TABLE II
COMPARISON RESULTS ON FU AND REGISTER COUNTS

	BIP		Ours		Inc (%)		BIP	Ours	Inc (%)
Bench	N_+	N_*	N_+	N_*	N_+	N_*	Reg	Reg	
aircraft	38	40	38	40	0	0	159	159	0
chem	15	16	15	16	0	0	48	48	0
dir	8	7	8	7	0	0	72	72	0
feig_dct	37	12	40	12	8	0	114	132	17
honda	7	6	7	7	0	17	24	24	0
mcm	12	9	13	9	8	0	38	40	5
pr	5	8	6	9	20	13	18	20	11
u5ml	16	17	16	17	0	0	60	60	0
wang	5	8	5	8	0	0	20	21	5
arai	6	3	6	3	0	0	14	19	36
lee	8	4	8	4	0	0	17	20	18
diffeq	2	2	2	3	0	50	7	7	0
fir11	1	2	1	2	0	0	11	11	0
cftmdl	15	16	16	16	7	0	34	41	21
cftb1st	12	6	14	6	17	0	52	58	12
fft	4	4	5	5	25	25	16	16	0
idct	10	9	11	9	10	0	39	39	0
matmul	16	32	16	32	0	0	48	48	0
wavelet	8	16	8	16	0	0	25	25	0
jacob	10	8	10	8	0	0	30	30	0
chendct	12	8	13	8	8	0	33	39	18
chenidct	15	12	15	12	0	0	39	40	3
kalman	2	2	3	2	50	0	8	9	13
lowpass	6	8	6	8	0	0	44	49	11
AVG					6	4			7

tion in percentage. As the table shows, our algorithm can reduce the multiplexer input by 33%, 29%, and 22% on average in comparison to LEA, BIP, and k-cof algorithm, respectively.

In order to verify that reducing the number of multiplexer inputs leads to global interconnect minimization, we generated the layout of all benchmark designs using Cadence SOC Encounter, a widely used commercial EDA tool, and measured the actual total wirelength. Specifically, all binding results were first converted into Verilog register transfer level (RTL) description. The same place and routing flow was applied to all the RTL designs. The FUs were predesigned as hard macro-cells. The timing target of each benchmark design was kept constant. After circuit layouts were created, we used SOC Encounter to report the total wirelength of all interconnects beyond the macro-cells. Columns 9–15 in Table I show the wirelength comparison results. Our algorithm reduces total global interconnects by 26%, 19%, and 18% over LEA, BIP, and k-cof algorithm, respectively.

Table II shows the comparison of FU and register counts between the optimal algorithm, i.e., BIP and our algorithm. Since our benchmarks only contain additions and multiplications, N_+ and N_* are used to represent adder counts and multiplier counts, respectively. Columns 2–5 list the absolute FU counts whereas columns 6 and 7 give the increase percentages of our scheme over BIP. As Table II shows, the average increases of the numbers of adders and multipliers are 6% and 4%, respectively. Column 8 and 9 show the register counts. The last column shows the register count increase of our scheme in percentage. For half of the benchmarks, our algorithm produces the same results as BIP. On average, our algorithm increases the register count slightly by 7%. Such an increase does not affect the total layout area substantially.

VII. CONCLUSION

In this paper, we present a simultaneous FU and register binding algorithm for interconnect reduction. Our algorithm identifies long paths in the compatibility graph generated from a DFG, and conducts FU and register binding concurrently. Our scheme targets the minimization of multiplexer inputs by analyzing the flow dependency and common inputs of operations. Experimental results show that our algorithm reduces the number of multiplexer inputs by more than 20%, on average, in comparison to previously proposed algorithms [1], [4], [5]. Our scheme achieves a total wirelength reduction by 18% on average at the cost of slight FU and register increases.

