

Feasibility of Automated Synthesis in Ultra-Low Power Application-Specific Embedded Processor Design

Tom J Kazmierski and Xuan Ji

Electronics & Computer Science University of Southampton, Southampton, United Kingdom

Email: T.J.Kazmierski,xj2n21@soton.ac.uk

Abstract—Existing automatic generation methods for application-specific processors primarily enhance design metrics through instruction set extensions and customization. These approaches typically treat hardware architecture as a fixed entity, constraining the optimization space. We propose a new approach which adopts a global optimization perspective, collaboratively exploring interactions among algorithms, instructions, hardware architectures, and encoding to uncover a broader range of superior designs. Most state-of-the-art methods focus on utilizing High-Level Synthesis (HLS) to generate data for Design Space Exploration (DSE). Although HLS DSE can optimize designs locally, also - given enough data - using AI techniques, HLS DSE optimization generally suffers from low fidelity so alternative approaches are needed. We base our automation on automatic design generation, from assembly to RTL level, using multiple simulations and synthesis loops, with the objective of selecting processor designs optimized for size or power consumption. Unlike random design generation methods, our proposed method ensures that no resources are wasted on analysing functionally incorrect designs.

We have implemented a simple case study, in which 40 different processor designs were automatically generated with extremely small sizes, by more than an order of magnitude smaller than state-of-the-art ultra-low power processors. The results show potential for automatic generation of large numbers of designs to provide data for further application-specific optimization.

Index Terms—Embedded processors, ultra-low power, application specific processors, automated synthesis, design space exploration, optimisation.

I. INTRODUCTION

High-Level Synthesis (HLS) and Design Space Exploration (DSE) methodologies in recent years tend to evolve towards more efficient approaches to solving optimisation problems by enhancing the traditional methods with Artificial Intelligence (AI) algorithms and Machine Learning (ML). HLS DSE also widely uses a variety of modeling techniques and new design generation mechanisms for faster speeds [1].

However, the fidelity of HLS tools facing complex design tasks remains a problem and has been exposed as the source of low accuracy [2]. Even different versions of the same HLS tool might use different strategies and yield different results. At the same time, the low accuracy of most DSE algorithms using free FPGA-based HLS tools reduces their significance. HLS appears to be more successful in software development, while application-specific hardware designed by HLS is merely passable [3]. It is possible to employ other heuristic methods

such as Genetic Algorithms (GA) with minimal data volume requirements instead of ML, which has considerable potential but high data volume requirements. However GA still requires access to high-quality datasets [4]. Our proposed method does not use heuristics. Instead it relies on a multi-tier processor design generation process from the assembly level to the RTL level as explained in some detail below.

It should be noted that Application-Specific Embedded Processors (ASEP) have traditionally been designed by hand. Instead of automation, expert knowledge and extensive verification is used to generate processors handcrafted for a special purpose [5]–[9]. A design process that requires expert knowledge is tedious and, therefore, expensive [10], [11]. But it is done despite the cost because ASEPs offer significant advantages. Firstly, unlike general-purpose processors, ASEPs have unique and simple architectures that make them resilient to cybersecurity threats that affect both software and hardware [12]. Secondly, handcrafted ASEPs can achieve excellent performance and power efficiency characteristics that can exceed those of HLS-generated application-specific hardware.

There have been a number of attempts to develop automated synthesis tools to generate ASEPs [13]–[15] as an alternative to HLS-generated hardware. However, further research is needed to alleviate the need for manual adjustment, achieve better performance, and reduce the power consumption.

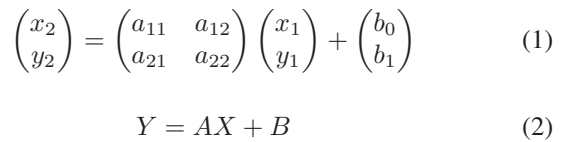
II. PROPOSED SYNTHESIS TECHNIQUE

We illustrate our approach using a simple case study where embedded processor designs are automatically generated and synthesised to execute a 2D affine transformation algorithm.

The salient feature of the proposed approach is to automatically generate a large number of compliant designs for a design requirement and then find optima according to performance metrics.

In this case study the choice architectural options - such as the one presented in Figure 1 - is limited and explores only several pre-defined variants of RISC and NISC (No Instruction Set Computer) structures. In each architectural option various instruction sets and hardware block parameters are selected to generate multiple designs as explained below.

