IPSJ Transactions on System LSI Design Methodology Vol.6 122–126 (Aug. 2013)

[DOI: 10.2197/ipsjtsldm.6.122]

Short Paper

Quantitative Evaluation of Resource Sharing in High-level Synthesis Using Realistic Benchmarks

Yuko Hara-Azumi[1,a) Toshinobu Matsuba[3,†1] Hiroyuki Tomiyama[2]
Shinya Honda[3] Hiroaki Takada[3]

Received: November 28, 2012, Revised: March 8, 2013,
Accepted: April 26, 2013, Released: August 5, 2013

Abstract: For FPGA-based designs generated through high-level synthesis (HLS), effects of resource sharing/unsharing on clock frequency, execution time, and area are quantitatively evaluated for several practically large benchmarks on multiple FPGA devices. Through experiments, we observed five important findings about resource sharing/unsharing, which are contrary to conventional wisdom or have not been sufficiently handled. These five findings will be useful for the further development and advance of the practical HLS technology.

Keywords: high-level synthesis, multiplexer, resource sharing

1. Introduction

For increasing the productivity of embedded systems, circuit designs through high-level synthesis (HLS) and the use of FPGAs as alternative devices of conventional circuits (i.e., ASICs) have been both getting prevalent these days. In HLS, whether or not resources (i.e., functional units (FUs) and registers) are shared affects both performance (i.e., clock frequency and execution cycles) and area of synthesized circuits [1]. For example, in general, resource sharing saves area but may degrade clock frequency due to multiplexer (MUX) insertion, whereas unsharing improves clock frequency but suffers area overhead, especially when using a number of large FUs. These effects are particularly large for FPGA-based designs. It is a well-known fact, but these effects have not been quantitatively evaluated using large benchmarks, because of which even today, strategies for resource sharing are still rough and impractical. For example, for the sake of circuit area reduction, a number of works have presented heuristics on resource sharing (e.g., Refs. [2], [3]), based on which other optimizations (e.g., Ref. [4]) have been also studied. Such existing works are based on a traditional premise that resource sharing can reduce area. However, unfortunately, they may not achieve area reduction as they expected, due to the observations we will show later in this paper.

This paper, to the best of our knowledge, presents a first attempt to quantitatively evaluate effects of resource sharing/unsharing on clock frequency, execution time, and area of HLS-generated FPGA-based designs using several realistic benchmarks. This evaluation is done for one 4-input LUT-based FPGA and two 6-input LUT-based FPGAs in order to see the tendency of the effects of resource sharing/unsharing on different FPGA devices. Experimental results brought us the following useful findings; (1) FU unsharing is more effective for clock improvement than register unsharing. Interestingly, unsharing registers only may rather degrade the clock frequency; (2) For newer FPGA devices, unsharing results in more clock improvement; (3) When unsharing FUs, smaller effects of further clock improvement by register unsharing may be expected for newer FPGA devices, especially for larger designs; (4) Not only FU unsharing but also register unsharing can increase the potential of improving the operational-level parallelism (i.e., execution cycle reduction); and (5) For some benchmarks, unsharing FUs and/or registers countintuitively leads to not only clock improvement but also area reduction, which was by up to 57%. These findings are expected to contribute to the further development and advance of the HLS technology.

2. Experimental Setup

In general, MUXs are inserted by FU and register sharing as displayed in **Fig. 1** (b) and Fig. 1 (c), respectively, where circuits synthesized from a behavioral description in Fig. 1 (a) are depicted. These MUXs can be removed by simply unsharing the resources. As shown in **Fig. 2** (b), MUXs may be also inserted even before registers which are not shared by multiple variables but assigned to a multi-defined variable like r in Fig. 2 (a). As described in Fig. 2 (d), such MUXs can be removed by renaming variables so that all assignments are to variables with distinct names (i.e., Static Single Assignment (SSA) transformation [5]: Fig. 2 (c)) and by unsharing registers. Some commercial HLS tools such as eXCite [6] locally perform SSA transformation for dataflow analysis.

