Comparison of Boolean Satisfiability Encodings on FPGA Detailed Routing Problems*

Miroslav N. Velev‡ Ping Gao

Aries Design Automation, LLC

‡Contact author: miroslav.velev@aries-da.com

Abstract

We compare 12 new encodings for representing of FPGA detailed routing problems as equivalent Boolean Satisfiability (SAT) problems against the only 2 previously used encodings. We also consider two symmetry-breaking heuristics. Compared to other methods for FPGA detailed routing, SAT-based approaches have the advantage that they can prove the unroutability of a global routing for a particular number of tracks per channel, and that they consider all nets simultaneously. The experiments were run on the standard MCNC benchmarks. The combination of one new encoding with a new symmetry-breaking heuristic resulted in speedup of 3 orders of magnitude or 1,139× of the total execution time on the collection of benchmarks, when proving the unroutability of FPGA global routings. The maximum obtained speedup was 9,499× on an individual benchmark. On the other hand, most of the encodings had comparable and very efficient performance when finding solutions for configurations that were routable. The availability of many SAT encodings, that can each be combined with various symmetry-breaking heuristics, opens the possibility to design portfolios of parallel strategies—each a combination of a SAT encoding and a symmetry-breaking heuristic—that can be run in parallel on different cores of a multicore CPU in order to reduce the solution time, with the rest of the runs terminated as soon as one of them returns an answer. We found that a portfolio of three particular parallel strategies produced additional speedup of more than 2×.

1. Introduction

In the last seven years, dramatic improvements were achieved in both the speed and capacity of SAT solvers [10, 12, 15, 24, 32], which are now up to 5 orders of magnitude faster and can handle problems that are up to 4 – 5 orders of magnitude bigger. The new efficient SAT solvers open new possibilities for applying this technology. By translating hard Computer Science problems to equivalent SAT problems, we can directly benefit from the recent tremendous advances in SAT, and the constant stream of innovations in this extremely active research field. This paper studies techniques to efficiently solve FPGA detailed routing problems by translation to SAT.

Previous methods for FPGA detailed routing by translation to SAT [17, 18, 25, 26, 46] have implicitly exploited the observation that FPGA detailed routing problems can be represented as equivalent graph coloring problems [45], and that graph coloring problems can be reformulated as equivalent SAT problems [19, 21, 33]. They have done so by developing tools that directly translate FPGA global routing problems into equivalent Boolean formulas that are satisfiable if and only if a detailed routing exists for a particular number of tracks per channel *W*. In contrast, in this paper we first translate the FPGA detailed routing problem to an equivalent graph coloring problem in the DIMACS format [8, 20, 35] that can then be translated to SAT by any tool for representing of graph coloring problems as SAT problems, allowing us to also benefit from the progress in this field [1, 3, 4, 6, 9, 11, 14, 16, 19, 22, 27, 28, 29, 30, 31, 35, 36, 42, 43, 41].

Graph coloring is a class of Constraint Satisfaction Problems (CSPs) [5]. A CSP is a triple (*V*, *D*, *C*), where *V* is a set of variables, *D* the set of their domains, and *C* a set of constraints that the variables must satisfy. CSPs are usually solved by specialized search algorithms [5]. Alternatively, CSPs can be cast as SAT problems, as pioneered in the early 1990s by Kasif [21], Selman et al. [33], and Iwama and Miyazaki [19].

Given an undirected graph, *G*(*V*, *E*), where *V* is the set of vertices and *E* the set of edges, *graph coloring* is the problem of assigning colors to the vertices, such that no two adjacent vertices have the same color and the number of used colors is minimum. This problem is known to be NP-complete [13]. The minimum coloring uses the smallest possible number of colors, called the *chromatic number* of a graph. The task of finding a *k*-coloring for a given graph can be modeled as a CSP, where each variable represents a vertex, all vertices have the same domain, such that each value in that domain represents a possible color, and variables for adjacent vertices are constrained by disequality constraints.

The advantages of SAT-based FPGA detailed routing are that it can prove that a particular global routing does not have a detailed routing for a given number of tracks per channel, and so can guarantee optimality when a detailed routing is found for a particular number of tracks per channel *W*, such that the configuration with *W* – 1 tracks is proven unroutable. Furthermore, because in SAT-based FPGA detailed routing all constraints for the existence of a detailed routing are expressed in a monolithic Boolean formula, SAT solvers that are used to solve such formulas end up considering simultaneously the routability constraints for all nets, thus allowing the SAT solvers to potentially converge faster to a solution. This is in contrast to the one-net-at-a-time approach used in most non-SAT-based FPGA detailed routers.

Devadas was the first to reformulate a routing problem—conventional 2-layer channel routing—as an equivalent SAT problem [7]. Wood and Rutenbar [44] used Binary Decision Diagrams (BDDs) [2] for channel routing in FPGAs, but because of the limited scalability of BDDs could apply their tool to only one vertical channel at a time by introducing additional constraints on entrance and exit points for each net in the channel. Later methods for SAT-based FPGA detailed routing [17, 18, 25, 26, 46] used recent efficient SAT solvers, but did the translation to SAT with only two encodings—the log and muldirect encoding (see Sect. 2)—that are much less efficient than the encodings considered in our experiments.

