Mapping Multiplexers onto Hard Multipliers in FPGAs

Peter Jamieson and Jonathan Rose

The Edward S. Rogers Sr. Department of Electrical and Computer Engineering
University of Toronto
Toronto, Ontario, Canada M5S 3G4
jamieson, jayar@eecg.toronto.edu

***Abstract*—**

Modern FPGAs now contain a selection of "hard" digital structures such as memory blocks and multipliers [2], [12], [9], [1], [8] in addition to the usual "soft" programmable logic typically consisting of Lookup Tables (LUTs) and flip-flops. These hard structures are a major benefit (in area and speed) for those applications that need them, but are completely wasted if an application circuit does not require them. Finding other ways to use these structures will benefit these applications. In this paper, we present a technique to map multiplexers to unused hard multipliers on an FPGA. We have created an RTL synthesis tool flow that implements this technique over a set of benchmarks. While some circuits see no reduction in LUT count at all, others show meaningful improvements ranging from 10% to 70%. On average across the whole set of circuits the technique achieves a 7.3% reduction on the number of LUTs used. In some cases, however, the operating frequency of the circuit is reduced significantly.

I. Introduction

Modern Field-Programmable Gate Arrays (FPGAs) consist of a "soft" programmable fabric and specific "hard" logic structures. Common hard structures that have been added to commercial FPGAs include multipliers, memory blocks, and complex input/output blocks [2], [12], [9], [1], [8]. The use of these structures to implement a design can decrease the soft logic LUT usage, power consumption, and critical path delay [10], [2], [12].

One drawback of hard structures on an FPGA is that if an application circuit does not use the structure, then the area dedicated to it is wasted. This area could be large as it includes not just the logic function, but the expensive programmable routing required to connect it [4]. This potential waste motivates us to look for other ways to make use of these structures. In this paper, we present a method to use unused multipliers on FPGAs to implement multiplexers. This method has been implemented in a Register Transfer Level (RTL) synthesis tool.

Previous work suggests various techniques to utilize hard structures in an FPGA to implement functionality not normally intended for the structures. For example, memories can implement large logic functions by acting as large LUTs [11], [5]. Memory blocks can also implement multiplication operations, and these implementations perform particularly well when the width of the multiplicand is small [7]. The multiplier blocks in an FPGA can also implement a barrel shifter with significant area and speed savings [6]. This work proposes another option: using the multiplier block to implement multiplexers.

This paper is organized as follows. Section II describes how multiplexers can be implemented on an FPGA multiplier and the potential gains in different circumstances. Section III describes a greedy algorithm that decides which multiplexers to map to unused multipliers. Section IV shows how this technique performs on a set of benchmarks that contain and do not contain multipliers in the original design. Finally, we conclude the paper and describe some future work in Section V.

II. Multiplexer to Multiplier Mapping

A. Multiplexer to Multiplier Transformation

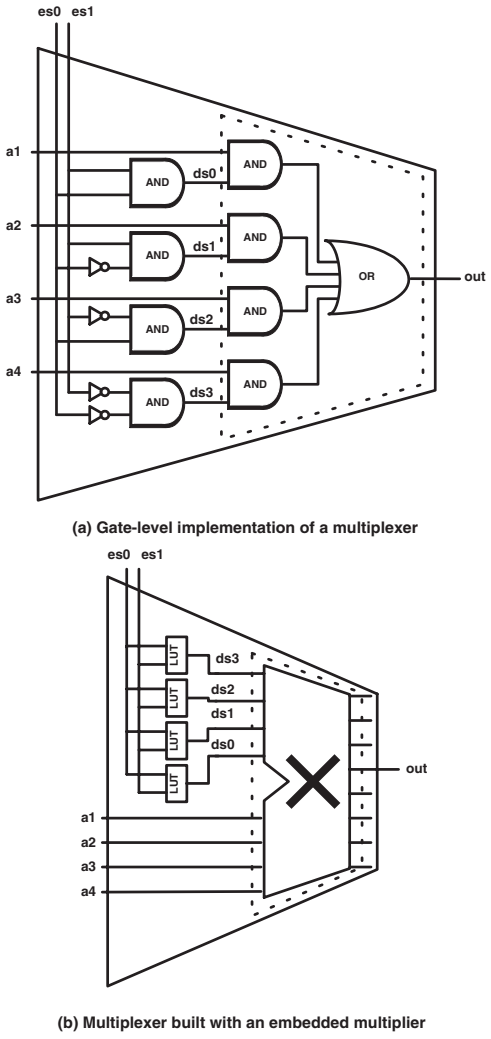

Fig. 1. Illustration of how a 4:1 decoded multiplexer (surrounded by the dashed box) can be mapped to a 4x4 multiplier.

