMMT memory swap.

4.2.2.5 - Validating Test Bench Operation

Because the modification of task parameters using the test bench header file is tested in test-case A, test bench operation is validated by ensuring the proper operation of the access functions and utility functions provided by the test bench. Each of these functions is tested using the debugger. Test bench functions are used to specify test cases to run and global parameters to be changed. A single task is loaded and run for each set of test bench functions being validated, and breakpoints in the code are used to stop execution and examine the contents of variables to ensure that the correct values have indeed been loaded into memory.

4.3 – EXPERIMENT METHODOLOGY

The experiment methodology portion of the Chapter describes how the SPMMT simulator is used to compare the different SPM management strategies (ACETRB SPMMT, WCETRB SPMMT, SDSPMS) under different RT task conditions. In particular, test cases are defined and the reasoning behind each one is examined. This portion of the Chapter is split into two sections:

- Section 1 – The One-Task Experiment Methodology: Defines experiments to record real-time task execution time data for simulations of individual tasks running on the RTOS under various conditions with varying parameters. This allows a baseline to be established.
4.3.1 – The One-Task Experiment Methodology

The goal of the one-task experiments is to observe the effect of changes to SPMMT simulator task parameters on task runtime performance. Specifically, each of the three primary task attributes is independently tested:

- Experiment Set A – Average-case / worst-case memory access tests
- Experiment Set B – Average-case / worst-case cache block load tests
- Experiment Set C – Computations per memory access tests
- Experiment Set D – SDRAM to SRAM Equivalency Ratio

For all of the experiments, parameters are increased by orders of magnitude so that overall trends can be determined (dominance of one parameter on execution time, etc.). Furthermore, all of these tests are performed under both the ACETRB SPMMT and the WCETRB SPMMT strategies. The SDSPMS strategy is not employed, because the task is always configured such that its deadline will be met.

The primary purpose of the one-task experiments is to test performance metrics that are not affected by real-time schedulability conditions. However, a secondary purpose is to provide a
basis for comparison for the multi-task experiments. The one-task experiments provide insight into how task parameter ratios influence execution time, and when certain task parameters dominate execution time. Furthermore, the one-task experiments serve to determine when one SPM management strategy always performs better or worse than the other.

4.3.1.1 - Independent Variables – Average and Worst-case Memory Accesses

For the average and worst-case memory access experiment, three levels of memory accesses are tested: small, medium and large.

- Small: 100 Accesses
- Medium: 1,000 Accesses
- Large: 10,000 Accesses

Furthermore, four experiments are performed. The first two experiments utilizes an ACET to WCET ratio of 1:1. The second two experiments utilizes an ACET to WCET ratio of 1:2. For each set of two experiments, CBLs are modified. For the first experiment in each set, CBLs are set to 2 ACET Floating, 2 ACET Static, 2 WCET Floating, and 2 WCET static. Furthermore, the C per MA ratio is 2:1. For the second experiment in each set, all CBLs are set to 0 to indicate a system where all loads occur out of SPM. Because all loads occur in SPM, only the ACETRB SPMMT strategy is used for the second set of tests. The following experiments are performed:

1. CBL Set # 1. ACETRB SPMMT. # Accesses = Small, Medium, Large.
2. CBL Set # 2 (No CBLs). ACETRB SPMMT. # Accesses = Small, Medium, Large.
3. CBL Set # 1. WCETRB SPMMT. # Accesses = Small, Medium, Large.
4. CBL Set #2 (No CBLs). WCETRB SPMMT. # Accesses = Small, Medium, Large.

4.3.1.2 - Independent Variables – Average and Worst-case CBLs

Because there are many different combinations of CBLs that all differently affect execution time depending on which SPM management strategy is employed, several experiments are performed which test CBLs. For all of these experiments, the C per MA ratio is set to 2:1. The number of memory accesses is changed for the second segment of testing. The following experiments are performed:

1. WCET:ACET Ratio = 1:1. ACETRB SPMMT. AC/WC Non-avertable CBLs = 2/4, 8/16, 32/64
2. WCET:ACET Ratio = 1:1. WCETRB SPMMT. AC/WC Non-avertable CBLs = 2/4, 8/16, 32/64
3. WCET:ACET Ratio = 2:1. ACETRB SPMMT. AC/WC Non-avertable CBLs = 2/4, 8/16, 32/64
4. WCET:ACET Ratio = 2:1. WCETRB SPMMT. AC/WC Non-avertable CBLs = 2/4, 8/16, 32/64
5. WCET:ACET Ratio = 1:1. ACETRB SPMMT. AC/WC Avertable CBLs= 2/4, 8/16, 32/64
6. WCET:ACET Ratio = 1:1. WCETRB SPMMT. AC/WC Avertable CBLs= 2/4, 8/16, 32/64
7. WCET:ACET Ratio = 2:1. ACETRB SPMMT. AC/WC Avertable CBLs= 2/4, 8/16, 32/64
8. WCET:ACET Ratio = 2:1. WCETRB SPMMT. AC/WC Avertable CBLs= 2/4, 8/16, 32/64

4.3.1.3 - Independent Variable – Computation per Memory Access

The number of computations per memory access affects the total amount of task execution time, and also the ratio of total execution time to CBL-based execution time. For all of these tests, CBLs are set to 2 ACET Floating, 2 ACET Static, 2 WCET Floating, and 2 WCET static. Two experiments are performed:

1. 1000 ACET:2000 WCET Accesses ACETRB SPMMT C:MA = 1:1, 2:1, 5:1, 10:1, 20:1
2. 10000 ACET:20000 WCET Accesses ACETRB SPMMT C:MA = 1:1, 2:1, 5:1, 10:1, 20:1

4.3.1.3 - Independent Variable – SDRAM to SRAM Equivalency Ratio

The SDRAM to SRAM equivalency ratio directly affects CBL time. As the ratio increases, the dominance of CBL time on task execution time increases. For all of these tests, CBLs are set to 2 ACET Floating, 2 ACET Static, 2 WCET Floating, and 2 WCET static. C to MA is set to 2:1. One experiment is performed for SDRAM:SRAM = 25, 50, 100, 200, 400.

4.3.2 – The Multi-Task Experiment Methodology

The goal of the multi-task experiments is to observe the effect of changes to task runtime and scheduling conditions, and the effect of different SPM memory allocation strategies on task execution time and slack time performance. Testing each task parameter individually for each of several tasks for each of several SPM allocation strategies is infeasible because it would require thousands of test cases. Therefore, experiments which show the effect of several different types
of changes on task parameters are performed in order to show trends only. Specifically, the following experiments are performed.

- Experiment Set A – The effect of ACET:WCET ratio and average-cases per worst-case on ACETRB and WCETRB performance.
- Experiment Set B – The effect of worst-case processor utilization on RT performance with the WCETRB, ACETRB, and SDAWS SPMMTs. This is the most comprehensive experiment set performed for this thesis.
- Experiment Set C – The effect of memory swap time on SDAWS performance.
- Experiment Set D – The effect of task desired deadlines on SDAWS performance.

For all of the experiments, parameters are increased by orders of magnitude so that overall trends can be determined, e.g., dominance of one parameter on execution time. For tests performed within each experiment, average and worst-case execution time and slack time are obtained. These times are compared to one another to determine when different SPM allocation strategies may be employed to increase WCET or ACET performance.

The primary purpose of the multi-task experiments is to answer the following questions:

1. When should the WCETRB SPMMT be used instead of the ACETRB SPMMT?
2. How much ACET improvement the ACETRB SPMMT holds over the WCETRB SPMMT in each of many different scenarios?
3. How much slack-time improvement the WCETRB SPMMT holds over the ACETRB SPMMT in each of many different scenarios?
4. Under what scenarios the SDAWS achieves smaller slack times than ACETRB SPMMT, but faster ACETs than WCETRB SPMMT?
5. Under which scenarios does the SDAWS achieve the greatest performance?

3.4.2.1 - Independent Variables – ACET:WCET Ratio and Average-Cases per Worst-Case

The ACET:WCET ratio and the number of average-case path executions per worst-case path execution affect average execution time reduction (not ACET). Therefore, it is necessary to change these parameters to determine overall performance gains that can be achieved through the use of the ACETRB SPMMT. All of these tests are performed with a C to MA ratio of 2:1. For all of these tests, CBLs are set to 2 ACET Floating, 2 ACET Static, 2 WCET Floating, and 2 WCET static. ACET:WCET experiments are performed for 9 average case path executions per worst-case. Average-cases per worst-case experiments are performed with an ACET:WCET of 1:2. All of these experiments are executed with two identical tasks running on the processor. The following experiments are performed:

1. ACET:WCET Experiment: ACETRB SPMMT ACET:WCET = 1:1, 1:2, 1:4
2. ACET:WCET Experiment: WCETRB SPMMT ACET:WCET = 1:1, 1:2, 1:4
3. ACs per WC Experiment: ACETRB SPMMT ACs per WC = 9, 3, 1
4. ACs per WC Experiment: WCETRB SPMMT ACs per WC = 9, 3, 1

3.4.2.2 - Independent Variable – Worst-Case Processor Utilization

A large number of worst-case processor utilization experiments are performed in order to measure slack time performance of tasks under the various SPMMTs. Some of these experiments utilize two tasks, some utilize three tasks, and others utilize a variable number of tasks. All of these experiments are performed with a C to MA ratio of 2:1. For all of these experiments, CBLs are set to 2 ACET Floating, 2 ACET Static, 2 WCET Floating, and 2 WCET
static. Further, these experiments use an ACET:WCET ratio of 1:2, and one average case path execution for every worst-case path execution (to increase worst-case task stack-up frequency).

IMPORTANT NOTE: The worst-case processor utilization used for these experiments is calculated according to the ACETRB SPMMT to ensure a fair basis for comparison.

3.4.2.2.1 – 2 Identical Task Experiments: Proc Utilization Via Varied Task Repetition Periods

1. ACETRB SPMMT. Proc Utilization = 25%, 50%, 75%, 90%.
2. WCETRB SPMMT. Proc Utilization = 25%, 50%, 75%, 90%.
3. SDAWS SPMMT. Proc Utilization = 25%, 50%, 75%, 90%.

3.4.2.2.2 – 2 Identical Task Experiments: Proc Utilization Via Varied Task Execution Times

1. ACETRB SPMMT. Proc Utilization = 25%, 50%, 75%, 90%.
2. WCETRB SPMMT. Proc Utilization = 25%, 50%, 75%, 90%.
3. SDAWS SPMMT. Proc Utilization = 25%, 50%, 75%, 90%.

3.4.2.2.3 – Processor Utilization by Varied Numbers of Tasks

1. ACETRB SPMMT. # Tasks(Proc Util) = 2(24%), 3(36%), 4(48%), 6(73%), 8(97.5%)
2. WCETRB SPMMT. # Tasks(Proc Util) = 2(24%), 3(36%), 4(48%), 6(73%), 8(97.5%)

3.4.2.2.4 – 3-Task, Aperiodic Distortion Experiments (Two Small, Frequent, Low Priority Tasks, 1 Large, Infrequent, High Priority Task)

1. ACETRB SPMMT. Proc Utilization = 97%
2. WCETRB SPMMT. Proc Utilization = 97%
3. SDAWS SPMMT. Proc Utilization = 97%
3.4.2.3 - Independent Variable – SPM Swap Time

SPM swap time is the primary factor affecting the ability of the SDAWS SPMMT to serve as a compromise solution between the ACETRB and WCETRB SPMMTs. The following experiment is performed as an extension of the 3-Task, aperiodic task experiments. As with those experiments, this experiment is performed with a C to MA ratio of 2:1. CBLs are set to 2 ACET Floating, 2 ACET Static, 2 WCET Floating, and 2 WCET static. Further, these experiments use an ACET:WCET ratio of 1:2, and one average case path execution for every worst-case path execution (to increase worst-case task stack-up frequency).

SPM swap time itself it not enough to characterize the performance of the SDAWS SPMMT. This is because what matters is the ratio between SPM swap time and worst-case task stack up WCET. Therefore, the following SPM swap time to WCET ratios are tested (in terms of SPM swap time as a percentage of WCET): 0.17%, 0.45%, 0.87%, 1.7%, 6.31%, and 8.4%.

3.4.2.4 - Independent Variable – Desired Deadline Threshold

The task desired deadline threshold affects when the RTOS determines when to perform SPM swaps. A small desired deadline threshold (e.g.: 50% of task the task deadline), results in more SPM swaps, while a large desired deadline threshold (e.g.:85% of task deadline), results in fewer SPM swaps. As with other experiments, this experiment is performed with a C to MA ratio of 2:1. CBLs are set to 2 ACET Floating, 2 ACET Static, 2 WCET Floating, and 2 WCET static. Further, these experiments use an ACET:WCET ratio of 1:2, and one average case path execution for every worst-case path execution (to increase worst-case task stack-up frequency).

This experiment utilizes five identical tasks with different periodicities and priorities running simultaneously on the RTOS at 95% worst-case processor utilization. Because this
experiment tests SDAWS slack time performance versus the ACETRB and WCETRB SPMMTs, a large memory swap time to WCET ratio is utilized (at a ratio of 1:12). The experiment is performed for the following deadline thresholds: 45%, 65%, 85%.
CHAPTER 5 – RESULTS AND ANALYSIS

This chapter summarizes the results obtained from the experiments conducted using the RTOS and SPMMT simulator. The results displayed in this chapter highlight differences between the ACETRB and WCETRB SPMMTs and quantify the performance of the SDAWS SPMMT. A majority of the results of experiments performed for this thesis are not displayed in this chapter — only a subset of the most pertinent pieces of data are presented in order to show important trends. It is outside of the scope of this thesis to test all possible real-time/resource-constraint scenarios; therefore, while results from each experiment indicate performance trends / ranges, the precise microsecond values displayed are not indicative of any particular system. (They are artifacts of the Freescale Coldfire MCF5225 development kit on which the SPMMT simulator was run.)

While independent variables were adjusted over a wide range for each experiment, the entire range of possible values could not be covered for every test. Each experiment measures the effect of a single independent variable. Experiments are not run for every possible independent variable combination – because this would require hundreds of additional experiments. However, the trends displayed in the results of this section indicate the effect of parameter changes on performance, so a trend shown in one experiment – showing the effect of a certain parameter – would carry-over to any other experiment involving changes to that same parameter.

Simulator is used to refer to the SPMMT simulator in this section. Experiment refers to a set of tests measuring the effect of change to an independent variable. Test refers to an individual test for a specific parameter set. Because this thesis performed an extensive set of experiments, created unique metrics, and used them to evaluate the performance of different SPMMTs, the results and analysis chapter is split into three sections.
1. **INTRODUCTION**: The introduction defines the metrics used to evaluate SPMMT performance and briefly summarizes findings from all experiments. This section also lays out which experiments will be discussed in the subsequent sections.

2. **COMPARISON OF ACETRB AND WCETRB SPMMTs**: This section describes the performance differences between the ACETRB and WCETRB SPMMTs. The SPMMTs were evaluated under resource-unconstrained conditions using single task experiments and under resource-constrained conditions via multi-task experiments. Further, performance of each of these SPMMTs translates directly to performance of SDAWS while operating under the corresponding (ACETRB or WCETRB) SPM mode.

3. **THE EFFECT OF SDAWS OVERHEAD ON SDAWS PERFORMANCE**: This section shows the effect of RTOS overhead incurred due to SDAWS. This includes schedulability-analysis overhead and SPM swapping overhead. This part of the chapter summarizes the results of all experiments used to evaluate the overhead performance of SDAWS.

### 5.1 – INTRODUCTION

The introduction section consists of three parts:

1. **Metric Definitions**: Carefully defines each of the metrics utilized by this thesis, how they are used, and why they are used.

2. **SPMMT-Independent Simulation Parameters**: Explains the effect of simulator parameters that affect task execution time equally regardless of the SPMMT loaded.
3. **Experiment List and Brief Summary of Findings**: Lists each experiment performed for this thesis and briefly describes the significant results.

### 5.1.1 – Metric Definitions

This thesis defines several different metrics that are used to characterize SPMMT performance for different experiments. Each of these metrics is explained and is defined by an equation. The use for each metric is described as well. (Note that the equations in this section do not include normalization to 100%.)

#### 5.1.1.1 – ACETRB SPMMT: Percent ACET Reduction

The percent ACET reduction (equation 9) is the average-case execution path speedup due to the ACETRB SPMMT. ACET reduction under the ACETRB SPMMT is measured relative to ACET under the WCETRB SPMMT.

\[
\text{Percent ACET Reduction} = 1 - \frac{ACET(\text{ACETRB})}{ACET(\text{WCETRB})}
\]  

(9)

Percent ACET reduction is used to characterize performance improvements of the ACETRB SPMMT versus the WCETRB SPMMT.

#### 5.1.1.2 – WCETRB SPMMT: Percent WCET Reduction

The percent WCET reduction (equation 10) is the worst-case execution path speedup due to the WCETRB SPMMT. WCET reduction under the WCETRB SPMMT is measured relative to WCET under the ACETRB SPMMT.
Percent WCET Reduction = 1 – \( \frac{WCET(WCETRB)}{WCET(ACETRB)} \) 

(10)

Percent WCET reduction is used to characterize performance improvements of the WCETRB SPMMT versus the ACETRB SPMMT.

5.1.1.3 – ACETRB SPMMT: Percent Average Execution Time Decrease

While ACET and WCET only depend upon execution time for particular execution paths through code, average execution time depends upon execution time through all execution paths. The average execution time (equation 11) is the average amount of time that it took for a single task instance to complete execution.

\[
\text{Average Execution Time (AET)} = \frac{\text{TotalTaskExeTime}}{\text{Number of Task Instances}}
\]

(11)

Percent average execution time (AET) reduction (equation 12) is the average task instance speedup due to the ACETRB SPMMT.

\[
\text{Percent AET Reduction} = 1 - \frac{AET(ACETRB)}{AET(WCETRB)}
\]

(12)

Percent AET reduction is used to determine whether it is worthwhile to use the ACETRB SPMMT.

5.1.1.4 – Percent Slack Time Increase

*Slack time* is the amount of time between when a task completes and the hard deadline for that task. The WCETRB and SDAWS SPMMTs offer increased slack time versus the ACETRB SPMMT. Therefore, slack time increases are measured with respect to the ACETRB SPMMT.
The percent slack time increase (equation 13) is the amount of additional slack time gained through the use of either the WCETRB or SDAWS SPMMTs.

\[
\text{Percent Slack Time Increase} = \frac{\text{Slack Time (WCETRB or SDAWS)}}{\text{Slack Time (ACETRB)}} - 1
\]

(13)

Slack time is used to characterize task schedulability performance gains of the WCETRB and SDAWS SPMMTs versus the ACETRB SPMMT.