*. Patents pending. Contact Aries Design Automation (aries-da.com) for licensing.

978-3-9810801-3-1/DATE08 © 2008 EDAA

Table 1. The log and muldirect encodings for representing CSPs as SAT that have been previously used to solve FPGA detailed routing problems, as well as encoding direct from which the muldirect encoding has been derived. They are illustrated on an example graph-coloring problem with two vertices, *v* and *w*, each having a domain of 3 values {0, 1, 2}. A dash indicates the absence of clauses of the corresponding type.

Encoding	Clauses			
	at-least-one	at-most-one	conflict	excluded-illegal-values
log	———	———	$l_{v1} \vee l_{v2} \vee l_{w1} \vee l_{w2}$ $\neg l_{v1} \vee l_{v2} \vee \neg l_{w1} \vee l_{w2}$ $l_{v1} \vee \neg l_{v2} \vee l_{w1} \vee \neg l_{w2}$	$\neg l_{v1} \vee \neg l_{v2}$ $\neg l_{w1} \vee \neg l_{w2}$
direct	$x_{v0} \vee x_{v1} \vee x_{v2}$ $x_{w0} \vee x_{w1} \vee x_{w2}$	$\neg x_{v0} \vee \neg x_{v1}$ $\neg x_{v0} \vee \neg x_{v2}$ $\neg x_{v1} \vee \neg x_{v2}$ $\neg x_{w0} \vee \neg x_{w1}$ $\neg x_{w0} \vee \neg x_{w2}$ $\neg x_{w1} \vee \neg x_{w2}$	$\neg x_{v0} \vee \neg x_{w0}$ $\neg x_{v1} \vee \neg x_{w1}$ $\neg x_{v2} \vee \neg x_{w2}$	———
muldirect	$x_{v0} \vee x_{v1} \vee x_{v2}$ $x_{w0} \vee x_{w1} \vee x_{w2}$	———	$\neg x_{v0} \vee \neg x_{w0}$ $\neg x_{v1} \vee \neg x_{w1}$ $\neg x_{v2} \vee \neg x_{w2}$	———

The contributions of this paper are: 1) the use of a tool flow that first translates an FPGA detailed routing problem to an equivalent graph-coloring problem in the DIMACS format, and then applies a second tool for translating of graph coloring problems to SAT, thus allowing us to benefit from the progress in that field; 2) the first application of 12 new SAT encodings for CSP [41] to FPGA detailed routing; 3) the use of symmetry-breaking heuristics when solving of graph coloring problems derived from FPGA detailed routing problems; 4) the comparison of the 12 new SAT encodings with 2 SAT encodings that have been previously used for FPGA detailed routing; and 5) the identification of the combination of one of the new encodings with a new symmetry-breaking heuristic as the most efficient single strategy.

2. Background

The regular structure of island-style FPGA arrays allows us to reformulate their detailed routing as an equivalent graph coloring problem [45]. Namely, each multi-pin net is decomposed into a collection of 2-pin nets. For each connection block that a 2-pin net passes through, we impose *exclusivity constraints* with respect to all other 2-pin nets that pass through the same connection block and that belong to other multi-pin nets, i.e., the exclusivity constraints require that 2-pin nets that belong to different multi-pin nets should not get routed on the same track in a connection block. Thus, we can represent the 2-pin nets as vertices in a CSP graph, with an edge between two vertices that correspond to 2-pin nets that should be routed on different tracks in the same connection block. Since each switching block preserves the track assignment from the connection block where a 2-pin net comes from before it gets routed to another connection block, then we only need to impose exclusivity constraints once for each pair of 2-pin nets that belong to different multi-pin nets and that pass together through several connection blocks.

To the best of our knowledge, previous work on FPGA detailed routing by translation to SAT has used only two encodings for representing CSPs as SAT—the log encoding in [17, 18, 25], and the muldirect encoding in [26, 46]. We will illustrate these encodings with the clauses that they generate for the coloring of a graph with two adjacent vertices, *v* and *w*, each having a domain of 3 values, {0, 1, 2}, representing 3 possible colors (see Table 1)—i.e., the constraints for detailed routing of two electrically distinct 2-pin nets that pass through the same connection block with 3 tracks:

Log. Uses log number of Boolean variables in the size of the domain of each CSP variable, by employing all of the Boolean variables in order to select a value from the domain of that CSP variable. Then, for each pair of CSP variables for adjacent vertices in the CSP graph, and for each common domain value of those variables, a *conflict clause* is introduced to prevent both variables from having the same value. Also introduced are clauses that exclude illegal values that are not in the domain of each CSP variable. This encoding was proposed by Iwama and Miyazaki [19].

