Accurate Cost Estimation of Memory Systems

Utilizing Machine Learning and Solutions from Computer Vision for Design Automation

Lorenzo Servadei, Edoardo Mosca, Elena Zennaro, Keerthikumara Devarajegowda, Michael Werner, Wolfgang Ecker, *Member, IEEE*, and Robert Wille, *Senior Member, IEEE*

***Abstract*—Hardware/software co-designs are usually defined at high levels of abstractions at the beginning of the design process in order to provide a variety of options of how to realize a system. This allows for design exploration which relies on knowing the costs of different design configurations (with respect to hardware usage and firmware metrics). To this end, methods for cost estimation are frequently applied in industrial practice. However, currently used methods oversimplify the problem and ignore important features, leading to estimates which are far off from 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). Based on that, an approach is proposed which outperforms existing methods for cost estimation. Experimental evaluations within an industrial context show that, while the accuracy of the state-of-the-art approach is frequently off by more than 20% for area estimation and more than 15% for firmware estimation, the method proposed in this work comes rather close to the actual values (just 5-7% off for both area and firmware). Furthermore, our approach outperforms existing methods for scalability, generalization and decrease in manual effort.**

I. Introduction

Increasing the productivity in industrial hardware/software co-designs is a central issue, as it allows for an efficient development flow as well as a reduction of the design costs.

Automating design tasks early in the development process enable designers to efficiently construct the complex modern chips. Hardware (HW) and Firmware (FW) are 2 major constituents of any modern System-on-Chip (SoC) and, several techniques have been proposed for automating hardware and firmware components [27], [9], [12]. These techniques adapt model-driven approaches in order to specify the system at an higher abstraction level. This in turn allows systems to be represented in abstract structures with required attributes and excluding implementation details. The designer then has the flexibility to create multiple design configurations and realize the system with the desired configuration.

In order to automate the hardware components, a generation flow that focuses on the design-centric models is developed [11]. Here the generation is conceptually divided into multiple layers, which includes model-to-model transformations. The most abstract models are close to the specifications of the system and the least abstract models are close to the system realization in a specific platform (ex: RTL in VHDL/Verilog/SystemVerilog). In order to enhance the configuration capability, the system can be built in a top-down fashion using the generation flow. For the same, a system can be considered as a component tree with the top level component instantiating its child components (with additional program logic) and so on. Individual components are automated considering all required/specified configurations and hence, the system is an assembly of components and sub-components with high degree of configuration capability.

This enables design exploration at early stages of the design process, since plenty of options of 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 needed customized fashion, and 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 try to extrapolate the cost of each implementation based on the configuration. 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.

However, the cost estimation methods that are currently used in industrial practice [23], [29], [35] often oversimplify the problem and neglect important features that strongly influence costs. 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 investigated an alternative solution which borrows and re-adapts concepts based on *Machine Learning* (ML, [15]) which have been found suitable for problems in the domain of *Computer Vision* (CV, [3]).

In this work,[1] we report on these investigations as well as propose the correspondingly resulting methodology. 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.

[1] A preliminary work-in-progress report has been published before in [21].

Experimental evaluations conducted within an industrial environment show that the resulting approach clearly outperforms the state-of-the-art used thus far. While a state-of-the-art approach used until now frequently is off by more than 20% for area estimation and more than 15% for firmware estimation, the method proposed in this work comes rather close to the actual values (just 5-7% off for both area and firmware). Moreover, the proposed method offers higher scalability and flexibility since further features can be directly and automatically learned through the ML training process, and do not need a further hand-tailored pre-processing.

In the following, the proposed approach is described as follows: Section II, briefly reviews cost estimation for memory systems as conducted thus far – including a discussion of the shortcomings of today's state-of-the-art methods. Motivated by this, we outline and describe the proposed idea in Section III followed by a description of the implementation of the methods for accurate cost estimation in Section IV. Finally, Section V summarizes the results of the conducted evaluation before the paper is concluded in Section VI.

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 briefly introduce the hardware generation method used in our industrial environment flow as described in [30]. Afterwards, we present the specification of *Register Interface* (RI) components inspired from [10] which serve as running example in the remainder of this work. We opt for the implementation of this component because of its major importance in the industrial design case. In fact, RIs are almost ubiquitous in SoCs and essential for HW/FW configurability, as described in [10]. Using this example, today's shortcomings of cost estimation are illustrated – providing the motivation of this work.

In the hardware generation flow of our industrial environment illustrated in Fig. 1, we aim to instantiate hardware designs complying to given specifications. Hence, through a model-based generation approach (thoroughly described in [7], [30]), the designer instantiates a particular configuration of the component through an instance of the *meta-model* following the design requirements. This configuration would then trigger the generation of RTL as well as firmware code. Finally, the cost with respect to hardware and firmware metrics is obtained from the synthesis/compilation of the previously generated code. The obtained cost would then guide the designer towards a suitable implementation of hardware together with firmware on an FPGA board. This same procedure applies to a set of hardware components, including RIs.

RIs are common components in an SoC and provide the control and status 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:

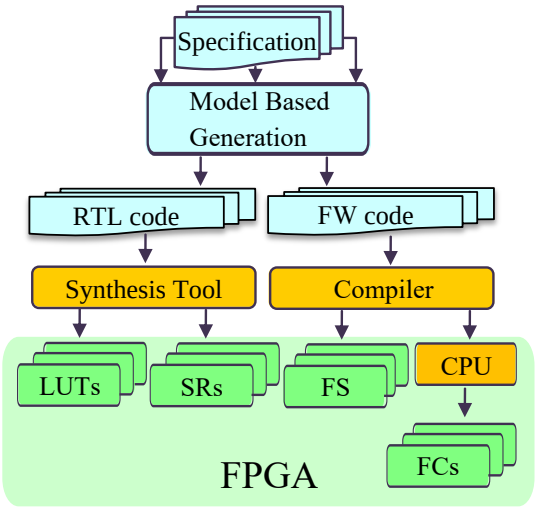


Figure 1: Model based hardware generation flow

Example 1. *Fig. 2 shows the meta-model of the considered RI component.* [2] *Here, we describe sub-components 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/*Unit*s 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).* [3] *Additionally, the corresponding FW (which eventually has to obey these accessibility settings) is described using the additional sub-components* Configuration*, which enlist the bitfields involved in the reading/writing operations, and* Setting*, which specify the default/reset value for the bitfields. Last, the* DontCareList *sub-component contains a reference to the bitfields not considered in the FW writing operation, and* Parameter *which is 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*. Appropriate configurations have to be chosen such that the intended behavior is realized (e.g. enabling proper access to peripheral devices), while satisfying specified cost constraints. For the latter, it is important to have an accurate estimation of the costs for a correspondingly considered configuration. In the following, as shown in Fig. 1, we thereby consider:

- the area, i.e. the number of *Configurable Logic*

[2] Note that, for sake of clarity, we focus only on the sub-components which are relevant for the remaining discussions. A complete description of this meta-model is available in an online appendix [11].

