High-Level Library Mapping for Arithmetic Components

Pradip K. Jha and Nikil D. Dutt

Technical Report #94-15
April 10, 1994

Dept. of Information and Computer Science
University of California at Irvine
Irvine, CA 92717-3425
Phone: (714) 856-8059
Fax: (714) 856-4056
Email: pradip@ics.uci.edu

Abstract

We present High-level Library Mapping (HLLM), a technique that permits reuse of complex RT-level databook components (specifically ALUs). HLLM can be used to couple existing databook libraries, module generators and custom-designed components with the output of architectural or behavioral synthesis. In this report, we define the problem of high-level library mapping, present several algorithmic formulations for HLLM of ALUs, and demonstrate the versatility of our approach on a variety of libraries. We also compare HLLM against the traditional mapping approach using logic synthesis. Our experiments show that HLLM for ALUs outperforms logic-synthesis in area, delay and runtime, indicating that HLLM is a promising approach for reuse of datapath components in architectural design and high-level synthesis.

Contents

List of Figures

List of Tables

1 Introduction

With the trend towards the design of increasingly complex electronic systems, the need for design reuse techniques at different design levels becomes critical. Our motivation for this work stems from this need for design reuse at abstraction levels higher than the logic level. Design reuse, in this context, means utilization of previously designed circuits, parts or subsystems in new designs. Design reuse is therefore applicable in a variety of design scenarios, a few of which are listed below:

- *Design Volume.* Typically, system design proceeds through various phases, from prototypes to large volume implementations. Initially, the designer might make a prototype of the design in order to verify the functionality of the design within its environment. Design prototypes and low-volume designs are often manufactured using FPGAs. These FPGA designs are often migrated to gate arrays when the volume of demand goes above a threshold. In a high-volume environment, the same design may get implemented using a custom methodology. Design reuse is applicable in all of these cases.

- *Design Versions.* Demands from the marketplace often necessitate redesign for different versions of the same functional design. Examples include standard and low-power versions of a processor.

- *New Features.* A new design could be generated by upgrading an existing design with additional features. The functionality of the old design is enhanced with those of the new features.

- *Coupling HLS with Physical Design.* The output of high-level synthesis (HLS) typically consists of a generic RTL datapath and a sequencing table; the generic datapath components have to be realized using existing libraries. This RT-level design can be reused by mapping the generic components to different libraries.

- *Technology Retargetting.* Newer technologies with smaller feature sizes result in circuits with better performance and smaller area. Hence there is a need to migrate existing designs to newer technologies. In some instances, the older libraries may no longer be supported, forcing the need for technology or library migration.

In all of these design scenarios, we would like to replace a component with a new component, insert a new component, or retarget the whole design onto components from a new library. A complete redesign is an option, but is typically not considered unless reuse of the existing design is not possible.

Design reuse can be implemented at several design levels, from behavior down to layout. Typically, designs are reused at the layout level when handcrafted or custom layout blocks are stored in a library. However, present-day design methodologies involving schematic capture and simulation typically use RT components, making the RT-level a good candidate for design reuse. Databook libraries are also often specified at the RT level (e.g., TTL databook). Thus RT-level components such as ALUs and register-files are good candidates for design reuse. Furthermore, RT components have well-defined behavior and are often regularly structured; they are typically optimized and placed in a library with the intention of facilitating design reuse. All of these factors indicate tremendous potential for design reuse at the RT-level. However, there has not been much previous work in implementing design reuse at the RT level.

In this report, we propose a novel approach to design reuse at the RT-level through high-level library mapping (HLLM), an approach that effects design reuse by mapping an RT-level component onto another RT-level component. In this approach, library mapping is defined as a source-to-target component mapping, where both the source and target components are similar RT-level components (e.g., ALUs or register files). This approach can reuse components not only from from standard libraries, but also from datapath generators, as well as handcrafted components.

This report specifically describes a high-level mapping technique for arithmetic-logic unit (ALUs). ALUs are good candidates for design reuse due to several reasons: they are often handcrafted and stored in libraries for future use; different versions of ALUs exist in databases or design groups within a company; and logic synthesis and logic-level technology mapping techniques do not work well for regularly-structured components such as ALUs. We consider

ALUs that perform any subset of arithmetic, logic and comparison functions, using a well-defined set of operations for the ALUs. Hence it is possible to formulate mapping schemes that map an ALU onto another ALU based on the sets of functions they perform.

In this report, we define high-level library mapping problem for ALUs, describe few algorithms to solve the problem, and provide experimental results to validate its effectiveness. Specifically, we demonstrate the versatility of this approach by applying HLLM to ALUs drawn from different libraries. We also compare the HLLM approach versus a traditional logic synthesis approach and demonstrate the advantages of HLLM for complex datapath components.

This report is organized is as follows. Section 2 describes related work. Section 3 defines high-level library mapping for ALUs based on their functional behavior. Section 4 discusses the overall approach to the mapping problem. Section 5 presents some algorithms that map a source ALU onto a target ALU. Section 6 demonstrates the comprehensiveness and quality of designs produced by our approach. The report concludes with a summary and future work.

2 Related work

There has been a fair amount of related work in applying design reuse at different levels of design. In the context of high-level design and synthesis, reuse techniques can be divided into two categories. One school of thought attempts to directly incorporate components from a technology library into the high-level design phase. This approach combines two steps of the high-level design process (design synthesis and technology mapping), and provides a direct approach to the reuse of RT-level library components. Although this approach can yield good results for a specific library, it requires a lot of effort in tuning the synthesis and refinement tools to accommodate variances in RT-level technology components. Furthermore, changes in the RT-library may necessitate a complete rewrite of the core synthesis algorithms that implement HLS with the RT-level components. Work described in [Marw93] [AnDu94] [RuGB93] [GCDM93] falls into this category.

The other (more traditional) school of thought solves the problem using a two-phase approach. First, the design refinement phase maps the behavior into some intermediate (generic) level, and second, technology mapping realizes the final design by mapping this intermediate (generic) level design using cells from a technology library. We can classify the component mapping problem into four approaches, based on the levels of building blocks used to realize a generic component:

Logic-level Mapping. At the logic level, a component's functionality can be described using Boolean equations for the transformation of the inputs into outputs. These equations can then be mapped to low-level technology-specific components such as gates, flip-flops and latches. For example, the ALU in Figure 1 can be described with Boolean equations for each output(O0, OCOUT and OZERO) in terms of the inputs I0, I1, ICIN and C. Each of these equations can be mapped to components from a logic-level technology library (e.g., NOR gates) using structural (tree-based) , Boolean matching or rule-based mapping techniques. [BRSW87] [Keut87] [Syno92] [MaMi90] [DJBT94]. This type of mapping is also commonly called logic-level technology mapping.

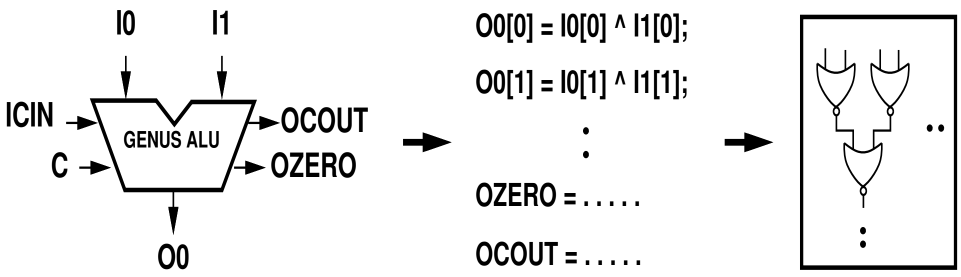


Figure 1: Logic-level mapping of an ALU

At the logic level, we have well-characterized primitive cells and technology mapping provides good results for small and random logic designs. As soon as the complexity of the circuit grows, the run-time of logic level tools becomes prohibitive. [CaTr89] presents an investigation of the relationship between logic-level and high-level synthesis and presents some basic tradeoffs. It is commonly known that designs produced by logic synthesis for regularly-structured datapath components are often of poor quality, indicating the need to apply mapping techniques at higher levels of abstraction. MILO [VaGa88] is one approach that combines logic-level mapping techniques with microarchitectural optimization to realize a netlist of RT-level components.

Functional Decomposition. At the RT-level, regular-structured datapath components can be mapped to MSI-level blocks from a technology library. Each component can be functionally and/or structurally decomposed into smaller building blocks based on well-defined techniques for building datapath components of larger sizes. For instance, an ALU can be implemented as separate AU and LU blocks that are MUXed at the output. Alternatively, an ALU can be built using replicated bit-slices of one-bit ALUs.

The choice of such construction schemes leads to a design space of alternative implementations, where the RT-level component is represented as a hierarchical tree of alternative decompositions using library primitives. The root of the tree represents the source component (i.e., the one to be mapped), while leaves of the tree consist of the MSI/SSI-level blocks from the technology library. Figure 2 shows a sample decomposition tree for an ALU. This ALU is realized by composing the leaf cell blocks (such as 4-bit adders, FAs, MUX2, gates) from a technology-specific component library. The DTAS system [DuKi91] follows this mapping approach. The functional decomposition approach is useful when the target component bit widths are much smaller than that of the source component.

High-level Library Mapping. High-level Library Mapping (HLLM) can be viewed as a source-to-target component mapping approach where the source and target components have overlapping functionality and are of approximately equal size and complexity. In this approach, the source component is implemented using the target component, with a minimal amount of glue logic to satisfy the design constraints.

Our work investigates HLLM for RT-level datapath components that have well-defined functional behavior. The source and target components are described as a set of well-characterized functions; these functional specification are then used to drive the

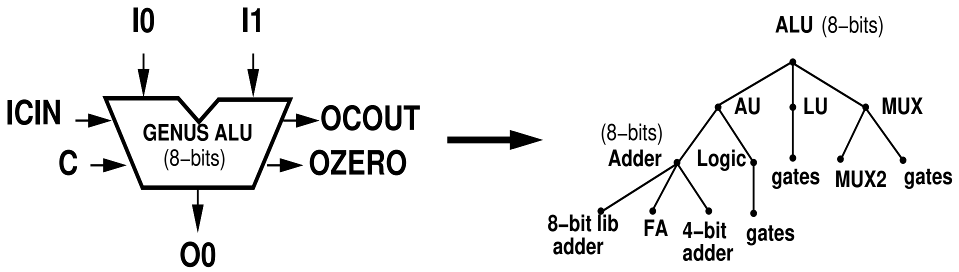


Figure 2: Functional decomposition of an ALU

source-to-target mapping process. To the best of our knowledge, prior to this work, there has not been any work that investigates HLLM for RT-level components.

System-level mapping. At the system level, mapping can be performed between system-level components such as processors, memories and interface units. MICON [BiGS89] is one approach that tries to reuse off-the-shelf system-level parts such as processors, memories and peripherals to build a single-board computer system. The input to MICON is a set of system-level specifications that describe the functionality of the required computer in terms of the type of processor, amount and type of memory, etc., along with the design constraints (board size, cost, etc.). MICON generates a design (netlist of the above components) that satisfies the requirements given to the system.

The work on HLLM described in this report therefore complements existing mapping approaches at the logic and system levels, and bridges the gap between these two design levels.

3 Problem Definition

The high-level library mapping problem for ALUs is based on functional specifications of the source and target components, which are compared with respect to a "canonical" functional representation to derive an effective mapping result. When the functionality of the target component does not exactly match that of the source component, the target component may need to be padded with additional (glue) logic. Figure 3 illustrates this high-level mapping approach between a source and a target ALU.

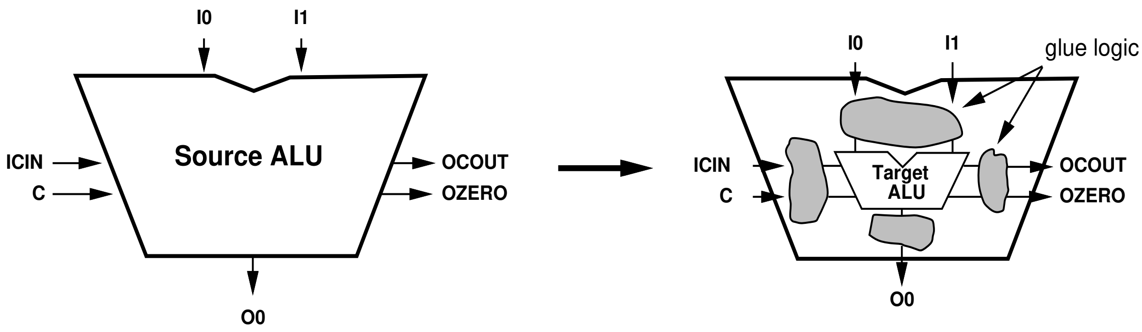


Figure 3: High-level library mapping of an ALU

We therefore define the high-level library mapping problem in terms of a *Source component (S)*, a *Target component (T)* and a set of *Mapping rules (R)* that maps the source component onto the target component. In order to establish equivalence between the source and the target components, each component is described in terms of a set of RT-functions that are defined using a canonical representation of the component. A mapping rule in R describes an alternative for implementing a function in S using a function in T; each source function can potentially be implemented by different target functions. The task of high-level library mapping, then, reduces to selecting a set of rules, one for each function in source component S, that realizes the best mapping of S on T with respect to the cost function (e.g., area or delay).

In the rest of this section we discuss our assumptions, the canonical representation used for components and functions, and the mapping rules along with cost function to be used in our approach.

3.1 Assumptions

In this work, we make the following assumptions:

- All data and arithmetic use the 2's complement representation.
- A source component can perform only one function at a time. For example, a comparator can implement several RT-functions (e.g., EQ, NEQ, GT, LT, etc.), but only one function is performed at a time.
- We restrict ourselves to arithmetic, logic and comparison functions. These functions are defined using a universal ALU that performs a set of canonical arithmetic, logic and comparison functions (as described in the next section).
- Each of the target component's RT-functions should either be a canonical RT-function or a simple negation of a canonical RT-function.
- The source (S) and the target (T) components have the same bit-widths.

These assumptions are made in order to to make the problem size tractable and also to facilitate illustration of the high level mapping approach. We believe that these assumptions can be relaxed once the basic HLLM approach is defined and well understood.

3.2 Universal ALU representation

In order to provide a reference model for HLLM, we define a universal ALU (U) that performs a canonical set of ALU functions: 5 arithmetic functions (ADD, SUB, RSUB, INC, DEC), 16 logic functions (all Boolean functions of 2 variables), and 6 comparison functions (EQ, NEQ, GT, GEQ, LT, LEQ). Using these canonical ALU functions, we can build any other ALU including library-specific ALUs. These canonical ALU functions are described in more detail later in this section.

Figure 4 shows an n-bit universal ALU with the following ports:

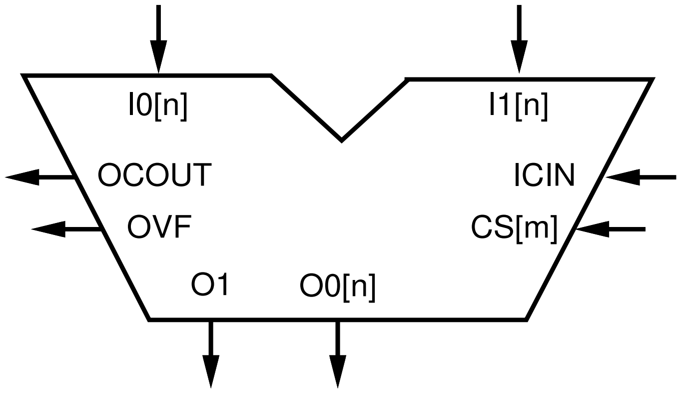


Figure 4: The universal ALU

- **I0[n]:** primary left input.
- **I1[n]:** primary right input.
- **O0[n]:** primary output for arithmetic and logic functions.
- **ICIN:** carry input used by the arithmetic functions.
- **OCOUT:** carry output used by arithmetic functions.
- **OVF:** secondary output that is used by arithmetic functions.
- **O1:** comparison output.
- **CS[m]:** control input.

3.3 Canonical ALU functions

Each canonical ALU function defines a functional mapping between the inputs and the outputs of the universal ALU. Note that an ALU function need not use all the ports of a universal ALU. Table 1 presents the canonical representation for 5 arithmetic functions. The arithmetic functions make use of the following ports: I0[n], I1[n], O0[n], ICIN, OCOUT and OVF. The canonical representation for 16 logic functions are shown in Table 2. These

Function	Operation
ADD	(OCOUT, O0[n], OVF) = I0[n] + I1[n] + ICIN
SUB	(OCOUT, O0[n], OVF) = I0[n] + $\overline{I1[n]}$ + ICIN
RSUB	(OCOUT, O0[n], OVF) = $\overline{I0[n]}$ + I1[n] + ICIN
INC	(OCOUT, O0[n], OVF) = I0[n] + 1
DEC	(OCOUT, O0[n], OVF) = I0[n] - 1

Table 1: Canonical arithmetic functions

Function	Operation
ZERO	O0[n] $= 0[n]$
ONE	O0[n] $= 1[n]$
AND	O0[n] $= I0[n] \wedge I1[n]$
NAND	O0[n] $= \overline{I0[n] \wedge I1[n]}$
OR	O0[n] $= I0[n] \vee I1[n]$
NOR	O0[n] $= \overline{I0[n] \vee I1[n]}$
XOR	O0[n] $= I0[n] \oplus I1[n]$
XNOR	O0[n] $= \overline{I0[n] \oplus I1[n]}$
LID	O0[n] $= I0[n]$
RID	O0[n] $= I1[n]$
LNOT	O0[n] $= \overline{I0[n]}$
RNOT	O0[n] $= \overline{I1[n]}$
LINHI	O0[n] $= \overline{I0[n]} \wedge I1[n]$
RINHI	O0[n] $= I0[n] \wedge \overline{I1[n]}$
LIMPL	O0[n] $= \overline{I0[n]} \vee I1[n]$
RIMPL	O0[n] $= I0[n] \vee \overline{I1[n]}$

Table 2: Canonical logic functions

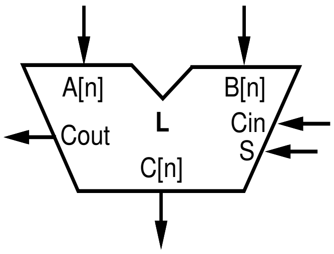
Function	Operation
EQ	O1 $= (I0[n] = I1[n])$
NEQ	O1 $= (I0[n] \neq I1[n])$
GT	O1 $= (I0[n] > I1[n])$ †
LT	O1 $= (I0[n] < I1[n])$ †
GEQ	O1 $= (I0[n] \geq I1[n])$ †
LEQ	O1 $= (I0[n] \leq I1[n])$ †

†Assumes that the data is in 2's complement form.

Table 3: Canonical comparison functions

functions use only primary inputs (I0[n], I1[n]) and the primary output (O0[n]). Table 3 describes the canonical representation of 6 comparison functions. These comparison functions use primary inputs (I0[n] and I1[n]) and the primary output O1. The universal ALU therefore has a total of 27 canonical ALU functions (5 arithmetic, 16 logic and 6 comparison).

3.4 Representation of library components



(a)

Source Function	Canonical Function	Input			Output		Control
		I0	I1	ICIN	C	Cout	S
$\bar{A}$+B+Cin	ADD	$\bar{A}$	AB	Cin	O0	OCOUT	0
$\overline{A \vee B}$	OR	A	B	–	$\overline{O0}$	–	1

(b)

Figure 5: A library component example: (a) Port description (b) Function description

We need a representation of library components (e.g., S or T in our problem) that not only captures the functionality but also facilitates the comparison between them. A representation based on canonical functions supports both these needs. Specifically, an ALU library component is described by its port lists and the set of functions it performs. A function in the library component is a variant of one of the 27 canonical ALU functions. It is described by providing the corresponding canonical function and the Boolean relationship between the ports of the library component and the ports of the universal component. The Boolean relation between the library component ports and the universal component ports is described in a tabular fashion. The table contains a row for each function in the library component; there is a column for each input port in the universal ALU, a column for each output port in the library component and a column for the control configuration. The control configuration lists the values of control lines for each function. An entry in this table consists of a Boolean expression showing relationship between the ports of a library component L and the universal ALU U. For example, a library component (L) that performs the functions

($C = \overline{A} + AB + Cin$) and $C = \overline{A \vee B}$ is shown in Figure 5(a), and the corresponding tabular representation is shown in Figure 5(b).

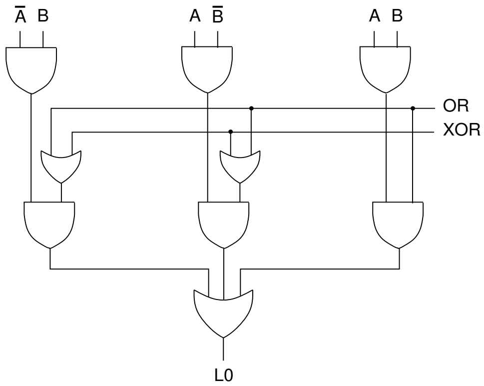


Figure 6: Implementation of logic unit with two functions: OR and XOR

3.5 Logic function representation

Each logic function of the ALU is described using a standard minterm representation of two primary inputs. Note that we have four minterms with two inputs A and B, namely $\overline{AB}$, $\overline{A}B$, $A\overline{B}$ and AB. A specific logic function selects a subset of these four minterms. As an example, the OR function is given by the following minterms: $\overline{A}B$, $A\overline{B}$ and AB. In other words, when one or more of these three minterms are active, output of the OR function is 1. Table 4 lists minterms for all the 16 logic functions. A set of logic functions is implemented by ANDing the minterms for each function with the corresponding control lines and feeding the output to an OR gate. As an example, Figure 6 shows an implementation for two logic functions OR and XOR.

