Accurate Cost Estimation of Memory Systems Inspired by Machine Learning for Computer Vision

Lorenzo Servadei[1,2], Elena Zennaro[1],
Keerthikumara Devarajegowda[1,3], Martin Manzinger[1,4], Wolfgang Ecker[1,4], Robert Wille[1,2]
[1]Infineon Technologies AG – [2]Johannes Kepler University Linz – [3]TU Kaiserslautern - [4]TU Munich
Email: name.surname@infineon.com robert.wille@jku.at

***Abstract*—Hardware/software co-designs are usually defined at high levels of abstractions at the beginning of the design process in order to allow plenty of options how to eventually realize a system. This allows for design exploration which in turn heavily relies on knowing the costs of different design configurations (with respect to hardware usage as well as firmware metrics). To this end, methods for *cost estimation* are frequently applied in industrial practice. However, currently used methods for cost estimation oversimplify the problem and ignore important features – leading to estimates which are far off from the real values. In this work, we address this problem for memory systems. To this end, we borrow and re-adapt solutions based on *Machine Learning* (ML) which have been found suitable for problems from the domain of *Computer Vision* (CV) – in particular age determination of persons depicted in images. We show that, for an ML approach, age determination from the CV domain is actually very similar to cost estimation of a memory system.**

I. Introduction

Increasing the productivity in industrial hardware/software co-designs is a central issue, as it allows for an efficient design and testing as well as a reduction of the design costs. In this context, utilizing automation already in early stages of the design process is essential for supporting the designer in handling the complexity of modern chips. Among different automation approaches, automated hardware generation and automated firmware generation from model-based designs got established in industrial practice today [3], [5]. Here, the desired system is specified on an abstract level (e.g. by means of design-centric meta-models) and, afterwards, depending on the respectively applied design configuration, realized.

This allows for a design exploration at early stages of the design process, since plenty of options how to realize a system can be evaluated prior to its actual implementation. By this, the designer can make sure that a system is realized in the respectively needed customized fashion, but also satisfies certain cost constraints (e.g. with respect to area, number of firmware instructions, etc.). In order to evaluate the different possible design configurations, methods for *cost estimation* are essential. They take properties from the design configuration and, based on that, try to extrapolate what costs an implementation realizing those configuration eventually would have. This information is then utilized by the designer to eventually decide which configuration shall be realized. Within this process, the quality of the cost estimation obviously is a crucial criteria – misleading cost estimates will eventually lead to the implementation of design configurations which likely is not satisfactory. In this work, we consider this issue for the design of memory systems.

Here, methods for cost estimations currently applied in industrial practice thus far [7], [10], [12] actually oversimplify the problem and ignore important features that heavily affect the cost. Consequently, they frequently lead to cost estimates that are far off from the real values (this is discussed in more detail later in Section II). At the same time, we observe that explicitly recognizing those features and "hard-code" an algorithm considering all features in order to derive a more accurate estimation is a cumbersome task. Hence, we propose an alternative solution which borrows and re-adapts concepts based on *Machine Learning* (ML, [6]) which have been found suitable for problems from the domain of *Computer Vision* (CV, [1]). More precisely, we observe that, for typical CV problems such as the determination of the age of a person depicted in an image, ML indeed properly recognizes the respective features and is capable of determining rather accurate estimates. Further, we show that age determination in CV and cost estimation of a memory system share some similarities and can actually be addressed by the same scheme. Based on that, we propose an alternative solution for cost estimation which adapts the concepts from the CV domain. The proposed method offers higher scalability and flexibility since further features can directly and automatically learned through the ML training process.

II. Cost Estimation for Memory Systems

In this section, we illustrate the problems and challenges of cost estimation for memory systems in early stages of the design flow. To this end, we first introduce the specification of *Register Interface* (RI) components such as presented in [4] which serve as running example in the remainder of this work. Using this example, today's shortcomings of cost estimation are illustrated – providing the motivation of this work.

