Automated technology mapping

by J. L. Gilkinson
S. D. Lewis
B. B. Winter
A. Hekmatpour

Logic "mapping," or "transformation," refers to the process of converting a logic design from one form of specification to another. The output is usually a specific technology implementation and the input could range from a previous technology implementation to a high-level design language. Motivated initially by the problem of test case generation for new technologies, a logic transformation system, known as the Technology Mapping System (TMS), was developed. This system has focused on the problem of technology-to-technology mapping involving gate array or standard cell logic families. TMS makes use of an intermediate notation, called GLN, and uses several forms of "rules" to control the mapping process. This paper discusses the history and general operation of TMS, and makes a comparison of transformations from different types of sources.

Introduction

Logic design is becoming increasingly difficult. There are more technologies to understand, more circuits per machine, more circuits per chip, longer manufacturing turnaround times, shorter development schedules, and more tests for a limited design force to perform. The resulting problems in productivity and quality are met with an increasing number of logical and physical design automation tools. Relieving the designer of the detailed implementation decisions is the job of a logic transformation system, a relatively recent addition to the DA field. This tool attacks the productivity problem by either allowing a more efficient source specification, such as a technology-independent language, or by directly utilizing a previous, technology-specific, implementation.

The terms *mapping*, *remapping*, *transformation*, and *synthesis* have been used to describe a variety of conversions of logic designs from one form to another, the target usually being an implementation in a specific circuit technology. There are four general types of design specifications that are relevant:

1. High-level hardware design languages.
2. Array or truth table specifications such as Programmable Logic Arrays (PLAs).
3. Technology-independent structures.
4. Technology-dependent structures such as the technology-specific Basic Design Language for Structure (BDL/S) used by the IBM Engineering Design System (EDS).

The term *synthesis* is usually used with type 1, while *remapping* is often applied to type 4.

There have been many efforts at logic transformation reported in the literature [1]. One of the earliest efforts within IBM was the ALERT system in the late 1960s [2, 3]. This system showed that a logic design could be automatically generated from a design language, but the results were not competitive with manual designs. More recently, the Logic Synthesis System (LSS) was developed [4–6]. This system has brought design language synthesis of random logic from a promising idea to a practical reality. LSS has also been used for the remapping application.

Within the IBM Engineering Design System, the Logic Transformation System (LTS) has been developed to provide the corporation with a general-purpose transformation system [7]. A system that is used to automatically generate two-level logic, such as PLAs, from a high-level design language is the Interactive Design Language (IDL) system [8]. In Japan, the LORES system [9] was developed to do remapping applications, and ANGEL [10, 11] specializes in design language synthesis. This paper describes a system called TMS, or the Technology Mapping System, that has been involved in all four types of transformations, with the primary emphasis on technology-to-technology conversions (type 4).

Background

TMS was not started with the usual motivation of designer productivity; rather, it was intended to support new circuit technology development. A way was needed to generate examples and test cases easily. The scale of integration had reached a point where the generation of design data to describe a single chip demanded a significant amount of manpower.

The manual mapping of an existing microprocessor into a new, denser, 1200-gate circuit technology introduced us to technology mapping in 1976. We found that the time required to do a mapping project manually is not much less than the time required for an entirely new design. The mechanical generation of technology-dependent BDL/S is a big job because of the high data volume and the tendency to introduce errors. Simple transcription mistakes can take a long time to find or go undetected prior to manufacturing. We also found that the only true definition of a design is the implementation data, the original BDL/S. Other documentation does not cover everything and the original designers do not remember everything. For example, a condition that the designer may consider as a "don't care" might be tested by a diagnostic program that he was not even aware existed. Using the design data as the source, a complete understanding of the logic is not necessary for the majority of the labor involved.

In view of other similar mapping projects in 1978, the decision was made to experiment with automating part of the mapping job. At the time there were no logic transformation tools available. The original plan was based on a programmed conversion of each source block to target blocks on a one-to-one or simple one-to-several basis, followed by manual modifications. After the initial objectives were completed, a review of the manual procedures was started, and over the following years the programs grew and evolved. Started as a simple aid to mapping, TMS was expanded to automate the entire process.

It wasn't until a second target technology was considered that a cohesive methodology emerged which included an intermediate notation and a concept of *rules*. Seventeen sources and eight targets were added gradually as new technologies and applications developed. The technologies include both FET and bipolar circuits of the gate array (masterslice) or standard cell (masterimage) type, ranging from 100 to 10 000 equivalent gates per chip. In addition, several PLAs have been transformed into random logic.

When the ability to map from one technology into another was established, we became more involved with product design applications. For new designs a technology-independent form of design specification called TIF (Technology-Independent Functions) was developed. Expressed in BDL/S, TIF is a set of block functions that includes primitives such as AND and OR, and high-level functions such as registers, selectors, comparators, decoders, and user-defined macros. A mix of TIF and target technology macros is also an option as a source specification. Designers find TIF relatively easy to work with because there are no concerns about fan-in or fan-out limits and the same block set is used for all technologies. A similar concept is the generic operators used in the EDS/DAV system [12]. TIF can be used as a target as well as a source. Often a designer wants to convert a design from an old technology into a new one and wants to alter the function slightly. A convenient way to do this is to first map into TIF, make the design changes in TIF, and then map into the target technology. TIF has also been used as an intermediate form during synthesis from a hardware description language. An initial structure in TIF is generated from a compiler that reads the source language.