Multiplexers can be mapped to hard multipliers by making use of the observation that multiplying any number by 2^n re-

TABLE I: NUMBER OF DSP BLOCKS AND MULTIPLIERS AVAILABLE ON THE STRATIX FAMILY OF FPGAS

Device	DSP Blocks	Total 9 ×9 Multipliers	Total 18 ×18 Multipliers	Total 36 ×36 Multipliers
EP1S10	6	48	24	6
EP1S20	10	80	40	10
EP1S25	10	80	40	10
EP1S30	12	96	48	12
EP1S40	14	112	56	14
EP1S60	18	144	72	18
EP1S80	22	176	88	22

TABLE II: TABLE (A) SHOWS THE COST OF IMPLEMENTING MULTIPLEXERS ON A STRATIX FPGA. TABLE (B) SHOWS THE COST OF USING A DSP BLOCK ON A STRATIX FPGA FOR DIFFERENT MULTIPLIER WIDTHS.

Multiplexer size	Total LEs	Max Operating Frequency (MHz)
2:1	1	422
3:1	2	422
4:1	3	422
5:1	3	422
6:1	4	422
7:1	5	422
8:1	6	422
9:1	7	422
10:1	7	422
11:1	7	422
12:1	8	422
13:1	9	351
14:1	9	397
15:1	10	403
16:1	11	389
17:1	11	370
18:1	12	359

(a)

Multiplier Size	9x9 DSP Units	Max Operating Frequency (MHz)
2x2	1	307.98
3x3	1	307.98
4x4	1	307.98
5x5	1	307.98
6x6	1	307.98
7x7	1	307.98
8x8	1	307.98
9x9	1	307.98
10x10	2	248.14
11x11	2	248.14
12x12	2	248.14
13x13	2	248.14
14x14	2	248.14
15x15	2	248.14
16x16	2	248.14
17x17	2	248.14
18x18	2	248.14

(b)

sults in a left shift by n places. Using this property we can attach the inputs of a multiplexer to one port of the multiplier, and we can attach the decoded multiplexer select signals to the other port of the multiplier; the multiplexer output is then one of the output pins on the multiplier (output n for a $n:1$ multiplexer). Figure 1 (a) shows an example of the mapping. In Figure 1 (a), we show a four-to-one multiplexer with both encoded and decoded signals labelled as es_x and ds_y respectively. Figure 1 (b) shows how our example four-to-one multiplexer can be implemented using one four by four unsigned multiplier. Note that the multiplier can implement the decoded multiplexer portion of the circuit (encapsulated by the dashed box in Figure 1 (a) and (b)), which is controlled by the ds_y select signals. In general, a $n:1$ decoded multiplexer can be implemented on a $n*n$ multiplier. An encoded multiplexer needs additional soft logic to implement the decoding of the select signals, and the area savings will not be as big, but in many cases these select signals are shared across many multiplexers and the decoding logic can be amortized.

B. Mapping Multiplexer to Multiplier - Area

The area savings from mapping multiplexers to multipliers depends on the costs of implementing multiplexers in the soft fabric of an FPGA (which typically consists of LUTs) and the hard multipliers. To determine the mapping costs, we choose a specific FPGA architecture with hard multipliers; in this paper Altera's Stratix FPGA family [2] is used to test our approach.

Multipliers, as well as other multiplier functions like multiply accumulate, are available for the Stratix FPGAs in the DSP block [2]. Table I shows the number of the DSP blocks available in each member in the Stratix family where each block can be configured to implement either eight 9x9 multipliers, four 18x18 multipliers or one 36x36 multiplier as well as other operations. The total number of multipliers for each multiplier width is summarized in columns 3, 4 and 5 of Table I.

Table II (a) gives the area and speed of decoded multiplexers implemented in the soft logic fabric of a Stratix FPGA. Column two contains the number of Logic Elements (LEs) required to implement each multiplexer (an LE is Altera's terminology for a LUT with some additional circuitry). Column three shows the speed for each decoded multiplexer implemented on a Stratix FPGA (Note that these are *decoded* multiplexers and do not include the area or speed penalty for implementing the decoding logic of select signals).