5.1.1.5 – RTOS Overhead Percentage

RTOS overhead is defined as any execution time that is not spent executing task simulations, which includes timing analyses, kernel operation, etc. Overhead is computed from three types of measured times: total task execution time (i.e., the sum of all task execution times for a single task), idle time (i.e., the amount of time the processor spends performing no operations), and total runtime (i.e., the time from when a test begins to when the test stops running). The percent RTOS overhead (equation 14) is the average percentage of processor time that is spent executing overhead functions.

\[
\text{Percent RTOS Overhead} = 1 - \frac{(\text{Idle Time} + \sum_{0}^{n} \text{Total Task Exe Time}[i])}{\text{Total Runtime}}
\]

(14)

RTOS overhead is used to characterize the performance of the SDAWS SPMMT versus the ACETRB and WCETRB SPMMTs.

5.1.1.6 – Worst-Case Processor Utilization

The worst-case processor utilization is the percentage of processor time that must be spent executing tasks in order for tasks to meet their deadlines under a worst-case task stack-up.
This work defines a *worst-case task stack-up* as a scheduling event in which all tasks are scheduled to run at the same time, and all tasks take their WCETs to run. The worst-case task slack time is the smallest slack time that occurs out of all task instances run. The desired completion time (DCT) of the most frequent task is the amount of time that all tasks scheduled during a worst-case task stack-up had to complete; therefore, the worst-case processor utilization (equation 15) is the percentage of the most frequent task’s DCT that is spent executing tasks under a worst-case task stack-up.

\[
\text{Worst Case Processor Utilization} = \frac{(DCT[Most Frequent Task] - WorstCaseSlackTime)}{DCT[Most Frequent Task]} - 1
\]

(15)

Worst case processor utilization is used to characterize the performance of the ACETRB, WCETRB, and SDAWS SPMMTs under resource constrained conditions.

### 5.1.1.7 – SPM Swap Time : WCET Ratio As Percentage

The SPM swap time (SPMST) to WCET ratio is the ratio of the total amount of time required to perform an SPM swap for all tasks to the total amount of execution time required to execute all tasks in the event of a worst-case task stack-up. This thesis presents the SPM swap time (SPMST) to WCET ratio as the percentage of total worst-case task stack-up WCET required to perform the SPMST (equation 16).

\[
\text{SPMST:WCET Percentage} = \frac{SPMST}{\sum_{i=0}^{n} WCET[i]}
\]

(16)

SPMST:WCET percentage is used to characterize task slack time performance and or detriment due to SDAWS.
5.1.2 – SPMMT-Independent Simulation Parameters

Each task parameter specifiable in the SPMMT simulator has a different effect on task performance. While some of these parameters affect the execution time of a simulated task under one SPMMT more than under another (i.e., SPMMT-dependent parameters), other parameters affect the execution time of tasks regardless of the SPMMT being tested (i.e., SPMMT-independent parameters). This section briefly reviews the SPMMT-independent simulation parameters. The following parameters are characterized as SPMMT-independent:

1. Memory Accesses
2. Computations Per Memory Access
3. SRAM to SDRAM Memory Access Ratio

5.1.2.1 – The Effect of Memory Accesses

This thesis uses memory accesses or accesses as a shorthand for simulated load and store memory operations. This is because the simulator performs one integer load operation and one integer store operation for each access. The parameters for this experiment shown in Table 2 (below), indicate that one arithmetic computation is performed per memory access in addition to the load and store operation.

Table 3: Experiment Parameters: Memory Accesses

<table>
<thead>
<tr>
<th>Experiment Independent Variables:</th>
<th>Memory Accesses</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Tasks:</td>
<td>1</td>
</tr>
<tr>
<td>SRAM to SDRAM Memory Access Ratio</td>
<td>200</td>
</tr>
</tbody>
</table>
5.1.2.2 – The Effect of Computations per Memory Access

In the previous experiment, computations per memory access (C per MA) was held constant at one C per MA while the number of memory accesses was varied. In the following experiment, the C to MA ratio is changed while other parameters are held constant (see Table 4 below).

<table>
<thead>
<tr>
<th># Accesses</th>
<th>ACET(uS)</th>
<th>WCET(uS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>339</td>
<td>357</td>
</tr>
<tr>
<td>1000</td>
<td>3368</td>
<td>3383</td>
</tr>
<tr>
<td>10000</td>
<td>33697</td>
<td>33713</td>
</tr>
</tbody>
</table>

Table 4 shows the effect of the number of memory accesses on ACET and WCET in microseconds. There is a crisp, linear relationship between the number of accesses and the execution time. This is due to the fact that no cache block loads are simulated for this experiment. Furthermore, there is only one execution path for this experiment (containing the specified number of accesses within it); therefore, the difference between ACET and WCET is not due to task parameters. The WCET reflects fluctuations caused by the RTOS and simulator.
**Table 5 – Computations per Memory Access (10,000 Memory Accesses)**

<table>
<thead>
<tr>
<th>Computations per MA</th>
<th>WCET</th>
<th>ACET</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>28851</td>
<td>28075</td>
</tr>
<tr>
<td>2</td>
<td>35884</td>
<td>35376</td>
</tr>
<tr>
<td>5</td>
<td>57309</td>
<td>57276</td>
</tr>
<tr>
<td>10</td>
<td>93944</td>
<td>93778</td>
</tr>
<tr>
<td>20</td>
<td>167215</td>
<td>166737</td>
</tr>
</tbody>
</table>

Figure 10 – ACET by Computations Per Memory Access (10,000 Memory Accesses)

Table 5 and figure 10 show that a small C to MA ratio does not drive execution time. This is because, for small C to MA ratios, the majority of execution time is spent performing load and store operations. As the C to MA ratio rises, the effect of computations on execution time increases. To confirm this, the experiment was performed for 1,000 accesses. This second experiment produced the same relationship between the C to MA ratio and task ACET.
5.1.2.3 – The Effect of The SRAM to SDRAM Memory Access Ratio

Because the RTOS and SPMMT simulator of this thesis are executed on a Freescale MCF5225 development board, no external SDRAM is available. Therefore, instead of using SDRAM to execute experiments, SDRAM is simulated via repeated SRAM accesses. The number of SRAM accesses equating to one SDRAM access is referred to as the SRAM to SDRAM access ratio.

Cache block loads are necessary in order to test the SRAM to SDRAM access ratio because the simulator does not perform memory accesses out of SDRAM. Rather, the simulator assumes that any accesses that do not occur out of SRAM occur out of cache. Therefore, the simulator simulates CBLs by looping through SRAM accesses according to the SRAM to SDRAM access ratio such that each CBL adds a significant amount of delay. Table 6 (below) shows the parameters of the memory access ratio experiment.

Table 6: Experiment Parameters: SRAM to SDRAM Memory Access Ratio

<table>
<thead>
<tr>
<th>Experiment Independent Variables:</th>
<th>SRAM to SDRAM Memory Access Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Tasks:</td>
<td>1</td>
</tr>
<tr>
<td>Computations Per Memory Access</td>
<td>1</td>
</tr>
<tr>
<td>Number of Memory Accesses</td>
<td>1,000</td>
</tr>
<tr>
<td>Cache Block Loads</td>
<td>2 Average-Case, 2 Worst-Case</td>
</tr>
</tbody>
</table>
In Figures 11 and 12, the time below the dark gray line represents time spent performing load, store, and arithmetic operations, while the time above the dark gray line represents time spent waiting for CBLs. As the SRAM to SDRAM access ratio increases, the amount of time waiting for CBLs increases proportionally. Because there are twice as many worst-case path
CBLs as average-case path CBLs, the WCET increases more rapidly versus the ACET as the SRAM to SDRAM access ratio increases.

5.1.3 – Experiment List and Brief Summary of Findings

This section briefly lists the general findings for each set of experiments that has a significant effect on SPMMT performance. All of the experiments are listed according to the independent variable that was modified in order to determine a corresponding performance trend. Experiments listed in boldface are discussed in this chapter. Experiments that are italicized are not discussed in this chapter — their results are shown in Appendix A. Experiments that directly affect SPMMT performance are underlined. Brief explanations are provided next to these experiments indicating why they directly affect SPMMT performance, and how. Experiments are listed in the order in which they are discussed in this chapter (or ordered in Appendix A, for those not discussed in this chapter).

- **Average and Worst-Case CBLs**
  - *Same Average and Worst-Case CBLs*
  - *Twice Worst-Case as Average-Case CBLs*
    - *Equal Non-Avertable and Avertable CBLs*
      - Note: Floating CBLs are CBLs that may be avoided depending on the SPMMT used
      - Note: Avertable CBLs are CBLs that cannot be avoided regardless of the SPMMT used
    - **Varied Non-Avertable CBLs**
- **Varied Avertable CBLs** – Avertable CBLs affect SPMMT performance because they are performed only under either the ACETRB SPMMT or the WCETRB SPMMT, but not both. The number of memory accesses into each avertable CBL determines the influence of avertable CBLs on SPMMT performance.

- **ACET:WCET Ratio** – As WCET increases, average task execution time increases, thereby limiting the ability of the ACETRB SPMMT to reduce average execution time.

- **Average-Cases Per Worst-Case** – As worst-case path frequency increases, average task execution time increases, thereby limiting the ability of the ACETRB SPMMT to reduce average execution time.

- **Worst-Case Processor Utilization** – As worst-case processor utilization increases, slack time gains achieved through the use of SDAWS or through the WCETRB SPMMT increase. The slack time requirements specified by developers determine whether the SDAWS or WCETRB SPMMTs should be used instead of the ACETRB SPMMT.
  - **Varied Task Frequencies**
  - **Varied Task Execution Times**
  - **Varied Number of Tasks**
  - **Aperiodic Disturbance**
    - **Varied Distortion and Task Frequency**
    - **Varied Distortion Size**
• **Varied Task Frequency (RTOS Overhead Experiment)** – Task frequency affects the number of computations that must be performed by the RTOS. Because the SDAWS SPMMT requires more RTOS computations per task instance than the ACETRB or WCETRB SPMMTs, task frequency directly affects the performance of the SDAWS SPMMT.

• **Varied SPM Swap Times** – SPM swap time affects the ability of SDAWS to achieve increased slack time performance versus the ACETRB SPMMT. As SPM swap time increases, slack time gains due to SDAWS are reduced. It is possible for slack times under SDAWS to be smaller than slack times under the ACETRB SPMMT if SPM swap time is too large. When this is the case, SDAWS should not be used.

• **Desired Deadline Threshold** – As the desired deadline threshold is increased (toward 100% of a task’s hard deadline), the number of SPM swaps performed by SDAWS decreases. Fewer SPM swaps result in less RTOS overhead and less swap-back time, (i.e., time required to swap back to the ACETRB SPMMT once the worst-case stack-up has completed). It is possible for swap-back time to result in reduced SDAWS slack time performance for large SPM swap times, if the swap-back intersects with the next worst-case task stack-up, (which must wait for the swap-back to complete before executing).

5.2 – COMPARISON OF THE ACETRB AND WCETRB SPMMTS

The comparison of the ACETRB and WCETRB SPMMTs is divided into three sections.

• Section 1 – Analysis of Non-Avertable and Avertable Cache Block Loads
5.2.1 – Analysis of Non-Avertable and Avertable Cache Block Loads

ACET and WCET performance is affected by cache block load time. The amount of time required to perform each CBL and the number of CBLs to perform both affect execution time. These are the only two factors when no SPM is used; however, the use of SPM causes CBLs to be averted.

Under the ACETRB SPMMT, CBLs are averted along the average-case execution path. Under the WCETRB SPMMT, CBLs are averted along the worst-case execution path. However, it is not always possible to avert all CBLs along each path, depending on the size of SPM relative to the total size of variables along each path. CBLs that can be averted depending on the SPMMT loaded are referred to by this thesis as avertable CBLs. CBLs that cannot be averted regardless of the SPMMT loaded are referred to by this thesis as non-avertable CBLs.

5.2.1.1 – The Effect of Non-Avertable Cache Block Loads (CBLs)

Tables 7, 8, and 9 show the parameters and results of an experiment of the effect of a relative increase in non-avertable CBLs on ACET/WCET under each of the ACETRB and WCETRB SPMMTs.
### Table 7: Experiment Parameters: Non-Avertable CBLs

<table>
<thead>
<tr>
<th>Experiment Independent Variables</th>
<th>Non-Avertable CBLs</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Number of Tasks:</strong></td>
<td>1</td>
</tr>
<tr>
<td><strong>WCET:ACET Ratio:</strong></td>
<td>2:1</td>
</tr>
<tr>
<td><strong>Average-Case Avertable CBLs:</strong></td>
<td>2</td>
</tr>
<tr>
<td><strong>Average-Case Non-Avertable CBLs:</strong></td>
<td>Variable</td>
</tr>
<tr>
<td><strong>Worst-Case Avertable CBLs:</strong></td>
<td>4</td>
</tr>
<tr>
<td><strong>Worst-Case Non-Avertable CBLs:</strong></td>
<td>Variable</td>
</tr>
</tbody>
</table>

### Table 8 – ACETRB SPMMT (2 Avertable CBLs for Average-Case, 4 Avertable CBLs for Worst-Case) with Average-Case Accesses: Worst-Case Accesses = 1000:2000

<table>
<thead>
<tr>
<th>SPMMT</th>
<th>AC to WC Static CBLs</th>
<th>Average-Case Path Execution Time</th>
<th>Worst-Case Path Execution Time</th>
<th>Average/Worst-Case Path Execution Time Difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACETRB</td>
<td>2 to 4</td>
<td>5661</td>
<td>9081</td>
<td>37.66%</td>
</tr>
<tr>
<td>ACETRB</td>
<td>8 to 16</td>
<td>11580</td>
<td>22892</td>
<td>49.41%</td>
</tr>
<tr>
<td>ACETRB</td>
<td>32 to 64</td>
<td>18832</td>
<td>38576</td>
<td>51.31%</td>
</tr>
</tbody>
</table>

### Table 9 – WCETRB SPMMT (2 Avertable CBLs for Average-Case, 4 Avertable CBLs for Worst-Case) with Average-Case Accesses: Worst-Case Accesses = 1000:2000

<table>
<thead>
<tr>
<th>SPMMT</th>
<th>AC to WC Static CBLs</th>
<th>Average-Case Path Execution Time</th>
<th>Worst-Case Path Execution Time</th>
<th>Average/Worst-Case Path Execution Time Difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>WCETRB</td>
<td>2 to 4</td>
<td>6394</td>
<td>8300</td>
<td>22.96%</td>
</tr>
<tr>
<td>WCETRB</td>
<td>8 to 16</td>
<td>12313</td>
<td>22111</td>
<td>44.31%</td>
</tr>
<tr>
<td>WCETRB</td>
<td>32 to 64</td>
<td>19566</td>
<td>37955</td>
<td>48.37%</td>
</tr>
</tbody>
</table>
Figure 13 – ACET and WCET Reduction for Corresponding SPMMTs (average/worst-case non-avertable CBLs)

Non-Avertable cache blocks are loaded regardless of the SPMMT used. As the number of non-avertable CBLs increases, ACET reduction and WCET reduction decreases. This is due to the fact the relative effect of the avertable CBLs on overall execution time is reduced as the number of non-avertable CBLs increases. Figure 13 (above) shows this effect clearly.

5.2.1.2 – The Effect of Avertable Cache Block Loads (CBLs)

Avertable cache blocks are loaded only when the SPMMT that keeps the associated cache block memory elements in SPM is not applied. Tables 10, 11, and 12 show the parameters and results of an experiment testing the effect of a relative increase in avertable CBLs on ACET and WCET under the ACETRB and WCETRB SPMMTs.
Table 10: Experiment Parameters: Avertable CBLs

<table>
<thead>
<tr>
<th>Experiment Independent Variables:</th>
<th>Avertable CBLs</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Number of Tasks:</strong></td>
<td>1</td>
</tr>
<tr>
<td><strong>WCET:ACET Ratio</strong></td>
<td>2:1</td>
</tr>
<tr>
<td><strong>Average-Case Avertable CBLs:</strong></td>
<td>Variable</td>
</tr>
<tr>
<td><strong>Average-Case Non-Avertable CBLs:</strong></td>
<td>2</td>
</tr>
<tr>
<td><strong>Worst-Case Avertable CBLs:</strong></td>
<td>Variable</td>
</tr>
<tr>
<td><strong>Worst-Case Non-Avertable CBLs:</strong></td>
<td>4</td>
</tr>
</tbody>
</table>

Table 11 – ACETRB SPMMT (2 Non-Avertable CBLs for Average-Case, 4 Non-Avertable CBLs for Worst-Case) with Average-Case Accesses: Worst-Case Accesses = 1000:2000

<table>
<thead>
<tr>
<th>SPMMT</th>
<th>AC to WC Floating CBLs</th>
<th>Average-Case Path Execution Time</th>
<th>Worst-Case Path Execution Time</th>
<th>Average/Worst-Case Path Difference</th>
<th>ACETRB % ACET DECREASE VERSUS WCETRB</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACETRB</td>
<td>2 to 4</td>
<td>5016</td>
<td>9765</td>
<td>48.63%</td>
<td>15.00%</td>
</tr>
<tr>
<td>ACETRB</td>
<td>8 to 16</td>
<td>5016</td>
<td>13871</td>
<td>63.84%</td>
<td>40.16%</td>
</tr>
<tr>
<td>ACETRB</td>
<td>32 to 64</td>
<td>5016</td>
<td>30296</td>
<td>83.44%</td>
<td>72.60%</td>
</tr>
</tbody>
</table>

Table 12 – WCETRB SPMMT (2 Non-Avertable CBLs for Average-Case, 4 Non-Avertable CBLs for Worst-Case) with Average-Case Accesses: Worst-Case Accesses = 1000:2000

<table>
<thead>
<tr>
<th>SPMMT</th>
<th>AC to WC Floating CBLs</th>
<th>Average-Case Path Execution Time</th>
<th>Worst-Case Path Execution Time</th>
<th>Average/Worst-Case Path Difference</th>
<th>WCETRB % WCET DECREASE VERSUS ACETRB</th>
</tr>
</thead>
<tbody>
<tr>
<td>WCETRB</td>
<td>2 to 4</td>
<td>5750</td>
<td>8300</td>
<td>30.72%</td>
<td>12.77%</td>
</tr>
<tr>
<td>WCETRB</td>
<td>8 to 16</td>
<td>7803</td>
<td>8300</td>
<td>5.99%</td>
<td>35.72%</td>
</tr>
<tr>
<td>WCETRB</td>
<td>32 to 64</td>
<td>16015</td>
<td>8300</td>
<td>-92.95%</td>
<td>68.88%</td>
</tr>
</tbody>
</table>
Figure 14 – ACET and WCET Decrease for Corresponding SPMMTs (via average/worst-case avertable CBLs)

Figure 14 demonstrates the execution time reduction that is achieved for each of the tested avertable CBL sets. The light-gray bars represent ACET reduction under the ACETRB SPMMT, and the dark gray bars represent WCET reduction under the WCETRB SPMMT. The larger the percentage of reduction, the greater effect the SPMMT has on reducing its corresponding execution path. Minor differences between ACET and WCET speedup result from RTOS overhead and additional non-avertable cache block loads that cause the total WCET to ACET ratio to be less than 2 to 1, while average-case to worst-case avertable CBLs maintain a 1 to 2 ratio.