Overall, TMS is used for five types of applications:

- Product design.
- Feasibility studies.
- Technology comparisons.
- Alternative-technology backup plans.
- Vehicles for design system evaluations.

The technology-independent design of chips for actual products attracts the most attention from tool developers and design groups. However, in the role of a technology support group, the last four items constitute the majority of applications. Estimating circuit counts in various technologies is an important capability when considering future technologies or potential business cases. In the past, intuitive considerations of circuit efficiency were often substituted for hard data, and estimates varied widely. Deciding to use a PLA versus random logic is an example of a difficult task where logic mapping programs can help [13].

Overview of TMS

To reduce the amount of code required by the various combinations of sources and targets, the TMS transformation process is divided into two phases. In the first

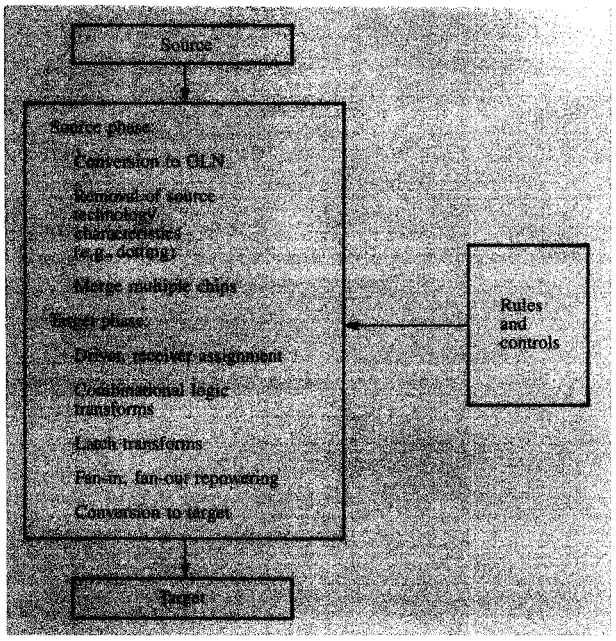

Figure 1

General TMS transformation process.

or *source* phase, the design is transformed into an intermediate notation called GLN or General Logic Notation, and the technology-dependent characteristics of the source technology are removed. In the second or *target* phase, the design is manipulated until the GLN description matches one-to-one with the target block set. The general transformation flow is shown in **Figure 1**.

GLN is a key concept in TMS. Its purpose is to provide a description of the design that is easy for programs to manipulate. Since the conversion process manipulates the GLN until it is one-to-one with the target, all of the target's basic functions should be represented in GLN. The functions include AND, OR, AND-OR, OR-AND, EXCLUSIVE-OR, parity function, and a variety of latches, clock drivers, off-chip drivers, and receivers. The LSSD [14] latch functions include the SRL, L1, L2, and L3 with polarity-hold, glitchless polarity-hold, and set/reset construction types. In addition, the GLN blocks must be able to take on technology characteristics such as high-power and low-power versions of a function.

There are several aspects to the conversion of a source design into GLN. In technology-to-technology mapping, the mapping system must handle a wide variety of notation and design conventions. Since the purpose of GLN is to have a standard form for each function, all of the unique technology or designer representation quirks must be removed, while the logic is being translated into the standard GLN form. All purely physical information is removed and the notation for model input and output signals is standardized.

One of the common complications in the source phase is "dotting," also known as the "wired-or" function. The transformation system must cope with a mix of implicit or explicit representations of dotted nets in addition to multi-block functions such as "extend ports" on latches. GLN does not allow multi-source nets, so all configurations that involve dots must be changed into simple equivalent functions such as AND or converted into larger macros such as multi-port latches. As shown later, the GLN restriction on multi-source nets provides a simplification for programming and does not preclude their existence in the target design.

All blocks in GLN are *dual-rail*—both true and complement signals are available. As a result, there are fewer blocks for the programs to manage and there is no searching for an inverter to reach a complementary signal. In the source transformation phase all inverters are removed except at primary inputs. Since there are no fan-out limits in GLN, all repowered signals or duplicated functions are consolidated into one source for each logical signal.

There are five parts to the target phase of the conversion process:

1. Off-chip driver/receiver assignment.
2. Combinational logic transformations.
3. Latch transformations.
4. Fan-in/fan-out repowering.
5. Conversion of GLN to the target.

Off-chip driver and receiver assignment involves either adding new blocks or maintaining existing ones. Most technology-to-technology mapping involves a merging of several smaller chips into a single, denser chip. In this case, the original driver and receiver functions of each chip are converted to GLN-equivalent functions and new drivers and receivers are placed at the new input and output pins of the merged design.

Combinational logic transforms reduce cell count or connection count, or improve delay. Without them, technology mapping would simply include block-to-block translations which would leave unnecessary and trivial logic in the new design. Removing the obvious types of circuit overhead, such as double inversions, is the initial goal of the logic reduction. As the number or sophistication of these programs is increased, mapping very dissimilar technologies becomes realistic. The programs look at small logic functions and transform them into configurations that will be more efficient in the target technology. A designer would do these types of changes by inspection; they would typically be implemented without writing out the Boolean equations or applying any classical minimization. The term *local*

transformations has been used to describe this type of logic reduction technique [5].