RIs are common components in an SoC and provide the communication mechanism between the core and peripheral devices. Accordingly, RI components may be required in different *configurations* (e.g. depending on the respectively applied peripherals) which is why the initial specification is usually provided generically in terms of a meta-model such as follows:

Example 1. *Fig. 1 shows the meta-model of the considered RI component. Here, sub-components are defined that describe the HW and FW structure of the RI. The sub-component* Interface *specifies the general features of the RI such as the* DataWidth *or the* AddressWidth. *The single registers are defined using instances of the sub-component* Unit, *which has a* Name, *a* Size, *and an* Address. *The accessibility to each bitwise position in the registers/*Units *is defined by dedicated bitfields, which is why each* Unit *has one or more instances of a corresponding sub-component* Contained *(specifying the start* Position *of a bitfield) and the sub-component* Bitfield *(specifying e.g. the* Size *and the allowed access; e.g.* HwRd *and* SwWr *regulating the property being read or written respectively by HW and FW). Finally, the corresponding FW (which eventually has to obey these accessibility settings) is described using the additional sub-components* Configuration, Setting, DontCareList, *and* Parameter *which are specified depending on the desired instances of the* Bitfield*s.*

Using this meta-model, the designer can now instantiate various configurations of an RI component. In the following, we call these instances *design configurations*. These configurations have to be chosen so that they, on the one hand, realize

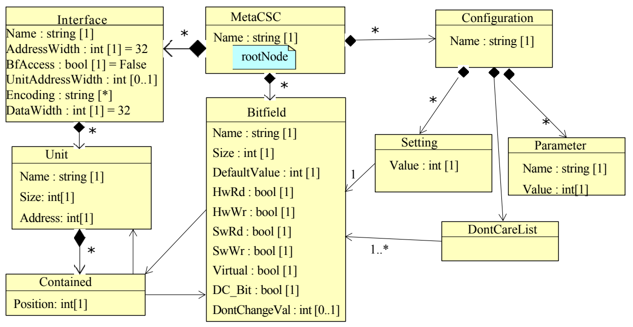


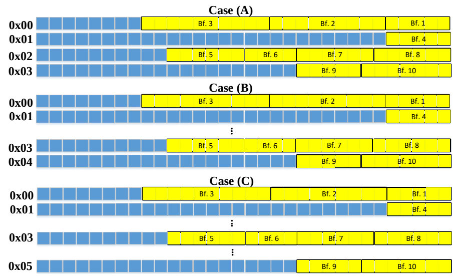
Figure 1: Meta-model of the *Register Interface* (RI)

the intended behavior (e.g. allow for proper access for the peripherals) but, on the other hand, also satisfy certain cost constraints which may exist. For the latter, it is important to have an accurate estimation of the costs for a correspondingly considered configuration. In the following, we exemplary consider thereby

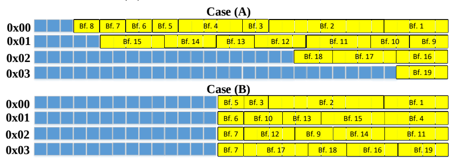
- the area, i.e. the number of *Configurable Logic Blocks* (CLBs) in terms of *Logic Units* (LUTs) and *Slice Registers* (SRs) which are needed to realize the configuration on an FPGA board, and
- the size of the generated binary FW code (FS) as well as the number of cycles of a pipelined CPU which are needed to execute the FW program (FCs).

However, in early stages of the design flow, the designer has no fully-fledged implementation of the respective design configurations at hand (they are still to be implemented). Accordingly, the respectively resulting costs need to be estimated in order to evaluate different design choices and, eventually, decide which indeed shall be realized. High-Level Synthesis Tools can be utilized for this purpose [8], and have as well been employed for HW/SW co-design of SoC FPGAs [11]. In our industrial context, with the purpose of learning a fast and accurate estimation for a whole set of designs, we utilize ML approaches. To this end, several methods using ML algorithms for *cost estimation* have been proposed in the past. They use e.g. coarse-grained inputs (e.g. means and aggregate values) as proposed in [7], [10], [12]. More precisely, in [12], the features used for CLBs estimation are aggregated for each design generation, resulting in a coarse grained feature space used for the prediction of the SoC area objective. The work [10] instead evaluates the power consumption of different algorithms using high-level features taken from single algorithmic blocks, such as *average working set size* and *total operations*. These are used for predicting power and performance of an FPGA-based soft processor. Finally, in [7], different features (here presented as tunable knobs) are evaluated as input for the multi-objective estimation problem, e.g. the *Num. Work Items per group*, the *Num. Work Items per group* and *Num. Private Variables*. However, with respect to memory systems, these approaches oversimplify the estimation and ignore important further information which also may affect the final costs.