Direct. A new Boolean variable x_{vi} is introduced for each CSP variable *v* and each domain value *i* of *v*, in order to encode whether *v* is assigned the value *i*. For each CSP variable, the introduced Boolean variables are constrained with an at-least-one clause that ensures that the CSP variable is assigned at least one value, and at-most-one clauses that guarantee that only one value is assigned. Conflict clauses are introduced accordingly to prevent equal values of CSP variables for adjacent vertices in the CSP graph. This encoding was proposed by de Kleer [6]. It has been used for board-level multiterminal net routing in [34].

Muldirect. The multivalued direct encoding is a variant of the direct encoding, where the at-most-one clauses are omitted. Therefore, a SAT solution could assign several domain values to a CSP variable, so that there is no longer a 1-to-1 correspondence between SAT and CSP solutions. From a multivalued SAT solution we extract a CSP solution by taking any one of the allowed values for each CSP variable. This encoding was first used by Selman et al. [33].

Given a CSP variable, its set of domain values, and the Boolean variables introduced for a SAT encoding of that CSP variable, we will refer to an assignment to those Boolean variables that selects a particular domain value as an *indexing Boolean pattern* for that domain value for the given CSP variable.

3. Structural SAT Encodings for CSP

We can represent each CSP variable with a tree of ITE (for "if-then-else") operators that selects a value from this CSP variable's set of domain values, which appear as leaves of the tree [41]. In this representation, an ITE operator ITE(i, t, e) takes three arguments: a Boolean variable i, such that if i is *true* then the ITE selects the then-argument t and otherwise the else-argument e, where the then- and else-arguments are either ITE subtrees or domain values that appear as leaves of the tree. We will refer to a Boolean variable that controls an ITE in the tree for a CSP variable as an *indexing Boolean variable*, since it helps to select one of the domain values for that CSP variable.

The ITE tree for each CSP variable depends on a unique set of indexing Boolean variables that is disjoint from the sets of indexing Boolean variables used in ITE trees for other CSP variables. We also impose the *restriction* that every indexing Boolean variable appear at most once on every path that starts from the root of an ITE tree for a CSP variable and ends with a leaf, i.e., a domain value of that CSP variable. Note that *an ITE tree functions like a multi-input multiplexor, and will select a leaf (domain value) for every assignment to the indexing Boolean variables in that ITE tree*. Furthermore, depending on the structure of the ITE tree, a domain value can be selected by a partial assignment to the indexing Boolean variables for that CSP variable, namely an assignment specifying values for only those indexing Boolean variables that appear on the only path from the given domain value to the root of the ITE tree for that CSP variable. Therefore, *this representation does not require at-least-one or at-most-one constraints for the Boolean variables introduced for each CSP variable*, since the structure of the ITE trees guarantees that each CSP variable will evaluate to exactly one domain value for each assignment to its indexing Boolean variables. Required are only conflict clauses to ensure that two adjacent CSP variables do not evaluate to the same common domain value.

Depending on the structure of the ITE tree introduced for a CSP variable, we can obtain various representations for CSP variables. One extreme case is a chain of ITE operators, where the second argument of every ITE is a domain value, while the third argument is either a nested chain of such ITEs, or an ITE selecting between two domain values. This structure is illustrated in Fig. 1.a for a CSP variable that has 13 domain values, $\{v_0, v_1, ..., v_{12}\}$, which get selected by means of 12 indexing Boolean variables, $i_{v0}, i_{v1}, ..., i_{v11}$. In the general case, given a CSP variable that has a domain of k values, the chain will have $k - 1$ ITE operators, each controlled by a different indexing Boolean variable. In this example, the first domain value, v_0, is selected when the first indexing Boolean variable i_{v0} is *true*; the second domain value, v_1, is selected when $\neg i_{v0} \wedge i_{v1}$ is *true*, and so on. We will refer to the resulting SAT encoding as *ITE-linear*, due to the linear structure of the ITE tree.

Another extreme case is when the ITE tree is balanced, such that every path in the tree goes through either $\lceil \log_2(k) \rceil$ or $\lceil \log_2(k) \rceil - 1$ ITE operators, as shown in Fig. 1.b. We call the resulting SAT encoding *ITE-log*. It can be viewed as a variant of the log encoding (see Table 1), where some of the indexing Boolean patterns do not contain the last indexing Boolean variable, so that no constraints are needed to exclude illegal indexing Boolean patterns, as in the log encoding. In general, the ITE tree for a CSP variable can have any structure.

Since there can be many structurally different ITE trees that have the same number of leaves, we can construct a different encoding from each such ITE tree with number of leaves that equals the number of domain values for a CSP variable. The different structure will result in different probabilities of selecting a particular domain value, assuming that all the indexing Boolean variables in the trees have equal probabilities of being assigned the value *true*.

Treating ITE trees as logic blocks in translation to Conjunctive Normal Form (CNF), and generating Boolean formulas with many big ITE trees resulted in up to 2 orders of magnitude speedup in solving of CNF representations of Boolean formulas from formal verification of complex pipelined, superscalar, and VLIW microprocessors [37 – 40].