[3] Note that the structure shown in Fig. 2 allows for different *Unit*s to employ the same *Bitfield*.

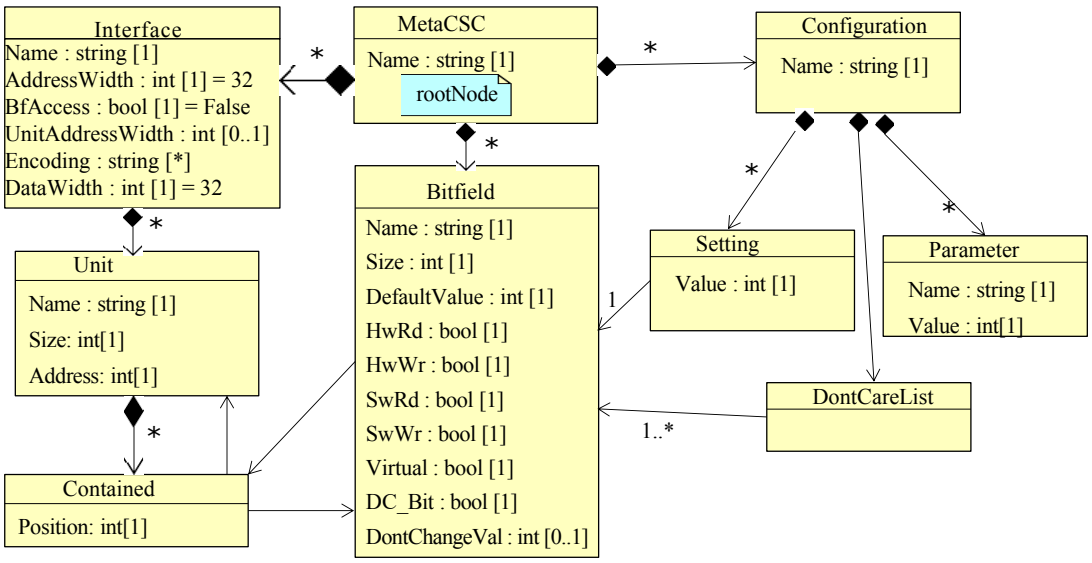


Figure 2: Meta-model of the *Register Interface* (RI)

Blocks (CLBs) in terms of *Logic Units* (LUTs) and *Slice Registers* (SRs) which are needed to realize the configuration on an FPGA board,

- 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).

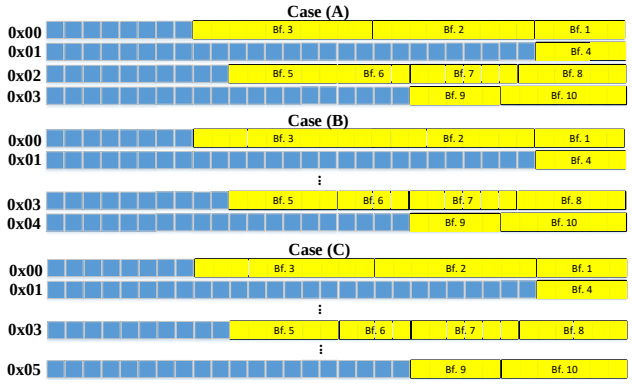
However, in early stages of the design flow, the designer has no fully-fledged implementation of the respective design configurations at hand (they are still to be implemented). Accordingly, the resulting costs need to be estimated in order to evaluate different design choices and, eventually, decide which indeed shall be realized. High-Level Synthesis Tools can be utilized for this purpose [26], and have as well been employed for HW/SW co-design of SoC FPGAs [32]. 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 [23], [29], [31], [35] or high level and application specific ones, as seen in [18]. More precisely, in [35], 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 [29] 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. In [23], 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*. Last, in [18], the estimation of the SoC area reduction is computed processing high level configuration data in an application specific manner, thus lacking in flexibility, generalization and not considering multiple objectives (e.g. both software and hardware related).

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

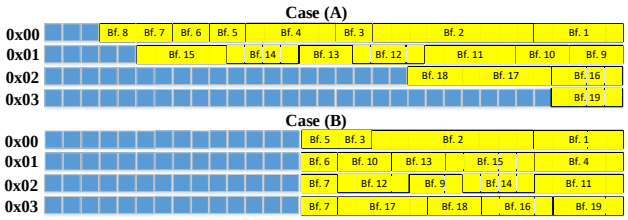
Example 2. *Consider the instances of the RI meta-model from Fig. 2 as shown in Fig. 3. Here, Fig. 3a shows three different instances which, at a first glance, look identical (e.g. in all three cases the bitfields are the same, and are instantiated in an identical fashion). In fact, the only difference 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, andc564 FCs for* all *cases. However, if we evaluate the HW Area and retrieve the FW metrics from the actual implementation of those instances,*[4] *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. 3a eventually costs 83 LUTs, 42 SRs, a FS of 2.2 Kb, and 524 FCs. The realization of Case (b) from Fig. 3a eventually costs 117 LUTs, 42 SRs, a FS of 2.2 Kb, and 524 FCs. Finally, in Case (c) of Fig. 3a, 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

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



(a) Different addresses



(b) Different bitfields mapping

Figure 3: Design configurations from the RI meta-model

the different cases shown in Fig. 3b. 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 the spatial position of the bitfields has an impact on costs; for this reason, 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 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 the recently developed fields of *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 actually easily 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 to pictures for CV algorithms. Eventually, the resulting representation is applied to (existing) ML approaches – providing accurate estimates.

A. Machine Learning for Computer Vision

Computer Vision (CV, [3]) is an interdisciplinary area which is mainly 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 can 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 were processed further [28]. Nowadays, through the availability of more sophisticated ML algorithms, pixel-based representations are often chosen as a direct input to the ML algorithms [5], [15]. In fact, through an extended use of *Neural Networks* (NNs, [5], [15]) (a set of ML algorithms 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 [5], [15].[5] For this reason, the representation of images becomes of central importance 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 $\mathcal{I}^{H \times W \times C}$. 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. 4a. 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. 4a. 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 $\mathcal{I}^{H \times W \times 3}$, 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 $\mathcal{O}^{H \times W}$ 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

[5]Note that ML and NNs have also succesfully been implemented in domains such as classification [19], text understanding [36], and human activity recognition [8].

algorithms [5], [15]. For example, let's take a CV task where, given a RGB image of a person, we want to estimate his/her age. To this end, several parts of the human body shown in the image may provide important information about 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", by minimizing a function that describes the difference between the age that our method predicts and the real age that is included as a label in the training data, all the features mentioned above not in an explicit (i.e. "hard-coded") fashion. In other words, the ML algorithm learns 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, as shown in [34].

