Boolean matching of sequential elements

Shankar Krishnamoorthy and Frederic Mailhot
Synopsys Inc.
700C, E.Middlefield Rd., Mountain View, CA 94043

Abstract — In most logic synthesis systems, technology mapping to a target technology is performed using structural matching techniques. Recently there has been a lot of interest on the usage of boolean techniques to do matching of combinational logic. In this paper, we present an extension of boolean matching to perform technology mapping of sequential elements. The new technique is capable of recognizing the presence of complex sequential elements including JK flip-flops, multiplexed flip-flops, flip-flops with asynchronous behavior and complex latches. The underlying algorithm uses a notion of "timed" variables and relies on a fast boolean matching technique to achieve efficiency. We have implemented these ideas in a simple technology mapper and contrast the results with two other sequential mapping techniques.

I. Introduction

With the ever increasing complexity of ASICs, synthesis tools are becoming widely accepted in the design community. Designers can now specify a high-level functional description of circuit behavior, set constraints on the design and specify the technology library in which the resulting circuit must be realized. This information is then fed to a set of synthesis tools which generate a netlist that is optimized with respect to the given constraints. The optimized netlist uses components from the specified target technology library. Synthesis is composed of two main steps: high-level synthesis and logic (or gate-level) synthesis. In this paper, we focus on technology mapping, an important phase in logic synthesis.

The technology mapping problem can be stated as follows: "Given a technology independent network (a directed acyclic graph of nodes and edges, representing the functionality of a logic circuit), and a technology library of gates, find a 'covering' or a realization of the original network in terms of the gates in the technology library that meets (or comes closest to meeting) the constraints imposed on the circuit". Technology mapping consists of two distinct steps:

1. The matching step:: We try to find all gates in the technology library that can be used to replace a sub-network in the original network.

2. The covering step:: We define a "covering" of a network as a set of matches (library elements), such that every node in the subject graph is contained in at least one match. The covering step finds sets of matches ("covers") and chooses one that is of least cost.

Several approaches have been proposed to solve the technology mapping problem. Keutzer proposed a tree matching solution which was implemented in DAGON[1, 2]. A graph matching scheme was proposed by Detjens et al. in MIS [3, 4]. A rule-based solution was implemented in SOCRATES [5]. All three solutions use structural equivalence as the basis for the matching step. Recently, a number of solutions based on boolean matching have been proposed [10, 11, 13]. Boolean matching differs from the structural matching techniques in that the emphasis is on logical equivalence rather than structural equivalence. We briefly review three boolean matching techniques in section III.

A commonly used technique to optimize sequential circuits is to partition the original circuit into its combinational and sequential parts. The combinational part of the circuit is characterized to reflect input phase relations, arrival and required times, area and other constraints. It is then optimized by a combinational optimization tool that performs technology independent optimizations followed by technology dependent optimizations such as technology mapping. The memory elements that were set aside earlier are inserted back into the circuit. At this stage, the circuit can be further optimized by mapping the sequential elements to the target technology. Technology mapping of sequential elements can improve the quality of the resulting circuit by reducing the number of levels of logic between sequential elements or primary inputs, thereby speeding up the circuit. It can also reduce the area of the resulting circuit.

Some techniques for technology mapping of sequential elements have been proposed earlier. SIS uses a structural technique similar to the one used in MIS for sequential mapping. The synchronous behaviour of sequential cells in the library are expressed as pattern trees. A tree matching algorithm is used to find matches in the subject graph [6].

In rule-based systems, sequential mapping is performed by enumerating a set of rules in the rule-base that represent pattern graphs for different sequential elements. This results in elaborate descriptions of various kinds of flip-flops like JK, D, load-enable, negative edge-triggered etc. in the rule base. Since there are a large variety of sequential elements in various technology libraries, more than 60% of a typical rule base could be devoted to rules pertaining to mapping of sequential elements. This figure was obtained by examining a rule base that was constructed for gates in 20 different standard cell libraries [7].