4. Using a Hierarchy of SAT Encodings

We can use a hierarchy of different SAT encodings [41] in order to select the domain values of a CSP variable. Namely, we can first use one SAT encoding to partition the domain of that CSP variable into subdomains, and then a different SAT encoding to either select the values in each subdomain, or to further partition it into smaller subdomains, and so on. For a given domain or a subdomain, we restrict its subdomains to not overlap. Then, we can use the same SAT encoding (with the same set of Boolean variables) to select the values in each subdomain at the same new level in the hierarchy, or to further divide the subdomains at that level into smaller subdomains. A domain value is selected if it gets selected in its corresponding subdomain at the lowest level in the hierarchy, and for each of the higher levels in the hierarchy, the corresponding larger subdomain that contains this value also gets selected by the SAT encoding for that level of the hierarchy.

For example, we can combine the ITE-log and ITE-linear encodings and thus obtain hybrid encodings, e.g., by first using several levels of the ITE-log encoding in order to partition each CSP variable's domain into subdomains, and then use the ITE-linear encoding to select the values from each subdomain. We will call this encoding ITE-log-i+ITE-linear, where i is the number of indexing Boolean variables used in the ITE-log part of the encoding, i.e., the number of ITE-log levels. The encodings ITE-log-1+ITE-linear and ITE-log-2+ITE-linear are illustrated in Fig. 1.c and Fig. 1.d, respectively, for the example CSP variable with 13 domain values. In the ITE-log-2+ITE-linear encoding shown in Fig. 1.d, the domain value v_4 is selected as the value of the ITE tree, i.e., as the value of the encoded CSP variable, when $i_{v0} \wedge \neg i_{v1} \wedge i_{v2}$ is *true*; the domain value v_5 is selected when $i_{v0} \wedge \neg i_{v1} \wedge \neg i_{v2} \wedge i_{v3}$ is *true*; and v_6 is selected when $i_{v0} \wedge \neg i_{v1} \wedge \neg i_{v2} \wedge \neg i_{v3}$ is *true*. Then, if j_{v0}, j_{v1}, j_{v2}, and j_{v3} are the corresponding indexing Boolean variables from the ITE tree for a CSP variable that is adjacent in the CSP graph to the CSP variable represented in Fig. 1.d, symmetric indexing Boolean patterns will be used to select that variable's domain values, so that the domain value v_4 will be selected as that CSP variable's value when $j_{v0} \wedge \neg j_{v1} \wedge j_{v2}$ is *true*, etc. Hence, the conflict clause that will prevent the two adjacent CSP variables from simultaneously evaluating to v_4 will be $(\neg(i_{v0} \wedge \neg i_{v1} \wedge i_{v2}) \vee \neg(j_{v0} \wedge \neg j_{v1} \wedge j_{v2}))$, i.e., $(\neg i_{v0} \vee i_{v1} \vee \neg i_{v2} \vee \neg j_{v0} \vee j_{v1} \vee \neg j_{v2})$.

We can similarly construct new encodings by using any of the previous simple encodings (see Sect. 2) or any variation of the ITE-tree encodings (see Sect. 3) for each level of the hierarchy in the resulting hybrid encoding. If the encoding used for a particular level allows the simultaneous selection of several domain values or several subdomains, respectively, then we extract a CSP solution by taking any one of the allowed values for each CSP variable. If at a given level in the hierarchy, some of the subdomains have fewer domain values than the rest of the subdomains at that level, we impose constraints