This experiment maintains the same number of accesses per CBL for both WCETRB and ACETRB SPMMT tests. However, in a real system it is possible that this would not be the case, because SPM size is fixed. Therefore it is possible that the same number of blocks would need to be allocated into SPM for the WCETRB and the ACETRB SPMMTs. Therefore, results such as
these, where ACET and WCET reduction (percentage wise) is approximately the same, reflect cases in which the ACETRB SPMMT can utilize only half of SPM effectively (due to having allocated all of task parameters into SPM, to increased numbers of small tasks to allocate into SPM, to ineffectiveness of placing certain parameters into SPM, etc.).

Figure 15 – ACET (average/worst-case avertable CBLs)

Figure 16 – WCET (average/worst-case avertable CBLs)
Figures 15 and 16 show that execution time remains constant as the number of avertable CBLs is increased for the execution path corresponding to the SPMMT being utilized. Avertable cache blocks are loaded only when the opposite SPMMT is utilized. That is, avertable average-case cache blocks are loaded only under the WCETRB SPMMT, and avertable worst-case cache blocks are loaded only under the ACETRB SPMMT. Therefore, as the number of avertable CBLs increases, the WCET and ACET reduction for the corresponding SPMMT type increases drastically. Note that for the 32:64 avertable CBL test, the ACET for the WCETRB SPMMT is greater than the WCET for the WCETRB SPMMT. This represents a scenario in which further WCET reduction is possible through allocation of average-case variables into SPM for the WCETRB SPMMT (because the WCET occurs during the average-case path execution).

5.2.2 – Analysis of ACETRB Average Execution Time Reduction

The previous section analyzed factors that affect WCET versus ACET reduction. While it is valuable to know the amount of reduction that can be achieved for any single task instance, often it is also valuable to know the average execution time for each task. Average execution time is the cumulative sum of all task instance execution time divided by the number of task instances. If the goal of SPMMT optimization is not to reduce ACET as much as possible, but instead to reduce average execution time, one must consider the effect of worst-case execution frequency and WCET on average execution time. The worst-case execution frequency and the WCET for a task affect its average execution time, because they cause the average execution time for a task to be offset. In some cases, it is even possible that the ACETRB SPMMT could provide little to no average execution time reduction versus the WCETRB SPMMT if the WCET is very large (or worst-case execution very frequent) compared to the ACET.
5.2.2.1 – The Effect of The WCET to ACET Ratio on Average Execution Time

Table 13 shows the parameters of an experiment measuring the effect of increased WCET on average execution time.

Table 13: Experiment Parameters: WCET:ACET Ratio

<table>
<thead>
<tr>
<th>Experiment Independent Variables:</th>
<th>WCET:ACET Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Tasks:</td>
<td>2</td>
</tr>
<tr>
<td>Average-Case Per Worst-Case:</td>
<td>9</td>
</tr>
<tr>
<td>Memory Accesses Per CBL:</td>
<td>400</td>
</tr>
<tr>
<td>Average-Case Avertable CBLs:</td>
<td>3</td>
</tr>
<tr>
<td>Average-Case Non-Avertable CBLs:</td>
<td>0</td>
</tr>
<tr>
<td>Worst-Case Avertable CBLs:</td>
<td>3</td>
</tr>
<tr>
<td>Worst-Case Non-Avertable CBLs:</td>
<td>3</td>
</tr>
</tbody>
</table>

Figure 17: Average Task Execution Time According to WCET to ACET Ratio

Figure 17: Average Task Execution Time According to WCET to ACET Ratio
Figure 18: Percent Average Execution Time for the ACETRB SPMMT over the WCETRB SPMMT for Different WCET to ACET Ratios

Figure shows that as WCET doubles, percentage average execution time decrease by approximately 2%. Therefore, the ACETRB SPMMT provides diminishing returns as WCET increases. However, this experiment assumes that sufficient SPM is available to maintain the worst-case access/CBL ratio as WCET increases. As soon as the worst-case execution path exceeds the allocation capabilities of SPM, the amount of WCET reduction that can be achieved through WCETRB SPMM drops off, and average execution time reduction from ACETRB SPMM flat-lines.

5.2.2.2 – The Effect of Average Cases per Worst Case on Average Execution Time

Table 14 shows the parameters of an experiment measuring the effect of increased worst-case execution path frequency on average execution time.
Table 14: Experiment Parameters: Average-Cases Per Worst-Case

<table>
<thead>
<tr>
<th>Experiment Independent Variables:</th>
<th>Average-Cases Per Worst-Case</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Tasks:</td>
<td>2</td>
</tr>
<tr>
<td>WCET:ACET Ratio:</td>
<td>2:1</td>
</tr>
<tr>
<td>Memory Accesses Per CBL:</td>
<td>400</td>
</tr>
<tr>
<td>Average-Case Avertable CBLs:</td>
<td>3</td>
</tr>
<tr>
<td>Average-Case Non-Avertable CBLs:</td>
<td>0</td>
</tr>
<tr>
<td>Worst-Case Avertable CBLs:</td>
<td>3</td>
</tr>
<tr>
<td>Worst-Case Non-Avertable CBLs:</td>
<td>3</td>
</tr>
</tbody>
</table>

Figure 19: Average Task Execution Time According to Average Cases per Worst-Case
Figure 20: Percent Average Execution Time for the ACETRB SPMMT over the WCETRB SPMMT for Different Average Cases per Worst-Case

Figures 19 and 20 shows that as worst-case execution frequency increases, percentage average execution time decrease from the ACETRB SPMMT shrinks significantly. As shown before, the ACETRB SPMMT provides diminishing returns. In Figure 20, when the worst-case execution path occurs at the same frequency as the average-case execution path, the WCETRB SPMMT demonstrates reduced average execution time versus the ACETRB SPMMT. However, in a real situation this is unlikely to be possible. If the ACETRB SPMMT uses profiling as its source for SPMA, it will possess in its allocation all of the variables that are accessed most frequently (even those contained within the worst-case execution path). Therefore, though the ACETRB SPMMT provides diminishing returns as worst-case path frequency increases, it will not provide reduced average-case execution time versus the WCETRB SPMMT.
5.2.3 – Analysis of Worst-Case Processor Utilization

WCETRB SPMM would not be meaningful if it were not for the necessity of deadline assurance. Under resource-unconstrained conditions such as have already been demonstrated, task slack time is not necessary to measure as tasks are ensured the ability to meet their deadlines. There are several kinds of resource constraints, including the total amount of SPM, cache, and processor execution time. The effect of cache block loads and SPM size have already been demonstrated. While these factors affect ACET and WCET execution times, they do not directly affect task slack time, and their effect on task slack time depends on the SPMMT employed.

Processor utilization can refer to many things, but generally refers to the percentage of time that the processor spends executing tasks and other functions over a time period. Because the experiments performed by this thesis include average-case and worst-case task runtime combinations within each individual test, it is important to clarify that the processor utilization referred to by this thesis does not correspond to the percentage of time that the processor spends executing tasks and other functions over the entire duration of an experiment. Rather, the processor utilization referred to by this thesis refers to the worst-case processor utilization, which is the percentage of time that the processor spends executing tasks/RTOS functions over a specified task repetition period in which all tasks running on the processor are executing according to their WCET.

In this way, average processor utilization may be only 50% or less, while the worst-case processor utilization is 95% or greater. The results shown in this chapter that specify processor utilization calculate processor utilization as though no SPMMT would be utilized. Therefore, the WCETRB SPMMT achieves less than the specified processor utilization percentage.
Furthermore, the worst-case processor utilization specified is an approximation based upon recorded slack times and WCETs, and varies by ±1% for each test. This section does not compare slack time reduction for a large set of WCET to ACET ratios, avertable CBL ratios, or other such factors. This is due to two factors: 1) because it is desired keep the analysis of processor utilization on slack time clear. 2) because it is outside of the scope of this thesis to test all possible real-time/resource-constraint scenarios.

5.2.3.1 – The Effect of Task Repetition Periods/Task Execution Times

5.2.3.1.1 – Modified Repetition Periods

Tables 15, 16, and 17 show the parameters and results effect of increased task repetition periods (while task execution time remain the same) on slack time reduction via the WCETRB SPMMT.

Table 15: Experiment Parameters: Worst-Case Processor Utilization Via Task WCETs

<table>
<thead>
<tr>
<th>Experiment Independent Variables:</th>
<th>Worst-Case Processor Utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Tasks:</td>
<td>2</td>
</tr>
<tr>
<td>WCET:ACET Ratio:</td>
<td>2:1</td>
</tr>
<tr>
<td>Memory Accesses Per CBL:</td>
<td>400</td>
</tr>
<tr>
<td>Average-Case Avertable CBLs:</td>
<td>3</td>
</tr>
<tr>
<td>Average-Case Non-Avertable CBLs:</td>
<td>0</td>
</tr>
<tr>
<td>Worst-Case Avertable CBLs:</td>
<td>3</td>
</tr>
</tbody>
</table>
Worst-Case Non-Avertable CBLs:

Table 16: ACETRB SPMMT – Worst-Case Processor Utilization via Varied Task Repetition Periods

<table>
<thead>
<tr>
<th>WC Processor Utilization %</th>
<th>Repetition Period</th>
<th>To Worst-Case Slack-Time</th>
<th>To Average Slack-Time</th>
<th>To Worst-Case Exe-Time</th>
<th>To Average Exe-Time</th>
<th>TL Worst-Case Slack-Time</th>
<th>TL Average Slack-Time</th>
<th>% Acet Decrease Versus WCETRB</th>
<th>% Aet Decrease Versus WCETRB</th>
<th>Idle Time:</th>
</tr>
</thead>
<tbody>
<tr>
<td>25%</td>
<td>100000</td>
<td>7558</td>
<td>91589</td>
<td>12132</td>
<td>6011</td>
<td>5386</td>
<td>7558</td>
<td>90147</td>
<td>7.14%</td>
<td>4.31%</td>
</tr>
<tr>
<td>50%</td>
<td>50000</td>
<td>25580</td>
<td>31913</td>
<td>12133</td>
<td>6042</td>
<td>5399</td>
<td>25592</td>
<td>29726</td>
<td>7.14%</td>
<td>4.17%</td>
</tr>
<tr>
<td>75%</td>
<td>12000</td>
<td>8574</td>
<td>21881</td>
<td>12133</td>
<td>6066</td>
<td>5399</td>
<td>8802</td>
<td>23896</td>
<td>7.14%</td>
<td>4.09%</td>
</tr>
<tr>
<td>90%</td>
<td>27777</td>
<td>3356</td>
<td>18050</td>
<td>12133</td>
<td>6072</td>
<td>5399</td>
<td>1385</td>
<td>19057</td>
<td>7.14%</td>
<td>4.06%</td>
</tr>
</tbody>
</table>

Table 17: WCETRB SPMMT – Worst-Case Processor Utilization via Varied Task Repetition Periods

The results in Tables 16 and 17 are graphed in Figure 20.

5.2.3.1.2 – Modified Task Execution Times via Task WCETs.

Tables 18, 19, and 20 show the parameters and results of an experiment measuring the effect of increased task WCET on slack time reduction via the WCETRB SPMMT.

Table 18: Experiment Parameters: Worst-Case Processor Utilization Via Task WCETs

<table>
<thead>
<tr>
<th>Experiment Independent Variables:</th>
<th>Worst-Case Processor Utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Tasks:</td>
<td>2</td>
</tr>
</tbody>
</table>
WCET:ACET Ratio: 2:1

Memory Accesses Per CBL: 400

Worst-Case: Average-Case CBLs 2:1

Table 19: ACETRB SPMMT – Worst-Case Processor Utilization via Varied Task Execution Times

<table>
<thead>
<tr>
<th>Processor Utilization %</th>
<th>To Worst-Case Slack-Time</th>
<th>To Average Slack-Time</th>
<th>To Worst-Case Exe-Time</th>
<th>To Average Exe-Time</th>
<th>T1 Worst-Case Slack-Time</th>
<th>T1 Average Slack-Time</th>
<th>% ACET Decrease Versus WCETRB</th>
<th>% AET Decrease Versus WCETRB</th>
<th>Idle Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>25%</td>
<td>75578</td>
<td>91598</td>
<td>12132</td>
<td>5891</td>
<td>5279</td>
<td>75589</td>
<td>90147</td>
<td>9.26%</td>
<td>6.22%</td>
</tr>
<tr>
<td>50%</td>
<td>74982</td>
<td>82015</td>
<td>24906</td>
<td>12656</td>
<td>11432</td>
<td>79777</td>
<td>9.31%</td>
<td>6.33%</td>
<td>7261690</td>
</tr>
<tr>
<td>75%</td>
<td>25157</td>
<td>75892</td>
<td>17344</td>
<td>19085</td>
<td>17277</td>
<td>251368</td>
<td>67124</td>
<td>9.64%</td>
<td>6.54%</td>
</tr>
<tr>
<td>90%</td>
<td>54999</td>
<td>60037</td>
<td>44943</td>
<td>22949</td>
<td>20750</td>
<td>9855</td>
<td>10.05%</td>
<td>6.96%</td>
<td>6047685</td>
</tr>
</tbody>
</table>

Table 20: WCETRB SPMMT – Worst-Case Processor Utilization via Varied Task Execution Times

<table>
<thead>
<tr>
<th>WC Processor Utilization %</th>
<th>To Worst-Case Slack-Time</th>
<th>To Average Slack-Time</th>
<th>To Worst-Case Exe-Time</th>
<th>To Average Exe-Time</th>
<th>T1 Worst-Case Slack-Time</th>
<th>T1 Average Slack-Time</th>
<th>WCETRB % Slack Time Increase Versus WCETRB</th>
<th>% WCET Decrease Versus ACETRB</th>
<th>Idle Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>22%</td>
<td>77320</td>
<td>91334</td>
<td>10962</td>
<td>6282</td>
<td>5814</td>
<td>77335</td>
<td>85590</td>
<td>5.10%</td>
<td>9.64%</td>
</tr>
<tr>
<td>45%</td>
<td>54721</td>
<td>80738</td>
<td>22558</td>
<td>13511</td>
<td>12606</td>
<td>54737</td>
<td>78494</td>
<td>9.30%</td>
<td>9.43%</td>
</tr>
<tr>
<td>68%</td>
<td>32195</td>
<td>72975</td>
<td>33825</td>
<td>20399</td>
<td>19077</td>
<td>32203</td>
<td>6589</td>
<td>27.96%</td>
<td>9.42%</td>
</tr>
<tr>
<td>81%</td>
<td>53703</td>
<td>70559</td>
<td>40653</td>
<td>24866</td>
<td>23088</td>
<td>18547</td>
<td>55205</td>
<td>88.31%</td>
<td>9.55%</td>
</tr>
</tbody>
</table>

The results in Tables 19 and 20 are graphed in Figure 20.
Figure 20: Percent Slack Time Increase for the WCETRB SPMMT over the ACETRB SPMMT for Different Worst-Case Processor Utilizations (via Contrasted Rep Periods and Task Runtimes)

Figure 20 shows that as worst-case processor utilization increases, slack time percentage improvement increases exponentially. This is because, as WCETs cause tasks to come closer and closer to their deadlines, the slack time that is gained via the WCETRB SPMMT becomes larger relative to the original slack time (under ACETRB SPMM). Actual slack time decreases exponentially as worst-case processor utilization approaches 100%, so the WCETRB SPMMT percentage increase in slack time behaves according to the inverse of actual slack time.

The primary difference between varied repetition periods and varied task runtimes as they affect worst-case processor utilization is that by varying repetition periods, RTOS overhead is increased substantially. Because worst-case processor utilization is determined using task WCETs and repetition periods, RTOS overhead is not included. However, slack time
calculations include RTOS overhead, because any overhead that has occurred before the slack
time is calculated causes slack time to decrease. Therefore, while varied repetition periods and
task runtimes have very similar slack time characteristics for tested worst-case processor
utilizations less than 90%, there is a substantial difference between slack times at 90% worst-
case processor utilization. The difference is due almost entirely to RTOS overhead, though the
gap between percentage slack time increase is widened by the fact that the RTOS must perform
extensive timing analysis after each task execution.

5.2.3.2 – The Effect of Worst-Case Task Stack-up

Tables 21, 22, and 23 show the parameters and results of an experiment measuring the
effect of the number of tasks being run on task slack time reduction under the WCETRB
SPMMT.

