© 2023 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

presented at IEEE ASICON 2023 (DOI: 10.1109/ASICON58565.2023.10396119)

Automatic Timing-Driven Top-Level Hardware Design for Digital Signal Processing

Wuqiong Zhao, Changhan Li, Zhenhao Ji, You You, Xiaohu You, and Chuan Zhang
[1]Lab of Efficient Architectures for Digital-communication and Signal-processing (LEADS)
[2]National Mobile Communications Research Laboratory, Southeast University, Nanjing, China
[3]Purple Mountain Laboratories, Nanjing, China
Email: {wqzhao, chzhang}@seu.edu.cn

***Abstract*—Though hardware auto generators can efficiently generate architectures of different design metrics based on generation formulas, top-level design for digital signal processing remains challenging. In this paper, we propose an automatic timing-driven top-level hardware generation scheme which integrates top-level timing arrangement, code generation and fast evaluation to further alleviate the heavy workload of hardware design for digital systems. To demonstrate the effectiveness of our proposed scheme, a channel impulse response estimator is implemented. It is shown that our scheme can explore design space and optimize hardware architecture automatically with different design constraints.**

***Index Terms*—Hardware design, top-level design, digital signal processing (DSP), wireless communications.**

I. Introduction

In recent years, application specific integrated circuits (ASIC) are being used in various applications such as baseband signal processing and image processing. Hardware design for digital signal processing (DSP) is significant for efficient ASIC implementation of digital systems [1].

For a specific digital signal processing system, there are various implementation strategies. Choosing among different strategies can be challenging due to the large design space and complicated design evaluation process. Traditionally, hardware designers rely on experience to determine the implementation strategy, and optimize the architecture by going through multiple design iterations, which is highly time-consuming.

To address the issue, researchers have proposed auto generators for hardware implementation [2], [3]. Auto generators use a formula-based register transfer level (RTL) code generation scheme, supporting different very large-scale integration (VLSI) design techniques and exploiting algorithm characteristics. With the auto generator, hardware designers can cover feasible design space without having to write RTL code for each specific case. Additionally, auto generator enables hardware evaluation and optimization based on formula, which greatly shortens the design period by sparing hardware designers from having to go through long design iterations for hardware optimization. Automatic quantization with deep reinforcement learning proposed in [4] presents an automatic approach for hybrid quantization scheme optimization, which is efficient especially combined with auto generators.

We propose an automatic top-level hardware optimizing scheme based on auto generators, which integrates RTL code generation, fast evaluation and top-level timing arrangement. With our proposed scheme, hardware designers no longer have to worry about the complicated timing constraints with different design parameters, and can do RTL writing, evaluation and optimization in the same coding environment. Our scheme could be seen as an enhanced realization platform for auto generators, providing developer-friendly and user-friendly interfaces with automatic optimizations.

Other automatic design attempts such as high-level synthesis (HLS) [5] have been made. HLS enables designers to design hardware using software language, which greatly reduces design period and difficulty, but suffers from inefficient hardware implementations. Works like [6], [7] propose automatic ways of implementing the retiming technique based on data flow graphs (DFG) which greatly shortened the critical path of the architecture. These works mainly focus on detailed optimization of architectures, and could be combined with our scheme which focus on top-level designing.

This paper discusses the automatic top-level hardware design for digital signal processing. The main contributions are summarized below:

1) We propose an automatic scheme for top-level hardware design, which can optimize hardware in a gradient-based way by iteratively adjusting parameters;
2) A hardware description language AHDW is designed to apply timing arrangement, fast evaluation and code generation automatically;
3) A channel impulse response (CIR) estimator for millimeter wave (mmWave) systems is implemented via AHDW with auto top-level design as a demonstration of the proposed methods.

II. Preliminaries

Auto generators can use VLSI design techniques such as folding and unfolding, numerical strength reduction, systolic design, and pipeline design to achieve trade-offs among various design metrics, such as area efficiency, energy efficiency, and power density. Basically, auto generators implement the following three steps:

1) Derive formula representation of the VLSI architecture of the selected algorithm;
2) Convert formula into RTL code through a generation script or program;

3) Obtain hardware implementation through electronic design automation (EDA) tools.

