Technical Report

Real-Time Aware Hardware Implementation Effort Estimation

Abildgren, Rasmus; Diguet, Jean-Philippe ; Gogniat, Guy; Koch, Peter; Le Moullec, Yannick

Publication date:
2010

Document Version
Early version, also known as pre-print

Technical Report: Real-Time Aware Hardware Implementation Effort Estimation

Rasmus Abildgren*, Jean-Philippe Diguet†, Guy Gogniat†, Peter Koch‡, and Yannick Le Moullec‡

*CISS
Aalborg University,
Selma Lagerlöfs Vej 300,
DK-9220 Aalborg East, Denmark

†European University of Brittany
UBS – CNRS, Lab-STICC
Centre de recherche BP 92116
F-56321 Lorient Cedex, France

‡CSDR
Aalborg University,
Fredriks Bajers Vej 7,
DK-9220 Aalborg East, Denmark

***Abstract*—This paper presents a structured method and underlying models for estimating the hardware implementation effort of hard real-time constrained embedded systems. We propose an optimization model which takes some of the most common optimization techniques into account as well as the order in which they should be applied. We suggest a set of two metrics used to characterise the effects of optimisations: one expressing how hard it is to reach an implementation satisfying the real-time constraints for the implementation, and another one to reflects how the distribution of parallelism in an algorithm influences the impact of the optimisations. Experimental results do not show an unambiguous result. However, for most algorithms our approach enables the estimation of the hardware implementation effort of hard real-time constrained applications.**

***Index Terms*—real-time; hardware; implementation; effort; FPGA**

I. Introduction

The need for continuous innovation combined with growing complexity, increased product release frequency, increasing time-to-market pressure and fierce competition, make the task of project managers working in the embedded systems industry more and more challenging. For example, the 2009 Embedded Market Study [1] reports that 63% of the projects were not finished on schedule and that the average lateness is 4.4 months.

In this context, accurate development time estimates are an essential tool which can make the difference between success and failure. However, obtaining such estimates is not a trivial task since development time depends on many factors, including both technical (hardware and software e.g. products built upon new platforms, area/time/energy constraints), human (e.g. skills, mood of the developers), and managerial aspects.

Whereas methods and techniques are readily available for estimating the development time of software executing on GPPs [2], mainly for desktop applications, to the best of our knowledge little efforts have been carried out in the hardware domain. Working with a systematic approach for estimating the development time for different projects requires a certain maturity in the organisation. Many small and medium sized enterprises (SMEs) usually do not have such priories, although they would benefit from it. Many SMEs perform ”ad hoc” estimations (e.g. based on experience or intuition), but in many cases these ad hoc approaches do not provide accurate estimates, which in turn means delayed projects. This work proposes a method and tools which offer a systematic and structured approach for estimating more precisely the implementation effort.

A. Our Prior Work

The work presented in this paper is part of a larger effort aiming at improving this situation [3]. In this paper we describe our contribution regarding the problem of estimating the hardware implementation effort (in terms of development time) for real-time constrained applications. This contribution is an extension of what is summarized below.

In [3] it has been shown that every path in the algorithm(s) that the designer must implement adds to the development time and that the complexity of a design can be expressed by the number of independent paths in the algorithm(s). It has also shown that, when the experience of the designer is taken into account, a relation between the number of independent paths and development time exists and that it is possible to estimate the hardware implementation effort (in terms of development time) of applications.

However, in many cases the implementation can become very challenging when (hard) real-time constraints need to be fulfilled. Typically in such cases only a limited number of “implementational tracks” lead to a (or sometimes the) satisfying solution. This can be illustrated as in Fig. 1, where only one implementation track (the thick red line) satisfying the contraints. The idea on which this work builds upon is that real-time constraints make the implementation more difficult and that, in order to fulfill these constraints, designers need to perform certain optimizations in a certain order. Identifying a or the suitable track(s) adds to the overall development time since extra efforts must be spent at evaluating and applying the right combination (i.e. type and order) of optimization techniques. In order to take these considerations into account, we propose to complement and augment our prior work with several extensions. These extensions are described in Section I-B.

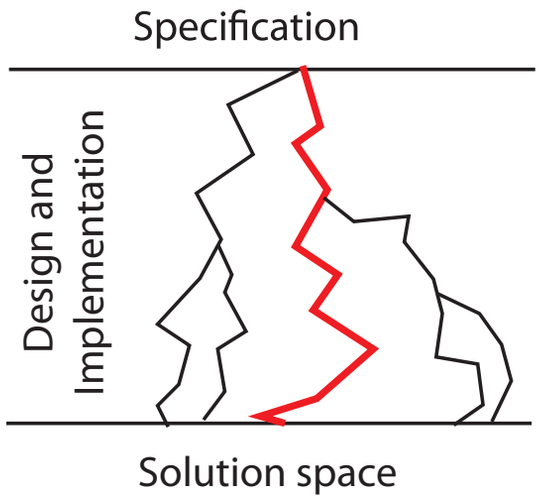


Fig. 1. The task of designing and implementing an algorithm can be seen as going from a specification to a solution. Typical more than one solution will satisfy the same specification. However, when constraints are introduced, the solution space narrows down and only a few or one "implementation tracks" (depicted as the thick red line) will result in a satisfying solution.

B. Contributions

One major contributor to the overall development time is the implementation effort. The contributions presented in this paper are i) a method and ii) a set of underlying models aiming at estimating the implementation effort, measured in time, of real-time constrained embedded applications. We propose an optimization model which takes some of the most common optimization techniques into account as well as the order in which they should be applied. An essential contribution of this work is a set of two metrics used to characterise the effects of optimisations. The first one expressing how hard it is to reach an implementation satisfying the real-time constraints for the implementation. The second one reflects how the distribution of parallelism in an algorithm influences the impact of the optimisations.