In this paper, we propose a formulation to define the functionality of a sequential element as a combinational logic equation. We also present a

scheme to perform technology mapping of sequential elements using boolean matching. The rest of the paper is organized as follows: In section II, we briefly describe the boolean matching technique and review existing algorithms to do boolean matching. In section III, we introduce some notations. Section IV motivates the need for mapping of complex sequential cells. In section V, we present a new model for sequential elements and demonstrate its effectiveness. Section VI outlines a new method to do technology mapping of sequential elements using boolean matching. In section VII, we present experimental results on a set of examples.

II. Boolean matching

Boolean matching is a technique to recognize logic equivalences between a sub-network in the subject graph and a library cell. In commonly used sequential elements such as MUXed flops, flip-flops with load-enable, JK flip-flops and flip-flops with asynchronous set-reset, there are reconvergent fanouts in the synchronous and asynchronous logic components of the cell. A structural technique (e.g. rule-based) would require exhaustive enumeration of all patterns of all sequential elements for completeness. On the other hand, boolean matching is a general technique that can be applied to match any sequential gate regardless of its structure. Therefore, we choose boolean matching as the basic paradigm for mapping sequential elements.

Definition: Two functions *f* and *g* are said to be *NPN equivalent* if and only if *f* can be obtained from *g* (or vice versa) by input variable negation, input variable permutation and function negation. Both *f* and *g* are said to be in the same *NPN equivalence class*. For example, AND, OR, NOR, NAND all belong to the same NPN class [8].

A boolean matching scheme must have an efficient way to determine if two functions belong to the same NPN class. This operation has a complexity of $O(n!\ 2^n)$ where n is the number of inputs of the function. Mailhot et al. presented a boolean matching technique, where the switching function represented by the library cell was compared against the function of a sub-graph in the subject graph to determine NPN-equivalence [9, 10]. In order to efficiently deal with the complexity of the NPN-equivalence check, they proposed a heuristic in which they used the unateness and symmetry properties. They demonstrated area/delay improvements over the structural matching techniques used in MIS-II. The main drawback with this heuristic is that the symmetry computation step requires $O(n^2)$ BDDs to be built.

Burch et al. observed that in order to check if two functions f_1 and f_2 belong to the same NPN class, it is sufficient to compare their *permutation-phase canonical forms* f_1* and f_2* for logic equivalence [11]. They presented an exact canonicalization procedure for phase assignment of inputs. However, this procedure has complexity $\psi = O(|V|^2 + |V| \cdot n^3)$ where |V| is the number of nodes in the BDD and n is the number of variables in the support of the function. Due to the complexity of the canonicalization step, the matching phase can be expensive.

Mohnke et al. observed that instead of using a computationally expensive but exact canonicalization algorithm, fast heuristics could be applied to pseudo-canonicalize the phase assignment and permutations of the inputs [12]. These heuristics were not exact (i.e did not guarantee a unique phase assignment or permutation). However, they did produce a unique phase assignment or permutation for several functions. The algorithm computes a set of signatures $\{S_i(v)\}$ for each variable v in the support of a function f using a set of signature generator functions $\{S_i\}$. Each signature generator function S_i provides a criterion for distinguishing variables in the support of the function that could not be distinguished by any of the earlier signatures in the set. This way it was possible to compute canonical forms for several functions very efficiently. However, there were some functions for which the signatures were ineffective and hence canonical forms could not be computed. In these cases, all the pseudo-canonical forms of the function would have to be stored, thereby increasing memory usage relative to the scheme presented in [11].

III. Notation

We adopt the following notation for the rest of the paper:

For a function *f*,
$\Omega(f)$ represents the support set of the function *f*
Φ_f represents a set of phase assignments to inputs of *f* obtained by applying a phase canonicalization procedure to *f*
π_f represents a set of input permutations obtained by applying a permutation canonicalization procedure to *f*
$\Phi_f(v)$ is the phase of the variable v under phase assignment Φ_f
$\pi_f(v)$ is the new position of the variable v under permutation π_f

Given two sets A and B, the set A — B contains all elements in A that are not in B.

A *flip-flop* refers to a sequential cell that is edge-sensitive. It does not have transparent behaviour.

A *latch* refers to a sequential cell that is level-sensitive. It has transparent behaviour.