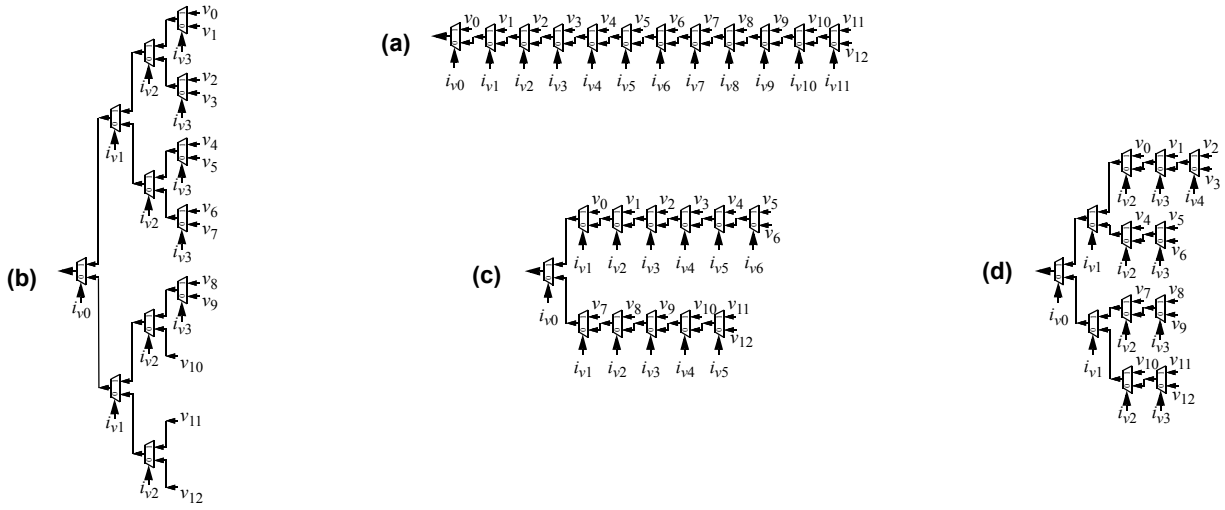

Figure 1. Four ITE trees for a CSP variable *v* that has a domain of 13 values, {v_0, v_1, ..., v_{12}}, resulting in the four new SAT encodings for CSP: (a) ITE-linear; (b) ITE-log; (c) ITE-log-1+ITE-linear; and (d) ITE-log-2+ITE-linear. ITEs are shown as multiplexors. In general, the ITE tree for a CSP variable can have any structure.

to restrict the Boolean variables that index the smaller subdomains in order to prevent the selection of non-existent values. Such constraints are necessary for the previous 3 encodings (see Sect. 2), while in the case of ITE-tree encodings we can use smaller versions of the ITE-trees for the smaller domains.

Note that it is not required that all the subdomains at a particular level of a hierarchical encoding be further divided into smaller subdomains (or their domain values be indexed) by using the same simple encoding. That is, we can have different simple encodings that are used to further partition the subdomains from the same level. However, in the experiments, we will use only one simple encoding for each level in a hierarchical encoding. For example, under such a construction, if we use a hierarchy of two levels of the muldirect encoding for a total number of colors K, such that the number of Boolean variables used for the first-level muldirect encoding is n, then the number of Boolean variables used for the second-level muldirect encoding will be $\lceil K/n \rceil$.

More recently, Kwon and Klieber [22] have proposed a SAT encoding for CSPs that, based on our terminology, can be classified as direct-i+direct for a 2-level version, where i is the number of Boolean variables used in the top encoding, and could include more than two levels, but with the direct encoding used at each level. However, our method for hierarchical composition of encodings is completely general, and allows for any simple encoding to be used at any level of a hierarchical encoding.

5. Symmetry-Breaking Heuristics

As noted by Van Gelder [36], if a graph-coloring problem is being solved for K colors, we can select any $K - 1$ vertices, and constrain the i^{th} of them to have a color of less than i, if the colors are numbered 0 through $K - 1$.

One of the heuristics that Van Gelder proposed, *b1*, starts by restricting the node of maximum degree in the graph to be the first node in the sequence of $K - 1$ vertices with restricted colors. Then, the neighbors of that node are sorted in descending order of their degrees and added to the sequence in that order up to the $(K - 2)^{\text{nd}}$ of them, such that ties are broken by the sum of the neighbors' degrees.

We also implemented a second symmetry-breaking heuristic, which we call *s1*, where the $K - 1$ vertices in the restricted sequence are picked as the $K - 1$ vertices of highest degrees in the CSP graph, sorted in descending order of their degrees with ties broken based on the sum of the neighbors' degrees.

6. Results

The experiments were run on the MCNC FPGA detailed routing benchmarks [23], based on their global routings provided with the source code of the SEGA-1.1 FPGA detailed router [23]. We used a Dell Precision 380 workstation with a dual-core 3.73-GHz Intel Pentium Extreme Edition processor, 2×2-MB on-chip L2-cache, and 8 GB of memory, under Red Hat Linux Enterprise v.4. (Only one core was used, and all experiments consumed less than 0.8 GB of memory.)

We compared the two SAT encodings for CSP that have been previously applied to FPGA detailed routing—the log encoding in [17, 18, 25], and the muldirect encoding in [26, 46]—with 12 new encodings—the ITE-linear, ITE-log, ITE-log-1+ITE-linear, ITE-log-2+ITE-linear, ITE-log-2+direct, ITE-log-2+muldirect, ITE-linear-2+direct, ITE-linear-2+muldirect, direct-3+direct, direct-3+muldirect, muldirect-3+direct, and muldirect-3+muldirect. The comparison was done on both routable and unroutable FPGA configurations—resulting in, respectively, satisfiable and unsatisfiable Boolean formulas.

We used the SAT solvers siege_v4 [32] and MiniSat [10], where MiniSat is the winner in the recent annual SAT-solver competitions, while siege_v4 was a winner before that. We found that siege_v4 was faster by at least a factor of 2 when proving the unsatisfiability of formulas from unroutable configurations—those Boolean formulas took much longer to solve, while the satisfiable formulas from routable configurations were solved by either SAT solver in usually a fraction of a second, such that MiniSat had a small advantage.