The synthesis process selects iteratively an architectural option and for each option generates feasible instruction sets. The application algorithm is then presented in the form of an assembly program and then RTL-level block generations are



```
graph TD
    subgraph Entire
        direction TB
        Start([Start]) --> Load[LOAD  
Height y & Width x]
        Load --> InitY[Initialisation  
%y1=0]
        InitY --> StoreY1[Store %y1 to  
y1]
        StoreY1 --> YLoop{ }
        YLoop -- 1 --> XLoop
        YLoop -- 0 --> Done([Done])
    end

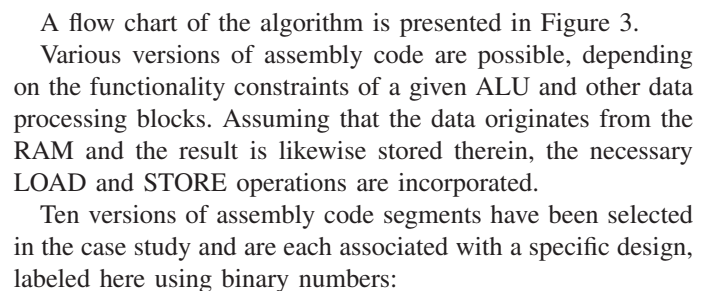
    subgraph y_loop [y loop]
        direction TB
        InitX1[Initialisation  
%x1=0]
        StoreX1[Store %x1 to  
x1]
        LoadX4[Load x1 to  
%4]
        CalcStoreX2[Calculation &  
Store x2]
        LoadY4[Load y1 to  
%4]
        CalcStoreY2[Calculation &  
Store y2]
        IncStoreX1[%x1+1 &  
Store]
        DecX1{ }
        IncStoreY1[%y1+1 &  
Store]

        InitX1 --> StoreX1
        StoreX1 --> LoadX4
        LoadX4 --> CalcStoreX2
        CalcStoreX2 --> LoadY4
        LoadY4 --> CalcStoreY2
        CalcStoreY2 --> IncStoreX1
        IncStoreX1 --> DecX1
        DecX1 -- 1 --> StoreX1
        DecX1 -- 0 --> IncStoreY1
    end

    subgraph x_loop [x loop]
        direction TB
        LoadX4
        CalcStoreX2
        LoadY4
        CalcStoreY2
        IncStoreX1
    end

    DecX1 -- 1 --> StoreX1
    DecX1 -- 0 --> IncStoreY1
    IncStoreY1 --> YLoop
```

Fig. 3. Implementation of 2D Affine Transformation Algorithm for image processing.



```

0000: MULI  %4=%4*a00; ACCZ acc=0; ACC acc+=%4; SIR
      %4=b0; ACC acc+=%4;
0001: MULI  %4=%4*a00; ACC0 acc=0+%4; SIR  %4=b0;
      ACC acc+=%4;
0010: MULI  %4=%4*a00; ACCZ acc=0; ACC acc+=%4; ACCI
      acc+=b0;
0011: MULI  %4=%4*a00; ACC0 acc=0+%4; ACCI acc+=b0;
0110: MULI  %4=%4*a00; ACCZ acc=0; ACCI acc+=b0; ACC
      acc+=%4;
0111: MULI  %4=%4*a00; ACCI0 acc=0+b0; ACC acc+=%4;
1000: ACCZ acc=0; MACI acc+=%4*a00; SIR  %4=b0; ACC
      acc+=%4;
1001: MACI0 acc=0+%4*a00; SIR  %4=b0; ACC acc+=%4;
1010: ACCZ acc=0; MACI acc+=%4*a00; ACCI acc+=b0;
1011: MACI0 acc=0+%4*a00; ACCI acc+=b0;

```

The application selected as a case study is an affine transformation of a two-dimensional image with a maximum size of 256 by 256 pixels. Details of the application-specific arithmetic utilized in the hardware calculations are as follows.

For each selected instruction format, the size of each necessary bit field is calculated to obtain the machine format and the size of the program counter and program memory. Two branching styles are implemented, the RISC V one and that used by MIPS. These choices affect the composition of the ALU-related modules. The I/O ports and function lists for all modules of all designs are stored in a dictionary for ease of later retrieval. The generated I/O and function lists of decoders for

each design are essentially a distribution of the corresponding modular decomposition of all the operations performed in a design, which is also added to the dictionary.

In contrast to RISC, NISC does not require the generation of an instruction decoder. In an NISC implementation, not only is the opcode replaced by the control word, but the appropriate bit width and start position of the function select signal required by each multifunction submodule are also determined by means of cyclic scanning.