IV. Technology mapping of sequential elements

Sequential elements are an integral part of digital circuits. A technology mapping solution that considers sequential elements should improve the area and delay of the final implementation. Area and delay improvements occur in designs where complex sequential gates replace simple sequential elements and surrounding logic. For example, assume a logic network with the function of a multiplexor (MUX) that drives the data pin of a D flip-flop which lies on the critical path. We can reduce the critical path delay if we replace the flip-flop and its surrounding logic by a multiplexed flip-flop. (Fig 1)

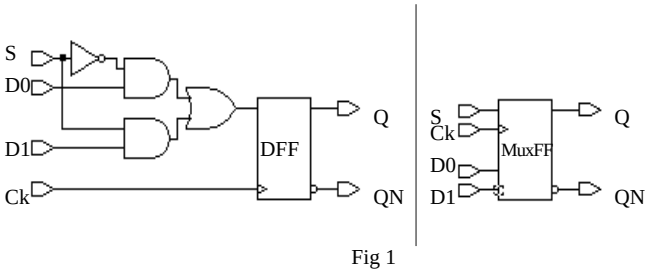

Fig 1

V. A NEW FORMULATION FOR SEQUENTIAL ELEMENTS

Boolean matching depends on the presence of a switching function representation for both the library cell and the sub-network to be matched. In order to incorporate boolean matching into technology mapping of sequential elements, we first represent a sequential element by a combinational logic function.

We now introduce a model for a sequential element. This model can represent all flip-flops and latches except Master-Slave latches. The generic model of a sequential element (GEN) is a cell with 6 inputs and 2 outputs. Table 1 explains the meaning of these pins. Table 2 describes commonly used sequential elements using this model.

TABLE 1

GENERIC SEQUENTIAL CELL (GEN)

Pin	Type	Function
sync	Input	synchronous behavior of cell (f_{sync})is input to this pin.
ck	Input	function driving the clock pin. (f_{clk})
s00	Input	asynchronous behavior resulting in Q=0, QB=0 (f_{00})
s01	Input	asynchronous behavior resulting in Q=0, QB=1 (f_{01})
s10	Input	asynchronous behavior resulting in Q=1, QB=0 (f_{10})
s11	Input	asynchronous behavior resulting in Q=1, QB=1 (f_{11})
Q	Output	output function 1
QB	Output	output function 2

TABLE 2

COMMONLY USED SEQUENTIAL GATES

Cell	f_{sync}	f_{clk}	f_{00}	f_{01}	f_{10}	f_{11}
D flop	D	CK	0	0	0	0
JK flop	$J.\neg Q + \neg K.Q$	CK	0	0	0	0
RS flop	D	CK	R.S	$R.\neg S$	$\neg R.S$	0
Gated clock RS flop	D	E.CK	0	$\neg R.S$	$R.\neg S$	$\neg R.\neg S$
D Latch	0	0	0	$G\neg D$	G.D	0

A sequential cell has at most two outputs Q and QB. If Q and QB are opposite to each other (e.g. flip-flops with no asynchronous behaviour, D-latches etc.), we say that Q and QB are "related". When Q and QB are not opposite to each other, we say that the two outputs are "unrelated".

A1. Assumption: The asynchronous functions are pairwise disjoint.

A crucial assumption we make in our generic model is that for a given input stimulus, atmost one of the four asynchronous functions is equal to 1. This assumption is valid because none of the outputs are ever driven to 0 and 1 at the same time. Using assumption A1, we can make the assertions: $f_{00}.f_{01} = 0$, $f_{01}.f_{10} = 0$ and so on.

We begin the discussion of the formulation with the introduction of the "plus" ($^+$)operator. The "plus" operator is used to represent the value of a variable or a function at an instant that is just after the present time. To understand this new operator, we state some of its properties. Let f be a function of n input variables $<x_1, x_2, ..., x_n>$.

P1. If f is a constant valued function i.e f is either a tautology or the zero function , then $f^+ = f$

P2. $f^+(x_1, x_2, x_3, .., x_n) = f(x_1^+, x_2^+, x_3^+, ..., x_n^+)$.