Table 2 shows the results for some of the challenging unroutable FPGA configurations, using the SAT solver siege_v4. From the 2 previously used encodings, the muldirect encoding outperformed the log encoding, and so the

Table 2. Comparison of the best performing encodings on some of the challenging unroutable FPGA configurations. Reported is the total CPU time [sec]—the sum of the times to generate the graph-coloring problem + its translation to Conjunctive Normal Form (CNF) + the time to SAT-solve it with siege_v4. Symmetry-breaking heuristics *b1* and *s1* are described in Sect. 5, and a dash indicates that no symmetry-breaking was used. Bold font designates the min. time for each benchmark and for the total time, as well as the max. speedup of the total time.

Benchmark	Total CPU time [sec] for: translation to graph coloring + translation to CNF + SAT solving														
	muldirect			ITE-linear		ITE-log		ITE-linear-2 +direct		ITE-linear-2 +muldirect		muldirect-3 +muldirect		direct-3 +muldirect	
	—	*b1*	*s1*	*b1*	*s1*	*b1*	*s1*	*b1*	*s1*	*b1*	*s1*	*b1*	*s1*	*b1*	*s1*
alu2	12.83	9.33	5.13	1.53	1.80	**0.10**	0.11	22.45	25.99	6.69	34.77	10.81	5.58	14.00	32.58
too_large	18.60	16.39	18.45	0.17	0.50	**0.12**	**0.12**	1.67	0.83	0.55	0.33	**0.12**	0.70	0.13	1.13
alu4	1,018.10	275.36	317.01	**22.20**	146.32	928.10	248.34	29.74	62.56	61.82	620.80	72.27	34.92	24.1	36.72
C880	1,147.36	1,202.57	14.82	19.95	**1.79**	93.80	38.17	16.32	3.21	11.65	3.05	34.80	12.15	29.68	28.86
apex7	1,443.80	245.82	20.71	140.11	3.62	76.30	8.81	22.23	2.17	30.53	**1.78**	49.50	6.89	91.12	4.51
C1355	3,231.10	**0.12**	75.78	0.15	7.67	0.15	17.85	0.30	2.83	0.20	2.47	0.14	8.29	0.14	5.10
vda	1,054,417	49,243	83,391	32,562	3,983	508	162	203	**111**	224	255	1,321	764	1,736	1,351
k2	470,235	10,497	10,967	21,767	11,798	15,541	21,882	11,675	10,862	10,218	**426**	33,178	1,060	11,569	13,734
Total	1,531,524	61,490	94,810	54,513	15,943	17,148	22,357	11,971	11,071	10,553	**1,344**	34,667	1,893	13,464	15,194
Speedup wrt. muldirect w/o symmetry	1.00×	24.91×	16.15×	28.09×	96.06×	89.31×	68.50×	127.94×	138.34×	145.12×	**1,139**×	44.18×	809.25×	113.75×	100.80×

results for the muldirect encoding are included in the table. (The muldirect encoding also performed better than the direct encoding.) The results from the best 6 of the 12 new encodings are also shown in the table.

Overall, the 12 new encodings produced up to 3 orders of magnitude speedup of the total execution time, relative to the 2 previously used encodings for FPGA detailed routing as SAT. The use of symmetry-breaking heuristics increased the speedup, such that each of the symmetry-breaking heuristics had an advantage on some of the benchmarks with some of the encodings. However, symmetry-breaking heuristic *s1* helped produce the greatest speedups.

The combination of encoding ITE-linear-2+muldirect with symmetry-breaking heuristic *s1* resulted in the greatest reduction of the total execution time for the most challenging unroutable configurations of the eight benchmarks, shown in Table 2, producing 3 orders of magnitude or 1,139× speedup relative to encoding muldirect (the better one of the previously used encodings log and muldirect) without symmetry-breaking. The maximum obtained speedup was almost 4 orders of magnitude or 9,499× for benchmark vda with encoding ITE-linear-2+direct and symmetry-breaking heuristic *s1*, relative to encoding muldirect without symmetry breaking. On the other hand, most of the encodings had comparable and very efficient performance when finding solutions for configurations that were routable—i.e., for which the equivalent Boolean formulas were satisfiable—with either siege_v4 or MiniSat.

The advent of multicore CPUs allows us to execute parallel programs. Then, the availability of many SAT encodings, that can each be combined with various symmetry-breaking heuristics, opens the possibility to design portfolios of parallel strategies—each a combination of a SAT encoding and a symmetry-breaking heuristic—that can be run in parallel on different cores of a multicore CPU in order to reduce the solution time, with the rest of the runs terminated as soon as one of them returns an answer. A portfolio of two parallel strategies—encoding ITE-linear-2+muldirect with symmetry-breaking heuristic *s1*, and encoding muldirect-3+muldirect with symmetry-breaking heuristic *s1*—produced an additional speedup of 1.84× relative to encoding ITE-linear-2+muldirect with symmetry-breaking heuristic *s1*, based on the total execution time for the eight benchmarks. A portfolio of three parallel strategies—the above two, together with encoding ITE-linear-2+direct with symmetry-breaking heuristic *s1*—produced a speedup of 2.30× relative to encoding ITE-linear-2+muldirect with symmetry-breaking heuristic *s1*.