The remainder of the paper is organized as follows: section II introduces related works. Section III details the proposed methodology and section IV details the metric for the distribution of the parallelism. Subsequently, section V presents and discusses the experimental results obtained. Finally, section VI concludes the paper.

II. State of the art - Effort Estimation

To the best of our knowledge, very few works address the problem of estimating the hardware implementation effort of hard real-time constrained applications. On the other hand, there exist several approaches for estimating the software implementation effort, some of them providing ideas and directions for the hardware oriented ones. Thus, in this section we start by reviewing the most relevant approaches for estimating the software implementation effort and proceed with the few existing approaches for hardware implementation effort estimation.

Some of the most known and used tools for estimating the software implementation effort are the COCOMO project [2], function point [4], and SPQR/20 [5]. They all build upon the same concept: firstly, in order to quantify certain properties of an algorithm, a measure or set of measures is defined. Secondly, a model describing the relation between the measure(s) and the implementation effort is derived.

The core idea in COCOMO (COnstructive COst MOdel) [2] is that the effort mainly depends on the project size, i.e., $Effort = A \cdot size^b$ where A and b are adjustable parameters which must be trained in order to reflects factors such as manpower, experience of the developers, etc. The remaining parameter in the equation, the size of a project, can be measured by means of e.g. Lines of Codes (LOC); however, this is subject to criticism and thus other measures have been proposed like function point.

Function point [4] consists of two main stages: the first stage consists in counting and classifying the function types of the software: identified functions are weighted to reflect their complexity, which in practice is left to the developers's perception. The second stage is the adjustment of the function points based on 14 parameters which are tuned according to the characteristics of the application and of its environment. Subsequently, the function points are converted into a LOC measure based on an implementation language-dependent factor, which in turn can be used as an implementation effort estimation metric.

SPQR/20 (Software Productivity, Quality and Reliability with regard to 20 influencing factors) has been proposed as a less heuristic-oriented variant of function point; experimental results [6] suggest that it can provide the same accuracy than function point while being simpler to work with.

Publications dealing with the estimation of hardware implementation effort are far less abundant than those dealing with software. Considering the context of the present work, interesting approaches include VHDL function point [7] and cost models such as [8]. Several other publications such as [9] compare actual hardware implementation efforts for different design methodologies but do not provide any systematic method to estimate those efforts.

VHDL function point, presented in [7], builds upon the idea of function points analysis and is modified to work with VHDL code. The approach consists in counting the number of internal I/O signals and components, and classifying these counts into levels. From there, a function point value related to VHDL is extracted. Experimental results considering the number of source lines in the LEON-1 processor project yields predictions which are within 20% of the real size. However, estimating the size does not always give an accurate indication of the implementation difficulty, especially when the application is subject to real-time constraints.

[8] introduces a cost model with the objective of understanding current Product Development Cycles (PDC) and evaluating the impact of new technologies on these PDC. In particular, the authors focus on cost and product development time and propose a PDC known as One Pass to Production

(OPP) which takes both software and hardware aspects of a complete system into consideration. Although promising, their approach is very specific (they consider a FPGA-based NOC backbone) and the numerous assumptions made by the authors (e.g. regarding the number of required engineers) make it challenging to see how their approach could be made sufficiently generic to be applied to much more varied types of applications.

We can safely conclude that there is currently a lack of suitable and systematic methods and tools for estimating the hardware implementation effort for real-time constrained applications. In what follows we present our contribution to improve this situation.

III. Methodology

In [3] it has been shown that the hardware implementation effort can then be modelled as

$$Effort = A(\eta(Dev) \cdot P(alg))^b \quad (1)$$

where η reflects the experience of the developer Dev, $P(alg)$ is the number of independent path in the algorithm alg, and A and b are trim parameters.

Experimental results have shown that it is possible to estimate the hardware implementation effort, expressed as the development time, of applications and that the proposed model is able to estimate the need implementation effort with a confidence interval of 95%. However, this approach is not specifically targeting real-time constrained applications and is therefore not suitable for this type of application.

Since we in this work want to take hard real-time constraint into account, we propose a method which adds a parameter $\tau(t_c)$ expressing the difficulty or hardness of reaching an implementaion which meets the time constraint, t_c. This parameter we will call implementation hardness and therefore the effort can be modeled as:

$$Effort = A(\eta(Dev) \cdot P(alg) \cdot \tau(t_c))^b \quad (2)$$

The underlying idea is that as far the execution time t_{exec} is from t_c the more difficult it will be to fullfil t_c. Whenever t_c is not met, optimizations have to be performed. However, modeling optimizations and their impact is not a trivial task for a designer; therefore, in what follows, we propose a method and a set of models which reflect the most common cases.

A. Real-Time Constraint

When optimizing the implementation to meet a real time constraint, the optimization strategies can be many fold. For a developer, the optimization strategy is very much application dependent but also depends on his experience and on his analytical thinking. Optimizations can fall into two different domains; spatial and temporal. Optimizations in the spatial domain include algorithmic parallelism exploitation on multiple functional units. For the temporal domain, different optimization techniques exist such as chaining and pipelining.

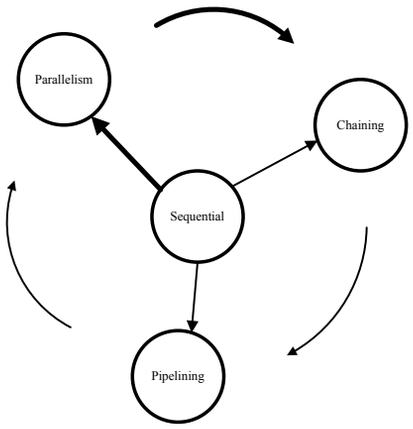


Fig. 2. The overall optimisation approach where the starting point usually will be the sequential version of an algorithm. The developer freely chooses in which order he/she performs the different optimisations strategies. For our approach we constrain the optimisation strategy to follow the thick line.