Table 21: Experiment Parameters: Worst-Case Processor Utilization Via Varied Numbers of
Tasks

<table>
<thead>
<tr>
<th>Experiment Independent Variables:</th>
<th>Worst-Case Processor Utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Tasks:</td>
<td>Variable Number (2 to 8)</td>
</tr>
<tr>
<td>WCET:ACET Ratio:</td>
<td>2:1</td>
</tr>
<tr>
<td>Memory Accesses Per CBL:</td>
<td>400</td>
</tr>
<tr>
<td>Average-Case Avertable CBLs:</td>
<td>3</td>
</tr>
<tr>
<td>Average-Case Non-Avertable CBLs:</td>
<td>0</td>
</tr>
<tr>
<td>Worst-Case Avertable CBLs:</td>
<td>3</td>
</tr>
<tr>
<td>Worst-Case Non-Avertable CBLs:</td>
<td>3</td>
</tr>
</tbody>
</table>
Table 22: ACETRB SPMMT – Worst-Case Processor Utilization via Number of Tasks

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ACETRB</td>
<td>2 (24.4% WC PU)</td>
<td>100000</td>
<td>695695</td>
<td>75573</td>
<td>90898</td>
<td>12132</td>
<td>5718</td>
<td>5077</td>
<td>12132</td>
</tr>
<tr>
<td>ACETRB</td>
<td>3 (36.1% WC PU)</td>
<td>100000</td>
<td>7266083</td>
<td>63964</td>
<td>86403</td>
<td>12132</td>
<td>6598</td>
<td>5077</td>
<td>12132</td>
</tr>
<tr>
<td>ACETRB</td>
<td>4 (48.3% WC PU)</td>
<td>100000</td>
<td>6457543</td>
<td>51151</td>
<td>88760</td>
<td>12133</td>
<td>6586</td>
<td>5077</td>
<td>12131</td>
</tr>
<tr>
<td>ACETRB</td>
<td>6 (73.1% WC PU)</td>
<td>100000</td>
<td>4793438</td>
<td>26698</td>
<td>70127</td>
<td>12133</td>
<td>8588</td>
<td>5077</td>
<td>12131</td>
</tr>
<tr>
<td>ACETRB</td>
<td>8 (97.5% WC PU)</td>
<td>100000</td>
<td>3109037</td>
<td>2161</td>
<td>69740</td>
<td>12133</td>
<td>8588</td>
<td>5077</td>
<td>12131</td>
</tr>
</tbody>
</table>

Table 23: WCETRB SPMMT – Worst-Case Processor Utilization via Number of Tasks

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>WCETRB</td>
<td>2 (24.4% WC PU)</td>
<td>100000</td>
<td>8424317</td>
<td>3.10%</td>
<td>77912</td>
<td>89896</td>
<td>10963</td>
<td>6611</td>
<td>6176</td>
<td>10961</td>
</tr>
<tr>
<td>WCETRB</td>
<td>3 (36.1% WC PU)</td>
<td>100000</td>
<td>7274182</td>
<td>5.53%</td>
<td>66868</td>
<td>81731</td>
<td>10864</td>
<td>8544</td>
<td>6376</td>
<td>10961</td>
</tr>
<tr>
<td>WCETRB</td>
<td>4 (48.3% WC PU)</td>
<td>100000</td>
<td>6963228</td>
<td>9.12%</td>
<td>59816</td>
<td>72022</td>
<td>10692</td>
<td>8544</td>
<td>6126</td>
<td>10961</td>
</tr>
<tr>
<td>WCETRB</td>
<td>6 (73.1% WC PU)</td>
<td>100000</td>
<td>4064586</td>
<td>26.25%</td>
<td>37706</td>
<td>71815</td>
<td>10692</td>
<td>8544</td>
<td>6176</td>
<td>10961</td>
</tr>
<tr>
<td>WCETRB</td>
<td>8 (97.5% WC PU)</td>
<td>100000</td>
<td>3126137</td>
<td>414.24%</td>
<td>11627</td>
<td>89276</td>
<td>10962</td>
<td>8544</td>
<td>6276</td>
<td>10961</td>
</tr>
</tbody>
</table>

Figure 21: Worst-Case Slack Time for Different Numbers of Identical Tasks

Figure 21 shows the actual worst-case slack times under the ACETRB SPMMT (in light-gray) and the WCETRB SPMMT (in dark gray). Two trends are notable on this graph. First, the
slack time difference between ACETRB and WCETRB SPMMTs grows as processor utilization increases. This is because the WCETRB provides a fixed amount of reduction for each task. As the number of tasks increases, so too does the total WCETRB SPMMT slack time reduction.

Second, the ACETRB slack time to WCETRB slack time ratio increases exponentially as worst-case processor utilization approaches 100%. This is because as ACETRB slack time approaches zero, the percent increase in slack time due to the WCETRB SPMMT approaches infinity (1/0).

Figure uuu (below) shows this effect very clearly.

Figure 22: Percent Slack Time Increase for the WCETRB SPMMT over the ACETRB SPMMT for Different Worst-Case Processor Utilizations (via Numbers of Identical Tasks)
5.3 – THE EFFECT OF OVERHEAD ON SDAWS PERFORMANCE

This section tests the features and parameters unique to the SDAWS SPMMT. This section does not go into great detail regarding the kinds of ACET reductions that can be achieved through the use of SDAWS, as they depend heavily upon the kind of system being run (and have already been discussed (in Section 5.2). The notable difference between SDAWS and its ACETRB and WCETRB counterparts rests in slack time performance under resource-constrained conditions. Therefore, this section discusses the effects of SDAWS-specific factors on task slack times – including when SDAWS makes them bigger and when SDAWS makes them smaller versus the ACETRB SPMMT. The analysis of the results SDAWS SPMMT testing is divided into four sections.

- Section 1 – SDAWS Results Summary
- Section 2 – Comparison of RTOS Overhead for Different SPMMTs
- Section 3 – Analysis of the SPM Swap Time (SPMST) to Task WCET Ratio.
- Section 4 – Analysis of Desired Deadlines on SDAWS Performance

In all of the experiments run for this section, the number of computations per memory access (C to MA) and the memory access ratio (MAR) are fixed.

- C to MA = 2
- MAR = 200 SRAM accesses per SDRAM access

5.3.1 – SDAWS Results Summary

Because the purpose of the SDAWS SPMMT is to reduce ACET versus the WCETRB SPMMT, and to increase slack time versus the ACETRB SPMMT, SDAWS should be used only
when both of those criteria are met. However, regarding ACET reduction, SDAWS will at worst achieve the same ACET as the WCETRB SPMMT. Therefore, the criterion that determines whether SDAWS will provide any benefit whatsoever is this: worst-case task slack times under SDAWS must be greater than worst-case task slack times under ACETRB SPMM.

In order for worst-case task slack times to be greater under SDAWS than under ACETRB SPMM, worst-case task stack-up scenarios (i.e., those that cause worst-case slack times) must be examined. Specifically, the sum of all task WCETs and overhead that may occur during a worst-case task stack-up must be compared under SDAWS and under ACETRB SPMM. However, RTOS overhead is the same for both SDAWS and ACETRB SPMM except for the addition of point-schedulability analyses and SPM swaps required for SDAWS. Therefore, only these two overhead factors need to be considered.

5.3.1.1 – When SDAWS Should Not Be Used

The following inequality expresses when SDAWS should not be used.

\[
PSAT + SPMST > \sum_{K=0}^{N} (WCET(ACETRB\text{SPMMT}, t_k)) - \sum_{K=0}^{N} (WCET(WCETRB\text{SPMMT}, t_k))
\]  

(17)

In this equation, \(N\) is the number of tasks in the system that can occur during a worst-case task stack-up. PSAT = Point-Schedulability Analysis Time; SPMST = SPM Swap Time; \(t_k\) = task \(k\), and WCET(\text{SPMMT, } t_k)\) is the worst-case execution time of \(t_k\) under the corresponding SPMMT.

(17) shows that the SDAWS SPMMT should not be used when WCET reductions achieved through swapping to WCETRB mode are smaller than the amount of additional overhead.
required to test for and perform the SPM swap, (i.e., point-schedulability analysis time and SPM swap time). If slack time increases versus the ACETRB SPMMT cannot be achieved, then ACETRB SPMMT would provide better performance than SDAWS.

### 5.3.1.2 – When SDAWS Provides Slack Time Benefit Over The ACETRB SPMMT

The inequality below describes the when SDAWS provides slack-time benefit over ACETRB SPMMT.

\[
\text{PSAT} + \text{SPMST} < \sum_{k=0}^{N} (\text{WCET(ACETRB}_{\text{SPMMT}}, t_k)) - \sum_{k=0}^{N} (\text{WCET(ACETRB}_{\text{SPMMT}}, t_k))
\]

(18)

In this expression, \( N \) is the number of tasks in the system that can occur during a worst-case task stack-up. \( \text{PSAT} = \text{Point-Schedulability Analysis Time}; \ \text{SPMST} = \text{SPM Swap Time}; t_k = \text{task k}, \) and \( \text{WCET(ACETRB}_{\text{SPMMT}}, t_k) \) is the worst-case execution time of \( t_k \) under the corresponding SPMMT.

(18) shows that the SDAWS SPMMT provides benefit over the ACETRB SPMMT when WCET reductions achieved through swapping to WCETRB mode are greater than the amount of additional overhead required to test for and perform the SPM swap. However, just because slack times are larger does not mean that SDAWS ought to be used. The WCETRB SPMMT provides the best slack time results because it requires no additional overhead.

### 5.3.1.3 – ACET Reduction and SDAWS Overhead

For a system that does not encounter a worst-case task stack-up often, ACET reductions will be very similar to the ACETRB SPMMT. That is, ACET reductions could be as high as 80%
This is because SDAWS operates in ACETRB mode until a worst-case task stack-up is encountered. Though additional overhead is required to perform SPM swaps and point-schedulability analyses (see 5.3.2), additional overhead is reduced when task frequency and the frequency of SPM swaps are reduced.

SPM swaps are performed only when transitions between SPM modes are required. Thus, so long as worst-case task stack-ups are infrequent, SPM swaps are infrequent. If task periods are large, overhead is minimized, due to a decreased number of schedulability analyses. If task periods are small, overhead is still significant due to increased schedulability analysis frequency.

Therefore, though ACET reductions of up to 80% over WCETRB SPMM can be achieved, ACET reductions could be eclipsed by additional RTOS overhead for systems with many very small tasks that have very small periods. Of course, task and period sizes are relative to the processor on which tasks are running. Therefore, no precise measurements can be given. ACETRB SPMMT testing must be performed on the system on which SDAWS is to be run before possible SDAWS gains can be measured.

5.3.1.4 – When SDAWS Should Be Used

Clearly, in order for SDAWS to be useful, the condition specified in (18) must be met. However, in order for SDAWS to be worthwhile, ACET reduction under SDAWS must be desirably greater than ACET reduction under WCETRB SPMM. Therefore, SDAWS should be used only when all of the following criteria are met:

1. Schedulability assurance under ACETRB SPMM is insufficient;
2. Schedulability assurance under SDAWS is sufficient; and
3. The SDAWS ACET reduction over WCETRB SPMM is desirable.
5.3.2 – Comparison of RTOS Overhead for Different SPMMTs

It is very important to acknowledge the effect of RTOS overhead in order to ensure that experiments performed by the SPMMT simulator reflect valid real-time scenarios running on a realistic RTOS. Therefore, in this section, some of the results of experiments performed in order to compare the WCETRB, ACETRB, and SDAWS SPMMTs are compared for the purpose of demonstrating RTOS overhead. RTOS overhead is affected by task periodicity and the SPMMT employed by the RTOS. The items included in the measured RTOS overhead include kernel operation, automated time statistic capture, automated point-schedulability analysis, and SPM swapping. The latter two are factors only under the SDAWS SPMMT. Table 24 shows the parameter set while testing RTOS overhead.

Table 24: Experiment Parameters: Aperiodic Disturbance Test

<table>
<thead>
<tr>
<th>Experiment Independent Variables:</th>
<th>SPMMT, Task Periodicity</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Number of Tasks:</strong></td>
<td>3 (2 small, low priority, high frequency; 1 large, high priority, low frequency)</td>
</tr>
<tr>
<td><strong>WCET:ACET Ratio:</strong></td>
<td>2:1</td>
</tr>
<tr>
<td><strong>Memory Accesses Per CBL:</strong></td>
<td>400</td>
</tr>
<tr>
<td><strong>Average-Case Avertable CBLs:</strong></td>
<td>3</td>
</tr>
<tr>
<td><strong>Average-Case Non-Avertable CBLs:</strong></td>
<td>0</td>
</tr>
<tr>
<td><strong>Worst-Case Avertable CBLs:</strong></td>
<td>3</td>
</tr>
<tr>
<td><strong>Worst-Case Non-Avertable CBLs:</strong></td>
<td>3</td>
</tr>
<tr>
<td><strong>Desired Deadline Percentage</strong></td>
<td>50% of task deadline</td>
</tr>
</tbody>
</table>
5.3.2.1 – The Effect of ACETRB and WCETRB SPMMTs on RTOS Overhead

Results of experiments involving both the ACETRB (Table 25) and WCETRB (Table 26) SPMMTs are shown here to demonstrate that the RTOS percent overhead is virtually identical for both SPMMTs. This is because, since neither of these SPMMTs use the automated pointschedulability analysis, or perform SPM swaps, the only factors affecting RTOS overhead are kernel operation and time statistic capture. The amount of time required for these RTOS operations is a direct function of task periodicity. The greater the periodicity, the less overhead is required to run tasks and grab their time statistics.

Table 25 – ACETRB SPMMT – RTOS Overhead Percentage For Aperiodic Disturbance Tests

<table>
<thead>
<tr>
<th>TOTAL TEST TIME (10 SECS)</th>
<th>T0 REPETITION PERIOD</th>
<th>T0 AVERAGE EXE-TIME</th>
<th>T1 REPETITION PERIOD</th>
<th>T1 AVERAGE EXE-TIME</th>
<th>IDLE TIME</th>
<th>RTOS OVERHEAD TIME</th>
<th>TOTAL TASK EXE TIME</th>
<th>RTOS OVERHEAD PERCENTAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>10000000</td>
<td>300000</td>
<td>16886</td>
<td>50000</td>
<td>8548</td>
<td>5893918</td>
<td>146634</td>
<td>395948</td>
<td>1.47%</td>
</tr>
<tr>
<td>10000000</td>
<td>600000</td>
<td>31748</td>
<td>100000</td>
<td>14695</td>
<td>5980690</td>
<td>125895</td>
<td>3093415</td>
<td>1.26%</td>
</tr>
<tr>
<td>10000000</td>
<td>1500000</td>
<td>85240</td>
<td>250000</td>
<td>42260</td>
<td>6675278</td>
<td>115892</td>
<td>3038830</td>
<td>1.16%</td>
</tr>
</tbody>
</table>

Table 26 – WCETRB SPMMT – RTOS Overhead Percentage For Aperiodic Disturbance Tests

<table>
<thead>
<tr>
<th>TOTAL TEST TIME (10 SECS)</th>
<th>T0 REPETITION PERIOD</th>
<th>T0 AVERAGE EXE-TIME</th>
<th>T1 REPETITION PERIOD</th>
<th>T1 AVERAGE EXE-TIME</th>
<th>IDLE TIME</th>
<th>RTOS OVERHEAD TIME</th>
<th>TOTAL TASK EXE TIME</th>
<th>RTOS OVERHEAD PERCENTAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>10000000</td>
<td>300000</td>
<td>16915</td>
<td>50000</td>
<td>8535</td>
<td>5897281</td>
<td>147250</td>
<td>3955469</td>
<td>1.47%</td>
</tr>
<tr>
<td>10000000</td>
<td>600000</td>
<td>31803</td>
<td>100000</td>
<td>15361</td>
<td>5986735</td>
<td>125585</td>
<td>3087680</td>
<td>1.26%</td>
</tr>
<tr>
<td>10000000</td>
<td>1500000</td>
<td>85244</td>
<td>250000</td>
<td>42307</td>
<td>6672363</td>
<td>116200</td>
<td>3011437</td>
<td>1.16%</td>
</tr>
</tbody>
</table>
5.3.2.1 – The Effect of SDAWS on RTOS Overhead

Table 27 shows the RTOS overhead required for execution of SDAWS. As with the ACETRB and WCETRB tests, the greater the period, the less overhead is required to run tasks and grab their time statistics.

Table 27 – SDAWS SPMMT – RTOS Overhead Percentage For Aperiodic Disturbance Tests

<table>
<thead>
<tr>
<th>TOTAL TEST TIME (10 SECS)</th>
<th>T0 REPETITION PERIOD:</th>
<th>T0 AVERAGE EXE-TIME:</th>
<th>T1 REPETITION PERIOD:</th>
<th>T1 AVERAGE EXE-TIME:</th>
<th>IDLE TIME:</th>
<th>RTOS OVERHEAD TIME</th>
<th>TOTAL TASK EXE TIME</th>
<th>RTOS OVERHEAD PERCENTAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000000</td>
<td>300000</td>
<td>16915</td>
<td>50000</td>
<td>8353</td>
<td>4884807</td>
<td>116743</td>
<td>3947456</td>
<td>11.68%</td>
</tr>
<tr>
<td>1000000</td>
<td>600000</td>
<td>33803</td>
<td>100000</td>
<td>14327</td>
<td>5625447</td>
<td>48574</td>
<td>388809</td>
<td>4.86%</td>
</tr>
<tr>
<td>1000000</td>
<td>150000</td>
<td>83633</td>
<td>250000</td>
<td>42122</td>
<td>8081973</td>
<td>125286</td>
<td>3792741</td>
<td>1.25%</td>
</tr>
</tbody>
</table>

Figure 23 – RTOS % Overhead for Aperiodic Disturbance Tests: Task Periodicity = “T1 Repetition Period”

The RTOS percent overhead is drastically increased for SDAWS because it requires automated point-schedulability analysis and SPM swaps. However, as task periodicity increases the RTOS overhead required by SDAWS levels out, primarily due to a decreased number of
schedulability analyses. In the aperiodic disturbance tests whose RTOS overhead results are shown in Figure 23, SPM swaps are performed only during the aperiodic disturbance event. Because SPM swap time is only approximately 4000 microseconds, and because only a small number of SPM swaps are performed, RTOS overhead is primarily due to point-schedulability analyses and timing analyses. An RTOS overhead of close to 12% is significant (the primary down-side to SDAWS); however, a modest 1.25% RTOS overhead makes SDAWS competitive with the other SPMMTs depending on SPM swap periodicity and swap time.

5.3.3 – The Effect of The SPMST to WCET Ratio on SDAWS

Though overhead due to schedulability analyses plays a larger role than memory swaps regarding overall RTOS overhead, the amount of time required for a single memory swap is much greater than the amount of time required for a single task scheduling-point or completion-point scan. (In this thesis’ testing, SPM swaps are greater by approximately 20:1.) SPM swaps are performed at task scheduling points and task completion points. However, with respect to task slack times, the primary difference between SDAWS and the ACETRB / WCETRB SPMMTs rests in the fact that SDAWS requires substantial memory swaps to be performed at task scheduling points — as WCETRB to ACETRB (completion point) memory swaps are performed only when tasks are assured to meet their deadlines.

Despite the dominance of SPM swap time on SDAWS performance, it is not enough to show SDAWS performance results according to swap times alone. This is because what matters is the overall effect swap time has on WCET reduction. Thus, what matters is the amount of SPM swap time required relative to the worst-case task stack-up WCET, (i.e., the sum of each
task’s possible WCET in the event of a stack-up). Therefore, this section shows the effect of ACETRB to WCETRB (scheduling-point) SPM swap time (SPMST) versus worst-case task stack-up WCET, abbreviated the SPMST:WCET ratio. In this context, swap time, memory swap time, SPM swap time are all synonymous with SPMST.

5.3.3.1 – Clear Example: When SDAWS Performs Worse Than The ACETRB SPMMT

Tables 28 and 29 (below) show the parameters and the results of an experiment in which the SPMST:WCET ratio was set to approximately 1:6. (SPMST is approximately 4000 microseconds, and worst-case task stack-up WCET is approximately 24200.) Therefore, the SPMST is approximately 16.67% of the WCET. An SPMST:WCET ratio of 1:6 is very large, and therefore works to the detriment of SDAWS.