Using our HLS framework described in **Fig. 3**, which inte-

1 Nara Institute of Science and Technology, Ikoma, Nara 630–0192, Japan
2 Ritsumeikan University, Kusatsu, Shiga 525–8577, Japan
3 Nagoya University, Nagoya, Aichi 464–8603, Japan
†1 Presently with TOYO Corporation
a) yuko.hara.azumi@ieee.org

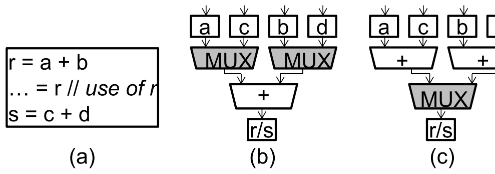

Fig. 1 MUX insertion by resource sharing: (a) A behavioral description, (b) FU sharing, and (c) Register sharing.

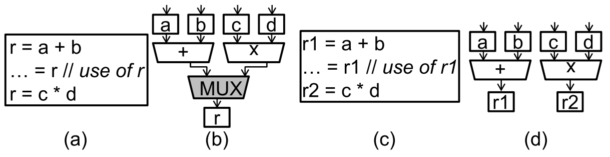

Fig. 2 MUX insertion by resource unsharing and removal of such MUXs: (a) A behavioral description with a multi-defined variable *r*, (b) MUX insertion before unshared register *r*, (c) An SSA-transformed behavioral description, and (d) MUX removal.

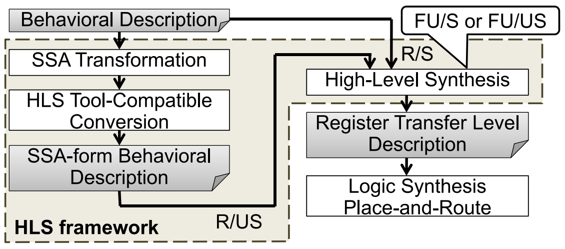
Behavioral Description
FU/S or FU/US
SSA Transformation
R/S
High-Level Synthesis
HLS Tool-Compatible Conversion
Register Transfer Level Description
SSA-form Behavioral Description
Logic Synthesis Place-and-Route
HLS framework
R/US

Fig. 3 Overall synthesis flow.

grates COINS [*1] [7] for SSA transformation and eXCite for HLS, in this paper, the effects of resource sharing/unsharing on clock frequency, execution time (i.e., the number of execution cycles × the clock period), and area of synthesized circuits are quantitatively evaluated by the following four methods: FU sharing & register sharing (FU/S+R/S), FU sharing & register unsharing (FU/S+R/US), FU unsharing & register sharing (FU/US+R/S), and FU unsharing & register unsharing (FU/US+R/US). For R/US, different registers were allocated to each instance of variables and a state. To do so, we have applied SSA transformation to input C programs during the framework described in Fig. 3 [*2]. Eight realistic benchmarks were used: FLOAT_ADD (fadd) and FLOAT_MUL (fmul) in Ref. [8], and adpcm, AES Encryption (aesenc), blowfish, gsm, mips, and sha in Ref. [9]. **Table 1** describes the number of variables for non-SSA and SSA descriptions in columns 2-3 and that of operations in columns 4-7. One 4-input LUT-based FPGA (Virtex-4: xc4vfx100-ff1152-12) and two 6-input LUT-based FPGAs (Virtex-5: xc5vlx110-ff676-3 and Virtex-6: xc6vcx195t-ff784-2) [10] were specified as target devices. Logic synthesis and place-and-route were done by Synplify Pro D-2010.03-SP1 and ISE 13.4, respectively. Besides applying gate-level register retiming, the constraint on clock frequency was automatically set during logic synthesis, so that the maximum clock frequency can be achieved.