The data in Table II (a) can be used to estimate the area savings for a set of multiplexers on a particular FPGA architecture when mapping multiplexers to hard multipliers. For example, the maximum area saving for a design that has an eight-to-one decoded multiplexer mapped to a multiplier is 6 LEs. This savings does not include the cost of the logic for decoding the select signals.

C. Mapping Multiplexer to Multiplier - Speed

To estimate speed of mapping a multiplexer to a multiplier, we compare the speed of both implementations. Table II (b) column three gives the maximum operating frequency of different sized hard multipliers on a Stratix FPGA. Similarly, column three of Table II (a) shows the speed of a decoded multiplexers implemented in soft logic. Using these two pieces of information, a nine-to-one multiplexer will operate approximately 114 MHz slower when mapped to a hard multiplier. This speed loss may or may not change the speed of an entire design. Section IV shows the overall impact on speed of our benchmarks.

D. Reducing Multiplexer Decoding Area Costs

The multiplexer to multiplier mapping technique potentially saves area at a speed cost. However, as Figure 1 illustrates, a multiplexer's encoded select signals need to be decoded before being attached to the multiplier, reducing the potential area benefits of the technique. Two factors mitigate this effect: one, the decoding logic may be amortized for multiplexers that share the same select signals, and two, digital designs contain decoded multiplexers.

The first factor reducing decoding logic is due to multiplexing buses; these buses share multiplexer select signals. Figure 2 shows a case structure in Verilog and a possible logic implementation of this case structure. This Figure shows how decoding logic can be amortized since both the select signals $a[0]$ and $b[0]$ only need to be decoded once to control the outputs $out[0]$ and $out[1]$.

The second factor reducing decoding logic is due to designs containing decoded multiplexers and associated decoding logic. If decoding logic consists of more than one signal then the multiplexer is best implemented as a decoded multiplexer. Decoded multiplexers appear in HDL designs in control structures that

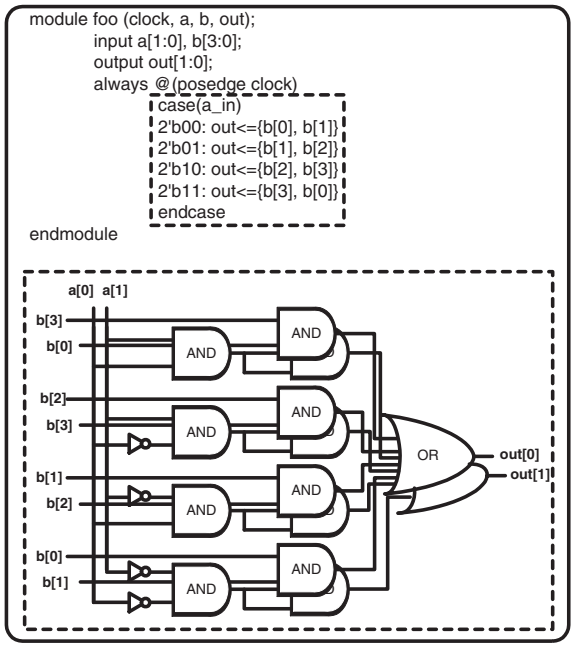

Fig. 2. This is a sample logic implementation and sharing of decoder logic of Verilog case structure.

```
algorithm greedy_mux_selection

input list_decoded_muxes
input list_of_muxes_area_saving
input paths_w_multipliers
output list_of_muxes_mapped_or_not

sort(list_of_muxes_area_saving )
for each in list_of_muxes_to_map
   for i in list_size(list_of_decoded_muxes)
      if ((width(list_of_muxes_area_saving)==width(list_of_decoded_muxes [i])) &&
        (fpga_multiplier_available() == TRUE) &&
        (in_path(paths_w_multipliers, list_decoded_muxes [i]) == FALSE) &&
        (common_select_signals(list_of_decoded_muxes [i]) is large))
           list_of_muxes_mapped_or_not [i] = TO_DSP
      else
           list_of_muxes_mapped_or_not [i] = TO_SOFT
```

Fig. 3. Algorithm for the selection of multiplexers to map to multipliers.

compare signals. For example, $a == 0$ and $a == b$ will result in very different logic implementations. The constant in $a == 0$ is encoded, which allows these structures to be mapped into encoded multiplexers. On the other hand, $a == b$ is mapped to decoding logic which feeds a decoded multiplexer. A decoded multiplexer is mapped to soft logic or a hard multiplier, and in both cases the decoding logic area cost is the same.