Typically, the type of optimization to be performed in the temporal domain is chosen depending on a) data/control dependencies in the algorithm and b) the constraint type:

- Throughput (pipelining)
- Latency (chaining)

Both type of constraints can benefit from parallelism exploration. Usually when analyzing an algorithm, for e.g. parallelism, the measure applied will indicate the potential of exploiting the entire parallelism in the algorithm, as for example with the measure γ [10]. Performing a straight manual implementation of an algorithm will usually not result in a complete exploitation of the parallelism. Either because it is not necessary or because the designer has omitted optimizations which could have a significant impact on the exploitation of the algorithm's inherent parallelism.

An illustration of the overall optimisation approach is shown in Fig. 2. The starting point will usually be a sequential version of the algorithm. The developer chooses in which order he/she performs one or several of the different optimisations strategies. The order and types of strategies will vary from algorithm to algorithm. To generalise our approach we constrain the optimisation strategy to follow the template denoted by the thick line in Fig. 2. This limits the overall strategy to complete the parallelism optimisation strategy before starting the chaining strategy. We will not consider pipelining optimisations further in this paper.

In order to guide the designer in the exploration of the parallelism, this work considers a fully spatially parallelized algorithm as an extreme. Similarly, a complete chained implementation is also considered as extremes, these extremes indicate the bounds of how much speedup can be obtained when applying the respective type of optimization, without rewriting the algorithm.

Furthermore, not knowing the exact strategy that a development engineer is following, but knowing which options he/she has, our hypothesis is that it is possible to estimate the minimum number of optimizations required in order to

fulfill a real-time constraint. This, in turn, provides useful information which can be converted into the implementation hardness parameter, τ, of Eq. 2 for estimating the required implementation effort.

It is therefore important to know how many optimizations inside the different categories should be applied in order to fulfill the time constraint, t_c. The next section describes the concept of estimating the execution time on basis of the number of optimisations.

B. Optimisation Dependent Execution Time Estimation

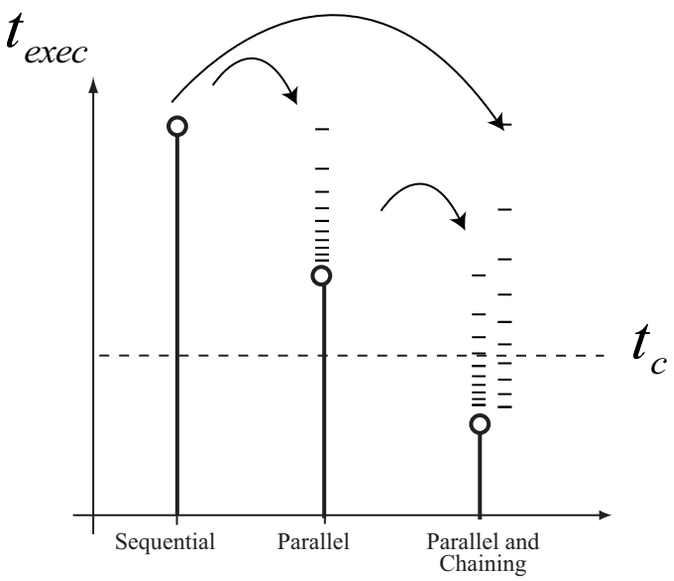


Fig. 3. Relation between the number of applied optimisations (represented by the small vertical lines) and the resulting execution time, t_{exec}, for several optimisation strategies (parallelization and parallelization+chaining). The reduction of the execution time gets smaller as the number of optimisations increases for a certain strategy (represented by the spacing between the small vertical lines). In order to arrive at an execution time equal to or smaller than the time constraint, t_c, several possible paths exist which depend on a combination of different optimisation strategies and the number of applied optimisations for each strategy.

Every optimisation yields a certain speed-up to the execution time. Fig 3 shows the execution time of an algorithm with different numbers of optimisations for the different optimisation categories. This is illustrated by the relation between the number of applied optimisations (represented by the small vertical lines) and the resulting execution time (t_{exec}) for several optimisation strategies (parallelization and parallelization+chaining).

The reduction (per optimisation) of the execution time gets smaller as the number of optimisations increases for a certain strategy. This is represented by the spacing between the small vertical lines. In order to arrive at an execution time equal to or smaller than the time constraint (t_c) represented by the dashed line, the designer can choose between several possible paths. An optimization path is the number of optimizations performed in the parallelization category followed by the number of optimizations performed by chaining. The number of optimisations in the different categories can vary since there can be more than one path satisfying the time constraint.

Therefore, it is necessary to know an estimate of the execution time for different optimization paths. In the following we describe how to estimate the execution time for the non-optimized case and the three different optimisation cases:

1) Case 0: No optimization (sequential execution): The most simple case is the sequential execution. To calculate the estimate, t_{exec}, we use the following equation:

$$t_{exec}(0) = NOP\frac{1}{\hat{f_{arch}}} \tag{3}$$

where NOP denotes the number of operations in the sequential algorithm and $\frac{1}{\hat{f_{arch}}}$ the time for executing one operation. In this work we assume that all operations can be considered as atomic and therefore have the same execution time.

2) Case 1: Parallel optimization: The estimated execution time, $t_{exec}(NOO_{PAR})$, when applying a certain number of parallelization optimisations, NOO_{PAR}, can be expressed as

$$t_{exec}(NOO_{PAR}) = \frac{NOP}{\gamma_{impl}(NOO_{PAR})}\frac{1}{\hat{f_{arch}}} \tag{4}$$

where $\gamma_{impl}(NOO_{PAR})$ expresses the degree of speed-up obtained with NOO_{PAR} number of optimisation. This can be calculated as:

$$\gamma_{impl}(NOO_{PAR}) = \frac{NOP}{NOP - NOP_{opt}(NOO_{PAR})} \tag{5}$$

where $NOP_{opt}(NOO_{PAR})$ expresses the reduction in executed operations in the critical path when NOO_{PAR}, number of optimizations, are applied. How to obtain this estimate is further discussed in section III-C.