7. Conclusion

We compared 12 new encodings for representing of FPGA detailed routing problems as equivalent SAT problems—the ITE-linear, ITE-log, ITE-log-1+ITE-linear, ITE-log-2+ITE-linear, ITE-log-2+direct, ITE-log-2+muldirect, ITE-linear-2+direct, ITE-linear-2+muldirect, direct-3+direct, direct-3+muldirect, muldirect-3+direct, and muldirect-3+muldirect—against 2 previously used encodings—the log and muldirect encodings. We also considered two symmetry-breaking heuristics—Van Gelder's heuristic *b1* [36], and a newly proposed heuristic *s1*. Encoding ITE-linear-2+muldirect with symmetry-breaking heuristic *s1* resulted in 3 orders of magnitude speedup of the total execution time for the most challenging unroutable configurations of the considered benchmarks. The maximum obtained speedup was almost 4 orders of magnitude or 9,499× for benchmark vda with encoding ITE-linear-2+direct and symmetry-breaking heuristic *s1*. A portfolio of three parallel strategies produced additional speedup of more than 2× on unroutable benchmarks. Most of the encodings had comparable and very efficient performance when finding solutions for configurations that were routable.

References

[1] C. Bessière, E. Hebrard, and T. Walsh, "Local Consistencies in SAT," *Conference Theory and Applications of Satisfiability Testing (SAT '06)*, LNCS 2919, May 2003.

[2] R.E. Bryant, "Graph-Based Algorithms for Boolean Function Manipulation," IEEE Transactions on Computers, Vol. C-35, No. 8 (August 1986), pp. 677–691

[3] A. Chmeiss, and L. Sais, "About Neighborhood Substitutability in CSPs," *Third Int'l Workshop on Symmetry in Constraint Satisfaction Problems*, Kinsale, Ireland, 2003.

[4] J. Culberson, Graph Coloring Page. http://www.cs.ualberta.ca/~joe/Coloring/

[5] R. Dechter, *Constraint Processing*, Morgan Kaufmann Publishers, 2003.

[6] J. de Kleer, "A Comparison of ATMS and CSP Techniques," *11th International Joint Conference on Artificial Intelligence (IJCAI '89)*, August 1989.

[7] S. Devadas, "Optimal Layout via Boolean Satisfiability," *International Conference on Computer-Aided Design (ICCAD '89)*, November 1989, pp. 294–297.

[8] DIMACS Graph-Coloring Problems. http://mat.gsia.cmu.edu/COLOR04/

[9] L. Drake, A.M. Frisch, I.P. Gent, and T. Walsh, "Automatically Reformulating SAT-Encoded CSPs," *Workshop on Reformulating Constraint Satisfaction Problems, 2002*.

[10] N. Eén, and N. Sörensson, MiniSat. http://www.cs.chalmers.se/Cs/Research/FormalMethods/MiniSat

[11] A. Frisch, and T. Peugniez, "Solving Non-Boolean Satisfiability Problems with Stochastic Local Search," *Int'l Joint Conference on Artificial Intelligence*, 2001.

[12] Z. Fu, Y. Mahajan, and S. Malik, "New features of the SAT'04 versions of zChaff," *Working Notes on the SAT'04 Solver Competition*, 2004.

[13] M.R. Garey, and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman and Co., New York, 1979.

[14] I. Gent, "Arc Consistency in SAT," *European Conf. on AI*, 2002.

[15] E. Goldberg, and Y. Novikov, "BerkMin: A Fast and Robust SAT-Solver," *Design, Automation, and Test in Europe*, March 2002.

[16] H.H. Hoos, "SAT-Encodings, Search Space Structure, and Local Search Performance," *Int'l. Joint Conference on Artificial Intelligence*, 1999.

[17] W.N.N. Hung, X. Song, E.M. Aboulhamid, A. Kennings, and A. Coppola, "Segmented Channel Routability via Satisfiability," ACM Transactions on Design Automation of Electronic Systems (TODAES), Vol. 9, No. 4 (October 2004).

[18] W.N.N. Hung, X. Song, T. Kam, L. Cheng, and G. Yang, "Routability Checking for Three-Dimensional Architectures," IEEE Transactions on VLSI Systems, Vol. 12, No. 12 (December 2004), pp. 1398–1401.

[19] K. Iwama, and S. Miyazaki, "SAT-Variable Complexity of Hard Combinatorial Problems," *IFIP World Computer Congress*, 1994, pp. 253–258.

[20] D.S. Johnson, and M.A. Trick, *eds.*, "The Second DIMACS Implementation Challenge," DIMACS Series in Discrete Mathematics and Theoretical Computer Science, 1993. http://dimacs.rutgers.edu/Challenges

[21] S. Kasif, "On the Parallel Complexity of Discrete Relaxation in Constraint Satisfaction Networks," *Artificial Intelligence*, Vol. 45, No. 3.