REFERENCES

[1] D. Chen and J. Cong, "Register binding and port assignment for multiplexer optimization," in *Proc. Asia South Pacific Design Autom. Conf.*, Jan. 2004, pp. 68–73.
[2] J. Cong, Y. Fan, and W. Jiang, "Platform-based resource binding using a distributed register-file microarchitecture," in *Proc. Int. Conf. Comput.-Aided Design*, Nov. 2006, pp. 709–715.
[3] C.-Y. Huang, Y.-S. Chen, Y.-L. Lin, and Y.-C. Hsu, "Data path allocation based on bipartite weighted matching," in *Proc. Design Autom. Conf.*, Jun. 1990, pp. 499–504.
[4] D. Chen, J. Cong, and Y. Fan, "Low-power high-level synthesis for FPGA architectures," in *Proc. Int. Symp. Low Power Electron. Design*, Aug. 2003, pp. 134–139.
[5] F. J. Kurdahi and A. C. Parker, "REAL: A program for register allocation," in *Proc. Design. Autom. Conf.*, Jun. 1987, pp. 210–215.
[6] L. Stok, "Interconnect optimization during data path allocation," in *Proc. Conf. Eur. Design Autom. (EURO-DAC)*, Sep. 1990, pp. 141–145.
[7] T. Kim and C. L. Liu, "An integrated data path synthesis algorithm based on network flow method," in *Proc. IEEE Custom Integr. Circuits Conf.*, May 1995, pp. 615–618.
[8] J. Cong and J. Xu, "Simultaneous FU and register binding based on network flow method," in *Proc. Design Autom. Test Eur.*, Mar. 2008, pp. 1057–1062.
[9] G. D. Micheli, "Resource sharing and binding," in *Synthesis and Optimization of Digital Circuits*. New York: McGraw-Hill, 1994, pp. 229–266.
[10] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, "Single-source shortest path," in *Introduction to Algorithms*, 2nd ed. Cambridge, MA: MIT Press, 2001, pp. 580–619.
[11] M. B. Srivastava and M. Potkonjak, "Optimum and heuristic transformation techniques for simultaneous optimization of latency and throughput," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 3, no. 1, pp. 2–19, Mar. 1995.
[12] N. Dutt and C. Ramchandran, "Benchmarks for the 1992 high level synthesis workshop," Univ. California, Irvine, Tech. Rep. 92-107, 1992.
[13] P. Panda and N. Dutt, "1995 high level synthesis design repository," in *Proc. Int. Symp. Syst. Synthesis*, Sep. 1995, pp. 170–174.
[14] P. G. Paulin and J. P. Knight, "Force-directed scheduling for the behavioral synthesis of ASICs," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 8, no. 6, pp. 661–679, Jun. 1989.

Accurately Handle Don't-Care Conditions in High-Level Designs and Application for Reducing Initialized Registers

Hong-Zu Chou, *Student Member, IEEE,* Kai-Hui Chang, *Member, IEEE,* and Sy-Yen Kuo, *Fellow, IEEE*

Manuscript received July 27, 2009; revised October 8, 2009, and December 10, 2009. Current version published March 19, 2010. This work was supported by the National Science Council, Taiwan, under Grant NSC 97-224-E-002-216-MY3, the Excellent Research Projects of National Taiwan University, under Grant 95R0062-AE00-05, and the Small Business Innovation Project of Ministry of Economic Affairs, Taiwan, under Grant 1Z960401. This paper was recommended by Associate Editor, I. Bahar.

H.-Z. Chou and S.-Y. Kuo are with the Department of Electrical Engineering, National Taiwan University, Taipei 10617, Taiwan (e-mail: hzchou@lion.ee.ntu.edu.tw; sykuo@cc.ee.ntu.edu.tw).

K.-H. Chang is with Avery Design Systems, Inc., Andover, MA 01810 USA (e-mail: changkh@avery-design.com).

Digital Object Identifier 10.1109/TCAD.2010.2042905

***Abstract*—Don't-care conditions are utilized by many synthesis tools because such conditions provide additional flexibility for logic optimization. However, most techniques only focus on the gate level because it is difficult to handle such conditions accurately at behavior and register transfer levels. This is problematic since the trend is to move toward high-level synthesis. In this paper, we propose innovative methods to handle such conditions accurately at high-level designs. In addition, we propose three novel algorithms based on our new methods to minimize the number of registers that need to be initialized, which can reduce the routing resources used by the reset signals and alleviate the routing problem. We applied our techniques to a five-stage pipelined processor and successfully reduced the number of control registers that need to be initialized by 53%, demonstrating the effectiveness of our approach.**

***Index Terms*—Don't-care (DC), initialized register reduction, symbolic simulation, synthesis.**

I. Introduction

To address the increasing need for smaller and lower power designs, many new logic optimization techniques are being developed. One of the most promising methods is to utilize don't-care (DC) conditions in logic synthesis [2], [8]. Since a DC (often denoted as X in logic simulation) in a logic design means the value can be either 0 or 1, it provides additional flexibility for synthesis tools to produce better netlists. Most of these techniques, however, focus on optimizing gate-level netlists only. Not being able to utilize DCs at a higher level of abstraction, such as the register transfer level (RTL) or electronic system level (ESL), greatly reduces the optimization power of high-level synthesis tools [8]. In particular, if DCs can be known at higher levels, then the synthesis tool can potentially generate quite different netlists, providing different architecture decisions or achieving better optimizations. One major reason why DCs are rarely utilized at higher levels for logic synthesis is that it is difficult to handle such DCs accurately at these levels. As Haufe and Rogin suggested [6], X is one major source of RTL and gate-level mismatch. One previous research suggests to eliminate X values throughout the design [1]. However, their methodology can significantly reduce synthesis quality because DCs are no longer available for optimization.