P3. $\neg(f^+(x_1, x_2, x_3, .., x_n)) = (\neg f(x_1, x_2, x_3, ..., x_n))^+$

Given a variable x_i whose value is known at time t, x_i^+ is just another variable that denotes the value of x_i at a time $(t + \varepsilon)$, that is just after t. In equation E1 given below, we present a logic function that accurately captures the value of the Q output of a sequential element at a timeε after the present.

$$Q^+ = [(\neg f_{clk}.\ f_{clk}^+.(f_{sync}) + (f_{clk} + \neg f_{clk}^+).Q)\ \neg f_{00}.\neg f_{01}] + f_{10} + f_{11} \qquad \text{(E1)}$$

Let us try to express a D flip-flop (Table 2) using this formulation. A D flip-flop exhibits the following behavior: Whenever the clock input (CK) rises from 0 to 1, the output Q is equal to the value at the data pin D. At all other times, the flip-flop stores its "previous" state. The "previous" state of the flip-flop is the value at the Q output of the flip-flop. We can write the equation for Q^+ in the following manner:

$$Q^+ = \neg CK.\ CK^+.\ D + (CK + \neg CK^+).Q \qquad \text{(E2)}$$

Let us now consider a D-latch (Table 2) and try to express its behavior using this new formulation. Note that a latch does not have any synchronous behavior as per our sequential model. Assuming that the latch has a data pin D and an enable pin G, we can write the equation for a latch in the following manner:

$$Q^+ = Q.\neg G + D.G \qquad \text{(E3)}$$

Equation E3 states that the output of a latch is 1 whenever both D and G pins are 1. This depicts the transparent behaviour of a latch. We can also see from E3 that when the enable pin G is 0, the output is the previous state. Note the absence of the clock variables CK, CK^+ in E3.

Now that we have seen two applications of this formulation, we now explain the meaning of E1. Equation E1 has two parts : a part that depicts the synchronous behavior of the sequential cell and a part that depicts the

asynchronous behavior. If either of the asynchronous functions f_{10} or f_{11} is equal to 1, then the value of Q^+ must be equal to 1 regardless of any of the other components of the equation. Similarly if either of f_{01} or f_{00} are equal to 1, the value of Q^+ must be 0. The synchronous behavior is always expressed in relation to a clock edge. If there is a transition in the clock function (f_{clk}) from 0 to 1, we want the output to follow the value of the synchronous functionality (f_{sync}) of the cell. At all other times, Q^+ remains in the "previous" state (Q).

In an analogous manner, we can write the equation for QB output of the cell :

$$QB^+ = [(\neg f_{clk}.\ f_{clk}^+.(\neg f_{sync}) + (f_{clk} + \neg f_{clk}^+).QB)\neg f_{00}.\neg f_{10}] + f_{01} + f_{11} \quad (E4)$$

Lemma 1: If $f_{00} = 0$ and $f_{11} = 0$ then $Q^+ = \neg QB^+$

VI. Matching of Sequential Elements

Our method for matching sequential elements has two phases - modeling and matching. In the modeling phase, we process the sequential gates in the target technology to store pseudo-canonical forms for them. In the matching phase, we identify the presence of these sequential gates in the subject graph and find the match with the best cost. We first present the modeling phase in sub-section *A*. The matching phase is presented in sub-section *B*.

A. Modeling of sequential elements

In this section, we present the algorithm `SEQ_model` which models a sequential cell. The algorithm is applied to every sequential gate in the library and the pseudo-canonical forms that are generated are stored for subsequent use in the matching phase.

Definition: We say that a phase assignment Φ of a sequential cell is *complete* if and only if the phase for each input of the cell has been determined. The definition is also extended to permutations π.

