Automatic Creation of Domain-Specific Reconfigurable CPLDs for SoC

Mark Holland, Scott Hauck
Department of Electrical Engineering
University of Washington, Seattle, WA 98195, USA
mholland@ee.washington.edu, hauck@ee.washington.edu

1. Introduction

Reconfigurable logic fills a useful niche between the flexibility provided by a processor and the performance provided by custom hardware. This usefulness extends to the SoC realm, where reconfigurable logic can provide cost-free upgradability, conformity to different but similar protocols, coprocessing hardware, and uncommitted testing resources. In SoC designs the computational domain is already known, so an opportunity exists for creating domain-specific reconfigurable logic tailored to the specific domain - thereby removing unneeded flexibility and improving performance. The dilemma then becomes creating these reconfigurable fabrics in a short enough time that they can be useful to SoC designers.

This work presents the creation of domain-specific CPLD architectures, a project termed Totem-CPLD (part of the larger Totem project at UW). For the scope of this work, a CPLD is a collection of PLAs connected by a full crossbar, and they are made "domain-specific" by altering the input, product term, and output capacities of the PLAs in the architecture.

2. Background

As a precursor to Totem-CPLD, we performed work in which we explored the feasibility of making domain-specific reconfigurable PLAs and PALs [1]. By intelligently removing 60%-70% of the programmable connections in the PLA/PAL arrays we were able to provide delay gains of 15%-30%. Depopulating the arrays in a PLA is very restrictive to future mappings, however, so we chose not to use PLA depopulation in Totem-CPLD.

We will be using a tool called PLAmap to perform our tech-mapping, as it is currently the best academic tech-mapping algorithm for CPLDs [2]. PLAmap is a performance driven mapping algorithm whose goal is to minimize the delay/depth of the mapped circuit. It is run by providing a PLA size (inputs, product terms, outputs) and a circuit (in .blif format) to be mapped, and it returns the mapping, which includes the number of PLAs required and the depth of the mapping.

3. Approach – Tool Flow

The tool flow for Totem-CPLD is as follows. To begin, the SoC designer provides us with a domain specification that contains the circuits that need to be supported. These circuits are fed into an Architecture Generator, which finds a CPLD architecture that provides good results for the selected domain, outputting the architecture description. The architecture description is then sent to a Layout Generator which creates a full VLSI layout of the specified CPLD architecture. The full VLSI layout is then returned to the designer as "IP" to be incorporated into the SoC device.

The Architecture Generator is responsible for reading in multiple circuits and finding a CPLD architecture that supports the circuits efficiently. Search algorithms are used to make calls to PLAmap, after which the results are analyzed according to area and delay models that we have developed. We developed four Architecture Generation algorithms for this work: Hill Descent, Successive Refinement, Choose N Regions, and Run M Points. All of the algorithms break the search into three sequential steps by searching for good input, output, and product term sizes, in that order. The searching done in the input step always uses PLAs with a 1x-2x-.5x IN-PT-OUT ratio (found through experimentation), while the output and product term steps always alter ONLY the output and product term values from data point to data point.

The Hill Descent algorithm searches each 1-D space by testing different PLA architectures and following the slope of the results down until a valley is reached, where it stops. The Successive Refinement algorithm sweeps each 1-D space for results and trims sub-optimal regions from the edges of the 1-D space, refining until it searches at maximum granularity. The Choose N Regions algorithm sweeps each 1-D space for results and chooses N regions to explore further, continually doing this until maximum granularity is reached (akin to breadth first search, N=2 used for results). The Run M Points algorithm sweeps each 1-D space for results and then repeatedly explores architectures next to the current best architecture, running until M points have been explored (akin to depth first search, M=15 used for results).