Our first contribution in this paper is to use high-level symbolic simulation to accurately handle the propagation of X values. In addition, we propose a SAT formulation that can prove whether or not the X values will be propagated to any observation points, such as primary outputs or registers. Our techniques not only provide more optimization opportunities in high-level synthesis but also solve many verification problems [3], [9], [10]. In this way, designers can use X values whenever appropriate and verify whether those Xs are indeed DCs using our methods, producing more optimization opportunities for synthesis tools.

Being able to handle X conditions accurately at higher levels enables many logic optimizations. For instance, with the miniaturization of transistors, routing has become a serious problem [7]. By reducing the number of registers which need to be initialized, the routing problem can be alleviated because routing resources used by the reset signal can be reduced. Our second contribution is three algorithms to find registers that can be uninitialized in a design. The first two algorithms are based on *Boolean satisfiability* (SAT) and *quantified Boolean formula* (QBF), respectively. These algorithms are optimal and can find the maximum set of registers that do not need the reset signal; however, they are computation intensive. Therefore, we also propose a fast heuristic algorithm that finds approximate solutions. Our empirical evaluations show that the heuristic algorithm can produce good results within significantly shorter time. Moreover, when the algorithm is applied to a five-stage pipelined processor, it found that approximately 53% of the control registers (36% of total registers) do not need to be initialized for a reset period that is five cycles long. These results show that our techniques can improve synthesis quality dramatically and help alleviate routing problems.

II. Background and Related Work

In this section, we first describe how Xs (DCs) are handled at the RTL[1] and then provide necessary background for the SAT and QBF problems.

A. Handling X at the RTL

In order to utilize DCs for logic synthesis optimizations, X values at the RTL should be modeled the same as those at the gate level. However, due to the semantic gap between RTL and gate-level models, RTL versus gate-level mismatch may occur [3], [10]. To cope with such a problem, Haufe and Rogin [6] proposed a solution that utilizes RTL code transformation to avoid unexpected X-propagation. This approach, however, only handles certain types of RTL constructs and is not generic. To address the above-mentioned problems, our preliminary work [4] presented an innovative method that exploits high-level symbolic simulation to accurately handle X, which will be described later in this paper.

B. SAT and QBF

Given a propositional Boolean formula, a SAT problem is to determine if there is a satisfying assignment—true and false values to the variables that make the expression evaluate to true. If a Boolean formula is represented as a conjunction of clauses, where a clause is a disjunction of literals, it is called a *conjunctive normal form (CNF)*. QBF is closely related to SAT. The difference is that all variables in SAT are existentially ($\exists$) quantified, while variables in QBF can be existentially ($\exists$) or universally ($\forall$) quantified. In general, QBF is presented in a prenex normal form shown as the following:

$$Q_1 V_1 Q_2 V_2 \ldots Q_n V_n | \Phi, (n \geq 0) \tag{1}$$

where Q_i denotes a quantifier, either existential ($\exists$) or universal ($\forall$), such that $Q_i \neq Q_{i+1}$. V_i denotes a set of Boolean variables, and Φ denotes the entire formula. A QBF problem is satisfiable if there exists an assignment that satisfies (1).

III. Accurately Handle X Values

To accurately handle X-propagation at the RTL and check whether the Xs can be observed, we propose a new scheme that converts the problem to a SAT instance. The built instance is shown in Fig. 1, and the scheme works as follows.

1) Duplicate design A to create an identical copy A'. Use symbol X_{Ai} to represent the initial state of register R_i in design A. Similarly, use symbol $X_{A'i}$ for design A'.

[1]In this paper, RTL includes both structural and behavior level.