Table 28: Experiment Parameters: Worst-Case Processor Utilization For Fixed Desired Deadline

<table>
<thead>
<tr>
<th>Experiment Independent Variables:</th>
<th>Worst Case Processor Utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Tasks:</td>
<td>2</td>
</tr>
<tr>
<td>WCET:ACET Ratio:</td>
<td>2:1</td>
</tr>
<tr>
<td>Memory Accesses Per CBL:</td>
<td>400</td>
</tr>
<tr>
<td>Average-Case Avertable CBLs:</td>
<td>3</td>
</tr>
<tr>
<td>Average-Case Non-Avertable CBLs:</td>
<td>0</td>
</tr>
<tr>
<td>Worst-Case Avertable CBLs:</td>
<td>3</td>
</tr>
<tr>
<td>Worst-Case Non-Avertable CBLs:</td>
<td>3</td>
</tr>
<tr>
<td>Desired Deadline Percentage</td>
<td>50% of task deadline</td>
</tr>
<tr>
<td>SPMST:WCET Percentage</td>
<td>16.67%</td>
</tr>
</tbody>
</table>
Table 29: Two Identical Tasks: SDAWS Performance For SPMST:WCET Equal To 1:6

<table>
<thead>
<tr>
<th>WC Processor Utilization %</th>
<th>Repetition Period</th>
<th>To Worst-Case Slack Time:</th>
<th>To Average Slack Time:</th>
<th>To Worst-Case Execution Time:</th>
<th>To Average Execution Time:</th>
<th>To Worst-Case Slack Time:</th>
<th>To Average Execution Time:</th>
<th>SDAWS % Slack Time Increase Versus ACETRB</th>
<th>SDAWS % WCET Decrease Versus ACETRB</th>
<th>SDAWS % ACET Increase Versus ACETRB</th>
</tr>
</thead>
<tbody>
<tr>
<td>24%</td>
<td>1000000</td>
<td>75570</td>
<td>93081</td>
<td>12153</td>
<td>6011</td>
<td>5399</td>
<td>73559</td>
<td>-0.04%</td>
<td>-0.01%</td>
<td>0.00%</td>
</tr>
<tr>
<td>40%</td>
<td>500000</td>
<td>25567</td>
<td>42595</td>
<td>12132</td>
<td>6042</td>
<td>5399</td>
<td>5558</td>
<td>-0.13%</td>
<td>0.01%</td>
<td>0.00%</td>
</tr>
<tr>
<td>66%</td>
<td>330000</td>
<td>6294</td>
<td>22275</td>
<td>10962</td>
<td>6125</td>
<td>5814</td>
<td>706</td>
<td>-17.51%</td>
<td>10.68%</td>
<td>7.68%</td>
</tr>
<tr>
<td>79%</td>
<td>27777</td>
<td>3051</td>
<td>18110</td>
<td>10962</td>
<td>6316</td>
<td>5814</td>
<td>1870</td>
<td>-44.43%</td>
<td>10.68%</td>
<td>7.68%</td>
</tr>
</tbody>
</table>

Important Note: The worst-case (WC) processor utilization shown in Figure 24 below is based upon the worst-case processor utilization under the ACETRB SPMMT. The WC processor utilization shown in Table 29 above is the SDAWS WC processor utilization, but does not include SPMST.

Figure 24: SDAWS % Slack Time Increase Versus ACETRB for SPMST:WCET Equal To 1:3

Figure 24 (above) shows the slack time performance of the SDAWS SPMMT over the ACETRB SPMMT for the 1:6 SPMST:WCET ratio experiment. Because the desired deadline threshold is 50%, the tests with 24% and 49% worst-case processor utilization do not experience any SPM swaps — and therefore achieve roughly the same slack time performance as the
ACETRB SPMMT. However, as soon as SPM swaps start to occur, problems arise. Because the SPMST is very large, SPM swaps delay task execution enough such that, (even though task WCETs are reduced), task slack time is reduced by as much as 45%.

Figure 25 (below) suggests the reason for this reduction in slack time. As soon as swaps are performed, the SDAWS WCET + SPMST combination exceeds the WCET under the ACETRB SPMMT. This clearly underlines the importance of the SPMST:WCET ratio, and indicates that for large SPMST:WCET ratios, SDAWS performs worse than ACETRB SPMM.

![Figure 25: Total WCET (Including SPM Swap Time) for SPMST:WCET Equal To 1:3](image_url)
5.3.3.2 – Memory Swap Time Testing – When SDAWS Works

Though large SPMSTs can cause SDAWS’ slack time performance to be worse than that of ACETRB SPMM, an SPMST of zero would cause SDAWS’ slack time performance to be identical to that of the WCETRB SPMMT. In between these two extremes are reasonable possibilities. The following results come from an experiment in which the SPMST:WCET ratio was changed by holding all other parameters constant while changing the SPMST.

Tables 30 and 31 show the parameters and results of an experiment testing the effect of the SPMST:WCET percentage. In this experiment, two identical tasks with low priorities, short execution times, and short deadlines run on the processor. Further, a third task with a very large period (to simulate an aperiodic disturbance) and a large execution time runs on the processor.

Table 30: Experiment Parameters: Aperiodic Disturbance Test with Varied SPMST:WCET Percentage

<table>
<thead>
<tr>
<th>Experiment Independent Variables:</th>
<th>SPMST:WCET Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Number of Tasks:</strong></td>
<td>3 (2 small, low priority, high frequency; 1 large, high priority, low frequency)</td>
</tr>
<tr>
<td><strong>WCET:ACET Ratio:</strong></td>
<td>2:1</td>
</tr>
<tr>
<td><strong>Memory Accesses Per CBL:</strong></td>
<td>400</td>
</tr>
<tr>
<td><strong>Average-Case Avertable CBLs:</strong></td>
<td>3</td>
</tr>
<tr>
<td><strong>Average-Case Non-Avertable CBLs:</strong></td>
<td>0</td>
</tr>
<tr>
<td><strong>Worst-Case Avertable CBLs:</strong></td>
<td>3</td>
</tr>
<tr>
<td><strong>Worst-Case Non-Avertable CBLs:</strong></td>
<td>3</td>
</tr>
<tr>
<td><strong>Desired Deadline Percentage</strong></td>
<td>50% of task deadline</td>
</tr>
</tbody>
</table>
The desired deadline threshold for this experiment is set to 60%. Note in Table 31 (above) that for the first three memory swap times, only three memory swaps are performed, (i.e., when all tasks execute according to their WCETs at 97% No SPM WC Proc Utilization). However, for the final three memory swap times, 18 memory swaps are performed, (i.e., whenever the aperiodic tasks executes). This is due to the fact that the point-schedulability analysis used to determine when to perform memory swaps includes the memory swap time in the equation. If memory swap times are large, swapping is performed sooner to provide more schedulability assurance and prevent memory swap oscillation.

<table>
<thead>
<tr>
<th>SDAWS</th>
<th>MEMORY SWAP NUMBER OF BYTES</th>
<th>MEMORY SWAP TIME</th>
<th>NUMBER OF MEMORY SWAPS</th>
<th>NO SPM: WORST-CASE PROC UTILIZATION</th>
<th>SDAWS WORST-CASE PROC UTILIZATION WITHOUT SPM SWAP TIME</th>
<th>ACTUAL SDAWS WORST-CASE PROC UTILIZATION</th>
<th>WORST-CASE TEST SLACK TIME</th>
<th>TOTAL WCET TIME</th>
<th>SPM SWAP TIME AS A PERCENTAGE OF WORST-CASE STACK-UP WCET</th>
<th>SDAWS % SLACK TIME IMPROVEMENT OVER ACRETB</th>
<th>WCET + SPM SWAP TIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>SDAWS</td>
<td>40</td>
<td>799</td>
<td>3</td>
<td>97%</td>
<td>93.07%</td>
<td>91.27%</td>
<td>43571</td>
<td>465369</td>
<td>0.17%</td>
<td>180.63%</td>
<td>466162</td>
</tr>
<tr>
<td>SDAWS</td>
<td>120</td>
<td>2092</td>
<td>3</td>
<td>97%</td>
<td>92.99%</td>
<td>91.69%</td>
<td>41555</td>
<td>464897</td>
<td>0.45%</td>
<td>167.03%</td>
<td>466989</td>
</tr>
<tr>
<td>SDAWS</td>
<td>240</td>
<td>4099</td>
<td>3</td>
<td>97%</td>
<td>92.96%</td>
<td>92.45%</td>
<td>37770</td>
<td>464782</td>
<td>0.87%</td>
<td>142.71%</td>
<td>468811</td>
</tr>
<tr>
<td>SDAWS</td>
<td>480</td>
<td>7870</td>
<td>18</td>
<td>97%</td>
<td>92.65%</td>
<td>93.99%</td>
<td>30046</td>
<td>463273</td>
<td>1.70%</td>
<td>93.07%</td>
<td>471143</td>
</tr>
<tr>
<td>SDAWS</td>
<td>1800</td>
<td>29304</td>
<td>18</td>
<td>97%</td>
<td>92.94%</td>
<td>98.81%</td>
<td>5981</td>
<td>464703</td>
<td>6.21%</td>
<td>-61.25%</td>
<td>494012</td>
</tr>
<tr>
<td>SDAWS</td>
<td>2400</td>
<td>39048</td>
<td>18</td>
<td>97%</td>
<td>92.94%</td>
<td>100.75%</td>
<td>-3772</td>
<td>464724</td>
<td>8.40%</td>
<td>-124.24%</td>
<td>503772</td>
</tr>
</tbody>
</table>
Figure 26: Aperiodic Distortion – SDAWS % Slack Time Improvement Over ACETRB for Memory Swap Times

Figure 26 (above) shows the effect of the SPMST:WCET ratio on SDAWS slack time improvement versus ACETRB SPMM. As the ratio increases, slack time improvement decreases and eventually drops into the negative range, (which means SDAWS slack times are smaller than ACETRB SPMM slack times).
Figure 27: Aperiodic Distortion – Total WCETs (Including SPM Swap Time) for Memory Swap Times

In Figure 27 (above), the reason for the decrease in SDAWS slack time improvement can be seen. The SDAWS WCET + SPMST sum exceeds the ACETRB SPMM WCET for the largest SPMST:WCET ratios, causing slack time performance to fall below that of the ACETRB SPMMT.
Figure 28: Aperiodic Distortion — Worst-Case Slack Times for Memory Swap Times

Figure 28 (above) shows that not only does SDAWS worst-case slack time fall below ACETRB SPMM worst-case slack time as the SPMST:WCET ratio increases, but also the lowest priority task under SDAWS actually fails to meet its deadline (negative slack time) for an SPMST that is 8.4% of the WCET. This continues to worsen as the ratio increases.
Figure 29: Aperiodic Distortion – Worst-Case Processor Utilization for Memory Swap Times

Figure 29 (above) shows SDAWS worst-case processor utilization versus ACETRB WCETRB SPMM worst-case processor utilization. In the case where the low priority task under SDAWS fails to meet its deadline, worst-case processor utilization exceeds 100%, (which confirms that the deadline could not be met). This graph shows the percentage of processor execution time that is gained through the use of SDAWS versus ACETRB SPMM and lost through the use of SDAWS versus WCETRB SPMM.
5.3.3.4 – The Effect of Unbalanced Task-Wise SPM Allocation on SDAWS

Instead of maintaining strictly balanced ACETRB / WCETRB SPM allocation (in which each task is assigned an equal amount of SPM), certain tasks may be allocated more SPM than others. Considering this further, one may be inclined to believe that unbalanced SPM allocation ought to have some effect on worst-case task slack times. However, this is not the case. Tables 32 and 33 show that there is no difference between worst-case slack times for two very different task-wise SPM allocations.

Table 32: Aperiodic Distortion – SRAM Priority To Aperiodic Tasks

<table>
<thead>
<tr>
<th>SPMMT</th>
<th>WORST-CASE PROC UTILIZATION VIA WCETs</th>
<th>WORST-CASE PROC UTILIZATION VIA SLACK TIME</th>
<th>WORST-CASE TEST SLACK TIME ORIGINAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACETRB</td>
<td>97.26%</td>
<td>97.36%</td>
<td>6536</td>
</tr>
<tr>
<td>WCETRB</td>
<td>88.68%</td>
<td>88.76%</td>
<td>28040</td>
</tr>
<tr>
<td>SDAWS</td>
<td>97.26%</td>
<td>96.03%</td>
<td>9934</td>
</tr>
</tbody>
</table>

Table 33: Aperiodic Distortion – SRAM Priority To Periodic Tasks

<table>
<thead>
<tr>
<th>SPMMT</th>
<th>WORST-CASE PROC UTILIZATION VIA WCETs</th>
<th>WORST-CASE PROC UTILIZATION VIA SLACK TIME</th>
<th>WORST-CASE TEST SLACK TIME ORIGINAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACETRB</td>
<td>97.26%</td>
<td>97.36%</td>
<td>6596</td>
</tr>
<tr>
<td>WCETRB</td>
<td>88.68%</td>
<td>88.78%</td>
<td>28050</td>
</tr>
<tr>
<td>SDAWS</td>
<td>97.26%</td>
<td>96.02%</td>
<td>9939</td>
</tr>
</tbody>
</table>

The reason for the lack of worst-case slack time difference between the different SPM distributions lies in the fact that the same number of worst-case loads are averted in each case,
(thus keeping SPMST:WCET ratio identical). In a real scenario it may be possible for alternate task distributions of SPM to have an effect on slack time performance if, for example, more worst-case CBLs can be averted for one task than for another due to the spread of data within a program. In this scenario, it is possible for worst-case slack times to be increased or decreased versus a balanced task-wise SPM distribution.

However, this thesis does not presume to make claims regarding SDAWS / ACETRB / or WCETRB SPMMT worst-case slack time performance under different task-wise SPM distributions, because to make such claims would require significantly more test data requiring information on common special localities for data in embedded systems. Therefore, this thesis assumes that special locality of data, (i.e., the relative locations of sets of data within a program — affecting CBLs), is the same for all simulated tasks. What is certainly true about unbalanced task-wise SPM distributions is that they can achieve different average execution time reductions. This is because, for a set of tasks with different periodicities, reducing the ACET of tasks that run more often will result in reduced average execution time.

5.3.4 – The Effect of Desired Deadlines on SDAWS

In addition to SPM swap time, SDAWS is affected by schedulability analyses. While the effect of schedulability analyses on SDAWS overhead has been discussed, the desired deadline threshold used by the schedulability analyses has not. The desired deadline threshold is the percentage of actual task deadline that triggers the SDAWS memory swap. If a single task is unable to meet its desired deadline, then under SDAWS the RTOS performs ACETRB to WCETRB swaps and enters WCETRB mode.
5.3.4.1 – A Desired Deadline Equal to The Actual Task Deadline

Table 34 (below) shows the parameters for an experiment in which the desired deadline percentage is held fixed while task periodicity is changed. The purpose of this experiment is to compare the ACETRB SPMMT to SDAWS when the RTOS remains in ACETRB mode.

Table 34: Experiment Parameters: Task WCET For Fixed Desired Deadline

<table>
<thead>
<tr>
<th>Experiment Independent Variables:</th>
<th>Task WCET</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Tasks:</td>
<td>2 (Identical)</td>
</tr>
<tr>
<td>WCET:ACET Ratio:</td>
<td>2:1</td>
</tr>
<tr>
<td>Memory Accesses Per CBL:</td>
<td>400</td>
</tr>
<tr>
<td>Average-Case Avertable CBLs:</td>
<td>3</td>
</tr>
<tr>
<td>Average-Case Non-Avertable CBLs:</td>
<td>0</td>
</tr>
<tr>
<td>Worst-Case Avertable CBLs:</td>
<td>3</td>
</tr>
<tr>
<td>Worst-Case Non-Avertable CBLs:</td>
<td>3</td>
</tr>
<tr>
<td>Desired Deadline Percentage</td>
<td>100% of task deadline</td>
</tr>
</tbody>
</table>

In Table 35 (above) it is clear that the RTOS never enters WCETRB mode and that no SPM swaps are ever performed. However, it is worth noting that there is virtually no difference
in worst-case slack time, WCET, and ACET between ACETRB and SDAWS SPMMTs. SDAWS worst-case slack time gets slightly worse as worst-case processor utilization increases, but only by 0.13% (due to point-schedulability analysis).

5.3.4.1 – Desired Deadline Testing – What Desired Deadlines Should Be

Tables 36 and 37 show the parameters and results of an experiment in which five identical tasks with varying periodicities are run simultaneously.

Table 36: Desired Deadline Experiment Parameters

<table>
<thead>
<tr>
<th>Experiment Independent Variables:</th>
<th>Desired Deadline Percentage, SPMMT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Tasks:</td>
<td>5 (Identical with different periodicities)</td>
</tr>
<tr>
<td>WCET:ACET Ratio:</td>
<td>2:1</td>
</tr>
<tr>
<td>Memory Accesses Per CBL:</td>
<td>400</td>
</tr>
<tr>
<td>Average-Case Avertable CBLs:</td>
<td>3</td>
</tr>
<tr>
<td>Average-Case Non-Avertable CBLs:</td>
<td>0</td>
</tr>
<tr>
<td>Worst-Case Avertable CBLs:</td>
<td>3</td>
</tr>
<tr>
<td>Worst-Case Non-Avertable CBLs:</td>
<td>3</td>
</tr>
</tbody>
</table>
Table 37: Five Identical Tasks w/ Different Periodicity – Desired Deadlines

<table>
<thead>
<tr>
<th>NUMBER OF TASKS</th>
<th>SPMMT</th>
<th>ACTUAL WORST-CASE PROCESSOR UTILIZATION</th>
<th>WORST-CASE SLACK-TIME:</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>ACETRB</td>
<td>95.04%</td>
<td>31747</td>
</tr>
<tr>
<td>5</td>
<td>WCETRB</td>
<td>86.65%</td>
<td>85431</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SPMMT</th>
<th>DESIRED DEADLINE %</th>
<th>ACTUAL WORST-CASE PROCESSOR UTILIZATION</th>
<th>WORST-CASE SLACK-TIME:</th>
<th>SLACK TIME IMPROVEMENT VERSUS ACETRB</th>
<th>SLACK TIME IMPROVEMENT VERSUS WCETRB</th>
</tr>
</thead>
<tbody>
<tr>
<td>SDAWS</td>
<td>45</td>
<td>95.43%</td>
<td>29262</td>
<td>-7.83%</td>
<td>-65.75%</td>
</tr>
<tr>
<td>SDAWS</td>
<td>65</td>
<td>92.09%</td>
<td>50641</td>
<td>59.51%</td>
<td>-40.72%</td>
</tr>
<tr>
<td>SDAWS</td>
<td>85</td>
<td>91.06%</td>
<td>57216</td>
<td>80.22%</td>
<td>-33.03%</td>
</tr>
</tbody>
</table>

SDAWS worst-case processor utilization decreases as the desired deadline threshold increases (Table 37). This is due to the fact that a small desired deadline threshold causes a greatly increased number of memory swaps. As the number of memory swaps required decreases, the worst-case processor utilization decreases. One is inclined to think that because the additional memory swaps do not occur during a worst-case task stack-up they should not affect worst-case processor utilization. This is not the case, because this experiment is performed under very resource-constrained conditions. After each worst-case task stack-up has completed, the RTOS may swap back to ACETRB mode. The amount of time required to swap back to ACETRB mode affects the next set of task execution times. Therefore, a greater number of swaps means a greater disturbance of worst-case task stack-up execution time.