Although the reduction programs can be run independently and in different sequences, there can be interactions between them. One transformation may put the logic into a configuration which will make a following transformation more effective. In addition, the transformations may be iterative. They will be repeated several times in a row, and run at different times within the same transformation job.

Latch transformations are generally alterations of the form of a latch to make it one-to-one compatible with something that exists in the target technology. TMS does not alter the number of latches in the design. Some transforms are done to reduce cell count or improve delay, but there is not much that can be done to alter the cell count that is attributed to each latch. Changing the number of data ports is an example of a latch transformation. Since technologies vary in their abilities to support multi-port latches, the transformation system must be able to read in multi-port latches and output single-port latches. Latches are reduced to the minimum number of ports in the source phase, and if multi-port latches are available in the target, they will be created or recreated in the target phase. Increasing the number of ports improves the path delay into the latch but also increases the loading on the clock signals. The inversion of a latch is a transformation that can save extra cells by reducing the number of surrounding inverters, as shown in **Figure 2**. Other latch transforms include splitting SRLs into component L1 and L2 latches and the inverse transform of combining L1s and L2s to form SRLs. Clock drivers that are typically used by bipolar technologies must be added or deleted and the latch input signal polarities must be adjusted to match the target requirements.

During fan-in repowering, new blocks are added to reduce the fan-in of each block to the limits accepted by the target technology. The fan-out repowering phase is similar, but more involved. Fan-out violations are fixed by increasing the power level of blocks or by adding blocks in parallel or series. Fan-out repowering also converts dual-rail outputs to single-rail if required by the target technology.

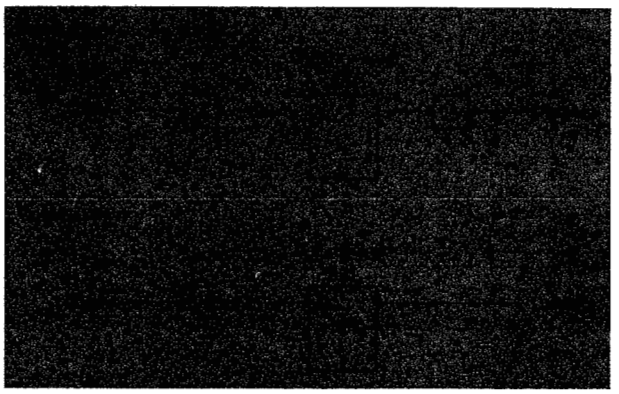

Figure 2

Example of a latch inversion transformation: D = data input to SRL; N = inverter.

Rules and programs

The need for a structured approach became evident after working with a second target technology. A bug in a program that dealt with the first technology would be likely to reappear in the corresponding program for the second technology. This duplication of basic functions in programs for each technology created a maintenance problem, and it made the addition of new target technologies very time-consuming.

The classical solution for program maintenance in IBM design systems is a *rules-driven* approach. Conceptually, *rules* can refer to any structured way of modifying the function of a program. The form of a rule can be as simple as column-oriented data, or as complicated as a special source language that is compiled to become part of a program. No matter what format is used, a *rule* is distinguished by being easier to modify, and is usually more restricted in function, than a more general *program*. Ideally, the additions and changes to the mapping system should be focused on the rules rather than the programs. Our effort to create rules was successful in that most of the programs in TMS can be used to map to any of the target technologies. However, the concept of rules in TMS does not have the same rigor that is usually associated with rules in the IBM Engineering Design System (EDS).

TMS rules take four different forms:

- Action parameters.
- Tables.
- Macros (block transformation rules).
- "Twist" rules.

Action parameters have several uses. They are used to control the mode of a program or establish criteria for some type of action to take place. For example, the primary goal of an action could be the reduction of cells, connections, or delay, as selected by the choice of parameters. In addition, parameters may be used to control global values of technology constraints, such as fan-in or fan-out limits when they are not specified in a table. These are considered to be rules because they can be a way of providing target technology information to a program.

The tables contain the detailed information that must be distinguished by block type and/or pin type. For example, the limits used in the final fan-in and fan-out repowering

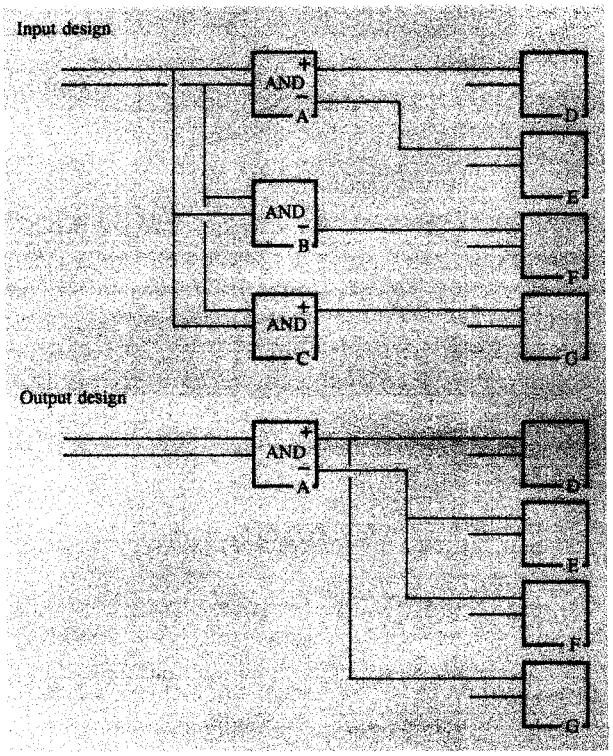

Figure 3

Example of parallel reduction.

transforms are contained in tables. Data within the table refer directly to the GLN blocks and not to the target technology blocks, which means that the rules coder must know the final GLN-to-target transformation when he writes the rules.