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. With 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.

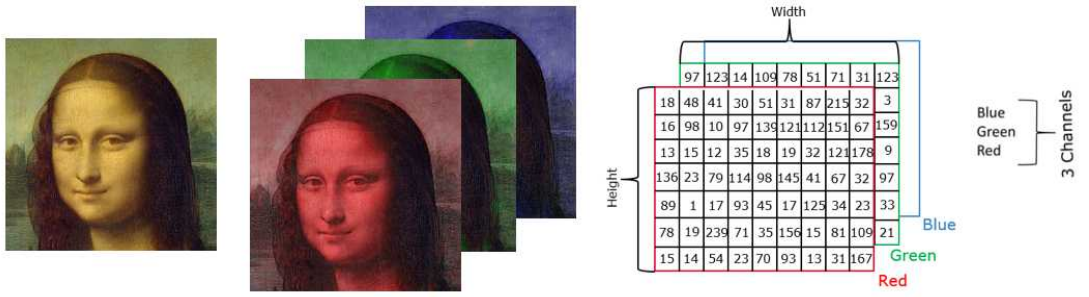
The hardware data-structure (called *HW image* in the following) represents the respective properties of the memory system in terms of a function $j(q, l, b)$ where q, l, b are coordinates of a 3D matrix $\boldsymbol{\mathcal{A}}^{Q \times L \times B}$. In contrast to CV, Q now defines the number of 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. 3a and discussed in Example 2. Those properties are usually represented in terms of a .uml file as sketched in left-hand side of Fig. 4b. Overall, this yields properties with respect to hardware write, hardware read, software write, etc. as sketched in the center of Fig. 4b. Those are eventually represented in terms of matrices as sketched in the right-hand side of Fig. 4b.*

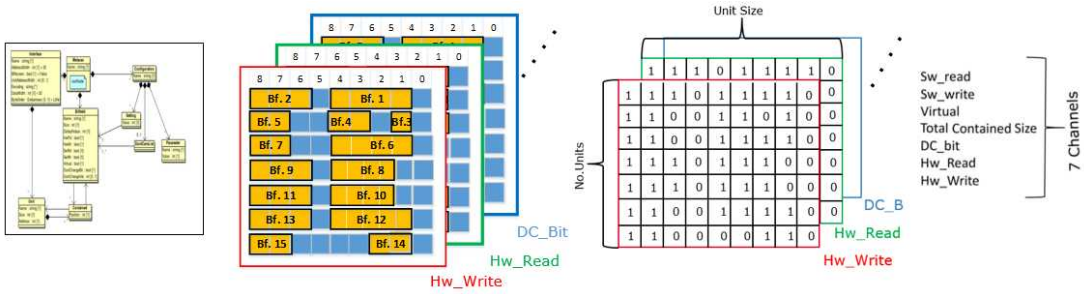
Using this data-structure, the HW costs of a memory system can be determined similarly to the age determination in the CV-domain discussed before. That is, first a training set of memory system instances is provided for which the respective real area costs are available. This is used to "learn" the respective 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 data-structure for the firmware, we get inspiration once again from the CV domain. The evaluation of the firmware cost can be thought of in fact as the same estimation of the age of a person through a multi-channel video. This actually gives 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



(a) Images RGB



(b) Hardware Images

Figure 4: Representation for CV and cost estimation tasks

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 $\boldsymbol{\mathcal{F}}^{Q\times L\times D\times P}$. 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 $\boldsymbol{\mathcal{F}}_d$ 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 $\boldsymbol{\mathcal{F}}_1^p$ contains the value 1 in the bitwise position of registers containing bitfields to be written; the same approach is applied to the reading operation ($\boldsymbol{\mathcal{F}}_2^p$). Both of these pieces of information are contained in the property *Configuration* of the RI meta-model. A further representation is added for the property *DontCareList* ($\boldsymbol{\mathcal{F}}_3^p$). 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 $\boldsymbol{\mathcal{F}}^p$. This composes the dynamic part of the data-structure for the firmware estimation. Successively, three further representations are added to the same frame, corresponding to a static representation of the *HwRd*-feature ($\boldsymbol{\mathcal{F}}_4^p$) and the *DC_Bit*-feature ($\boldsymbol{\mathcal{F}}_5^p$) as well as a representation of the total bitfields ($\boldsymbol{\mathcal{F}}_6^p$) of the design. These values do not change from frame to frame and, hence, are implemented as repeated static representations at each point $\boldsymbol{\mathcal{F}}^p$ of the sequence.

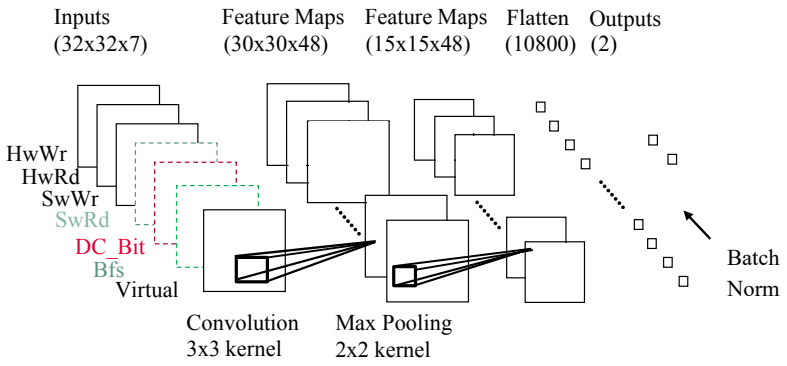
Using this representation, an ML algorithm is equipped with all the information related to firmware and can conduct the cost estimation in a similar fashion as sketched above for the hardware image.