We define Φ to be the phase assignment and π to be the permutation of all inputs of the cell. If f is an output (Q or QB) of the cell then computing Φ_f does not necessarily mean that Φ is *complete*, because the support of the sequential cell might be larger than Ω(f). Let PHASE(f) and PERMUTE(f) represent the phase and permutation canonicalization procedures that are applied to function f to obtain a set S of functions h that represent the pseudo-canonical forms (PCF's) of the function f. The "effectiveness" of the procedures PHASE and PERMUTE is measured by the cardinality of the set S. Associated with each function h in S is a permutation π_h and a phase assignment Φ_h, such that applying Φ_h and π_h to f produces h. We shall present a brief overview of the procedures PHASE and PERMUTE later in this section.

```
algorithm SEQ_model(T, cell) {
/* T - a search table to store the PCFs */
Initialize Φ and π to any phase and permutation.
Q = first_output(cell) /* Equation E1 */
QN = second_output(cell) /* Equation E4 */
process_seq(Q, QN, T, cell)
if (Q and QN are not related) {
  process_seq(QN, Q, T, cell)
}
} end_algorithm;

procedure process_seq(f, fn, T, cell) {
/* f is the primary function, fn is the secondary function
*/
Apply PHASE(f), PERMUTE(f)to obtain set S1 = {<h,Φh,πh>}.
foreach <h, Φh,πh> in S1 { /* loop L1 */
  Key1 = h
  Assign Φh and πh to corr.variables in Φ and π
  Compute function t by applying Φ and πto fn
  Key2 = t
  if (Φ or π is not complete){/*some inputs unas
signed*/
  Apply PHASE(t),PERMUTE(t) to obtain S2 =
 {<g,Φg, πg>}

    foreach <g,Φg, πg> in S2 { /* loop L2 */
    Incrementally update Φ and π using Φg, πg
    Key2 = g
    Store <cell, Φ, π> in T with keys <Key1,Key2>
    }
  } else { /* all inputs assigned */
    Store <cell, Φ, π> in T with keys <Key1,Key2>
  }
}
} end_procedure
```

At any point in the algorithm, we assign permutations and phase assignments to only those inputs that are unassigned. For sequential cells that have Q and QB outputs opposite to each other, the procedure `process_seq` is called just once. The canonicalization step on the primary function f is sufficient to find the Φ and π of the entire cell. The inner loop L2 that applies the canonicalization procedures to the secondary output fn does not get exercised. If a sequential cell has outputs Q and QB that are unrelated, procedure `process_seq` is called twice, once for Q and once for QB. The canonicalization procedures, when applied to the secondary function fn, compute phase assignment and permutation for variables that are in the set $\Omega(QB^+) - \Omega(Q^+)$. From E1 and E4 in section VI-A, this set has exactly one member, QB.

In our implementation, we adopted the paradigm of fast pseudo-canonicalization procedures proposed by Mohnke et al. The procedure PHASE tries to assign phases to the variables of a function using cofactor minterm counts (also known as Chow parameters [14]). The complexity of the PHASE procedure is O(V) where V is the number of nodes in the BDD of the function. PERMUTE assigns permutations to the variables using

cofactor minterm counts, hamming distances and variable symmetry. The complexity of the symmetry computation component of the PERMUTE procedure is $O(n^2)$ where n is the number of variables in the support of the function. The variable symmetry heuristic is useful when neither the minterm count nor the hamming distance heuristics are effective. The minterm count heuristic is O(V) and the hamming distance heuristic has a complexiy of $O(n^2.V)$ [12]. For the sake of brevity, we use ψ to refer to the complexity of the canonicalization step.

Another component of the complexity of the algorithm `SEQ_model` is the cardinality of the set S1. The set S1 is the set of all PCF's of the primary function f. In theory, the cardinality of this set is $n!2^n$ provided PERMUTE and PHASE are completely "ineffective" for the function f. In practice, we observed that the largest number of PCF's generated (over a set of 5 ASIC vendor libraries) was 16 for a sequential cell that had 11 inputs. Assuming M to be the cardinality of the set S1, the complexity of the algorithm for the related outputs case is $O(\psi + M)$. The complexity of the algorithm for the unrelated outputs case is $O(\psi.(M + 1))$, because the procedures PERMUTE and PHASE need to be executed once for each element in S1 to canonicalize the secondary function fn. In the complexity analysis presented above, we have ignored the cost of building the BDD of the function.