We use an ordered vector (logic vector or LV) of length 4 for representing the implementation of one or more logic functions. The first entry in the logic vector represents the first minterm ($\overline{AB}$), the second entry represents ($\overline{A}B$), the third entry represents ($A\overline{B}$) and the fourth entry represents the last minterm (AB). For example, the logic function OR is represented by the vector "0111", since it does not need minterm $\overline{AB}$ but requires all the

Function	$\overline{A}\overline{B}$	$\overline{A}B$	$A\overline{B}$	AB
ZERO	0	0	0	0
ONE	1	1	1	1
AND	0	0	0	1
NAND	1	1	1	0
OR	0	1	1	1
NOR	1	0	0	0
XOR	0	1	1	0
XNOR	1	0	0	1
LID	0	0	1	1
RID	0	1	0	1
LNOT	1	1	0	0
RNOT	1	0	1	0
LINHI	0	1	0	0
RINHI	0	0	1	0
LIMPL	1	0	1	1
RIMPL	1	1	0	1

Table 4: Minterms for logic functions

other minterms $\overline{A}B$, $A\overline{B}$ and AB. A logic vector (LV) for a set of logic functions is obtained by adding the vectors (entry wise) for each function. Thus, the logic vector for the two logic functions OR ("0111") and XOR ("0110") is "0221".

3.6 Mapping rule representation

A mapping rule describes how to implement a canonical function from another canonical function. Given a set of mapping rules, we can implement all the functions in the source component including the ones that are not present in the target component. Let SF and TF be the source and the target canonical functions respectively. Let us define port names for the source and target canonical component (CS and CT) as shown in Figure 7. A mapping rule describes the mapping of CS ports onto CT ports such that SF is implemented using TF.

A rule is described in a tabular fashion similar to the library component representation. Table 5 lists some sample rules. Each row of this table contains a rule name, a source

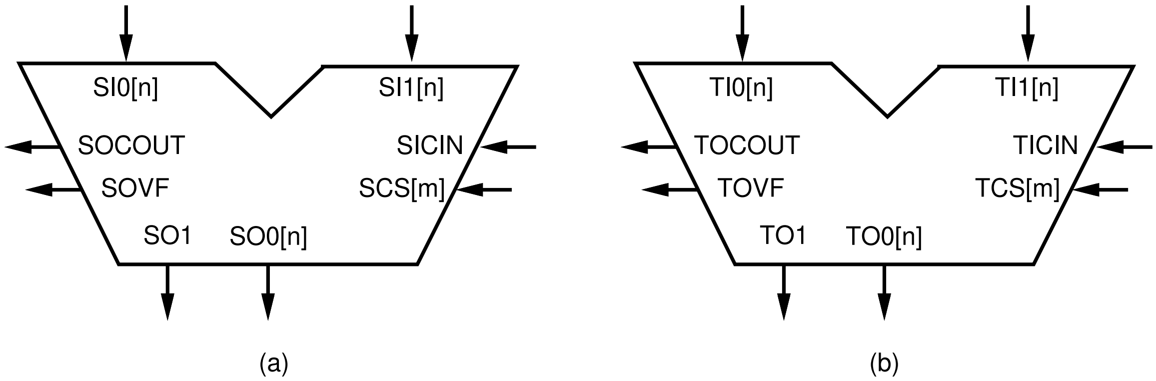


Figure 7: Port names: (a) Source canonical ALU (b) Target canonical ALU

function (SF), a target function (TF), and the port mapping information. The first set of port mappings expresses the target component's inputs in terms of the source component's inputs. The second set of port mappings describes the the source component's outputs in terms of the target component's outputs.

Each rule describes the implementation of a source function using a target function and indicates the port mappings required to implement the mapping. Note that each source function can be implemented using several alternative target functions. For example, the source *ADD* function in Table 5 could be implemented using target *ADD* function (rule "AA1"), or with the target *SUB* function (rule "AS1"). The input and output port entries indicate the connectivity and additional logic required to implement the mapping rule. For instance, the second rule "AS1" in Table 5 implements the source function *ADD* with the target function *SUB* by inverting the right input ($\overline{SI1}$).

For each source logic function, there is a rule that implements the function from scratch without using any target function. To this rule, we add another entry in the table: a logic vector (LV) as discussed previously. The primary output for a logic rule is given by the logic output (LO). Table 6 lists some sample logic rules. For example, the source *AND* logic function could be implemented from scratch (rule "AN_") by adding the minterm corresponding to the LV "0001", i.e. AB. Appendix A contains an extensive list of rules for

Rule name	*Source function*	*Target function*	*Input ports*			*Output ports*			
			TI0	*TI1*	*TICIN*	*SO0*	*SOCOUT*	*SOVF*	*SO1*
AA1	ADD	ADD	SI0	SI1	SICIN	TO0	TOCOUT	TOVF	–
AS1	ADD	SUB	SI0	$\overline{SI1}$	SICIN	TO0	TOCOUT	TOVF	–
IS1	INC	SUB	SI0	"1..1"	'1'	TO0	TOCOUT	TOVF	–
EX1	EQ	XOR	SI0	SI1	–	–	–	–	nor(TO0)

Table 5: Sample mapping rules

Rule name	*Source function*	*Target function*	*Input ports*			*Output ports*				*LV*
			TI0	*TI1*	*TICIN*	*SO0*	*SOCOUT*	*SOVF*	*SO1*	
ANAN	AND	AND	SI0	SI1	–	TO0	–	–	–	–
AN_	AND	–	–	–	–	TL0	–	–	–	"0001"
ANNA	AND	NAND	SI0	SI1	–	$\overline{TO0}$	–	–	–	–
XO_	XOR	–	–	–	–	TL0	–	–	–	"0110"

Table 6: Sample mapping rules for logic functions

all the ALU functions.

3.7 The cost function

A good cost function captures the important characteristics of an efficient source-to-target component mapping. We use a cost function based on two criteria: (a) an area metric, represented by the *gate-count* of the hardware overhead, and (b) a delay metric, represented by the worst case delay or *max-delay* of the generated design.

3.7.1 Gate-count

Mapping S on T requires extra hardware that could arise due to:

- Routing data from the inputs of S to the inputs of T and from the output of T to the output of S.
- Mapping a function in S onto some other function on T.

- Generating a function (for example, a logic function) of S from basic gates.
- Mismatch between the canonical functions and the functions in S and T.
- Mapping the control lines of S onto the control lines of T.

In our current formulation, we use the *gate-count* (GC) of the extra hardware as a measure of the hardware cost. Specifically, we use the number of equivalent 2-input gates as the cost function to guide our algorithm. Note that the actual cost of a design should also include the cost of the target component. However, since this cost function is used just to compare two designs, it does not matter if we exclude the component cost in our cost function, because this portion of the cost will be present in each design. Therefore, we use the gate-count of the hardware overhead as an area optimization criterion.

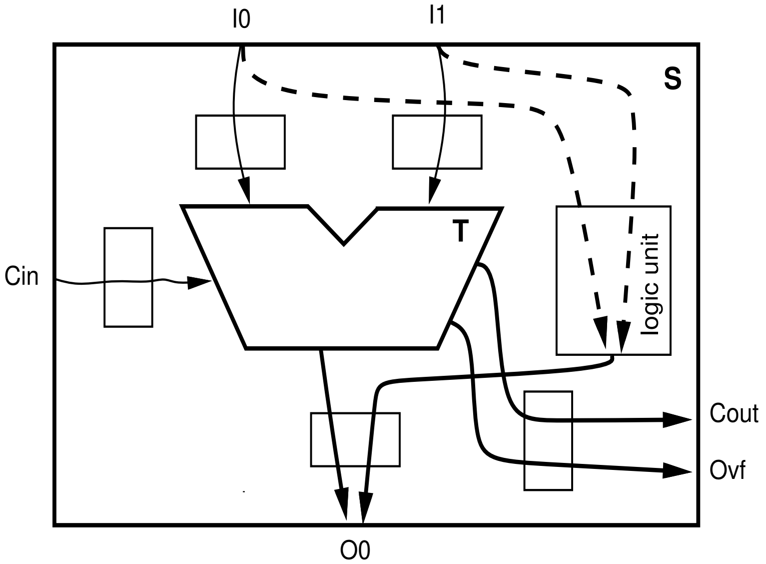


Figure 8: Worst case delay for a design

3.7.2 Max-delay

We use the worst-case delay of the design as a delay metric. The worst-case delay, *max-delay* (MD), for a design is given by the maximum delay through all paths of the design. It is an approximation of the delay of the resultant design.

As an example, consider the design shown in Figure 8, that shows a sample source component (S) mapped to a target component (T). Let (MD_t) be the maximum delay through the component T. We can calculate MD for the design S in the following way:

1. Calculate the max-delay to the inputs(MD_i) of T. It is given by the maximum of delays through all the paths shown by light lines in Figure 8.

2. Calculate the max-delay to the output(MD_o) of T. It is given by the maximum of delays through all the paths shown by thick lines in Figure 8.

3. Calculate the worst case delay for logic-circuit(MD_l) of T. It is given by the maximum delay through all the paths shown by shaded lines in Figure 8.

4. The MD of the design, then, is given by maximum of the two figures:

$$Max(MD_i + MD_o + MD_t, MD_l + MD_o)$$

Note that unlike the previous area metric (gate-count) calculation, we cannot ignore the delay of the target component T. This is because MD for all the designs may not include the delay of T; the worst case path might pass through the logic unit.

4 Overall Approach

Figure 9 illustrates our overall approach to the mapping problem. The inputs to the system consist of the source component (S), the target component (T) and the mapping rule database (R). As mentioned before, both S and T are library components and they are described as source canonical and target canonical components using the representation discussed in the previous section. The mapping rule database (R) contains all the rules required to map one RT-function onto another RT-function. The rule database is also represented in a tabular fashion. In the first step, the mapping algorithm implements the source component using the target canonical component. In the second step, the target canonical function is mapped

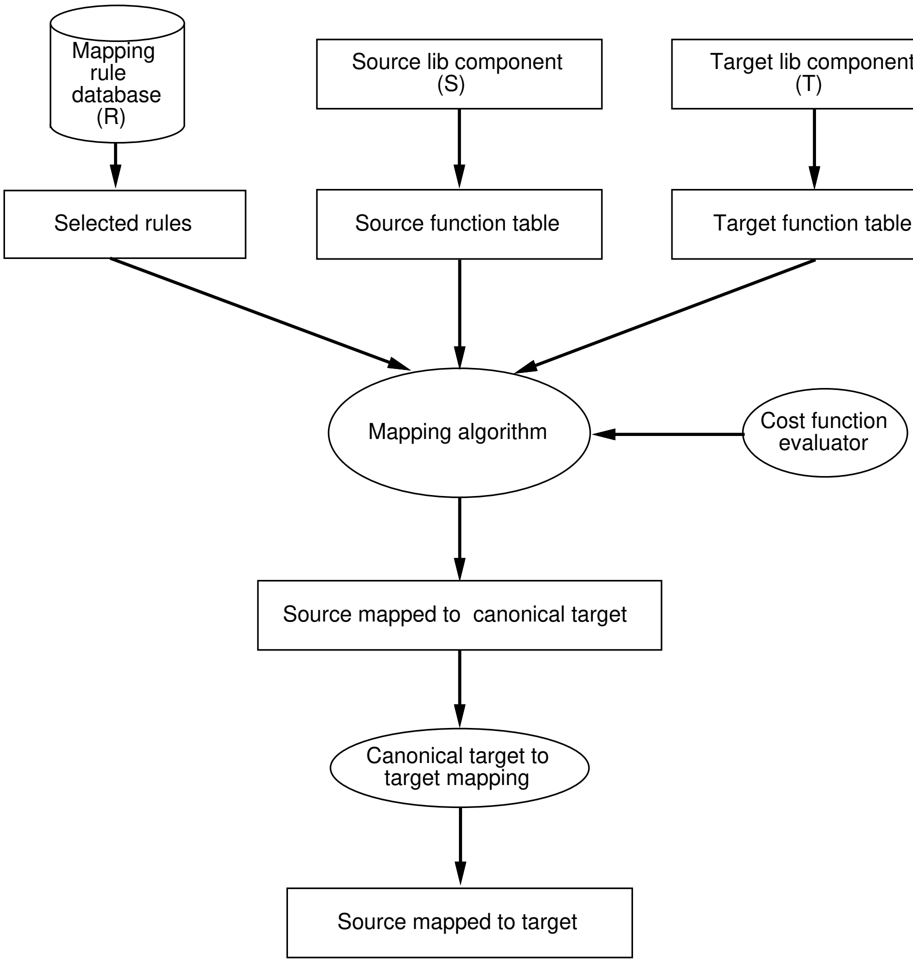
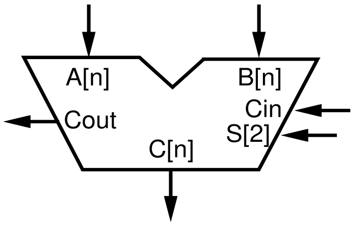


Figure 9: Overall approach to mapping problem

to the actual target component. The output of the system is an implementation of S on the target component T with some additional (glue) logic surrounding T. This report focuses primarily on the first mapping step and describes several mapping algorithms for it. The second step consists of simple tasks such as the matching of port names and is relatively trivial.

We illustrate each of the steps in the approach by walking though an example. Let S be an arithmetic unit that can perform three functions: ADD, SUB and RSUB. Let T be another arithmetic unit that can perform two functions: ADD, SUB. These two components are shown in Figures 10 and 11 respectively. Note the differences in the port names of these

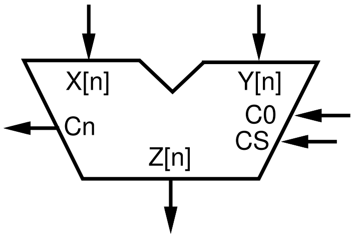


(a)

Function	Input			Output		Control	
	I0	I1	ICIN	C	Cout	S[1]	S[0]
ADD	A	B	Cin	O0	OCOUT	0	0
SUB	A	B	Cin	O0	OCOUT	0	1
RSUB	A	B	Cin	O0	OCOUT	1	0

(b)

Figure 10: Example source component: (a) Port description (b) Function table



(a)

Function	Input			Output		Control
	I0	I1	ICIN	Z	Cn	CS
ADD	X	Y	C0	O0	OCOUT	0
SUB	X	Y	C0	O0	OCOUT	1

(b)

Figure 11: Example target component: (a) Port description (b) Function table

two components. From these component descriptions, we extract the function tables for S and T as shown in Figures 10(b) and 11(b).

The mapping rule database (R) contains an extensive set of rules to map one RT-function onto another. From this database, we extract rules that map a source function onto a target function. Table 7 shows some interesting rules that have been extracted for our example.

In the next step, the mapping algorithm implements S onto a canonical ALU (C). This canonical ALU uses only those functions that are present in T. This mapping is achieved by finding a set of rules, one for each source function, such that cost of extra hardware (i.e., gate count) is minimized. The set of selected rules provides the connectivity between the ports of S and C. Figure 12 shows one such solution in terms of the selected set of rules and the canonical implementation.

Rule name	*Source function*	*Target function*	*Input ports*			*Output ports*			
			TI0	*TI1*	*TICIN*	*SO0*	*SOCOUT*	*SOVF*	*SO1*
AA1	ADD	ADD	SI0	SI1	SICIN	TO0	TOCOUT	TOVF	–
AS1	ADD	SUB	SI0	$\overline{SI1}$	SICIN	TO0	TOCOUT	TOVF	–
SA1	SUB	ADD	SI0	$\overline{SI1}$	SICIN	TO0	TOCOUT	TOVF	–
SS	SUB	SUB	SI0	SI1	SICIN	TO0	TOCOUT	TOVF	–
RA1	RSUB	ADD	$\overline{SI0}$	SI1	SICIN	TO0	TOCOUT	TOVF	–
RS	RSUB	SUB	SI1	SI0	SICIN	TO0	TOCOUT	TOVF	–

Table 7: Selected set of rules for mapping example

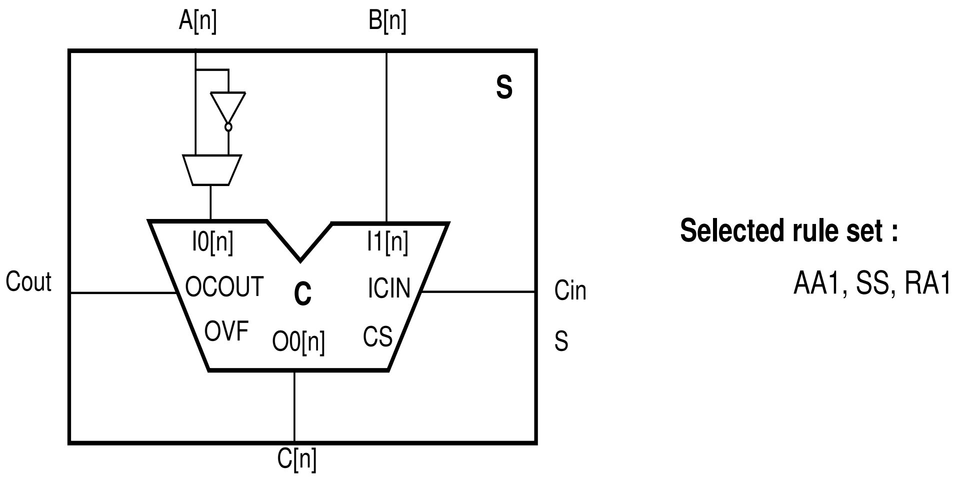


Figure 12: Mapping of source component (S) to canonical component (C)

The final step involves mapping the canonical ALU (C) onto the target component (T). This is usually a simple process since we restrict T to be very close to C (see Section 3.1). This step connects the ports of C to the ports of T. Figure 13 shows the final implementation.

Note that generation of the final design requires solving many other subproblems such as bit-width mapping, control mapping, secondary input and output mapping, port name mapping, etc. Again, it is important to note that the work described in this report focuses on the algorithms for the functional mapping of the source to the target component, which is the heart of the mapping problem (the first mapping step in Figure 9).

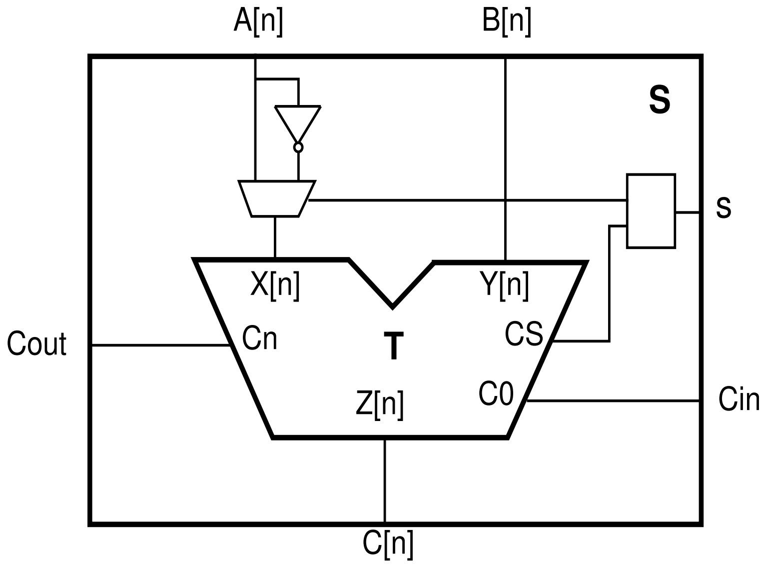


Figure 13: Mapping of source component (S) to target component (T)

5 Mapping Algorithms

The task of mapping a source component (S) onto target component (T) is accomplished by selecting a set of mapping rules, one for each function in source component S, that realizes the best mapping of S on T with respect to the cost function (e.g., area or delay). Recall that not only are there multiple mapping rules for each source function, but the selection of mapping rules for various source functions are also interdependent. For example, consider the rules presented in Table 7. If we decide to use the rule "AS1" for mapping the source function ADD, the rule "SA1" for source function SUB would lead to an efficient implementation since it shares the factor $\overline{SI1}$ for the right primary input. Thus a strategy is required to select a mapping rule out of multiple alternatives for each source function. In this section, we present a suite of mapping algorithms that perform this selection of mapping rules in an efficient manner. These algorithms take as input the function tables of T and S along with the selected rule set, and map a source component (S) onto the canonical component (C). This corresponds the to first (and major) mapping step in Figure 9. The algorithms generate as output the required mapping in terms of the set of rules and port connectivity.

We first define the following components and sets:

S : The source component to be implemented

T : The technology library component onto which the source component has to be implemented.

f(S) : The set of the functions in the source component S.

f(T) : The set of the functions in the library component T.

C : The canonical component that has f(T) functions.

r(ST) : The set of rules that map a function in f(S) onto a function in f(T).

Also, we define cardinalities for some of these sets. These cardinalities will be used in analyzing the complexity of the mapping algorithm :

nS : the number of functions in the source component ($nS = |f(S)|$).

nT : the number of functions in the technology component ($nT = |f(T)|$).

nST : the number of rules ($nST = |r(ST)|$).

mST : the average number of rules per function in source component T ($mST = \frac{nTS}{nS}$).

Our mapping algorithms employ a constructive approach where an initial (possibly null) partial solution is refined into the final solution. While working through an algorithm, we need to keep track of the partial mapping. A partial mapping is represented by storing the list of inputs to the input ports of C and the output ports of S. Eventually, all these inputs are multiplexed based on the control signals. Specifically, a partial solution keeps the following list of inputs to each of these terminals:

I0: Left input, given by Boolean expressions with primary inputs of S and constants.

I1: Right input, given by Boolean expressions with primary inputs of S and constants.

ICIN: Carry input, given by Boolean expressions with carry input and constants.

Primary output of T: given by Boolean expressions with primary output of C, logic output (LO) and constants.

Comparison output of T: could be generated with the comparison output of C and some functions of O0 and L0.

Secondary output of T: e.g., carry out and overflow signal.

Logic vector(LV): used to keep track of the implementation of logic functions from scratch (as discussed in Section 3.5). This is required only if logic functions are to be synthesized.

A textual representation of the partial solution could be described in a tabular fashion just like the mapping rule. In fact, each mapping rule is a partial mapping with only one function. Since we know the ordering of input and output ports (e.g., I0, I1, ICIN, Primary output, Comparison output, Secondary outputs and LV), we need not be explicit in associating the terminal to its input list. Instead, an ordered arrangement of list of inputs for each terminal should suffice. As an example, the output of the mapping algorithm in the previous section (Figure 12) is shown in Figure 14. It shows both the explicit tabular and the implicit representation.

Note that we need not keep track of all the above input lists. Depending on the requirements (set of functions to be mapped) some of the port mappings might not be required and could be omitted. For example, if we are not dealing with logic functions, LV could be omitted. If we are using the implicit representation, we have to be more careful. Note that it is very easy to compute the hardware cost(gate-count) and the implementation from this partial solution representation. This is evident from the example shown in Figure 14 as we see that there is a one-to-one correlation between the hardware and the representation.