Macros are used to expand a single block into multiple blocks. For example, a parity function block may be expanded into a tree of EXCLUSIVE-ORs. These macros are implemented as BTRs (Block Transformation Rules), which are a rule type used by EDS.

The "twist" rule is the glue that keeps the system together. This set of rules guides block-to-block and pin-to-pin transformations and is used to call up a macro (BTR) expansion. Primarily, this type of rule is used to control the conversion to and from GLN. Twist rules are the only rules that undergo a compiling process; they are read by a preprocessing program and converted into PL/I subroutines. The rule syntax is designed to make it the easiest of all the rules to deal with, giving the coder an easy way to test the details of a block. Today, most of the additions and/or changes to TMS are made in twist rules rather than in the other rule formats or the programs themselves.

In addition to the rules used in the mapping process, there is a separate set of rules used in the verification process. To validate the functional equivalence of the transformations, the EDS Static Analysis System (SAS) [15] is used. The purpose of the TMS validation rules is to provide SAS with a common equivalent logic model for latches. Unlike combinational logic, latches of equivalent function can have construction differences that make a Boolean comparison difficult. For combinational logic the normal simulation rules are used to generate the equivalent logic expansion. The use of SAS is vital to maintaining the productivity and quality of the mapping tool.

Controlling the transformation process

Each pair of source and target technologies requires a strategy that includes decisions on what to expand into lower-level GLN blocks and what to leave alone. Making these decisions requires an understanding of the reduction steps that will be most effective in the transformation. For example, TMS cannot convert a tree of EXCLUSIVE-ORs or NANDs into a parity function block. Therefore, if the source and target have similar parity functions as single blocks, the parity function block will not be expanded into another form. If it is to be expanded, there may be more than one alternative to consider. For a parity function, the best implementation in the target may be EXCLUSIVE-ORs, NANDs forming EXCLUSIVE-OR functions, three input NANDs, or some other special configuration of target technology blocks. There is also the decision of where in the job sequence to do the expansion. If the expanded blocks can undergo some logic reduction, the expansion is done early; otherwise it is left to the final transformation step.

The overall conversion strategy is implemented with three levels of control mechanisms. First, the basic framework is set by the choice of "twist" rules and macro expansions. Second, the manipulation of the GLN is controlled by the selection and order of the actions. Third, the individual actions are controlled by the rules and parameters. Because all of the logic reduction is done in GLN, the rules must provide essential target technology data in terms of the GLN blocks. The purpose of the GLN is to give the programs an easy form to manipulate but not to preclude transformations that have technology-dependent criteria. Programs used for logic reduction have varying degrees of control or technology data input, but can be divided into three general categories:

- Programs without control.
- Programs with control.
- Programs with control and scoring.

Programs without control are completely technology-independent, running the same way every time. An example of this type of action is parallel reduction, as shown in **Figure 3**. The nature of these algorithms is such that they

always have a desirable effect. Other reduction programs are not as simple, requiring additional controls to help guide the algorithm.

An example of how the strategy, rules, and programs interrelate is the problem of forming the AND-OR function (or its dual, the OR-AND). Implementations of AND-OR functions vary widely among technologies, often involving a wired-or or dotting configuration. One way of dealing with this technology capability is to convert the source to the target technology, followed by applying special technology-dependent programs which could find the AND-OR function or utilize the dotting capability of the target. Initial attempts were along those lines of thought. We found that the algorithm to find an AND-OR function was basically the same, regardless of the target technology. The primary variables were the dimensions of the AND-OR block (fan-in, fan-out, number of ANDs) and the form of the target technology implementation

The solution to the AND-OR formation problem was to create a program that created AND-OR blocks from configurations of GLN blocks. The program operation is guided by action parameters which set the maximum dimensions that the AND-OR blocks can take, based on limitations in the target technology. The AND-OR block is another GLN primitive, so nothing is lost as far as being able to do further manipulation on this block. The fan-in and fan-out programs always see this function as a single block since GLN does not allow multi-source nets. At the time of final transformation, the AND-OR block is converted to its final form. If that final form involves a dotted net, a macro is used to expand the AND-OR into its proper multi-block representation.

The AND-OR function is an example of a block that is treated differently than the parity function in the overall mapping strategy. In the source phase the original AND-OR (or OR-AND) blocks are usually expanded into lower-level functions whether or not the target has similar blocks available. The reason is that the AND-OR formation program generally does as good a job or better of finding the AND-OR functions than the original designer.

Only the "common term" reduction program deals with a "scoring" function in a general way. This action identifies common inputs in a swappable pin group and creates a new block whose inputs are the common signals, as shown in **Figure 4**. The general steps in the common-term process could apply to any local transform algorithm:

1. Find candidates for the local transform.
2. Evaluate the net change that would result if the transform were actually completed. This change is computed as the "score." In the case of TMS, we use two scores: 1) cells (or circuits) and 2) connections.
3. Examine the input criteria. Input parameters are used to specify limits on the cell and connection scores that are considered satisfactory. All candidates that do not meet the input criteria are thrown out.
4. Rank the acceptable candidates in descending order by score.
5. Select the candidates one at a time until no more transforms can be done. The entire list may not be used because one candidate may share some blocks with other candidates. In one pass of the reduction cycle, a block cannot be altered twice. After completion of all the changes that can be made in one pass, the whole process is repeated on the altered design. This process continues until no more candidates exist.