The formula representation of hardware architecture is introduced in [8]. To achieve efficient implementation, the formula should be concise, while fully exploiting algorithm characteristics. An effective approach of formula derivation is to break the architecture into submodules, and rearrange them in top-level. However, the top-level design can be challenging due to complicated timing constraints, which impose difficulties for hardware designers. Furthermore, the formula often contains various design parameters such as parallelism to enable design trade-offs. Optimizing the design parameters can be very challenging due to the large number of parameters. Therefore, we propose an automatic scheme for top-level hardware design including top-level timing arrangement, parameter optimizing and RTL code generation.

III. Proposed Auto Design Methods

A. Timing Arrangement

The basic timing arrangement is shown in Algorithm 1. More complicated scenarios will be discussed in section IV.

Algorithm 1: Basic Timing Arrangement

```
Input: Timing Information, Timing Constraint,
       Hardware Architecture
1  Generate graph representation of the architecture;
   // Timing-Oriented Optimization
2  while timing constraint not satisfied do
3      Find deciding path $S$ concerning timing constraint;
4      repeat
5          Adjust one parameter in $S$;
6      until Updated $S$ no longer deciding path;
7  end
   // Area-Oriented Optimization
8  foreach module not on deciding path do
9      Adjust parameters for smaller areas while
         maintaining timing constraint-compliant;
10 end
Output: Constraint-Compliant Hardware Design
```

Before performing timing optimization, it is necessary to represent the hardware architecture as a weighted *directed graph* in step 1. A comprehensive understanding of the topology is crucial for optimization purposes, as it enables module reuses and facilitates the deployment of parallelism and pipelining.

The main process consists of two steps: timing-oriented optimization and area-oriented optimization. These processes are illustrated in Fig. 1.

In timing-oriented optimization, all paths of the design are explored, and respective timing metrics such as latency and pipeline interval are calculated. The deciding path is then determined as the path with the longest latency or pipeline interval. It is worth noting that the timing considered here is based on clock cycles, different from the critical path,

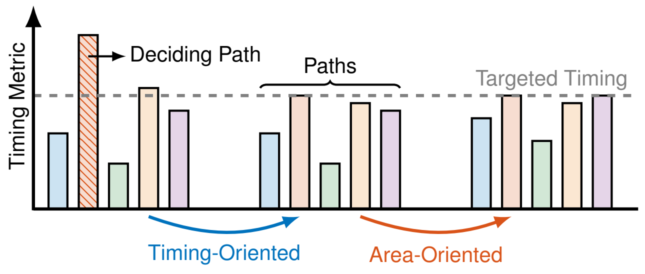


Fig. 1. Illustration of timing arrangement.

which is the longest physical path from register to register. Parameters along the deciding path are adjusted to meet the timing requirement. This process is repeated until all paths in the design comply with the timing requirement.

After timing-oriented optimization, area-oriented optimization can be performed to relax certain parameters and reduce the area while maintaining the hardware's adherence to the timing constraint.

For latency optimization, the deciding path is the longest path in the weighted graph. When considering throughput optimization, the pipeline interval becomes a primary factor. Therefore, the deciding path encompasses the submodule with the maximum pipeline interval.

B. Fast Evaluation

In B5G/6G wireless communications systems, the hardware architecture for baseband signal processing could be extremely complicated, making thorough evaluation of which highly time consuming. Therefore, fast evaluation of hardware designs are critical for design space exploration. Since the optimization of reconfigurable parameters is based on (smart) searching, we have to determine the area metric efficiently. Fast evaluation does not rely on synthesis but is based on the rough estimation of each module and calculates the total area. The rough estimation can be achieved by synthesizing basic modules with certain parameters, or by expert experience.

C. Gradient-Based Optimization

Since the overall quality of a generated hardware design encompasses multiple aspects such as performance, power, area, throughput, and latency, it is challenging to define a single optimal hardware design. Consequently, it is more practical to obtain a set of Pareto-optimal hardware designs for users to choose from in this multi-objective optimization problem.