IV. Implementation

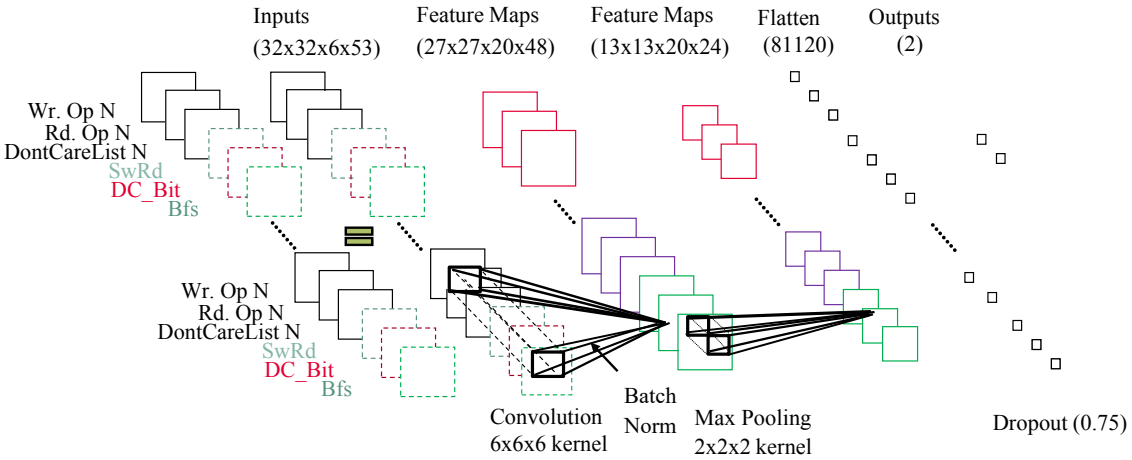
In this section, we provide the technical details about a possible implementation of the concepts introduced above. To this end, we first present how ML algorithms for CV are utilized to the proposed data-structure so that they can be utilized for the considered cost estimation problem. Afterwards, we provide details on how we optimize the non-trainable parameters of the utilized networks to further improve the obtained accuracy.

A. Implementation of the Machine Learning Algorithms

In order to describe the implementation of the methods proposed above, we distinguish between the estimation of the HW costs (LUTs and SRs) and the estimation of the FW costs (FCs and FS). For this reason, we consider two multiple outputs regressions as supervised learning problems, i.e. where the ML algorithm is guided through the learning phase by the real values of the HW Area and of the FW Metrics. The task of the regressions is then to predict the *outputs*, indicated as $\mathbf{y}_1, \ldots, \mathbf{y}_4$, from the *inputs*, which correspond to the values contained in the proposed data-structures. Accordingly, LUTs (represented by $\mathbf{y}_1$) and Slice Registers (represented by $\mathbf{y}_2$) of the RI are the outputs which indicate the HW Area. We call the prediction towards these outputs *Area Regression* (AR). The FW Cycles (represented by $\mathbf{y}_3$) and FW Size (represented by $\mathbf{y}_4$) instead are the selected FW Metrics. We call the corresponding forecast as *FW Metrics Regression* (FMR). AR and FMR are obtained, as mentioned before, through distinct ML models.



(a) Area Network (2D CNN)



(b) Firmware Network (3D CNN)

Figure 5: Architectures of the used *Convolutional Neural Networks* (CNNs)

For both regressions, a set of training data $\{(\boldsymbol{\mathcal{X}}^1, \boldsymbol{\mathcal{Y}}^1), \ldots, (\boldsymbol{\mathcal{X}}^N, \boldsymbol{\mathcal{Y}}^N)\}$ with N being the size of the training set is used.

In the case of the AR, the 3D matrix of feature measurements is of size of $\boldsymbol{\mathcal{A}}$, that is $\boldsymbol{\mathcal{X}}^i_{AR} \in \mathbb{N}^{Q \times L \times B}$, whereas in the FMR, the 4D matrix is of size of $\boldsymbol{\mathcal{F}}$, that is $\boldsymbol{\mathcal{X}}^i_{FWR} \in \mathbb{N}^{Q \times L \times D \times P}$.

In the following paragraphs we describe the structure of the ML algorithms for AR (i.e. *Area Network*) and for the FMR (i.e. *Firmware Network*).

1) The Area Network: For the AR, we consider as an input the HW Images, and a 2D CNN (Convolutional Neural Network) [15]. We select a 2D CNN because of its capability to capture spatially related information. In our case, the HW images are processed along the Q and L dimensions of each feature representation. This allows us to accurately and easily elaborate spatial relations and, hence, to overcome the issues discussed in Section II – yielding the desired accuracy for a cost estimation of memory systems. The architecture of the used 2D CNN is sketched in Fig. 5a.

2) The Firmware Network: With regards to the FMR, we consider the FW sequence as input, and a 3D CNN [33] for the estimation. This type of NN can take into account a further dimension of the inputs with respect to the 2D CNN. The FW sequence is processed through local regions of the dimensions Q, L as well as D, where the different feature representations are positioned. This is performed on each single frame $\boldsymbol{\mathcal{F}}^p$ of FW sequence. Selecting this architecture, we can explore the spatial as well as the sequential information over the FW operations, for obtaining the FW cost estimation. Fig. 5b shows a sketch of the implemented 3D CNN's structure.

B. Optimization of the Utilized Networks

Once the ML algorithms and inputs are selected for the AR and FMR, further optimization potential exists in properly setting the non-trainable parameters of the network, i.e. the network *hyper-parameters*. This optimization step aims to select the best configuration possible for minimizing the Network estimation error and improve the accuracy for both the Area and the Firmware Network [6]. For doing so, the hyper-parameters optimization algorithm selects the best possible configuration out of the hyper-parameters space. We introduce an hyper-parameters space for the considered neural networks, where we define an interval from where the algorithm can select a value for any specific hyper-parameter (i.e. for the Firmware Network, we tried range of different *Filter sizes* e.g. 2 x 2 x 2, 3 x 3 x 3, etc., *No. of Layers* e.g. 1, 2, 3, etc. and other hyper-parameters). If none of the hyper-parameters optimized value tends to the extreme of interval, we hold the predifined interval, else we shift it.

As an hyper-parameters optimization algorithm, we adopt the *Tree-structured Parzen Estimator* (TPE) approach [4]. The

TPE is a Bayesian model-based optimization that explores the hyper-parameters space in a non-trivial manner by running several possible configurations of the network and selecting the best performing hyper-parameters for our data. Moreover, through the TPE algorithm, we are able to integrate an ablation study in the network optimization algorithm. The ablation study process, also used in ML approaches such as [13] and [14], consists of removing parts from the network (e.g. a specific layer/group of layers) and evaluating the change in performance of the algorithm. Thanks to this procedure, we could reach at the same time high accuracy and a reduction of both networks to their essential components. This leads to an accurate estimation of HW Area/FW Metrics and yet reduces the complexity of the networks.

As a final result of the TPE approach, we can define an optimized architecture for the estimation of the area as well as for the firmware cost. A description of the hyper-parameters and technical details of the 2D/3D CNNs can be found in [15], [33]. The optimal hyper-parameters that we have found thanks to this process are shown in Table I, as well as in Fig. 5a and in Fig. 5b.

In the following paragraphs, we analyze the structure of the optimized Area and Firmware Networks obtained using the TPE approach.

