Abstract—In hard real-time systems, each task has to provably finish its execution within its respective deadline. Compiler optimizations can be used to improve each task’s timing behavior. However, current compilers do not consider tasks’ deadlines and can therefore not be used to reliably optimize hard real-time systems with regard to its schedulability. We propose a compiler optimization framework based on Integer-Linear Programming which allows for schedulability aware code optimizations of hard real-time multitasking systems. We evaluate the framework using an instruction scratchpad optimization. The results show that our approach can be used to improve the schedulability of hard real-time systems significantly.

Index Terms—WCRT optimization; compiler; schedulability; Scratchpad memory;

I. INTRODUCTION

In hard real-time multitasking systems, each task must provably finish its execution before a given deadline. A system is called schedulable, if its tasks can be executed in a way such that no timing constraints are violated. Besides the scheduling algorithm, each task’s Worst-Case Execution Time (WCET) is crucial for determining whether a system is schedulable or not. The WCET describes the maximum execution time of the task if it is not interrupted by any other task.

To illustrate the importance of precise WCET optimizations in hard real-time multitasking environments, assume the system given in Table I. This system consists of 3 different

<table>
<thead>
<tr>
<th>Task</th>
<th>c</th>
<th>d</th>
<th>T</th>
<th>r</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>3</td>
<td>20</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>8</td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>20</td>
<td>20</td>
<td>23</td>
</tr>
</tbody>
</table>

tasks \(\tau_i\), each defined by its WCET \(c_i\), its deadline \(d_i\) and its execution period \(T_i\). To achieve schedulability, tasks are scheduled according to either fixed or dynamically assigned priorities. For the example, let us assume Deadline Monotonic Scheduling (DMS), i.e., fixed-priority preemptive scheduling where the task with the tightest deadline is assigned the highest priority.

The Worst-Case Response Time (WCRT) is defined as the maximum possible time interval between the activation of a task and its end, including timing penalties inflicted by preemptions and blockings by other tasks with higher priorities. Given these information, the WCRT \(r_i\) of each task \(\tau_i\) in the example set can be calculated using equation (1) which was proposed in [1]. This iterative formula assumes that the numerically lowest task (i.e., \(\tau_0\)) has the highest priority. If \(r_i\) does not increase between two subsequent calculations and \(r_1 \leq d_i\), then the formula converges and the task will meet its deadline. If \(r_i > d_i\), the calculation is aborted and the system is not schedulable.

\[
r_i = c_i + \sum_{j=0}^{i-1} \left[ \frac{r_j}{T_j} \right] \cdot c_j
\]

The WCRT \(r_i\) for each task is listed in Table I. In this example, \(\tau_2\) will miss its deadline by 3 time units (t.u). Sensitivity analysis techniques may be used to determine which property of which task should be changed to achieve schedulability. However, relaxing constraints like deadlines or periods might not be feasible for a real system, as those properties are usually predetermined by the physical environment. Therefore, usually the only way to guarantee a valid real-time behavior of the system is to decrease one or several tasks’ WCET. This may be achieved by removing functionality, increasing the computational power or by applying compiler optimizations. Again, removing functionality might not be possible within a real system. Increasing the computational power might lead to a schedulable system but will scale all tasks’ execution times in equal measure, usually leading to an over-dimensioned design. This will most likely inflict unnecessarily high power consumption and production costs. Finally, compiler optimizations are able to manipulate the execution time of each task without the need for tampering with the system’s real-time constraints or modifying the underlying hardware. Unfortunately, current compilers like LLVM or GCC are not aware of the worst-case execution behavior of a program. Scientific WCET-oriented compilers exist [2], however until now they focused on reducing each task’s WCET individually, without paying attention to scheduling effects. For the example in Table I, let us assume the hardware offers the resources to decrease the WCET of one of the three tasks by 2 t.u. The compiler might decide to optimize the highest priority task \(\tau_0\), thus reducing the WCET of all lower priority tasks. This will lead to a WCET \(c_0 = 0\). In spite of this overly optimistic

This work was partially supported by Deutsche Forschungsgemeinschaft (DFG) under grant FA 1017/1-2.
assumption, eq. (1) resolves to \( r_2 = 21 \) t.u., so \( r_2 \) will still miss its deadline by 1 t.u. As another simple heuristic, the compiler could optimize \( r_2 \) itself, leading to an optimized WCET \( c_{2,\text{opt}} = 3 \) t.u. In this scenario, \( r_2 \) will be at 21 t.u., thus again missing its deadline. In this example, optimizing \( \tau_1 \) will lead to a schedulable system. With \( c_{1,\text{opt}} = 6 \) t.u., the WCRT of task \( r_2 \) will be reduced to \( r_2 = 19 \) t.u., leading to a schedulable system.