In general, considering an objective function f, the requirements for the top-level hardware design can be represented as the following optimization problem:

$$\max f(t_D, a_D), \quad \text{s.t. } t_D \in \mathcal{T}, a_D \in \mathcal{A}, \tag{1}$$

where t_D and a_D are the timing and area parameters for design D, respectively, and $\mathcal{T}$ and $\mathcal{A}$ are the constraint sets for timing and area, respectively. For the sake of analysis convenience, the function f is assumed to have a gradient for the design D. Unlike gradient descent, which aims to minimize the loss function [9], the parameters t_D and a_D in (1) are correlated, and the direction with the maximum gradient is often infeasible for a hardware module. When updating the design to D', the changes in timing and area parameters are denoted as:

$$\Delta t \triangleq t_{D'} - t_D, \quad \Delta a \triangleq a_{D'} - a_D. \tag{2}$$

To increase the objective function f, the changes need to satisfy

$$\nabla f(t_D, a_D) \cdot \begin{bmatrix} \Delta t \\ \Delta a \end{bmatrix} > 0, \tag{3}$$

where $\cdot$ is the dot product. When $\Delta t \neq 0$, indicating that the design change affects the deciding path, (3) can be organized assuming $\Delta t \Delta a < 0$ as follows:

$$\begin{cases} -\dfrac{\Delta t}{\Delta a} > \dfrac{\frac{\partial f(t_D, a_D)}{\partial t_D}}{\frac{\partial f(t_D, a_D)}{\partial a_D}}, & \text{if } \Delta t < 0, \Delta a > 0, \\ -\dfrac{\Delta t}{\Delta a} < \dfrac{\frac{\partial f(t_D, a_D)}{\partial t_D}}{\frac{\partial f(t_D, a_D)}{\partial a_D}}, & \text{if } \Delta t > 0, \Delta a < 0. \end{cases} \tag{4}$$

Thus, parameters for a particular module can be optimized for either timing or area, based on (4), by comparing $-\Delta t/\Delta a$ with a threshold.

IV. Implementation and Discussions

A. Implementation

Summarizing the proposed methods as well as hardware auto generation, a hardware description language AHDW (**A**uto **H**ar**DW**are) is developed, where its overall design is depicted in Fig. 2, combining timing arrangement, automatic generation and fast evaluation.

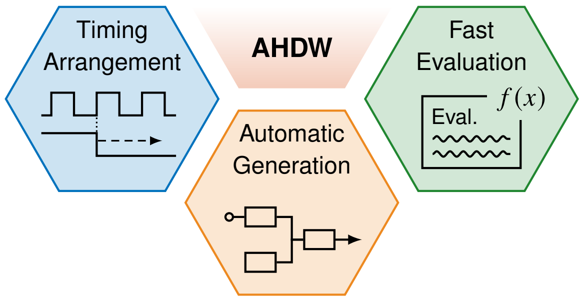


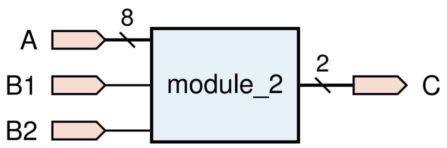
Fig. 2. Overview of AHDW.

AHDW is a configurable framework that follows a modular structure similar to a hardware description language. It comprises a top module and functional submodules, where the top module describes the entire algorithm, and each submodule represents an operation in the formula. AHDW offers support for high-level behavior descriptions and Verilog syntax simultaneously, enabling designers with extensive freedom to meet custom design objectives.

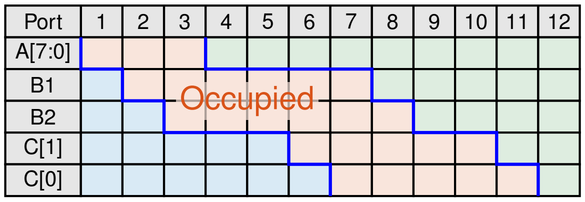
The main challenge lies in obtaining timing information and implementing optimizations in a programmatic manner. We base our directed graph implementation on the C++ library in [10]. Fig. 3 illustrates an example AHDW module declaration, which includes a configurable parameter p and ports A, B (disaggregated), and C. Timing data is provided by specifying the arrival clock for incoming data (`tin`) and the clock for the next frame (`tnext`). A time table can be used as a visual representation of timing information, as shown in Fig. 3(c). This table captures the occupied clocks, allowing for clever arrangement of timing by utilizing pipelining techniques.

```
MODULE module_{{ p }} // p is a parameter
  p = PARAM min=2 max=8 step=2
  port_A = PORT in 8 tin=0 tnext=p+1
  i = LOOP from=1 to=p
    port_B{{ i }} = PORT in tin=i tnext=i+6
  END
  port_C = PORT out p tout=2*p+7-b tnext=p+4-b
  // module implementations ...
END
```

(a) AHDW code.