1) Optimized Area Network: As previously mentioned, for the AR we use a 2D CNN architecture, sketched in Fig. 5a. Here, each one of the seven channels of the HW Images is locally processed (convolution operation) by 48 different 3×3 kernels, which stride on the HW image and elaborate the spatial positions of properties in the registers. This operation takes place in the *2D Convolutional Layer* of the CNN. Out of this operation, 48 *Feature Maps* of dimensionality 30×30 are generated. These represent the results of the convolution between the HW image and the above mentioned kernels. Successively, a *Max Pooling Layer* performs pooling operations on the Feature Maps, selecting the maximum values out of local regions in the maps. This operation halves the width and height of the *Feature Maps*. Next, the maps are concatenated in a one dimensional vector (*Flatten Layer*) that is fully connected towards the two final outputs of the network. Before the final output, a *Batch Normalization* operation is performed. The dimensions of the layers and components of the network are sketched in Fig. 5a.

2) Optimized Firmware Network: The architecture of the 3D CNN used for the FMR is sketched in Fig. 5b. Here, the selected kernels have dimension $6 \times 6 \times 6$, and are able to process through a convolution operation all the six channels of the FW sequence at each frame $\mathcal{F}^p$ (this takes place in the *3D Convolutional Layer*). Elaborating the six channels of the FW sequence at each frame through the selected kernels allows to process the spatial configuration of the bitfield properties through time. This operation generates 20 *Feature Maps* of dimension $27 \times 27 \times 48$. The maps are successively taken as input to the *Max Pooling Layer*, which performs a cubic pooling operation, taking maximum values out of $2 \times 2 \times 2$ local regions. Similarly to the Area Network, a *Flatten Layer* concatenates the maps into a one dimensional vector. A *Dropout Layer* randomly masks out some of the neurons of the vector, with a certain dropout probability. The values out of the active neurons of the network will be propagated into a *Batch Norm Layer*, where a batch-dependent normalization is performed. Finally, the two outputs of the Firmware network are fully connected through a *Fully Connected Layer*. The Firmware Network dimensions are sketched in Fig. 5b. As pointed out in Table I, we use *Adam* [20] as an optimizer and *ReLU* [25] activation functions for both 2D and 3D CNN.

Table I: Optimal Network's Hyper-parameters

Networks Optimization				
Hyper-parameters	**Area**		**FW Metrics**	
	MLP	**2D CNN**	**MLP**	**3D CNN**
Initial LR	/	0.001	/	0.001
LR Scheduling	/	Exp D	/	Exp D
Dropout Rate	/	/	/	0.75
Batch Size	716	64	716	64
Optimizer	L-BFGS	Adam	L-BFGS	Adam
Activations	Relu			
No. of Filters	/	48	/	20
Filter size	/	3 x 3	/	6 x 6 x 6
No. of Layers	2	3	3	3

LR: Learning Rate, *Const*: Constant Learning Rate, *Exp D*: Exponential Decay of the Learning Rate

Besides the issues discussed above, the choice of the kernel size plays a key role in the hyper-parameter optimization for both the 2D CNN for the AR and 3D CNN for the FWR. In fact this component is the core of the convolution operation, establishing how the spatial information in the data is taken into account and how to process the bitfield properties in the underlying data-structure (i.e. HW Images or FW Sequence). The size of the kernels is therefore a key hyper-parameter in the realization of both proposed networks. Typically, kernels used in computer vision have equal dimensions along the height and the width of the image (e.g. 3×3 squared kernel, $6 \times 6 \times 6$ cubic kernel) [15]. These proportions are intended for the kernel to respond equally on variations along different dimensions of the input signal (e.g. image). For the FMR nevertheless, this property is not strictly needed. In fact, in the structure of the firmware program, the writing and reading operations on bitfields take into account a single register at a time. This means, configurations of other registers apart from the accessed one (e.g. for writing/reading a bitfield), do not influence the single access operation, and thus the firmware metrics. Therefore, by using a non-cubic $1 \times 6 \times 6$ kernel, we take into account only the bitfield properties of one single register at a time. As shown in the Table II, an optimized network provided with the selected non-cubic kernels and 5 *Feature Maps* avails itself of a reduced number (decrease by factor 4x) of parameters w.r.t. to the best performing network architecture. Nonetheless, its results are only slightly lower in terms of accuracy. An important remark on that is that a decrement in the number of parameters can be related to lower power consumption, higher speed and efficiency for the training and inference phase of the network [16], [22].

V. Evaluation

The proposed data-structures representing hardware and firmware features (i.e. the HW image and FW sequence) have

Table II: Kernel Performance Comparison

Opt FW NNs	K Dim	No. Par	Avg RMSE	Avg R^2 Sc
1.	6 x 6 x 6	**188,222**	**88.1**	**0.942**
2.	1 x 6 x 6	**45,037**	**90.9**	**0.937**

Opt FW NNs: Optimized Firmware Networks, *K Dim*: Kernel Dimension, *No. Par*: Amount of trainable parameters in the Network, *Avg RMSE*: provides the average of the *Root of Mean Squared Error* as loss for the Firmware Network, *Avg R^2 Sc*: provides the average of the explained variance (R^2 Score).

been realized and applied to the proposed implementations of the ML algorithms as described above. Afterwards, we evaluated the corresponding results, i.e. the obtained numbers using the ML approach inspired by CV has been compared to the results obtained by the state-of-the-art cost estimation. This latter method is currently used within our industrial environment at Infineon Technologies AG [35]. In this section, we summarize the results as well as the drawn conclusions. To this end, we first provide some more details about the correspondingly used environment as well as the considered metrics (which follow basic evaluation schemes from the ML domain). Afterwards, the obtained results are presented and conclusions are drawn.

A. Used Learning Environment

Our evaluations have been performed with the following system configurations: as software development environment, we chose Python v3.6, Tensorflow-GPU v1.0.1, and Hyperopt. For optimizing the algorithm on the Nvidia GPU, we used the CUDA Toolkit 9.0 and cuDNN v7.0. With respect to hardware, we considered a Nvidia Tesla P100 for training the ML algorithms, an Intel Core i7-8700K CPU, and DIMM 16 GB DDR4-3000 module of RAM. For evaluating LUTs and SRs, we retrieved reports from the Vivado Synthesis on an Arty-7 FPGA board from Xilinx,[6] while for measurement of the FCs we used a RISC-V CPU implementing 32 bit Base Integer Instruction Set architecture [1].

For the purpose of evaluating the proposed approach, we considered a large dataset of valid design configurations for memory systems, as described in details in [21]. We first uniformly sample the amount of considered bitfields in each design configuration within a range, such the *No. of Bitfields* $\in [25, 50]$. This constitutes a predefined design space taken from design experience. Then, we uniformly sample the attributes of each bitfield including *Size* and properties (i.e. *HwWr*, *HwRd*, etc.). Last, we dispose the bitfields in a random order and we insert them in a fitting *No. of Units*. With this procedure we are able to sample out of different *Bitfields Configurations*. The dataset contains variations over spatial distribution of bitfields – similar to the ones shown in Fig. 3. In total, the complete dataset is composed of 1024 generated design configurations. We chose this amount after observing that the generation of further samples would not improve the accuracy of the tested ML algorithms.