In fact, spatial information such as the spatial position of the bitfields and units inside the memory system have a significant impact on the eventual costs and scalability of the estimation. Each bitfield property is set in a configuration context – representing features as a total or mean of values. If provided on a too higher (imprecise) level, this may lead to a diminishing of specific information that affect the real cost. At the same time, the complexity of considering those spatial information increases with the size of the considered systems – yielding either a significantly increasing manual effort for



(a) Different addresses



(b) Different bitfields mapping

Figure 2: Design configurations from the RI meta-model

the retrieval of feature values or making it even impossible at all.

Example 2. *Consider the instances of the RI meta-model from Fig. 1 as shown in Fig. 2. Here, Fig. 2a shows three different instances which, at a first glance, look identical (e.g. in all three cases the bitfields are the same, and are instantiated in an identical fashion). In fact, the only difference is that the Units have different* Addresses. *At a first glance, this should not cause significant differences in the costs. Accordingly, existing methods for cost estimation methods yield an estimation of 110 LUTs, 36 SRs, a FS of 2.6 Kb, and 564 FCs for* all *cases. However, if we evaluate the HW Area and retrieve the FW metrics from the actual implementation of those instances*[1]*, we obtain different values for each case. In fact, as discussed above, the spatial information of each unit indeed has an impact which is why e.g. the realization of Case (a) from Fig. 2a eventually costs 83 LUTs, 42 SRs, a FS of 2.2 Kb, and 524 FCs. The realization of Case (b) from Fig. 2a eventually costs 117 LUTs, 42 SRs, a FS of 2.2 Kb, and 524 FCs. Finally, in Case (c) of Fig. 2a, the costs are 156 LUTs, 42 SRs, a FS of 2.2 Kb, and 524 FCs.*

These differences between estimations and real costs become even more substantial if variations on the spatial position of bitfields inside the registers occur as illustrated e.g. by the different cases shown in Fig. 2b. Here, exactly the same bitfields are applied in both cases – only their distribution amongst the Units/*registers, i.e. their spatial distribution, is different. Accordingly, state-of-the-art cost estimation methods extrapolate the* same *costs for* both *cases, namely 89 LUTs, 97 SRs, a FS of 2.0 Kb, and 476 FCs. However, as discussed above, also here spatial position of the bitfields has an impact to the costs which is why the actual costs for Case (a) are 97 LUTs, 89 SRs, FS of 2.2 Kb, and 576 FCs, while for Case (b), they are 79 LUTs, 89 SRs, FS of 1.4 Kb, and 388 FCs.*

These examples illustrate that both, with respect to actual but also relative values between the cases, previously proposed cost estimations are far off and actually lead to rather misleading results – a serious problem when it comes to finally decide which design configuration shall be realized. At the same time, this motivates the development of more precise cost estimation methods which additionally consider the further characteristics discussed above. However, considering all possible features relevant for cost estimation is a cumbersome task which

[1]In the (industrial) setting considered here, we used the commercial *Vivado Design Suite* by Xilinx to explicitly implement the given instances.

cannot easily be incorporated e.g. by simply adding additional features. Because of that, we are proposing a complementary approach which is described in the next section.

III. Proposed Solution

In this work, we address the problem of cost estimation sketched above. To this end, we borrow concepts from *Machine Learning* (ML) as well as *Computer Vision* (CV). We show that existing methods e.g. to process images and videos based on ML can be adapted for the purpose of cost estimation. In the following, we describe the proposed methodology as follows: We first review data preparation and basic approaches of ML methods used today for CV tasks. Afterwards, we describe how the features which affect the costs of a memory system can be represented in a similar fashion than pictures for CV algorithms.