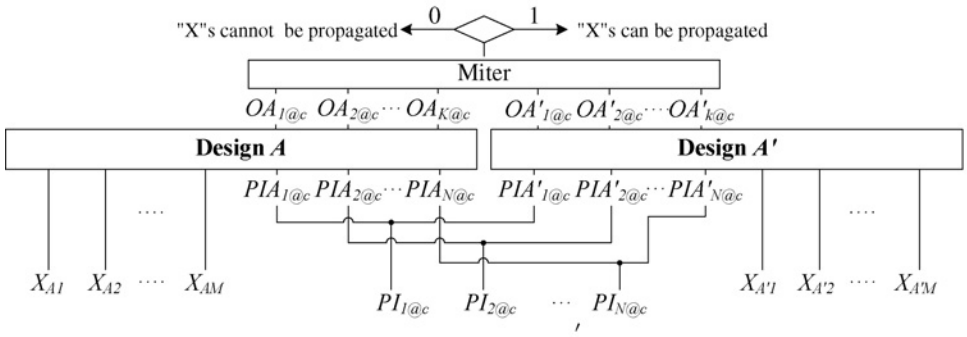

Fig. 1. Illustration of symbolic X-propagation checking ($PI_{i@c}$, $PIA_{i@c}$ and $PIA'_{i@c}$ denote primary inputs, X_{Ai} and $X_{A'i}$ denote registers' initial states, and $OA_{i@c}$ and $OA'_{i@c}$ denote observation points).

2) Inject a symbol $PI_{i@c}$ to both design's inputs PI_i at cycle c, perform symbolic simulation for C cycles.
3) Observation points $OA_{1@c}$, ..., $OA_{K@c}$ and $OA'_{1@c}$, ..., $OA'_{K@c}$ are connected to a miter. When the miter's output is 1, there exists a pair of observation points, $OA_{i@c}$ and $OA'_{i@c}$, whose values are different; otherwise, the equivalency of these observation points is proved.

Note that Fig. 1 only shows a snapshot at cycle c for PI and O. The SAT instance should include PI and O from cycle 1 to cycle C. Finally, a SAT solver is called to find if a solution exists to make $miter$ = 1. If the solver can find a solution, then "X"s can be propagated to one or more observation points.

IV. Reducing Initialized Registers

In this section, we first propose two optimal algorithms using SAT and QBF formulations to minimize the number of initialized registers. Since they may be time-consuming, we propose a heuristic algorithm that requires significantly lower computation cost. We also provide several scalability enhancements and propose a new method to find good reset sequences in this section.

A. SAT-Based Optimal Solution

Our first method to find the minimal number of initialized registers is based on the SAT-instance construction described earlier. To serve the new purpose, however, two components are added to the SAT-instance formulation, including 1) multiplexers, and 2) cardinality constraints. A multiplexer is added to every register to model the initial state in the SAT formulation. More specifically, when the select line S_i is 1, the initial states of X_{Ai} and $X_{A'i}$ are both 0, which means register R_i is initialized to 0. When S_i is 0, X_{Ai} and $X_{A'i}$ are regarded as free variables and become the primary inputs of the SAT instance. In other words, they are uninitialized. Cardinality constraints restrict the number of select lines that can be set to 1 simultaneously to m. This number also represents the number of registers that need to be initialized. The cardinality constraints are implemented by an adder that performs a bitwise addition of the select lines and outputs the sum, m.

The algorithm is presented in Fig. 2. In the algorithm, ss denotes a set of registers to be initialized. SS_m denotes the solution space that contains different combinations of exactly m registers that should be initialized, where

```
1: for (m = 0; m <= M; m++) do
2:    ss = check_X(miter = 1, m, SS_m);
3:    while (ss) do
4:       SS_m = SS_m \ {ss};
5:       ss = check_X(miter = 1, m, SS_m);
6:    end while
7:    if (SS_m is not empty) then
8:       return SS_m;
9:    end if
10: end for
```

Fig. 2. SAT-based optimal method for finding a minimum set of registers that need to be initialized.

$0 \leq m \leq M$ (M = total number of registers). Function $check\_X(miter, m, SS)$ denotes the procedure to check the observability of X values in solution space SS under the constraints $miter$ and m using the constructs shown in Fig. 1. Initially, m is set to 0, and a SAT solver is called to find if a solution exists to make $miter$ = 1 under the constraint m. If the problem is satisfiable and solution ss is returned, it means that initializing the combination of m registers returned by $check\_X$ in the solution ss will propagate X values to observation points, and ss should be excluded from the solution space SS_m. The procedure is repeatedly performed until no solution is returned. At this point, two situations should be considered: 1) SS_m is empty, which means that if only m registers are initialized, no matter which combination, X values in one or more of the uninitialized registers will be propagated to observation points. When this happens, we increment m to look for solutions with one more register to initialize; and 2) SS_m is not empty, which means initializing m registers using any combination of registers that remain in SS_m can make sure no Xs can be propagated to observation points, and the solution is returned. Note that observation points typically include two types of signals: primary outputs and registers. In practice, users can decide which signals to observe based on design requirements.