IV. POST-SYNTHESIS ANALYSIS AND OPTIMISATION

For each design, all necessary parameters and connections are extracted and stored in the design and test files. Firstly, a test file is created with the corresponding addresses of all required design files and the total simulation time. Then, similarly to the previous step, the ports are added to the list, which can later be viewed as a textualised waveform diagram to display simulation results. Modelsim is run repeatedly, and specific generation steps based on errors are iteratively modified until there are no errors. The simulation script is run to verify that each design works correctly. As the number of designs increases, the latter step can be further developed in the future to provide automatic functionality verification. The Quartus synthesis script has 6 configurable parts, which are: address, number of designs required to synthesize, all design file names and the top-level entity name, power estimation instruction, and I/O ports binding assertion.

Key performance metrics are then extracted from the Quartus results report file, including the total number of ALMs along with specific overhead distribution, power consumption, total number of DSPs along with the specific arrangement, and maximum clock frequency at two different temperatures. The results are saved for subsequent extraction of the optimal designs.

V. RESULTS

Although the number of designs that have been generated in this study is relatively modest, only 40, it should be noted that the automatic synthesis framework proposed here is extensible and that the total number of designs ultimately generated can be much greater at different stages, resulting from different choices. Consequently, it may be reasonably anticipated that 3,000 or 4,000 distinct designs satisfying the same functional requirements can be generated using straightforward extensions. The generated differ in terms of details but have similar block diagrams. The block diagram for the design '0000' in table I is provided in figure 4 as an example.

The results of Quartus synthesis for the ten RISC variants are presented in I for Stratix V. Based on the concept of Pareto optimality, the three optimal solutions are identified in bold. Although there is no appreciable improvement over equivalent fine-tuned, hand-optimized designs, the framework for automatic generation of different designs is a logical approach as it saves time and reduces the need for highly skilled engineers. As illustrated in the two tables concerning RISC, I and III, the overhead of ALM ranges from 94 to 121, which supports the rationale for this approach.

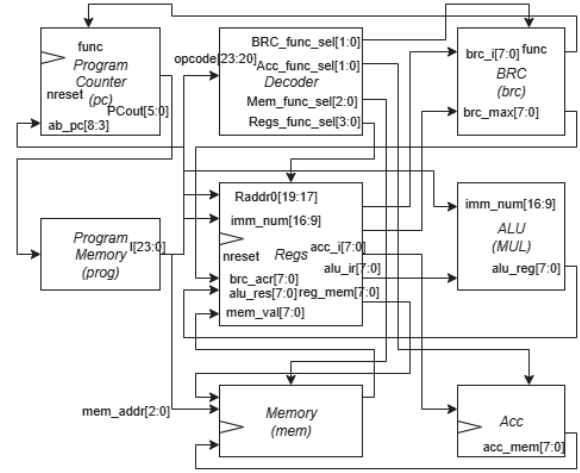


Fig. 4. The block diagram of RISC design0000

TABLE I
MAIN METRICS OF 10 RISC DESIGNS WITH RISC-V BRANCHING STYLE

Name	ALM	P/mW	DSP	$F_{\max}85^{\circ}\text{C}/\text{MHz}$	$F_{\max}0^{\circ}\text{C}/\text{MHz}$
design0000	110	1049.17	9x1	260.89	262.05
design0010	121	1050.52	9x1	252.97	252.72
design0110	119	1049.86	9x1	246	252.08
design0111	104	1048.65	9x1	266.1	267.67
design0011	94	1048.71	9x1	265.32	266.03
design0001	113	1048.67	9x1	263.23	266.1
design1000	101	1049.5	9x1	234.8	235.96
design1010	107	1049.68	9x1	243.07	244.02
design1001	120	1048.59	9x1	235.74	238.21
design1011	98	1049.49	9x1	248.14	250.31

TABLE II
MAIN METRICS OF 10 NISC DESIGNS WITH RISC-V BRANCHING STYLE

Name	ALM	P/mW	DSP	$F_{\max}85^{\circ}\text{C}/\text{MHz}$	$F_{\max}0^{\circ}\text{C}/\text{MHz}$
design0000	112	1048.42	9x1	251.89	254.13
design0010	110	1048.46	9x1	259	262.19
design0110	108	1048.5	9x1	257.73	258.4
design0111	101	1048.2	9x1	234.96	238.78
design0011	102	1048.24	9x1	250.5	252.72
design0001	104	1048.45	9x1	256.61	257.33
design1000	120	1048.99	9x1	240.73	242.9
design1010	127	1049.52	9x1	236.46	236.63
design1001	120	1048.37	9x1	241.6	245.7
design1011	119	1048.73	9x1	231.48	234.69

