A Method for A Priori Implementation Effort Estimation for Hardware Design

Rasmus Abildgren*, Jean-Philippe Diguet†, Pierre Bomel†, 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 metric-based approach for estimating the hardware implementation effort (in terms of time) for an application in relation to the number of independent paths of its algorithms. We define a metric which exploits the relation between the number of independent paths in an algorithm and the corresponding implementation effort. Furthermore, we complement the metric with a correction function taking the designer's experience into account. Our experimental results show that with the proposed approach it is possible to estimate the hardware implementation effort, and thereby facilitating designers and managers needs for estimating the time-to-market schedule.

1. Introduction

Companies, especially small and medium-sized enterprises (SMEs), designing embedded products are facing major challenges to maintain their competitive power. Time-to-market in particular puts a lot of pressure on these companies and a common problem is to estimate the amount of time required to map and implement an algorithm onto an architecture. A survey [1] indicates that more than 50% of embedded design projects are running behind schedule. A reason is that the project planning involves many parameters that can be difficult to measure. Typical parameters are [2]: Manpower, social relation between team members, the developers commitment to the project, their experience and skills, the suitability and efficiency of the tools, availability of relevant SW/HW IP code/cores, the hardness of design constraints, etc.

In the litterature, many approaches have been proposed for estimating parameters such as area, power, and time. Parameters which are then used in a cost-function for HW/SW partitioning. However, none of these works include the man-power cost which is, very often, the most important parameter for companies.

The work presented in this paper addresses this issue by adding certain aspects of the man-power cost parameter into the cost-function for guiding HW/SW partitioning. More specifically we concentrate on the mapping process, i.e., the process of mapping an algorithm onto an architecture and the implementation effort (i.e., time) related to the complexity of that algorithm.

1.1. Contribution

It have been observed that what makes algorithms complex to implement is related to their number of components and the number of signals/paths. Our hypothesis is that there exists a strong relation between those and the design complexity the engineers are facing. We choose to only measure the number of linear independent paths, and thereby ensure that not only the number of paths is counted but also that a high number of components are present in the measure. Furthermore this ensures that components which occur several times during the execution are counted only once, which represents the actual implementation effort very well.

We explore this idea and propose an original solution for estimating the implementation effort by extending the concept of the cyclomatic complexity measure [3]. Moreover, we propose a more realistic estimation model by including a correction function to take the designer's experience into account.

The remainder of the paper is organized as follows: section 2 gives an overview of the state-of-the-art in methods for estimating the implementation effort and indicates the need for further work. In section 3 a new metric is defined. Then section 4 presents some test cases and analyses the experimental results. Finally we conclude in section 5.

978-1-4244-2315-6/08/$25.00 ©2008 IEEE.

Authorized licensed use limited to: UNIVERSITE DE BRETAGNE SUD. Downloaded on July 28, 2009 at 08:41 from IEEE Xplore. Restrictions apply.

2. State of the art - Effort Estimation

Most research about estimating the implementation effort is found in the software domain, especially within the COCOMO project [4]. The problem of estimating the implementation effort is twofold. First, a reasonable measure needs to be developed for being able to quantify the algorithm. Second, a model need to be developed, describing a rational relation between the measure and the implementation effort. To start with the latter, a typical power model has been proposed inside the COCOMO experiment [4]:

$$Effort = A \times Size^b \quad (1)$$

where $Size$ is an estimate of the project size, and A and b are adjustable parameters. These parameters are influenced by many external factors (see section 1), but can be train, based on previous project data.

To use this COCOMO measure there is a need for expressing the size of the project. A dominating metric in software is lines of code, LOC. HW developers on the other hand tend to dislike this measure, since they do not feel that it is a representative measure for HW, which table 2 also indicates. However, we do not claim that no relation between LOC and the implementation effort exist, e.g. is impossible to write 10k lines in one day, but for VHDL the relation is not always straight forward.

2.1. Function Points Analysis for HW

For making a priori determination of the size of a software project, function points based estimation seems to be more robust than LOC [5]. It was first introduced by Albrecht [6] and have later been improved and modified in may ways. It consists of two main stages: 1) Counting and classifying the function types for the software. 2) Adjusting of the function points according to the application and environment, based on 14 parameters.

To the knowledge of the authors, limited research has been carried out in the field of estimating the implementation difficulty of hardware designs. Fornaciari. et al. [7] have taken up the idea from the function points analysis, and modified it to fit VHDL.