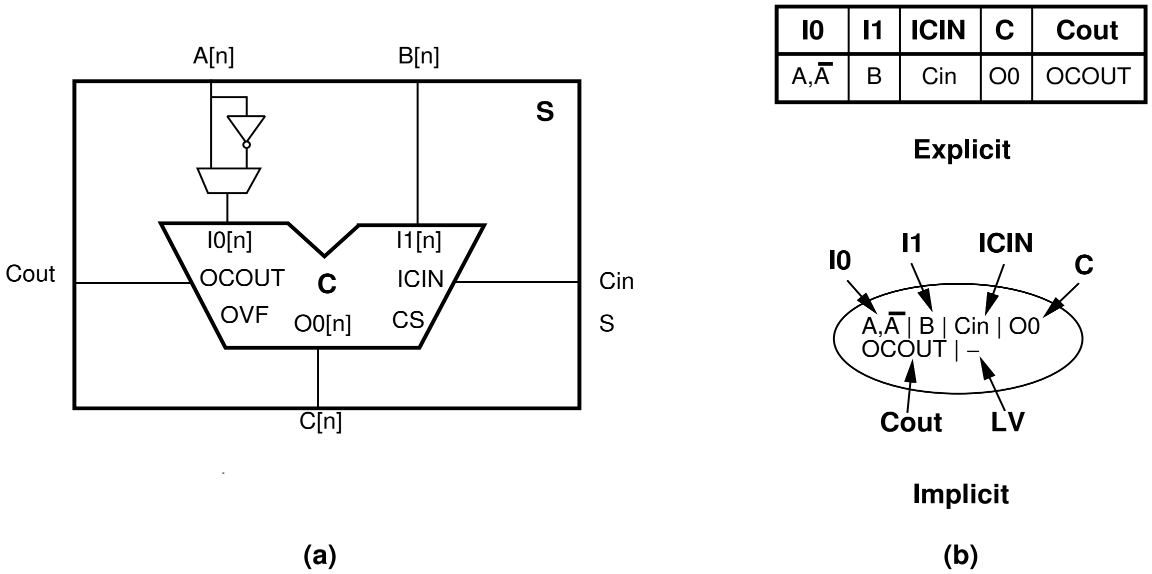


Figure 14: Partial solution example: (a) Pictorial representation (b) Textual representation

5.1 The search space

Since many feasible mappings exist and since we use a constructive approach, we need to define the search space for our mapping problem. The search space is built by applying different sets of valid mapping rules for the mapping problem. Each of the algorithms mentioned in this section goes though the partial solutions and finally leads to a complete solution. We introduce the search space for the mapping example discussed in the last section. It is described by the tree shown in Figure 15. The leaves of this tree represent a complete solution whereas internal nodes represent partial solutions. At the root of the tree, we have a null partial solution. At each level, a function in f(S) is selected and all the mapping rules for this function are explored. An internal node represents the partial solution using the rules that are on the path from the root to this node. Figure 15 shows some of the partial solutions along with a few complete solutions. Some of the algorithms (1-greedy and m-greedy) could be illustrated using this tree of the search space. Note that a trivial way of finding an optimal solution is the exhaustive method that generates all the leaf nodes and selects the best mapping. Of course, this is a very time-inefficient solution.

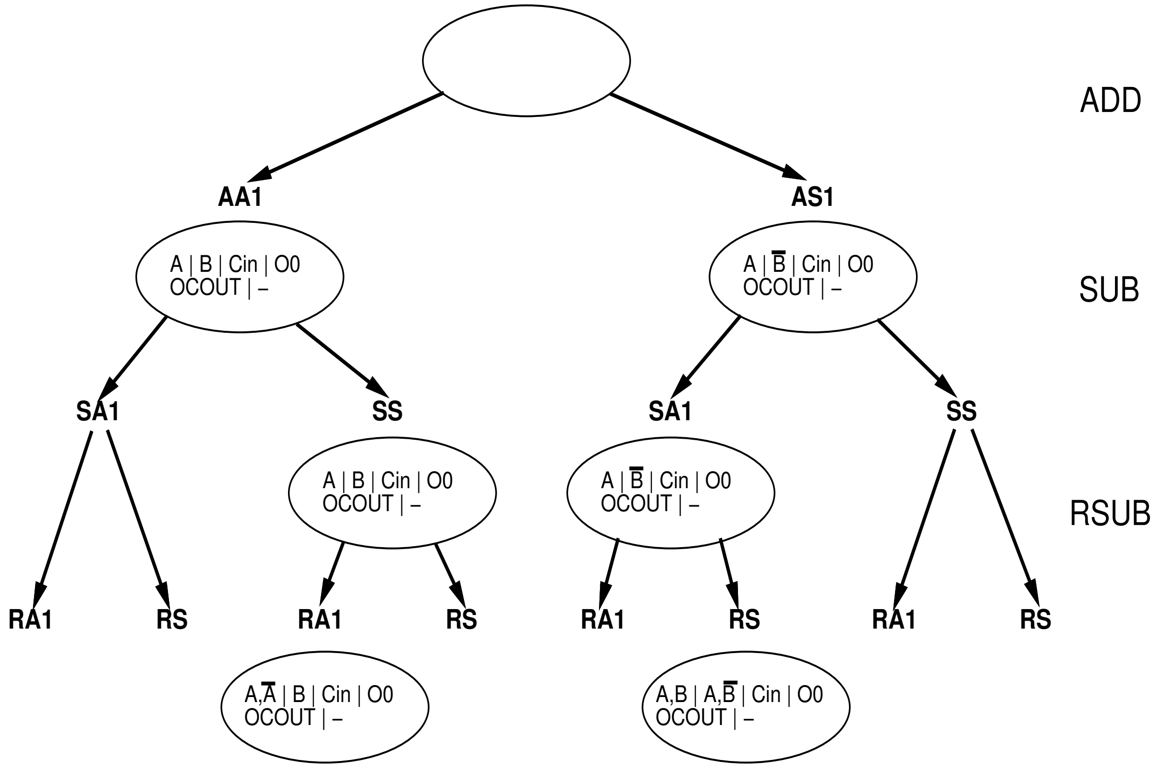


Figure 15: The search space for an example

We now discuss the actual algorithms used to solve the mapping problem. Note that the algorithms discussed here are independent of the cost functions and that these algorithms could be applied with either of the cost functions discussed before.

5.2 1-Greedy algorithm

Algorithm 5.1 : 1-Greedy algorithm

INPUT: f(S), f(T), r(ST).

OUTPUT: A set of rules, one for each function in f(S), with minimum cost.

1. $f_s(S)$ = sorted f(S) in increasing number of mapping rules.
2. rule-set = ϕ.
3. **while** $\exists$ an unmapped function $\in f_s(S)$ **loop**

   3.1 F = first function $\in f_s(S)$.

   3.2 r = rule that implements F with minimum additional cost.

   3.3 rule-set = rule-set + r.

   3.4 Remove F from $f_s(S)$.
4. **end loop**
5. **return** rule-set.

The 1-Greedy algorithm starts with a null partial solution. It first sorts all the source functions in increasing number of mapping rules. Next it selects a source function at a time and generates a set of new partial solutions, one for each rule for the selected function. Each of these new partial solutions has a feasible mapping for all the source functions considered so far. Out of these partial solutions, the algorithm chooses the one with minimum cost. This process is repeated until it has mapped all the source functions onto some target function and we have a complete solution. Figure 16 shows an application of this algorithm. Note that at each step (each source function), we keep track of the single best solution out of all the partial solutions generated.

5.2.1 Complexity analysis

The algorithm spends most of its time in the loop at statement 3. It repeats itself for nS times. At each iteration, we generate mST partial solutions and then we choose the best solution among these. Assuming creation of partial solution takes constant time, the complexity of the *1-greedy* algorithm is $O(mST * nS)$.

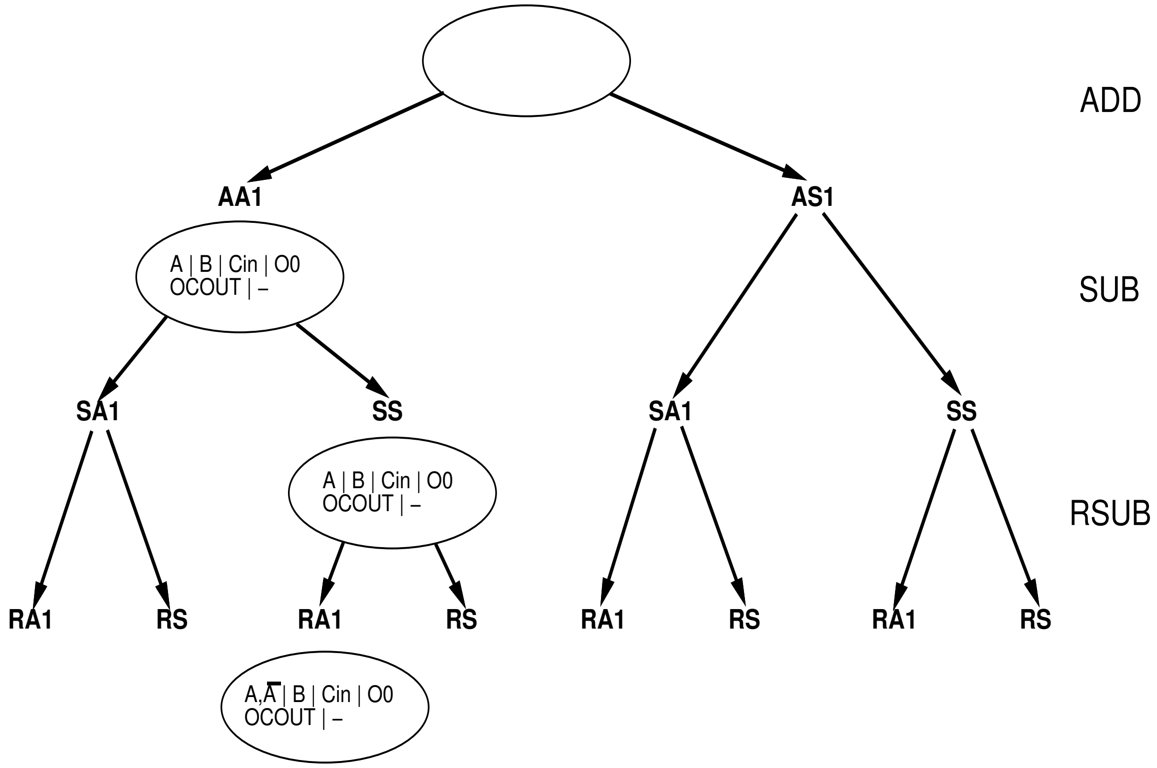


Figure 16: Execution of 1-greedy algorithm on an example

5.3 k-Greedy algorithm

Algorithm 5.2 : k-Greedy algorithm

INPUT: f(S), f(T), r(ST), k.

OUTPUT: A set of rules, one for each function in f(S), with minimum cost.

1. $f_s(S)$ = sorted f(S) in increasing number of mapping rules.
2. par-sol = ϕ.
3. **while** $\exists$ an unmapped function $\in f_s(S)$ **loop**

   3.1 F = first function $\in f_s(S)$.

   3.2 par-sol = k-best solution from par-sol and implements F.

   3.3 Remove F from $f_s(S)$.
4. **end loop**
5. **return** min(par-sol).

The k-Greedy algorithm is very similar to 1-greedy except for the fact that in each step we keep track of k solutions instead of 1. Figure 15 shows an application of this algorithm with k = 2. Note that at each step we are keeping track of 2 partial solutions.

5.3.1 Complexity analysis

In this algorithm, each iteration of loop at statement 3, generates $k * mST$ solutions and we have to choose k best solutions out these. This would take O($k * mST$) time. Thus, the complexity of the whole algorithm is O($mST * nS * k$).

5.4 Dynamic programming algorithm

The Dynamic Programming (DP) technique performs a better global search as compared to greedy algorithms [CoLR90]. Instead of making decisions based only on the mapping of the current function, it keeps track of the best solutions for all subsets of functions. Typically, it works in a bottom-up fashion. The algorithm starts with the mapping for subsets of single functions, followed by mapping for subsets of two functions and so on till it has the mapping for the entire set of source functions. Each of the partial mapping for a subset of functions is stored in a table and subsequently used for building the partial solutions for subsets of bigger size.

Algorithm 5.3 is a dynamic programming algorithm that keeps track of k (bucket size) best partial solutions. Note that number of partial solutions increases exponentially with the size of function set. We restrict ourselves to a limited number(k) of partial solutions for each subset. Of course, by doing so we may sacrifice optimality with the advantage of requiring bounded storage space.

Algorithm 5.3 : Dynamic programming (DP) algorithm

INPUT: f(S), f(T), r(ST), k.

OUTPUT: A set of rules, one for each function in f(S).

1. Order f(S).
2. **for** i = 0 to nS-1 **do**
   2.1 Table[i,i] = k-best way of mapping f_i.
3. **end for**
4. **for** i = 1 to nS-1 **do**
   4.1 **for** j = 1 to (nS-i) **do**
      4.1.1 CreateEntry(j-1, i+j-1).
   4.2 **end for**
5. **end for**

Algorithm 5.4 : CreateEntry(row,col)

1. $min_k = 0$.
2. **for** i = 0 to (col-row-1) **do**
   2.1 min_k = k-best of combine(table(row,row+i), table(row+i+1,col)).
3. **end for**

As mentioned before, the dynamic programming algorithm builds up a table of partial solutions in a bottom-up fashion. This table is indexed by the number of source functions(nS) for both row and column. An entry $table(i, j)$ represents a set of partial solutions for source function i to source function j. Figure 17 shows the table with the bucket size of 2 for our walk-through example. Note that the table is upper-triangular.

This algorithm iteratively fills up the table with partial solutions. It starts by filling diagonal entries by generating the mapping for single functions. Each diagonal entry represents a set of partial solutions that map exactly one source function. Then it fills up the entries corresponding to two function sets and so on. The final solution is given by the top-row and right-most column. For our example shown in Figure 17, $table(0,2)$ lists few solutions that represent the complete mapping. Function $CreateEntry$ creates the list of partial solutions for a set of source functions using the partial solutions generated so far.

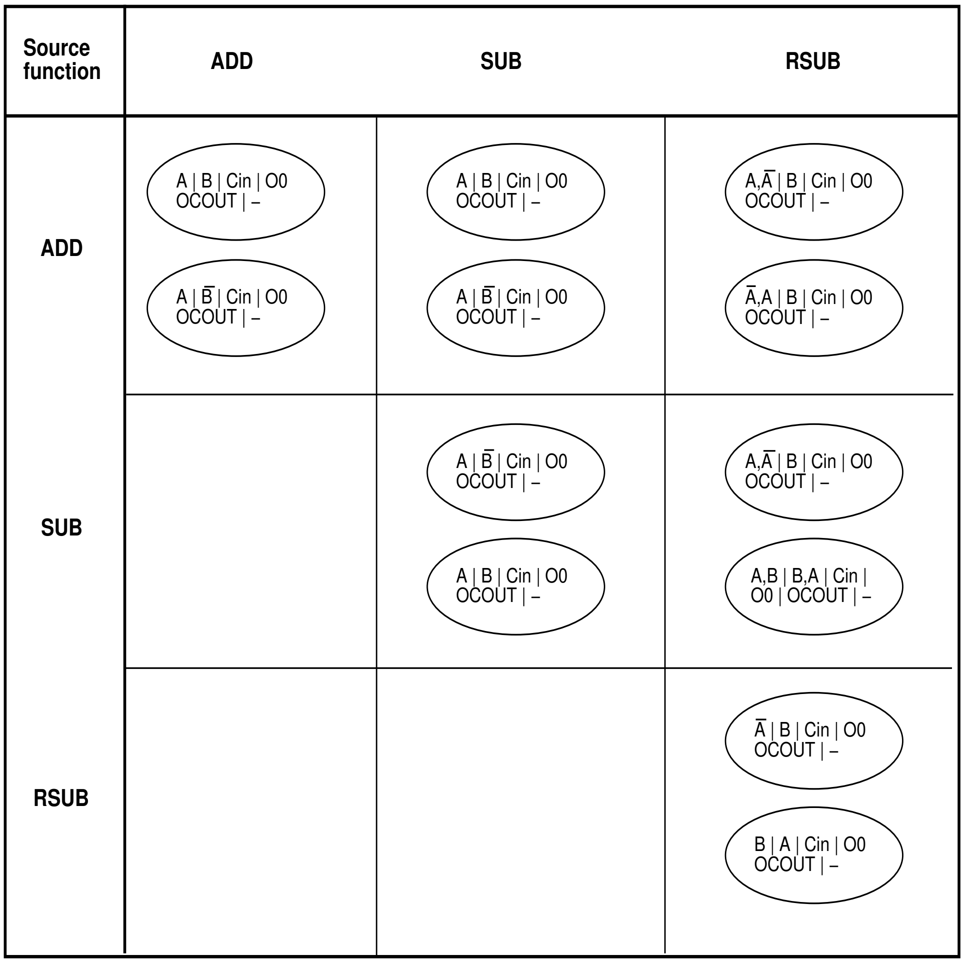


Figure 17: Execution of dynamic programming algorithm (bucket size = 2) on an example

5.4.1 Complexity analysis

Algorithm 5.4 has a loop with worst case iteration count of nS. Each iteration of this loop takes $O(k^2)$ time. Algorithm 5.3 has a doubly nested loop at statement 4. This loop runs for $O(nS^2)$ times and each iteration takes $O(k^2)$ time. Thus, the complexity of the whole algorithm is $O(nS^3k^2)$.

5.5 Graph clustering algorithm

The Graph clustering algorithm is a branch and bound algorithm based on the graph formulation of the mapping problem. Let us define a graph G = (V, E) with V being the set of vertices and E the set of edges.

V : {v | v is a rule to map a function in S to a function in T}.

p(v) : represents the partial solution associated with the node.

E : {e = (v_1, v_2)| There is a possibility that the two sets of rules associated with v_1 and v_2 are used together}. Note that we can't have an edge between two vertices that have rules for mapping the same function in S, since we can't use more than one rule to map a function.

w(e) : Let e = (v_1, v_2). Then w(e) is the "cost" (gate-count/max-delay) of the partial solution that would be generated if v_1 and v_2 are combined.

Figure 18 shows a subgraph with two nodes v_1 and v_2. The partial solutions associated with each of the two nodes and the edge-weight (for gate-count) are also shown in this figure.

Given the above graph G, we define the following function to combine two connected vertices v_1 and v_2. This function creates a new vertex v and generates new edges that connect v with other vertices in the graph.

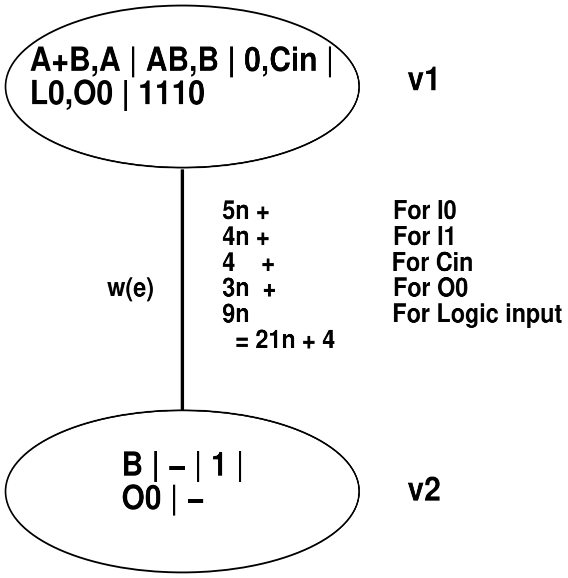


Figure 18: Edge weight example

Algorithm 5.5 : Combine(v_1, v_2)

1. Create a new node v.
2. p(v) = $p(v_1) \cup p(v_2)$.
3. Add v to G.
4. Delete edge (v_1, v_2).
5. $\forall v_k | \exists (v_1, v_k)$ and (v_2, v_k)

   5.1 Create an edge $e = (v, v_k)$.

   5.2 Calculate w(e).

   5.3 Insert e in the edge list.
6. Return v.

Now we define the actual cluster formation algorithm. The algorithm first builds an initial graph by generating a set of nodes, one for each mapping rule. It then connects each pair of nodes that do not map the same source function. In the next step, the algorithm sorts all the edges in the increasing order of their weights. Starting with this initial graph, the algorithm selects the first edge (with the smallest edge weight) and merges the corresponding pair of vertices. The edge-list is resorted to take care of newly created edges. This graph clustering is repeated till we have a node representing a complete solution.

Algorithm 5.6 : Graph cluster formation

INPUT: f(S), f(T), r(ST).

OUTPUT: A set of rules, one for each function in f(S), with minimum cost.

1. $\forall r \in f(ST)$ create a node.
2. $\forall(v_1, v_2)|f(v_1) \cap f(v_2) = \phi$
   2.1 Create an edge e.
   2.2 Calculate w(e).
3. Create edge-list by sorting all the edges.
4. done = False;
5. **While** not done
   5.1 **if** (edge-list = ϕ) **then** done = True; exit;
   5.2 Let e = (v_1, v_2) be the first edge.
   5.3 v = Combine(v_1, v_2).
   5.4 If v has one rule for each function in f(S)
      5.4.1 done = True.
      5.4.2 Return p(v).
6. **endWhile**
7. **Return** ϕ.