[*1] Although eXCite performs SSA transformation for dataflow analysis, we found that COINS more powerfully performs it than eXCite does.
[*2] If variables are assigned in exclusive conditions and used outside of the conditional statements, such variables were re-converted to a multi-defined variable, to which the same register was allocated.

Table 1 Characteristics of benchmark programs.

Benchmarks	Variables		Operations				
	Non-SSA	SSA	Add.	Mul.	Div.	Shft.	Cmp.
adpcm	270	1,279	286	70		4	182
aesenc	191	436	210	13	14		31
blowfish	126	414	278				27
fadd	186	378	31			13	31
fmul	116	198	23	4		4	20
gsm	155	452	212	53		3	93
mips	37	84	9			4	36
sha	56	240	132		3		37

3. Experimental Results

Against the results of FU/S+R/S, **Figs. 4**, **5**, and **6** describe clock improvement, execution time improvement, and area overhead [*3], respectively, by those of the remaining three methods, for each FPGA device. In each figure, results for each benchmark and average on the Virtex-4, 5, and 6 FPGAs are shown in (a), (b), and (c), respetively. In the following subsections, we will discuss observations and findings obtained from the results.

3.1 Clock Frequency

From Fig. 4, the following four features regarding clock frequency are observed:

(1) FU/S+R/US leads to clock improvement for some designs and degradation for the others against the baseline. This method is likely to decrease MUXs before registers but *increase* MUXs before FUs since a lot of incoming paths converge from a large number of registers to the small number of FUs. Because control logic for MUXs inserted before FUs tend to reside on critical paths [11], the increase of such MUXs critically affects the clock frequency, which led to clock degradation by up to 17%.

(2) For all designs, except for fmul [*4], FU/US+R/S improves clock against both the baseline and FU/S+R/US. Generally, paths from the controller to MUXs inserted before FUs go through more gates and encourage longer critical path delay rather than those from data registers to the MUXs. Thus, reduction of the former paths contributes to more clock improvement in this method than in FU/S+R/US.

(3) Although FU/US+R/US achieves the highest clock frequency mostly, FU/US+R/S outperforms in some designs. When using a lot of resources through unsharing, two counter effects would happen at the same time: clock improvement by removing all MUXs from critical paths, and clock degradation due to growing global interconnections (i.e., inter-LUT interconnections) along with an increase in area. In newer FPGA devices, inter-LUT interconnections are much longer against intra-LUT interconnections. Thus, for clock improvement, FU/US+R/S is preferable for large benchmarks on newer FPGA devices, otherwise FU/US+R/US. Actually, in Fig. 4, for larger benchmarks such as adpcm and blowfish on the Virtex-6 FPGA device, FU/US+R/S achieved higher clock frequency than FU/US+R/US.

(4) In most benchmarks, larger clock improvement is achieved by FU/US for newer FPGA devices. This is because in newer FPGA devices, which are based on more CMOS scaling tech-

[*3] Negative values mean area reduction.
[*4] Because fmul is relatively small, the absolute difference was not so large.

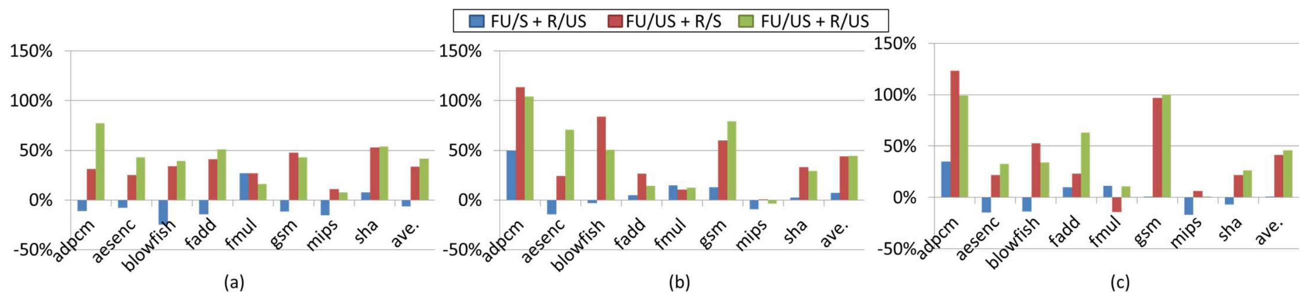

Fig. 4 Clock frequency improvement: (a) Virtex-4, (b) Virtex-5, and (c) Virtex-6.