(b) Corresponding module representation with $p = 2$.



(c) Time table of the module with $p = 2$.

Fig. 3. AHDW module declaration example.

B. Discussions

During the early concept design phase, there are several aspects in which the AHDW compiler implementation can be further extended:

1) Dynamic Timing: In contrast to the assumption made in Section III-A, where all timing information is assumed to be known, many modules involve dynamic timing. This means that the latency and/or pipeline interval are not predetermined. In such cases, handshake signals can be utilized instead of inserting registers to handle the dynamic timing requirements.

2) Loop: The treatment of loops in the architecture can be summarized as the process of *replica* and *merging*. Different hardware resources are allocated for each iteration, but when the timing requirements are met, certain modules can be automatically merged. If the iteration loop is regular or the timing requirements are not tight, there will be no additional hardware cost due to replica.

3) Reconfigurable Modules: The proposed method can also be applied to designs that employ reconfigurable modules. As long as timing information is provided, reconfigurable modules are considered as distinct functional modules but can be merged during the area-oriented optimization process.

4) Fast Compilation Mode: To expedite the top-level design process for large hardware architectures, a fast compilation mode can be introduced. This mode significantly reduces the design space by making more radical assumptions and pruning less likely designs at an earlier stage.

V. Demonstration

We demonstrate the effectiveness of our proposed top-level hardware design method by implementing Golay sequence aided CIR estimation for mmWave frequency-selective systems. This technique is utilized in IEEE Standard 802.11ad [11] for wireless local area network (WLAN). A real-time testbed for this purpose is implemented in [12]. Additionally, a channel estimator for wireless personal area networks (WPAN) is proposed in [13], which includes a parallel design of Golay correlators.

The Golay sequence aided CIR estimation can be reconfigured, similar to other auto-generated hardware designs. Among the important parameters, parallelism holds significant importance. AHDW facilitates the automatic generation of the top-level design of the CIR estimator for different parallelism settings. Furthermore, it allows for intelligent reuse of Golay correlators among I/Q components, providing the option to trade-off latency for a smaller area.

Fig. 4 presents the FPGA synthesis results and fast evaluation results using Vivado 2022.2 on Zynq UltraScale+ RFSoC ZCU111 [14]. The results demonstrate different parameter settings and hardware arrangements, showcasing the trade-off between area and latency, with a Pareto front. The fast evaluation values are normalized to the synthesis results.

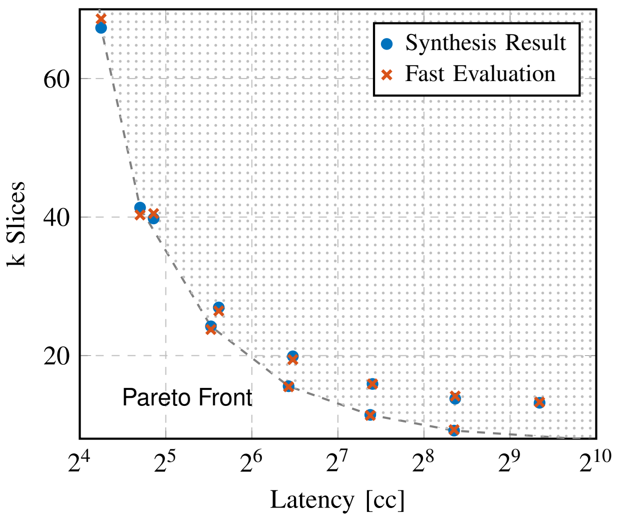


Fig. 4. Comparisons of different top-level designs for Golay sequence aided CIR estimation.

