RASP: A General Logic Synthesis System for SRAM-based FPGAs

Jason Cong and John Peck
Department of Computer Science
University of California, Los Angeles, CA 90024

Yuzheng Ding
AT&T Bell Laboratories, Murray Hill, NJ 07974

Abstract

In this paper, we present a general synthesis system for SRAM-based FPGAs named RASP. RASP consists of a core with a set of synthesis and optimization algorithms for technology independent logic synthesis and technology mapping for generating generic look-up tables (LUTs), together with a set of architecture-specific technology mapping routines to map the generic LUT network to programmable logic blocks (PLBs) for various SRAM-based FPGA architectures. Via a set of design representation converter routines, these architecture-independent and dependent synthesis algorithms are easily linked, and the entire system is seamlessly integrated into the design flow of commercial FPGA design systems. As a result, RASP can produce highly optimized designs for various SRAM-based FPGA architectures, and can be quickly adapted for new SRAM-based FPGA architectures. We compare RASP performance with that of several commercial synthesis systems on the MCNC logic synthesis benchmarks and a video compressor/decompressor. For almost all cases, RASP produces mapping solutions with significantly smaller critical path delay after place and route than current commercial synthesis systems.

1. Introduction

The RASP (RApid System Prototyping) system is a general synthesis and mapping system for SRAM-based FPGAs developed recently at UCLA. The development of RASP was motivated by the following two observations:

(1) The wide-spread use of the FPGA technology for rapid ASIC designs and rapid system prototyping has led to an exponential growth of the FPGA market, in which SRAM-based FPGAs have the major share. There are several SRAM-based FPGA vendors, who introduce multiple new FPGA architectures each year. The existing SRAM-based FPGA products include the Xilinx XC3000, XC4000, and XC5000 families [Xi94a], the Altera FLEX8000 and FLEX10000 families [Al95], AT&T 3000, ORCA1C, and ORCA2C families [AT&T95]. Many new architectures are always under development. The programmable logic blocks (PLBs) in various FPGA architectures are often different, in terms of the number of LUTs, their sizes, their connection patterns, and other glue logic in each type of PLB. As a result, it becomes an increasingly important yet challenging task for the EDA industry to develop high-quality synthesis and mapping tools for each FPGA architecture. There is a strong need for a general synthesis system for SRAM-based FPGAs which can be easily and effectively adapted to various FPGA architectures.

(2) Much work has been done on architecture independent LUT-based synthesis and mapping for particular optimization objectives, such as area minimization [FrRC90, MuNS90, Ka91a, Wo91a, LaPV93, SaTh92], delay minimization [FrRV91b, MuSB91b, ChCD92, CoDi94a], routability [ScKC92], or a combination of several objectives [CoDi94b, CoHw95a]. Although many of these algorithms have reported very encouraging results in terms of optimizing the LUT networks, most of them have not been tested on various real FPGA architectures. It would be interesting to know how effective these algorithms are when applied to each specific FPGA architecture. Therefore, there is a strong need for a general framework to evaluate each LUT-based synthesis and mapping algorithm on different FPGA architectures.

In order to fulfill these two needs, RASP was developed as a general synthesis system for SRAM-based FPGAs. It consists of a **core** with a set of synthesis and optimization algorithms for technology independent logic synthesis and technology mapping for generic look-up table (LUT) network generation, together with a set of architecture-specific technology mapping routines to map the generic LUT network to PLBs in various SRAM-based FPGA architectures. Via a set of design representation converter routines, these architecture-independent and dependent synthesis algorithms are easily linked, and the entire system is seamlessly integrated into the design flow of commercial FPGA design systems. As a result, RASP can produce highly optimized designs for various SRAM-based FPGA architectures, and can be quickly adapted for new SRAM-based FPGA architectures. Also, RASP provides a flexible and complete development framework in which various architecture independent LUT synthesis algorithms may be coupled with different architecture-specific algorithms which map LUTs into PLBs. New LUT synthesis and mapping algorithms can be easily integrated into RASP and put to trial against different kinds of SRAM-based FPGAs. RASP also provides the flexibility for

experimentation at each sub-step of the synthesis and mapping process, which is useful in tuning the optimization for a particular set of objectives. We have successfully incorporated several FPGA synthesis and technology mapping algorithms, including FlowMap [CoDi94a], FlowMap-R [CoDi94b], FlowSYN [CoDi93b], and CutMap [CoHw95a], into RASP as the LUT mapping engines; we have also developed heuristics for PLB mapping, including ones for the Xilinx XC3000 and XC4000 FPGA families. RASP has been successfully integrated with the design flow of commercial synthesis systems, such as XACT^{TM} [Xi94a] and MaxPlusII^{TM} [Al95] .