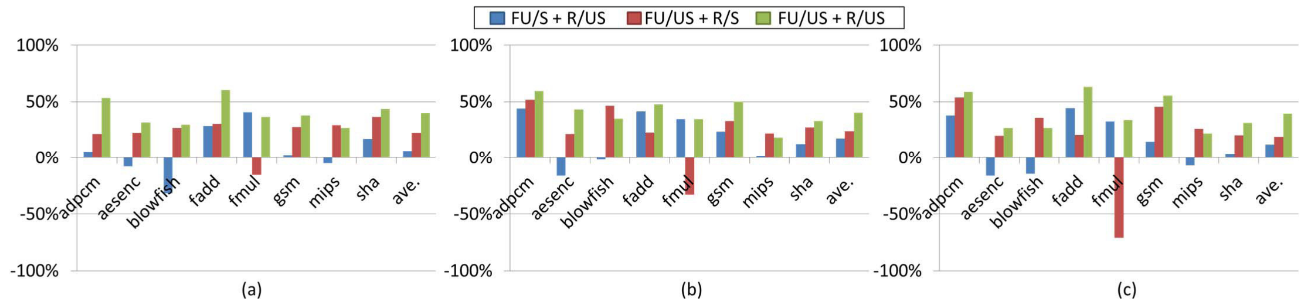

Fig. 5 Execution time improvement: (a) Virtex-4, (b) Virtex-5, and (c) Virtex-6.

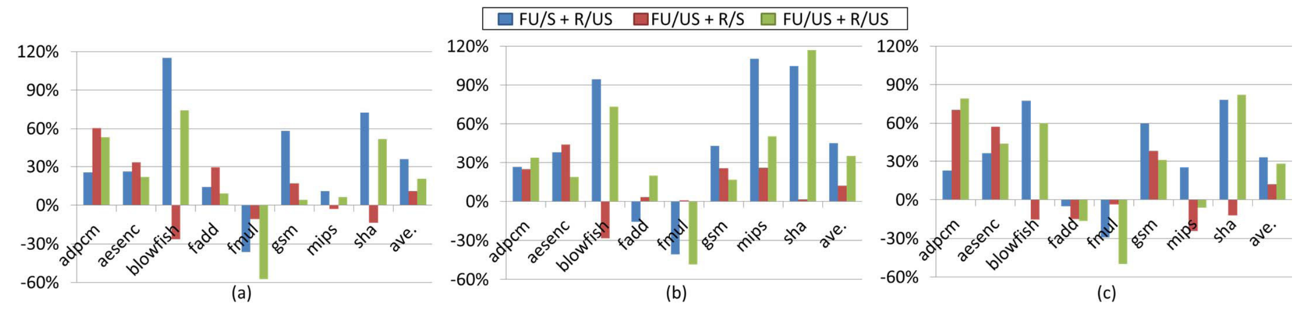

Fig. 6 Area overhead (in terms of Slices): (a) Virtex-4, (b) Virtex-5, and (c) Virtex-6.

nology, global interconnections are more dominant and reducing them through MUX removal is more effective to improve clock frequency. This effect is bigger for larger benchmarks, where the small number of FUs is shared by the large number of operations (i.e., a lot of MUXs are inserted) when FU/S.

From the aforementioned observations, the following conclusions are derived, for achieving higher clock frequency; a higher priority for unsharing should be given to FUs more than registers; FUs should be unshared for newer FPGA devices (e.g., in Virtex-6 than in Virtex-4); and when unsharing FUs, registers should be also unshared in older FPGA device (e.g., in Virtex-4 than in Virtex-6), especially for larger designs.