B. Matching of sequential elements

After the sequential gates in the library are processed and the PCF's have been computed, we examine the logic network to perform technology mapping of sequential elements. In order to simplify the covering step, we examine each sequential element in the network completely independently of the others. For each sequential element, we find the match with the best cost and implement it before moving on to the next sequential cell.

We assume every sequential element that is to be mapped is represented in terms of the generic sequential cell GEN, which was introduced in section V. For every GEN cell in the network, we use a DAG enumeration scheme to find all relevant DAGs that are rooted at the GEN cell. We then use the algorithm `SEQ_match` (presented below) to find all matches in the target library that can replace the GEN cell and surrounding combinational logic. These matches are then evaluated using a cost mechanism (e.g area and/or delay) and the best one is chosen.

```
algorithm SEQ_match(N, T) {
/* N is the sub-network to be matched */
/* T is the search table from SEQ_model */

Pick any output of the GEN cell as a primary
root R0.
Let the other root be R1
Compute function fRo for root Ro from N
Initialise ΦN, πN
Apply PERMUTE(fRo) and PHASE(fRo) to obtain S1 =
{<h, Φh, πh>}
Pick any h from S1
Update ΦN, πN with Φh, πh
Key1 = h
Compute function fR1 for R1
Apply ΦN, πN to fR1 to obtain t
Key2 = t
if (ΦN, πN are not complete) {
  Apply PHASE(t), PERMUTE(t) to get S2 = {<g, Φg, πg>}
  Pick any g in S2
  Update ΦN, πN with Φg, πg
  Key2 = g
}
Search T, with key <Key1,Key2> to get match set R
return (R).
} end_algorithm
```

In the algorithm `SEQ_match`, we pick any one PCF for the output functions, because the execution of `SEQ_model` has already computed all PCFs in the search table T. It is more efficient from a run time perspective to do the PCF enumeration once (at the library load time) and keep the matching step as fast as possible. The complexity analysis of the algorithm `SEQ_match` is very similar to the analysis presented for `SEQ_model` and is omitted here.

VII. Experimental results

In order to understand the effectiveness of the proposed technique we ran a set of experiments. Experiment A reflects the memory efficiency of our technique by enumerating the PCFs for sequential cells in a standard cell library. Experiment B demonstrates some transformations that are possible with the new technique. Experiment C compares three sequential mapping techniques: sd_map (simple D-type elements), rb_map (rule based) and bm_map(boolean matching based). We decided not to compare our results with the SIS system because the SIS timing model is not as accurate as the one used in our mapper. In addition, it is not possible to model complex sequential cells in SIS.

A. Standard cell library evaluation

For all our experiments, we used a popular ASIC standard cell library L. This library has 18 different types of sequential elements. Each sequential element type has two drive strengths with different delay and area characteristics. Table 3 presents some simple statistics for the library L. The column C1 reflects the number of sequential element types (a cell and its higher drives are all counted as one type since the function is the same). The column C2 reflects the number of PCF's that are stored for the corresponding type. On an average, we store 2.6 PCF's for each sequential element. The time taken to compute the PCFs for library L is 3 seconds on a SUN sparc 2.

TABLE 3

PCF'S FOR SEQUENTIAL CELLS IN L

Types	C1	C2
flip-flops	11	25
latches	7	22
Total	18	47

B. Examples to illustrate the effect of boolean matching

We present three simple examples C1, C2, C3 (Fig 2). Each example demonstrates a capability which can be achieved with our boolean technique but cannot be done easily with a structural technique The GEN model representation of 5 gates that appear in these three examples are given in Table 4.

TABLE 4

SEQUENTIAL GATES USED IN EXPERIMENT B

Cell	f_{sync}	f_{clk}	f_{00}	f_{01}	f_{10}	f_{11}
FD1	D	CK	0	0	0	0
FJK1	J.¬Q + ¬K.Q	CK	0	0	0	0
FD2	D	CK	0	¬CD	0	0
FD4	D	CK	0	0	¬SD	0
LD1	0	0	0	G.¬D	G.D	0

For each circuit, we also contrast the area and delay for the boolean matching scheme (bm) with a simple mapping scheme where only D flip-flops/D latches are instantiated (sd) (Table 5).