In this paper, we propose a framework which allows for the targeted WCET optimization of a given task set in order to achieve schedulability. The framework is based on Integer-Linear Programming (ILP). We show the impact of our framework by proposing an exemplary schedulability-oriented instruction scratchpad memory (SPM) optimization. The key contributions of this paper are:

- We propose a framework for the schedulability-aware WCET-optimization of multitasking systems.
- Our approach can be applied to fixed priority systems with deadlines lower than or equal to the task’s period and to implicit-deadline systems with dynamic priorities.
- We show the capabilities of our framework by integrating a schedulability-aware instruction SPM allocation.
- Our results show that the framework allows for significant improvements on the schedulability of hard real-time systems.

This paper is organized as follows: Section II gives an overview of related work. Section III explains the notational conventions used throughout the paper. Section IV discusses the underlying fundamentals of our framework. Section V describes our proposed framework. Section VI describes further additions to the ILP formulation to improve the ILP’s precision for our target platform. In section VII, we show and discuss our evaluation results. This paper closes with a conclusion and a brief discussion of future challenges.

### II. RELATED WORK

Response time analysis has been intensively researched over the last decades. Starting with Liu and Layland in 1973 [3] who covered periodic systems, many researchers worked on tight and precise predictions of worst-case timing behaviors for multitasking systems. Nowadays, response time analysis techniques can analyze complex event-triggered systems and give tight and safe WCRT estimates without having to rely on one single minimal period [4], [5], [6]. To be able to generate those results, they rely on the WCET of each task. One of the best known tools for WCET analysis in both industry and research is AbsInt aiT [7]. It guarantees safe bounds of a task’s WCET and offers support for a variety of real microcontrollers. Beyond tools like aiT, current research focuses on analyzing the timing penalties which are inflicted by context switches, e.g., additional delays due to preemption related cache evictions [8], [9]. Combining these approaches and tools, they allow for the precise WCRT analysis of complex multitasking real-time systems. However, while offering precise analysis results even for complex systems, these approaches do not provide any solutions if the analysis shows that a system will violate its timing constraints.

Sensitivity analysis as proposed in [10], [11], [12] and [13] tries to find the system parameters which may be changed in order to achieve a schedulable system. However, these approaches work on a high abstraction level. They consider the running tasks as black boxes with given parameters like WCET, period and deadline. Due to the high abstraction level, those approaches are not able to determine whether the proposed changes are technically feasible. The analysis cannot guarantee that the proposed WCET improvements can be realized on the underlying hardware. To avoid this problem, sensitivity analysis mostly focuses on changing timing parameters such as deadlines and periods to achieve schedulability, which may not be allowed due to physical constraints.

Over the past years, several approaches have been proposed to reduce the WCET of hard real-time tasks using compiler optimizations. Suhendra et al. propose an optimization framework for single-tasking systems based on Integer-Linear Programming [14]. Based on this framework, an SPM allocation for the Infineon TriCore microcontroller was built by [15] and integrated into a WCET-aware compiler framework [2]. These approaches try to minimize the WCET of a single-tasking system, but do not take the tasks’ scheduling characteristics like periods or deadlines into account. They can therefore not be used to reliably improve the schedulability of a multitasking system. An ILP-based approach which builds up on [14] was proposed in [16] to perform a schedulability-aware optimization of periodic hard real-time multitasking systems. However, this approach needs to solve the proposed ILP twice for fixed-priority systems, resulting in computational overhead and suboptimal results. In this paper, we improve on [16]. Our proposed approach only needs to solve the ILP once and removes the need risk of suboptimal results due to an overapproximated number of preemptions. Additionally, we improved the precision of the previously proposed SPM optimization by modeling the Infineon TriCore’s branch prediction in the ILP model.

### III. NOTATIONAL CONVENTIONS

Table II shows the abbreviations which are used in the equations throughout this paper. As a note, uppercase letters describe values which are calculated outside the ILP and are added to the ILP as constant factors. Lowercase letters denote ILP variables which are calculated by the ILP solver. As an exception, the deadline \( d_i \) and the WCET \( c_i \) of a task \( \tau_i \) are written in small letters to comply with common notational practice.

Additionally, we made the following assumptions:

- For this work, we focus on periodic independent tasks.
- For fixed-priority systems, a task with a lower index has a higher priority, i.e., \( \tau_0 \) is the task with the highest priority.
- The scheduler is preemptive.
- Preemption penalties are not modeled explicitly. They may be modeled as additional tasks with a high priority or may be added to a task’s WCET.
- Each task’s deadline is lower than or equal to its period.
- Using ILP implies that all variables are expressed as integers. This is achieved without loss of generality by using
TABLE II  
NOMENCLATURE USED WITHIN THIS PAPER.