3.2 Execution Time

As for execution time (i.e., the number of execution cycles × the clock period), the following two findings can be obtained from Fig. 5:

(1) FU/S tends to degrade execution time because limiting the number of available FU instances may increase the number of states, which directly affects an increase in execution time.

(2) In most benchmarks and FPGA devices, R/US improves execution time compared with its counterpart R/S (i.e., FU/S+R/US vs. FU/S+R/S, and FU/US+R/US vs. FU/US+R/S). If R/US achieved higher clock frequency than R/S, its improvement rate of execution time becomes further bigger, whereas if R/US was outperformed in clock frequency, the difference between R/S and R/US in execution time becomes smaller. In these experiments, register unsharing is performed after SSA transformation, which increased the chances of behavioral optimizations to encourage operation-level parallelism, i.e., execution cycle reduction.

From these results, we can say that not only FU unsharing but also register unsharing can increase the potential of improving the execution time.

3.3 Circuit Area

Finally, two findings regarding area (i.e., the number of slices used) can be observed from Fig. 6:

(1) For some benchmarks, circuit area is countintuitively reduced by FU/US and/or R/US, i.e., by up to 57% in fmul on Virtex-4. Because sharing small resources (e.g., adders and registers) by a number of operations/variables leads to insert a lot of MUXs, which occupy large portion of the total area in such benchmarks, unsharing is the effective way for area reduction. Due to the large MUX area in total, even when unsharing increased the circuit area, the rate of area overhead is smaller than that of clock improvement.

(2) The area overhead by FU/US increases when R/S, but reduces when R/US. This may be explained by the structure of

FPGAs. In FPGAs, a slice contains multiple LUTs and flip-flops (i.e., registers). If either of FUs and registers only are unshared and the use of components in slices is unbalanced (i.e., much more LUTs are used than flip-flops and vice versa), the number of slices utilized may increase largely.

The above-mentioned observations are contrary to conventional wisdom that clock frequency and area are in a trade-off relationship - but here, we witnessed that the same strategy of resource sharing/unsharing may be good for both clock frequency and area. This means that efforts which a number of existing researches have made in order to reduce area by resource sharing may not always work well for area reduction. In summary, we can say that exploring well-balanced points of resources sharing/unsharing is important for area reduction.

3.4 Discussion

In these experiments, we well-observed the effects of resource sharing/unsharing on clock frequency, execution time, and circuit area, on three different FPGA devices which are widely used. These results will help not only circuit designers to usefully utilize HLS tools but also researchers/developers to study better strategies for resource sharing/unsharing. We believe that as we learned from the above results, exploring of resources sharing/unsharing (i.e., selective resource sharing), considering behavioral features of applications, is essential for both area and performance improvement.

4. Concluding Remarks

This paper quantitatively evaluated effects of sharing/unsharing FUs and registers on clock frequency, execution time, and area of HLS-generated circuits through eight practically large benchmarks and three widely-used commercial FPGA devices. Through experiments, we revealed some important insights on resource sharing which have not been well-focused. We believe these insights will contribute to further advance of the HLS technology.

Acknowledgments This work is in part supported by KAKENHI 23300019.