After generating the design configurations, we retrieved the objectives measurements for each one of the design instances. We first synthesized the instances for obtaining the number of LUTs and SRs of each design. After that, we ran on each one of the memory system a firmware program, where the operations of the program are specified in the design configuration. From this, we eventually retrieved the Firmware Size and Firmware Cycles.

After obtaining the dataset, we divided this into a training set (70% of the considered design configurations) used for the training phase of the ML algorithms and a test set (20% of the considered design configurations) with which we evaluated the trained algorithms. Besides that, also a validation set (10% of the considered design configurations) has been used for fine-tuning the hyper-parameters of the ML algorithms. Using most of the data for training (e.g. $60 - 80\%$) is a common practice in CV. We did not consider different data split proportions after reporting an accuracy difference $< 1\%$ between training and test data. We can therefore conclude that the proposed method does not overfit training data and is able to generalize to configurations that it has not seen yet. In fact, we report high performance also on the validation and the test set, consistently with the guidelines pointed out in [15].

Following the representations introduced in Section III, the proposed approach utilizes $\boldsymbol{\mathcal{X}}^i_{AR} \in \mathbb{N}^{32\times32\times7}$ feature values to represent HW Images for *Area Regression* (AR) and $\boldsymbol{\mathcal{X}}^i_{FMR} \in \mathbb{N}^{32\times32\times6\times53}$ feature values to represent FW sequences for *FW Metrics Regression* (FMR). In contrast, the state-of-the-art approach used thus far only aggregates feature values in form of vectors of dimensions 1×9 as an input for the AR and 1×6 as an input for the FMR. We implement the state-of-the-art approach as a *Multi Layer Perceptron* (MLP, cf. [15]), which is a fully connected network, i.e. non-convolutional, for processing aggregated values. We optimize this architecture through the TPE algorithm. In the case of the AR, the MLP is composed by 1^{st} *Layer (11 Neurons)* - 2^{nd} *Layer (4 Neurons)* while for the FMR, it is composed by 1^{st} *Layer (15 Neurons)* - 2^{nd} *Layer (6 Neurons)*. For both MLPs, we use an L-BGFS optimizer, with RELU activation functions and L2 Weight Regularizations [15]. Hyper-parameters choices for the MLPs used are enlisted in Table I.

B. Obtained Results

Table III summarizes the respectively obtained results. Here, the top of the table provides the mean values and the range of the design configurations within the considered dataset with respect to LUTs (represented by $\mathbf{y}_1$), SRs (represented by $\mathbf{y}_2$), Firmware Cycles (represented by $\mathbf{y}_3$), and Firmware Size (represented by $\mathbf{y}_4$). Additionally, we provide a brief summary of *Inputs* and *Outputs* of our method, together with their dimensionality. Afterwards, the corresponding results are provided for area and firmware when the currently used state-of-the-art method is applied (denoted by *S-o-t-a*) and when the proposed method is applied (denoted by *Proposed*). As results, we provide the error of the respectively estimated values compared to the real values. More precisely, the row *RMSE* indicates the *Root of Mean Squared Error* (RMSE, [15]) on the estimates to the real values (in the form of $E_{\mathbf{y}_k}$). Besides that, the R^2 *Score* provides the explained

[6]Documentation to be found on the website: https://www.xilinx.com/products/boards-and-kits/arty.html

Table III: Dataset and Results

Dataset Mean	$\mu_{\mathbf{y}_1}$ = 456 $\mu_{\mathbf{y}_2}$ = 272	$\mu_{\mathbf{y}_3}$ = 1238 $\mu_{\mathbf{y}_4}$ = 3516 b
Dataset Range	$y_1 \in [228, 726]$ $y_2 \in [106, 408]$	$y_3 \in [724, 1656]$ $y_4 \in [828, 5758]$ b

Inputs	HW Images FW Sequences	(1024x32x32x7) (1024x32x32x6x53)
Outputs	HW Images FW Sequences	$\{\mathbf{y}_1, \mathbf{y}_2\}$ $\{\mathbf{y}_3, \mathbf{y}_4\}$

	Area		**Firmware Metrics**	
	S-o-t-a	**Proposed**	**S-o-t-a**	**Proposed**
RMSE	128 $E_{\mathbf{y}_1}$ 67 $E_{\mathbf{y}_2}$	**32** $E_{\mathbf{y}_1}$ **14** $E_{\mathbf{y}_2}$	272 $E_{\mathbf{y}_3}$ 261 b $E_{\mathbf{y}_4}$	**80** $E_{\mathbf{y}_3}$ **96** b $E_{\mathbf{y}_4}$
R^2 Score	0.72 $R^2_{\mathbf{y}_1}$ 0.76 $R^2_{\mathbf{y}_2}$	**0.93** $R^2_{\mathbf{y}_1}$ **0.95** $R^2_{\mathbf{y}_2}$	0.83 $R^2_{\mathbf{y}_3}$ 0.81 $R^2_{\mathbf{y}_4}$	**0.95** $R^2_{\mathbf{y}_3}$ **0.93** $R^2_{\mathbf{y}_4}$

RMSE provides the *Root of Mean Squared Error* (denoted as $E_{\mathbf{y}_k}$) and the *R^2 Score* provides the explained variance (denoted as $R^2_{\mathbf{y}_k}$) with respect to the number of LUTs ($\mathbf{y}_1$), SRs ($\mathbf{y}_2$), FCs ($\mathbf{y}_3$), and FS ($\mathbf{y}_4$). Here, $\mathbf{y}_4$ is indicated in terms of bits (b).

variance [15] of the estimations and is accordingly indicated as $R^2_{\mathbf{y}_k}$. Recall, both errors are provided for LUTs ($\mathbf{y}_1$), SRs ($\mathbf{y}_2$), FCs ($\mathbf{y}_3$), and FS ($\mathbf{y}_4$).

For example, Table III shows that the currently used state-of-the-art method yields LUTs estimates with a mean error of $E_{\mathbf{y}_1} = 128$ compared to the real LUTs values. Additionally taking the variance in the LUTs values of the considered dataset into account, we get $R^2_{\mathbf{y}_1} = 0.72$, i.e. the accuracy of this method is off by 28% on average.