The above formulation uses the constructs shown in Fig. 1 to find and exclude all combination of m registers that when only those registers are initialized, the X values in the uninitialized variables will be propagated to observation points. When $check\_X$ can no longer find a solution under constraint m, it means choosing any combinations of m registers that still

```
1: for (m = 0; m <= M; m++) do
2:    if (ss_m =QBF_check(Φ_m)) do
3:       return ss_m;
4:    end if
5: end for
```

Fig. 3. QBF-based optimal method for finding a minimum set of registers that need to be initialized.

exist in SS_m will not propagate X values to the observation points. Since we start our search from $m = 0$, we can find the minimal number of registers that should be initialized.

One inefficiency of this algorithm comes from iteratively solving all possible solutions under the same m. Therefore, in the next section, we formulate the problem using QBF so that this iterative step is no longer necessary.

B. QBF-Based Optimal Solution

In this section, we extend the proposed SAT formulation to a QBF one which can check the observability of all possible uninitialized registers simultaneously. We first create a Boolean formula Φ which denotes the property that no X values will be propagated to the observation points, then we formulate a QBF problem as follows:

$$\exists V_s \forall V_u \exists V_r | \Phi. \tag{2}$$

V_s denotes the set of select line, V_u denotes the free variables that connect to the inputs of multiplexers when the select line is 0, and V_r denotes the rest of the variables in the CNF. The formulation means "find a set of assignments to select lines so that no matter what values those uninitialized registers (i.e., X symbols) are, the values for the rest of the variables satisfy the CNF constraint." Select lines are put before the universal quantifier because the assignments to the select lines represent the set of registers that should be initialized. By universally quantifying the X symbols and putting them after the select lines, the solution we find guarantees that as long as we initialize the registers according to the solutions returned for the select lines, no X will propagate to any observation point because the CNF must be satisfied for all V_u.

The pseudo code of our QBF-based algorithm is presented in Fig. 3. In line 2, function $QBF\_check(\Phi_m)$ denotes the procedure to check the satisfiability of Boolean formula Φ_m, where the subscript m is a constraint to control the number of initialized registers. Since QBF implicitly encodes all possible solutions for the variables that are universally quantified, we can simply check the satisfiability of Φ_m under different constraint m. If a solution ss_m is found, it means that ss_m is an assignment of select lines which prevents X propagation.

C. Fast Heuristic Algorithm

The idea behind our heuristic algorithm is that not all combinations are checked for X-propagation. Instead, only T possible solutions are randomly selected for checking, where T is a predefined threshold determined empirically. Based on the results, a new solution space is created for the next round. The pseudo-code of the algorithm is presented in Fig. 4. Initially, $check\_X$ is used to find sets of registers to initialize so that no X can be propagated under the constraint $m = M - 1$ (i.e., only one register is uninitialized), and these solutions are added to a set $seed_{M-1}$. After that, solution space SS_m

```
1: seed_{M-1} = {ss_i ∈ SS_{M-1} &&
     check_X(minter = 1, m = M − 1, ss_i) == null};
2: for (m = M − 2, m ≥ 0; m − −) do
3:    SS_m = span(seed_{m+1}, seed_{M−1});
4:    {ss_1, ss_2, ... ss_T}=randomly_select(SS_m);
5:    foreach (ss_i) do
6:       if (check_X(minter = 1, m, ss_i) == null) do
7:          seed_m = seed_m ∪ ss_i;
8:       end if
9:    end foreach
10:   if (seed_m is empty) do
11:      return seed_{m+1};
12:   end if
13: end for
```

Fig. 4. Heuristic method for finding registers that can be uninitialized.

is created by spanning the solutions comprised in $seed_{m+1}$, where $0 \leq m \leq M - 2$. This is achieved in the span function by adding each register in $seed_{M-1}$ to each combination of registers in $seed_{m+1}$ (line 3) to form a new set of candidate combinations of registers for $check\_X$ in SS_m. To avoid cost for checking all possible solutions in SS_j, the algorithm randomly selects T solutions, $ss_1, ss_2, \ldots, ss_T$, for X-propagation checking. If ss_i cannot propagate X values, it will be added to $seed_m$. The same procedure is performed repeatedly with a smaller m until $seed_m$ is empty. At this point, $seed_{m+1}$ includes the solutions that $M - (m + 1)$ registers can be uninitialized.

D. Generating Reset Sequence

Given a reset sequence, the techniques described so far can reduce the number of initialized registers. However, a good reset sequence may further reduce the number of initialized registers because the sequence can initialize some of the registers. In this section, we propose a new method that uses QBF to find such a sequence. Note that this problem is considerably different from finding a homing sequence because uninitialized registers cannot be controlled.