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$i$</td>
<td>Index of a task.</td>
</tr>
<tr>
<td>$c_i$</td>
<td>WCET of task $i$.</td>
</tr>
<tr>
<td>$C_A$</td>
<td>net WCET of basic block $A$.</td>
</tr>
<tr>
<td>$C_{A,flash}$</td>
<td>net WCET of basic block $A$ when it is assigned to flash memory.</td>
</tr>
<tr>
<td>$C_{A,SPM}$</td>
<td>net WCET of basic block $A$ when it is assigned to SPM.</td>
</tr>
<tr>
<td>$C_i$</td>
<td>CPU cycles which are needed for a jump instruction</td>
</tr>
<tr>
<td>$d_i$</td>
<td>Deadline of task $i$.</td>
</tr>
<tr>
<td>$F_L$</td>
<td>Maximum number of iterations of loop $L$.</td>
</tr>
<tr>
<td>$G_A$</td>
<td>Timing gain if basic block $A$ is moved to SPM.</td>
</tr>
<tr>
<td>$J_A$</td>
<td>Binary ILP decision variable set to 1 if an additional jump is needed.</td>
</tr>
<tr>
<td>$r_i$</td>
<td>WCET of task $i$.</td>
</tr>
<tr>
<td>$T_i$</td>
<td>Minimal period between two releases of task $i$.</td>
</tr>
<tr>
<td>$\tau_i$</td>
<td>Abbreviation for task $i$.</td>
</tr>
<tr>
<td>$u$</td>
<td>Overall load of the system.</td>
</tr>
<tr>
<td>$w_A$</td>
<td>ILP variable which holds the accumulated execution time of basic block $A$ and all its successors.</td>
</tr>
<tr>
<td>$S_A$</td>
<td>Net Size of basic block $A$.</td>
</tr>
<tr>
<td>$s_A$</td>
<td>ILP Variable for the size of basic block $A$.</td>
</tr>
<tr>
<td>$S_L$</td>
<td>Size of a long jump instruction.</td>
</tr>
<tr>
<td>$S_{SPM}$</td>
<td>Size of the SPM.</td>
</tr>
<tr>
<td>$x_A$</td>
<td>Binary ILP variable set to 1 if basic block $A$ is assigned to SPM.</td>
</tr>
</tbody>
</table>

the CPU’s clock as time unit. As a result, all times within this paper are expressed as number of CPU clock cycles unless stated otherwise.

- Flow facts, which describe the maximum iteration count for loops, are already known.

IV. BACKGROUND

A. ILP Formulation

In [14], Suhendra et al. propose a method to use ILP to optimize the WCET of a single-tasking program. This paragraph will give a brief overview of this approach.

The approach operates on an acyclic Control Flow Graph (CFG) of the program which is to be optimized. A program is divided into disjoint basic blocks (BBs). A basic block is defined as a sequence of instructions that is executed from its top to its end. This means that a basic block may only contain any conditional or unconditional jump instructions as its last instruction. As an example, consider the CFG given in Fig. 1. The task’s WCET is modeled as a set of inequalities. The WCET $w_A$ of a BB $A$ is defined as the maximum execution time starting at BB $A$ until the program finishes. $w_A$ is therefore defined to be greater-equal its own execution time $C_A$ plus the WCET of its directly succeeding basic blocks. For this example, the overall WCET of the program would equal the accumulated execution time $w_A$ of block $A$:

$$w_A \geq C_A + w_B \quad (2)$$

$$w_B \geq C_B + w_C \quad (3)$$

$$w_B \geq C_B + w_D \quad (4)$$

Blocks $D$ and $F$ form a loop. To get the acyclic CFG which is needed by the approach, a meta block $L$ is defined to hold the execution time of the loop:

$$w_D \geq C_D + w_F \quad (6)$$

$$w_L = 10 \cdot w_D \quad (7)$$

$w_L$ holds the summed execution time over all 10 loop iterations. We can then write

$$w_B \geq C_B + w_L \quad (8)$$

and finally add the remaining constraints:

$$w_D \geq C_D + w_E \quad (9)$$

$$w_E \geq C_E \quad (10)$$

$$w_F \geq C_F \quad (11)$$

These constraints fully model the program’s CFG as an ILP. To find the the tightest bound on the program’s estimated WCET, the ILP’s objective function is set to minimize the accumulated execution time of the task’s entry block:

$$\min (w_A) \quad (12)$$

After solving the ILP, $w_A$ will hold a valid and tight upper bound on the task’s WCET.