This experiment was performed with a high SPMST:WCET ratio of 1:12., in order to show the effect of desired deadline % on slack time performance. For very small SPMST:WCET ratios, the desired deadline threshold has no effect on slack time performance because the amount of time required to swap back to ACETRB mode does not impede worst-case task stack-
up execution time. In Figure 30 (below) it should be noted that for a 45% desired deadline threshold, SDAWS’ slack time performance is worse than that of the ACETRB SPMMT.

Figure 30: Identical Tasks w/ Different Periodicity – SDAWS Slack Time Improvement for Desired Deadlines

Finally, it needs to be mentioned that average execution time reduction is maximized when desired deadlines are as large as is acceptable. This is because a smaller deadline threshold means that more time is spent in WCETRB mode. Therefore, there are several reasons why small desired deadline thresholds should not be used. Though they may provide more assurance that the RTOS will be running in WCETRB mode when it needs to be, additional memory swaps can negate any benefit from SDAWS over the ACETRB SPMMT.
CHAPTER 6 – CONCLUSION

The goal of this thesis was to develop an SPM management technique that serves as a compromise approach between average-case and worst-case execution time reduction-based SPM management. In order to accomplish this goal, this thesis developed an SPM management technique that switches between average-case and worst-case execution time reduction-based SPM allocations according to task schedulability. It was found that the SDAWS technique developed by this thesis is capable of serving as a compromise solution between ACETRB and WCETRB SPMM. However, the performance of SDAWS is highly dependent upon scenario, RTOS, and task characteristics, and the technique is not able to serve as a compromise solution in every case. To verify that the developed SPMMT met the goal of serving as a compromise approach, this thesis contributed these primary achievements:

- Developed an RTOS and simulation platform from which to evaluate ACETRB, WCETRB, and SDAWS SPMMT performance.
- Performed comprehensive simulation testing of each of these SPMMTs under a variety of scenarios.
- Evaluated each SPMMT with respect to the others.

The RTOS and simulation platform were successfully able to run a large number of simulated hard real-time tasks simultaneously, collect timing statistics, and perform SPM management operations such as automatic schedulability analysis. The RTOS and simulator could be extended to test other SPM management techniques, or converted into a stand-alone RTOS solution which performs SDAWS. Moreover, it was found that the overhead introduced by the RTOS and simulator for the static SPMMTs was very minimal. While RTOS overhead for
SDAWS was significant for certain task scenarios, it did not prevent SDAWS from demonstrating clear improvements over the static SPMMTs.

Difficulty was encountered collecting and developing an SPMMT simulator that accounted for all relevant task parameters. Further difficulty was encountered generating test scenarios that thoroughly covered the range of parameter changes which affected SPMMT performance. This thesis invested an equal amount of time into RTOS development, simulator development, and performing SPMMT experiments.

It was found that differences between ACETRB and WCETRB SPMMT performance are highly dependent upon task execution path characteristics. In particular, the number of cache block loads along the worst-case and average-case execution paths and the ratio of cache block loads to total execution path execution time play a dominant role. Moreover, this thesis shows that ACET reductions under the ACETRB SPMMT and WCET reductions under the WCETRB SPMMT range between 10% and 60% for the tested scenarios, where the amount of reduction is dependent upon the cache block load to execution time ratio. These results are consistent with [3, 8, and 17], which claim ACET or WCET reductions between 35% and 60%. Due to WCET reductions, use of the WCETRB SPMMT results in a minimum of 10% reduction in worst-case processor utilization under heavily resource-constrained conditions. The percentage of slack time gained through the use of the WCETRB SPMMT increases exponentially versus the ACETRB SPMMT as worst-case processor utilization increases, but for 90% worst-case processor utilization (under the ACETRB SPMMT), this thesis shows that the WCETRB SPMMT reduces slack times by between 70% and 90%.

Because SDAWS relies on these two techniques, any relative improvement from SDAWS over either technique is dependent upon these execution path characteristics as well.
The primary factors affecting ability of SDAWS to serve as a compromise approach are its ability to offer increased slack time versus the ACETRB SPMMT and reduced ACET versus the WCETRB SPMMT. Increases in slack time are driven by SDAWS SPM swap time, while reductions in ACET are driven by RTOS overhead due to schedulability analyses. For scenarios in which SPM swap and schedulability overhead exceeded reductions in WCET due to ACETRB to WCETRB swapping, SDAWS performs worse than the ACETRB SPMMT according to every metric, and is therefore unable to serve as a compromise solution. However, for scenarios in which the additional overhead due to SDAWS is smaller than execution time gains due to swapping, SDAWS is able to serve as a compromise solution. Findings regarding the relative performance of SDAWS, ACETRB, and WCETRB SPMMTs under simulated system conditions are shown in table uuu. It should be noted that, because SDAWS requires additional RTOS overhead in order to operate, some system conditions affect SDAWS performance but not ACETRB or WCETRB performance.

Table 38: Summary of Findings

<table>
<thead>
<tr>
<th>System Conditions</th>
<th>SPMMT</th>
<th>SPMMT Performance Characterization</th>
</tr>
</thead>
<tbody>
<tr>
<td>High task frequency</td>
<td>SDAWSs</td>
<td>Weak performance due to high overhead</td>
</tr>
<tr>
<td>Low task frequency</td>
<td>SDAWS</td>
<td>Strong performance due to low overhead</td>
</tr>
<tr>
<td>High SPM Swap Time :WCET Ratio *</td>
<td>SDAWS</td>
<td>Weak performance due to high overhead</td>
</tr>
<tr>
<td>Low SPM Swap Time :WCET Ratio *</td>
<td>SDAWS</td>
<td>Strong performance due to low overhead</td>
</tr>
<tr>
<td>SPMST + PSAT *** &lt; SDAWS’ WCET Improvement</td>
<td>SDAWS</td>
<td>SDAWS slack times &gt; ACETRB slack times. *</td>
</tr>
<tr>
<td>SPMST + PSAT *** &gt; SDAWS’ WCET Improvement</td>
<td>SDAWS</td>
<td>SDAWS slack times &lt; ACETRB slack times. SDAWS cannot be used for the RT system. *</td>
</tr>
<tr>
<td>High Worst-Case Processor Utilization</td>
<td>SDAWS, WCETRB</td>
<td>Strong slack time improvements</td>
</tr>
<tr>
<td>Percentage *</td>
<td>versus ACETRB*</td>
<td></td>
</tr>
<tr>
<td>--------------</td>
<td>----------------</td>
<td></td>
</tr>
<tr>
<td>Low Worst-Case Processor Utilization %</td>
<td>SDAWS, WCETRB</td>
<td>Weak slack time improvements versus ACETRB *</td>
</tr>
<tr>
<td>Small / infrequent worst-case path execution</td>
<td>ACETRB</td>
<td>Strong average execution time reduction versus WCETRB*</td>
</tr>
<tr>
<td>Large / frequent worst-case path execution</td>
<td>ACETRB</td>
<td>Weak average execution time reduction versus WCETRB*</td>
</tr>
<tr>
<td>Large number of avertable cache block loads per execution path **</td>
<td>ACETRB, WCETRB, SDAWS</td>
<td>Significant execution path reduction due to SPM allocation</td>
</tr>
<tr>
<td>Small number of avertable cache block loads per execution path **</td>
<td>ACETRB, WCETRB, SDAWS</td>
<td>Insignificant execution path reduction due to SPM allocation</td>
</tr>
</tbody>
</table>

* See Chapter 5, section 5.1.1
** See Chapter 5, section 5.3.2
*** See Chapter 5, section 5.3.3

Because it is possible to decrease SDAWS overhead through the use of DMA and code optimizations, and because embedded processors are becoming increasingly powerful and are able to handle additional overhead, the SDAWS technique is ready to be adopted by industry. However, SDAWS is dependent upon ACETRB and WCETRB SPMMT performance. Because ACETRB and WCETRB SPMMTs have yet to be widely adopted by industry, developers may wish to wait until these methods have been vetted before attempting to implement a more complex approach. Further, implementation of SDAWS requires a significant time investment (much more so than static SPMMTs); therefore it is likely that developers will wish to wait for researchers to vet SDAWS at an application level before incorporating it into a commercial system.
CHAPTER 7 – CONSIDERATIONS FOR FUTURE WORK

No SPM management technique can be understood so well that there is no benefit to additional testing. Furthermore, implementation of any SPMMT can always be optimized to fit the system on which it is running, or tweaked to increase performance. In addition, software tools may be developed in order to categorize or generate SPM allocations better for different SPMMTs. While these avenues could be exploited in order to optimize other types of SPMMTs, this Chapter discusses additional testing, optimizations, and black-box software solutions that should be investigated in order to better characterize and increase the performance of the SDAWS SPMMT.

6.1 – ADDITIONAL SDAWS SPMMT TESTING

This thesis thoroughly tested the SDAWS SPMMT with respect to the ACETRB and WCETRB SPMMTs – verifying that performance gains can be achieved through its use. However, because SDAWS is a new technique, additional testing should be performed either through simulations or through use in applications in order to prove its effectiveness under a multitude of scenarios. Furthermore, all of the simulations performed for this thesis were executed on a single 32-bit Freescale Coldfire platform. Therefore, it would be beneficial to perform additional testing in order characterize the performance of SDAWS on different platforms (e.g.: 8-bit, 64-bit, etc).

6.1.1 – Additional Simulation Testing with The RTOS and SPMMT Simulator
While enough data were obtained for this thesis through testing and simulation to validate the effectiveness of SDAWS for some scenarios, the amount of testing and analysis performed for this thesis was not exhaustive. The following testing could be performed in order to characterize the performance of SDAWS better:

1. Additional investigation into ACET reduction under SDAWS
   a. Under different RTOS overhead conditions
   b. Under varied resource-constrained real-time task scenarios
2. Additional investigation into task slack time performance under SDAWS under varied resource-constrained real-time task scenarios

6.1.2 – Comparison of SDAWS to other SPMMTs

This thesis compares the SDAWS SPMMT only to the ACETRB and WCETRB SPMMTs. However, there are many different types of SPMMTs, (e.g.: dynamic page-based, stack-based, RTOS minimization-based, etc). To form a more complete picture of SDAWS benefits / detriments, it would be valuable to compare SDAWS to these other SPMMTs. However, the ACETRB and WCETRB SPMMTs have not yet been fully characterized with respect to the other SPMMTs, so characterizing them would have to come first.

6.1.3 – Application Testing of SDAWS

In addition to testing via simulation, application testing is a means of validating the effectiveness of design techniques. While simulation testing gives a well-rounded picture of task execution times under SDAWS in terms of benefit / detriment to execution times and slack times in the best and worst cases, simulation testing does not absolutely confirm that SDAWS will
achieve similar results on a real system. Real scenarios can be very complex, and simulation parameters cannot account for every possible variable. Instead, simulation parameters account only for those variables that are expected to have an effect on performance under SDAWS. Consequently, application testing through the implementation of the SDAWS SPMMT on a real system would provide concrete evidence to reinforce the performance results obtained for this thesis.

6.2 – SDAWS IMPLEMENTATION OPTIMIZATION

Though the RTOS and SPMMT simulator used simulated tasks to validate SDAWS performance gains, the RTOS implemented for this thesis used a real implementation of the SDAWS SPMMT (not a simulation of SDAWS – though memory swap times were simulated). The advantage of the real implementation is to minimize the number of simulation factors present in each test case. However, this means that the results obtained for this thesis are directly affected by the amount of overhead incurred by the specific SDAWS implementation employed. Therefore, it is possible to obtain better SDAWS results through the optimization of the SDAWS implementation.

6.2.1 – Schedulability Analysis Optimization

The results obtained for this thesis show that a significant amount of RTOS overhead is caused by schedulability analyses performed in order to determine whether or not to swap SPM allocation modes. Therefore, it makes sense to consider ways of reducing this overhead in order to increase performance.
6.2.1.1 – Schedulability Analysis Computation Time Optimization

The schedulability analysis algorithm used by this thesis (described in 4.1.1) is implemented in the RTOS in a non-ideal manner. That is, the algorithm is implemented in a way such that an unnecessarily large portion of time is spent iterating through task structures. The schedulability analysis algorithm implementation could be optimized by utilizing look up tables (LUTs) in place of task structures and loops.

6.2.1.2 – Predictive/Schedule-Aware Schedulability-Analysis

This thesis does not execute schedulability analysis code until tasks are ready to run on the processor (due to the design of the automatic task scheduler used by the RTOS). This is very inefficient, as it requires the schedulability analysis time to be appended to any worst-case task execution times, reducing worst-case slack time. It would be much more beneficial to execute schedulability analysis code as soon as the RTOS can determine when tasks will need to run. This could add additional overhead in the form of additional task scheduling time / schedule tables used by the RTOS; however, the advantage of doing this would be to detect worst-case task stack-ups before they occurred, either by executing schedulability analysis code as soon as tasks are scheduled to run in the future, or while the processor is idle. In addition, predictive schedulability analysis could theoretically be performed in which the RTOS uses runtime parameters to predict when task worst-case executions will occur and perform schedulability analyses only during the predicted worst-case time slices.
6.2.1.3 – Stable-State WCETRB to ACETRB Swapping

This thesis’ SDAWS implementation requires the RTOS to swap to ACETRB mode as soon as all tasks currently scheduled on the processor are able to meet their deadlines. This is problematic, as in the case where worst-case task stack-ups occur frequently it is likely that SPM allocation modes will oscillate continuously as tasks complete and are scheduled again. In order to avoid the tremendous amount of overhead and performance detriment that this kind of oscillation causes (see 5.3.3.1), a condition can be added to the WCETRB to ACETRB SPM allocation mode swap that requires that all tasks are able to meet their desired deadlines for a set interval of time (say, five milliseconds, for example) before the RTOS is allowed to swap back to ACETRB SPM allocation mode.

6.2.2 – SPM Swap Optimization

SPM swap time adds a considerable amount of overhead and directly affects the ability of SDAWS to perform under resource-constrained conditions (see 5.3.2). Consequently, it is essential that SPM swap time be reduced as much as possible. Though the effect of swap time on SDAWS performance was demonstrated, this thesis used simulated amounts of SPM to control memory swap time. Therefore, this thesis does not provide an optimized implementation of the SPM swap controller. The RTOS for this thesis uses memcpy operations.

6.2.2.1 – Direct Memory Access (DMA)

SPM swap efficiency could be drastically improved (even more so than is demonstrated by the results of this thesis) through the use of direct memory access (DMA). DMA is a hardware mode that is incorporated on many embedded processors, which allows for large
memory blocks to be copied from one segment of memory to another (i.e.: from main memory to SPM) while allowing the processor to execute other instructions. Therefore, DMA could allow SPM swap time to be free, but only on the condition is that no task can use the memory involved in the DMA while the DMA is taking place. Therefore, SPM swaps for tasks not being executed currently could be performed while other tasks are being executed, (though the DMA may slow the effective memory bus speed temporarily for the tasks being executed).

6.2.3 – Alternate RTOS Configurations

This thesis uses a non-preemptive, fixed priority operating system that requires SPM swaps for all tasks to occur at the same time. The non-preemptive and fixed priority nature of the RTOS limits what can be applied from the results obtained for this thesis to other kinds of operating systems. Additional testing should be performed for these types of operating systems. Further, SDAWS overhead can be reduced through the use of alternate RTOS configurations.

6.2.3.1 – Earliest-Deadline First Scheduling

Earliest-deadline first (EDF) scheduling involves the use of task deadline knowledge to reprioritize tasks on the fly such that the tasks that need to complete the soonest are always run first. Specifically, EDF scheduling requires task completion point and task scheduling point scanning in order to find tasks with the soonest deadlines and then run them. This completion and scheduling point scanning that must be performed is extremely similar to the SDAWS schedulability analysis equations in 4.1.1. Therefore, the two schedulability analyses could be combined in order to reduce overhead incurred by the RTOS due to SDAWS. Combination of
schedulability analysis algorithms would significantly reduce the PSAT (point-schedulability analysis time) utilized in the equations described in 5.3.1.

6.2.3.2 – RTOS Preemption

Barring priority inversion scenarios, preemptive operating systems offer more schedulability assurance than non-preemptive operating systems due to immediate response to high priority tasks. Therefore, since many large embedded systems use preemptive RTOSes, it is worth testing SDAWS on a system that incorporates such an RTOS. In addition, SDAWS could take advantage of preemption in two ways:

1. While tasks are able to complete well before their deadlines, additional schedulability analyses could be performed during RTOS context switches.

2. Schedulability analyses could be re-run partway through task execution such that current task WCET would not need to be incorporated into the schedulability equation (only task ACET). As the tasks continue running, SPM swaps could be performed only after the current task has run for a long enough interval such that if other tasks executed according to their WCETs, they would not be able to complete. This would reduce unnecessary SPM swaps.

6.3 – DEVELOPMENT OF A BLACK-BOX SOLUTION

Because the development and implementation of an SPMMT on any RTOS requires a significant time investment, there is a need in the embedded market for tools which remove the burden of SPM allocation / SPM management from the developer. While some of the following tools may be in the process of being developed, none of them currently exist on the market.
6.3.1 – Compile-Time SDAWS SPM Allocator

A large portion of recent (published within the last ten years) SPMMT research surveyed by this thesis has targeted the creation of compile-time techniques for SPM allocation (see Chapter 3). In particular, ACETRB and WCETRB SPM allocation strategies have been developed, (though the WCETRB SPM allocation strategies rely on estimates that make them not ideal versus profiling-based SPM allocation techniques). Using these established techniques, it would be possible to obtain both WCETRB and ACETRB SPM allocations for any given task or set of tasks. These allocations could then be used by SDAWS. Ideally, these techniques would be incorporated into a compiler of some type that would automatically divide task memory into segments and generate source code for swapping different segments into and out of SPM.

6.3.2 – Debugger-Based Automated Profiling Tool For SDAWS Allocation

Instead of determining SPM allocations via techniques employed at compile time, profiling can be used to determine ACETRB and WCETRB SPM allocations. However, because profiling must be done at runtime, any profiling tool must count the precise number of memory accesses for every variable in the program. A debugger is generally the only tool that has access to this information; therefore, it is logical to incorporate profiling into the debugger. Because most debuggers come linked to a specific tool set designed to work with a specific microcontroller, it would be logical for the debugger/development tool suite to segment task memory according to different SPMMTs, and provide source code for accessing each segment as well as swapping different segments into and out of SPM.
6.3.3 – Debugger-Based Cache Block Load Analyzer

Because the goal of SPMMTs is to avoid cache block loads, it falls to reason that developers may desire to create their own SPM allocation that optimizes specific executions paths according to the number of cache block loads that occur along them. This is not possible without knowing how many cache block loads actually occur (in the worst-case and on average) along any given execution path. Furthermore, it is not possible to know speedup due to SPM without knowing how many CBLs were averted through the use of the SPMMT employed. Therefore, a cache block load analyzer would allow developers to determine along what execution paths the most CBLs were occurring and create an SPMMT to mitigate their effect on execution time. Further, such a tool would simply be useful, as it would help developers to avoid additional execution time caused by “spaghetti code” – in which a multitude of large jumps between code segments causes unnecessary CBLs (in addition to being bad style and hard to follow).
BIBLIOGRAPHY


APPENDIX A

This appendix includes the data tables associated with the experiments performed for this thesis. It only contains those data tables that have not already been displayed in Chapter 5 – results and analysis. All times indicated in the data tables are in microseconds. This section does not discuss the results displayed here. In order to do that, the reader will need to refer back to the corresponding SPMTT experiments in Chapter 5.