All in all this gives:

$$t_{exec}(NOO_{PAR}) = (NOP - NOP_{opt}(NOO_{PAR}))\frac{1}{\hat{f_{arch}}} \tag{6}$$

3) Case 2: Chaining: Similarly, for chaining we can express the estimated execution time, $t_{exec}(NOO_{Chain})$, as:

$$t_{exec}(NOO_{Chain}) = (NOP - NOP_{opt}(NOO_{Chain}))\frac{1}{\hat{f_{arch}}} \tag{7}$$

where NOO_{Chain} denotes the number of applied chaining optimizations. Please note that, $\hat{f_{arch}}$, the frequency of the architecture will typically change when creating larger operators.

4) Case 3: Combined: Combining the parallelized and chained cases will leave us with the following equations:

$$t_{exec}(NOO_{PAR}, NOO_{Chain}) = \left(\frac{NOP}{\gamma_{impl}(NOO_{PAR})} - \frac{NOP_{opt}(NOO_{Chain}|NOO_{PAR})}{\varphi(NOO_{PAR})^{-1}}\right)\frac{1}{\hat{f_{arch}}} \tag{8}$$

where $\varphi(NOO_{PAR})$ is a parallelism distribution measure which takes the fact that a chaining optimisation in the parallel context does not necessarily result in a reduction of the execution time. We discuss this later in section IV.

C. Optimisation Impact Estimation

When implementing an algorithm containing loops, different loops have different numbers of iterations. Typically, loops with larger numbers of iterations contribute more to the execution time of the algorithm than loops with small numbers of iterations. Optimizing an operation in a loop with a large number of iterations yields a larger reduction of the execution time. Assuming that the effort required to perform an optimisation does not change with the number of iterations, processing the loops with the largest number of iterations first, pays a larger impact on the reduction of the execution time for a given implementation effort. It is therefore essential to take this into account when estimating the optimisation impact on the execution time.

The real impact of an optimisation on algorithms that include loops can not be known without deep inspection of the algorithm; however, an approximation would be beneficial. We therefore propose a measure approximating that. The requirements for defining such a measure include that it should reflect the number of executed operations compared to the number of operations that need to be implemented. In Fig. 4 a random algorithm containing loops is considered. The figure shows the relation between the number of optimisations and the corresponding reductions in the executed number of operations. The optimisations are ordered according to their impact on the execution time of the algorithm. The solid line represents the real impact. The dashed line, the average and the dotted line, a first order logarithmic based approximation. The real line corresponds to the case where the optimisations are fully prioritised according to their impact, the average line corresponds to the mean impact of a random optimisation strategy.

One interesting point in the graph in Fig 4 is the end point. The number of operations which can be parallelized as well as the number of operations which can be chained limit the possible number of optimisations. We denote the maximum number of optimisations for the parallel case as:

$$|NOO_{PAR}|_{max} = NOP_{impl} - CP_{impl} \tag{9}$$

where NOP_{impl} represents the number of implemented operations and CP_{impl} the number of implemented operations which are present in the critical path. These numbers are different from NOP and CP when loops are present since the operations inside a loop are executed several times. Similarly to the measure in Eq. 9 a measure, $NOP_{opt}(|NOO_{PAR}|_{max})$, for the maximum number of executed optimised operations can be calculated. The ratio between these two measures reflects the average impact of the loops present in the algorithm when taking the parallelism into account. This will be the slope of the average line in Fig. 4.

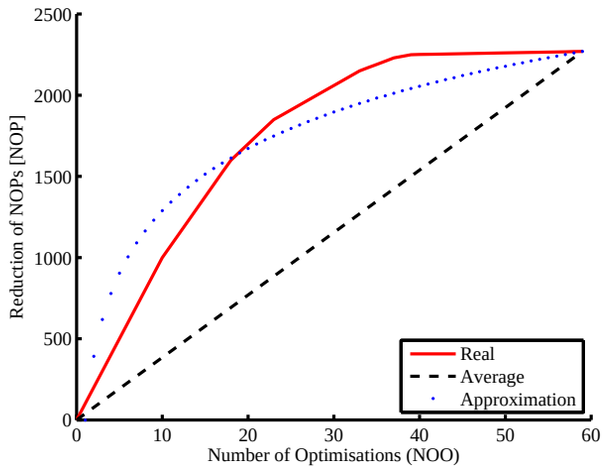


Fig. 4. The figure shows the impact (in terms of reduction) of optimisations in an algorithm with operations in different loops and also outside loops. The operations are optimised following the order of their impact on the execution time of the algorithm. The solid line represents the real impact. The dashed line, the average and the dotted line, the approximated. The real line corresponds to the case where the optimisations are fully prioritised according to their impact, the average line corresponds to the mean impact of a random optimisation strategy.

A similar approach is used for the chaining case except that the maximum number of optimisations is calculated as:

$$|NOO_{Chain}|_{max}(NOO_{PAR}) = NOP_{impl} - P(NOO_{PAR}) \tag{10}$$

where $P(NOO_{PAR})$ denotes the number of paths in the algorithm, which is further detailed in section IV.

It turns out to be difficult to obtain a good and stable first order approximation of the impact of the loops in the algorithm based on the limited number of data which we have available. We have therefore decided to use the average as a measure for the impact of an optimisation which can be calculated as:

$$NOP_{opt}(NOO_{PAR}) = \frac{NOP_{opt}(|NOO_{PAR}|_{max})}{|NOO_{PAR}|_{max}} NOO_{PAR} \tag{11}$$

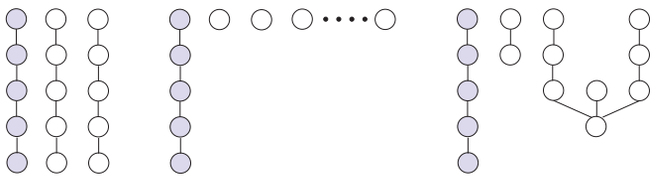
and

$$NOP_{opt}(NOO_{Chain}|NOO_{PAR}) = \frac{NOP_{opt}\left(|NOO_{Chain}|_{max}(NOO_{PAR})\right)}{|NOO_{Chain}|_{max}(NOO_{PAR})} NOO_{Chain} \tag{12}$$

IV. Metrics

A. Metric of distribution of parallelism

When chaining operators, the impact on the execution time depends on whether the optimisations are done in the critical path or in other paths. For this work we expect that the developer has carefully analysed the algorithm and is only optimising where it is most feasible, i.e. in the critical path.