B. SPM allocation

The approach by Suhendra et al. was used by [15] to perform an SPM allocation for the Infineon TriCore microcontroller. This paragraph briefly illustrates the modifications of the ILP by [14] to implement the SPM optimization. Constraints which are not essential for an understanding of the principal behavior will be skipped due to the limited space.

To model the gain by moving a BB to the SPM, a WCET analysis was performed first using AbsInt aiT with all blocks assigned to flash memory. The resulting timing for a block $A$ is called $C_{A,flash}$. The SPM was then virtually increased.
such that the whole program could be assigned to the SPM, resulting in $C_{A,SPM}$ for the time of block $A$ after a second WCET analysis. The gain $G_A$ is then easily calculated by

$$G_A = C_{A,flash} - C_{A,SPM}$$

The accumulated execution time of block $A$ can now be rewritten to

$$w_A \geq C_{A,flash} - x_A \cdot G_A + w_B$$

with $C_j$ being a constant denoting a safe over-approximation for the additional execution delay for the long jump. The needed size of BB $A$ can be modeled by:

$$s_A \geq S_A + j_A \cdot S_j$$

$S_A$ is the net size that block $A$ will need in each case. $S_j$ is a constant that denotes the additional size for a long jump. The sizes of all other blocks within the program are modeled accordingly. Another ILP constraint can now be added to account for the limited SPM size in the real system. We define $S_{SPM}$ to be the overall size of the SPM:

$$S_{SPM} \geq x_A \cdot s_A + x_B \cdot s_B + \cdots$$

Although the multiplication terms do not appear to be linear, multiplying an integer variable with a binary variable can be modeled as a Boolean AND operation and can be expressed in a linear way [17].

With these constraints, the ILP solver will assign those blocks to the SPM which will minimize the WCET of the task the most with regard to the overall size of the SPM.

V. MULTITASKING SUPPORT

The previously outlined approach is not able to optimize multitasking systems with regard to the system’s schedulability. Our approach therefore defines that each basic block is executed in the “context” of a given task. The ILP formulation described in Section IV is generated for each task. The $w$ variable associated with the entry block of each task $\tau_i$ describes a safe upper bound of the task’s WCET. For BBs which are shared between several tasks, the accumulated execution time $w$ and net execution time $c$ may vary depending on the task which executes the corresponding code block. If block $A$ is called by task $\tau_0$, we define $w_{A,\tau_0}$ as $A$’s accumulated worst-case execution time in the context of task $\tau_0$ and $c_{A,\tau_0}$ as its net execution time, respectively. This way, shared code (e. g., common math functions) can be modeled more precisely, as its WCET may vary depending on the block’s context.

When trying to adapt ILP-based optimizations to multitasking environments, modeling the WCET of each task alone is not sufficient. Instead, the system’s scheduling must be modeled within the ILP to ensure that each task will always finish its execution before its respective deadline.

A. Dynamic Priorities

For systems scheduled under dynamic priorities, our approach is currently limited to implicit-deadline systems where each task’s deadline equals its period. We illustrate our approach using the optimal scheduling algorithm Earliest Deadline First (EDF). When using EDF, the task with the earliest absolute deadline has the highest priority. If a new task with an even earlier deadline is triggered, priorities will change dynamically. Liu and Layland showed in [3] that a system which is scheduled under EDF is schedulable if and only if the system load $u$ is less than or equal to 1:

$$u = \sum \frac{c_i}{T_i} \leq 1$$

$c_i$ denotes the WCET of task $\tau_i$, and $T_i$ is its corresponding period. As shown in [16], the use of EDF for the ILP approach presented in section IV implies that changing the ILP’s objective function to $\min u$ and therefore minimizing the overall system load is sufficient to optimize the task set with regard to the system’s schedulability.

Alternatively, eq. (18) may be added as an inequation to the ILP allowing for an arbitrary objective function. Hence, any ILP-based compiler optimization may easily be adapted to a multitasking system which is scheduled under EDF.

B. Fixed-Priority Scheduling

When using fixed priorities, eq. (18) is still a necessary constraint, but not sufficient. A system that uses fixed priorities might not be schedulable even if its system load is lower than or equal to 1. Additionally, minimizing the system load as proposed for dynamic priorities will not necessarily lead to the best solution for fixed-priority scheduling.