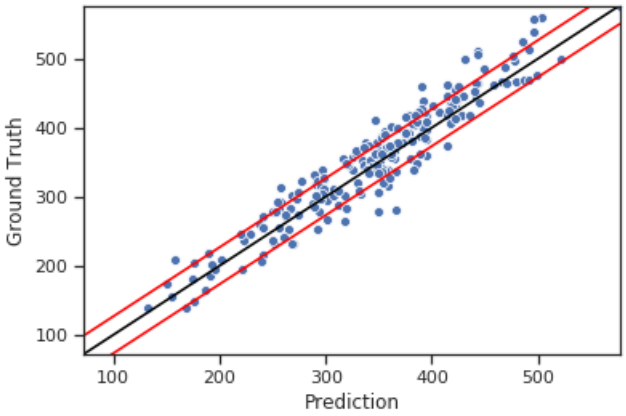


Figure 6: LUTs (y_1) prediction vs. ground truth

The results clearly show the significantly improved accuracy of the proposed method compared to the state-of-the-art used so far. The mean errors are significantly reduced. The factor of reduction of the mean error varies from a maximum of approx. 5 for the $E_{\mathbf{y}_2}$ to a minimum of approx. 2.5 for the $E_{\mathbf{y}_4}$. Much more important, however, is how close the considered approaches estimate the actual values: While the accuracy of the state-of-the-art approach frequently is off by more than 20% for area estimation and more than 15% for firmware estimation, the elaborated consideration of spatial information of the proposed method comes rather close to the actual values (just 5-7% off for both area and firmware). A visualization of the error distribution is presented in Fig. 6. Here, we show the predictions of our method on unseen (i.e. test) data vs. the

Table IV: Performance Comparison

	R^2 Score	**Tr. Time**	**Tr. Params**
2D CNN (Proposed AR)	0.9407	129.25 s	24,640
3D CNN (Proposed FMR)	0.942	747.65 s	188,222
LSTM (FMR)	0.8654	587.02 s	1,629,106
LSTM + Attention (FMR)	0.8306	1023.28 s	847,888

Tr. Time provides the training time of the benchmarked neural networks, while *Tr. Params* provides the *No. of Trainable Parameters* of the considered neural networks models.

ground truth cost retrieved with Xilinx Vivado synthesis. These results refer to the objective y_1, on which our model performs at its worst among the 4 objectives (error standard deviation of 20.4, pointed out by the red lines), as shown in Table III. One can see how every data-point is close to the diagonal, i.e. very accurately predicted. The confidence interval between the two lines created with the standard deviation of the errors is very narrow with just a few points outside of it. This shows the robustness of the proposed method everywhere and so, not overfitting a particular subset of configurations. Furthermore, the method proposed in this work offers higher flexibility (e.g. could easily support additional features such as further types of writing operations, conditional logic, etc.) since they are directly and automatically learned in the training process from easy-to-implement feature representations. At the same time, our method is more scalable when compared to the state-of-the-art since in CNNs the parameters depend less on the input size than in fully connected networks such as MLP, as shown in [15]. Therefore processing bigger inputs, i.e. RI with more units/bitfields, will lead to a much minor increase in the computational effort. The state-of-the-art is inferior when it comes to automatic learning since it requires still manual effort in the pre-process of data aggregation and feature representation. Our model also provides a solution to this last point.

In order to compare our proposed 3D CNN for FMR to other popular neural networks architectures for sequential information processing, we benchmarked it against an LSTM model, first proposed in [17], and against an LSTM model with attention, initially proposed in [2]. Table IV shows the performance comparison of different types of models for *R^2 Score*, *Training Time*, and number of *Trainable Parameters*, which relates to (but do not exclusively define) the complexity of the model. We first report, for the sake of completeness, the 2D version of the CNN (i.e. *Proposed Method* for AR). To benchmark our 3D CNN model for FMR, we build an LSTM Network with two LSTM layers and two Fully Connected layers. Additionally, we implemented an LSTM with Attention with only one LSTM layer, one Attention layer, and two Fully Connected layers. The 3D CNN has a minor number of parameters to the LSTM model and LSTM with Attention model but has a better accuracy than both.

VI. Conclusion and Future Works

In this work, we proposed a cost estimation method for memory systems that explicitly takes spatial information into account in order to derive much more accurate values. To this end, we observed that problems from the domain of

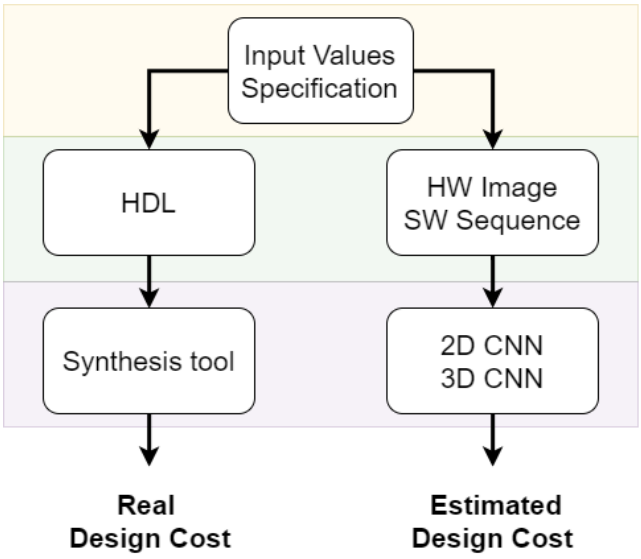


Figure 7: Flow of the proposed method (right branch) compared to the current industrial flow (left branch)

Computer Vision – in particular age determination of persons depicted in images – are rather similar to cost estimation of memory systems. Accordingly, we re-adapted corresponding solutions based on *Machine Learning* which have been found suitable for computer vision problems for the purpose of cost estimation. The overall flow of the proposed method is sketched in Fig. 7 together with the one currently used in our industrial environment. Experimental evaluations within an industrial context showed that, while the accuracy of the state-of-the-art approach is far off from the actual values, the method proposed in this work comes rather close to them. We believe that learning the parameters directly on the data is the key to make our method suitable for similar estimation tasks. Moreover, we used well establish tools to conduct an extensive study on how to choose the optimal hyper-parameters and we finally chose the configuration that achieved the highest performance. Nevertheless, we have shown that using a non-cubic kernel $1 \times 6 \times 6$ for the FMR reported quasi-optimal performance while reducing considerably the computational effort needed. Furthermore, the proposed approach can easily be extended by further features. Future work will focus on exploring this flexibility as well as the the resulting complexity – approaching a hardware/software trade-off analysis. At the same time, in order to develop a ML model that is measuring uncertainty in the design cost prediction, we are exploring Bayesian Machine Learning [24]. In this way, important information about the confidence of the model on the prediction can be shared with the designer, increasing his/her understanding of the model cost estimation.

Acknowledgement

This work has partially been supported by the LIT Secure and Correct Systems Lab funded by the State of Upper Austria.

References

[1] Krste Asanovi and David A. Patterson. Instruction sets should be free: The case for risc-v. Technical Report UCB/EECS-2014-146, EECS Department, University of California, Berkeley, Aug 2014.