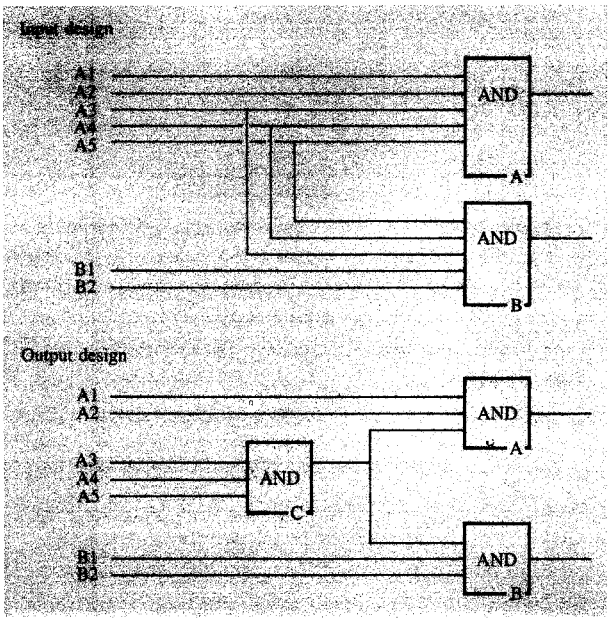

Figure 4

Example of common-term reduction.

The score is a projection of the change in final cost function created by the transform in the *target* technology. The word *target* is emphasized because the decisions that are to be made are very technology-dependent. Since the GLN design description is not one-to-one with target technology at the time the program is run, the score must be based on more than a simple block count. The changes in GLN must be translated to projected changes that will occur after fan-in, fan-out repowering and final transformation. The required technology-dependent data come from a rule in the form of a table. This rule indicates the limits on fan-in and fan-out, and the cell and connection counts for each block.

There are eight subtotals that make up the total score:

	Cells	*Connections*
Base count	X1	Y1
Inverter penalty	X2	Y2
Fan-out penalty	X3	Y3
Fan-in penalty	X4	Y4
Totals	X	Y

The "base" is a score based on the target technology assuming that both polarities are available and that there are no fan-in or fan-out limits. The inverter penalty refers to additional circuits required because the target does not always have both polarities available. For example, a noninverting AND function takes an additional circuit in a pure NAND technology. Similarly, the fan-in and fan-out penalties measure the circuits required to abide by the finite technology limits. These penalties are particularly important in common-term reduction because the actual reduction is only in these penalty counts. The base cell count increases because the transformation adds an AND block.

Operational characteristics

The transformation process is a series of steps that are controlled by the person setting up the job and by the rules information. Even though there is a general set of process phases and rules, job setup requires familiarity with TMS, and the source and target technologies. The tool is complex enough that we do not burden the individual designers with the task of learning how it works. Normally the job does not depend on the design itself unless the chip is very full or there is a special requirement involved. In addition to setting up the job according to the designer's requirements, the TMS support person generally runs the job and reviews the results. Sometimes reruns are executed by the designer. A key element in achieving a quality mapping is good communication with the designers so that all assumptions, limitations, and special requirements are established early.

TMS is run strictly as a background job. The programs are coded in PL/I and there is one main TMS procedure that is controlled by action statements and control variables. All macro expansions are done with a standard EDS procedure, so a typical run is a mix of TMS and EDS job steps. Generally, the JCL refers to the actions desired, but the detailed parameters and rules datasets are selected by a default-setting program which examines the technology name and action name. All parameters and rules can be overidden to customize a run.

TMS does not have an underlying database or special program interface to use when writing transforms in the same way that LSS [5] or LTS [7] does. Writing new transforms in TMS is not particularly easy, at least for tasks that are beyond the scope of the twist and macro rules. TMS is characterized by having a relatively small set of versatile transforms that are controlled by rules or action parameters. Because we have to deal with many technologies, there is a strong motivation to reduce or eliminate the writing of special transformations for each technology. Keeping the design in GLN as long as possible during the transformation has helped generalize the individual actions. All eight target technologies, so far, share the same transforms, but are supported by different rules and different action sequences. To go beyond what has been done with TMS, however, it is clear that more technology-dependent transformations are required and more underlying program support for the creation of new transformations would be desirable.

One analysis feature that was added to TMS was a statistics program that can estimate a final circuit count even when the design is described in GLN. This program aids the analysis of intermediate results. The effect of the logic reduction is more evident when the statistics are taken at different places during the run. Estimates of the circuits required for inversions and for fan-in and fan-out repowering are made following the same general method described for scoring in the common-term reduction action. In addition to taking intermediate statistics, the GLN model can be saved in the form of BDL/S for inspection or further analysis.