For example, let us assume the system given in Table III with each task’s deadline being equal to its period. The system was optimized for EDF by putting specific basic blocks of each task into the SPM. $c_{EDF}$ denote the resulting optimized WCETs for each task. With eq. (18), this leads to a system load $u = 0.9667$ and therefore, the system would be schedulable under EDF. However, when calculating the response time $r_1$ for rate-monotonic scheduling (RMS) using equation (1) with the EDF-optimized task set, the WCRT of task $\tau_1$ equals to $r_1 = 36$, missing the deadline. If the ILP optimization however would assign different parts of the task set to the SPM, leading to the WCETs in column $c_{RMS}$, the system load will increase to $u = 1$. However, the system will now be schedulable using RMS. We propose the following approach for fixed-priority systems: The approach was previously outlined in [16] in a rudimentary way. Although using the same underlying approach, [16] could not precisely model the number of
preemptions of a task by higher priority tasks in the ILP formulation. This leads to the necessity of solving the formulated ILP twice and to the possibility of still over-approximating the number of preemptions. In contrast, the approach presented in the following models the response time analysis shown in equation (1) for periodic tasks directly within the ILP, removing both the need of multiple ILP runs and the possible over-approximation.

When equation (1) is expressed as an ILP, each task’s WCRT $r_i$ has to be modeled as an integer variable. Additionally, because this approach aims at optimizing the program code to achieve schedulability, each task’s WCET $c_i$ has to be modeled as a variable as well. However, this leads to the problem that the term $\left\lceil \frac{r_i}{T_j} \right\rceil \cdot c_j$ from equation (1) introduces quadratic complexity to the ILP. Therefore, the equations for a task’s WCRT cannot be included directly in the ILP.

To solve this problem, an integer variable $x_{i,j}$ is added to the ILP for any combination of lower priority task $\tau_i$ and higher priority task $\tau_j$. $x_{i,j}$ denotes the timing penalty which is added to the WCRT of $\tau_i$ due to preemptions by $\tau_j$. The response time of $\tau_i$ can then be added as an equality to the ILP:

$$r_i = c_i + \sum_{j=0}^{i-1} x_{i,j}$$

If $r_i$ is lower than or equal to the higher priority task’s period $T_j$, $\tau_i$ can be preempted at most one time by $\tau_j$, thus resulting in $x_{i,j} = 1 \cdot c_i$. If $r_i$ is greater than $T_j$ but lower than or equal to $2 \cdot T_j$, the term $\left\lceil \frac{r_i}{T_j} \right\rceil$ from equation (1) leads to $x_{i,j} = 2 \cdot c_i$. The maximum allowed number of preemptions is limited by the WCRT $r_i$, and therefore upper-bounded by the corresponding deadline $d_i$. $r_i$ can be bounded in the ILP by adding the constraint

$$r_i \leq d_i$$

If $r_i$ may never be greater than $d_i$ in a feasible system, the maximum allowed number of preemptions is $\left\lceil \frac{d_i}{T_j} \right\rceil$. This leads to the following formulation of the preemption penalty $x_{i,j}$:

$$x_{i,j} \geq \begin{cases} 1 \cdot c_j & \text{if } r_i \geq 0 \cdot T_j \\ 2 \cdot c_j & \text{if } r_i \geq 1 \cdot T_j \\ \vdots \\ \left\lceil \frac{d_i}{T_j} \right\rceil \cdot c_j & \text{if } r_i \geq \left( \left\lceil \frac{d_i}{T_j} \right\rceil - 1 \right) \cdot T_j \end{cases}$$

Due to the upper bound on $r_i$, these equations will give a safe upper bound on the preemption penalties.

**Proof.** The following three cases may occur:

$$r_i \leq \left( \left\lceil \frac{d_i}{T_j} \right\rceil - 1 \right) \cdot T_j$$

$$r_i = \left\lceil \frac{d_i}{T_j} \right\rceil \cdot T_j$$

$$r_i > \left\lceil \frac{d_i}{T_j} \right\rceil \cdot T_j$$

If eq. (22) holds, the preemption penalty is given correctly by equation (21), as it is simply a rewritten form of the term $\left\lceil \frac{r_i}{T_j} \right\rceil$ from equation (1).

Because obviously the period $T_j$ must be greater than 0, eq. (23) can be rewritten as

$$r_i \leq \frac{d_i}{T_j}$$

By definition of the ceiling function, this leads to

$$\frac{d_i}{T_j} \leq r_i \leq \frac{d_i}{T_j} + 1$$

If $r_i > d_i$, then inequation (20) will be violated, therefore $r_i$ must be equal to $d_i$, leading to

$$\left\lceil \frac{r_i}{T_j} \right\rceil \cdot c_j = \left\lceil \frac{d_i}{T_j} \right\rceil \cdot c_j \equiv x_{i,j}$$

Finally, in case of eq. (24), the WCRT $r_i$ will always be greater than $d_i$, violating inequation (20). As a result, if the set of inequations holds, $x_{i,j}$ will equal the preemption penalty given by equation (1). If a task’s WCRT $r_i$ is greater than its deadline $d_i$, the inequation system will be infeasible.