(a) Extreme 1 (b) Extreme 2 (c) Average

Fig. 5. Illustration of the distribution of the parallelism in the algorithm. Fig 5a and 5b shows the two extremes, with either the critical paths (grey) being comparable to the other paths (denoted highly distributed case) or completely unique (denoted narrow case). Fig 5c shows a more average case. All examples have 15 nodes and a critical path of 5, which will give a speedup measure, $\gamma = 3$. When operations get chained, the different cases lead to different reductions of the critical path. This makes it difficult to predict the reduction of the critical path per chaining optimisation. This calls for a metric indicating the distribution of the parallelism.

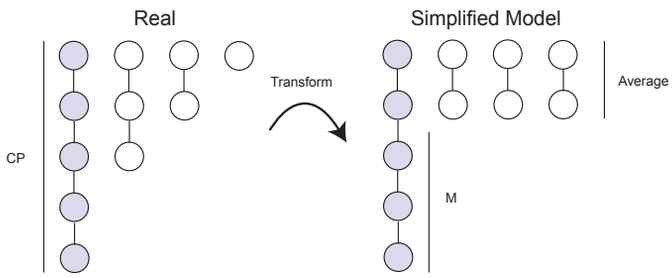


Fig. 6. Every graph can be transformed into a graph which can be treated as the two extremes of Fig 5. M denotes the length of the critical path which is longer than the average of the off critical paths length.

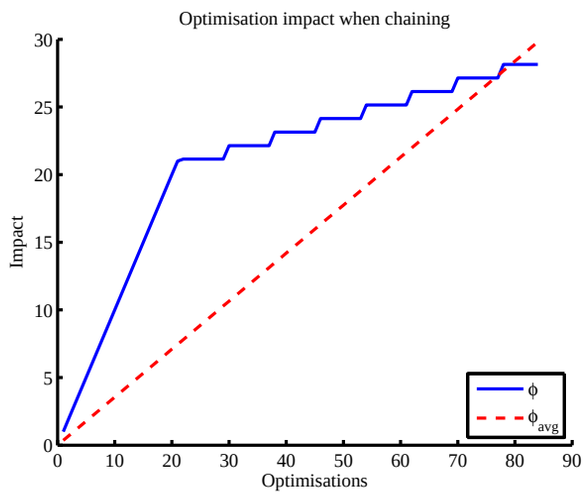


Fig. 7. This figure shows the relation between the number of optimisations and the corresponding impact when performing chaining optimisations. The solid line shows the impact when working with the transformed graph. In the beginning every optimisation is performed in the critical path, where every optimisation results in a reduction. When the average length of the off critical path paths is reached, chaining optimisations need to be performed in every path to induce a reduction. The dashed line denotes the average impact of the chaining optimisations.

However, chaining operations in the critical path can lead to a situation where the path which was originally the critical one is reduced, due to the optimisations, so another path becomes the the longest one. Fig 5 shows three different examples, all having 15 nodes and a critical path of 5, which gives a speedup measure, $\gamma = 3$. Fig 5a shows an example where the initial critical path (grey) has the same length as the two other paths in the algorithm (highly distributed case). Chaining two operations in the critical path will change the longest path to one of the two others. Opposite to this, Fig 5b shows an example where the initial critical path is significantly longer (narrow case), which means that chaining operations in this case will lead to a reduction of operations in the initial critical path. In between, Fig 5c shows a more average example.

Knowing the graph would make it possible to derive the exact reduction of the algorithm's critical path with a specific number of chaining optimisations. However, not knowing the graph but only the average speedup, γ, the number of operations and the length of the initial critical path makes it challenging to predict this reduction.

In order to obtain a more sufficient estimate of the effect of an average chaining optimisation, we propose a metric which considers the distribution of the parallelism in the algorithm.

It is desirable that such a metric has the following properties: in case of a highly distributed parallelism (see Fig 5b), i.e. many paths in the algorithm, the value of the metric should converge towards one. In case the distribution is "narrow" (see Fig 5a), i.e. the number of paths is equivalent to the speedup, the metric should give a value close to zero.

Most graphs will not fall into the two extremes from Fig 5, but will be more like the average case. In order to obtain the metric of the the average contribution of chaining optimisations, we propose a mechanism with which any graph can be transformed and handled as a combination of the two extremes. The transformation is illustrated in Fig 6. The mechanism is as follows: keep the critical path fixed and substitute the off critical paths (i.e. all paths excluding the critical one) with paths having their average length.

Doing this transformation brings us to a simplified problem where we first can handle chaining as the narrow distributed case, and second as the highly distributed case. It is given that the impact of the chaining optimisation will always be better or equal to this simplified model, with this number of paths. An optimisation following this model is shown in Fig 7.

In order to handle that graph as the narrow distributed case, it is important to know how large the difference between the critical path and the off critical paths is. To do so we utilize $P(NOO_{PAR})$, the number of paths of the parallel-optimised algorithm. The difference between the critical path and the average of the off critical paths can then be calculated as:

$$M = \left(1 - \frac{\gamma - 1}{P(NOO_{PAR}) - 1}\right) CP \qquad (13)$$

when the critical path is changed so that the critical path has the same length as the average of the off critical paths, the scenario changes to the highly distribution extreme. The rest of the chaining optimisations are handled as so.