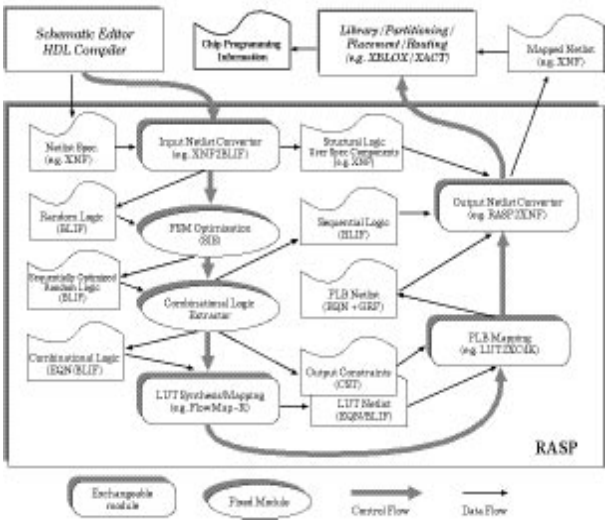

Fig. 1: RASP Framework Overview

2. Overall System Architecture

The overall framework of RASP is shown in Fig. 1. We will illustrate it using a particular application, namely synthesis and mapping for Xilinx XC4000 Family FPGAs, in the following sections. The entire process is automated through use of a shell script which may be modified to accommodate changes in process flow for experimentation. In the framework, exchangeable modules can be assigned by the user via system parameters whereas the fixed modules do not change. The user may also modify parameters for both exchangeable and fixed modules to control their performance.

A vendor provided netlist translator or HDL synthesizer is first used to produce a design file representing the netlist (RTL or gate netlist), in a supported format, such as Xilinx netlist format (XNF). For each supported format, an *input netlist converter* first separates the "non-synthesizable logic", which will be defined shortly, from the "random logic" that is available for synthesis. The former is preserved internally using the original netlist format, which is later re-incorporated into the final synthesis result. The latter is translated into the Berkeley Logic Interchange Format (BLIF); which is used as one of the internal representation formats in RASP. The connections to each non-synthesizable component are replaced by primary I/Os in the BLIF representation, and redundant logic is trimmed during the translation. In the case of XNF input, the module ***xnf2blif*** is used to carry out the conversion.

"Non-synthesizable logic" includes target architecture specific logic such as I/O pads and clock buffers, structural logic such as datapath components or any special modules supported by vendor-supplied library mapping tools (e.g. XBLOX^{TM}), as well as designer mapped logic (i.e. "hardwired PLBs"). Structural logic is specified as hard or soft macros in the design, and has a highly optimized implementation from the vendor library, therefore making it unlikely that further optimization can be accomplished by the synthesis and mapping modules. Designer mapped logic may be used to satisfy special design requirements (such as timing) that cannot be guaranteed by the automatic design tool. Therefore, these are regarded as non-synthesizable. However, our input netlist converter does provide an option that allows the user to merge structural and user mapped logic into the random logic for re-synthesis and re-mapping if the user so desires.

Then, the BLIF network representing the random logic portion of the design is processed for sequential logic optimization. RASP uses the SIS logic synthesis system [SeSL92] to perform such optimization, which gives the user a rich variety of algorithm choices for state extraction, state minimization and state assignment. The optimized sequential network is then passed to the *combinational logic extractor*, which separates the sequential components from the combinational components in the network, preserving the sequential portion in BLIF representation for later integration into the final synthesis/mapping solution. The extractor represents the combinational portion as a Boolean network in BLIF or EQN (Boolean equation) format, in which connections to the sequential components are replaced by primary I/Os. It is noteworthy that such an extraction may lose important information about the feasibility of a synthesis mapping solution, since each PLB contains both combinational and sequential elements. If two or more combinational logic elements are to be mapped into one PLB it is required that their associated sequential logic elements, if any, be *compatible*. For example, in almost all types of SRAM-based FPGAs, the flip-flops in a PLB are required to share the same clock signal. To retain such constraints without complicating the mapping procedure, the combinational logic extractor also produces another file in internal constraint file format (CST), which represents the compatible classes of the sequential elements which have corresponding primary I/Os in the Boolean network according to the given specification of compatibility for the target FPGA type. Redundant logic trimming is also performed during the extraction.

Next, the Boolean network is passed to the LUT mapping routine, which can be any implemented algorithm that recognizes BLIF or EQN input and produces output in these formats. For example, the ***FlowMap-R*** algorithm [CoDi94b] can be used. In the next section we shall introduce several algorithms which RASP supports. Optionally, technology independent logic synthesis can be performed prior to the mapping by using SIS commands. The output of the mapping algorithm is a network of LUTs which is passed to the PLB mapping routine.

The PLB mapping module is selected according to the target FPGA type. For example, for Xilinx XC4000 FPGAs, we have developed the module ***lut2xc4k***. The PLB mapping module takes the LUT network and the CST file produced by the combinational logic extractor, and maps the compatible LUTs into PLBs of the target FPGA. More details on this procedure will be presented in the next section. The output uses annotated EQN format with necessary "group" (PLB) information. Note that depending on the target FPGA type and the parameters used during LUT synthesis and mapping, the PLB mapping module may decompose and/or merge some LUTs for better PLB mapping.