Limitations of TMS

TMS provides a mechanical translation from one logic design structure to another. However, the general technology mapping problem has many aspects, only one of which is generating a mechanically accurate design. It is important to understand those areas that are not accounted for in the conversion process used by TMS:

- *Non-LSSD to LSSD conversion.* Although this type of transformation has often been requested, changing the latch and clocking structure to meet LSSD requirements would be beyond the scope of TMS. The source must be an LSSD design.
- *Partitioning.* The input and target I/Os remain unchanged. Partitioning must be done before or after conversion.
- *Delay.* The logic reduction steps can be oriented toward improving delay, but a final analysis to ensure timing integrity is a separate problem. In addition, the original design must not rely on minimum circuit delay to obtain the proper function other than in simple latch clock drivers. An example of unmappable logic is a clock generator which controls spacing and pulse width with circuit delays.

Also, the rules and programs of TMS have efficiency implications. Some target technology properties that can reduce conversion efficiency are

- Macro-oriented, or higher-function, blocks.
- Blocks with multiple-function outputs.

- A large mix of primitive functions.

Reactions to these properties are difficult to incorporate into general logic reduction programs. In general, TMS is best at "macro-to-micro" conversions, where the source design is described with blocks that are at the same or a higher functional level than the target block set. There are some actions that do form higher function blocks, such as AND-OR formation, but a general micro-to-macro transform would be difficult to make rules-driven, and has not been attempted in TMS. More complex and technology-specific transformations are an option, but these programs can have less return on investment than the ones that can apply to several target technologies. In TMS, we have chosen not to implement several transformations because they did not apply to multiple technologies. Clearly, this is a business decision and not a scientific one.

To get an idea of the relative value of certain technology blocks, it is beneficial to map out of and back into the same technology. Larger macros or special blocks are expanded into equivalent GLN functions and undergo the logic reduction that is available. Analyzing the resulting design versus the source design indicates the penalty of not having more transforms. The source design must be manually generated and hopefully representative of an efficient design.

Comparison of types of transformations

Although the primary function of TMS has been to do technology-to-technology mapping, it has been used in conversions from all four types of source descriptions that were mentioned in the introduction:

- Hardware design languages.
- PLAs.
- Technology-independent structures.
- Technology-dependent structures.

In the type of transformations being addressed, the source description always contains explicit information on the memory elements, or latches, and how they are logically clocked. The transformation maintains the basic latch and clocking structure throughout the process, providing a legal and efficient implementation in a target technology. The mechanics of doing each type of transformation are similar but each has its own characteristics. In the following paragraphs these transformations are discussed and compared. It is important to realize that the advantages and disadvantages pointed out apply only to the mapping process itself and do not indicate which source is better from an overall design methodology point of view.

The conversion from hardware design languages generally represents the greatest change in form and requires the most robust transformation system. A behavioral description is changed into a structural description by using a small set of functions. In the first step, a simple initial structure, without logic reduction, is created by a language compiler. This approach follows a general set of process steps established by the LSS system [4]. Two characteristics of this structure are a relatively large number of blocks, and the existence of constants *one* and *zero*. The constants are removed by propagating them through the logic, and the mapping proceeds in the same way as technology-to-technology mapping. TMS has been used to complete the transformation after the removal of constants.

Managing the delay aspects of the design can be difficult in any transformation, but using a design language as the source appears to aggravate the problem to some extent. The initial structure is usually made without regard to depth of logic, and the designer's coding techniques may force the mapping system to deal with speeding up an initially slow design. A problem that appears to be somewhat unique to the hardware design languages is that of testability. In the LSS system, which has dealt extensively with transformations from languages, there are sophisticated programs that directly attack untestable faults in the target design [16]. These faults are most often put into the design unknowingly by the designer. Our experience with TMS indicates that the other design forms are less susceptible to untestable faults. In coping with either delay or testability problems, the establishment of designer coding practices and/or the addition of special transformations may be required. The LSS system has demonstrated that these problems can be managed successfully [17].

The PLA source designs allow the transformation system to deal with initial structures that always look similar. In the case of TMS, the initial structure is the equivalent logic model that is also used for test pattern generation. This structure is characterized by a high connection to block ratio and a depth of logic that is relatively shallow. Because the initial logic depth is controlled, the PLA source transformation is the easiest to control with regard to delay. Most of the logic reduction is done by the factoring or common-term reduction programs which were mentioned earlier. This type of program can have problems with job run time on PLAs with high connection counts because there are so many candidates to consider. An efficient implementation of the factoring algorithms and ways of limiting the candidates can be important considerations. The reason the TMS version of common-term reduction has scoring, rules, and several control parameters is that it was developed while transforming PLAs.

The easiest of all of the sources to deal with are technology-independent structures that have been manually generated. The input blocks are relatively high-level, so the design can be efficiently expanded into GLN blocks, and all problems with source technology notation such as dotting are eliminated. Conceptually, there could be the same problems of initial logic depth and testability observed with the design languages, but we have not experienced them.

One problem that we have observed deals with "no concern" conditions. For example, a multiplexor can be implemented in different ways depending on the state in which the multiplexor is shut off, or in which no data inputs are selected. If the designer is not concerned about what that state is, it may be possible to choose a more efficient implementation relative to one that is Boolean equivalent with the multiplexor in the initial design. So far, we have left it to the designer to make an efficient choice in his source design. However, in simple configurations it would be possible to find states that could not logically occur, and change the implementation accordingly. This problem is more often encountered in technology-independent designs where the designer would tend to use noninverting blocks, compared to the technologies which favor inverting blocks.