By counting the number of internal I/O signals and components, and classifying these counts into levels, they extract a function point value related to VHDL. They have related their measure to the number of source lines in the LEON-1 processor project and their predictions are within 20% of the real size. However, as stated previously, estimating the size does not always give a good indication of the implementation difficulty, and the necessary implementation time.

By measuring the number of internal I/O signals and components, their work goes along the same road as our initial observations indicate. However our approach is pointing towards estimating implementation effort.

3. Methodology

The proposed flow for estimating the implementation effort is illustrated in Fig 1. It takes its outset in a behavioural description of the algorithm, in C-language, which is intended to be implemented in hardware. From this description, we use our framework to generate a Hierarchical Control Data Flow Graph (HCDFG) which is then measured to identify the number of independent paths. The resulting measure, together with the experience of the developers, gives an estimate of the needed implementation effort.

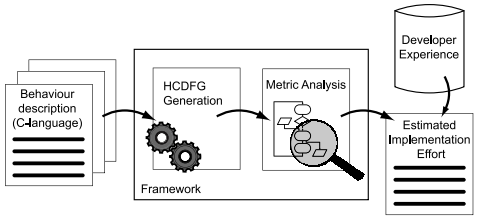

Figure 1: The flow for estimating the needed implementation effort.

3.1. Cyclomatic complexity

As described in section 1.1, the number of independent paths is expected to be correlated with the complexity that the engineers are facing when working on the implementation. A metric measuring that is the cyclomatic complexity measure proposed by Thomas J. McCabe [3]. The number of paths $P(G)$, can simply be expressed as:

$$P(G) = \pi + 1 \quad (2)$$

where π represents the number of condition nodes in the graph G.

In this work we propose an adapted version of the cyclomatic complexity to estimate, a priori to implementation, the number of independent paths in a Hierarchical Control/Data Flow Graph (HCDFG).

Authorized licensed use limited to: UNIVERSITE DE BRETAGNE SUD. Downloaded on July 28, 2009 at 08:41 from IEEE Xplore. Restrictions apply.

3.2. Cyclomatic Complexity on CDFG

The hierarchy of an HCDFG is shown in Fig 2. An HCDFG consist of nodes, n, which can represent other HCDFGs, Control/Data Flow Graphs (CDFGs) and Data Flow Graphs (DFGs) as well as elementary nodes (processing, memory, and control nodes). The nodes are all connected via dependency edges.

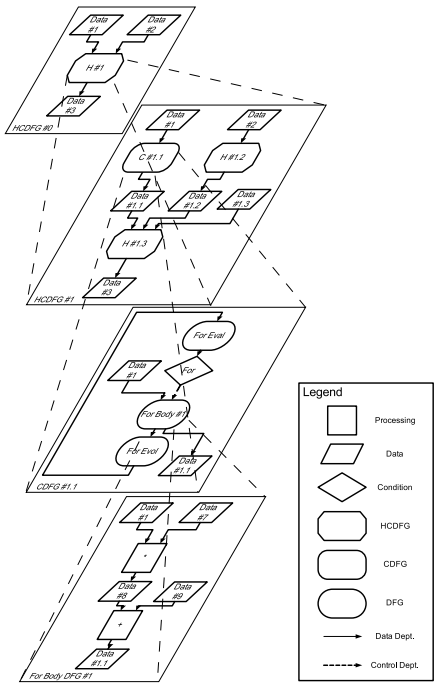

Figure 2: An overview of how the hierarchy in a HCDFG allows analysis of an algorithm on different levels and how the levels are related.

To calculate the cyclomatic complexity, (2) gives that we only need to focus on the CDFGs and HCDFGs. For these graphs, a set of equations describing how to measure the number of independent paths is deduced. These are summarised in table 1.

3.3. Experience impact

The experience of the designer has an impact on the difficulty of tackling the challenge that he/she is facing when developing a system. Experience is influenced by many parameters, but in this work we only focus on the time the developers have worked with the implementation language and the target architecture.

In the literature, e.g. [8], experience curves are obtained based on historical data. From these studies we can observe that the shape of the curves describing the effect of the experience tend to have a negative accelerating slope, of power or logarithmic nature.

Table 1: The equations to calculate the number of independent paths for the different graph types.