Table 1. Architecture results for domain-specific algorithms and fixed architectures

	Algorithms												Fixed Architectures		
	Hill Descent			Succ. Refinement			Choose N Regions			Run M Points			10-12-4	10-20-5	36-48-16
Domain	Arch	A*D	Runs	Arch	A*D	Runs	Arch	A*D	Runs	Arch	A*D	Runs	A*D	A*D	A*D
Combinational	12-25-4	2.38	13	12-71-4	1.00	91	12-71-4	1.00	40	5-16-2	2.34	49	10.72	4.14	10.91
Sequential	12-23-6	1.00	11	12-23-6	1.00	79	12-23-6	1.00	38	12-23-6	1.00	50	2.08	1.65	1.54
Floating Point	9-28-5	2.44	15	4-8-1	2.75	88	10-24-2	1.00	67	10-24-2	1.00	85	10.80	6.05	24.17
Arithmetic	10-20-2	1.00	14	10-20-2	1.00	57	10-20-2	1.00	66	10-20-2	1.00	85	37.68	13.58	36.63
Encryption	10-23-3	1.00	13	10-46-3	1.00	63	10-46-3	1.00	39	10-46-3	1.00	51	2.91	1.99	5.26
G Mean / Avg		**1.42**	**13.2**		**1.22**	**75.6**		**1.00**	**50.0**		**1.19**	**64.0**	**7.66**	**4.07**	**9.53**

4. Methodology

We are using five sets or "domains" of benchmarks to evaluate our algorithms – the domains are: combinational, sequential, arithmetic, floating point, and encryption. The domain specific architectures that we create are compared to results obtained by implementing all the designs in fixed CPLD architectures. We use fixed architectures that are composed of 10-12-4 PLAs (as suggested by [3]), 10-20-5 PLAs (as suggested by our preliminary work), and 36-48-16 PLAs (used to model a commercial-like CPLD architecture)

5. Results

For each domain, a PLA architecture was found by using each of our four algorithms. Additionally, we mapped the circuits of each domain to the fixed architectures that we described earlier. These results are shown in Table 1. All results are normalized to the values obtained for the Choose N Regions algorithm.

From Table 1 it is apparent that creating domain-specific CPLD architectures is a win over using fixed architectures. All four algorithms found architectures that outperform the fixed architectures for each domain. Considering the mean performance, the fixed architectures perform 4.1x to 9.5x worse than the Choose N Regions algorithm, which performs the best.

An interesting aside is that the architectures found by our search algorithms are likely to be more efficient than the fixed architectures that we compare our results to in Table 1. But the best architectures found by our algorithms, when used across all domains, still perform 2.0x to 7.0x worse than the domain-specific results. This shows that even if you manage to pick the best possible domain-generic fixed architecture, there is a bound as to how close you can come to domain-specific results – in this case, domain-specific beats fixed architectures by 2x.

6. Conclusion

Our work presents a complete tool flow for creating domain-specific CPLDs for System-on-a-Chip devices. When compared to realistic fixed CPLD architectures, the domain-specific architectures perform 4.1x to 9.5x better in terms of area-delay product. Of the base results, the Choose N Regions algorithm provided the best results in terms of performance, with a runtime that was beaten only by the simple Hill Descent algorithm.

Although not described in depth here, this work also includes a Layout Generator which takes pre-made layout units and tiles them to make full VLSI CPLD layouts in the TSMC .18-micron process.

7. Future Work

Future analysis will need to incorporate power values in order to robustly evaluate our architectures. Additional routing architectures should also be explored, as crossbars are prohibitively large for moderately sized designs. Also, different metrics should be incorporated into the cost function, as area-delay product is just one of many ways to analyze an architecture. Finally, work should be undertaken that explores the resource mix that should be added to these architectures in order to most effectively add flexibility to them.

Acknowledgments

Thanks to Mike Hutton and Swati Pathak at Altera, Steve Wilton, Deming Chen, and Kenneth Eguro. Mark Holland was supported in part by an NSF Fellowship, and Scott Hauck by a Sloan Fellowship.

References

[1] M. Holland, S. Hauck, "Automatic Creation of Reconfigurable PALs/PLAs for SoC", *14th International Conference on Field-Programmable Logic and Applications*, 2004, pp. 536-545.

[2] D. Chen, J. Cong, M. Ercegovac, Z. Huang, "Performance-Driven Mapping for CPLD Architectures", *ACM/SIGDA 9th International Symposium on Field-Programmable Gate Arrays*, 2001, pp. 39-47.

[3] J. Kouloheris, A. El Gamal, "FPGA Performance vs. Cell Granularity", *IEEE Proceedings of the Custom Integrated Circuits Conference*, 1991, pp. 6.2/1-6.2/4.