Figures 19 and 20 illustrate various stages of application of this algorithm on our walk-through example. Recall that a node in these figures represent a partial solution; an arc signifies that the corresponding partial solutions could be combined together to get a new partial solution. The arc cost represents the cost of the partial solution that will be generated by combining the two nodes. The 'n' in the cost expressions represents bit-width. The name associated with a node represents a set of mapping rule(s) that has been used to create a

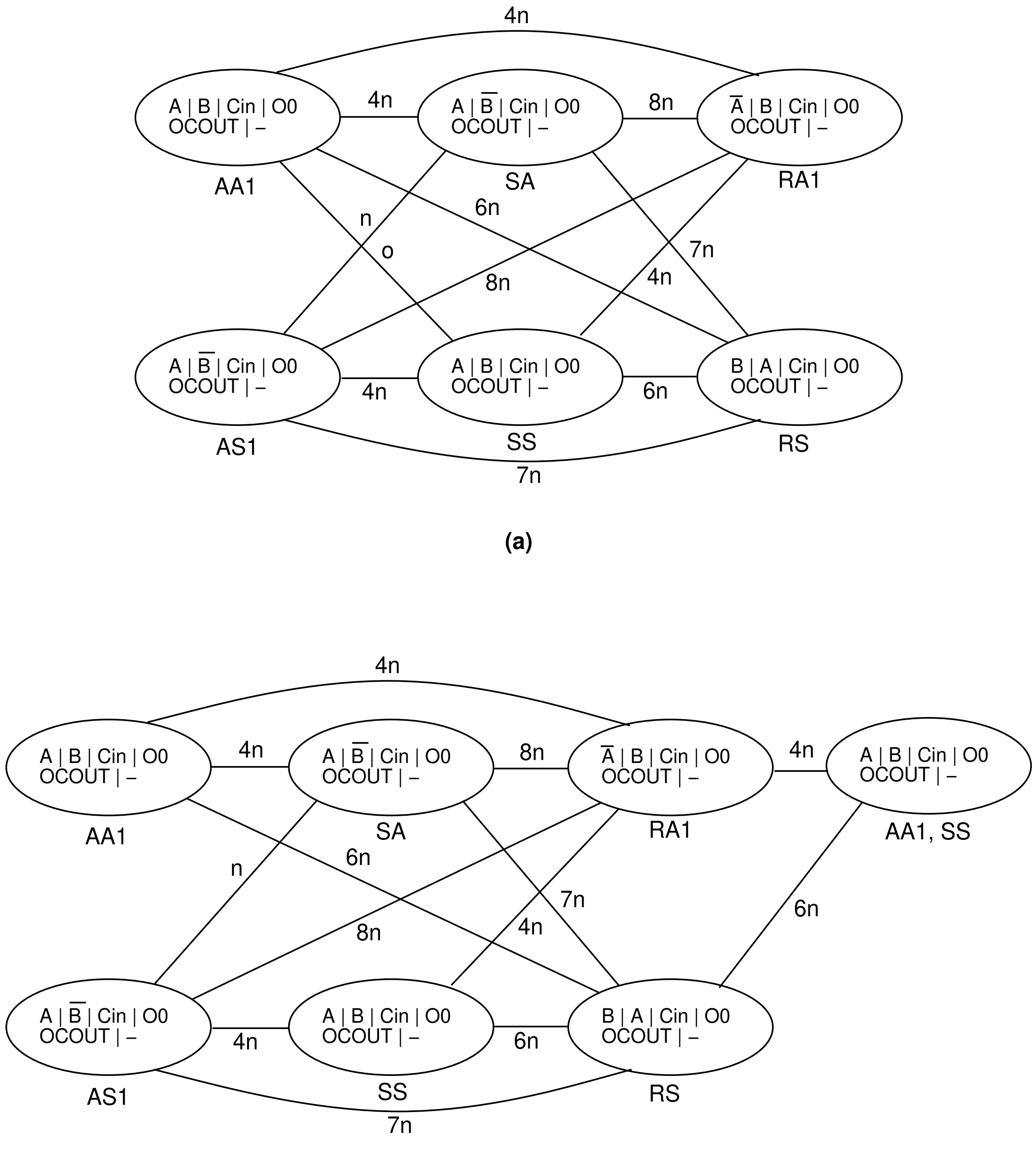


Figure 19: Application of graph clustering algorithm on an example: (a) Stage 1 (b) Stage 2

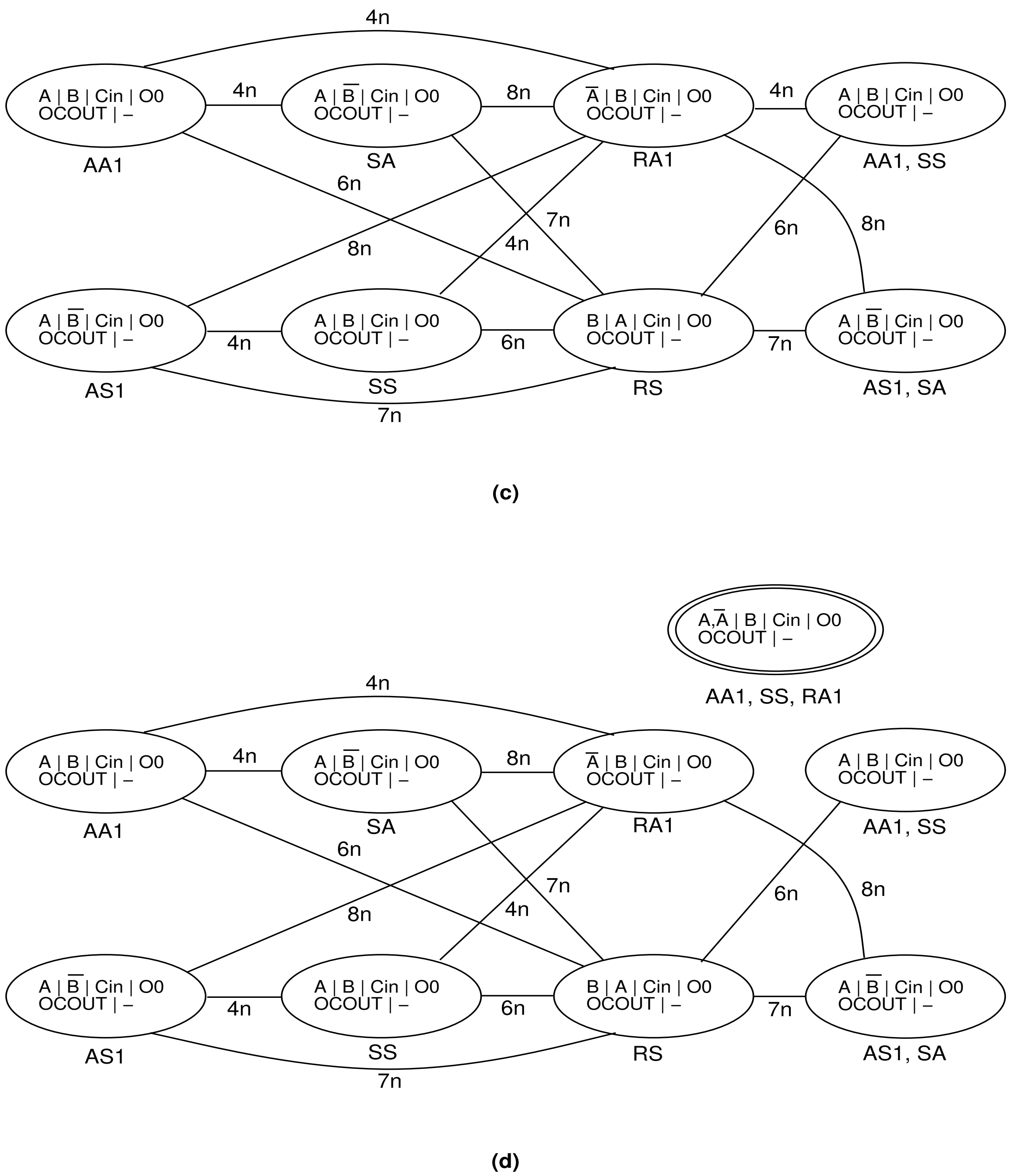


Figure 20: Application of graph clustering algorithm on an example (continued): (c) Stage 3 (d) Final stage

partial solution.

The first stage (Figure 19(a)) consists of 6 nodes (partial solutions), one for each mapping rule. In the next step (Figure 19(b)), we combine the two partial solutions corresponding to rules AA1 and SS as the arc connecting these partial solutions has minimum cost(0). A new node (AA1,SS) is created and new sets of edges (with cost) between this node and preexisting nodes (RA1 and RS) are created. We delete the arc between AA1 and SS. Next(Figure 20(c)), we combine nodes AS1 and SA and generate a new node AS1,SA. The resulting graph has 6 edges with minimum cost '4n'. Out of these we choose the edge between nodes RA1 and (AA1,SS), since the resulting node will have more functions implemented. The next stage is shown in Figure 20(d). It so happens that the node (AA1,SS,RA1) implements all the source functions; thus we have a complete solution.

5.5.1 Complexity analysis

The graph-clustering algorithm is a combination of the branch-and-bound and the dynamic programming techniques. Unlike previous algorithms this algorithm guarantees an optimal solution. In the worst case, we may have to consider all possible combinations of source functions and mapping rules. The number of such combinations are exponential with respect to number of source functions nS.

6 Experimental Results

We performed two sets of experiments to validate our approach. The first set of experiments tests the comprehensiveness of the approach in terms of mapping different arithmetic components. The second set of experiments compares the metrics of the designs generated by our approach against the ones generated by previous approach, namely the logic synthesis approach. First, we demonstrate the comprehensiveness of our approach, followed by the comparative study of design quality.

6.1 Comprehensiveness

Example	Source component			Target component		
	Library	Fns	Width	Library	Fns	Width
1.	GENUS	ADD,SUB,EQ,LT	32	VDP	ADD,SUB	32

Algorithm	Bucket size	Cost fn	Result		Run time (second)	Optimal	
			GC (2–input)	Delay (ns)		GC	MD
Greedy	1	GC	37	56.40	5.1	yes	yes
Greedy	1	MD	37	56.40	4.8	yes	yes
Dyn Prog	1	GC	37	56.40	4.9	yes	yes
Dyn Prog	2	GC	37	56.40	5.5	yes	yes
Dyn Prog	4	GC	37	56.40	8.1	yes	yes
Dyn Prog	1	MD	37	56.40	5.0	yes	yes
Dyn Prog	2	MD	37	56.40	5.4	yes	yes
Dyn Prog	4	MD	37	56.40	8.1	yes	yes

Figure 21: Design metrics for example 1

In this section, we present experimental results that establish the generality of our approach across different source and target libraries using algorithms discussed in the last section. Specifically, we list design metrics for sample ALUs mapped by two algorithms: *1-Greedy* and *Dynamic programming*. These algorithms were run with two cost functions: gate-count(GC) and max-delay(MD) as optimization criteria. Recall that gate-count is an approximation of extra hardware required to implement the source ALU, whereas max-delay represents the maximum delay though all the ports of the generated design.

We considered a variety of ALUs, both source and target, in our experiments. These ALUs vary in terms of the library they come from, number and the set of functions they perform. We present the mapping results for ALUs from four libraries : GENUS[Dutt91], CASCADE[Casc92], VDP300[VTI91] and AMD[Am2901]. GENUS contains an ALU gen-

Example	Source component			Target component		
	Library	Fns	Width	Library	Fns	Width
2.	GENUS	ADD,SUB,RSUB, INC,DEC,	32	CASCADE	ADD	32

Algorithm	Bucket size	Cost fn	Result		Run time (second)	Optimal	
			GC (2–input)	Delay (ns)		GC	MD
Greedy	1	GC	387	24.50	4.1	yes	yes
Greedy	1	MD	387	24.50	4.2	yes	yes
Dyn Prog	1	GC	389	24.50	4.3	no	yes
Dyn Prog	2	GC	389	24.50	5.5	no	yes
Dyn Prog	4	GC	387	24.50	9.9	yes	yes
Dyn Prog	1	MD	389	24.50	4.5	no	yes
Dyn Prog	2	MD	389	24.50	5.6	no	yes
Dyn Prog	4	MD	387	24.50	9.8	yes	yes

Figure 22: Design metrics for example 2

erator parametrized by the set of functions, bit-width, etc. The ALUs in CASCADE and VDP300 have a fixed set of functions. The AM2901 ALU is a commonly used 8-function ALU.

We covered a wide range of ALUs in terms of the number of functions they perform. Starting from a simple uni-function adder, the most complex ALU had 32 functions. Note that an ALU, as we have defined, can have only 27 distinct canonical functions. Thus, some ALUs in our experiments have variants of the canonical functions repeated in the function set. For example, one of the ALUs in our experiments has 9 ADD functions, each with different port configurations. The ALUs in our experiments perform different sets of functions, covering different functional categories: arithmetic, logic and comparison. We also chose ALUs of different bit-widths.

Note that the examples in our experiments were restricted by the availability of design

Example	Source component			Target component		
	Library	Fns	Width	Library	Fns	Width
3.	GENUS	ADD,SUB,AND, NAND,OR,NOR, XOR,XNOR,LID, RID,LNOT,RNOT, ONE,NEQ,GT,	48	VDP300	ADD,SUB,RSUB, AND,NAND,OR, NOR,XNOR, RIMPL,LIMPL, LINHI,RINHI	48

Algorithm	Bucket size	Cost fn	Result		Run time (second)	Optimal	
			GC (2–input)	Delay (ns)		GC	MD
Greedy	1	GC	630	72.10	8.5	NK	NK
Greedy	1	MD	630	72.10	8.3	NK	NK
Dyn Prog	1	GC	726	75.30	23.2	NK	NK
Dyn Prog	2	GC	630	72.10	80.6	NK	NK
Dyn Prog	4	GC	630	72.10	667.1	NK	NK
Dyn Prog	1	MD	1206	71.30	21.8	NK	NK
Dyn Prog	2	MD	1110	68.10	73.6	NK	NK
Dyn Prog	4	MD	1110	68.10	604.3	NK	NK

Figure 23: Design metrics for example 3

metrics and not by the limitations of our approach. For example, the delay for target components in a library were available only for specific bit-widths such as 8, 16, 48. Thus, all our experiments are for ALUs with one of the above bit-widths. Recall that our approach is independent of bit-widths and that it will require same amount of computation for all bit-widths. Similarly, we had to restrict ourselves to only those libraries that provide design metrics. Even though we considered other libraries such as XBLOX[XBLO92], LSI[LsiLogic], Toshiba gate array[Tosh90] etc., we could not run our algorithms due to lack of metric data (gate counts, performance) for these libraries. We also had to restrict our mapping examples to ALUs with fixed sets of functions, since these are the only ALUs supported by some of these libraries.

Figures 21 through 27 summarize the designs generated by our mapping algorithms for seven examples. These tables are in two parts; the top part contains the component de-

Example	Source component			Target component		
	Library	Fns	Width	Library	Fns	Width
4.	AMD	ADD,SUB,RSUB, OR,AND,LINHI, XOR,XNOR	16	VDP300	ADD,SUB	16

Algorithm	Bucket size	Cost fn	Result		Run time (second)	Optimal	
			GC (2–input)	Delay (ns)		GC	MD
Greedy	1	GC	432	33.20	4.4	yes	no
Greedy	1	MD	464	32.40	4.4	no	yes
Dyn Prog	1	GC	464	32.40	6.5	no	yes
Dyn Prog	2	GC	432	33.20	11.1	yes	no
Dyn Prog	4	GC	432	33.20	21.4	yes	no
Dyn Prog	1	MD	464	32.40	6.4	no	yes
Dyn Prog	2	MD	464	32.40	11.1	no	yes
Dyn Prog	4	MD	464	32.40	22.0	no	yes

Figure 24: Design metrics for example 4

scriptions and the bottom part describes the result. The component description includes example number followed by description of the source and the target component. Each of the component description includes the library name, set of functions that a given ALU can perform and the bit-width of the ALU.

The result section(bottom part) describes the design metrics and the run time for each of the two algorithms: *1-Greedy* and *dynamic programming*. Both the algorithms were run with two cost functions: *gate-count* (GC) and *max-delay* (MD). The *Dynamic Programming* algorithm was tried with various bucket sizes in the range of 1 to 4.

For each combination of algorithm, bucket size and cost function, we report GC, MD, run-time and whether the design is optimal or not. The gate-count (GC) is the approximate number of extra 2-input gates that will be required to build the source component using target component. Delay represents max-delay in nanoseconds, i.e., this is the maximum

Example	Source component			Target component		
	Library	Fns	Width	Library	Fns	Width
5.	GENUS	ADD,SUB,AND, NAND,OR,NOR, XOR,XNOR,LID, RID,LNOT,RNOT, ONE,NEQ,GT,	16	CASCADE	ADD,SUB,RSUB, INC,DEC,ZERO, ONE,AND,NAND, OR,NOR,XOR, XNOR,RIMPL, LIMPL,LINHI, RINHI,LID,RID	16

Algorithm	Bucket size	Cost fn	Result		Run time (second)	Optimal	
			GC (2–input)	Delay (ns)		GC	MD
Greedy	1	GC	134	31.40	7.5	NK	NK
Greedy	1	MD	278	28.20	8.1	NK	NK
Dyn Prog	1	GC	134	31.40	20.9	NK	NK
Dyn Prog	2	GC	134	31.40	63.6	NK	NK
Dyn Prog	4	GC	134	31.40	572.7	NK	NK
Dyn Prog	1	MD	278	28.20	21.7	NK	NK
Dyn Prog	2	MD	278	28.20	64.6	NK	NK
Dyn Prog	4	MD	278	28.20	498.0	NK	NK

Figure 25: Design metrics for example 5

delay through all the input-output port combination of the resultant design. The run time column shows execution time (user + system) for the given example on Sparc 2 (sun-670-mp for examples 6 and 7). The last two columns tells whether the generated design is optimal with respect to gate-count and max-delay respectively. Note that an 'NK' (Not Known) in these two columns means that we don't know whether the design is optimal.

As mentioned before, the seven examples in our experiments are from different libraries and are of varying complexity. Example 1 maps a GENUS ALU with 2 arithmetic and 2 comparison functions onto a VDP ADD-SUB component. Example 2 implements a GENUS ALU with all the arithmetic functions on CASCADE adder. The third example maps an-

Example	Source component			Target component		
	Library	Fns	Width	Library	Fns	Width
6.	CASCADE	ADD(0),ADD(1), ADD(2),ADD(3), ADD(4),ADD(5), ADD(6),ADD(7), ADD(8),SUB, INC(0),INC(1), INC(2),DEC(0), DEC(1),DEC(2), ZERO,ONE,AND, NAND,OR,NOR, XOR,XNOR, RIMPL,LIMPL, LINHI,RINHI,LID, RID,LNOT,RNOT	16	VDP300	ADD,SUB,RSUB, AND,NAND,OR, NOR,XNOR, RIMPL,LIMPL, LINHI,RINHI	16

Algorithm	Bucket size	Cost fn	Result		Run time (second) *	Optimal	
			GC (2–input)	Delay (ns)		GC	MD
Greedy	1	GC	453	36.20	16.3	NK	NK
Greedy	1	MD	453	36.20	15.7	NK	NK
Dyn Prog	1	GC	661	39.40	453.7	NK	NK
Dyn Prog	2	GC	453	36.10	5322.1	NK	NK
Dyn Prog	1	MD	917	42.50	491.2	NK	NK
Dyn Prog	2	MD	485	36.20	4558.4	NK	NK

* On sun–670–mp

Figure 26: Design metrics for example 6

other GENUS ALU with functions covering all the three categories (arithmetic, logic and comparison) onto a VDP ALU that covers some of the arithmetic and logic functions. The next example maps the AM2901 ALU with 3 arithmetic and 5 logic functions onto a VDP adder-subtracter. The fifth example maps the same source component as in Example 3, but this time the target component is a CASCADE ALU that covers some of the arithmetic and all the 16 logic functions. Examples 6 and 7 use same source component: a complex ALU from CASCADE that has 32 functions with many repetitions of same canonical function. In Example 6, the target component is a VDP ALU whereas in Example 7 the target component is the AM2901 ALU. Appendix B describes each of the output designs for these examples.

Example	Source component			Target component		
	Library	Fns	Width	Library	Fns	Width
7.	CASCADE	ADD(0),ADD(1), ADD(2),ADD(3), ADD(4),ADD(5), ADD(6),ADD(7), ADD(8),SUB, INC(0),INC(1), INC(2),DEC(0), DEC(1),DEC(2), ZERO,ONE,AND, NAND,OR,NOR, XOR,XNOR, RIMPL,LIMPL, LINHI,RINHI,LID, RID,LNOT,RNOT	16	AMD	ADD,SUB,RSUB, ORAND,LINHI, XOR,XNOR	16

Algorithm	Bucket size	Cost fn	Result		Run time (second) *	Optimal	
			GC (2–input)	Delay (ns)		GC	MD
Greedy	1	GC	757	–	13.2	NK	NK
Dyn Prog	1	GC	933	–	506.2	NK	NK
Dyn Prog	2	GC	757	–	5500.7	NK	NK

* On sun–670–mp

Figure 27: Design metrics for example 7

6.1.1 Analysis of results

As previously mentioned, Figures 21 through 27 describe the design metrics for the mapped ALUs. For each example, we report the output (optimized for gate-count and delay) from greedy algorithm. From these tables, we observe that the greedy algorithm provides a quick way of mapping a source ALU onto a target ALU. The greedy algorithm often produces an optimal solution (Examples 1, 2 and 4). The run-time for the greedy algorithm is in the range of 4.1 to 16.3 seconds.

We also have mapping results from the dynamic programming algorithm. This algorithm was tried with various bucket size in the range of 1 to 4. Recall that the bucket size refers to the number of partial solutions for each table entry. Increasing the bucket size increases the likelihood of obtaining an optimal solution. The dynamic programming algorithm with infinite bucket size is guaranteed to produce an optimal design.

We have run the dynamic programming algorithm for two cost functions: gate-count and max-delay. From the tables in Figures 21 through 22, we observe that as the bucket size increases, we tend to get better solutions. A bucket size of 2 is often sufficient to produce a good design. Note that this algorithm has been able to produce better results as compared to the greedy approach. For example, refer to the results for Example 3 shown in Figure 23. The designs generated by this algorithm with delay optimization have lower delay value (71.30 and 68.10) as compared to the delay (72.10) of the designs generated by the greedy algorithm. Of course, better designs come at the cost of longer run-times.

Note that we have used a restricted set of rules to map a source function onto a target function. Appendix A shows the list of rules that has been used in our experiments. These set of rules are not complete, particularly with respect to rules for mapping logic functions. These algorithms may generate even better results, if provided with a comprehensive set of rules.

In summary, our approach is quite versatile in the sense that it can map ALUs from one library onto another. We have demonstrated the versatility of our approach by applying it on ALUs from wide variety of libraries. Also, our approach can handle ALUs with diverse complexity in terms of bit-width, number of functions and set of functions. Our algorithms generate designs of high quality, often optimal designs. Finally, we have provided two algorithms: the greedy algorithm can be used for quick results, whereas the dynamic programming generates better designs at the expense of longer run-times.