Since most algorithms do not fall into one of the two extremes, when estimating the impact of a certain number of

chaining optimisations the average measure is more representative than the measure obtained based on considering the two extremes.

Furthermore, it can be shown that the real impact from optimisation will be equal to or better than the average of the simplified model[1]. Using such a measure will therefore ensure that the estimates of the chaining optimisation impact are not overestimated.

The measure for the impact of a chaining optimisation can therefore be denoted by the following:

$$\varphi(NOO_{PAR}) = \left(\frac{M}{NOP} + \frac{CP - M}{NOP - M}\right) \tag{14}$$

With all these defined we are now ready to find the number of optimisations need to fulfil the real-time constraint, and define the metric expressing how hard it is to reach this implementation. We call this metric for implementation hardness.

B. Implementation hardness

Knowing the maximal number of possible optimisations and the minimum needed to meet the time constraint, the ratio (Eq: 15) between these two indicates how much the implementation needs to be investigated. Using the analogy with the implementation tracks, a number close to one indicates that almost all possible optimisations in the algorithm need to be considered, and only a very limited number of tracks will lead to a solution. Finding these solutions will require a lot of effort. On the other hand a number close to zero indicates that few optimisations are required to meet the time constraint and thus almost every implementation track will result in a satisfying solution. Hereby less effort is probably needed.

$$\tau(t_c) = \frac{|NOO_{PAR,Chain}(t_c)|_{min}}{|NOO_{PAR,Chain}|_{max}} \tag{15}$$

Using $\tau(t_c)$ in Eq. 2 we are now able to express the implementation hardness, $\tau(t_c)$, and thereby refine the estimated implementation effort.

V. Results

Similar to when we developed the implementation effort estimation technique in [3], we will verify the proposed improvement by first building a model on basis of the same training data as in [3], and then validate the model with a set of validation data, which is also the same as used earlier. By doing so, we are able to first test if our considerations are valid and tune the proposal, and still use the second set of real-life data to evaluate whether it generalizes.

To summarize, the training data originates from two different application types that are both developed as academic projects in universities in France. The training data has therefore not been produced specifically for this project, but is comparable to data from industrial projects. The first application is composed of five different video processing algorithms that are able to track moving objects in a video sequence. The second application is a cryptographic system, where we use the hashing algorithms, MD5, AES and SHA-1, as well as a combined crypto engine, which is also part of the system. The developers for the training data have been a Ph.D. student and M.Sc.EE. students, as can be seen in Table I.

The validation data originates from a local company, ETI A/S, which is a Danish SME. The dataset contains algorithms from a state-of-the-art network system and consists of Ethernet applications implemented on FPGAs, as well as corresponding testbeds. The system is a real-time system with hard time constraints, and all algorithms were implemented as to meet these constraints. The developers for the validation data had some experience before starting the implementation as shown in Table I. For more information about the data we would like to point the reader to [3].

A. Training Data

Fig. 8 shows the training data where the uncorrected complexity as defined by the number of linearly independent paths (as defined in [3]) is plotted in relation to the needed effort. A small update in the method of how to measure the independent paths have been applied compared to [3]. This implies that we now only measure the core of the algorithm, which is the part going to be implemented on the FPGA, and do not include small fragments of data formatting code. Taking these data and applying the original experience transformation on the data, results in the picture shown in Fig. 9. A least-squares fit trend line can be extracted to form our model (Eq. 1):

$$Effort = A(\eta(Dev) \cdot P(alg))^b \tag{16}$$

where the trim parameters $A = 0.196$ and $b = 1.191$. This is depicted as the dash-dot-dash line.

In Fig 11, the new parameter $\tau(t_c)$ taking the real-time constraint into account is applied. The $\tau(t_c)$ value for the different algorithms is shown in the upper part of Table II. This changes the complexity for the different algorithms a little, and a new least-squares fit line of our model is depicted with the dashed line. The trim parameters of our model (Eq. 2):

$$Effort = A(\eta(Dev) \cdot P(alg) \cdot \tau(t_c))^b \tag{17}$$

are now $A = 0.209$ and $b = 1.181$.

A comparison of the two models is shown in Table III, where the model taking the real-time constraint into account performs slightly better. However the result is not statistical significant. However, we continue with the

B. Validation Data

We continuing by validating the correctness of the model using the validation data. In Fig 11, the corrected validation data are shown together with the model, which is depicted by the dashed line, and a 95% confidence interval. Both the model and confidence interval are extracted from the training data. It is clear that most algorithms fit nicely with the proposed model and are well within the confidence interval. The exceptions are

[1]equal to or better than only applies when considering the lower integer value of the achieved reduction, since no partial operators exist.

algorithm SS4 and SS5. In the next section we will discuss this in details. The mean and variance of the prediction errors are shown both with and without SS4 and SS5 in Table IV.

C. Discussion of result

Most of the algorithms fit nicely with the proposed model and are well within the confidence interval. The improved model indeed does perform better than the original model, which had a mean error of 0.2 and a variance of 8. Taking a closer look at Table II shows that for all the Ethernet algorithms, except for SS3, we obtain an implementation hardness value $\tau(t_c)$ very close to one. This indicates that the implementations have been very close to the maximum achievable with the algorithm. This fits very well with the knowledge that these algorithms are used in state of the art high performance systems. However, the result for SS3 shows that it should have been possible to choose a less optimised solution and still meet the constraints, which would have resulted in a reduction of the implementation time, at least if the model holds for this algorithm.

An exception to these results are the SS4 and SS5 algorithms, where the estimates do not fit the model. Looking at table II again, we see that their implementation hardness value is set to 1. This is an error value actually indicating that the time constraint cannot be met with the current algorithm, and an algorithm transformation is needed. This indicates that the algorithm which is used for estimating the complexity of the implementation and thereby the needed effort, will not be able to fulfill the requirements, and an algorithm transformation is probably needed. Not going into details with the two algorithms, we can say that their final implementations involve a lot of bit manipulation which is not easily reflected in the initial C algorithm which is used for the measurement. A safe conclusion is therefore that if the implementation hardness factor indicates the need for algorithm transformation, the result would hardly be covered by the proposed model.