Equation (21) can be reformulated expressing that if $r_i$ is preempted by $\tau_j$ at least $N$ times, then $x_{i,j} \geq (N + 1) \cdot c_j$ must hold, too. Conditional constraints of this form can be described as a set of ILP inequations by transforming them into a logical OR, as shown in [17]. The number of preemptions $p_{i,j}$ of $\tau_i$ by $\tau_j$ can be constrained by

$$p_{i,j} \geq \frac{1}{T_j} \cdot r_i$$

Because the ILP solver tries to achieve a schedulable system, it implicitly sets $p_{i,j}$ to the smallest value which fulfills inequation (28). In result, for each $N$, $0 \leq N \leq \left\lfloor \frac{d_i}{T_j} \right\rfloor - 1$, two conditional constraints are added to the ILP:

$$p_{i,j} + L_{i,j} \cdot b \leq N$$

$$x_{i,j} + \sum_{b=0}^{b=N} \sum_{j=0}^{p_{i,j}} (1-b) \geq N \cdot c_j$$

$b$ is an auxiliary binary ILP decision variable. Eq. (29) is used to determine whether a specific case as introduced in eq. (21) must hold or not. If $p_{i,j}$ resolves to be greater than or equal to $N$, then $b$ is forced to 1, to avoid a violation of the inequation. $L_{i,j}$ is a constant so large that $L_{i,j} \cdot b$ will lead to inequation (29) being always fulfilled if $b = 1$. Because the maximum allowed number of preemptions is limited and occurs for $r_i = d_i$, a safe lower bound for $L$ is given by

$$L_{i,j} = \left\lceil \frac{d_i}{T_j} \right\rceil + 1$$

---

### Table III

**Exemplary task-set using RMS scheduling, with EDF and RMS optimized WCETs. Times are in fictive time units.**

<table>
<thead>
<tr>
<th>Task $\tau$</th>
<th>$T$</th>
<th>$c_{EDF}$</th>
<th>$c_{RMS}$</th>
<th>$r(\tau_{EDF})$</th>
<th>$r(\tau_{RMS})$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>20</td>
<td>15</td>
<td>13</td>
<td>15</td>
<td>13</td>
</tr>
<tr>
<td>1</td>
<td>30</td>
<td>6</td>
<td>7</td>
<td>36</td>
<td>20</td>
</tr>
<tr>
<td>2</td>
<td>60</td>
<td>1</td>
<td>7</td>
<td>-</td>
<td>60</td>
</tr>
</tbody>
</table>

---

39
If \( b \equiv 1 \), then \( x_{i,j} \) in eq. (30) must be set to a value greater than or equal to \( N \cdot c_j \). If \( b \equiv 0 \), then the constant \( M_{i,j} \) is added on the left-hand side of the inequation, and \( x_{i,j} \) is not bounded by \( N \cdot c_j \). To achieve this, \( M_{i,j} \) must be chosen sufficiently big. If the system is schedulable, each task’s WCRT must be smaller than its respective deadline. Therefore, a safe upper bound for \( M_{i,j} \) is the higher priority task’s deadline multiplied by the current preemption counter \( N \). \( M_{i,j} \) is therefore chosen as

\[
M_{i,j} = d_j \cdot N
\]  

(32)

These inequalities will ensure that the response time of each task is restricted to be lower than or equal to its respective deadline. If the deadline is too strict and this goal cannot be reached, the ILP will be infeasible.

C. ILP Objective Function

As shown above, achieving schedulability is ensured by ILP constraints. This leaves us to freely choose an arbitrary objective function for the ILP.

For both fixed and dynamic priority systems, we chose to use eq. (17) and set the ILP objective to minimize the amount of occupied scratchpad memory:

\[
\min (x_A \cdot s_A + x_B \cdot s_B + \ldots)
\]  

(33)

This way, system costs may be further reduced by reducing the size of the SPM in the system design. If the SPM size is fixed, the remaining SPM may freely be used to achieve additional design goals like optimizing the ACET.

VI. Modeling the Branch Prediction

Modern microcontrollers offer a variety of features which directly or indirectly affect the system’s timing behavior. When trying to minimize the WCET of a single task, most of these features can be neglected, as long as the timing benefits of moving a BB to the SPM are modeled roughly correct. The optimization’s result can be backed by running a precise WCET analysis using tools like aIT after applying the SPM allocation. This way, over- or even under-approximating the WCET within the ILP model does not cause any serious issues. As shown in eq. (20), the ILP model proposed in this paper forces each task’s WCRT to be lower than or equal to its deadline. Therefore, our framework needs to have a safe and tight over-approximation of each task’s WCET. If the over-approximation is too big, the ILP will be infeasible, although there might be a valid solution in reality. If the ILP formulation under-approximates the worst-case behavior, the ILP might propose a solution which is not schedulable in reality.