6.2 Goodness

Now we present the results our experiments that compare metrics of the designs produced by our approach against the ones produced by a commercial logic synthesis tool[Syno92]. Figure 28 describes the experimental set-up. For each source component, we first map this component onto a macro in the Synopsys DesignWare library[Degn92], with additional glue logic using our approach. Then we pass the resultant design through logic synthesis to

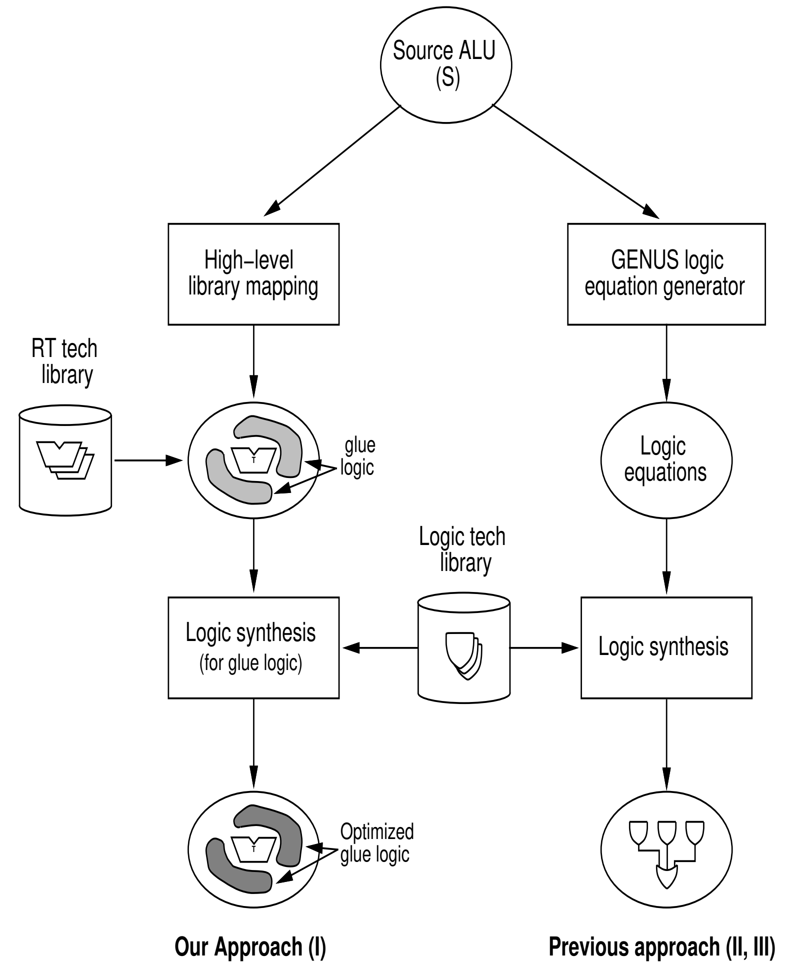


Figure 28: Experimental set-up for comparative study

optimize the glue logic. We present metrics for the designs (I) arrived by this path (our approach). The left portion of Figure 28 illustrates this path. In the next phase of the experiment, we generate the logic equations for the input source component and directly map onto gate-level cells using the logic synthesis approach. Logic synthesis is tried with two levels of optimizations: low and medium. The designs generated by the low optimization script are refered to as (II) and those generated by the medium optimization script are refered to as (III). For each of these designs, we present design metrics: gate-count, max-delay and run-time and compare them against the metrics for designs generated by our approach (I).

For each input source component, we perform two sets of experiments. In the first,

Ex	Source component (GENUS)	Target component (Designware)	Bw/ Style	Metrics	Our approach I	Logic synthesis (Synopsys) II	%diff	III	%diff
1	ADD,SUB, RSUB	ADD,SUB	32/ cla	Area(2−imp)	432	1735	**302**	1735	**302**
				Delay(ns)	42.10	122.74	**191**	122.74	**191**
				Run−time(min) (alg+map)	.11+1.25	10.02	**637**	11.67	**758**
2	ADD,SUB, RSUB,INC, DEC	ADD	27/ ripple	Area(2−imp)	418	1161	**178**	1159	**177**
				Delay(ns)	49.94	65.27	**31**	65.66	**31**
				Run−time(min) (alg+map)	.09+3.33	4.45	**30**	7.43	**117**
3	ADD,SUB, RSUB,OR, AND,LINHI, XOR,XNOR	ADD,SUB	32/ cla	Area(2−imp)	726	2489	**243**	2489	**243**
				Delay(ns)	59.34	122.14	**106**	122.14	**106**
				Run−time(min) (alg+map)	.19+5.90	10.35	**70**	22.20	**264**
4	ADD,SUB, AND,NAND, OR,NOR, XOR,XNOR, LID,RID, LNOT,RNOT, ONE,NEQ,GT	ADD,SUB	16/ cla	Area(2−imp)	369	1236	**235**	1236	**235**
				Delay(ns)	29.24	75.79	**159**	75.79	**159**
				Run−time(min) (alg+map)	1.2+3.31	13.03	**189**	22.04	**389**

I : low optimization   II : low optimization   III : medium optimization

Area : in 2−input gates   Delay : in nanoseconds   Run−time : in minutes on sparc2

Figure 29: Our approach verses logic synthesis: **optimized for area**

we generate designs that are optimized for area (minimum gate-count). For this set of experiments, both the logic synthesis tool and our algorithm are configured to generate best area designs. In the other set of experiments, we configured our algorithm and logic synthesis tool to optimize the worst case delay. We present designs metrics for each of these experimental sets.

Figure 29 describes metrics for designs that are optimized for area (in terms of gate-count). This table describes four examples; each example uses different source component that are mapped onto various macros in the DesignWare library [Degn92]. Columns 2 through 4 describe the source and target components. The sixth column in this figure

Ex	Source component (GENUS)	Target component (Designware)	Bw/ Style	Metrics	Our approach I	Logic synthesis (Synopsys) II	%diff	III	%diff
1	ADD,SUB, RSUB	ADD,SUB	32/ cla	Area(2−imp)	962	2450	**154**	2296	**139**
				Delay(ns)	11.68	31.92	**173**	13.60	**16**
				Run−time(min) (alg+map)	0.11+13.90	19.00	**36**	38.20	**173**
2	ADD,SUB, RSUB,INC, DEC	ADD	27/ ripple	Area(2−imp)	421	2473	**487**	1502	**257**
				Delay(ns)	36.53	64.85	**78**	32.00	**−12**
				Run−time(min) (alg+map)	0.09+6.03	22.33	**265**	47.98	**684**
3	ADD,SUB, RSUB,OR, AND,LINHI, XOR,XNOR	ADD,SUB	32/ cla	Area(2−imp)	1237	3285	**165**	3313	**168**
				Delay(ns)	14.27	30.57	**114**	14.39	**1**
				Run−time(min) (alg+map)	0.19+17.40	27.00	**53**	55.48	**215**
4	ADD,SUB, AND,NAND, OR,NOR, XOR,XNOR, LID,RID, LNOT,RNOT, ONE,NEQ,GT	ADD,SUB	16/ cla	Area(2−imp)	543	2506	**361**	2244	**313**
				Delay(ns)	12.88	17.32	**34**	14.90	**16**
				Run−time(min) (alg+map)	1.25+11.80	22.31	**71**	40.90	**213**

I : low optimization   II : low optimization   III : medium optimization

Area : in 2−input gates   Delay : in nanoseconds   Run−time : in minutes on sparc2

Figure 30: Our approach verses logic synthesis: **optimized for delay**

reports area, delay and run-time for the designs generated by our approach (I). In this table, gate-count is measured in 2-input generic gates, delay is in nanoseconds and run-time is in minutes. Note that the run-time for the designs generated by our approach is the sum of two numbers: the first number is the time taken by our algorithm to perform mapping and the second number represents the time taken by the logic synthesis tool to optimize the glue logic and map the macro onto the target technology. The last two columns in Figure 29 present metrics for the designs produced by logic synthesis approach with low(II) and medium(III) optimization effort. These two columns also report percentage difference between design metrics for II and III as compared to I. Figure 30 compares metrics for the

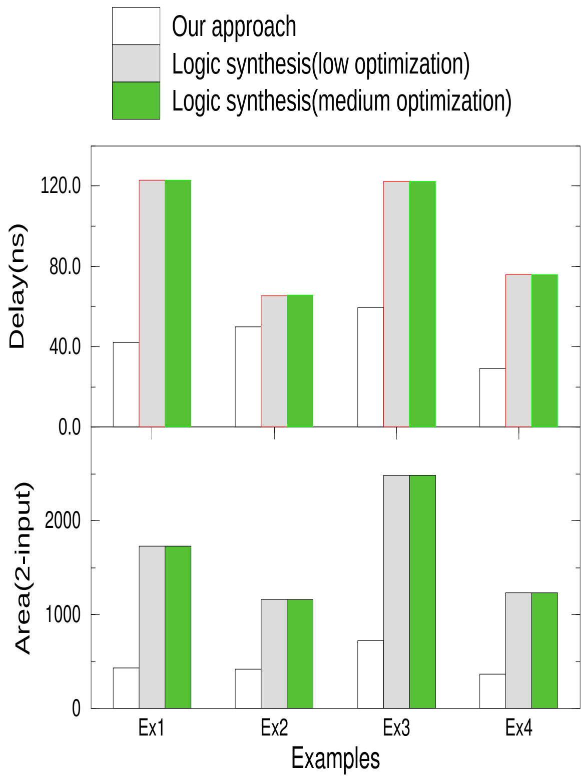


Figure 31: Pictorial representation of the comparative study: **performance metrics for area-optimized designs**

designs optimized for delay.

Figures 31 through 34 graphically present the experimental results shown in Figures 29 and 30. Figures 31 and 33 illustrate area and delay for area-optimized and delay-optimized designs respectively. There are three columns for each design in these two figures. These three columns represent performance metrics for the designs generated by the three approaches I, II and III. Figures 32 and 34 depict run-time for area-optimized and delay-optimized designs respectively. Note that there is one extra column for each design in these two figures. The runtime for generating designs with our approach is split into two columns. The first column represents runtime for mapping algorithm and the second column represents runtime for optimizing the glue logic.

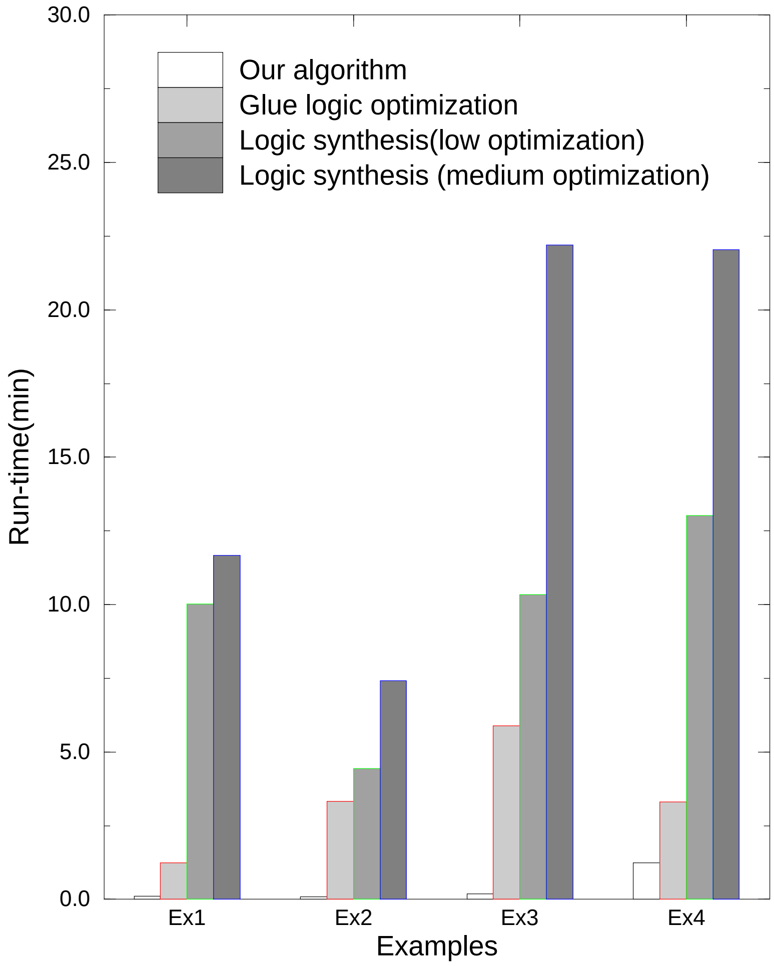


Figure 32: Pictorial representation of the comparative study: **run-time for area-optimized designs**

6.2.1 Analysis of results

Figures 29 and 30 lists the percentage difference between the designs generated by our approach (I) as compared to the ones generated by logic synthesis (II,III). In these two figures, we observe that except for one (delay for design 2 generated by approach III in figure 30), all these percentages are positive. This means that **designs generated by our approach(I) are better with respect to all the three metrics: gate-count, delay and runtime.** Designs generated by our approach are not just better, but substantially better than the ones generated by the logic synthesis tools. This phenomenon is easily seen

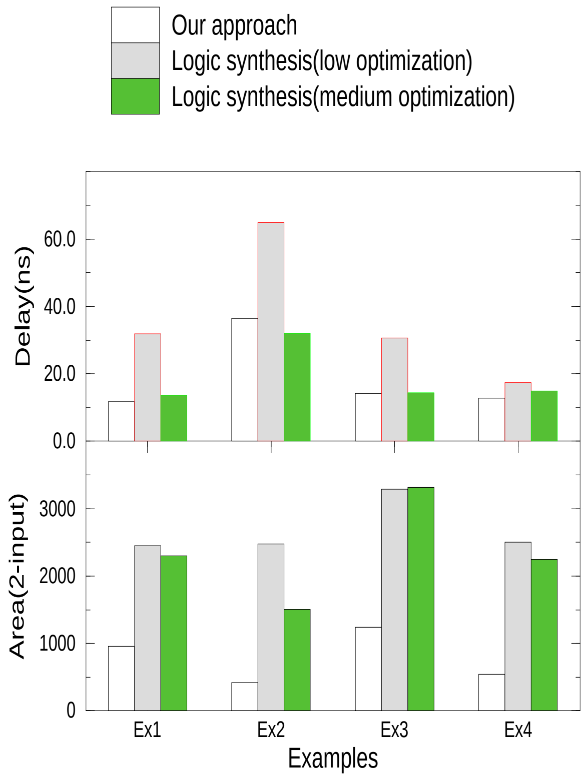


Figure 33: Pictorial representation of the comparative study: **performance metrics for delay-optimized designs**

in the bar-charts shown in Figures 31 to 34. For example, designs generated by logic synthesis tools are inferior in area by 139% to **487%**, in delay by -12% to **191%** and in runtime by 37% to **834%**.

Note that percentage difference reported for runtimes in Figures 29 and 30 considers the sum of the runtimes for mapping algorithm and glue logic optimization. If we consider the time for just mapping algorithm, we outperform logic synthesis by orders of magnitude. This is quite apparent by the bar-charts shown in Figures 32 and 34. Note that the bars for mapping algorithm in these figures are barely visible for the first three examples, indicating how relatively small they are.

We conclude this analysis with two comments. First, note that in this experiment, we have compared metrics from the netlist of generic gates. The effects of regularity are

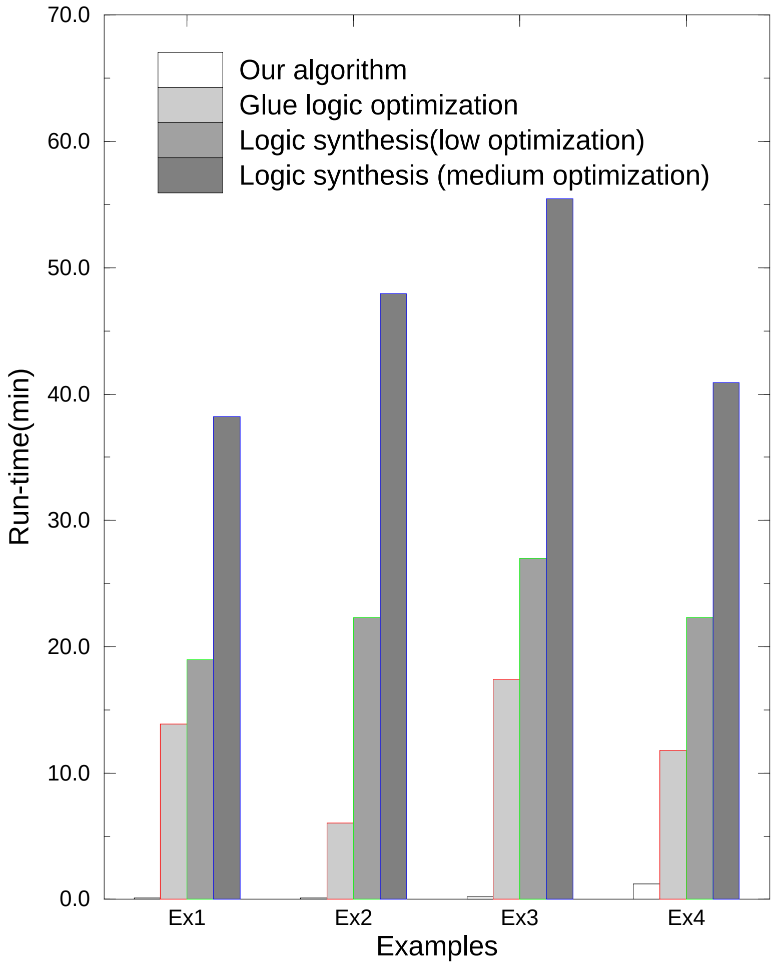


Figure 34: Pictorial representation of the comparative study: **run-time for delay-optimized designs**

more pronounced when we map these designs onto layout; designs from our approach would perform even better. Second, the reason we have been able to outperform logic synthesis tools is that these tools are designed for optimizing random or control logic. These tools cannot exploit the regular structures inherent in data-path components. The logic equations for a moderately size component (32-bit ALU in our example), are too big to be handled by these tools. Thus, a combination of the two approaches: logic synthesis for control logic and our approach for datapath components would be a good design strategy.

7 Summary and Future Work

We presented a novel library mapping approach at the RT-level based on functional specification of the source and target component. HLLM can reuse components from standard library, datapath generators or even handcrafted components. Specifically we formulated and solved the problem of ALU mapping. We presented a set of algorithms (greedy, dynamic programming and graph clustering) for HLLM. These algorithms could be used for generating either area-optimized or delay-optimized designs.

In our experiments, we demonstrated the versatility of our approach by applying HLLM on ALUs drawn from a wide variety of libraries. We also demonstrated the superiority of our approach over logic synthesis for complex ALU components in all the three metrics: area, delay and runtime. We believe that the HLLM approach needs to complement logic synthesis and traditional technology mapping techniques to bridge the output of architectural synthesis with RT-level databook libraries, module generators and handcrafted components.

Future work will involve applying HLLM for other regularly structured datapath components such as register-file, counters, etc. The final goal is to produce a mapping scheme for a complete netlist of RT level components.

8 Acknowledgements

This research was supported by SRC contract #93-DJ-146. We are grateful for their support.

References

[Am2901] "Am2901c: Four-bit Bipolar Microprocessor Slice," *Advanced Micro Devices, Sunnyvale, California,* 1993.

[AnDu94] R. Ang and N. Dutt, "An Algorithm for Allocation of Functional Units from Realistic RT Component Libraries," *The 7th International Symposium on High-Level Synthesis,* pp164-169, May 1994.

[BiGS89] W. P. Birmingham, A. P. Gupta and D. P. Siewiorek, "The MICON System for Computer Design," *The 26th ACM/IEEE Design Automation Conference,* pp135-139, 1989.

[BRSW87] R. K. Brayton, R. Rudell, A. Sangiovanni-Vincentelli and A. R. Wang, "MIS : A multiple-level logic optimization system," *IEEE Transaction on Computer-aided Design,* pp1062-1081, November 1987.

[Casc92] "Cascade Design Automation Databook," *Cascade Design Automation, Bellevue, WA,* 1992.

[CaTr89] R. Camposano and L. Trevillyan, "The Integration of Logic Synthesis and High-Level Synthesis," *International Symposium on Circuits and Systems (ISCAS-89),* 1989.

[CoLR90] T. H. Cormen, C. E. Leiserson and R. L. Rivest, "Introduction to Algorithms," *The MIT Press, Cambridge, Massachusetts* 1990.

[Degn92] "DesignWare Databook, Version 3.0," *Synopsys ® Inc., Mountain View,* December 1992.

[DJBT94] J. Darringer, W. Joyner, L. Berman and L. Trevillyan, "LSS: A System for Production Logic Synthesis," *IBM Journal of Research and Development,* vol. 28, No. 5, pp. 537-545, September 1984.

[DuKi91] N. D. Dutt and J. R. Kipps, "Bridging High-Level Synthesis to RTL Technology Libraries," *Proc. 28th Design Automation Conference,* June 1991.

[Dutt91] N. D. Dutt, "Generic Component Library Characterization for High-Level Synthesis," *Proc. VLSI Design 91,* January 1991.

[GCDM93] W. Guerts, F. Catthoor, and H. De Man, "Heuristic Techniques for the Synthesis of Complex Functional Units," *Proceedings of the European Conference on Design Automation (EDAC)*, pp. 552–556, 1993.

[Kahr86] M. Kahrs, "Matching a parts library in a silicon compiler," *Proceedings of ICCAD*, pp. 169–172, 1986.

[Keut87] K. Keutzer, "DAGON: Technology Binding and Local Optimization by DAG Matching" *The 24th ACM/IEEE Design Automation Conference,* 1987.

[LsiLogic] "Block Synthesis User's Guide," *LSI Logic Corporation, Milpitas*

[MaMi90] F. Mailhot and G. Micheli, "Technology Mapping Using Boolean Matching and Don't Care Sets," *European Design Automation Conference,* pp 212-216, 1990.

[Marw93] Peter Marwedel, "Tree-based Mapping of Algorithms to Predefined Structures," *Proc. of ICCAD,* pp. 586-593, 1993.