Technology-to-technology mapping has both advantages and disadvantages when compared with the other three types of conversions. The advantage is that the source design tends to be efficient from a delay and logic reduction point of view. If the block sets for the source and target technologies are similar, a reasonable target design can be generated with a few transforms. As was mentioned earlier, a problem that is unique to this type of transformation is handling the myriad of technology and designer notations for similar functions. Another problem is that the input design is often constructed from low-level blocks, such as pure NANDs, but the target offers higher-level functions. When there is no TMS transform that can find the higher-level function, the resulting implementation will be less efficient than desired or the designer will make some changes manually. The ubiquitous parity function is the best example of a complex target function.

There can be problems with the handling of latches in technology-to-technology mapping. Although some latch transformations take place, the basic latch implementation remains constant, including the number of clocks and master resets. Since each technology has a finite latch offering, it is not always possible to find a target latch that is compatible with the latch used in the source. In this circumstance, the recourse is redesign of the source or manual changes after the transformation is complete. In addition to basic latch types, there are electrical characteristics to consider, such as latch speed and "glitching" properties. The reason that this problem surfaces in technology-to-technology mapping is that the latch requirements must be gleaned from a previous implementation instead of an original specification. Although the situation varies depending on the design philosophy of the product group, designers often tend to use everything the technology offers, which can aggravate the compatibility problems. With new designs and technology-independent sources, the designer can control his latch selection with the target in mind.

Human factors

A problem common to all forms of logic transformation is the human factors of the target design. No matter how much the transformation system does to improve the readability of the output and maintain a reference between key source and target landmarks, the end product will still not look familiar to the designer. Reactions have varied among designers, but we generally receive much more feedback on the human factors than on the efficiency of the mappings. As expected, the greater the change in form during the process, the greater the human factors problems. The logic that was originally specified as a PLA or in a design language is the most difficult to handle. In particular, the conversions from PLAs lack meaningful internal signal names, although the basic structure of the AND array, OR array, and bit-partitioning logic is usually still identifiable.

The obvious answer to human factors problems is to remove the need for the designer to deal with the detailed implementation. However, that is not always easy to do and in the case of technology-to-technology mapping it is not appropriate. Ideally, logic transformation fits only into a top-down methodology, but in practice it can fill more than one role. In some cases the transformation system can be used to keep the designer at a higher level of description and, in other cases, it becomes a sophisticated data entry tool. To be successful, the designer and the tool developers must meet each other halfway, to deal with the problem of human factors.

Looking to the future

There are two general areas for future work in logic transformation. The first is advancing the tools and techniques of the process itself. The second is establishing logic transformation as a normal part of logic design.

Some of the potential improvements in the tools and techniques are the following:

- More capacity for the efficient transformation of larger chips.
- More complete usage of complex target technology block sets which include larger macros and a mixed set of primitive functions.
- Better delay control for more technologies and for more complex clocking schemes.
- More generality in the transformation process to reduce the amount of technology-dependent code that is required.
- More sources, such as additional and more complex hardware description languages.
- More sophistication in the synthesis process, including alteration of the number of latches.
- More control over the generation of testability problems.

Initially, our biggest concern was the circuit efficiency of the target designs. Although there have been few cases where

designs have been done both manually and by TMS, the general reaction by the designers is that the output is competitive with manual design. There are always more transforms that can be added, particularly with certain target technologies or sources, but the general methodology and the use of local transforms have been proven to be effective. In addition, more quantitative comparisons have been done with the LSS system [5] which also establish the effectiveness of a local transformation approach.

Clearly, circuit efficiency and comparisons with manual designs are important, but our experience has shown that these are only parts of the total picture. Success must be based on user acceptance, which includes many aspects. The stability and availability of the tools, the size and skill of the support groups, the management of the human factors elements, the ability to deal with designs at different levels of description, the communication of limitations and assumptions among the people involved, are just some of the things that contribute to the whole job. Great strides have been made in the last few years and logic transformation tools are no longer in a purely experimental role. But the challenges will remain for years to come, for both the tool developers and the tool users and supporters, to increase the effectiveness and acceptance of logic transformation as the link between the logic designer and the technology.

Summary

TMS is characterized by a two-phase conversion process and an intermediate notation called GLN. The mapping process manipulates the GLN design until it is one-to-one with the target block set. The system relies on a set of rules in several forms that are used to control the process and provide target technology characteristics to the GLN form of the design. A key aspect of TMS that is shared with other systems, in particular LSS [4–6] and LTS [7], is a reliance on local transformations to produce an efficient design.

Automated logic generation of gate array or standard cell designs has progressed from experimentation to practical application in the last six years. Several systems of programs now exist to do the job from various types of design specifications. TMS has focused on the job of technology-to-technology mapping, which has proven to be useful both for product designs and for technology development.

Acknowledgment

The authors are indebted to Larry Segar, who managed the TMS project during its initial years.