Furthermore it is also important to stress that our set of data originates from a single company with few developers. So strictly speaking we can only conclude that this model can be applied to the specific SME setup involved in the study and partially to the academic environment studied. A large volume and variety of experimental data for training and validation is needed to generalise our model. Also, the model can be refined with more parameters for more precise results.

VI. Conclusion

Accurate development time estimates are an essential tool for project managers working in the embedded systems industry. Obtaining such estimates is challenging and in particular very few existing works can provide hardware implementation effort estimates. In this paper we have presented our contribution to this topic, namely a systematic and structured approach for estimating the hardware implementation effort of hard real-time constrained applications.

The underlying idea off this work is that implementing a system is more difficult when hard real-time constraints must

TABLE I
FACTS ABOUT THE DEVELOPERS. DEVELOPERS FOR TRAINING DATA (TOP) AND VALIDATION DATA (BOTTOM)

Developer	Education	Years in the domain
Dev 1	Ph.D. stud.	0
Dev 2	Stud. (EE)	0
Dev 3	Stud. (EE)	0
Dev 4	Stud. (EE)	0
Dev 5	BSc.EE.	9
Dev 6	MSc.EE.	15
Dev 7	MSc.EE.	9
Dev 8	MSc.EE.	8
Dev 9	MSc.EE.	8

TABLE II
TRAINING DATA (TOP) AND VALIDATION DATA (BOTTOM). ALGORITHMS ARE RELATED TO THE IMPLEMENTATION HARDNESS, $\tau(t_c)$, THE DEVELOPERS, AND THEIR EXPERIENCE AT THE GIVEN TIME. COMPLEXITY IS NOT CORRECTED.

Algorithm	Complexity	$\tau(t_c)$	Developer	Dev. Exp.
T1	10	0.97	Dev 1	2
T2	24	0.99	Dev 1	10
T3	12	0.91	Dev 1	18
T4	14	0.96	Dev 2	1
T5	4	0.89	Dev 1	20
MD5	10	0.98	Dev 3	1
AES	10	0.99	Dev 4	8
SHA-1	27	0.98	Dev 4	14
Combined	59	0.99	Dev 4	14
SS1	25	0.99	Dev 6,7	150
SS2	35	0.98	Dev 5	150
SS3	17	0.26	Dev 5,6,7,8	150
SS4	50	1	Dev 6	6
SS5	29	1	Dev 7	3
SS6	25	0.99	Dev 5,6,7	3
Ethernet app	60	0.99	Dev 5,6,7,8,9	150
App 4	9	0.94	Dev 6	150

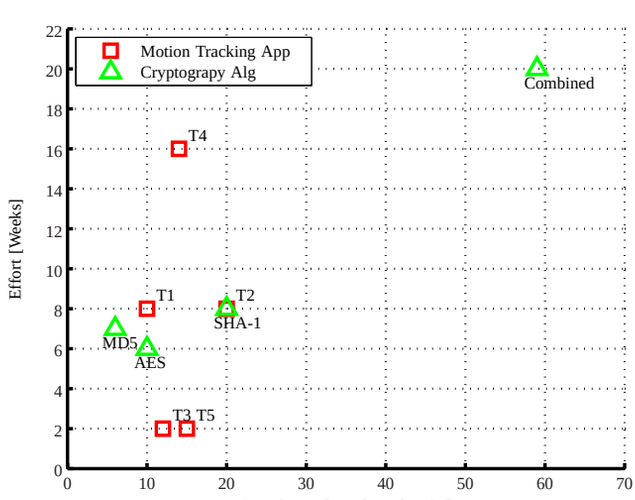


Fig. 8. Relation between the implementation effort [number of weeks] and the uncorrected complexity.

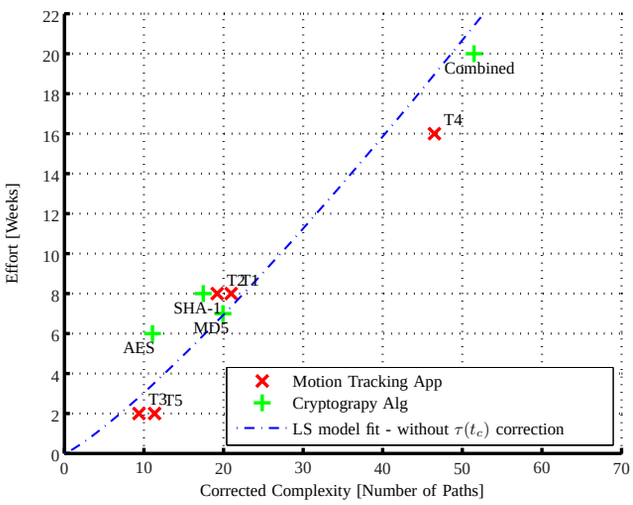


Fig. 9. Relation between the implementation effort [number of weeks] and the complexity corrected according to the designers experience model.

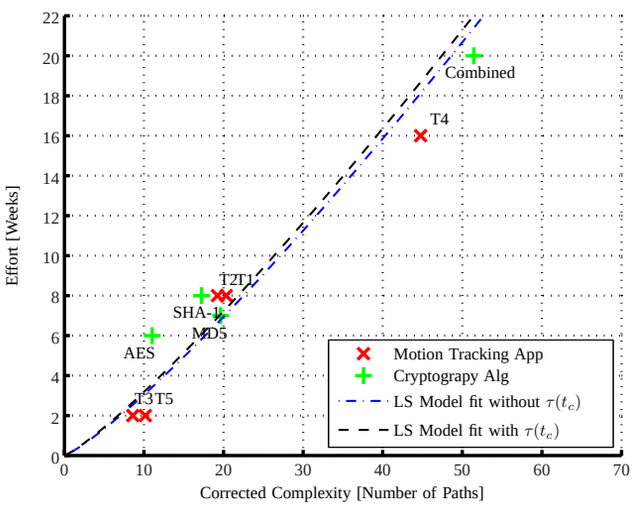


Fig. 10. Relation between the implementation effort [number of weeks] and the complexity corrected according to the designers experience model and the hardness of meeting the real-time constraint.