TABLE III
MAIN METRICS OF 10 RISC DESIGNS WITH MIPS BRANCHING STYLE

Name	ALM	P/mW	DSP	$F_{\max}85^{\circ}\text{C}/\text{MHz}$	$F_{\max}0^{\circ}\text{C}/\text{MHz}$
design0000	117	1050.15	9x1	254.52	253.74
design0010	110	1049.99	9x1	259.54	260.96
design0110	109	1049.35	9x1	269.47	269.47
design0111	106	1049.81	9x1	248.51	250.38
design0011	94	1049.88	9x1	256.87	261.3
design0001	117	1049.94	9x1	251.95	254.26
design1000	110	1051.62	9x1	247.77	249.07
design1010	106	1049.65	9x1	245.94	245.76
design1001	113	1050	9x1	244.14	246.12
design1011	102	1050.01	9x1	232.02	235.13

TABLE IV
MAIN METRICS OF 10 NISC DESIGNS WITH MIPS BRANCHING STYLE

Name	ALM	P/mW	DSP	$F_{max}85c/MHz$	$F_{max}0c/MHz$
design0000	113	1048.97	9x1	229.99	234.96
design0010	114	1049.36	9x1	258.4	259.27
design0110	111	1048.41	9x1	245.76	249.81
design0111	108	1049.1	9x1	233.7	236.91
design0011	113	1049.1	9x1	263.78	266.6
design0001	116	1048.92	9x1	251.26	254.32
design1000	142	1049.66	9x1	261.64	260.62
design1010	116	1048.96	9x1	238.61	239.64
design1001	130	1049.76	9x1	241.78	244.26
design1011	122	1050.43	9x1	234.8	237.42

Compared with the RISC designs, the corresponding NISC designs do not seem to have an advantage in the number of ALMs, but in terms of power consumption, they are all smaller, except for the design '1011' in IV. However, these trends are not conclusive because the sample size is small and the 'control word' part in instructions is simply spliced without any targeted optimisation.

VI. CONCLUSION

Compared with traditional ASEP synthesis, which primarily explores instruction-set extensions, our method explores a broader design space by co-optimizing algorithmic expressions, hardware units, and micro-architecture. The primary outcome is an illustration of the feasibility of generating diverse designs through the parsing of computational and storage operations. Based on the experience of this study, it can be postulated that even for simple applications, the potential number of application-specific designs that can be generated is likely to be in the range of several thousand, far beyond the scope of traditional ASEP generation. The implementation process becomes increasingly complex and the code more extensive to facilitate a greater number of potential processors. Compared to random generation methods described in the literature, e.g. [16], the proposed approach method ensures that every design is feasible in terms of framework and principle, thus saving unnecessary computation.