Graph	Path equation
If	$P(n_{if})=P(n_{true})+P(n_{false})+P(n_{eval})-1$
Switch	$P(n_{switch})=P(n_{eval})-1+\sum_{i=1}^{N} P(n_{case_i})$
For	$P(n_{for})=P(n_{for-body})+P(n_{eval})+P(n_{evol})-2$
While	$P(n_{while})=P(n_{while-body})+P(n_{eval})-1$
Fct.	$P(n_{HCDFG_{function}})=\begin{cases}0 & \text{if reuse}\\ P(n_{HCDFG}) & \text{else}\end{cases}$
Parallel	$P(n_{HCDFG_{Parallel}})=\sum_{i=1}^{N} P(n_{HCDFG_i})$
Serial	$P(n_{HCDFG_{Serial}})=\sum_{i=1}^{N} P(n_{HCDFG_i})-(N-1)$

In order to get the best possible outset for predicting the implementation effort, it is of great importance to have good knowledge about the individual experience curve of the developers. Using this method, the curve could possibly be refined after each completed implementation. However, this has not been the purpose of this work[1], and therefore we have, for the experiments in this study, selected the following model:

$$\eta_{experience}(Dev) = \frac{1}{\alpha \log(Experience(Dev)+\beta)} \qquad (3)$$

where α and β are trim parameters which can be used to optimise the curve to fit reality, $Experience$ is the number of weeks which the developer, Dev, has worked with the language and architecture, and set the parameters $\alpha = 1$ and $\beta = 1$. Fig 3 depicts the shape the experience model.

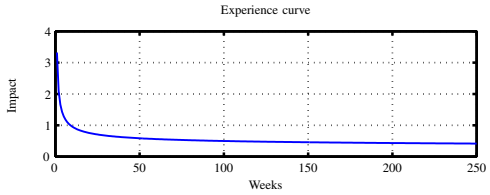

Figure 3: How the lack of experience impacts on the difficulty the engineers are facing.

4. Results

In order to verify the hypothesis, a classical test has been conducted. The test is dual phased and consists of: i) a training phase, using a first set of real-life data, during which the hypothesis is said to be true, and ii) a validation phase during which a second set of real-life

1. This will be addressed in future work.

Authorized licensed use limited to: UNIVERSITE DE BRETAGNE SUD. Downloaded on July 28, 2009 at 08:41 from IEEE Xplore. Restrictions apply.

data is used to evaluate whether the hypothesis holds true or not.

4.1. Phase One – Training

The real-life data used as training data originate from two different application types that are both developed as academic projects in universities in France. The first application is composed of five different video processing algorithms for an intelligent camera which is able to track moving objects in a video sequence. The second application is a cryptographic system, able to encrypt data with different cryptographic/hashing algorithms, i.e., MD5, AES and SHA-1. The system consists of one combined engine [9] as well as individual implementations. These projects have been selected since they all follow the same methodology, using a behavioral specification in C, as a starting point for the VHDL implementation. A common point for all these data is that none of the developers have created the behavioral specification in C: for the cryptographic algorithms the behavioral specification comes from the standards, and the video algorithms were based on a previous project.

Using the behavioral description as a starting point, the experiment consists in studying the relation between the complexity (as defined in section 3) of the algorithms and the implementation effort (i.e., time) required to implement them in VHDL (including testbed and heuristic tests).

The developers involved in these projects have all been Master and Ph.D. students starting out with an electrical engineering background but no other VHDL background than the one they have obtained doing their studies, see table 3. All developers are taught VHDL by other instructors than the authors, but inside our university. Table 4 summaries the training data.

Table 3: 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

Fig 4 shows the relation between the implementation effort and the measured complexity for the individual algorithms. Please note that in this graph, the complexity values are not corrected for the designers' experience.

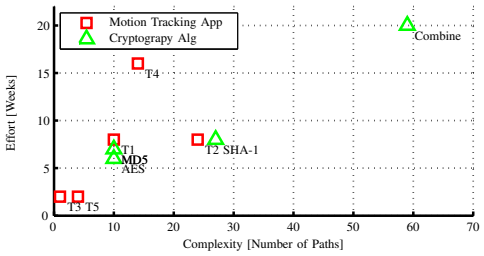

Figure 4: Relation between the implementation effort [weeks] and the uncorrected complexity (as defined in section 3).

A first examination of the data points indicates a possible relation between some of them; however many other points are located far away from any relation. These data are uncorrected for the designers experience and, as earlier mentioned, we strongly believe that the experience of the individual designer has a non-negligible influence on the development time. If we consider the data with more attention, it is clear that the points diverging most are those of the implementations where the developers had very limited knowledge and experience with the VHDL language.