Finally, an output netlist converter combines the PLB netlist with preserved logic, including the non-synthesizable logic and the sequential logic, and generates a complete design representation in the vendor specific format, such as XNF, that can be used as input to a commercial design system such as XBLOX^{TM}/XACT^{TM}, which map the structural logic and perform the target architecture layout and routing. For example, the ***rasp2xnf*** converter is used to produce the XNF netlist.

3. Synthesis and Mapping Algorithms in RASP

The architecture independent LUT synthesis/ mapping and PLB mapping modules are the most important parts of the RASP core which will be discussed in this section.

3.1 Architecture Independent LUT Synthesis and Mapping

For a particular FPGA based design, the design engineer may have a particular optimization interest in mind whether it be minimum delay subject to an area constraint, hard delay constraint with area as a secondary concern, or hard area constraint with delay as a secondary concern. RASP incorporates a wide range of LUT-based FPGA synthesis and technology mapping algorithms with different objectives. Currently it consists of the following algorithms.

FlowMap: FlowMap [CoDi94a] is a LUT-based FPGA technology mapper that produces depth-optimal mapping solutions for general Boolean networks. The basic idea of the FlowMap algorithm is to find a depth-optimal mapping for each node in the network, according to the topological order starting from the PI nodes. The depth-optimal mapping of each node is achieved by computing a *minimum height K-feasible cut* (*X, X'*) in the fanin cone of the node. It was shown that such a cut can be computed in polynomial time.

FlowMap-R: The FlowMap-R [CoDi94b] algorithm uses *depth relaxation* and re-mapping to generate a set of mapping solutions with depth/area trade-offs. It first uses FlowMap [CoDi94a] algorithm to compute a depth-optimal mapping solution as a starting point for depth relaxation. Then, for each given depth bound of the mapping solution, the FlowMap-R algorithm first applies a sequence of depth relaxation heuristics to produce an intermediate network, and then carries out re-mapping for area minimization on the intermediate network using an area-optimal *duplication-free* mapping and two area-reduction algorithms that exploit beneficial logic duplication. To generate a set of mapping solutions, the depth bound is gradually increased and mapping is carried out for each depth bound. This algorithm is particularly suitable for RASP since it provides flexibility for meeting optimization objectives.

FlowSYN: The FlowSYN [CoDi93b] algorithm inherits the combinatorial optimization techniques from FlowMap. In addition, it uses global structural information obtained during combinatorial optimization to selectively re-synthesize parts of the given network using Boolean logic operations for further depth and area optimization. The FlowSYN algorithm follows the FlowMap control flow by computing a mapping solution at each node in the topological ordering. Re-synthesis using BDD-based functional decomposition is applied where combinatorial optimization fails to produce a good result.

CutMap: The CutMap [CoHw95a] algorithm also improves FlowMap by performing area minimization under depth constraint, but uses an approach different than that used in FlowMap-R: it first computes the minimum depth to which each node can be mapped, then implements the LUTs not according to the minimum-height K-feasible cut, but according to the *minimum-cost* K-feasible cut where a cost function is used to encourage the sharing of input signals among LUTs and guarantees that the critical path depth will not increase. As a result, it preserves the depth optimality of FlowMap while substantially reducing the number of LUTs.

3.2 Architecture Dependent PLB Generation from LUT Networks

After the architecture independent synthesis and mapping algorithm produces an LUT netlist, RASP converts it into a netlist of PLBs for the target FPGA using the appropriate converter from the RASP architecture specific PLB generation toolkit. Each of the converters exploits the PLB architecture of a specific FPGA type. Mapping LUTs to PLBs involves choosing one or more LUTs to be packed into each PLB taking into account the PLB architectural constraints, so that the number of PLBs used is minimized by compact packing, the delay of the PLB network is minimized by effective use of multi-level LUT interconnections inside a PLB, and the routability is maintained by careful selection of closely connected LUTs for each PLB. Due to the availability of auxiliary logic and internal connections, a group of LUTs may be mapped into a group of LUTs of different sizes and connections in a PLB (for example, a 5-LUT can be implemented by the two 4-LUTs and the auxiliary MUX in a PLB of XC3000 FPGA).

Clearly, this is a complex optimization problem, and we have developed heuristics for various FPGA architectures and optimization objectives. Each of these heuristics is based on a sequence of *maximum weighted matching* operations on a *compatibility graph* which yields proper grouping of LUTs into PLBs without violating the constraints.

3.2.1 Forming the Compatibility Graph

At each step, the vertices of the compatibility graph represent the partial PLBs (initially the LUTs) that will be

considered for grouping at this step. An edge is formed between two vertices if the two corresponding partial PLBs can be grouped into one. When generating an edge in the compatibility graph, we check the constraints on the input size and pattern, output size and pattern, and if there are registered outputs, the number of such outputs and their compatibility, in terms of the shared control signals of the registers, using the information in the constraint file.