[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

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

[4] Bergstra, James and Bardenet, Rémi and Bengio, Yoshua and Kégl, Balázs. Algorithms for hyper-parameter optimization. In *Proceedings of the 24th International Conference on Neural Information Processing Systems*, NIPS'11, pages 2546–2554, USA, 2011. Curran Associates Inc.

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

[6] Marc Claesen and Bart De Moor. Hyperparameter search in machine learning. *arXiv preprint arXiv:1502.02127*, 2015.

[7] Keerthikumara Devarajegowda, Johannes Schreiner, Rainer Findenig, and Wolfgang Ecker. Python based framework for hdsls with an underlying formal semantics: (invited paper). *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1019–1025, 2017.

[8] Tushar Dobhal, Vivswan Shitole, Gabriel Thomas, and Girisha Navada. Human Activity Recognition using Binary Motion Image and Deep Learning. *Procedia Computer Science*, 58:178 – 185, 2015. Second International Symposium on Computer Vision and the Internet (VisionNet15).

[9] S. H. M. Durand and V. Bonato. A tool to support Bluespec SystemVerilog coding based on UML diagrams. In *IECON 2012 - 38th Annual Conference on IEEE Industrial Electronics Society*, pages 4670–4675, Oct 2012.

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

[11] Wolfgang Ecker and J Schreiner. Metamodeling and code generation in the hardware/software interface domain. In *Handbook of Hardware/Software Codesign*, pages 1051–1091, 11 2017.

[12] T. Farkas, C. Neumann, and A. Hinnerichs. An Integrative Approach for Embedded Software Design with UML and Simulink. In *2009 33rd Annual IEEE International Computer Software and Applications Conference*, volume 2, pages 516–521, July 2009.

[13] Chris Fawcett and Holger H Hoos. Analysing differences between algorithm configurations through ablation. *Journal of Heuristics*, 22(4):431–458, 2016.

[14] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

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

[16] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*, pages 1135–1143, 2015.

[17] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997.

[18] Hsuan Hsiao and Jason H Anderson. Sensei: An area-reduction advisor for fpga high-level synthesis. In *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 25–30. IEEE, 2018.

[19] Yoon Kim, Yacine Jernite, David Sontag, and Alexander M Rush. Character-Aware Neural Language Models. In *AAAI*, pages 2741–2749, 2016.

[20] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[21] K. Devarajegowda M. Manzinger W. Ecker L. Servadei, E. Zennaro and R. Wille. Accurate cost estimation of memory systems inspired by machine learning for computer vision. In *2019 Design, Automation and Test in Europe(DATE)*. IEEE, 2018.

[22] Zhouhan Lin, Matthieu Courbariaux, Roland Memisevic, and Yoshua Bengio. Neural networks with few multiplications. *arXiv preprint arXiv:1510.03009*, 2015.

[23] 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 *Proceedings of the 2016 Conference on Design, Automation & Test in Europe*, pages 918–923. EDA Consortium, 2016.

[24] Kevin P Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

[25] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[26] 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*, 35(10):1591–1604, Oct 2016.

[27] Gabriela Nicolescu and Pieter J Mosterman. *Model-based design for embedded systems*. Crc Press, 2009.

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

[29] 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. *J. Syst. Archit.*, 59(10):1144–1156, November 2013.

[30] Johannes Schreiner and Wolfgang Ecker. Digital hardware design based on metamodels and model transformations. In *VLSI-SoC: System-on-Chip in the Nanoscale Era - Design, Verification and Reliability - 24th IFIP WG 10.5/IEEE International Conference on Very Large Scale Integration, VLSI-SoC 2016, Tallinn, Estonia, September 26-28, 2016, Revised Selected Papers*, pages 83–107, 2016.

[31] Lorenzo Servadei, Elena Zennaro, Tobias Fritz, Keerthikumara Devarajegowda, Wolfgang Ecker, and Robert Wille. Using machine learning for predicting area and firmware metrics of hardware designs from abstract specifications. *Microprocessors and Microsystems*, 2019.

[32] 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 *Proceedings of the 11th International Conference on Distributed Smart Cameras*, ICDSC 2017, pages 174–179, New York, NY, USA, 2017. ACM.

[33] Du Tran, Lubomir D. Bourdev, Rob Fergus, Lorenzo Torresani, and Manohar Paluri. C3D: Generic Features for Video Analysis. *CoRR*, abs/1412.0767, 2014.

[34] Dong Yi, Zhen Lei, and Stan Z Li. Age estimation by multi-scale convolutional network. In *Asian conference on computer vision*, pages 144–158. Springer, 2014.

[35] Elena Zennaro, Lorenzo Servadei, Keerthikumara Devarajegowda, and Wolfgang Ecker. A Machine Learning Approach for Area Prediction of Hardware Designs from Abstract Specifications. In *Proceedings of the 21st Euromicro Conference on Digital System Design*, 2018.

[36] Xiang Zhang and Yann LeCun. Text Understanding from Scratch. *CoRR*, abs/1502.01710, 2015.

Lorenzo Servadei is pursuing his Ph.D's degree at Infineon Technologies AG, in collaboration with the Johannes Kepler University Linz. His research focus is Hardware Optimization with Machine Learning. He is currently lecturing on Machine Learning at the Technical University of Munich.

Edoardo Mosca currently pursues a master's degree in Mathematics in Data Science at the Technical University of Munich, Germany. He works at Infineon Technologies AG in Munich as Machine Learning researcher and teaches Machine Learning at the Technical University of Munich.

Elena Zennaro received the bachelor's degree in Information Engineering and the master's in Automation Engineering from University of Padova (Italy), respectively in 2015 and 2017. She is currently working at Infineon Technologies AG in Munich, as Machine Learning engineer in the Design Department.

Keerthikumara Dewarajegowda received his master's degree from University of Kaiserslautern in the year 2016. He is currently pursuing his PhD at the University of Kaiserslautern. The research focus includes Formal Verification methods for modern digital designs and Design Automation methods.

Michael Werner received his M. Sc. in Electrical Engineering and Information Technology in 2017 from the Technical University of Munich, Germany. He is currently pursuing as a (Ph.D) research scholar at Infineon Technologies AG in cooperation with the Technical University of Munich with a focus on model-driven Firmware Development.

Wolfgang Ecker is Senior Principal Engineer at Infineon and Professor at Technical University of Munich. Wolfgang Ecker is (co-)author of over 200 papers on Modelling and Design Automation, received 5 best paper awards, was granted with the German EDA achievement award. He is member of Acatech, the German Academy of Science and Engineering. Wolfgang Ecker leads the Infineon Deep Learning internal think tank. In addition, he is member of the AI commission of inquiry of the German Government.

Robert Wille is Full Professor at the Johannes Kepler University Linz. His research interests are in the Design Automation for conventional and emerging technologies. In this field, he co-authored over 300 papers, got frequently awarded, and served the community various capacities.