Table 4: Training Data (Top) and Validation Data (Bottom). Algorithms are related to the developers and their experience at the given time. At this stage complexity is still uncorrected.

Algorithm	Complexity	Developer	Dev. Exp.
T1	10	Dev 1	2
T2	24	Dev 1	10
T3	12	Dev 1	18
T4	14	Dev 2	1
T5	4	Dev 1	20
MD5	10	Dev 3	1
MD5	10	Dev 4	1
AES	10	Dev 4	8
SHA-1	27	Dev 4	14
Combined	59	Dev 4	14
SS1	25	Dev 6,7	150
SS2	35	Dev 5	150
SS3	17	Dev 5,6,7,8	150
SS4	50	Dev 6	6
SS5	29	Dev 7	3
SS6	25	Dev 5,6,7	3
Ethernet app	60	Dev 5,6,7,8,9	150
App 4	9	Dev 6	150

Authorized licensed use limited to: UNIVERSITE DE BRETAGNE SUD. Downloaded on July 28, 2009 at 08:41 from IEEE Xplore. Restrictions apply.

Table 2: Line of code, area, and time constraints for the validation data.

Algorithm	SS1	SS2	SS3	SS4	SS5	SS6	Ethernet	App 4
Dev. Time [weeks]	3.6	6.4	2.4	16.4	12	17.2	16	2
LOC-VHDL	994	1195	776	1695	760	2088	3973	232
Slices	564	2212	382	888	372	2171	3372	750
FlipFlops	913	2921	1290	1366	1208	2077	6149	942
LUTs	997	3157	6453	1569	6443	3458	18255	567
Time Constraint. [ns]	112	128	360	112	360	248	696	56

Applying the proposed (3) (non-linear) experience transform onto the data results in a significantly different picture, as depicted in Fig 5. Now there is a clear trend in the plotted data toward a relation. From the COCOMO II project [4] it is known that the relationship between the implementation time and the complexity measure (in their case lines of code, LOC) can be expressed as a power function with a weak slope. We showed its nature in (1), and with correction for experience it looks like:

$$Effort = A \times \eta_{experience}(Dev) \times P(n_{HCDFG_{Alg}})^{b} \quad (4)$$

The parameters A and b are, found via a least square (LS) fit on our training data, to be $A = 0.226$ and $b = 1.103$. In Fig 5 the dashed line illustrates the relationship, with the above given parameters.

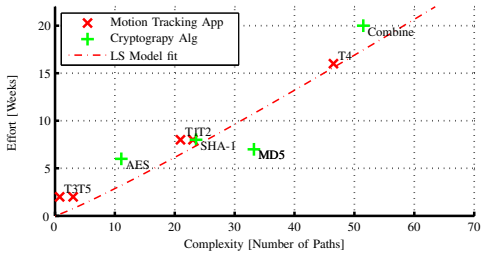

Figure 5: The corrected training data together with the proposed effort model.

4.2. Phase two – Validation

After having elaborated a model based on the training data, we proceed with the validation of its correctness. For this, a new set of data provided by ETI A/S, a Danish SME, is used. The dataset originates from a networking system and consists of Ethernet applications which have been implemented on an FPGA, as well as corresponding testbeds. This Ethernet application is part of an already existing system with which it needs to interact. Table 2 shows additional implementation information about these applications. The system is real-time with hard time-constraints and all algorithms have been implemented so they meat these constraints. Similarly to the training data, the development flow for this application has been as follows: a behavioral C++ model of the application has been constructed before the VHDL-based implementation on the FPGA architecture. The behavioral model have been developed by other developers than the ones doing the implementation. The developers taking care of the implementation have obtained their skills in VHDL from a professional course, which have no relation to our university in Denmark.

The time spent for the implementation process covers: the design and implementation of the VHDL code of the functionalities and testbed as well as the tests of the different modules in the applications. These data are shown in the lower part of table 4. The time data originate from the companies internal registration system for the project and therefore corresponds to the effective time used.

The relation between the implementation effort and the complexity is plotted in Fig 6. It can be seen that these data, corrected for the designers experience, follow the model derived from the training data well (dashed line). Fig 6 also shows the 95% confidence interval, which indicates that with 95% confidence, future predictions of the implementation effort will lay inside this, given that the model holds.