The reset sequence generation problem can be formulated as follows. Given a design and the number of reset cycle C, find an input sequence and a set of registers to initialize so that the number of initialized registers is minimal and no Xs are propagated to any observation point.

To solve this problem, we reuse the QBF formulation shown in (2) with one change: the symbols that model the inputs to the circuit (i.e., $PI_{n@C}$ in Fig. 1) should be moved to the first existential quantifier. This new formulation then means: "find a set of assignments to select lines and an input sequence so that no matter what values of those undefined variables are, the values for the rest of the variables satisfy the CNF constraint." Note that both select lines and input sequences are existential quantified, and free variables are universally quantified. Therefore, if the QBF problem can be satisfied, then the returned solution of select lines and input sequences can make sure no X will propagate to the observation points. By applying the algorithm shown in Fig. 3, we can then find a set of minimal number of registers to initialize along with a reset sequence.

If the reset state is given, then the initialized register minimization problem can actually be solved without duplicating the instance: just use the reset state to constrain the CNF [Φ in (2)] and no miter is necessary.

TABLE I
COMPARISON OF OPTIMAL SAT, OPTIMAL QBF, AND HEURISTIC ALGORITHMS USING PIPE

Cycles C	#Clauses	SAT-Based Optimal				QBF-Based Optimal				Heuristic ($T = 2$)			
		Init. Reg.	Runtime (s)	Max. Mem. (MB)	#SAT Calls	Init. Reg.	Runtime (s)	Max. Mem. (MB)	#QBF Calls	Init. Reg.	Runtime (s)	Max. Mem. (MB)	#SAT Calls
1	5139	11	2414.69	311.47	4095	11	3.58	15.22	12	11	1.91	56.20	12
2	5098	10	2401.49	311.47	4083	10	3.33	20.45	11	10	1.82	56.20	13
3	5016	8	2326.83	311.41	3797	8	2.69	20.42	9	8	1.80	56.20	17
4	4934	6	1531.38	311.40	2510	6	2.06	20.65	7	6	1.98	56.20	21
5	4893	5	841.76	213.91	1586	5	1.76	20.64	6	5	2.15	56.20	23
6	4811	3	82.23	117.38	299	3	1.21	20.86	4	3	2.07	56.20	27
7	4811	3	85.3	117.38	299	3	1.20	20.86	4	3	2.06	56.20	27
8	4770	2	17.46	76.87	79	2	0.92	20.89	3	2	2.22	56.20	29
9	4721	1	2.39	60.78	13	1	0.59	20.79	2	1	2.48	56.20	31
10	4721	1	2.47	60.79	13	1	0.60	20.79	2	1	2.55	56.20	31
11	4721	1	2.39	60.79	13	1	0.60	20.79	2	1	2.54	56.20	31
12	4670	0	0.01	57.27	1	0	0.29	20.76	1	0	3.48	56.20	33

TABLE II
RESET SEQUENCE GENERATION FOR PIPE USING QBF FORMULATION

Cycles C	#Clauses	QBF-Based Optimal			
		Init. Reg.	Runtime (s)	Max. Mem. (MB)	#QBF Calls
1	5139	11	3.63	15.23	12
2	5098	10	3.43	20.5	11
3	5016	8	2.82	20.41	9
4	4934	6	2.26	20.61	7
5	4893	5	1.93	20.63	6
6	4811	3	1.34	20.86	4
7	4811	3	1.33	20.88	4
8	4770	2	1.01	20.91	3
9	4721	1	0.66	20.71	2
10	4721	1	0.66	20.71	2
11	4721	1	0.67	20.73	2
12	4670	0	0.34	20.76	1

V. EXPERIMENTAL RESULTS

In our experiments, we used two designs, PIPEline (PIPE) and DLX, to evaluate the performance of our algorithms. Since the heuristic algorithm is based on a random scenario, we ran each configuration of experiment 30× to obtain their average, minimal, and maximal. Only register values at the end of the reset period were considered as observation points, and the experiments were conducted with a commercial symbolic simulator called Insight [12]. The reported runtime includes symbolic simulation and SAT/QBF solving. The machine we used was a Linux machine with 2 GHz Xeon processor and 17G memory. MiniSat [5] and sKizzo [14] were selected as the SAT and QBF solvers, respectively. For SAT-based experiments, we used ABC [11] to perform some synthesis-for-verification optimizations before calling SAT.

A. Comparing Optimal and Heuristic Algorithms Using PIPE

To compare the optimal and heuristic algorithms, we developed a simple design called PIPE containing 12 word-level registers, and each register can be uninitialized after certain cycles. The results are shown in Table I. Note that the symbolic simulator that we used performs certain word-level optimizations before the problem is converted to bit-level Boolean expression; therefore, in this benchmark, the number of clauses actually decreases when the simulation cycle increases. In general, more simulation cycles will produce larger CNF instances with more clauses, as our next benchmark shows in Table III.