TABLE 5

AREA/DELAY FOR EXPERIMENT B

Circuit	sd area	sd delay	bm area	bm delay
C1	7	1.37	5	1.30
C2	15	4.38	11	1.69
C3	10	1.67	8	1.79

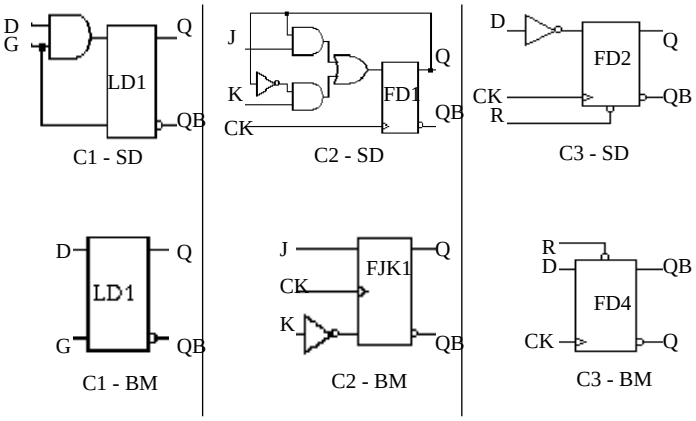

Fig 2

C. Experiments on large circuits

We implemented the ideas described in this paper on top of an existing logic synthesis system. In order to evaluate the effectiveness of the new technique, we ran three different sequential mapping techniques on a set of 10 benchmark circuits from the ISCAS 89 benchmark suite.

1. sd_map - simple D flip-flops only (i.e no sequential mapping)
2. rb_map - a rule-based technique (based on the ideas presented in [5])
3. bm_map - the new boolean matching scheme

For each circuit, we used two scripts - one for area optimization and one for delay optimization. For the rule-based technique, the rule base was composed by examining the sequential cells in 20 different ASIC vendor libraries. All other parameters in the experiment were unchanged. The results are presented in Tables 6 and 7.

TABLE 6

TABLE OF BENCHMARK RESULTS (Area)

	Area Optimization		
Circuit	sd_map	rb_map	bm_map
s27	32	30	30
s298	181	181	181
s444	264	262	260
s1423	957	953	956
s1488	550	550	550
s5378	2274	2268	2266
s9234	1898	1873	1870
s13207	6431	6396	6366
s15850	6419	6313	6309
s35932	20239	19526	19247

TABLE 7

TABLE OF BENCHMARK RESULTS (Delay)

	Delay Optimization		
Circuit	sd_map	rb_map	bm_map
s27	4.2	4.1	4.1
s298	6.71	6.4	6.2
s444	7.0	6.8	6.9
s1423	13.5	13.4	13.2
s1488	8.6	8.2	8.2
s5378	8.4	8.3	8.3
s9234	10.9	10.9	10.8
s13207	14.6	14.6	14.7
s15850	17.68	17.58	15.7
s35932	8.5	8.5	8.4

We can see that sd_map performs worse than rb_map and bm_map in all examples either in area or in delay. This establishes that a sequential mapping technique that can infer more complex flops than just D flip-

flops is beneficial as it can improve both the area and the delay of the circuit. The technique presented in this paper outperforms the rule-based scheme in 6 out of 10 examples for area optimization, and in 5 out of 10 examples for delay optimization. All experiments were carried out on a SUN Sparc 2 with 16MB of memory. The memory consumption of bm_map is 10 - 20% lesser than the rb_map. The CPU times are comparable to rb_map.

We used examples from the ISCAS 89 benchmark suite so as to establish a common reference point. The new technique shows substantial improvements in several large commercial designs over the rule-based and the simple mapping schemes. In these designs, complex sequential cells are matched causing a reduction in area as well as delay. Table 8 shows area and delay improvements for a set of 4 commercial designs. The numbers have been normalized to show the improvement.