Relating the predicted effort (dashed line in Fig 6) with the real effort (*), we see that some estimation errors occur. The average estimation error is 0.2 weeks with a variance of 8. In the next section we discuss the validity of the model.

4.3. Discussion

The trend identified in the training data holds very well for the validation data too: they both fall very much in line with the underlying rationale, and we are quite confident about the strength of the proposed model. The model shows it generality by being versa-

Authorized licensed use limited to: UNIVERSITE DE BRETAGNE SUD. Downloaded on July 28, 2009 at 08:41 from IEEE Xplore. Restrictions apply.

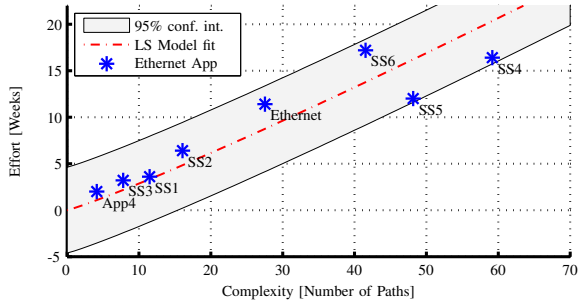

Figure 6: Validation data plot: relation between the implementation effort [number of weeks] and the complexity corrected according to the designers experience model.

tile in the number of application domains, but also in the variety and number of developers it covers.

The results clearly shows the need for the proposed correction function and also that the proposed logarithmic nature works well, even though the correction function has not been trimmed to fit the individual developers due to the lack of available data. In that light our approach must be seen as the engine of a global methodology for the management of design project, which imposes a systematic registration of man-power. With such a registration, a database of the developers experience can easily be constructed, and the correction function can be trimmed to fit the companies individual designers. Several iterations of this process would converge toward a more precise estimation of the implementation effort.

The limited set of data on which the model is built also limits the complexity window on which this model can be applied: having no algorithm for which the corrected complexity value is larger than 51, extrapolating the model further would weaken the current conclusion. More data, from larger and more varied projects, would allow a more refined model.

Nevertheless, the results described in this paper are very encouraging with all the real-life cases that we have examined and we are quite confident that this model can easily be applied for other types of applications.

5. Conclusion

The contribution presented in this paper is a metric-based approach for estimating the needed time for hardware implementation in relation to the complexity of an algorithm. We have deduced that a strong relation between the number of independent paths in the algorithm and the corresponding implementation effort exists. We have proposed an original solution for estimating the implementation effort, that extends the concept of the cyclomatic complexity.

To further improve our solution, we developed a more realistic estimation model, which includes a correction function to take the designer's experience into account.

We have implemented this solution in our tool Design-Trotter, of which the input is a behavioral description in C language and output is the number of independent paths. Based on this output and the proposed model, we are able to predict the needed implementation effort. Our experimental results, using industrial Ethernet applications confirmed that the data, corrected for the designers' experiences, follow the derived model very well and that all data fall inside its 95% confidence interval. This pave a way for an implementation effort estimator of which the accuracy iteratively improves after each project.

References

[1] R. Nass, "An insider's view of the 2008 embedded market study," Embedded.com – CMP Media, Tech. Rep., 2008, http://www.embedded.com/products/softwaretools/210200580.

[2] S. McConnell, *Software Estimation – Demystifying the Black Art*. Wasthington: Microsoft Press, 2006.

[3] T. J. McCabe, "A complexity measure," *IEEE Transaction on Software Engineering*, vol. SE-2, no. 4, pp. 308–320, December 1976.

[4] 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.

[5] G. Low and D. Jeffery, "Function points in the estimation and evaluation of the software process," *Software Engineering, IEEE Transactions on*, vol. 16, no. 1, pp. 64–71, January 1990.

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

[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. Heathcote, S. Brown, and D. J. K. Mewhort, "The power law repealed: The case for an exponential law of practice." *Psychonomic Bulletin & Review.*, 2000.

[9] S. Ducloyer, R. Vaslin, G. Gogniat, and E. Wanderley, "Hardware implementation of a multi-mode hash architecture for MD5, SHA-1 and SHA-2," in *Proceedings on the Design and Architectures for Signal and Image Processing 2007 Workshop*, 2007.

Authorized licensed use limited to: UNIVERSITE DE BRETAGNE SUD. Downloaded on July 28, 2009 at 08:41 from IEEE Xplore. Restrictions apply.