3.2.2 Weighting the Compatibility Graph

Once the compatibility graph is constructed, we assign weights on the edges to guide the matching algorithm to select the best merging of partial PLBs. Different weights are assigned for different optimization objectives. For delay minimization, a larger weight is given to an edge corresponding to the grouping of two LUTs that may reduce the length of a critical path in the PLB network. For routability, a larger weight is given to an edge that corresponds to the grouping of two "close" LUTs so that we do not create complex interconnection patterns in the final mapping solution. For area minimization, we simply want a matching of maximum cardinality, but we can still use weights to break ties and obtain a maximum cardinality matching with good delay and/or routability.

The "closeness" of two LUTs can be measured in various ways. In our current implementation, we use the overlap of their fanin subnetworks as the measurement. More precisely, for an edge (v, w) we use $(|N_v \cap N_w|) / (|N_v| + |N_w|)$ as the weight. We have found that this simple measurement is usually sufficient to produce routable solutions. More accurate measurements can also be derived, e.g. by also including the fanout subnetworks into consideration. The actual weight $W(e)$ of each edge e consists of two parts, a "nominal weight" $W_N(e)$ that is the same for all edges, and a "preference weight" $W_P(e)$ that varies among edges, i.e. $W(e) = W_N(e) + W_P(e)$. The preference weight is used for optimization objectives other than area minimization. For example, for routability optimization, the closeness measurement is used as the preference weight. The nominal weight is used to balance the optimization for the objective emphasized by the preference weight, and the objective of area minimization. As the value of nominal weight increases, the emphasis on the cardinality of the matching increases, and the maximum weighted matching solution yields more area reduction. For example, for routability optimization, we use the following weight function: $W(v, w) = W_P(v, w) + W_N(v, w) = [(|N_v \cap N_w|) / (|N_v| + |N_w|)] + \alpha |N|$, where N is the network, and $0 \le \alpha \le 1$ is a parameter to adjust the trade-off between area and routability. When $\alpha = 1$, the maximum weighted matching is guaranteed to have maximum cardinality. If the resulting solution is not routable, α is reduced.

3.2.3 Example of LUT to PLB Mapping

Since the heuristics for PLB mapping differ for different FPGA architectures, we illustrate our approach using two examples, the XC3000 and XC4000 families. Following the terms used by the vendor, the LUTs are called *function generators*, and the PLB is called the *configurable logic block* (CLB).

Mapping for Xilinx XC3000 FPGAs

The Xilinx XC3000 family CLB architecture contains one combinatorial function unit which can either be used as two 4-variable function generators F and G with totally no more than 5 inputs, or be used as one 5-variable function generator F. Each functional generator can have one (registered or non-registered) output, and all register control signals are shared.

The architecture independent synthesis and mapping algorithm in RASP produces a network of LUTs with input size no more than 5. Each 5-LUT is then implemented by one CLB using the F mode of CLB operation. Then, smaller LUTs are matched according to the input constraints, as well as the output constraints (if there are registered outputs), using the weighted matching approach discussed earlier[1]. Unmatched LUTs are then treated as 5-LUTs and implemented by separate CLBs.

Although the constraint on total number of inputs usually requires input sharing between the matched LUTs, unrelated small LUTs can still be packed together if preference weight is ignored. Therefore, we use the number of shared inputs (a simple measurement of closeness) as the preference weight for routability control.

Mapping for Xilinx XC4000 FPGAs

The Xilinx XC4000 family CLB architecture contains three function generators. Two of the function generators named *F* and *G* each have four input variables. The third function generator *H* has three inputs, two of which are connected to the outputs of function generators *F* and *G* and the third is unrestricted. Moreover, if *H* is used, only one of *F* and *G* can have its output (registered or otherwise) accessible by outside. We have developed a multi-step matching heuristic for this architecture as follows.

Step 1: Starting with a network of LUTs with input size no more than 5, we first decompose all 5-LUTs using Shannon decomposition $f(x_1,...,x_5) = x_1 * f_1(x_2,...,x_5) + \bar{x}_1 * f_0(x_2,...,x_5)$. Note that if f is unate on x_1, then $f_1 = 0$ or $f_0 = 0$, and the decomposition can be implemented by one 4-LUT and one 2-LUT; otherwise, it can be implemented by two 4-LUTs and one 3-LUT. In the latter case, it will occupy a CLB by itself; while in the former case, it will occupy the F and H function generators and has a vacant G slot that may be filled by another 4-LUT, which is preferred. Therefore, we look through the inputs to find a unate one for the decomposition before accepting the decomposition of three LUTs. After such decompositions, each 5-LUT is associated with a fully or partially filled CLB. Then, we associate each of the other LUTs (of input size no more than 4) with a CLB and let the LUT occupy the G function generator.

Step 2: We construct a compatibility graph over the CLBs containing only one LUT, where an edge (v, w) is drawn if CLB v has a 3-LUT LUT_v, and CLB w has an LUT LUT_w whose only output is LUT_v. Clearly, v and w can be merged by putting LUT_v and LUT_w into the *F* and *H*

[1] A similar approach for XC3000 CLB generation has been used by, for example, [MuSB91a].

function generators of *v*. We compute a maximum weighted matching with the depth of LUT_w as preference weight and large enough nominal weight to guarantee a maximum cardinality. The use of depth as preferred weight is to encourage the packing of LUTs of large depth first. The matched pairs are merged. A matching candidate for this step is illustrated in Fig 2(a).