ONE-TASK EXPERIMENTS – ADDITIONAL RESULTS

Computations per Memory Access Experiment

Table 40 – Computations per Memory Access (1,000 Memory Accesses)

<table>
<thead>
<tr>
<th>cs per ma</th>
<th>WCET</th>
<th>ACET</th>
<th>AVACET</th>
<th>AVWCET</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACETRB</td>
<td>1</td>
<td>4843</td>
<td>4163</td>
<td>4095</td>
</tr>
<tr>
<td>ACETRB</td>
<td>2</td>
<td>5734</td>
<td>5081</td>
<td>5015</td>
</tr>
<tr>
<td>ACETRB</td>
<td>5</td>
<td>8407</td>
<td>7837</td>
<td>7780</td>
</tr>
<tr>
<td>ACETRB</td>
<td>10</td>
<td>12862</td>
<td>12429</td>
<td>12386</td>
</tr>
<tr>
<td>ACETRB</td>
<td>20</td>
<td>21775</td>
<td>21613</td>
<td>21598</td>
</tr>
</tbody>
</table>

Figure 31 – ACET for Computations per Memory Access (1,000 Memory Accesses)
SDRAM Equivalency Ratio Experiment

Table 41 – SRAM to SDRAM Access Ratio (1,000 Memory Accesses)

<table>
<thead>
<tr>
<th>S TO SDRAM</th>
<th>WCET</th>
<th>ACET</th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>4582</td>
<td>4484</td>
</tr>
<tr>
<td>50</td>
<td>4717</td>
<td>4554</td>
</tr>
<tr>
<td>100</td>
<td>5056</td>
<td>4730</td>
</tr>
<tr>
<td>200</td>
<td>5734</td>
<td>5081</td>
</tr>
<tr>
<td>400</td>
<td>7089</td>
<td>5784</td>
</tr>
</tbody>
</table>

Average and Worst-case CBL Experiments – Equal Non-Avertable and Avertable CBLs

The following tables show the results of CBL experiments in which average-case CBLs are equal to worst-case CBLs, and there are an identical number of worst-case and average-case memory accesses. Non-avertable and avertable CBLs were varied. The WCETRB SPMMT was not tested because the worst-case and average-case execution paths are identical; therefore the WCETRB SPMMT performs identically to the ACETRB SPMMT.

Table 42 – Non-avertable CBLs for the ACETRB SPMMT (2 AC:WC CBLs = 1:1) with Average-Case Accesses: Worst-Case Accesses = 1000:1000

<table>
<thead>
<tr>
<th>SPMMT</th>
<th>static CBLs</th>
<th>WCET</th>
<th>ACET</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACETRB</td>
<td>2</td>
<td>5734</td>
<td>5081</td>
</tr>
<tr>
<td>ACETRB</td>
<td>8</td>
<td>7787</td>
<td>7025</td>
</tr>
<tr>
<td>ACETRB</td>
<td>32</td>
<td>15998</td>
<td>14801</td>
</tr>
</tbody>
</table>
Table 43 – Avertable CBLs for the ACETRB SPMMT (2 AC:WC CBLs = 1:1) with Average-Case Accesses: Worst-Case Accesses = 1000:1000

<table>
<thead>
<tr>
<th>SPMMT</th>
<th>floating CBLs</th>
<th>WCET</th>
<th>ACET</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACETRB</td>
<td>2</td>
<td>5734</td>
<td>5081</td>
</tr>
<tr>
<td>ACETRB</td>
<td>8</td>
<td>7787</td>
<td>5268</td>
</tr>
<tr>
<td>ACETRB</td>
<td>32</td>
<td>15999</td>
<td>6015</td>
</tr>
</tbody>
</table>

**Average and Worst-case CBL Experiments – Avertable CBLs**

The following tables show the results of CBL experiments in which avertable average-case and avertable worst-case CBLs were modified, but there are an identical number of worst-case and average-case memory accesses.

Table 44 – AC Avertable CBLs for the ACETRB SPMMT with Average-Case Accesses: Worst-Case Accesses = 1000:1000

<table>
<thead>
<tr>
<th>SPMMT</th>
<th>floating AC CBLs</th>
<th>WCET</th>
<th>ACET</th>
<th>AVACET</th>
<th>AVWCET</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACETRB</td>
<td>2</td>
<td>6418</td>
<td>5144</td>
<td>5016</td>
<td>6415</td>
</tr>
<tr>
<td>ACETRB</td>
<td>8</td>
<td>10524</td>
<td>5517</td>
<td>5016</td>
<td>10521</td>
</tr>
<tr>
<td>ACETRB</td>
<td>32</td>
<td>26949</td>
<td>7010</td>
<td>5016</td>
<td>26946</td>
</tr>
</tbody>
</table>

Table 45 – WC Avertable CBLs for the ACETRB SPMMT with Average-Case Accesses: Worst-Case Accesses = 1000:1000

<table>
<thead>
<tr>
<th>SPMMT</th>
<th>floating WC CBLs</th>
<th>WCET</th>
<th>ACET</th>
<th>AVACET</th>
<th>AVWCET</th>
</tr>
</thead>
<tbody>
<tr>
<td>WCETRB</td>
<td>2</td>
<td>5753</td>
<td>5681</td>
<td>5750</td>
<td>4997</td>
</tr>
<tr>
<td>WCETRB</td>
<td>8</td>
<td>7806</td>
<td>7548</td>
<td>7803</td>
<td>4997</td>
</tr>
<tr>
<td>WCETRB</td>
<td>32</td>
<td>16018</td>
<td>15013</td>
<td>16015</td>
<td>4997</td>
</tr>
</tbody>
</table>
Average and Worst-case CBL Experiments – Non-Avertable CBLs

**Figure 32 – ACET (average/worst-case non-avertable CBLs)**

**Figure 33 – WCET (average/worst-case non-avertable CBLs)**
TWO-TASK EXPERIMENTS – ADDITIONAL RESULTS

Average Execution Time – ACET:WCET Ratio Experiment

Table 46 – ACET:WCET Ratio for SDAWS

<table>
<thead>
<tr>
<th>WCET TO ACET EXE TIME RATIO</th>
<th>T0 WORST-CASE SLACK-TIME</th>
<th>T0 AVERAGE SLACK-TIME</th>
<th>T0 WORST-CASE EXE-TIME</th>
<th>T0 AVERAGE EXE-TIME</th>
<th>T1 WORST-CASE SLACK-TIME</th>
<th>T1 AVERAGE SLACK-TIME</th>
<th>IDLE TIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>89021</td>
<td>93287</td>
<td>5739</td>
<td>5430</td>
<td>5399</td>
<td>88365</td>
<td>90144</td>
</tr>
<tr>
<td>2</td>
<td>75559</td>
<td>93081</td>
<td>12132</td>
<td>6011</td>
<td>5399</td>
<td>75570</td>
<td>88608</td>
</tr>
<tr>
<td>4</td>
<td>51354</td>
<td>91170</td>
<td>24236</td>
<td>7111</td>
<td>5399</td>
<td>51361</td>
<td>87217</td>
</tr>
</tbody>
</table>

Average Execution Time – WCET to ACET Ratio Experiment

Table 47: ACETRB SPMMT – Varied WCET to ACET Ratio

<table>
<thead>
<tr>
<th>WCET TO ACET EXE TIME RATIO</th>
<th>T0 WORST-CASE SLACK-TIME</th>
<th>T0 AVERAGE SLACK-TIME</th>
<th>T0 WORST-CASE EXE-TIME</th>
<th>T0 AVERAGE EXE-TIME</th>
<th>T1 WORST-CASE SLACK-TIME</th>
<th>T1 AVERAGE SLACK-TIME</th>
<th>ACETRB % ACET DECREASE VERSUS WCETR</th>
<th>ACETRB % AET DECREASE VERSUS WCETR</th>
<th>IDLE TIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>88369</td>
<td>92069</td>
<td>6419</td>
<td>5430</td>
<td>5399</td>
<td>87581</td>
<td>7.14%</td>
<td>5.88%</td>
<td>8652312</td>
</tr>
<tr>
<td>2</td>
<td>75578</td>
<td>91369</td>
<td>12132</td>
<td>6011</td>
<td>5399</td>
<td>75809</td>
<td>7.14%</td>
<td>4.31%</td>
<td>8599644</td>
</tr>
<tr>
<td>4</td>
<td>51366</td>
<td>90780</td>
<td>24236</td>
<td>7111</td>
<td>5399</td>
<td>51361</td>
<td>7.14%</td>
<td>2.27%</td>
<td>8328999</td>
</tr>
</tbody>
</table>

Table 48: WCETRB SPMMT – Varied WCET to ACET Ratio

<table>
<thead>
<tr>
<th>WCET TO ACET EXE TIME RATIO</th>
<th>T0 WORST-CASE SLACK-TIME</th>
<th>T0 AVERAGE SLACK-TIME</th>
<th>T0 WORST-CASE EXE-TIME</th>
<th>T0 AVERAGE EXE-TIME</th>
<th>T1 WORST-CASE SLACK-TIME</th>
<th>T1 AVERAGE SLACK-TIME</th>
<th>WCETRB % SLACK TIME INCREASE VERSUS ACETR</th>
<th>% WCET DECREASE VERSUS ACETR</th>
<th>IDLE TIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>88209</td>
<td>91379</td>
<td>5815</td>
<td>5769</td>
<td>5814</td>
<td>82230</td>
<td>0.73%</td>
<td>9.41%</td>
<td>8586722</td>
</tr>
<tr>
<td>2</td>
<td>77920</td>
<td>93334</td>
<td>10962</td>
<td>6262</td>
<td>5814</td>
<td>77935</td>
<td>3.01%</td>
<td>9.64%</td>
<td>8487652</td>
</tr>
<tr>
<td>4</td>
<td>50045</td>
<td>90278</td>
<td>21039</td>
<td>7278</td>
<td>5014</td>
<td>87056</td>
<td>8.44%</td>
<td>9.65%</td>
<td>8297045</td>
</tr>
</tbody>
</table>
Average-Cases Per Worst-Case Experiments

Table 49: ACETRB SPMMT – Varied Average Cases Per Worst-Case

<table>
<thead>
<tr>
<th>AVG CASES PER WORST CASE</th>
<th>TO WORST-CASE SLACK-TIME</th>
<th>TO AVERAGE SLACK-TIME</th>
<th>TO WORST-CASE EXE-TIME</th>
<th>TO AVERAGE EXE-TIME</th>
<th>T1 WORST-CASE SLACK-TIME</th>
<th>T1 AVERAGE SLACK-TIME</th>
<th>% ACETRB DECREASE VERSUS WCETRB</th>
<th>AVERAGE % AET DECREASE VERSUS WCETRB</th>
<th>IDLE TIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>75573</td>
<td>90899</td>
<td>12132</td>
<td>573</td>
<td>5977</td>
<td>7599</td>
<td>91715</td>
<td>17.79%</td>
<td>13.31%</td>
</tr>
<tr>
<td>3</td>
<td>75575</td>
<td>85207</td>
<td>12132</td>
<td>673</td>
<td>5977</td>
<td>7599</td>
<td>90200</td>
<td>17.79%</td>
<td>7.48%</td>
</tr>
<tr>
<td>1</td>
<td>75572</td>
<td>86551</td>
<td>12132</td>
<td>856</td>
<td>5977</td>
<td>7599</td>
<td>87512</td>
<td>17.79%</td>
<td>-0.28%</td>
</tr>
</tbody>
</table>

Table 50: WCETRB SPMMT – Varied Average Cases Per Worst-Case

<table>
<thead>
<tr>
<th>AVG CASES PER WORST CASE</th>
<th>TO WORST-CASE SLACK-TIME</th>
<th>TO AVERAGE SLACK-TIME</th>
<th>TO WORST-CASE EXE-TIME</th>
<th>TO AVERAGE EXE-TIME</th>
<th>T1 WORST-CASE SLACK-TIME</th>
<th>T1 AVERAGE SLACK-TIME</th>
<th>WCETRB % SLACK TIME INCREASE VERSUS ACETRB</th>
<th>WCETRB % WCEP DECREASE VERSUS ACETRB</th>
<th>IDLE TIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>77913</td>
<td>85896</td>
<td>10963</td>
<td>681</td>
<td>6176</td>
<td>77928</td>
<td>90580</td>
<td>3.00%</td>
<td>9.64%</td>
</tr>
<tr>
<td>3</td>
<td>77913</td>
<td>88911</td>
<td>10962</td>
<td>739</td>
<td>6176</td>
<td>77928</td>
<td>88851</td>
<td>3.00%</td>
<td>9.64%</td>
</tr>
<tr>
<td>1</td>
<td>77911</td>
<td>82747</td>
<td>10962</td>
<td>854</td>
<td>6176</td>
<td>77927</td>
<td>88196</td>
<td>3.00%</td>
<td>9.64%</td>
</tr>
</tbody>
</table>

Table 51 – Average-Cases Per Worst-Case for SDAWS

<table>
<thead>
<tr>
<th>AVG CASES PER WORST</th>
<th>TO WORST-CASE SLACK-TIME</th>
<th>TO AVERAGE SLACK-TIME</th>
<th>TO AVERAGE EXE-TIME</th>
<th>TO AVERAGE</th>
<th>T1 WORST-CASE SLACK-TIME</th>
<th>T1 AVERAGE SLACK-TIME</th>
<th>IDLE TIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>75554</td>
<td>92056</td>
<td>12132</td>
<td>5718</td>
<td>5977</td>
<td>75569</td>
<td>90534</td>
</tr>
<tr>
<td>3</td>
<td>75555</td>
<td>90858</td>
<td>12133</td>
<td>6787</td>
<td>5977</td>
<td>75565</td>
<td>88505</td>
</tr>
<tr>
<td>1</td>
<td>75554</td>
<td>88156</td>
<td>12133</td>
<td>8568</td>
<td>5977</td>
<td>75565</td>
<td>85863</td>
</tr>
</tbody>
</table>

167
Worst-Case Processor Utilization Experiments

Table 52 – ACETRB SPMMT: Average and Worst-Case Task Slack Times for the Processor Utilization by Task Numbers Experiment

<table>
<thead>
<tr>
<th>SPMMT</th>
<th>NUMBER OF TASKS</th>
<th>T1 WORST-CASE SLACK-TIME</th>
<th>T1 AVERAGE SLACK-TIME</th>
<th>T2 WORST-CASE SLACK-TIME</th>
<th>T2 AVERAGE SLACK-TIME</th>
<th>T3 WORST-CASE SLACK-TIME</th>
<th>T3 AVERAGE SLACK-TIME</th>
<th>T4 WORST-CASE SLACK-TIME</th>
<th>T4 AVERAGE SLACK-TIME</th>
<th>T5 WORST-CASE SLACK-TIME</th>
<th>T5 AVERAGE SLACK-TIME</th>
<th>T6 WORST-CASE SLACK-TIME</th>
<th>T6 AVERAGE SLACK-TIME</th>
<th>T7 WORST-CASE SLACK-TIME</th>
<th>T7 AVERAGE SLACK-TIME</th>
<th>T8 WORST-CASE SLACK-TIME</th>
<th>T8 AVERAGE SLACK-TIME</th>
<th>T9 WORST-CASE SLACK-TIME</th>
<th>T9 AVERAGE SLACK-TIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACETRB 2 (24.4% WC PU)</td>
<td>75519</td>
<td>91715</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ACETRB 3 (36.1% WC PU)</td>
<td>63360</td>
<td>78638</td>
<td>63379</td>
<td>83022</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ACETRB 4 (48.3% WC PU)</td>
<td>63119</td>
<td>85040</td>
<td>51151</td>
<td>74852</td>
<td>51168</td>
<td>66741</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ACETRB 5 (78.1% WC PU)</td>
<td>20705</td>
<td>69244</td>
<td>20710</td>
<td>60831</td>
<td>20719</td>
<td>70212</td>
<td>20726</td>
<td>70446</td>
<td>20757</td>
<td>71194</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ACETRB 8 (97.5% WC PU)</td>
<td>2261</td>
<td>76688</td>
<td>63169</td>
<td>71238</td>
<td>51001</td>
<td>62054</td>
<td>38634</td>
<td>54031</td>
<td>20666</td>
<td>53324</td>
<td>14498</td>
<td>51376</td>
<td>2330</td>
<td>48000</td>
<td>2261</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 53 – WCETRB SPMMT: Average and Worst-Case Task Slack Times for the Processor Utilization by Task Numbers Experiment

<table>
<thead>
<tr>
<th>SPMMT</th>
<th>NUMBER OF TASKS</th>
<th>T1 WORST-CASE SLACK-TIME</th>
<th>T1 AVERAGE SLACK-TIME</th>
<th>T2 WORST-CASE SLACK-TIME</th>
<th>T2 AVERAGE SLACK-TIME</th>
<th>T3 WORST-CASE SLACK-TIME</th>
<th>T3 AVERAGE SLACK-TIME</th>
<th>T4 WORST-CASE SLACK-TIME</th>
<th>T4 AVERAGE SLACK-TIME</th>
<th>T5 WORST-CASE SLACK-TIME</th>
<th>T5 AVERAGE SLACK-TIME</th>
<th>T6 WORST-CASE SLACK-TIME</th>
<th>T6 AVERAGE SLACK-TIME</th>
<th>T7 WORST-CASE SLACK-TIME</th>
<th>T7 AVERAGE SLACK-TIME</th>
<th>T8 WORST-CASE SLACK-TIME</th>
<th>T8 AVERAGE SLACK-TIME</th>
<th>T9 WORST-CASE SLACK-TIME</th>
<th>T9 AVERAGE SLACK-TIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>WCETRB 2 (24.4% WC PU)</td>
<td>77926</td>
<td>90040</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WCETRB 3 (36.1% WC PU)</td>
<td>66871</td>
<td>82179</td>
<td>66885</td>
<td>78646</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WCETRB 4 (48.3% WC PU)</td>
<td>55822</td>
<td>77880</td>
<td>55849</td>
<td>58352</td>
<td>55851</td>
<td>77243</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WCETRB 5 (78.1% WC PU)</td>
<td>33724</td>
<td>64284</td>
<td>33734</td>
<td>71353</td>
<td>33737</td>
<td>64287</td>
<td>33743</td>
<td>75939</td>
<td>33772</td>
<td>72509</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WCETRB 8 (97.5% WC PU)</td>
<td>11627</td>
<td>66986</td>
<td>11627</td>
<td>61755</td>
<td>11629</td>
<td>69521</td>
<td>22053</td>
<td>63833</td>
<td>11655</td>
<td>53221</td>
<td>11659</td>
<td>50643</td>
<td>11693</td>
<td>61739</td>
<td>22625</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table 54 – ACETRB and WCETRB Comparison for Processor Utilizations, Task WCETs, and Task Periodicities for Aperiodic Disturbance (No Clear Trends – Only General Statements About Relative Performance Can Be Made)