III. Algorithm mapping Multiplexers Multipliers

The fact that decoded multiplexers exist in designs and multiplexer decoding logic can be amortized over multiple multiplexers suggests that we can map some multiplexers to multipliers during the synthesis stage of an entire FPGA Computer Aided Design (CAD) flow to improve area usage. To study this technique, we implement a greedy algorithm that finds multiplexers in a design and maps decoded multiplexers to hard multipliers when this mapping results in the greatest area saving.

To determine the greatest area saving, we compare the costs of implementing a multiplexer in the soft logic of an FPGA and determine what size of multiplier can implement this multiplexer. For example, on the Stratix architecture, multiplier mapping a nine-to-one multiplexer will result in a saving of seven LEs per 9x9 multiplier. This is a greater LE saving compared to mapping an eighteen-to-one multiplexer to an 18x18 multiplier, which saves six LEs per 9x9 multiplier (one 18x18 multiplier can be replaced by 2 9x9 multipliers on a Stratix FPGA).

A greedy selection algorithm picks the best multiplexers to map; this algorithm is shown in Figure 3.

The algorithm also performs a speed optimization that is applied when a design contains multipliers. Our algorithm will never map multiplexers to multipliers if the multiplexer is on a combinational path containing a multiplier. Since the mapping technique results in slower paths and paths containing multipliers are in general slow, this rule reduces the chance that the mapping technique will increase the critical path of a design.

The algorithm also amortizes decoding logic costs by grouping multiplexers that have greater than two common multiplexer selection signals, or selecting "case" and "if" structures that compare multiple signals. This rule picks multiplexers that will incur the smallest area cost for decoding the multiplexer signals and improves our area savings.

We have implemented this greedy algorithm in a CAD flow that maps to Stratix FPGAs. First, our high-level RTL synthesis tool, developed by us, converts Verilog designs into structural Verilog net-lists consisting of gate primitives and Library Parametrized Modules (LPMs) targeted for Stratix FPGAs. This net-list is passed into Altera's Quartus CAD flow that maps the net-list to an FPGA and generates area and timing results. Using the Quartus CAD flow is one way to map our designs to industrial FPGAs and get real speed and area results [3].

IV. Results

To study our algorithm we used our RTL synthesis tool attached to the Quartus CAD flow to generate both baseline and "multiplexer mapped to multiplier" results for a set of benchmarks.

Table III shows the experimental results for speed and area of our mapping technique for each benchmark averaged over 5 random seeds. Columns 2, 3 and 4 contain the number of LEs, number of DSP units, and the maximum operating frequency respectively for our baseline results of the benchmarks mapped to Stratix FPGAs without using the multiplexer to multiplier mapping. Columns 5, 6, and 7, contain similar area and speed results as above except these values are for circuits mapped using the presented technique. Finally, column 8 and 9 show the the ratios for baseline area over optimized area and baseline speed over optimized speed respectively. Both of these ratios are geometrically averaged over all benchmarks; we also grouped circuits with and without multipliers together and geometrically averaged the ratios in these groups.

On average, the number of LEs is reduced by 7.3%. The benchmarks, however, suffer from speed losses ranging from 1 to 206 MHz. If we break these benchmarks into two categories consisting of benchmarks with multipliers and benchmarks without multipliers, then our results are different. In Table III, we show the two separate groups by shading in grey the benchmarks that contain multipliers.

For benchmarks without multipliers in the original design,

TABLE III: AREA/SPEED RESULTS MAPPED BENCHMARKS ON STRATIX FPGAS WITH AND WITHOUT THE MULTIPLEXER TO MULTIPLIER MAPPING.