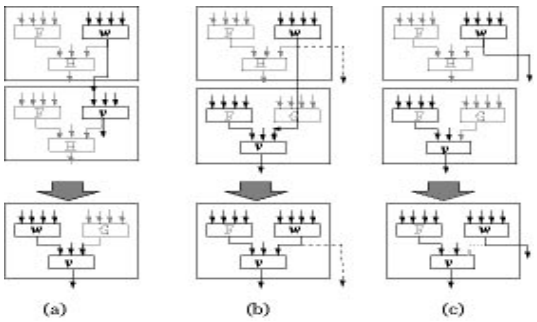

Fig 2 Mergeable CLBs during XC4000 CLB mapping

Step 3: We construct a bipartite graph, where at one side of the graph are the CLBs which contain two LUTs and whose *H*-slot LUT has at least 2 inputs, and at the other side of the graph are the partial CLBs which contain one LUT. An edge (*v*, *w*) is drawn if (i) the *H*-slot LUT of CLB *v*, LUT_v, is a fanout of the LUT of CLB *w*, LUT_w; and (ii) the registered outputs of LUT_v and LUT_w, if any, are compatible. Clearly, *v* and *w* can be merged by putting LUT_w into the vacant *G*-slot of the CLB *v*. The preference weight used for the matching is input(LUT_v), whose reason will be clear in the next step. Again, we use a large enough nominal weight to guarantee a maximum cardinality matching. Each matched pair is merged into a full CLB. A matching candidate for this step is illustrated in Fig 2(b).

Step 4: If there are still CLBs with one LUT, clearly they cannot be merged with a CLB of two LUTs by a real connection to its *H*-slot LUT. Therefore, we construct another similar bipartite graph where one side of the graph are CLBs of two LUTs and the other side of the graph are CLBs of one LUT, except that for the CLBs of two LUTs, the *H*-slot LUT must have no more than 2 inputs. Because the *H*-slot LUT of the two-LUT CLB has an unused input, any single-LUT CLB can be merged with it as long as the outputs are compatible. Therefore, an edge is drawn if the registered outputs are compatible. For this matching, we use the preference weight for routability as discussed in Section 3.2.2, and compute a maximum weighted matching, and merge the matched pairs into full CLBs. Fig 2(c) illustrates a candidate for the matching of this step.

Step 5: Finally, if there are still CLBs with one LUT, they cannot be merged with remaining CLBs of two LUTs. So we have to merge them with each other by finding a maximum matching among them, where the only constraint is the compatibility of their registered outputs. We use the preference weight for routability in Section 3.2.2 for the matching. The matched pairs are merged by using the *F* and *G* function generators of a CLB. At this point, no further merging of CLBs is possible and we have obtained a mapped solution.

Mapping for Altera FLEX-8000 FPGAs

The Altera Flex-8000 family PLB is called a logic cell (LCELL). Each LCELL contains one 4-variable function generator and a register. RASP is used to produce 4-LUT solutions which are placed one per LCELL. The Altera design flow is resumed after RASP creates solutions in a Text Design File (TDF), a standard interface to the Altera Hardware Descriptor language (AHDL).

4. Experimental Results and Comparative Study

We have compared RASP performance using our technology mapping algorithms for LUT synthesis/mapping and our PLB mapping heuristics for XC3000/XC4000 FPGAs with several commercially available logic synthesis and technology mapping systems. The commercial synthesis systems are XACT™ 5.1 from Xilinx Corporation, MaxPlusII from Altera Corporation and another state-of-the-art synthesis system donated to us for use in our research which we refer to as System X.

The RASP results in the figures are chosen by a solution selector which uses a cost function based on PLB count and depth before placement and routing. Suppose that RASP produces *n* PLB mapping solutions of the form $S_1 = (PLB_1, D_1)$, $S_2 = (PLB_2, D_2)$, $S_3 = (PLB_3, D_3)$, ..., where PLB_i and Di are the PLB count and depth of solution Si. Then, we define a cost function $cost(S_i) = a*PLB_i/min\_PLB + b*D_i/min\_Depth$ where $min\_PLB = min\{PLB_1, PLB_2, ...\}$, $min\_Depth = min\{D_1, D_2, ...\}$ and *a* and b are weighting factors the user may input. In our experimental results, we have used $a = 1$, and $b = 1.2$, making depth 20% more important than PLB count. The solution with the minimum cost is selected by RASP for each design.

The total CPU time required to produce all three PLB solutions (by FlowMap, CutMap, and FlowSYN) considered by the solution selector in RASP is about 5% of the time required by place and route in the XACT tools and on the same order of magnitude as Altera's MaxPlusII.