VI. Conclusion

In this paper, we propose an effective automatic top-level hardware design method that combines the graph features of the overall hardware architecture and specific module timing characteristics to offer efficient solutions. To implement our proposed method, we developed a hardware generation language AHDW that serves as a platform for auto generator developing and hardware optimizing. Furthermore, we demonstrates the effectiveness of our method by implementing a Golay sequence aided CIR estimator with AHDW. The current design is only at its early age and has much potential to be further enhanced. Detailed implementation optimizing is left for future work.

References

[1] Y. Fu, K. Chen, W. Song, G. He, S. Shen, H. Wang *et al.*, "A DSP-purposed reconfigurable acceleration machine (DREAM) for high energy efficiency MIMO signal processing," *IEEE Trans. Circuits Syst. I*, vol. 70, no. 2, pp. 952–965, Feb. 2023.

[2] C. Ji, Y. Shen, Z. Zhang, X. You, and C. Zhang, "Autogeneration of pipelined belief propagation polar decoders," *IEEE Trans. VLSI Syst.*, vol. 28, no. 7, pp. 1703–1716, Jul. 2020.

[3] Z. Zhong, W. J. Gross, Z. Zhang, X. You, and C. Zhang, "Polar compiler: Auto-generator of hardware architectures for polar encoders," *IEEE Trans. Circuits Syst. I*, vol. 67, no. 6, pp. 2091–2102, Jun. 2020.

[4] Y. Ge, Z. Ji, Y. Huang, Z. Zhang, X. You, and C. Zhang, "Automatic hybrid-precision quantization for MIMO detectors," *IEEE Trans. Signal Process.*, vol. 71, pp. 1039–1052, Mar. 2023.

[5] S. Lahti, P. Sjövall, J. Vanne, and T. D. Hämäläinen, "Are we there yet? a study on the state of high-level synthesis," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 38, no. 5, pp. 898–911, May 2019.

[6] H. Mehra and M. S. Bhat, "High level optimization methodology for high performance DSP systems using retiming techniques," in *Proc. IEEE Dist. Comput. VLSI Electr. Circ. Robot. (DISCOVER)*, Aug. 2018, pp. 163–168.

[7] S. Jalaja and A. M. V. Prakash, "Design of low power based VLSI architecture for constant multiplier and high speed implementation using retiming technique," in *Proc. IEEE Int. Conf. Microelectron. Comput. Commun. (MicroCom)*, Jan. 2016, pp. 1–6.

[8] M. Puschel, J. Moura, J. Johnson, D. Padua, M. Veloso, B. Singer *et al.*, "SPIRAL: Code generation for DSP transforms," *Proc. IEEE*, vol. 93, no. 2, pp. 232–275, Feb. 2005.

[9] S. Ruder, "An overview of gradient descent optimization algorithms," 2016, *arXiv:1609.04747*. [Online]. Available: https://arxiv.org/abs/1609.04747

[10] W. Zhao. (2023, Apr.) DG-CPP: Directed graph in C++. [Online]. Available: https://dg-cpp.tvj.one

[11] *IEEE Standard for Information technology – Telecommunications and information exchange between systems – Local and metropolitan area networks – Specific requirements – Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications – Amendment 3: Enhancements for Very High Throughput in the 60 GHz Band*, IEEE Std. 802.11ad, 2012.

[12] J. O. Lacruz, R. R. Ortiz, and J. Widmer, "A real-time experimentation platform for sub-6 GHz and millimeter-wave MIMO systems," in *Proc. ACM Int. Conf. Mob. Syst. Appl. Serv.*, Jun. 2021, pp. 427–439.

[13] W.-C. Liu, F.-C. Yeh, T.-C. Wei, C.-D. Chan, and S.-J. Jou, "A digital Golay-MPIC time domain equalizer for SC/OFDM dual-modes at 60 GHz band," *IEEE Trans. Circuits Syst. I*, vol. 60, no. 10, pp. 2730–2739, Oct. 2013.

[14] AMD. (2023) Zynq UltraScale+ RFSoC ZCU111 evaluation kit. [Online]. Available: https://www.xilinx.com/products/boards-and-kits/zcu111.html