	Baseline			With Multiplexer mapping				
Benchmarks	Number of LEs	Number of 9x9 DSP-units	Speed (MHz)	Number of LEs	Number of 9x9 DSP-units	Speed (MHz)	Area Ratio	Speed Ratio
fft_258_6	3190	32	146.16	3145	48	148.23	1.014	0.99
iir1	501	7	82.53	501	7	83.01	1.000	0.99
iir2	338	10	109.158	300	15	108.23	1.127	1.01
fir_3_8_8	84	4	251.928	84	4	252	1.000	1.00
fir_24_16_16	1591	48	75.042	1591	48	75.02	1.000	1.00
fir_scu_rtl	548	17	109.54	317	48	97.17	1.729	1.13
diffeq_f_systemC	271	40	41.11	251	48	41.05	1.080	1.00
diffeq_paj_convert	369	40	29.858	324	48	29.77	1.139	1.00
sv_chip1	17145	96	122.31	17145	96	121.21	1.000	1.01
rt_raygentop	2679	27	127.24	2536	44	131.03	1.056	0.97
rt_raygentop_no_mem	2815	27	136.916	2671	44	132.92	1.054	1.03
oc45_cpu	3101	2	61.63	2905	48	65.21	1.067	0.95
reed_sol_decoder1	1183	13	82.69	1090	48	81.65	1.085	1.01
reed_sol_decoder2	1957	9	53.58	1864	48	53.93	1.050	0.99
md	14867	112	34.942	14867	112	35	1.000	1.00
			Summary for circuit with multipliers:				1.083	1.00
cordic_8_8	838	0	256.436	808	48	103	1.037	2.49
cordic_18_18	4104	0	222.72	4058	48	178	1.011	1.25
MAC1	2812	0	98.532	2669	48	70.17	1.054	1.40
MAC2	9720	0	74.876	9538	48	66.74	1.019	1.12
des_area	1305	0	194.372	1074	48	205	1.215	0.95
des_perf	3838	0	199.44	3638	48	81.49	1.055	2.45
sv_chip0	12729	0	120.16	12650	48	92.13	1.006	1.30
sv_chip0_no_mem	7122	0	146.202	7025	48	100.02	1.014	1.46
sv_chip3_no_mem	134	0	328.94	109	5	122.55	1.229	2.68
rt_frambuf_top	784	0	127.75	711	10	80.81	1.103	1.58
rt_frambuf_top_no_mem	909	0	139.3	901	1	78.96	1.009	1.76
rt_boundtop	3895	0	83.35	3810	27	63.5	1.022	1.31
rt_boundtop_no_mem	4026	0	86.69	3807	27	65.31	1.058	1.33
			Summary for circuit without multipliers:				1.062	1.54
			Summary for all circuits:				1.073	1.23

we use 6.2% less LEs, and the maximum operating frequency of these benchmarks decreases by an average of 54%.

Benchmarks that do contain multipliers save 8.3% on LEs and in general show an average speed loss of 0.5%. This result show the value of the optimization in saving LEs without any speed decrease.

In one multiplier case, "fir_scu_rtl" we do see a significant loss in speed (-12.87 MHz), but this loss occurs because the original multipliers in "fir_scu_rtl" are small. The multipliers used to implement multiplexers in this design become the critical path of the circuit.

V. CONCLUSIONS AND FUTURE WORK

In this paper, we have presented a simple technique to map multiplexers to multipliers and a high-level synthesis algorithm to select which multiplexers to perform this mapping on. Our experimental results show that this mapping technique can save a significant number LEs at a speed loss of 0.5% if critical paths can be determined up front.

Future work includes finding a better way to determine which multiplexers to map to multipliers so that we get the greatest area savings at no speed cost. To do this, we must make the mapping decision at a CAD stage with better timing information, by either having a good way of estimating timing during RTL synthesis, or by using an iterative CAD flow that obtains accurate timing information from a fully placed and routed design of the last CAD iteration.

REFERENCES

[1] Actel. *ProASIC 500K Family*.

[2] Altera. *Stratix Device Handbook*, Jul 2003.

[3] Altera. *Quatus II University Interface Program (QUIP) Tutorial*, February 2004.

[4] S. D. Brown, R. J. Francis, J. Rose, and Z. G. Vranesic. *Field-Programmable Gate Arrays*. Kluwer Academic Publishers, Boston, 1992.

[5] J. Cong and S. Xu. Performace-driven technology mapping for heterogeneous fpgas. *IEEE Transaction on Computer Aided Design of Integrated Circuits and Systems*, 19(11):1268–1280, Nov. 2000.

[6] P. Gigliotti. Implementing barrel shifters using multipliers. *XAPP - Application Note: Virtex-II Family*, pages 1–4, Aug. 2004.

[7] S. Knapp. Constant-coeficient multipliers save fpga space, time. *Personal Engineering*, pages 45–48, 1999.

[8] Lattice. *ispXPGA Family*, Jan 2004.

[9] QuickLogic. *Eclipse Family Data Sheet*, 2003.

[10] S. Wilton, J. Rose, and Z. Vranesic. The memory/logic interface in fpga's with large embedded memory arrays. *IEEE Transactions on Very-Large Scale Integration Systems*, 7(1), March 1999.

[11] S. J. Wilton. Implementing logic in fpga memory arrays: Heterogeneous memory architectures. In *IEEE Custom Integrated Circuits Conference*, pages 142–149, May 2002.

[12] Xilinx. *Virtex-II Pro Platform FPGAs: Functional Description*, Oct 2003.