References

1. J. Werner, "Progress Toward the 'Ideal' Silicon Compiler, Part 1: The Front End," *VLSI DESIGN* **4**, No. 5, 38–41 (1983).
2. T. D. Friedman and S. C. Yang, "Methods Used in an Automatic Logic Design Generator (ALERT)," *IEEE Trans. Computers* **C-18**, 593–614 (1969).
3. T. D. Friedman and S. C. Yang, "Quality of Designs from an Automatic Logic Generator (ALERT)," *Proceedings of the Seventh Design Automation Conference*, San Francisco, CA, 1970, pp. 71–89.
4. J. A. Darringer and W. H. Joyner, "A New Approach to Logic Synthesis," *Proceedings of the Seventeenth Design Automation Conference*, Minneapolis, MN, 1980, pp. 543–549.
5. John A. Darringer, William H. Joyner, Jr., C. Leonard Berman, and Louise Trevillyan, "Logic Synthesis Through Local Transformations," *IBM J. Res. Develop.* **25**, 272–280 (1981).
6. J. A. Darringer, W. H. Joyner, L. Berman, and L. Trevillyan, "Experiments in Logic Synthesis," *Proceedings of the IEEE International Conference on Circuits and Computers ICCC'80*, Port Chester, NY, 1980, pp. 234–237A.
7. J. B. Bendas, "Design Through Transformations," *Proceedings of the Twentieth Design Automation Conference*, Miami, FL, 1983, pp. 253–256.
8. L. I. Maissel and D. L. Ostapko, "Interactive Design Language: A Unified Approach to Hardware Simulation, Synthesis and Documentation," *Proceedings of the Nineteenth Design Automation Conference*, Las Vegas, NV, 1982, pp. 193–201.
9. S. Nakamura, S. Murai, C. Tanka, M. Terai, H. Fujiwara, and K. Kinsoshita, "LORES—Logic Reorganization System," *Proceedings of the Fifteenth Design Automation Conference*, Las Vegas, NV, 1978, pp. 250–260.
10. Y. Kitamura, M. Nagatani, and T. Hoshino, "Evaluation of the LSI Automatic Design Using The Function Description Language," *All Japan Electronic Communication Society Convention Record*, Article No. 434, p. 2–171 (1983).
11. M. Endo, T. Hoshino, and M. Nagatani, "An Engineering Project: Gate Automatic Generation Program," *Record of the 25th Convention of the Information Processing Society* (1982).
12. F. Rubin and P. W. Horstman, "A Logic Design Front-End For Improved Engineering Productivity," *Proceedings of the Twentieth Design Automation Conference*, Miami, FL, 1983, pp. 239–245.
13. D. Brand, "PLAs Versus Random Logic," *Research Report RC-9505*, IBM Thomas J. Watson Research Center, Yorktown Heights, NY, 1982.
14. E. B. Eichelberger and T. W. Williams, "A Logic Design Structure for LSI Testability," *Proceedings of the Fourteenth Design Automation Conference*, New Orleans, LA, 1977, pp. 462–468.
15. Gordon L. Smith, Ralph J. Bahnsen, and Harry Halliwell, "Boolean Comparison of Hardware and Flowcharts," *IBM J. Res. Develop.* **26**, 106–116 (1982).
16. D. Brand, "Redundancy and Don't Cares in Logic Synthesis," *Research Report RC-9386*, IBM Thomas J. Watson Research Center, Yorktown Heights, NY, 1982.
17. John A. Darringer, Daniel Brand, John V. Gerbi, William H. Joyner, Jr., and Louise Trevillyan, "LSS: A System for Production Logic Synthesis," *IBM J. Res. Develop.* **28**, 537–545 (1984, this issue).

Received December 2, 1983; revised March 9, 1984

James L. Gilkinson *IBM System Products Division, 3605 Highway 52 North, Rochester, Minnesota 55901.* Mr. Gilkinson is an advisory engineer in the circuit technology area of the Rochester laboratory. Joining IBM in 1969, he was assigned to optical character recognition development and worked on the IBM 3886 optical reader. He joined the circuit technology group in 1976 and has been involved with automated technology mapping since 1978. He received B.S. and M.S. degrees in electrical engineering from the University of Minnesota, Minneapolis, in 1967 and 1969. Mr. Gilkinson received an IBM Outstanding Innovation Award in 1983 for his work in technology mapping.

Amir Hekmatpour *IBM System Products Division, 3605 Highway 52 North, Rochester, Minnesota 55901.* Mr. Hekmatpour is a senior associate engineer in the circuit technology area of the Rochester laboratory. He joined IBM as an associate engineer in 1981. He received a B.S. in electrical engineering, a B.S. in computer and information engineering, and an M.S. in electrical engineering fom the University of Illinois, Chicago, in 1979, 1980, and 1981. Mr. Hekmatpour is a member of the Institute of Electrical and Electronics Engineers.

Steven D. Lewis *IBM System Products Division, 3605 Highway 52 North, Rochester, Minnesota 55901.* Mr. Lewis is a staff engineer in the advanced systems engineering area of the Rochester laboratory. He joined IBM as an associate engineer in 1976. He received the B.S. and M.S. degrees in electrical engineering from Purdue University, Lafayette, Indiana, in 1975 and 1976. He has been involved with the IBM magnetic bubble program and circuit technology, and is currently working on design verification for the IBM System/38. Mr. Lewis received an IBM Outstanding Innovation Award in 1983 for his work in technology mapping.

Bruce B. Winter *IBM System Products Division, 3605 Highway 52 North, Rochester, Minnesota 55901.* Mr. Winter is a senior associate engineer in Rochester's circuit technology area. He received a B.S. in electrical engineering from Montana State University, Bozeman, in 1980 and an M.S. in electrical engineering from the University of Wisconsin, Madison, in 1981.