[RuGB93] Elke A. Rundensteiner, D. D. Gajski and L. Bic, "Component Synthesis From Functional Descriptions," *IEEE Transaction on Computer Aided Design,* pp1287-1299, September 1993.

[Syno92] "Design Analyzer Reference Manual, Version 3.0," *Synopsys* ® *Inc., Mountain View,* December 1992.

[Tosh90] "Toshiba ASIC Gate Array Library," *Toshiba Corporation, Tokyo, Japan,* 1990.

[VTI91] "VDP300 CMOS Datapath Library," *VLSI Technology, Inc., San Hose, California,* November 1991.

[VaGa88] N. Vander Zanden and D.D. Gajski, "MILO: A Microarchitecture and Logic Optimizer," *Proc. 25th Design Automation Conference,* June 1988.

[WPAV92] A. van der Werf, M. Peek, E. Aarts, J. Van Meerbergen, P. Lippens, W. Verhaegh, "Area Optimization of Multi-Functional Processing Units," *Proceedings of the International Conference on Computer-Aided Design*, pp. 292–299, 1992.

[XBLO92] S. H. Kelem and J. P. Seidel, "Shortening the Design Cycle for Programmable Logic Devices," *IEEE Design and Test of Computers,* pp40-50, 1992.

A Rule Database

This section lists mapping rules for ALU functions.

A.1 Arithmetic functions

A.1.1 ADD

```
NAME : AA1;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_ADD;
TARGET_FN : FM_ADD;

LEFT_IN : TI0 = SI0;
RIGHT_IN : TI1 = SI1;
CARRY_IN : TICIN[0:0] = SICIN[0:0];
MAIN_OUT : SO0 = TO0;
CARRY_OUT : SOCOUT[0:0] = TOCOUT[0:0];
OVF_OUT : SOVF[0:0] = TOVF[0:0];

NAME : AA2;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_ADD;
TARGET_FN : FM_ADD;

LEFT_IN : TI0 = SI1;
RIGHT_IN : TI1 = SI0;
CARRY_IN : TICIN[0:0] = SICIN[0:0];
MAIN_OUT : SO0 = TO0;
CARRY_OUT : SOCOUT[0:0] = TOCOUT[0:0];
OVF_OUT : SOVF[0:0] = TOVF[0:0];

NAME : AS1;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_ADD;
TARGET_FN : FM_SUB;

LEFT_IN : TI0 = SI0;
RIGHT_IN : TI1 = NOT SI1;
CARRY_IN : TICIN[0:0] = SICIN[0:0];
MAIN_OUT : SO0 = TO0;
CARRY_OUT : SOCOUT[0:0] = TOCOUT[0:0];
OVF_OUT : SOVF[0:0] = TOVF[0:0];

NAME : AS2;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_ADD;
```

```
TARGET_FN : FM_SUB;

LEFT_IN : TI0 = SI1;
RIGHT_IN : TI1 = NOT SI0;
CARRY_IN : TICIN[0:0] = SICIN[0:0];
MAIN_OUT : SO0 = TO0;
CARRY_OUT : SOCOUT[0:0] = TOCOUT[0:0];
OVF_OUT : SOVF[0:0] = TOVF[0:0];

NAME : AR1;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_ADD;
TARGET_FN : FM_RSUB;

LEFT_IN : TI0 = NOT SI0;
RIGHT_IN : TI1 = SI1;
CARRY_IN : TICIN[0:0] = SICIN[0:0];
MAIN_OUT : SO0 = TO0;
CARRY_OUT : SOCOUT[0:0] = TOCOUT[0:0];
OVF_OUT : SOVF[0:0] = TOVF[0:0];

NAME : AR2;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_ADD;
TARGET_FN : FM_RSUB;

LEFT_IN : TI0 = NOT SI1;
RIGHT_IN : TI1 = SI0;
CARRY_IN : TICIN[0:0] = SICIN[0:0];
MAIN_OUT : SO0 = TO0;
CARRY_OUT : SOCOUT[0:0] = TOCOUT[0:0];
OVF_OUT : SOVF[0:0] = TOVF[0:0];
```

A.1.2 SUB

```
NAME : SA1;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_SUB;
TARGET_FN : FM_ADD;

LEFT_IN : TI0 = SI0;
RIGHT_IN : TI1 = NOT SI1;
CARRY_IN : TICIN[0:0] = SICIN[0:0];
MAIN_OUT : SO0 = TO0;
CARRY_OUT : SOCOUT[0:0] = TOCOUT[0:0];
OVF_OUT : SOVF[0:0] = TOVF[0:0];

NAME : SA2;
GATE_COUNT : 8;
```

```
MAX_DELAY : 8;
SOURCE_FN : FM_SUB;
TARGET_FN : FM_ADD;

LEFT_IN : TI0 = NOT SI1;
RIGHT_IN : TI1 = SI0;
CARRY_IN : TICIN[0:0] = SICIN[0:0];
MAIN_OUT : SO0 = TO0;
CARRY_OUT : SOCOUT[0:0] = TOCOUT[0:0];
OVF_OUT : SOVF[0:0] = TOVF[0:0];

NAME : SS;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_SUB;
TARGET_FN : FM_SUB;

LEFT_IN : TI0 = SI0;
RIGHT_IN : TI1 = SI1;
CARRY_IN : TICIN[0:0] = SICIN[0:0];
MAIN_OUT : SO0 = TO0;
CARRY_OUT : SOCOUT[0:0] = TOCOUT[0:0];
OVF_OUT : SOVF[0:0] = TOVF[0:0];

NAME : SR;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_SUB;
TARGET_FN : FM_RSUB;

LEFT_IN : TI0 = SI1;
RIGHT_IN : TI1 = SI0;
CARRY_IN : TICIN[0:0] = SICIN[0:0];
MAIN_OUT : SO0 = TO0;
CARRY_OUT : SOCOUT[0:0] = TOCOUT[0:0];
OVF_OUT : SOVF[0:0] = TOVF[0:0];
```

A.1.3 RSUB

```
NAME : RA1;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_RSUB;
TARGET_FN : FM_ADD;

LEFT_IN : TI0 = NOT SI0;
RIGHT_IN : TI1 = SI1;
CARRY_IN : TICIN[0:0] = SICIN[0:0];
MAIN_OUT : SO0 = TO0;
CARRY_OUT : SOCOUT[0:0] = TOCOUT[0:0];
OVF_OUT : SOVF[0:0] = TOVF[0:0];
```

```
NAME : RA2;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_RSUB;
TARGET_FN : FM_ADD;

LEFT_IN : TI0 = SI1;
RIGHT_IN : TI1 = NOT SI0;
CARRY_IN : TICIN[0:0] = SICIN[0:0];
MAIN_OUT : SO0 = TO0;
CARRY_OUT : SOCOUT[0:0] = TOCOUT[0:0];
OVF_OUT : SOVF[0:0] = TOVF[0:0];

NAME : RS;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_RSUB;
TARGET_FN : FM_SUB;

LEFT_IN : TI0 = SI1;
RIGHT_IN : TI1 = SI0;
CARRY_IN : TICIN[0:0] = SICIN[0:0];
MAIN_OUT : SO0 = TO0;
CARRY_OUT : SOCOUT[0:0] = TOCOUT[0:0];
OVF_OUT : SOVF[0:0] = TOVF[0:0];

NAME : RR;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_RSUB;
TARGET_FN : FM_RSUB;

LEFT_IN : TI0 = SI0;
RIGHT_IN : TI1 = SI1;
CARRY_IN : TICIN[0:0] = SICIN[0:0];
MAIN_OUT : SO0 = TO0;
CARRY_OUT : SOCOUT[0:0] = TOCOUT[0:0];
OVF_OUT : SOVF[0:0] = TOVF[0:0];
```

A.1.4 INC

```
NAME : IA1;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_INC;
TARGET_FN : FM_ADD;

LEFT_IN : TI0 = SI0;
RIGHT_IN : TI1 = REPEAT (FM_WIDTH) "0";
CARRY_IN : TICIN[0:0] = "1";
MAIN_OUT : SO0 = TO0;
CARRY_OUT : SOCOUT[0:0] = TOCOUT[0:0];
```

```
OVF_OUT : SOVF[0:0] = TOVF[0:0];

NAME : IA2;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_INC;
TARGET_FN : FM_ADD;

LEFT_IN : TI0 =  REPEAT (FM_WIDTH) "0";
        RIGHT_IN : TI1 = SI0;
CARRY_IN : TICIN[0:0] = "1";
MAIN_OUT : SO0 = TO0;
CARRY_OUT : SOCOUT[0:0] = TOCOUT[0:0];
OVF_OUT : SOVF[0:0] = TOVF[0:0];

NAME : IA3;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_INC;
TARGET_FN : FM_ADD;

LEFT_IN : TI0 =  SI0;
        RIGHT_IN : TI1 = (REPEAT (FM_WIDTH-1) "0") CONCAT "1";
CARRY_IN : TICIN[0:0] = "0";
MAIN_OUT : SO0 = TO0;
CARRY_OUT : SOCOUT[0:0] = TOCOUT[0:0];
OVF_OUT : SOVF[0:0] = TOVF[0:0];

NAME : IA4;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_INC;
TARGET_FN : FM_ADD;

LEFT_IN : TI0 = (REPEAT (FM_WIDTH-1) "0") CONCAT "1";
        RIGHT_IN : TI1 = SI0;
CARRY_IN : TICIN[0:0] = "0";
MAIN_OUT : SO0 = TO0;
CARRY_OUT : SOCOUT[0:0] = TOCOUT[0:0];
OVF_OUT : SOVF[0:0] = TOVF[0:0];

NAME : IS1;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_INC;
TARGET_FN : FM_SUB;

LEFT_IN : TI0 = SI0;
RIGHT_IN : TI1 = REPEAT (FM_WIDTH) "1";
CARRY_IN : TICIN[0:0] = "1";
MAIN_OUT : SO0 = TO0;
CARRY_OUT : SOCOUT[0:0] = TOCOUT[0:0];
OVF_OUT : SOVF[0:0] = TOVF[0:0];
```

```
NAME : IS2;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_INC;
TARGET_FN : FM_SUB;

LEFT_IN : TI0 = SI0;
RIGHT_IN : TI1 = (REPEAT (FM_WIDTH-1) "1") CONCAT "0";
CARRY_IN : TICIN[0:0] = "0";
MAIN_OUT : SO0 = TO0;
CARRY_OUT : SOCOUT[0:0] = TOCOUT[0:0];
OVF_OUT : SOVF[0:0] = TOVF[0:0];

NAME : IR1;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_INC;
TARGET_FN : FM_RSUB;

LEFT_IN : TI0 = REPEAT (FM_WIDTH) "1";
RIGHT_IN : TI1 = SI0;
CARRY_IN : TICIN[0:0] = "1";
MAIN_OUT : SO0 = TO0;
CARRY_OUT : SOCOUT[0:0] = TOCOUT[0:0];
OVF_OUT : SOVF[0:0] = TOVF[0:0];

NAME : IR2;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_INC;
TARGET_FN : FM_RSUB;

LEFT_IN : TI0 = (REPEAT (FM_WIDTH-1) "1") CONCAT "0";
RIGHT_IN : TI1 = SI0;
CARRY_IN : TICIN[0:0] = "0";
MAIN_OUT : SO0 = TO0;
CARRY_OUT : SOCOUT[0:0] = TOCOUT[0:0];
OVF_OUT : SOVF[0:0] = TOVF[0:0];

NAME : II;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_INC;
TARGET_FN : FM_INC;

LEFT_IN : TI0 = SI0;
MAIN_OUT : SO0 = TO0;
```

A.1.5 DEC

```
NAME : DA1;
GATE_COUNT : 8;
```

```
MAX_DELAY : 8;
SOURCE_FN : FM_DEC;
TARGET_FN : FM_ADD;

LEFT_IN : TI0 = SI0;
RIGHT_IN : TI1 = REPEAT (FM_WIDTH) "1";
CARRY_IN : TICIN[0:0] = "0";
MAIN_OUT : SO0 = TO0;
CARRY_OUT : SOCOUT[0:0] = TOCOUT[0:0];
OVF_OUT : SOVF[0:0] = TOVF[0:0];

NAME : DA2;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_DEC;
TARGET_FN : FM_ADD;

LEFT_IN : TI0 =  REPEAT (FM_WIDTH) "1";
        RIGHT_IN : TI1 = SI0;
CARRY_IN : TICIN[0:0] = "0";
MAIN_OUT : SO0 = TO0;
CARRY_OUT : SOCOUT[0:0] = TOCOUT[0:0];
OVF_OUT : SOVF[0:0] = TOVF[0:0];

NAME : DA3;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_DEC;
TARGET_FN : FM_ADD;

LEFT_IN : TI0 =  SI0;
        RIGHT_IN : TI1 = (REPEAT (FM_WIDTH-1) "1") CONCAT "0";
CARRY_IN : TICIN[0:0] = "1";
MAIN_OUT : SO0 = TO0;
CARRY_OUT : SOCOUT[0:0] = TOCOUT[0:0];
OVF_OUT : SOVF[0:0] = TOVF[0:0];

NAME : DA4;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_DEC;
TARGET_FN : FM_ADD;

LEFT_IN : TI0 = (REPEAT (FM_WIDTH-1) "1") CONCAT "0";
        RIGHT_IN : TI1 = SI0;
CARRY_IN : TICIN[0:0] = "1";
MAIN_OUT : SO0 = TO0;
CARRY_OUT : SOCOUT[0:0] = TOCOUT[0:0];
OVF_OUT : SOVF[0:0] = TOVF[0:0];

NAME : DS1;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_DEC;
```

```
TARGET_FN : FM_SUB;

LEFT_IN : TI0 = SI0;
RIGHT_IN : TI1 = REPEAT (FM_WIDTH) "0";
CARRY_IN : TICIN[0:0] = "0";
MAIN_OUT : SO0 = TO0;
CARRY_OUT : SOCOUT[0:0] = TOCOUT[0:0];
OVF_OUT : SOVF[0:0] = TOVF[0:0];

NAME : DS2;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_DEC;
TARGET_FN : FM_SUB;

LEFT_IN : TI0 = SI0;
RIGHT_IN : TI1 = (REPEAT (FM_WIDTH-1) "0") CONCAT "1";
CARRY_IN : TICIN[0:0] = "1";
MAIN_OUT : SO0 = TO0;
CARRY_OUT : SOCOUT[0:0] = TOCOUT[0:0];
OVF_OUT : SOVF[0:0] = TOVF[0:0];

NAME : DR1;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_DEC;
TARGET_FN : FM_RSUB;

LEFT_IN : TI0 = REPEAT (FM_WIDTH) "0";
RIGHT_IN : TI1 = SI0;
CARRY_IN : TICIN[0:0] = "0";
MAIN_OUT : SO0 = TO0;
CARRY_OUT : SOCOUT[0:0] = TOCOUT[0:0];
OVF_OUT : SOVF[0:0] = TOVF[0:0];

NAME : DR2;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_DEC;
TARGET_FN : FM_RSUB;

LEFT_IN : TI0 = (REPEAT (FM_WIDTH-1) "0") CONCAT "1";
RIGHT_IN : TI1 = SI0;
CARRY_IN : TICIN[0:0] = "1";
MAIN_OUT : SO0 = TO0;
CARRY_OUT : SOCOUT[0:0] = TOCOUT[0:0];
OVF_OUT : SOVF[0:0] = TOVF[0:0];

NAME : DD;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_DEC;
TARGET_FN : FM_DEC;
```

```
LEFT_IN : TI0 = SI0;
MAIN_OUT : SO0 = TO0;
```

A.2 Logic functions

A.2.1 ZERO

```
NAME : ZOZO;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_ZERO;
TARGET_FN : FM_ZERO;

MAIN_OUT : SO0 = TO0;

NAME : ZOOE;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_ZERO;
TARGET_FN : FM_ONE;

MAIN_OUT : SO0 = NOT TO0;

NAME : ZOAD1;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_ZERO;
TARGET_FN : FM_AND;

LEFT_IN : TI0 = REPEAT (FM_WIDTH) "0";
MAIN_OUT : SO0 = TO0;

NAME : ZOAD2;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_ZERO;
TARGET_FN : FM_AND;

RIGHT_IN : TI1 = REPEAT (FM_WIDTH) "0";
MAIN_OUT : SO0 = TO0;

NAME : ZOOR1;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_ZERO;
TARGET_FN : FM_OR;

LEFT_IN : TI0 = REPEAT (FM_WIDTH) "1";
MAIN_OUT : SO0 = NOT TO0;

NAME : ZOOR2;
GATE_COUNT : 8;
```

```
MAX_DELAY : 8;
SOURCE_FN : FM_ZERO;
TARGET_FN : FM_OR;

RIGHT_IN : TI1 = REPEAT (FM_WIDTH) "1";
MAIN_OUT : SO0 = NOT TO0;
```

A.2.2 ONE

```
NAME : OEOE;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_ONE;
TARGET_FN : FM_ONE;

MAIN_OUT : SO0 = TO0;

NAME : OEZO;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_ONE;
TARGET_FN : FM_ZERO;

MAIN_OUT : SO0 = NOT TO0;

NAME : OEAD1;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_ONE;
TARGET_FN : FM_AND;

LEFT_IN : TI0 = REPEAT (FM_WIDTH) "0";
MAIN_OUT : SO0 = NOT TO0;

NAME : OEAD2;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_ONE;
TARGET_FN : FM_AND;

RIGHT_IN : TI1 = REPEAT (FM_WIDTH) "0";
MAIN_OUT : SO0 = NOT TO0;

NAME : OEOR1;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_ONE;
TARGET_FN : FM_OR;

LEFT_IN : TI0 = REPEAT (FM_WIDTH) "1";
MAIN_OUT : SO0 = TO0;
```

```
NAME : OEOR2;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_ONE;
TARGET_FN : FM_OR;

RIGHT_IN : TI1 = REPEAT (FM_WIDTH) "1";
MAIN_OUT : SO0 = TO0;
```

A.2.3 AND

```
NAME : ADAD1;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_AND;
TARGET_FN : FM_AND;

LEFT_IN : TI0 = SI0;
RIGHT_IN : TI1 = SI1;
MAIN_OUT : SO0 = TO0;

NAME : ADAD2;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_AND;
TARGET_FN : FM_AND;

LEFT_IN : TI0 = SI1;
RIGHT_IN : TI1 = SI0;
MAIN_OUT : SO0 = TO0;

NAME : ADNE;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_AND;
TARGET_FN : FM_NONE;

LOGIC_LEFT_IN : TLI0 = SI0;
LOGIC_RIGHT_IN : TLI1 = SI1;
MAIN_OUT : SO0 = LO0;
LOGIC_OUT : LO = 0,0,0,1;

NAME : ADND1;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_AND;
TARGET_FN : FM_NAND;

LEFT_IN : TI0 = SI0;
RIGHT_IN : TI1 = SI1;
MAIN_OUT : SO0 = NOT TO0;
```

```
NAME : ADND2;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_AND;
TARGET_FN : FM_NAND;

LEFT_IN : TI0 = SI1;
RIGHT_IN : TI1 = SI0;
MAIN_OUT : SO0 = NOT TO0;
```

A.2.4 NAND

```
NAME : NDND1;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_NAND;
TARGET_FN : FM_NAND;

LEFT_IN : TI0 = SI0;
RIGHT_IN : TI1 = SI1;
MAIN_OUT : SO0 = TO0;

NAME : NDND2;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_NAND;
TARGET_FN : FM_NAND;

LEFT_IN : TI0 = SI1;
RIGHT_IN : TI1 = SI0;
MAIN_OUT : SO0 = TO0;

NAME : NDNE;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_NAND;
TARGET_FN : FM_NONE;

LOGIC_LEFT_IN : TLI0 = SI0;
LOGIC_RIGHT_IN : TLI1 = SI1;
MAIN_OUT : SO0 = LO0;
LOGIC_OUT : LO = 1,1,1,0;

NAME : NDAD1;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_NAND;
TARGET_FN : FM_AND;

LEFT_IN : TI0 = SI0;
RIGHT_IN : TI1 = SI1;
MAIN_OUT : SO0 = NOT TO0;
```

```
NAME : NDAD2;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_NAND;
TARGET_FN : FM_AND;

LEFT_IN : TI0 = SI1;
RIGHT_IN : TI1 = SI0;
MAIN_OUT : SO0 = NOT TO0;
```

A.2.5 OR

```
NAME : OROR1;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_OR;
TARGET_FN : FM_OR;

LEFT_IN : TI0 = SI0;
RIGHT_IN : TI1 = SI1;
MAIN_OUT : SO0 = TO0;

NAME : OROR2;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_OR;
TARGET_FN : FM_OR;

LEFT_IN : TI0 = SI1;
RIGHT_IN : TI1 = SI0;
MAIN_OUT : SO0 = TO0;

NAME : NRNE;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_OR;
TARGET_FN : FM_NONE;

LOGIC_LEFT_IN : TLI0 = SI0;
LOGIC_RIGHT_IN : TLI1 = SI1;
MAIN_OUT : SO0 = LO0;
LOGIC_OUT : LO = 0,1,1,1;

NAME : ORNR1;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_OR;
TARGET_FN : FM_NOR;

LEFT_IN : TI0 = SI0;
RIGHT_IN : TI1 = SI1;
```

```
MAIN_OUT : SO0 = NOT TO0;

NAME : ORNR2;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_OR;
TARGET_FN : FM_NOR;

LEFT_IN : TI0 = SI1;
RIGHT_IN : TI1 = SI0;
MAIN_OUT : SO0 = NOT TO0;
```

A.2.6 NOR

```
NAME : NRNR1;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_NOR;
TARGET_FN : FM_NOR;

LEFT_IN : TI0 = SI0;
RIGHT_IN : TI1 = SI1;
MAIN_OUT : SO0 = TO0;

NAME : NRNR2;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_NOR;
TARGET_FN : FM_NOR;

LEFT_IN : TI0 = SI1;
RIGHT_IN : TI1 = SI0;
MAIN_OUT : SO0 = TO0;

NAME : NRNE;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_NOR;
TARGET_FN : FM_NONE;

LOGIC_LEFT_IN : TLI0 = SI0;
LOGIC_RIGHT_IN : TLI1 = SI1;
MAIN_OUT : SO0 = LO0;
LOGIC_OUT : LO = 1,0,0,0;

NAME : NROR1;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_NOR;
TARGET_FN : FM_OR;

LEFT_IN : TI0 = SI0;
```

```
RIGHT_IN : TI1 = SI1;
MAIN_OUT : SO0 = NOT TO0;

NAME : NROR2;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_NOR;
TARGET_FN : FM_OR;

LEFT_IN : TI0 = SI1;
RIGHT_IN : TI1 = SI0;
MAIN_OUT : SO0 = NOT TO0;
```