Xilinx Part: XC3195APG223-3

Benchmark Circuit	RASP Algorithm	Delay (ns)			Depth			# CLB		
		RASP	XACT 5.1	System X	RASP	XACT 5.1	System X	RASP	XACT 5.1	System X
5xp1	FlowSYN	32.4	47.8	48.7	2	5	5	16	23	36
9sym	FlowSYN	30.3	68.8	75.8	3	7	6	7	57	62
9symml	FlowSYN	28.1	65.1	81.5	3	7	6	7	54	77
alu2	FlowSYN	80.2	121.5	143.9	6	16	14	118	102	155
alu4	CutMap	148.7	204.4	192.4	13	16	17	213	190	277
apex7	FlowSYN	60.5	66.0	70.9	3	6	6	65	59	90
C499	FlowSYN	76.1	111.1	105.9	5	9	8	128	108	162
C880	CutMap	104.9	144.1	114.3	8	15	13	183	137	199
count	CutMap	56.3	71.8	63.8	3	9	8	58	47	70
duke2	CutMap	80.8	103.2	118.5	4	8	10	151	112	178
misex1	FlowSYN	28.8	37.0	41.9	2	3	4	12	14	22
rd84	FlowSYN	29.7	63.5	67.3	3	7	7	12	37	60
vg2	FlowSYN	57.6	45.2	47.2	4	5	5	31	27	41
z4ml	FlowSYN	23.6	37.2	38.0	2	6	6	5	12	19
misex3	FlowSYN	76.3	120.9	117.1	6	13	14	68	64	83
s1196	FlowSYN	83.3	146.8	117.0	4	16	10	137	149	139
s1488	CutMap	67.1	87.7	97.4	4	5	6	166	177	176
s1494	CutMap	88.0	96.8	95.3	5	4	6	164	175	178

Fig 3: RASP Comparison with commercial synthesis systems on MCNC logic synthesis benchmarks using XC3000 LUT to CLB matching heuristic.

Xilinx Part: XC4010PQ208–4

Benchmark Circuit	RASP Algorithm	Delay (ns)			Depth			# Packed CLBs		
		RASP	XACT 5.1	System X	RASP	XACT 5.1	System X	RASP	XACT 5.1	System X
5xp1	FlowSYN	35.9	57.7	62.7	3	6	6	14	17	18
9sym	FlowSYN	46.9	61.2	52.3	3	5	5	7	38	7
9symml	FlowSYN	46.9	72.7	55.0	3	7	5	7	37	7
alu2	CutMap	120.5	133.0	107.9	12	16	11	89	80	110
alu4	CutMap	143.6	153.8	--	12	17	--	138	143	--
apex7	FlowSYN	55.8	63.0	64.2	4	5	6	42	47	50
C499	FlowSYN	84.5	122.4	91.3	5	12	9	67	99	48
C880	CutMap	113.1	124.0	147.7	11	14	18	113	109	100
count	CutMap	61.3	81.4	62.7	4	9	6	40	40	40
duke2	CutMap	72.9	92.5	86.4	4	8	9	93	88	76
misex1	FlowSYN	76.4	41.0	41.3	3	4	3	9	10	9
rd84	FlowSYN	51.0	85.0	49.8	4	7	4	10	26	11
vg2	CutMap	42.8	48.4	50.6	4	4	5	20	20	15
z4ml	FlowSYN	31.4	57.1	43.3	3	6	4	4	9	3
mm4a	FlowSYN	85.6	130.0	135.9	6	15	15	50	54	54
s1196	FlowSYN	80.0	110.8	121.6	6	11	13	99	109	109
s1488	FlowSYN	63.9	79.3	87.6	4	8	8	134	133	133
s1494	FlowSYN	68.8	80.4	79.5	5	8	7	135	133	133

Fig 4: RASP Comparison with Commercial Synthesis Systems on MCNC Logic Synthesis benchmarks using XC4000 LUT to CLB matching heuristic.

Xilinx Part: XC3195APG223–3

Benchmark Circuit	-c 0		-c 1		-c 2		-c 3	
	Routed?	# CLBs	Routed?	# CLBs	Routed?	# CLBs	Routed?	# CLBs
C880	No	183	Yes	186	Yes	196	Yes	206
duke2	No	159	No	161	Yes	166	Yes	185

Fig 5: Trade-off in routability and number of CLBs for XC3000 LUT to CLB matching heuristic. All other benchmarks routed without difficulty.

Flex 8000 Part: EPF8452AGC160–2

Benchmark Circuit	RASP Algorithm	Delay (ns)		# LCELLs	
		RASP	MaxPlus II	RASP	MaxPlus II
5xp1	CutMap	22.2	26.2	37	38
9sym	FlowSYN	21.8	37.9	15	86
9symml	FlowSYN	24.1	33.1	15	78
alu2	CutMap	66.1	76.3	196	157
alu4	CutMap	68.5	141.8	332	270
apex7	CutMap	26.9	31.4	104	80
C499	FlowSYN	41.5	44.8	161	126
C880	CutMap	60.6	70.4	250	172
count	CutMap	31.2	31.5	84	48
duke2	CutMap	36.8	46.3	198	173
misex1	CutMap	19.1	17.9	21	21
rd84	FlowSYN	24.5	38.0	54	56
vg2	CutMap	24.1	28.3	48	34
z4ml	FlowSYN	15.1	15.3	10	8
mm4a	FlowSYN	49.3	49.3	155	79
s1196	FlowSYN	47.5	61.5	234	227
s1488	CutMap	38.7	38.6	313	278
s1494	CutMap	32.8	43.4	311	277

Fig 6: RASP Comparison with Altera MaxPlusII on MCNC Logic Synthesis benchmarks.