A. Machine Learning for Computer Vision

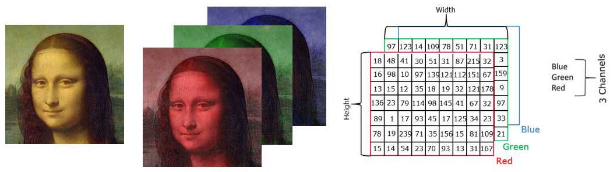
Computer Vision (CV, [1]) is an interdisciplinary area which is concerned with the computational analysis and understanding of single images or sequence of them (i.e. videos). Typically CV requires a particular processing of the image signal, so that specific tasks could be performed (e.g. object detection, classification, age estimation, etc.). Originally, the state-of-the-art methods to accomplish such tasks heavily relied on manual labor, i.e. features were often extracted manually from image pixels before they are processed further [9]. Nowadays, through the availability of more sophisticated ML algorithms, pixel-based representations are often chosen as a direct input to the ML algorithms [2], [6]

. In fact, through an extended use of *Neural Networks* (NNs, [2], [6]) (an ML algorithm which is loosely inspired by biological neurons), high accuracy in CV tasks can be reached by taking raw images as input and compute features automatically within the learning phase of the ML process [2], [6]. For this reason, the representation of images becomes a central issue in CV.

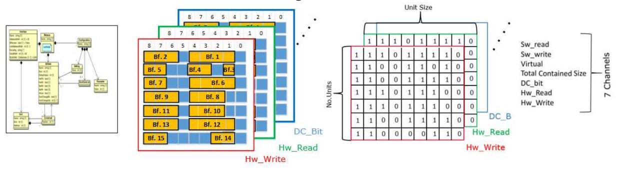
In the following, we utilize the common representation of images in terms of a function $f(h, w, c)$, where h, w, c are coordinates of a 3D matrix $\boldsymbol{\mathcal{I}}^{H\times W\times C}$. Here, H is the height of the image, W the width of the image, and C the number of so-called *channels* of this image. All entries of the 3D matrix (and, hence, all functional values of $f(h, w, c)$) define the *intensity value* v for the respective position and channel. In other words, for a gray scale image, the matrix has $C = 1$ channel where each matrix entry represents the respective gray scale at the corresponding position. Similarly, for an RGB image, the 3D matrix has $C = 3$ channels (one for each of the colors red, green, and blue) where each 3D matrix entries represents the respective portion of these three colors at the corresponding position. For RGB images, the value v is usually within $v \in [0, 255]$ for each one of the three channels.

Example 3. *Consider the image shown on the left-hand side of Fig. 3a. Following a typical RGB structure, this image can be defined as the combination of the corresponding red, green, and blue portions as sketched in the center of Fig. 3a. This in turn can be represented in terms of a function $f(h, w, c)$ with a total of $C = 3$ channels, i.e. as a 3D matrix $\boldsymbol{\mathcal{I}}^{H\times W\times 3}$, where H is the pixel-wise height of the RGB image and W is the pixel-wise width. Each one of the channels is therefore structured as a $\boldsymbol{\mathcal{O}}^{H\times W}$ 2D matrix with one pixel value v at each coordinate.*

Using this representation, CV tasks such as object detection, classification, segmentation can now be conducted using ML algorithms [2], [6]. For example, let's take a CV task where,



(a) Images RGB



(b) Hardware Images

Figure 3: Representation for CV and cost estimation tasks

given a RGB image of a person, we want to estimate the his/her age. To this end, several parts of the human body shown in the image may indicate the age, e.g. wrinkles in the skin, color of the hair, etc. However, it obviously is rather difficult to explicitly "hard-code" a computer program which reliably reveals these useful features (e.g. because of the variety in the images set: some images might be zoomed, some badly/differently illuminated, some might be occluded, etc.). Here, ML can provide valuable support.

More precisely, the task is handled as a regression problem (which is suitable, since the desired output should be a number indicating the person's age) where first a *training set* of images is provided for which the respective age of the person is known. Using this training set, the ML algorithm can "learn" all the features mentioned above not in an explicit (i.e. "hard-coded") fashion, but through associations with the image and the respectively given age (to this end, meaningful patterns in the image, the distribution of pixels, and the linked labels guide the ML algorithm to outpoint the estimated age of the person). The pixels of the image are thereby algorithmically processed in order to establish a consistent association of the given image and the given age.