The Infineon TriCore TC1796 microcontroller which was used to evaluate this framework features a static branch prediction. Our tests showed that a precise modeling of the branch prediction is crucial and sufficient for tight WCET approximates within the ILP model. Modeling additional features like e.g., the prefetching unit would further increase the ILP’s accuracy but also increase the complexity of the ILP and therefore its solving time. Therefore, in the following we will account for branch prediction behavior but not for other microarchitectural features.

The prediction of a conditional jump instruction as “taken” or “not taken” depends on the offset between the instruction address of the conditional jump itself and the conditional jump target’s instruction address. Branches with negative offset or forward branches which are realized by a 16 bit wide instruction are predicted “taken”, 32 bit wide branches with positive offset are predicted “not taken” [18]. By moving a branch target from flash to SPM or by moving the branch instruction itself, the prediction may change. To ensure tight WCET estimates, we therefore modeled the branch prediction as additional constraints within our framework:

If a BB \( B \) ends with a conditional jump which is predicted “taken” in the unoptimized system, and \( B \) resides in flash but its explicit branch target \( E \) in SPM, the branch instruction must be converted to a 32 bit instruction due to the high address offset between flash and SPM. For the TC1796, the SPM’s address space is numerically higher than the address space of the flash. Therefore, the aforementioned SPM allocation will invert the outcome of the branch prediction.

Given a BB \( B \) which ends with a conditional jump instruction, its explicit branch target \( E \) and its implicit branch target \( I \), the execution time \( w_B \) can be modeled as shown in eq. (15):

\[
w_B \geq C_{B,\text{flash}} - x_B \cdot G_B + j_B \cdot C_j + w_E + p_{\text{branch,E}}
\]  

(34)

\[
w_B \geq C_{B,\text{flash}} - x_B \cdot G_B + j_B \cdot C_j + w_I + p_{\text{branch,I}}
\]  

(35)

The additional summands \( p_{\text{branch,E}} \) and \( p_{\text{branch,E}} \) account for changes in the timing behavior due to a changed branch prediction. The summand which has to be added to \( w_B \) in order to account for the branch prediction depends on the prediction behavior of the unoptimized system. The prediction behavior will therefore change if BB \( B \) is allocated to flash and the explicit successor BB \( E \) is moved to SPM:

\[
p_{\text{branch,E}} = P \cdot (\neg x_B \land x_E)
\]  

(36)

The inverted branch prediction additionally reduces the timing delay if the implicit successor is executed:

\[
p_{\text{branch,I}} = -G \cdot (\neg x_B \land x_E)
\]  

(37)

As a result, the program path covering the explicit branch target \( E \) receives additional \( P \) cycles penalty, and the implicit branch target \( I \) receives an additional gain \( G \). For the TC1796, \( P = 2 \) and \( G = 1 \) cycles.

If the branch was originally predicted “not taken”, the inequations which are added to the ILP are:

\[
p_{\text{branch,E}} = P \cdot (x_B \land \neg x_E)
\]  

(38)

\[
p_{\text{branch,I}} = -G \cdot (x_B \land \neg x_E)
\]  

(39)

The constraints for modeling the logical NOT and AND are not described in this paper due to the limited amount of space but can be taken from [17]. If the SPM’s address space is numerically lower than the flash memory’s addresses, the constraints can be formulated accordingly.
TABLE IV
WCETs for PapaBench “Autopilot”, times in CPU clock cycles

<table>
<thead>
<tr>
<th>Task Name</th>
<th>(c_{SPM}) (byte)</th>
<th>(c_{flash}) (byte)</th>
</tr>
</thead>
<tbody>
<tr>
<td>altitude_control</td>
<td>169</td>
<td>318</td>
</tr>
<tr>
<td>climb_control</td>
<td>435</td>
<td>715</td>
</tr>
<tr>
<td>link_Ibw_send</td>
<td>42</td>
<td>62</td>
</tr>
<tr>
<td>radio_control</td>
<td>1899</td>
<td>3268</td>
</tr>
<tr>
<td>stabilisation</td>
<td>537</td>
<td>918</td>
</tr>
</tbody>
</table>

TABLE V
WCETs for PapaBench “Fly by Wire”, times in CPU clock cycles

<table>
<thead>
<tr>
<th>Task Name</th>
<th>(c_{SPM}) (byte)</th>
<th>(c_{flash}) (byte)</th>
</tr>
</thead>
<tbody>
<tr>
<td>check_failsafe</td>
<td>497</td>
<td>1002</td>
</tr>
<tr>
<td>check_mega12k_values</td>
<td>7</td>
<td>17</td>
</tr>
<tr>
<td>send_data_to_autopilot</td>
<td>7</td>
<td>17</td>
</tr>
<tr>
<td>servo_transmit</td>
<td>426</td>
<td>1082</td>
</tr>
<tr>
<td>test_ppm</td>
<td>1563</td>
<td>2338</td>
</tr>
</tbody>
</table>