Fig 3 and Fig 4 compare RASP performance with two commercial synthesis systems on the MCNC logic synthesis benchmarks (both combinational and sequential) for two families of Xilinx target architectures. On the MCNC logic synthesis benchmarks using the XC3195A part, RASP has lower measured delay after place and route 94% of the time, lesser depth in terms of block levels[2] 94% of the time and smaller CLB count 50% of the time. On the MCNC logic synthesis benchmarks using the XC4010 part, RASP has lower measured delay after place and route 83% of the time, lesser depth in terms of block levels 94% of the time and smaller CLB count 55% of the time. Fig 5 shows the increased routability of two CLB netlists when we require that the two LUTs placed together in each CLB share at least a certain minimum number of common inputs. The **-c x** parameter shown as a column heading is the number of common inputs required for pairing. We have observed the transition from an un-routable to a routable CLB netlist for the C880 and duke2 benchmarks with only an increase of 3 and 7 CLBs respectively. All other benchmarks routed without difficulty with no input restrictions. Fig 6 compares RASP performance with the Altera MaxPlusII synthesis package. On the MCNC logic synthesis benchmarks using the 8452A part, RASP has lower measured delay after place and route 94% of the time, and smaller LCELL count 16% of the time.

Xilinx Part: XC4010PQ208–4

Benchmark Circuit	Solution 1			Solution 2			Solution 3		
	Delay (ns)	# CLBs	R value	Delay (ns)	# CLBs	R value	Delay (ns)	# CLBs	R value
5xp1	44.3	14	2	--	--	--	--	--	--
9sym	54.9	37	2	66.8	36	1	--	--	--
9symml	64.2	36	1	72.1	35	4	74.7	34	6
alu2	126.7	78	1	129.2	77	4	--	--	--
alu4	150.8	155	5	167.0	149	2	171.6	146	4
apex7	65.3	42	0	--	--	--	--	--	--
C499	86.2	82	4	87.4	80	2	--	--	--
C880	121.3	92	4	--	--	--	--	--	--
count	53.3	38	7	91.4	89	4	--	--	--
duke2	79.3	93	3	87.3	95	2	91.4	89	4
misex1	32.3	9	2	33.4	8	4	--	--	--
rd84	59.8	27	1	62.3	26	2	--	--	--
vg2	48.7	20	2	--	--	--	--	--	--
z4ml	38.6	7	0	--	--	--	--	--	--
Vid Comp	138.5	278	1	156.3	272	7	159.8	270	6
Vid DeComp	185.4	262	0	197.5	260	5	203.7	257	2

Fig 7: Families of solutions generated using RASP/FlowMap-R and XC4000 LUT to CLB matching heuristic.

Fig 7 shows the RASP solutions for the MCNC logic synthesis benchmarks and a reconfigurable FPGA based real-time video compressor/decompressor design [ScVMJ95]. The **r** column indicates the relaxation value used during the generation of the solution. The XNF netlist for the video compressor circuit consists of 1281 simple gates, 234 registers and 79 I/Os. The video decompressor consists of 1205 simple gates, 249 registers, and 75 I/Os. RASP solution performance exceeds that of XACT™ 5.1 and System X which produced solutions of 267 CLBs, 163.7 (ns) and 274 CLBs, 185.7 (ns) respectively. RASP generates solutions which have lower critical path delay and only slightly higher CLB count for both the compressor and decompressor designs.

[2] Depth in block levels is obtained using the Xilinx Xdelay™ tool which first determines the critical path in terms of measured delay and then counts the number of block levels along the critical path. It is therefore possible that the critical path in terms of delay will be of the same or lesser depth than the critical path in terms of LUT levels in the technology mapped solution.

5. Limitations and Ongoing Work

While RASP insures that no two LUTs with conflicting registered outputs will be matched together in a CLB, it does not currently map registers to specific CLBs but instead relies on the XACT^{TM} mapper for this function. RASP does not currently support all types of non-resynthesizable elements in the XNF specification, however, this support can be easily added as future research demands. Also, although RASP is extremely powerful in both depth and final delay minimization, there are cases where the CLB usage by RASP is considerably higher. We are in the process of developing more effective area minimization algorithms while maintaining the advantage in delay minimization. Efforts are also underway to utilize the RASP framework for ongoing research in the area of retiming and automatic pipelining.

Acknowledgment

This work is partially support by ARPA/CSTO under Contract DABT63-93-C-0055 and NSF Young Investigator Award MIP9357582.

References

[Al95] Altera, *Flex 8000 and Flex 10000 Programmable Logic Device Family Data Sheets.* San Jose, CA: Altera 1995.