B. Corresponding Representation for a Memory System

Now, recall that, as discussed and illustrated in Section II, current methods for cost estimation suffer from the fact that they do not properly consider all features that might affect the costs. At the same time, "hard-coding" a computer program which reliably reveals these useful features is hard as well (because of the same reasons why it is hard in CV to determine the age of a person). However, the same method applied for age determination in the domain of CV can also be applied in the domain of cost estimation for memory systems. To this end, we just need corresponding representations for the domain considered here. This is introduced in this section.

More precisely, rather than a RGB image (e.g. providing the pixel intensity v for each coordinate and channel), a similar data-structure for the considered memory system (e.g. providing bitfield positions, bitfield properties, etc.) is required. Having such a representation, the same solutions can be used as already successfully utilized in the CV domain. In the following, we describe the proposed data-structures – one for the hardware and one for the firmware.

The hardware data-structure (called *HW image* in the following) represents the respective properties of the memory system in terms of a function $j(q, l, b)$ where q, l, b are coordinates of a 3D matrix $\boldsymbol{\mathcal{A}}^{Q\times L\times B}$. In contrast to CV, Q now defines the number of Units/registers in the memory system, while L defines the respective bit-width and B defines the number of bitfield properties considered. Instead of three channels as in the CV domain (for red, green, and blue), we now use a total of seven channels to represent the HW features. The channels indicate whether the corresponding position in the memory system contains a bitfield (*Bfs*), and if the bitfield allows for a hardware write (*HwWr*), hardware read (*HwRd*), software write (*SwWr*), software read (*SwRd*), a don't change bit (*DC_Bit*), or a virtual bit (*Virtual*). Since these properties either hold for a position in a bitfield or not, they can easily be structured in a binary representation, i.e. each entry of the 2D representation matrix is now a binary value where 1 denotes that the position of the bitfield has the considered property, whereas the value 0 is inserted elsewhere. This results in a *feature representation*, which preserves the spatial structure of the RI for a specific bitfield property and, at the same time, allows a simple addition or removal of features to the hardware data-structure. Furthermore, the data-structure captures aggregate values and statistics among features, which do not need to be explicitly specified. All the above-mentioned traits address the desired characteristics for the cost estimation of memory systems.

Example 4. *Consider again the instance of a memory system as shown in the top of Fig. 2a and discussed in Example 2. Those properties are usually represented in terms of a .uml file as sketched in left-hand side of Fig. 3b. Overall, this yields properties with respect to hardware write, hardware read, software write, etc. as sketched in the center of Fig. 3b. Those are eventually represented in terms of matrices as sketched in the right-hand side of Fig. 3b.*

Using this data-structure, the HW costs of a memory system can be determined similarly to the age determination in the CV-domain discussed before. That is, first a training set of memory system instances is provided for which the respective real area costs are available. This is used to "learn" the respective features relations and an accurate information processing. Afterwards, proper estimations for the instances for which we actually want to determine the costs can be obtained.

Next, in order to obtain the cost for firmware operations, the corresponding data-structure becomes a bit more challenging. Here, the sequence of reading/writing operations specifies which bitfield/s should be read or written, from/in which register, at each point of the sequence. This is particularly interesting for a FW cost estimation of the RI, as the type of writing operation performed (i.e. the no. of instructions needed) depends on the affected register configuration. Furthermore, distinct types of writing operations have a different impact on the FW size and firmware cycles for a particular design.

In order to outline a corresponding date-structure for the firmware, we get inspiration once again from the CV domain. The evaluation of the firmware cost can be thought in fact as the same estimation of the age of a person through a multi-channel video. This gives in fact a further dimension, so that we can observe different angles on the wrinkles or a person´s walking posture over time. In order to adapt this approach for firmware cost estimation, we transform once again the writing and reading operations, together with other RI properties, into binary representations. These correspond to the features representations at each point of the sequence (i.e. frames).