VII. Evaluation

For our evaluation we use the PapaBench benchmark which aims at controlling a small unmanned aerial vehicle [19]. It consists of two parts, “Autopilot” and “Fly by Wire”. Each consists of 5 tasks. As target platform, we used the Infineon TriCore TC1796 microcontroller running at 150MHz. A memory access to flash memory needs 6 cycles while accessing the SPM is completed in 1 cycle. All timings in this section are given in CPU clock cycles. Because the PapaBench benchmark is quite small compared to the TriCore’s SPM size, we scaled the TC1796’s instruction SPM size down to 50% of the tested benchmark’s overall code size. All WCETs were obtained using AbsInt aiT version b217166. To allow for a comparison of the results for EDF and RMS, we assume that each task’s deadline \(d_i\) equals its period \(T_i\). We use the WCC compiler framework presented in [2] as basis for our evaluations. We applied WCC’s compiler optimization \(-O2\) prior to running our SPM allocation which is roughly comparable to the \(-O2\) option offered by compilers like GCC or LLVM. After this, we performed a WCET analysis using aiT. We randomly generated periods to achieve distinct system loads greater than 1 for the system after \(-O2\) optimizations but prior to the SPM allocation. For each task load a uniformly distributed random value \(X_i\) was generated for each task \(\tau_i\) with \(0 \leq X_i \leq 1\). Assuming a system with \(n\) tasks, the period \(T_i\) of task \(\tau_i\) is then calculated by:

\[
T_i = \frac{c_{i, flash}}{u} \cdot X_i \cdot \sum_{j=0}^{n-1} X_j
\]  

(40)

To avoid statistical spikes, we generated 100 sets of periods for each benchmark for each system load. Tables IV and V show the WCETs of each task for both systems after \(-O2\) optimizations, if the whole benchmark is allocated to flash and if the whole benchmark is allocated to SPM.

As the system load is greater 1, each of these generated multitasking SPM allocation in order to repair the system and achieve schedulability. After the SPM allocation, a final run of aiT is used to calculate safe and tight WCETs.

Figure 2 shows the result for the Autopilot benchmark. It illustrates how many of the generated task sets could be repaired by our proposed approach. Additionally, the percent of fixable systems under optimal circumstances, i.e., if all tasks could be completely moved to the SPM, is plotted. Although this is not realistic, it allows for a better assessment of the optimization’s quality. For loads up to 1.5, almost all task sets can be optimized for both EDF and RMS. For relatively small loads up to 1.3 few task sets can be optimized for RMS but not for EDF. This anomaly is caused by the used ILP solver IBM CPLEX. For EDF scheduling, eq. (18) has to be added as constraint. If a task’s period is very big, the resulting factor \(\frac{1}{\tau_i}\) may get so small that the ILP solver will return an unbounded result due to a computational underflow. Figures 3 and 4 show the amount of needed SPM to ensure schedulability for Autopilot. The SPM size was limited to 3213 byte. As expected, the SPM size is usually smaller for EDF scheduling. For higher loads, EDF has higher amounts of used SPM, as the needed SPM size was only considered for systems which may be scheduled. This basically shows that for EDF scheduling the available SPM can be used more efficiently.

Figures 5, 6 and 7 show the results for Fly by Wire. Its code size is only 4580 byte, resulting in a SPM size of 2290 byte. The results are very similar to Autopilot, except that a reasonable amount of task sets may be repaired for system loads up to 2.3 due to a tighter WCET over approximation. Both benchmarks show that if a schedulable system is created, only a small fraction of the code has to be moved to the SPM.

![Fig. 2. Repairable systems for PapaBench Autopilot](image)

![Fig. 3. Needed SPM size if a task set could be repaired for Autopilot and EDF scheduling.](image)

![Fig. 4. Needed SPM size if a task set could be repaired for Autopilot and RMS.](image)
In conclusion, our framework is able to adapt most existing ILP-based schedulability analysis tools, which is particularly important for programs with complex inter-task dependencies. It should be mentioned that our model does not automatically account for context switching costs. Scheduler costs may be modeled as another task which is called periodically. This is feasible, as schedulers are often called by an Interrupt Service Routine which is triggered by a hardware clock. Additionally, it is crucial to handle shared code which is used by several different tasks. In the future, we will propose extensions to our framework, such as the integration of support for inter-task dependencies.

**REFERENCES**