A.2.7 XOR

```
NAME : XRXR1;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_XOR;
TARGET_FN : FM_XOR;

LEFT_IN : TI0 = SI0;
RIGHT_IN : TI1 = SI1;
MAIN_OUT : SO0 = TO0;

NAME : XRXR2;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_XOR;
TARGET_FN : FM_XOR;

LEFT_IN : TI0 = SI1;
RIGHT_IN : TI1 = SI0;
MAIN_OUT : SO0 = TO0;

NAME : XRNE;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_XOR;
TARGET_FN : FM_NONE;

LOGIC_LEFT_IN : TLI0 = SI0;
LOGIC_RIGHT_IN : TLI1 = SI1;
MAIN_OUT : SO0 = LO0;
LOGIC_OUT : LO = 0,1,1,0;

NAME : XRXN1;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_XOR;
TARGET_FN : FM_XNOR;
```

```
LEFT_IN : TI0 = SI0;
RIGHT_IN : TI1 = SI1;
MAIN_OUT : SO0 = NOT TO0;

NAME : XRXN2;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_XOR;
TARGET_FN : FM_XNOR;

LEFT_IN : TI0 = SI1;
RIGHT_IN : TI1 = SI0;
MAIN_OUT : SO0 = NOT TO0;
```

A.2.8 XNOR

```
NAME : XNXN1;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_XNOR;
TARGET_FN : FM_XNOR;

LEFT_IN : TI0 = SI0;
RIGHT_IN : TI1 = SI1;
MAIN_OUT : SO0 = TO0;

NAME : XNXN2;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_XNOR;
TARGET_FN : FM_XNOR;

LEFT_IN : TI0 = SI1;
RIGHT_IN : TI1 = SI0;
MAIN_OUT : SO0 = TO0;

NAME : XNNE;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_XNOR;
TARGET_FN : FM_NONE;

LOGIC_LEFT_IN : TLI0 = SI0;
LOGIC_RIGHT_IN : TLI1 = SI1;
MAIN_OUT : SO0 = LO0;
LOGIC_OUT : LO = 1,0,0,1;

NAME : XNXR1;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_XNOR;
TARGET_FN : FM_XOR;
```

```
LEFT_IN : TI0 = SI0;
RIGHT_IN : TI1 = SI1;
MAIN_OUT : SO0 = NOT TO0;

NAME : XNXR2;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_XNOR;
TARGET_FN : FM_XOR;

LEFT_IN : TI0 = SI1;
RIGHT_IN : TI1 = SI0;
MAIN_OUT : SO0 = NOT TO0;
```

A.2.9 LID

```
NAME : LDLD;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_LID;
TARGET_FN : FM_LID;

LEFT_IN : TI0 = SI0;
MAIN_OUT : SO0 = TO0;

NAME : LDLN;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_LID;
TARGET_FN : FM_LNOT;

LEFT_IN : TI0 = NOT SI0;
MAIN_OUT : SO0 = TO0;

NAME : LDRN;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_LID;
TARGET_FN : FM_RNOT;

RIGHT_IN : TI1 = NOT SI0;
MAIN_OUT : SO0 = TO0;

NAME : LDAD1;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_LID;
TARGET_FN : FM_AND;

LEFT_IN : TI0 = SI0;
RIGHT_IN : TI1 = REPEAT (FM_WIDTH) "1";
```

```
MAIN_OUT : SO0 = TO0;

NAME : LDAD2;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_LID;
TARGET_FN : FM_AND;

LEFT_IN : TI0 = REPEAT (FM_WIDTH) "1";
RIGHT_IN : TI1 = SI0;
MAIN_OUT : SO0 = TO0;

NAME : LDOR1;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_LID;
TARGET_FN : FM_OR;

LEFT_IN : TI0 = SI0;
RIGHT_IN : TI1 = REPEAT (FM_WIDTH) "0";
MAIN_OUT : SO0 = TO0;

NAME : LDOR2;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_LID;
TARGET_FN : FM_OR;

LEFT_IN : TI0 = REPEAT (FM_WIDTH) "0";
RIGHT_IN : TI1 = SI0;
MAIN_OUT : SO0 = TO0;


NAME : LDNE;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_LID;
TARGET_FN : FM_NONE;

LOGIC_LEFT_IN : TLI0 = SI0;
LOGIC_RIGHT_IN : TLI1 = SI1;
MAIN_OUT : SO0 = LO0;
LOGIC_OUT : LO = 0,0,1,1;

NAME : LDRD;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_LID;
TARGET_FN : FM_RID;

RIGHT_IN : TI1 = SI1;
MAIN_OUT : SO0 = TO0;
```

A.2.10 RID

```
NAME : RDRD;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_RID;
TARGET_FN : FM_RID;

RIGHT_IN : TI1 = SI1;
MAIN_OUT : SO0 = TO0;

NAME : RDLN;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_RID;
TARGET_FN : FM_LNOT;

LEFT_IN : TI0 = NOT SI1;
MAIN_OUT : SO0 = TO0;

NAME : RDRN;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_RID;
TARGET_FN : FM_RNOT;

RIGHT_IN : TI1 = NOT SI1;
MAIN_OUT : SO0 = TO0;

NAME : RDAD1;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_RID;
TARGET_FN : FM_AND;

LEFT_IN : TI0 = SI1;
RIGHT_IN : TI1 = REPEAT (FM_WIDTH) "1";
MAIN_OUT : SO0 = TO0;

NAME : RDAD2;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_RID;
TARGET_FN : FM_AND;

LEFT_IN : TI0 = REPEAT (FM_WIDTH) "1";
RIGHT_IN : TI1 = SI1;
MAIN_OUT : SO0 = TO0;

NAME : RDOR1;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_RID;
TARGET_FN : FM_OR;
```

```
LEFT_IN : TI0 = SI1;
RIGHT_IN : TI1 = REPEAT (FM_WIDTH) "0";
MAIN_OUT : SO0 = TO0;

NAME : RDOR2;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_RID;
TARGET_FN : FM_OR;

LEFT_IN : TI0 = REPEAT (FM_WIDTH) "0";
RIGHT_IN : TI1 = SI1;
MAIN_OUT : SO0 = TO0;

NAME : RDNE;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_RID;
TARGET_FN : FM_NONE;

LOGIC_LEFT_IN : TLI0 = SI0;
LOGIC_RIGHT_IN : TLI1 = SI1;
MAIN_OUT : SO0 = LO0;
LOGIC_OUT : LO = 0,1,0,1;

NAME : RDLD;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_RID;
TARGET_FN : FM_LID;

LEFT_IN : TI0 = SI1;
MAIN_OUT : SO0 = TO0;
```

A.2.11 LNOT

```
NAME : LNLN;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_LNOT;
TARGET_FN : FM_LNOT;

LEFT_IN : TI0 = SI0;
MAIN_OUT : SO0 = TO0;

NAME : LNLD;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_LNOT;
TARGET_FN : FM_LID;
```

```
LEFT_IN : TI0 = NOT SI0;
MAIN_OUT : SO0 = TO0;

NAME : LNRD;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_LNOT;
TARGET_FN : FM_RID;

RIGHT_IN : TI1 = NOT SI0;
MAIN_OUT : SO0 = TO0;

NAME : LNAD1;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_LNOT;
TARGET_FN : FM_AND;

LEFT_IN : TI0 = NOT SI0;
RIGHT_IN : TI1 = REPEAT (FM_WIDTH) "1";
MAIN_OUT : SO0 = TO0;

NAME : LNAD2;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_LNOT;
TARGET_FN : FM_AND;

LEFT_IN : TI0 = REPEAT (FM_WIDTH) "1";
RIGHT_IN : TI1 = NOT SI0;
MAIN_OUT : SO0 = TO0;

NAME : LNOR1;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_LNOT;
TARGET_FN : FM_OR;

LEFT_IN : TI0 = NOT SI0;
RIGHT_IN : TI1 = REPEAT (FM_WIDTH) "0";
MAIN_OUT : SO0 = TO0;

NAME : LNOR2;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_LNOT;
TARGET_FN : FM_OR;

LEFT_IN : TI0 = REPEAT (FM_WIDTH) "0";
RIGHT_IN : TI1 = NOT SI0;
MAIN_OUT : SO0 = TO0;

NAME : LNNE;
GATE_COUNT : 8;
```

```
MAX_DELAY : 8;
SOURCE_FN : FM_LNOT;
TARGET_FN : FM_NONE;

LOGIC_LEFT_IN : TLI0 = SI0;
LOGIC_RIGHT_IN : TLI1 = SI1;
MAIN_OUT : SO0 = LO0;
LOGIC_OUT : LO = 1,1,0,0;

NAME : LNRN;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_LNOT;
TARGET_FN : FM_RNOT;

RIGHT_IN : TI1 = SI1;
MAIN_OUT : SO0 = TO0;
```

A.2.12 RNOT

```
NAME : RNRN;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_RNOT;
TARGET_FN : FM_RNOT;

RIGHT_IN : TI1 = SI1;
MAIN_OUT : SO0 = TO0;

NAME : RNLD;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_RNOT;
TARGET_FN : FM_LID;

LEFT_IN : TI0 = NOT SI1;
MAIN_OUT : SO0 = TO0;

NAME : RNRD;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_RNOT;
TARGET_FN : FM_RID;

RIGHT_IN : TI1 = NOT SI1;
MAIN_OUT : SO0 = TO0;

NAME : RNAD1;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_RNOT;
TARGET_FN : FM_AND;
```

```
LEFT_IN : TI0 = NOT SI1;
RIGHT_IN : TI1 = REPEAT (FM_WIDTH) "1";
MAIN_OUT : SO0 = TO0;

NAME : RNAD2;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_RNOT;
TARGET_FN : FM_AND;

LEFT_IN : TI0 = REPEAT (FM_WIDTH) "1";
RIGHT_IN : TI1 = NOT SI1;
MAIN_OUT : SO0 = TO0;

NAME : RNOR1;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_RNOT;
TARGET_FN : FM_OR;

LEFT_IN : TI0 = NOT SI1;
RIGHT_IN : TI1 = REPEAT (FM_WIDTH) "0";
MAIN_OUT : SO0 = TO0;

NAME : RDOR2;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_RNOT;
TARGET_FN : FM_OR;

LEFT_IN : TI0 = REPEAT (FM_WIDTH) "0";
RIGHT_IN : TI1 = NOT SI1;
MAIN_OUT : SO0 = TO0;

NAME : RNNE;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_RNOT;
TARGET_FN : FM_NONE;

LOGIC_LEFT_IN : TLI0 = SI0;
LOGIC_RIGHT_IN : TLI1 = SI1;
MAIN_OUT : SO0 = LO0;
LOGIC_OUT : LO = 1,0,1,0;

NAME : RNLN;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_RNOT;
TARGET_FN : FM_LNOT;

LEFT_IN : TI0 = SI1;
MAIN_OUT : SO0 = TO0;
```

A.2.13 LINHI

```
NAME : LILI;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_LINHI;
TARGET_FN : FM_LINHI;

LEFT_IN : TI0 = SI0;
RIGHT_IN : TI1 = SI1;
MAIN_OUT : SO0 = TO0;

NAME : LIRI;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_LINHI;
TARGET_FN : FM_RINHI;

LEFT_IN : TI0 = SI1;
RIGHT_IN : TI1 = SI0;
MAIN_OUT : SO0 = TO0;

NAME : LINE;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_LINHI;
TARGET_FN : FM_NONE;

LOGIC_LEFT_IN : TLI0 = SI0;
LOGIC_RIGHT_IN : TLI1 = SI1;
MAIN_OUT : SO0 = LO0;
LOGIC_OUT : LO = 0,1,0,0;

NAME : LIRL;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_LINHI;
TARGET_FN : FM_RIMPL;

LEFT_IN : TI0 = SI0;
RIGHT_IN : TI1 = SI1;
MAIN_OUT : SO0 = NOT TO0;
```

A.2.14 RINHI

```
NAME : RIRI;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_RINHI;
TARGET_FN : FM_RINHI;

LEFT_IN : TI0 = SI0;
```

```
RIGHT_IN : TI1 = SI1;
MAIN_OUT : SO0 = TO0;

NAME : RILI;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_RINHI;
TARGET_FN : FM_LINHI;

LEFT_IN : TI0 = SI1;
RIGHT_IN : TI1 = SI0;
MAIN_OUT : SO0 = TO0;

NAME : RINE;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_RINHI;
TARGET_FN : FM_NONE;

LOGIC_LEFT_IN : TLI0 = SI0;
LOGIC_RIGHT_IN : TLI1 = SI1;
MAIN_OUT : SO0 = LO0;
LOGIC_OUT : LO = 0,0,1,0;

NAME : RILL;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_RINHI;
TARGET_FN : FM_LIMPL;

LEFT_IN : TI0 = SI0;
RIGHT_IN : TI1 = SI1;
MAIN_OUT : SO0 = NOT TO0;
```

A.2.15 LIMPL

```
NAME : LLLL;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_LIMPL;
TARGET_FN : FM_LIMPL;

LEFT_IN : TI0 = SI0;
RIGHT_IN : TI1 = SI1;
MAIN_OUT : SO0 = TO0;

NAME : LLRL;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_LIMPL;
TARGET_FN : FM_RIMPL;
```

```
LEFT_IN : TI0 = SI1;
RIGHT_IN : TI1 = SI0;
MAIN_OUT : SO0 = TO0;

NAME : LLNE;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_LIMPL;
TARGET_FN : FM_NONE;

LOGIC_LEFT_IN : TLI0 = SI0;
LOGIC_RIGHT_IN : TLI1 = SI1;
MAIN_OUT : SO0 = LO0;
LOGIC_OUT : LO = 1,1,0,1;

NAME : LLRI;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_LIMPL;
TARGET_FN : FM_RINHI;

LEFT_IN : TI0 = SI0;
RIGHT_IN : TI1 = SI1;
MAIN_OUT : SO0 = NOT TO0;
```

A.2.16 RIMPL

```
NAME : RLRL;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_RIMPL;
TARGET_FN : FM_RIMPL;

LEFT_IN : TI0 = SI0;
RIGHT_IN : TI1 = SI1;
MAIN_OUT : SO0 = TO0;

NAME : RLLL;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_RIMPL;
TARGET_FN : FM_LIMPL;

LEFT_IN : TI0 = SI1;
RIGHT_IN : TI1 = SI0;
MAIN_OUT : SO0 = TO0;

NAME : RLNE;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_RIMPL;
TARGET_FN : FM_NONE;
```

```
LOGIC_LEFT_IN : TLI0 = SI0;
LOGIC_RIGHT_IN : TLI1 = SI1;
MAIN_OUT : SO0 = LO0;
LOGIC_OUT : LO = 1,0,1,1;

NAME : RLLI;
GATE_COUNT : 8;
MAX_DELAY : 8;
SOURCE_FN : FM_RIMPL;
TARGET_FN : FM_LINHI;

LEFT_IN : TI0 = SI0;
RIGHT_IN : TI1 = SI1;
MAIN_OUT : SO0 = NOT TO0;
```

A.3 Comparison functions

A.3.1 EQ

```
NAME : EQEQ1;
GATE_COUNT : 0;
MAX_DELAY : 0;
SOURCE_FN : FM_EQ;
TARGET_FN : FM_EQ;

LEFT_IN : TI0 = SI0;
RIGHT_IN : TI1 = SI1;
COMP_OUT : SO1[0:0] = TO1[0:0];

NAME : EQEQ2;
GATE_COUNT : 0;
MAX_DELAY : 0;
SOURCE_FN : FM_EQ;
TARGET_FN : FM_EQ;

LEFT_IN : TI0 = SI1;
RIGHT_IN : TI1 = SI0;
COMP_OUT : SO1[0:0] = TO1[0:0];

NAME : EQNEQ1;
GATE_COUNT : 1;
MAX_DELAY : 1;
SOURCE_FN : FM_EQ;
TARGET_FN : FM_NEQ;

LEFT_IN : TI0 = SI0;
RIGHT_IN : TI1 = SI1;
COMP_OUT : SO1[0:0] = NOT TO1[0:0];

NAME : EQNEQ2;
GATE_COUNT : 1;
```

```
MAX_DELAY : 1;
SOURCE_FN : FM_EQ;
TARGET_FN : FM_NEQ;

LEFT_IN : TI0 = SI1;
RIGHT_IN : TI1 = SI0;
COMP_OUT : SO1[0:0] = TO1[0:0];

NAME : EQAD1;
GATE_COUNT : 1;
MAX_DELAY : 1;
SOURCE_FN : FM_EQ;
TARGET_FN : FM_ADD;

LEFT_IN : TI0 = SI0;
RIGHT_IN : TI1 = NOT SI1;
CARRY_IN : TICIN[0:0] = "1";
COMP_OUT : SO1[0:0] = GNOR TO0;


NAME : EQAD2;
GATE_COUNT : 1;
MAX_DELAY : 1;
SOURCE_FN : FM_EQ;
TARGET_FN : FM_ADD;

LEFT_IN : TI0 = NOT SI0;
RIGHT_IN : TI1 = SI1;
CARRY_IN : TICIN[0:0] = "1";
COMP_OUT : SO1[0:0] = GNOR TO0;

NAME : EQAD3;
GATE_COUNT : 1;
MAX_DELAY : 1;
SOURCE_FN : FM_EQ;
TARGET_FN : FM_ADD;

LEFT_IN : TI0 = SI1;
RIGHT_IN : TI1 = NOT SI0;
CARRY_IN : TICIN[0:0] = "1";
COMP_OUT : SO1[0:0] = GNOR TO0;


NAME : EQAD4;
GATE_COUNT : 1;
MAX_DELAY : 1;
SOURCE_FN : FM_EQ;
TARGET_FN : FM_ADD;

LEFT_IN : TI0 = NOT SI1;
RIGHT_IN : TI1 = SI0;
CARRY_IN : TICIN[0:0] = "1";
COMP_OUT : SO1[0:0] = GNOR TO0;
```

```
NAME : EQSB1;
GATE_COUNT : 1;
MAX_DELAY : 1;
SOURCE_FN : FM_EQ;
TARGET_FN : FM_SUB;

LEFT_IN : TI0 = SI0;
RIGHT_IN : TI1 = SI1;
CARRY_IN : TICIN[0:0] = "1";
COMP_OUT : SO1[0:0] = GNOR TO0;

NAME : EQSB2;
GATE_COUNT : 1;
MAX_DELAY : 1;
SOURCE_FN : FM_EQ;
TARGET_FN : FM_SUB;

LEFT_IN : TI0 = SI1;
RIGHT_IN : TI1 = SI0;
CARRY_IN : TICIN[0:0] = "1";
COMP_OUT : SO1[0:0] = GNOR TO0;

NAME : EQSB3;
GATE_COUNT : 1;
MAX_DELAY : 1;
SOURCE_FN : FM_EQ;
TARGET_FN : FM_SUB;

LEFT_IN : TI0 = NOT SI0;
RIGHT_IN : TI1 = NOT SI1;
CARRY_IN : TICIN[0:0] = "1";
COMP_OUT : SO1[0:0] = GNOR TO0;

NAME : EQSB4;
GATE_COUNT : 1;
MAX_DELAY : 1;
SOURCE_FN : FM_EQ;
TARGET_FN : FM_SUB;

LEFT_IN : TI0 = NOT SI1;
RIGHT_IN : TI1 = NOT SI0;
CARRY_IN : TICIN[0:0] = "1";
COMP_OUT : SO1[0:0] = GNOR TO0;

NAME : EQRB1;
GATE_COUNT : 1;
MAX_DELAY : 1;
SOURCE_FN : FM_EQ;
TARGET_FN : FM_RSUB;

LEFT_IN : TI0 = SI0;
RIGHT_IN : TI1 = SI1;
CARRY_IN : TICIN[0:0] = "1";
COMP_OUT : SO1[0:0] = GNOR TO0;
```

```
NAME : EQRB2;
GATE_COUNT : 1;
MAX_DELAY : 1;
SOURCE_FN : FM_EQ;
TARGET_FN : FM_RSUB;

LEFT_IN : TI0 = SI1;
RIGHT_IN : TI1 = SI0;
CARRY_IN : TICIN[0:0] = "1";
COMP_OUT : SO1[0:0] = GNOR TO0;

NAME : EQRB3;
GATE_COUNT : 1;
MAX_DELAY : 1;
SOURCE_FN : FM_EQ;
TARGET_FN : FM_RSUB;

LEFT_IN : TI0 = NOT SI0;
RIGHT_IN : TI1 = NOT SI1;
CARRY_IN : TICIN[0:0] = "1";
COMP_OUT : SO1[0:0] = GNOR TO0;

NAME : EQRB4;
GATE_COUNT : 1;
MAX_DELAY : 1;
SOURCE_FN : FM_EQ;
TARGET_FN : FM_RSUB;

LEFT_IN : TI0 = NOT SI1;
RIGHT_IN : TI1 = NOT SI0;
CARRY_IN : TICIN[0:0] = "1";
COMP_OUT : SO1[0:0] = GNOR TO0;

NAME : EQXR1;
GATE_COUNT : 1;
MAX_DELAY : 1;
SOURCE_FN : FM_EQ;
TARGET_FN : FM_XOR;

LEFT_IN : TI0 = SI0;
RIGHT_IN : TI1 = SI1;
COMP_OUT : SO1[0:0] = GNOR TO0;

NAME : EQXR2;
GATE_COUNT : 1;
MAX_DELAY : 1;
SOURCE_FN : FM_EQ;
TARGET_FN : FM_XOR;

LEFT_IN : TI0 = SI1;
RIGHT_IN : TI1 = SI0;
COMP_OUT : SO1[0:0] = GNOR TO0;
```

```
NAME : EQXR3;
GATE_COUNT : 1;
MAX_DELAY : 1;
SOURCE_FN : FM_EQ;
TARGET_FN : FM_XOR;

LEFT_IN : TI0 = NOT SI0;
RIGHT_IN : TI1 = NOT SI1;
COMP_OUT : SO1[0:0] = GNOR TO0;

NAME : EQXR4;
GATE_COUNT : 1;
MAX_DELAY : 1;
SOURCE_FN : FM_EQ;
TARGET_FN : FM_XOR;

LEFT_IN : TI0 = NOT SI1;
RIGHT_IN : TI1 = NOT SI0;
COMP_OUT : SO1[0:0] = GNOR TO0;

NAME : EQXN1;
GATE_COUNT : 1;
MAX_DELAY : 1;
SOURCE_FN : FM_EQ;
TARGET_FN : FM_XNOR;

LEFT_IN : TI0 = SI0;
RIGHT_IN : TI1 = SI1;
COMP_OUT : SO1[0:0] = GAND TO0;

NAME : EQXN2;
GATE_COUNT : 1;
MAX_DELAY : 1;
SOURCE_FN : FM_EQ;
TARGET_FN : FM_XNOR;

LEFT_IN : TI0 = SI1;
RIGHT_IN : TI1 = SI0;
COMP_OUT : SO1[0:0] = GAND TO0;

NAME : EQXN3;
GATE_COUNT : 1;
MAX_DELAY : 1;
SOURCE_FN : FM_EQ;
TARGET_FN : FM_XNOR;

LEFT_IN : TI0 = NOT SI0;
RIGHT_IN : TI1 = NOT SI1;
COMP_OUT : SO1[0:0] = GAND TO0;

NAME : EQXN4;
GATE_COUNT : 1;
MAX_DELAY : 1;
SOURCE_FN : FM_EQ;
```

```
TARGET_FN : FM_XNOR;

LEFT_IN : TI0 = NOT SI1;
RIGHT_IN : TI1 = NOT SI0;
COMP_OUT : SO1[0:0] = GAND TO0;
```