This results in a firmware data-structure (called *FW sequence* in the following) covering the properties of the memory system in terms of a function $z(q, l, d, p)$, where q, l, d, p are coordinates of a 4D matrix $\mathcal{F}^{Q \times L \times D \times P}$. Again, Q defines the number of Units/registers in the memory system, while L defines the respective bit-width and D defines the number of considered bitfield properties. That is, in the case of the firmware structure, we have six different channels where each channel is denoted by $\mathcal{F}_d$ with $d \in [1, 6]$. Besides that, P now additionally defines the number of frames and, by this, incorporates the time aspect ($p \in [1, P]$). Following this, each binary feature representation $\mathcal{F}_1^p$ contains the value 1 in the bitwise position of registers containing bitfields to be written; the same approach is applied to the reading operation ($\mathcal{F}_2^p$). Both of these information are contained in the property *Configuration* of the RI meta-model. A further representation is added for the property *DontCareList* ($\mathcal{F}_3^p$). Once the writing and reading operations, as well as the *DontCareList*, are structured in the corresponding binary representations, they are stacked at the same sequence frame $\mathcal{F}^p$. This composes the dynamic part of the data-structure for the firmware estimation. Finally, three further representations are added to the same frame, corresponding to a static representation of the *HwRd*-feature ($\mathcal{F}_4^p$) and the *DC_Bit*-feature ($\mathcal{F}_5^p$) as well as a representation of the total Bitfields ($\mathcal{F}_6^p$) of the design. These values do not change from frame to frame and, hence, are implemented as repeated static representations at each point $\mathcal{F}^p$ of the sequence. Using this representation, an ML algorithm is equipped with all the information related to firmware and can conduct the cost estimation in a similar fashion as sketched above for the hardware image.

IV. Conclusion and Further Work

In this paper, we present an approach for an accurate cost estimation of Memory Systems, inspired by Machine Learning for Computer Vision. The representations discussed herein apply to the hardware as well as to the firmware configurations. As a further work, we are going to apply standard ML algorithms on the representations obtained for a cost estimation. In particular, we propose to implement 2D CNNs for the Hardware and 3D CNNs for the Firmware cost estimation.

References

[1] Dana Harry Ballard and Christopher M. Brown. *Computer Vision*. Prentice Hall Professional Technical Reference, 1st edition, 1982.

[2] Roberto Cipolla, Sebastiano Battiato, Giovanni Maria Farinella, et al. *Machine Learning for Computer Vision*, volume 5. Springer, 2013.

[3] S. H. M. Durand and V. Bonato. A tool to support Bluespec SystemVerilog coding based on UML diagrams. In *IEEE Industrial Electronics Society*, pages 4670–4675, Oct 2012.

[4] Wolfgang Ecker, Wolfgang Mueller, and Rainer Doemer. *Hardware-dependent Software: Principles and Practice*. Springer Publishing Company, Incorporated, 1st edition, 2009.

[5] T. Farkas, C. Neumann, and A. Hinnerichs. An Integrative Approach for Embedded Software Design with UML and Simulink. In *International Computer Software and Applications Conference*, pages 516–521, 2009.

[6] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[7] Pingfan Meng, Alric Althoff, Quentin Gautier, and Ryan Kastner. Adaptive threshold non-pareto elimination: Re-thinking machine learning for system level design space exploration on FPGAs. In *Design, Automation & Test in Europe*, pages 918–923, 2016.

[8] R. Nane, V. Sima, C. Pilato, J. Choi, B. Fort, A. Canis, Y. T. Chen, H. Hsiao, S. Brown, F. Ferrandi, J. Anderson, and K. Bertels. A Survey and Evaluation of FPGA High-Level Synthesis Tools. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2016.

[9] Mark Nixon and Alberto S. Aguado. *Feature Extraction & Image Processing for Computer Vision, Third Edition*. Academic Press, Inc., Orlando, FL, USA, 3rd edition, 2012.

[10] Adam Powell, Christos Savvas-Bouganis, and Peter Y. K. Cheung. High-level Power and Performance Estimation of FPGA-based Soft Processors and Its Application to Design Space Exploration. *Journal of Systems Architecture*, 59(10):1144–1156, November 2013.

[11] Streit, Franz-Josef and Letras, Martin and Schid, Matthias and Falk, Joachim and Wildermann, Stefan and Teich, Jürgen. High-level synthesis for hardware/software co-design of distributed smart camera systems. In *International Conference on Distributed Smart Cameras*, 2017.

[12] Elena Zennaro, Lorenzo Servadei, Keerthikumara Devarajegowda, and Wolfgang Ecker. A Machine Learning Approach for Area Prediction of Hardware Designs from Abstract Specifications. In *Euromicro Conference on Digital System Design*, 2018.