TABLE III
COMPARISON OF THE DEVELOPMENT TIME AND ESTIMATED DEVELOPMENT TIME FOR THE TWO MODELS MEASURED IN WEEKS.

Algorithm	Original Model Error	New Model Error
T1	0.67	2.19
T2	1.38	3.29
T3	-0.82	-1.56
T4	-2.96	-4.45
T5	-1.53	-3.14
MD5	0.09	0.51
AES	2.57	6.65
SHA-1	2.10	5.28
Combined	-1.40	-2.83
Mean (Variance):	1.50 (3.39)	1.42 (3.07)

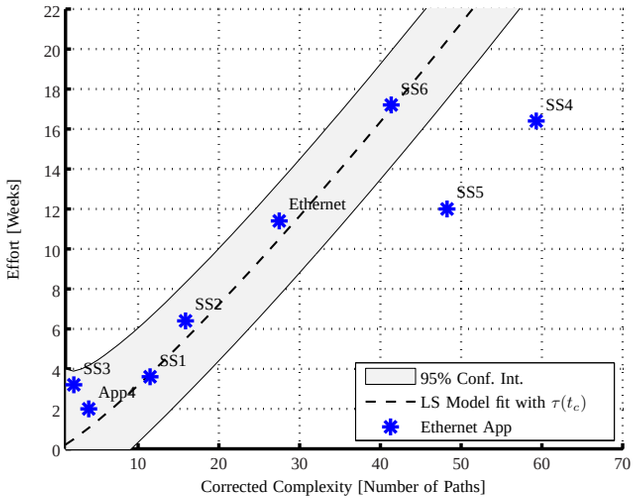


Fig. 11. Validation data plot: relation between implementation effort [number of weeks] and complexity, corrected according to the designers experience model and the hardness of meeting the real-time constraint.

be fulfilled since designers have to identify a or the suitable implementational track(s) that lead to a satisfying solution. We have proposed an optimization model which takes some of the most common optimization techniques into account as well as the order in which they are applied.

In particular we have suggested a set of two metrics which are used to characterise the effects of optimisations. The first one, the implementation hardness metric, reflects how hard it is to reach an implementation satisfying the real-time constraints for the application. The second one, the parallelism distribution metric, reflects how the distribution of parallelism in an algorithm influences the impact of the optimisations.

The experimental results is not unambiguous: for the model the major improvement of the accuracy comes from refining the way the complexity of training data is measured compared to our prior work. A small and not statistical significant improvement comes applying the implementation hardness measure. When validating the model with the validation data,

TABLE IV
COMPARISON OF THE DEVELOPMENT TIME AND PREDICTED DEVELOPMENT TIME MEASURED IN WEEKS.

Algorithm	Estimation Error
SS1	-0.14
SS2	0.91
SS3	2.71
SS4	-9.64
SS5	-8.42
SS6	0.19
Ethernet app	0.90
App 4	0.96
Mean (Variance):	-1.56 (22.02)
Mean without SS4 and SS5 (Variance):	0.92 (0.98)

most of the data approve the model, and fit with it very well. A mean error of 0.92 week (variance 0.98) is achieved, when not considering two outlying data points for which our implementation hardness measure indicate that the time constraint for these algorithms can not me met. Strong algorithm transformation is probably need here and safe conclusion will therefore be that the proposed model will hardly cover these cases.

In order to strengthen the result this work need to be evaluated with more cases. The work would also benefit from making room for other optimisation strategies such as pipelining.

REFERENCES

[1] R. Nass, D. Blaza, and M. Barr, "2009 embedded market study," http://www.techonline.com/learning/livewebinar/216500641.

[2] B. W. Boehm, C. Abts, A. W. Brown, S. Chulani, B. K. Clark, E. Horowitz, R. Madachy, D. Reifer, and B. Steece, *Software Cost Estimation with COCOMO II*. Prentice Hall, 2000.

[3] R. Abildgren, J. Philippe Diguet, P. Bomel, G. Gogniat, P. Koch, and Y. Le Moullec, "A priori implementation effort estimation for hardware design based on independent path analysis," *EURASIP Journal on Embedded Systems*, September 2008.

[4] A. J. Albrecht, "Measuring application development productivity," in *Proc. IBM Applications Development Symp.*, 1979.

[5] T. Jones, *Programming Productivity*. New York: McGraw–Hill, 1986.

[6] D. Jeffery, G. Low, and M. Barnes, "A comparison of function point counting techniques," *Software Engineering, IEEE Transactions on*, vol. 19, no. 5, pp. 529–532, May 1993.

[7] W. Fornaciari, F. Salice, U. Bondi, and E. Magini, "Development cost and size estimation starting from high-level specifications," in *Proceedings of the ninth international symposium on Hardware/software codesign*, 2001, pp. 86–91.

[8] A. Agarwal and R. Shankar, "Cost feasibility analysis for embedded system development and the impact of various methodologies on product development cycle," Florida Atlantic University, Tech. Rep., 2008.

[9] L. Piga and S. Rigo, "Comparing rtl and high-level synthesis methodologies in the design of a theora video decoder ip core," in *Programmable Logic, 2009. SPL. 5th Southern Conference on*, 2009.

[10] Y. Le Moullec, N. B. Amor, J.-P. Diguet, M. Abid, and J.-L. Philippe, "Multi-granularity metrics for the era of strongly personalized SOCs," in *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition*, 2003.