<table>
<thead>
<tr>
<th>SPMMT</th>
<th>LARGE DISTORTION WITH TASK SIZE =</th>
<th>TO REPETITION PERIOD:</th>
<th>WORST-CASE PROC UTILIZATION VIA WCETs:</th>
<th>WORST-CASE PROC UTILIZATION VIA SLACK TIME:</th>
<th>TOTAL WCET TIME:</th>
<th>% ACET DECREASE VERSUS WCETRB</th>
<th>% AET DECREASE VERSUS WCETRB</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACETRB</td>
<td>APERIODIC BASE</td>
<td>300000</td>
<td>97.04%</td>
<td>97.68%</td>
<td>48702</td>
<td>0.21%</td>
<td>1075</td>
</tr>
<tr>
<td>ACETRB</td>
<td>APERIODIC BASE X2</td>
<td>600000</td>
<td>97.32%</td>
<td>97.56%</td>
<td>97318</td>
<td>0.41%</td>
<td>2439</td>
</tr>
<tr>
<td>ACETRB</td>
<td>APERIODIC BASE X5</td>
<td>1500000</td>
<td>97.26%</td>
<td>97.36%</td>
<td>243160</td>
<td>1.65%</td>
<td>6596</td>
</tr>
<tr>
<td>ACETRB</td>
<td>APERIODIC BASE X10</td>
<td>300000</td>
<td>96.84%</td>
<td>96.89%</td>
<td>484211</td>
<td>0.83%</td>
<td>15542</td>
</tr>
<tr>
<td>AETRB</td>
<td>APERIODIC BASE X20</td>
<td>300000</td>
<td>96.83%</td>
<td>96.80%</td>
<td>968594</td>
<td>0.41%</td>
<td>51427</td>
</tr>
<tr>
<td>SPMMT</td>
<td>SMALL DISTORTION WITH TASK SIZE =</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ACETRB</td>
<td>APERIODIC BASE</td>
<td>500000</td>
<td>65.37%</td>
<td>73.23%</td>
<td>32684</td>
<td>12.24%</td>
<td>13383</td>
</tr>
<tr>
<td>ACETRB</td>
<td>APERIODIC BASE X5</td>
<td>1500000</td>
<td>72.96%</td>
<td>73.65%</td>
<td>182392</td>
<td>2.13%</td>
<td>67967</td>
</tr>
<tr>
<td>ACETRB</td>
<td>APERIODIC BASE X10</td>
<td>300000</td>
<td>72.54%</td>
<td>58.53%</td>
<td>364658</td>
<td>1.10%</td>
<td>207342</td>
</tr>
<tr>
<td>SPMMT</td>
<td>LARGE DISTORTION WITH TASK SIZE =</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WCETRB</td>
<td>APERIODIC BASE</td>
<td>300000</td>
<td>88.82%</td>
<td>83.31%</td>
<td>44408</td>
<td>9.01%</td>
<td>8346</td>
</tr>
<tr>
<td>WCETRB</td>
<td>APERIODIC BASE X2</td>
<td>600000</td>
<td>88.73%</td>
<td>85.97%</td>
<td>88731</td>
<td>4.51%</td>
<td>14026</td>
</tr>
<tr>
<td>WCETRB</td>
<td>APERIODIC BASE X5</td>
<td>1500000</td>
<td>88.68%</td>
<td>92.70%</td>
<td>221708</td>
<td>1.89%</td>
<td>18092</td>
</tr>
<tr>
<td>WCETRB</td>
<td>APERIODIC BASE X10</td>
<td>300000</td>
<td>87.96%</td>
<td>87.54%</td>
<td>437347</td>
<td>0.51%</td>
<td>62291</td>
</tr>
<tr>
<td>SPMMT</td>
<td>SMALL DISTORTION WITH TASK SIZE =</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WCETRB</td>
<td>APERIODIC BASE</td>
<td>500000</td>
<td>87.49%</td>
<td>87.52%</td>
<td>874911</td>
<td>0.48%</td>
<td>124855</td>
</tr>
<tr>
<td>WCETRB</td>
<td>APERIODIC BASE X5</td>
<td>1500000</td>
<td>86.55%</td>
<td>86.55%</td>
<td>323518</td>
<td>1.20%</td>
<td>167272</td>
</tr>
</tbody>
</table>

Note: “Aperiodic Base” is equal to Task 0 WCET and Rep Period for Test 1 (These are what is varied)
Unclear Results: SPMST to WCET Ratio Experiment

Several of the tests performed for this thesis involved modifying the SPMST:WCET ratio by keeping SPMST constant while modifying task WCETs and task repetition periods. While this approach was successful in that it provided results which showed significant SDAWS slack time improvement over the ACETRB SPMMT and small SDAWS slack time detriment versus the WCETRB SPMMT (see table uuu below), it did not provide clear results. This is because changing task WCETs and task repetition periods affects more than the SPMST:WCET ratio. Changing these things affects worst-case processor utilization and RTOS overhead. Therefore, slack time improvements due to SDAWS are not a function merely of the SPMST:WCET ratio, but of several factors and cannot be graphed. Below is an example of one such unsuccessful test.

Table 55: Aperiodic Distortion Results – Different Task Sizes

<table>
<thead>
<tr>
<th>No SPM: WORST-CASE PROC UTILIZATION ESTIMATE</th>
<th>WORST-CASE PROC UTILIZATION VIA SLACK TIME</th>
<th>TOTAL WCET TIME</th>
<th>SPM SWAP TIME AS A PERCENTAGE OF WORST-CASE STACK-UP WCET</th>
<th>WORST-CASE TEST SLACK TIME</th>
<th>SDAWS % SLACK TIME INCREASE VERSUS ACETRB</th>
<th>SDAWS % SLACK TIME INCREASE VERSUS WCETRB</th>
<th>SDAWS % AET DECREASE VERSUS WCETRB</th>
<th>SDAWS % AET DECREASE VERSUS ACETRB</th>
</tr>
</thead>
<tbody>
<tr>
<td>97%</td>
<td>89.03%</td>
<td>46554</td>
<td>8.39%</td>
<td>5487</td>
<td>410.42%</td>
<td>-34.26%</td>
<td>2.18%</td>
<td>2.33%</td>
</tr>
<tr>
<td>97%</td>
<td>90.81%</td>
<td>93023</td>
<td>4.30%</td>
<td>9190</td>
<td>276.79%</td>
<td>-34.46%</td>
<td>7.24%</td>
<td>2.57%</td>
</tr>
<tr>
<td>97%</td>
<td>94.10%</td>
<td>243152</td>
<td>1.64%</td>
<td>147435</td>
<td>123.54%</td>
<td>-18.32%</td>
<td>0.64%</td>
<td>0.23%</td>
</tr>
<tr>
<td>97%</td>
<td>91.45%</td>
<td>460857</td>
<td>0.87%</td>
<td>42023</td>
<td>173.89%</td>
<td>-31.58%</td>
<td>2.54%</td>
<td>2.25%</td>
</tr>
<tr>
<td>97%</td>
<td>90.65%</td>
<td>874912</td>
<td>0.46%</td>
<td>93548</td>
<td>197.67%</td>
<td>-25.07%</td>
<td>10.45%</td>
<td>5.55%</td>
</tr>
</tbody>
</table>
Table 56 – SDAWS SPMMT Task Data Set: SPM Swap Times for 97% WC Processor Utilization via Aperiodic Disturbance

<table>
<thead>
<tr>
<th>SPMT</th>
<th>MEMORY SWAP NUMBER OF BYTES</th>
<th>MEMORY SWAP TIME</th>
<th>T2 WORST CASE SLACK-TIME</th>
<th>T1 WORST CASE SLACK-TIME</th>
<th>TO WORST CASE EXE-TIME</th>
<th>TO AVERAGE SLACK-TIME</th>
<th>TO AVERAGE EXE-TIME</th>
<th>T1 WORST CASE ELAPSED PERIOD</th>
<th>T1 REPETITIVE PERIOD</th>
<th>T1 AVERAGE SLACK-TIME</th>
<th>T1 AVERAGE EXE-TIME</th>
<th>T1 WORST CASE ELAPSED PERIOD</th>
<th>T2 WORST CASE SLACK-TIME</th>
<th>T2 AVERAGE SLACK-TIME</th>
<th>T2 AVERAGE EXE-TIME</th>
<th>T2 AVERAGE ELAPSED PERIOD</th>
<th>CPU IDLE TIME</th>
<th>AVG TASK EXE TIME</th>
<th>TOTAL TASK EXE TIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>SDAWS</td>
<td>46</td>
<td>789</td>
<td>4387</td>
<td>169124</td>
<td>77274</td>
<td>845286</td>
<td>221592</td>
<td>153259</td>
<td>119417</td>
<td>30000</td>
<td>169134</td>
<td>390522</td>
<td>121732</td>
<td>82532</td>
<td>49349</td>
<td>428671</td>
<td>386109</td>
<td>6294967</td>
<td>1122080</td>
</tr>
<tr>
<td>SDAWS</td>
<td>123</td>
<td>2792</td>
<td>41555</td>
<td>165159</td>
<td>761212</td>
<td>844593</td>
<td>221709</td>
<td>153436</td>
<td>119000</td>
<td>50000</td>
<td>165199</td>
<td>380264</td>
<td>121594</td>
<td>81348</td>
<td>49251</td>
<td>415555</td>
<td>319108</td>
<td>6291035</td>
<td>1122779</td>
</tr>
<tr>
<td>SDAWS</td>
<td>345</td>
<td>4029</td>
<td>37770</td>
<td>144479</td>
<td>754968</td>
<td>761690</td>
<td>221450</td>
<td>153280</td>
<td>119721</td>
<td>50000</td>
<td>144079</td>
<td>357182</td>
<td>121565</td>
<td>81348</td>
<td>492736</td>
<td>377700</td>
<td>6567969</td>
<td>1139510</td>
<td>3601700</td>
</tr>
<tr>
<td>SDAWS</td>
<td>480</td>
<td>7670</td>
<td>10946</td>
<td>159441</td>
<td>770370</td>
<td>836996</td>
<td>221625</td>
<td>153380</td>
<td>119250</td>
<td>50000</td>
<td>159441</td>
<td>382360</td>
<td>116024</td>
<td>83348</td>
<td>59905</td>
<td>100464</td>
<td>289432</td>
<td>5992723</td>
<td>1130979</td>
</tr>
<tr>
<td>SDAWS</td>
<td>1102</td>
<td>29304</td>
<td>85555</td>
<td>238172</td>
<td>479451</td>
<td>726462</td>
<td>249062</td>
<td>160534</td>
<td>119281</td>
<td>50000</td>
<td>238172</td>
<td>364974</td>
<td>110813</td>
<td>82241</td>
<td>56527</td>
<td>855555</td>
<td>301394</td>
<td>5802421</td>
<td>2094009</td>
</tr>
<tr>
<td>SDAWS</td>
<td>2465</td>
<td>39043</td>
<td>129117</td>
<td>228413</td>
<td>604853</td>
<td>666859</td>
<td>221648</td>
<td>139556</td>
<td>98252</td>
<td>50000</td>
<td>228433</td>
<td>347616</td>
<td>121538</td>
<td>82890</td>
<td>56298</td>
<td>239117</td>
<td>222653</td>
<td>5075666</td>
<td>2348714</td>
</tr>
</tbody>
</table>
Table 57 – ACETRB, WCETRB, SDAWS SPMMTs – Task Data Set: Desired Deadline Comparison For 97% WC Processor Utilization via the Aperiodic Disturbance Test (1:12 SPMMT:WCET Ratio)

<table>
<thead>
<tr>
<th>NUMBER OF TASKS</th>
<th>ACETRB</th>
<th>WCETRB</th>
<th>SDAWS</th>
<th>DESIRED DEADLINE %</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>WCETRB</td>
<td>WCETRB</td>
<td>WCETRB</td>
<td>WCETRB</td>
</tr>
<tr>
<td>5</td>
<td>944039</td>
<td>31247</td>
<td>67808</td>
<td>123666</td>
</tr>
<tr>
<td>5</td>
<td>41772</td>
<td>247651</td>
<td>381915</td>
<td>119046</td>
</tr>
<tr>
<td>5</td>
<td>41772</td>
<td>247651</td>
<td>381915</td>
<td>119046</td>
</tr>
<tr>
<td>5</td>
<td>263153</td>
<td>33222</td>
<td>304411</td>
<td>119046</td>
</tr>
<tr>
<td>5</td>
<td>263153</td>
<td>33222</td>
<td>304411</td>
<td>119046</td>
</tr>
<tr>
<td>5</td>
<td>263153</td>
<td>33222</td>
<td>304411</td>
<td>119046</td>
</tr>
<tr>
<td>5</td>
<td>263153</td>
<td>33222</td>
<td>304411</td>
<td>119046</td>
</tr>
</tbody>
</table>
APPENDIX B

The RTOS and SPMMT simulator were developed for a Freescale Semiconductor MCF5225 Tower Module Development Kit. In particular, the microcontroller on the development module was a 32-bit Freescale Coldfire version MCF52259CAG80.

SOURCE COMPILATION AND TOOLS

The source code for the RTOS and SPMMT simulator is integrated into one body of code available on the CD provided with this thesis. Therefore, it is not possible to compile them separately.

Further, in order to use any development tools with the MCF5225 board, a firmware update may be required. The P&E Microcomputer Systems USB debugger interface is used on this board. The P&E firmware update tool can be found within the contents of this thesis’ CD.

The P&E Firmware Updater can also be found on P&E’s website:  
http://www.pemicro.com/blog/post.cfm/osbdm-firmware-update-utility

All tools used for development, compilation, and debugging are available from Freescale Semiconductor. Development, compilation, and debugging tools are incorporated into the Codewarrior 10.1 SDK. 30-Day Trial “Evaluation” licenses and unlimited-time “Special Edition” licenses are available. The “Evaluation” licenses offer full access to development features, while the “special edition” licenses place code-size and feature restrictions. This thesis was developed using a “special edition” license.

Codewarrior 10.1 is available here:  

BRIEF USERS GUIDES

IMCRTOS Users Guide

The RTOS developed for this thesis does not exist as a black-box application. Therefore, in order to use the IMCRTOS, one must write functions and link them into RTOS tasks. Though the RTOS offers ACETRB / WCETRB swapping according schedulability analysis, ACETRB and WCETRB SPMMTs are simulated (in the simulation function execute_tailor_task()) and therefore offer no real performance benefits to a generic system. The schedulability analysis and SPM swapping could be extended if real ACETRB / WCETRB allocations were created.

The IMCRTOS does offer priority-based scheduling, automatic task rescheduling, and automatic timing analysis routines that may be used by any generic system. To create a task, one must first modify parameters within the RTOS’ priority task list. The task list is ordered such that the tasks with index 0 have the lowest priority and the tasks with the highest index have the highest priority. Up to 255 priority levels are available in the priority task list.
The file *RTSPMM_TestBench.c* contains examples of task parameters that may be specified. A majority of these task parameters are utilized only by the SPMMT simulation function and therefore are irrelevant for general RTOS use. The following code snippets indicate the task parameters that must be specified in order to use the RTOS independently from the simulator:

- `priorityTaskList[i].TASK_FUNCTION = INSERT_YOUR_TASK_HERE;`
- `priorityTaskList[i].REPETITION_PERIOD = YOUR_REP_PERIOD_IN_US;`
- `priorityTaskList[i].DCT_INTERVAL = YOUR_TASK_DEADLINE_IN_US;`

So long as the task function is not set to `execute_tailor_task`, ACETRB and WCETRB SPMMT simulation is disabled. To disable the SDAWS SPMMT, the following function must be called:

- `SET_SWAPPING_ENABLED(FALSE);`

Calling this function will disable additional RTOS overhead due to SDAWS SPM swapping and point-schedulability analyses.

Finally, a series of functions must be called in order to initialize the RTOS, run tasks, and obtain timing statistics. The functions are shown below:

- `Initialize_oC_RTOS();` - Initializes RTOS Parameters
- `InitializeTimer0();` - Initializes RTOS Timer
- `EnableThisManyTasks(n);` - Enables task with priority 0 through task with priority n-1.
- `TaskManager();` - Calls the RTOS kernel. Place inside of an infinite loop to run indefinitely.
- `printAllTaskStats();` - Prints task timing statistics obtained since the timer was initialized.

**SPMMT Simulator Users Guide**

Though special care must be taken to run the RTOS without running the simulator, running the simulator is easy. This is because the code provided in this thesis” CD is already configured to perform SPMMT simulation. To run tests, one must first modify the desired parameters in *RTSPMM_TestBench.h*. This file contains #defined parameter lists for several different task priorities. The file *RTSPMM_TestBench.c* contains functions which copy the #defined values into the priority task list.

- `initTaskAtThisPriority_toT1(task priority i)`
- `initTaskAtThisPriority_toT2(task priority i)`
- `initTaskAtThisPriority_toT3(task priority i)`
- ...

174
Each of these commands sets the indicated task priority index to the modified #defined sets in `RTSPMM_TestBench.h` (e.g. T1, T2, T3, etc).

Important to note are the following commands:

- `priorityTaskList[i].TASK_FUNCTION = execute_tailor_task;`
- `tailor_task_init(&priorityTaskList[task priority]);`

`execute_tailor_task` is the simulator function. This function simulates WCETRB, ACETRB, and SDAWS SPMMT performance depending on the SPMMT mode of the simulator. The initialization function must be called in order to initialize simulated memory spaces. The SPMMT mode is changed using the following lines of code:

To load the WCETRB SPMMT

- `SET_SWAPPING_ENABLED(FALSE);`
- `SPM_MODE = WCET`

To load the ACETRB SPMMT

- `SET_SWAPPING_ENABLED(FALSE);`
- `SPM_MODE = ACET`

To load the SDAWS SPMMT

- `SET_SWAPPING_ENABLED(TRUE);`
- `SPM_MODE = ACET`

Finally, the following functions are used to set up the SDRAM load time simulation (according to SRAM accesses per SDRAM access), and cache simulation. This thesis has the following parameters set to the values indicated for all experiments run, except for the ones that test the effect of SDRAM memory access times.

- `SET_SD_TO_SRAM_ACCESS_RATIO(200);`
- `SET_cache_SIMULATION(TRUE);`