Table I shows that the SAT-based optimal algorithm can always find the minimum set of registers that need to be initialized; however, the memory usage is much higher than QBF-based and heuristic algorithms. In addition, the runtime of SAT-based optimal algorithm is long due to its optimal nature and iterative scheme. From the results, we observed that the runtime decreases as the number of cycles increases. The reason is that as the reset period becomes longer, more registers can be uninitialized, resulting in smaller numbers of satisfiable solutions. Therefore, it is no longer necessary to try all combinations for a specific m. Similar to the SAT-based algorithm, the QBF-based algorithm can also find the optimal solutions. However, since QBF solver is called only once per iteration, it outperforms the SAT-based one most of the time. This result suggests the QBF formulation is a better choice to solve this problem.

Our results show that the optimal algorithms have scalability issues. Comparatively, the heuristic algorithm is a greedy approach that checks T possible combinations only. As shown in Table I, when $T = 2$, the heuristic algorithm produced the same results as the optimal ones. We also observed that as the number of cycles increases, SAT solver also requires more time to produce the result. However, the maximum runtime is still shorter than 4 s and the maximum memory usage is still less then 60 MB, suggesting that our heuristic algorithm can produce good results within a significantly shorter time.

To evaluate how well the QBF formulation described in Section IV-D can be used to generate reset sequences, we conducted an experiment using PIPE, and the results are shown in Table II. Compared with the QBF results in Table I, we found that their runtime and memory usage are similar, suggesting that asking the QBF solver to return a reset sequence did not make the problem more difficult to solve.

B. Evaluating Heuristic Algorithm on DLX

DLX is a 32-bit microprocessor with five-stage pipeline running the MIPS-lite instruction set. The BugUnder Ground project from Michigan [13] provides a DLX implementation, and it is used to evaluate the performance of our heuristic approach. The DLX implementation has 75 control registers, 32 general registers and three dummy registers (RDaddr_reg, Wrt_en and Control_state in cpu.v). The number of registers in the synthesized netlist is 1652, suggesting that utilizing DCs at the RTL is much more efficient because only 75 word-level registers need to be handled. We initialized general and dummy registers by default because their X values are known to be propagated. In addition, primary inputs Iin and Din were constrained to 0, and registers that need initialization were forced to 0 for all cycles.

We examined the effect of varying the number of cycles C and the threshold T on the results. As shown in Table III, as the number of cycles increases, there is a gradual increase of registers that can be uninitialized because the Xs may be

TABLE III
PERFORMANCE OF HEURISTIC ALGORITHM ON DLX ($C = 1-5$, $T = 2-6$)

Cycles C	#Clause	$T = 2$				$T = 3$				$T = 4$			
		Uninit. Reg.			Runtime	Uninit. Reg.			Runtime	Uninit. Reg.			Runtime
		Min.	Max.	Avg.	(s)	Min.	Max.	Avg.	(s)	Min.	Max.	Avg.	(s)
1	669 811	6	27	18.3	101.98	8	28	21.07	117.8	11	29	24.9	154.01
2	1 047 115	24	35	30.6	126.22	27	35	32.93	156.12	26	35	32.83	209.34
3	1 438 851	29	38	35.87	139.78	27	38	36.93	170.43	35	38	37.47	235.02
4	1 991 815	31	38	37	141.91	35	38	37.67	171.65	36	38	37.83	241.89
5	2 683 935	40	40	40	144.5	40	40	40	177.26	40	40	40	250.82

Cycles C	#Clause	$T = 5$				$T = 6$				
		Uninit. Reg.			Runtime	Uninit. Reg.			Runtime	Max. Mem.
		Min.	Max.	Avg.	(s)	Min.	Max.	Avg.	(s)	(MB)
1	669 811	17	29	25.37	196.04	18	29	26.33	207.88	61.55
2	1 047 115	31	35	34.47	226.83	31	35	34.5	230.55	62.08
3	1 438 851	34	38	37.67	246.44	35	38	37.44	251.45	62.22
4	1 991 815	36	38	37.9	250.43	37	38	37.97	254.56	62.22
5	2 683 935	40	40	40	249.99	40	40	40	260.38	61.80

masked with additional cycles. Therefore, in practice one can gradually increase C until the number of uninitialized registers stops increasing.