TABLE 8

TABLE OF RESULTS FOR COMMERCIAL DESIGNS

Circuit	Tech.	rb - area	bm-area	rb-delay	bm-delay
ex1	ACTEL	1.0	0.72	1.0	0.9
ex2	LUT	1.0	0.9	1.0	0.81
ex3	CMOS	1.0	0.92	1.0	0.98
ex4	CMOS	1.0	1.02	1.0	0.86

VIII. Future work

An interesting extension of the work presented in this paper is to utilise dont-care information while performing the matching step. Consider the Fig 3 shown below. In 3 (i), we have a D flip-flop being driven by a multiplexor that chooses between the previous state Q and the new data D using the signal EN. In configuration 3 (ii), we have a D flip-flop with the clock CK gated using signal EN. The configurations (i) and (ii) are logically equivalent if we are guaranteed that E does not go from 0 to 1 when the clock signal CK is 1. The transformation from (i) to (ii) is desirable if we want to optimize the circuit for low dynamic power consumption.

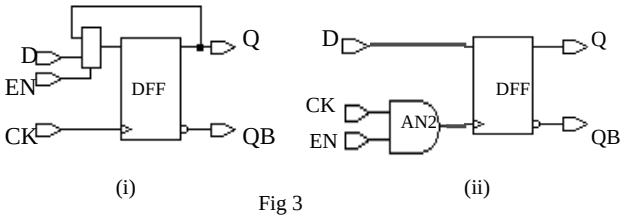

Fig 3

Another area for future research is the inference of Master Slave elements. The technique presented in this paper can map the Master latch and the Slave latch independently. However it is unable to treat them together as one element because the present sequential model is limited by the fact that there is only one clock that feeds the sequential cell. In order to support Master-Slave latches and multiple clock sequential cells, we need to extend the model further.

References

[1] K. Keutzer, "DAGON : Technology Binding and Local Optimization by DAG matching", *24th Design Automation Conference*, pp. 341 - 347, 1987.

[2] A.V. Aho, M. Ganapathi, "Efficient Tree Pattern Matching: an Aid to Code Generation", *Symposium on Principles of Programming Languages*, pp 334 - 340. ACM, January 1985.

[3] E. Detjens, G.Gannot, R.Rudell, A. Sangiovanni-Vincentelli, A. Wang, "Technology Mapping in MIS", *International Conference on Computer Aided Design*, pp. 116 - 119, 1987.

[4] R. Rudell, "Logic Synthesis for VLSI Design", *Ph. D Thesis, U.C. Berkeley Memorandum UCB/ERL M89/49*, Apr. 1989.

[5] D. Gregory, K. Bartlett, A. de Geus, G. Hachtel. "Socrates: a system for automatically synthesizing and optimizing combinational logic", *23rd Design Automation Conference*, pp. 79 -85, 1986.

[6] C. Moon, B. Lin, H. Savoj, R. Brayton, "Technology Mapping for Sequential Logic Synthesis", Proceedings of Intl. Workshop on Logic Synthesis, North Carolina, May 1989.

[7] P. Moceyunas - Personal Communication.

[8] S. Muroga, Threshold Logic and its Applications. John Wiley, 1971.

[9] F. Mailhot, G. De Micheli, "Technology Mapping using Boolean Matching and Don't Care Sets", *1st European Design Automation Conference*, pp. 212 - 216, 1990.

[10] F.Mailhot, G. De Micheli, "Algorithms for Technology Mapping Based on Binary Decision Diagrams and on Boolean Operations", *IEEE Trans. on Computer Aided Design of Integrated Circuits and Systems*, Vol 12, No.5 pp. 599 - 620, 1993.

[11] J.R. Burch, D.E. Long, "Efficient Boolean Function Matching", *International Conference on Computer Aided Design*, pp. 408 - 411, 1992.

[12] J. Mohnke, S. Malik, " Permutation and Phase Independent Boolean Comparison", Proceedings of *EDAC*, 1993.

[13] D.S. Kung, R.F. Damiano, T.A. Nix, D.J. Geiger, "BDDMAP: a technology mapper based on a new covering algorithm", 29th *Design Automation Conference*, pp. 484 - 487, 1992.

[14] S.L. Hurst, D.M. Miller, J.C. Muzio, Spectral Techniques in Digital Logic. Academic Press, 1985.