REFERENCES

- [1] B. C. Schafer and Z. Wang, "High-level synthesis design space exploration: Past, present, and future," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 10, pp. 2628–2639, Oct. 2020.
- [2] Q. Sun et al., "Correlated multi-objective multi-fidelity optimization for HLS directives design," in *Proc. 2021 Design, Automation & Test in Europe (DATE)*, Grenoble, France, 2021, pp. 46–51.
- [3] A. Sohrabizadeh, C. H. Yu, M. Gao, and J. Cong, "AutoDSE: Enabling software programmers design efficient FPGA accelerators," in *Proc. ACM/SIGDA Int. Symp. Field-Programmable Gate Arrays (FPGA)*, Virtual Event, USA, 2021, p. 147.
- [4] M. I. Rashid and B. C. Schafer, "Fast and inexpensive high-level synthesis design space exploration: Machine learning to the rescue," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 42, no. 11, pp. 3939–3950, 2023.
- [5] S. Khokhriakov, R. Reddy Manumachu, and A. Lastovetsky, "Multicore processor computing is not energy proportional: An opportunity for bi-objective optimization for energy and performance," *Appl. Energy*, vol. 268, p. 114957, 2020.
- [6] B. Dupont de Dinechin, C. Monat, P. Blouet, and C. Bertin, "DSP-MCU processor optimization for portable applications," *Microelectron. Eng.*, vol. 54, no. 1, pp. 123–132, 2000.
- [7] V. K. Mishra and A. Sengupta, "MO-PSE: Adaptive multi-objective particle swarm optimization based design space exploration in architectural synthesis for application specific processor design," *Adv. Eng. Softw.*, vol. 67, pp. 111–124, 2014.
- [8] D. Llamocca and M. Pattichis, "A dynamically reconfigurable pixel processor system based on power/energy-performance-accuracy optimization," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 23, no. 3, pp. 488–502, Mar. 2013.
- [9] M. Anguita, J. Diaz, E. Ros and F. J. Fernandez-Baldero, "Optimization strategies for high-performance computing of optical-flow in general-purpose processors," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 19, no. 10, pp. 1475–1488, Oct. 2009.
- [10] S. Saponara, L. Fanucci, S. Marsi, G. Ramponi, D. Kammler, and E. M. Witte, "Application-specific instruction-set processor for retinex-like image and video processing," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 54, no. 7, pp. 596–600, July 2007.
- [11] T. Pitkanen and J. Takala, "Low-power application-specific processor for fft computations," in *2009 IEEE International Conference on Acoustics, Speech and Signal Processing*, April 2009, pp. 593–596.
- [12] D. Meyer, J. Haase, M. Eckert, and B. Klauer, "A modern approach to application specific processors for improving the security of embedded devices," in *IECON 2021 – 47th Annual Conference of the IEEE Industrial Electronics Society*, 2021, pp. 1–6.
- [13] M. Rykunov, A. Mokhov, A. Yakovlev, and A. Koelmans, "Automated generation of processor architectures in embedded systems design," *Technical Report NCL-EECE-MSD-TR-2010–164*, Newcastle University, 2010.
- [14] A. Hirvonen, T. Leppänen, K. Hepola, J. Multanen, J. Hoozemans, and P. Jääskeläinen, "Aex: Automated high-level synthesis of compiler programmable co-processors," *Journal of Signal Processing Systems*, vol. 95, no. 9, pp. 1051–1065, 2023.
- [15] C. Mutigwe et al., "Automatic synthesis of application-specific processors," Ph.D. dissertation, Bloemfontein: Central University of Technology, Free State, 2012.
- [16] Z. Bao and T. Watanabe, "A new approach for circuit design optimization using genetic algorithm," in *Proc. 2008 International SoC Design Conference*, vol. 1, pp. I-383–I-386, 2008.
- [17] J. Op den Brouw, "The THUAS RISC-V 32-bit processor in VHDL," GitHub repository, v1.1.1.0 (Commit: 880cdf2), Jan. 2025. [Online]. Available: <https://github.com/jesseopdenbrouw/thuas-riscv> [Accessed: Mar. 12, 2025].
- [18] R. Jia, C. Y. Lin, Z. Guo, R. Chen, F. Wang, T. Gao, and H. Yang, "A survey of open source processors for FPGAs," in *Proc. 2014 24th International Conference on Field Programmable Logic and Applications (FPL)*, pp. 1–6, 2014.
- [19] P. Navarro-Torrero, M. C. Martínez-Rodríguez, Á. Barriga-Barros, and P. Brox, "Full open-source implementation of an academic RISC-V on FPGA," in *Proc. XVI Congr. Tecnol. Aprendizaje Enseñanza Electron. (TAAE)*, Malaga, Spain, 2024, pp. 1–5, doi:10.1109/TAAE59541.2024.10604934.
- [20] M. Hutton, J. Schleicher, D. Lewis et al., "Improving FPGA performance and area using an adaptive logic module," in *Proc. Int. Conf. Field Program. Logic Appl. (FPL)*, 2004, pp. 135–144.
- [21] F. Hussain and S. Sarkar, "Design and FPGA implementation of five stage pipelined RISC-V processor," in *Proc. 9th Int. Conf. Convergence Technol. (I2CT)*, Pune, India, 2024, pp. 1–6.
- [22] I. Elsadek and E. Y. Tawfik, "RISC-V resource-constrained cores: A survey and energy comparison," in *Proc. 19th IEEE Int. Conf. New Circuits Syst. (NEWCAS)*, Montreal, QC, Canada, 2021, pp. 1–5, doi:10.1109/NEWCAS51681.2021.9532772.
- [23] L. Poli, S. Saha, X. Zhai, and K. D. McDonald-Maier, "Design and implementation of a RISC-V processor on FPGA," in *Proc. 17th Int. Conf. Mob. Sens. Netw. (MSN)*, Osaka, Japan, 2021, pp. 161–166, doi:10.1109/MSN53354.2021.00034.
- [24] G. Hadjiyiannis, "An architecture synthesis system for embedded processors," Ph.D. dissertation, Dept. Elect. Eng. Comput. Sci., Massachusetts Institute of Technology, Cambridge, MA, 2000.
- [25] D. Goodwin and D. Petkov, "Automatic generation of application specific processors," in *Proc. Int. Conf. Compilers, Architecture and Synthesis for Embedded Systems (CASES)*, San Jose, CA, USA, Oct. 2003, pp. 137–147, doi: 10.1145/951710.951730.