Since DLX is a five-stage processor, many registers that can be uninitialized will be identified within five cycles. Therefore, when $C = 5$ the proposed approach found that 40 registers do not propagate their X values. We also observe that when the threshold T becomes larger, more combinations of registers will be checked, and better results can be produced. It is interesting to note that the results of $C = 5$ are the same regardless of the value of T. The reason is that all registers contained in $seed_{74}$ can be uninitialized when $C = 5$, and the SAT solver cannot find any satisfiable solution from the constrained solution space. We also analyzed the reason why some registers must be initialized and found that they are intermediate registers to store temporary values for specific instructions. Since we bind primary inputs to 0, those instructions cannot be generated to initialize those registers.

Runtime of the above experiments is presented in Table III. Apparently, the required runtime increases with the number of cycles C and the threshold T because of the complexity in SAT translation and satisfiability checks. However, the average maximum runtime is still smaller than 5 min. In addition, since the heuristic approach does not add constraints to the SAT problem, its memory usage is always less than 63 MB. The empirical results show that approximately 40/75 = 53% control registers (36% of total registers) can be uninitialized, which can already significantly save routing resources. Therefore, we do not need to use a large threshold which may increase runtime.

VI. CONCLUSION

DCs provide additional flexibility for circuit optimizations. However, existing techniques typically focus on gate-level netlists only, mostly due to the difficulty to handle X accurately at the RTL or ESL. Not being able to use DC conditions at those levels reduces the quality of synthesized netlists. To address this problem, we propose innovative techniques to accurately handle X values using high-level symbolic simulation and check observability of X values using SAT-solver. In addition, we utilize the DCs for a novel optimization: identifying the registers that can be uninitialized for synthesis optimizations. This new optimization reduces the number of registers that need reset signals and can alleviate routing problems. Our proposed methods found that 53% of control registers (36% of total registers) in a DLX processor can be uninitialized for a reset period that is five cycles long, and the runtime is smaller than 5 min, suggesting that our methods provide a practical building block for new synthesis optimizations.

REFERENCES

[1] L. Bening, "A two-state methodology for RTL logic simulation," in *Proc. Design Automat. Conf.*, 1999, pp. 672–677.
[2] R. A. Bergamaschi, D. Brand, L. Stok, M. Berkelaar, and S. Prakash, "Efficient use of large don't cares in high-level and logic synthesis," in *Proc. Int. Conf. Comput.-Aided Design*, 1995, pp. 272–278.
[3] D. Brand, R. A. Bergamaschi, and L. Stok, "Be careful with don't cares," in *Proc. Int. Conf. Comput.-Aided Design*, 1995, pp. 83–86.
[4] H.-Z. Chou, K.-H. Chang, and S.-Y. Kuo, "Handling don't-care conditions in high-level synthesis and application for reducing initialized registers," in *Proc. Design Automat. Conf.*, 2009, pp. 412–415.
[5] N. Éen and N. Sörensson, "An extensible SAT-solver," in *Proc. Theory Applicat. Satisfiability Testing*, 2003, pp. 502–518.
[6] C. Haufe and F. Rogin, "Ad hoc translations to close verilog semantics gap," in *Proc. Design Diagnostics Electron. Circuits Syst.*, 2008, pp. 1–6.
[7] M. D. Moffitt, J. A. Roy, and I. L. Markov, "The coming of age of (academic) global routing," in *Proc. Int. Symp. Phys. Design*, 2008, pp. 148–155.
[8] R. Ranjan, Y. Antonioli, A. Hunter, and O. Petlin, *Formal Verification Enables Safe X Handling*, Dec. 2008 [Online]. Available: http://www.scdsource.com/article.php?id=324
[9] A. Sülflow, G. Fey, C. Braunstein, U. Kühne, and R. Drechsler, "Increasing the accuracy of SAT-based debugging," in *Proc. Design, Automat. Test Eur.*, 2009, pp. 1326–1331.
[10] M. Turpin, "Why are X's dangerous?" *The Dangers of Living With an X*. Boston, MA: Synopsys Users Group, 2003, pp. 10–19.
[11] Berkeley Logic Synthesis and Verification Group, *ABC: A System for Sequential Synthesis and Verification* [Online]. Available: http://www.eecs.berkeley.edu/~alanmi/abc/abc.htm
[12] *Avery Design Systems, Inc.* [Online]. Available: http://www.avery-design.com
[13] V. Bertacco, T. Austin, and I. Wagner. (2007, Aug. 15). *Bug Under-Ground* [Online]. Available: http://bug.eecs.umich.edu
[14] M. Benedetti. (2007, Jan. 14). *sKizzo: A QBF Solver* [Online]. Available: http://skizzo.info