[AT&T95] AT&T Microelectronics, *Optimized Reconfigurable Cell Array (ORCA) Series FPGAs.* Allentown, PA: AT&T Microelectronics 1995.

[ChCD92] Chen, K. C., J. Cong, Y. Ding, A. B. Kahng, and P. Trajmar, "DAG-Map: Graph-based FPGA technology mapping for delay optimization," *IEEE Design and Test of Comput.*, pp. 7-20, Sept. 1992.

[CoDi92a] Cong, J. and Y. Ding, "An optimal technology mapping for delay optimization in lookup-table based FPGA designs," *Proc. IEEE Int. Conf. on Computer-Aided Design*, pp. 48-53, Nov. 1992.

[CoDi93b] Cong, J. and Y. Ding, "Beyond the Combinatorial Limit in Depth Minimization for LUT-Based FPGA Designs," *Proc. IEEE Int'l Conf. on Computer-Aided Design*, pp. 110-114, 1993.

[CoDi94a] Cong, J. and Y. Ding, "FlowMap: An Optimal Technology Mapping Algorithm for Delay Optimization in Lookup-Table Based FPGA Designs," *IEEE Trans. on Computer-Aided Design*, Vol. **13**(1) pp. 1-12, 1994.

[CoDi94b] Cong, J. and Y. Ding, "On Area/Depth Trade-off in LUT-Based FPGA Technology Mapping," *IEEE Trans. on VLSI Systems*, Vol. 2, June 1994.

[CoHw95a] Cong, J. and Y. Hwang, "Simultaneous Depth and Area Minimization in LUT-based FPGA Mapping," *Proc. ACM/SIGDA International Symposium on FPGAs*, 1995.

[FrRC90] Francis, R. J., J. Rose, and K. Chung, "Chortle: A Technology Mapping Program for Lookup Table-Based Field Programmable Gate Arrays," *Proc. 27th ACM/IEEE Design Automation Conference*, pp. 613-619, June 1990.

[FrRV91b] Francis, R. J., J. Rose, and Z. Vranesic, "Technology Mapping for Delay Optimization of Lookup Table-Based FPGAs," *MCNC Logic Synthesis Workshop*, 1991.

[Ka91a] Karplus, K., "XMap: A Technology Mapper for Table-lookup Field-Programmable Gate Arrays," *Proc. 28th ACM/IEEE Design Automation Conference*, pp. 240-243, June 1991.

[LaPV93] Lai, Y.-T., M. Pedram, and S. Vrudhula, "BDD Based Decomposition of Logic Functions with Application to FPGA Synthesis," *Proc. 30th ACM/IEEE Design Automation Conf.*, pp. 642-647, June 1993.

[MuNS90] Murgai, R., Y. Nishizaki, N. Shenoy, R. Brayton, and A. Sangiovanni-Vincentelli, "Logic Synthesis Algorithms for Programmable Gate Arrays," *Proc. 27th ACM/IEEE Design Automation Conf.*, pp. 620-625, 1990.

[MuSB91a] Murgai, R., N. Shenoy, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Improved Logic Synthesis Algorithms for Table Look Up Architectures," Proc. IEEE Int'l Conf. on Computer-Aided Design, pp. 564-567, Santa Clara, CA, Nov. 1991.

[MuSB91b] Murgai, R., Y. Nishizaki, N. Shenoy, R. Brayton, and A. Sangiovanni-Vincentelli, "Performance Directed Synthesis for Table Look Up Programmable Gate Arrays," *Proc. IEEE Intl'l Conf. on Computer-Aided Design*, pp. 572-575, Nov. 1991.

[SaTh92] Sawkar, P., D. Thomas, "Technology Mapping for Table-Look-Up Based Field Programmable Gate Arrays," *ACM/SIGDA Workshop on Field Programmable Gate Arrays*, pp. 82-88, Feb. 1992.

[ScKC92] Schlag, M., J. Kong, and P. K. Chan, "Routability-Driven Technology Mapping for Lookup Table-Based FPGAs," *Proc. 1992 IEEE International Conference on Computer Design*, pp. 86-90, Oct. 1992.

[ScVMJ95] B. Schoner, J. Villasenor, S. Molloy, and R. Jain, "Techniques for FPGA Implementation of Video Compression Systems," *Proc. ACM/SIGDA International Symposium on FPGAs*, 1995.

[SeSL92] Sentovich, E., K. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. Stephen, R. Brayton, and A. Sangiovanni-Vincentelli, "SIS: A System for Sequential Circuit Synthesis," *U.C. Berkeley Technical Report UCB/ERL M92/41*, May, 1992.

[Xi94a] Xilinx, *The Programmable Logic Data Book*. San Jose, CA: Xilinx 1994.

[Wo91a] Woo, N.S. "A Heuristic Method for FPGA Technology Mapping Based on the Edge Visibility," *Proc. ACM/IEEE Design Automation Conference*, pp. 248-251, San Francisco, CA, Jun. 1991.