[22] G. Kwon, and W. Klieber, "Efficient CNF Encoding for Selecting 1 from N Objects," *Fourth Workshop on Constraints in Formal Verification (CFV '07)*, July 2007.

[23] G. Lemieux, SEGA FPGA Routing Tool Website. http://www.eecg.toronto.edu/~lemieux/sega/sega.html

[24] M.W. Moskewicz, C.F. Madigan, Y. Zhao, L. Zhang, and S. Malik, "Chaff: Engineering an Efficient SAT Solver," *Design Automation Conference (DAC '01)*, June 2001.

[25] G.-J. Nam, K.A. Sakallah, and R.A. Rutenbar, "A New FPGA Detailed Routing Approach via Search-Based Boolean Satisfiability," IEEE Trans. on CAD of Integrated Circuits and Systems, Vol. 21, No. 6 (2002).

[26] G.-J. Nam, F.A. Aloul, K.A. Sakallah, R.A. Rutenbar, "A Comparative Study of Two Boolean Formulations of FPGA Detailed Routing Constraints," IEEE Transactions on Computers, Vol. 53, No. 6 (2004).

[27] S.D.Prestwich, "Maintaining Arc-Consistency in Stochastic Local Search," *Techniques for Implementing Constraint Programming Systems*, 2002.

[28] S.D. Prestwich, "Local Search on SAT-Encoded Colouring Problems," *Theory and Applications of Satisfiability Testing (SAT '03)*, 2003.

[29] S.D. Prestwich, "Full Dynamic Interchangeability with Forward Checking and Arc Consistency," *Workshop on Modeling and Solving Problems With Constraints*, 2004.

[30] S.D. Prestwich, "Full Dynamic Substitutability by SAT Encoding," *Principles and Practice of Constraint Programming (CP '04)*, 2004.

[31] A. Ramani, F. Aloul, I. Markov, and K. Sakallah, "Breaking Instance-Independent Symmetries in Exact Graph Coloring," Journal of Artificial Intelligence Research (JAIR), 2005.

[32] L. Ryan, *Efficient Algorithms for Clause Learning SAT Solvers*, M.S. thesis, Simon Fraser University, Canada, 2004.

[33] B. Selman, H. Levesque, and D. Mitchell, "A New Method for Solving Hard Satisfiability Problems," *AAAI '92*, 1992, pp. 440–446.

[34] X. Song, W.N.N. Hung, A. Mishchenko, M. Chrzanowska-Jeske, A. Kennings, and A. Coppola, "Board-Level Multi-Terminal Net Assignment for the Partial Crossbar Architecture," IEEE Transactions on VLSI Systems, Vol. 11, No. 3 (June 2003).

[35] M. Trick, Network Resources for Coloring a Graph. http://mat.gsia.cmu.edu/COLOR/color.html

[36] A. Van Gelder, "Another Look at Graph Coloring via Propositional Satisfiability," Discrete Applied Mathematics, 2007.

[37] M.N. Velev, "Efficient Translation of Boolean Formulas to CNF in Formal Verification of Microprocessors," *Asia and South Pacific Design Automation Conference (ASP-DAC '04)*, January 2004, pp. 310–315.

[38] M.N. Velev, "Exploiting Signal Unobservability for Efficient Translation to CNF in Formal Verification of Microprocessors," *DATE '04*, February 2004.

[39] M.N. Velev, "Comparative Study of Strategies for Formal Verification of High-Level Processors," *ICCD '04*, October 2004, pp. 119–124.

[40] M.N. Velev, "Comparison of Schemes for Encoding Unobservability in Translation to SAT," *ASP-DAC '05*, January 2005, pp. 1056–1059.

[41] M.N. Velev, "Exploiting Hierarchy and Structure to Efficiently Solve Graph Coloring as SAT," *International Conference on Computer-Aided Design (ICCAD '07)*, November 2007, pp. 135–142.

[42] T. Walsh, "Search in a Small World," *Sixteenth International Joint Conference on Artificial Intelligence (IJCAI '99)*, T. Dean, *ed.*, July – August 1999, pp. 1172–1177.

[43] T. Walsh, "SAT v CSP," *Principles and Practice of Constraint Programming*, 2000.

[44] R.G. Wood, and R.A. Rutenbar, "FPGA Routing and Routability Estimation via Boolean Satisfiability," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Vol. 6, No. 2 (June 1998), pp. 222–231.

[45] Y.-L. Wu, and M. Marek-Sadowska, "An Efficient Router for 2-D Field-Programmable Gate Arrays," *European Design Automation Conference*, 1994.

[46] H. Xu, R.A. Rutenbar, and K.A. Sakallah, "sub-SAT: A Formulation for Relaxed Boolean Satisfiability with Applications in Routing," *International Symposium on Physical Design (ISPD '02)*, April 2002.