References

[1] Gajski, D.D. et al.: *High-Level Synthesis: Introduction to Chip and System Design,* Kluwer Academic Publishers (1992).
[2] Cong, J., Fan, Y. and Xu, J.: Simultaneous Resource Binding and Interconnection Optimization Based on a Distributed Register-File Microarchitecture, *TODAES*, Vol.14, No.3, Article 35 (May 2009).
[3] Cong, J., Liu, B. and Xu, J.: Coordinated Resource Optimization in Behavioral Synthesis, *Proc. DATE*, pp.1267–1272 (2010).
[4] Pilato, C., Ferrandi, F. and Sciuto, D.: A Design Methodology to Implement Memory Accesses in High-Level Synthesis, *Proc. CODES+ISSS*, pp.49–58 (2011).
[5] Aho, A.V. et al.: *Compilers: Principles, Techniques, and Tools,* Addison-Wesley Publishing Company (2006).
[6] Y Exploration, Inc. (online), available from ⟨http://www.yxi.com/⟩ (accessed 2012-11-28).
[7] Abe, S., Hagiya, M. and Nakata, I.: A Retargetable Code Generator for the Generic Intermediate Language in COINS, *IPSJ Journal: Programming*, Vol.46, No.SIG 14 (PRO 27), pp.12–29 (Oct. 2005).
[8] Hauser, J.: SoftFloat (online), available from ⟨http://www.jhauser.us/arithmetic/SoftFloat.html⟩ (accessed 2012-11-28).
[9] Hara, Y. et al.: Proposal and quantitative analysis of the CHStone benchmark program suite for practical C-based high-level synthesis, *JIP*, Vol.17, pp.242–254 (Oct. 2009).
[10] Xilinx (online), available from ⟨http://www.xilinx.com⟩ (accessed 2012-11-28).
[11] Lee, S. and Choi, K.: High-Level Synthesis with Distributed Controller for Fast Timing Closure, *Proc. ICCAD*, pp.193–199 (2011).

Yuko Hara-Azumi received her Ph.D. degree in computer science from Nagoya University in 2010. From 2010 to 2012, she was a JSPS postdoctoral research fellow at Ritsumeikan University. Since 2012, she has been with the Graduate School of Information Science, Nara Institute of Science and Technology, where she is currently an assistant professor. Her research interests include system-level design automation for embedded/dependable systems. She currently serves as organizing and program committees of several premier conferences including ICCAD, ASP-DAC, and so on. She is a member of IEEE, IEICE and IPSJ.

Toshinobu Matsuba received his master degree in computer science from Nagoya University in 2010. From 2010, he is with TOYO Corporation.

Hiroyuki Tomiyama received his Ph.D. degree in computer science from Kyushu University in 1999. From 1999 to 2001, he was a visiting postdoctoral researcher with the Center of Embedded Computer Systems, University of California, Irvine. From 2001 to 2003, he was a researcher at the Institute of Systems & Information Technologies/KYUSHU. In 2003, he joined the Graduate School of Information Science, Nagoya University, as an assistant professor, and became an associate professor in 2004. In 2010, he joined the College of Science and Engineering, Ritsumeikan University as a full professor. His research interests include design automation, architectures and compilers for embedded systems and systems-on-chip. He currently serves as editor-in-chief for IPSJ Transactions on SLDM. He has also served on the organizing and program committees of several premier conferences including ICCAD, DAC, DATE, ASP-DAC, CODES+ISSS, and so on. He is a member of ACM, IEEE and IEICE.

Shinya Honda received his Ph.D. degree in the Department of Electronic and Information Engineering, Toyohashi University of Technology in 2005. From 2004 to 2006, he was a researcher at the Nagoya University Extension Course for Embedded Software Specialists. In 2006, he joined the Center for Embedded Computing Systems, Nagoya University, as an assistant professor, where he is now an associate professor. His research interests include system-level design automation and real-time operating systems. He received the best paper award from IPSJ in 2003. He is a member of ACM, IPSJ, IEICE, and JSSST.

Hiroaki Takada is a professor at the Department of Information Engineering, the Graduate School of Information Science, Nagoya University. He received his Ph.D. degree in information science from the University of Tokyo in 1996. He was a research associate at the University of Tokyo from 1989 to 1997, and was an assistant professor and then an associate professor at Toyohashi University of Technology from 1997 to 2003. His research interests include real-time operating systems, real-time scheduling theory, and embedded system design. He is a member of ACM, IEEE, IPSJ, IEICE, and JSSST.

(Recommended by Associate Editor: *Qiang Zhu*)