A.3.2 NEQ

```
NAME : NQNQ1;
GATE_COUNT : 0;
MAX_DELAY : 0;
SOURCE_FN : FM_NEQ;
TARGET_FN : FM_NEQ;

LEFT_IN : TI0 = SI0;
RIGHT_IN : TI1 = SI1;
COMP_OUT : SO1[0:0] = TO1[0:0];

NAME : NQNQ2;
GATE_COUNT : 0;
MAX_DELAY : 0;
SOURCE_FN : FM_NEQ;
TARGET_FN : FM_NEQ;

LEFT_IN : TI0 = SI1;
RIGHT_IN : TI1 = SI0;
COMP_OUT : SO1[0:0] = TO1[0:0];

NAME : NQEQ1;
GATE_COUNT : 1;
MAX_DELAY : 1;
SOURCE_FN : FM_NEQ;
TARGET_FN : FM_EQ;

LEFT_IN : TI0 = SI0;
RIGHT_IN : TI1 = SI1;
COMP_OUT : SO1[0:0] = NOT TO1[0:0];

NAME : NQEQ2;
GATE_COUNT : 1;
MAX_DELAY : 1;
SOURCE_FN : FM_NEQ;
TARGET_FN : FM_EQ;

LEFT_IN : TI0 = SI1;
RIGHT_IN : TI1 = SI0;
COMP_OUT : SO1[0:0] = NOT TO1[0:0];

NAME : NQAD1;
GATE_COUNT : 1;
MAX_DELAY : 1;
SOURCE_FN : FM_NEQ;
```

```
TARGET_FN : FM_ADD;

LEFT_IN : TI0 = SI0;
RIGHT_IN : TI1 = NOT SI1;
CARRY_IN : TICIN[0:0] = "1";
COMP_OUT : SO1[0:0] = GOR TO0;


NAME : NQAD2;
GATE_COUNT : 1;
MAX_DELAY : 1;
SOURCE_FN : FM_NEQ;
TARGET_FN : FM_ADD;

LEFT_IN : TI0 = NOT SI0;
RIGHT_IN : TI1 = SI1;
CARRY_IN : TICIN[0:0] = "1";
COMP_OUT : SO1[0:0] = GOR TO0;

NAME : NQAD3;
GATE_COUNT : 1;
MAX_DELAY : 1;
SOURCE_FN : FM_NEQ;
TARGET_FN : FM_ADD;

LEFT_IN : TI0 = SI1;
RIGHT_IN : TI1 = NOT SI0;
CARRY_IN : TICIN[0:0] = "1";
COMP_OUT : SO1[0:0] = GOR TO0;


NAME : NQAD4;
GATE_COUNT : 1;
MAX_DELAY : 1;
SOURCE_FN : FM_NEQ;
TARGET_FN : FM_ADD;

LEFT_IN : TI0 = NOT SI1;
RIGHT_IN : TI1 = SI0;
CARRY_IN : TICIN[0:0] = "1";
COMP_OUT : SO1[0:0] = GOR TO0;

NAME : NQSB1;
GATE_COUNT : 1;
MAX_DELAY : 1;
SOURCE_FN : FM_NEQ;
TARGET_FN : FM_SUB;

LEFT_IN : TI0 = SI0;
RIGHT_IN : TI1 = SI1;
CARRY_IN : TICIN[0:0] = "1";
COMP_OUT : SO1[0:0] = GOR TO0;

NAME : NQSB2;
```

```
GATE_COUNT : 1;
MAX_DELAY : 1;
SOURCE_FN : FM_NEQ;
TARGET_FN : FM_SUB;

LEFT_IN : TI0 = SI1;
RIGHT_IN : TI1 = SI0;
CARRY_IN : TICIN[0:0] = "1";
COMP_OUT : SO1[0:0] = GOR TO0;

NAME : NQSB3;
GATE_COUNT : 1;
MAX_DELAY : 1;
SOURCE_FN : FM_NEQ;
TARGET_FN : FM_SUB;

LEFT_IN : TI0 = NOT SI0;
RIGHT_IN : TI1 = NOT SI1;
CARRY_IN : TICIN[0:0] = "1";
COMP_OUT : SO1[0:0] = GOR TO0;

NAME : NQSB4;
GATE_COUNT : 1;
MAX_DELAY : 1;
SOURCE_FN : FM_NEQ;
TARGET_FN : FM_SUB;

LEFT_IN : TI0 = NOT SI1;
RIGHT_IN : TI1 = NOT SI0;
CARRY_IN : TICIN[0:0] = "1";
COMP_OUT : SO1[0:0] = GOR TO0;

NAME : NQRB1;
GATE_COUNT : 1;
MAX_DELAY : 1;
SOURCE_FN : FM_NEQ;
TARGET_FN : FM_RSUB;

LEFT_IN : TI0 = SI0;
RIGHT_IN : TI1 = SI1;
CARRY_IN : TICIN[0:0] = "1";
COMP_OUT : SO1[0:0] = GOR TO0;

NAME : NQRB2;
GATE_COUNT : 1;
MAX_DELAY : 1;
SOURCE_FN : FM_NEQ;
TARGET_FN : FM_RSUB;

LEFT_IN : TI0 = SI1;
RIGHT_IN : TI1 = SI0;
CARRY_IN : TICIN[0:0] = "1";
COMP_OUT : SO1[0:0] = GOR TO0;
```

```
NAME : NQRB3;
GATE_COUNT : 1;
MAX_DELAY : 1;
SOURCE_FN : FM_NEQ;
TARGET_FN : FM_RSUB;

LEFT_IN : TI0 = NOT SI0;
RIGHT_IN : TI1 = NOT SI1;
CARRY_IN : TICIN[0:0] = "1";
COMP_OUT : SO1[0:0] = GOR TO0;

NAME : NQRB4;
GATE_COUNT : 1;
MAX_DELAY : 1;
SOURCE_FN : FM_NEQ;
TARGET_FN : FM_RSUB;

LEFT_IN : TI0 = NOT SI1;
RIGHT_IN : TI1 = NOT SI0;
CARRY_IN : TICIN[0:0] = "1";
COMP_OUT : SO1[0:0] = GOR TO0;

NAME : NQXR1;
GATE_COUNT : 1;
MAX_DELAY : 1;
SOURCE_FN : FM_NEQ;
TARGET_FN : FM_XOR;

LEFT_IN : TI0 = SI0;
RIGHT_IN : TI1 = SI1;
COMP_OUT : SO1[0:0] = GOR TO0;

NAME : NQXR2;
GATE_COUNT : 1;
MAX_DELAY : 1;
SOURCE_FN : FM_NEQ;
TARGET_FN : FM_XOR;

LEFT_IN : TI0 = SI1;
RIGHT_IN : TI1 = SI0;
COMP_OUT : SO1[0:0] = GOR TO0;

NAME : NQXR3;
GATE_COUNT : 1;
MAX_DELAY : 1;
SOURCE_FN : FM_NEQ;
TARGET_FN : FM_XOR;

LEFT_IN : TI0 = NOT SI0;
RIGHT_IN : TI1 = NOT SI1;
COMP_OUT : SO1[0:0] = GOR TO0;

NAME : NQXR4;
GATE_COUNT : 1;
```

```
MAX_DELAY : 1;
SOURCE_FN : FM_NEQ;
TARGET_FN : FM_XOR;

LEFT_IN : TI0 = NOT SI1;
RIGHT_IN : TI1 = NOT SI0;
COMP_OUT : SO1[0:0] = GOR TO0;

NAME : NQXN1;
GATE_COUNT : 1;
MAX_DELAY : 1;
SOURCE_FN : FM_NEQ;
TARGET_FN : FM_XNOR;

LEFT_IN : TI0 = SI0;
RIGHT_IN : TI1 = SI1;
COMP_OUT : SO1[0:0] = GNAND TO0;

NAME : NQXN2;
GATE_COUNT : 1;
MAX_DELAY : 1;
SOURCE_FN : FM_NEQ;
TARGET_FN : FM_XNOR;

LEFT_IN : TI0 = SI1;
RIGHT_IN : TI1 = SI0;
COMP_OUT : SO1[0:0] = GNAND TO0;

NAME : NQXN3;
GATE_COUNT : 1;
MAX_DELAY : 1;
SOURCE_FN : FM_NEQ;
TARGET_FN : FM_XNOR;

LEFT_IN : TI0 = NOT SI0;
RIGHT_IN : TI1 = NOT SI1;
COMP_OUT : SO1[0:0] = GNAND TO0;

NAME : NQXN4;
GATE_COUNT : 1;
MAX_DELAY : 1;
SOURCE_FN : FM_NEQ;
TARGET_FN : FM_XNOR;

LEFT_IN : TI0 = NOT SI1;
RIGHT_IN : TI1 = NOT SI0;
COMP_OUT : SO1[0:0] = GNAND TO0;
```

A.3.3 LT

```
NAME : LTLT1;
GATE_COUNT : 1;
```

```
MAX_DELAY : 1;
SOURCE_FN : FM_LT;
TARGET_FN : FM_LT;

LEFT_IN : TI0 = SI0;
RIGHT_IN : TI1 = SI1;
COMP_OUT : SO1[0:0] =  TO1[0:0];

NAME : LTGQ;
GATE_COUNT : 1;
MAX_DELAY : 1;
SOURCE_FN : FM_LT;
TARGET_FN : FM_GQ;

LEFT_IN : TI0 = SI0;
RIGHT_IN : TI1 = SI1;
COMP_OUT : SO1[0:0] =  NOT TO1[0:0];

NAME : LTSB1;
GATE_COUNT : 1;
MAX_DELAY : 1;
SOURCE_FN : FM_LT;
TARGET_FN : FM_SUB;

LEFT_IN : TI0 = SI0;
RIGHT_IN : TI1 = SI1;
CARRY_IN : TICIN[0:0] = "1";
COMP_OUT : SO1[0:0] =  TO0[FM_WIDTH-1:FM_WIDTH-1];

NAME : LTSB2;
GATE_COUNT : 1;
MAX_DELAY : 1;
SOURCE_FN : FM_LT;
TARGET_FN : FM_SUB;

LEFT_IN : TI0 = SI1;
RIGHT_IN : TI1 = SI0;
CARRY_IN : TICIN[0:0] = "1";
COMP_OUT : SO1[0:0] =  (NOT TO0[FM_WIDTH-1:FM_WIDTH-1]) AND GNOR(TO0);

NAME : LTRB1;
GATE_COUNT : 1;
MAX_DELAY : 1;
SOURCE_FN : FM_LT;
TARGET_FN : FM_RSUB;

LEFT_IN : TI0 = SI1;
RIGHT_IN : TI1 = SI0;
CARRY_IN : TICIN[0:0] = "1";
COMP_OUT : SO1[0:0] =  TO0[FM_WIDTH-1:FM_WIDTH-1];

NAME : LTRB2;
GATE_COUNT : 1;
MAX_DELAY : 1;
```

```
SOURCE_FN : FM_LT;
TARGET_FN : FM_RSUB;

LEFT_IN : TI0 = SI0;
RIGHT_IN : TI1 = SI1;
CARRY_IN : TICIN[0:0] = "1";
COMP_OUT : SO1[0:0] =  (NOT TO0[FM_WIDTH-1:FM_WIDTH-1]) AND GNOR(TO0);

NAME : LTAD1;
GATE_COUNT : 1;
MAX_DELAY : 1;
SOURCE_FN : FM_LT;
TARGET_FN : FM_ADD;

LEFT_IN : TI0 = SI0;
RIGHT_IN : TI1 = NOT SI1;
CARRY_IN : TICIN[0:0] = "1";
COMP_OUT : SO1[0:0] =  TO0[FM_WIDTH-1:FM_WIDTH-1];

NAME : LTAD2;
GATE_COUNT : 1;
MAX_DELAY : 1;
SOURCE_FN : FM_LT;
TARGET_FN : FM_ADD;

LEFT_IN : TI0 = NOT SI1;
RIGHT_IN : TI1 = SI0;
CARRY_IN : TICIN[0:0] = "1";
COMP_OUT : SO1[0:0] =  TO0[FM_WIDTH-1:FM_WIDTH-1];

NAME : LTAD3;
GATE_COUNT : 1;
MAX_DELAY : 1;
SOURCE_FN : FM_LT;
TARGET_FN : FM_ADD;

LEFT_IN : TI0 = NOT SI0;
RIGHT_IN : TI1 = SI1;
CARRY_IN : TICIN[0:0] = "1";
COMP_OUT : SO1[0:0] =  (NOT TO0[FM_WIDTH-1:FM_WIDTH-1]) AND GNOR(TO0);

NAME : LTAD4;
GATE_COUNT : 1;
MAX_DELAY : 1;
SOURCE_FN : FM_LT;
TARGET_FN : FM_ADD;

LEFT_IN : TI0 = SI1;
RIGHT_IN : TI1 = NOT SI0;
CARRY_IN : TICIN[0:0] = "1";
COMP_OUT : SO1[0:0] =  (NOT TO0[FM_WIDTH-1:FM_WIDTH-1]) AND GNOR(TO0);
```

A.3.4 LEQ

```
NAME : LQLQ1;
GATE_COUNT : 1;
MAX_DELAY : 1;
SOURCE_FN : FM_LEQ;
TARGET_FN : FM_LEQ;

LEFT_IN : TI0 = SI0;
RIGHT_IN : TI1 = SI1;
COMP_OUT : SO1[0:0] =  TO1[0:0];

NAME : LQGT;
GATE_COUNT : 1;
MAX_DELAY : 1;
SOURCE_FN : FM_LEQ;
TARGET_FN : FM_GT;

LEFT_IN : TI0 = SI0;
RIGHT_IN : TI1 = SI1;
COMP_OUT : SO1[0:0] =  NOT TO1[0:0];

NAME : LQSB1;
GATE_COUNT : 1;
MAX_DELAY : 1;
SOURCE_FN : FM_LEQ;
TARGET_FN : FM_SUB;

LEFT_IN : TI0 = SI0;
RIGHT_IN : TI1 = SI1;
CARRY_IN : TICIN[0:0] = "1";
COMP_OUT : SO1[0:0] =  TO0[FM_WIDTH-1:FM_WIDTH-1] OR GNOR(TO0);

NAME : LQSB2;
GATE_COUNT : 1;
MAX_DELAY : 1;
SOURCE_FN : FM_LEQ;
TARGET_FN : FM_SUB;

LEFT_IN : TI0 = SI1;
RIGHT_IN : TI1 = SI0;
CARRY_IN : TICIN[0:0] = "1";
COMP_OUT : SO1[0:0] =  NOT TO0[FM_WIDTH-1:FM_WIDTH-1];

NAME : LQRB1;
GATE_COUNT : 1;
MAX_DELAY : 1;
SOURCE_FN : FM_LEQ;
TARGET_FN : FM_RSUB;

LEFT_IN : TI0 = SI1;
RIGHT_IN : TI1 = SI0;
CARRY_IN : TICIN[0:0] = "1";
COMP_OUT : SO1[0:0] =  TO0[FM_WIDTH-1:FM_WIDTH-1] OR GNOR(TO0);
```

```
NAME : LQRB2;
GATE_COUNT : 1;
MAX_DELAY : 1;
SOURCE_FN : FM_LEQ;
TARGET_FN : FM_RSUB;

LEFT_IN : TI0 = SI0;
RIGHT_IN : TI1 = SI1;
CARRY_IN : TICIN[0:0] = "1";
COMP_OUT : SO1[0:0] =  NOT TO0[FM_WIDTH-1:FM_WIDTH-1];

NAME : LQAD1;
GATE_COUNT : 1;
MAX_DELAY : 1;
SOURCE_FN : FM_LEQ;
TARGET_FN : FM_ADD;

LEFT_IN : TI0 = SI0;
RIGHT_IN : TI1 = NOT SI1;
CARRY_IN : TICIN[0:0] = "1";
COMP_OUT : SO1[0:0] =  TO0[FM_WIDTH-1:FM_WIDTH-1] OR GNOR(TO0);

NAME : LQAD2;
GATE_COUNT : 1;
MAX_DELAY : 1;
SOURCE_FN : FM_LEQ;
TARGET_FN : FM_ADD;

LEFT_IN : TI0 = NOT SI1;
RIGHT_IN : TI1 = SI0;
CARRY_IN : TICIN[0:0] = "1";
COMP_OUT : SO1[0:0] =  TO0[FM_WIDTH-1:FM_WIDTH-1] OR GNOR(TO0);

NAME : LQAD3;
GATE_COUNT : 1;
MAX_DELAY : 1;
SOURCE_FN : FM_LEQ;
TARGET_FN : FM_ADD;

LEFT_IN : TI0 = NOT SI0;
RIGHT_IN : TI1 = SI1;
CARRY_IN : TICIN[0:0] = "1";
COMP_OUT : SO1[0:0] =  NOT TO0[FM_WIDTH-1:FM_WIDTH-1];

NAME : LQAD4;
GATE_COUNT : 1;
MAX_DELAY : 1;
SOURCE_FN : FM_LEQ;
TARGET_FN : FM_ADD;

LEFT_IN : TI0 = SI1;
RIGHT_IN : TI1 = NOT SI0;
CARRY_IN : TICIN[0:0] = "1";
```

```
COMP_OUT : SO1[0:0] =  NOT TO0[FM_WIDTH-1:FM_WIDTH-1];
```

A.3.5 GEQ

```
NAME : GQGQ;
GATE_COUNT : 1;
MAX_DELAY : 1;
SOURCE_FN : FM_GEQ;
TARGET_FN : FM_GEQ;

LEFT_IN : TI0 = SI0;
RIGHT_IN : TI1 = SI1;
COMP_OUT : SO1[0:0] =  TO1[0:0];

NAME : LTGQ;
GATE_COUNT : 1;
MAX_DELAY : 1;
SOURCE_FN : FM_GEQ;
TARGET_FN : FM_LT;

LEFT_IN : TI0 = SI0;
RIGHT_IN : TI1 = SI1;
COMP_OUT : SO1[0:0]  =  NOT TO1[0:0];

NAME : GQSB1;
GATE_COUNT : 1;
MAX_DELAY : 1;
SOURCE_FN : FM_GEQ;
TARGET_FN : FM_SUB;

LEFT_IN : TI0 = SI0;
RIGHT_IN : TI1 = SI1;
CARRY_IN : TICIN[0:0] = "1";
COMP_OUT : SO1[0:0] =  (NOT TO0[FM_WIDTH-1:FM_WIDTH-1]) AND GNOR(TO0);

NAME : GQSB2;
GATE_COUNT : 1;
MAX_DELAY : 1;
SOURCE_FN : FM_GEQ;
TARGET_FN : FM_SUB;

LEFT_IN : TI0 = SI1;
RIGHT_IN : TI1 = SI0;
CARRY_IN : TICIN[0:0] = "1";
COMP_OUT : SO1[0:0] =  TO0[FM_WIDTH-1:FM_WIDTH-1];

NAME : GQRB1;
GATE_COUNT : 1;
MAX_DELAY : 1;
SOURCE_FN : FM_GEQ;
TARGET_FN : FM_RSUB;
```

```
LEFT_IN : TI0 = SI1;
RIGHT_IN : TI1 = SI0;
CARRY_IN : TICIN[0:0] = "1";
COMP_OUT : SO1[0:0] =  (NOT TO0[FM_WIDTH-1:FM_WIDTH-1]) AND GNOR(TO0);

NAME : GQRB2;
GATE_COUNT : 1;
MAX_DELAY : 1;
SOURCE_FN : FM_GEQ;
TARGET_FN : FM_RSUB;

LEFT_IN : TI0 = SI0;
RIGHT_IN : TI1 = SI1;
CARRY_IN : TICIN[0:0] = "1";
COMP_OUT : SO1[0:0] =  TO0[FM_WIDTH-1:FM_WIDTH-1];

NAME : GQAD1;
GATE_COUNT : 1;
MAX_DELAY : 1;
SOURCE_FN : FM_GEQ;
TARGET_FN : FM_ADD;

LEFT_IN : TI0 = SI0;
RIGHT_IN : TI1 = NOT SI1;
CARRY_IN : TICIN[0:0] = "1";
COMP_OUT : SO1[0:0] =  (NOT TO0[FM_WIDTH-1:FM_WIDTH-1]) AND GNOR(TO0);

NAME : GQAD2;
GATE_COUNT : 1;
MAX_DELAY : 1;
SOURCE_FN : FM_GEQ;
TARGET_FN : FM_ADD;

LEFT_IN : TI0 = NOT SI1;
RIGHT_IN : TI1 = SI0;
CARRY_IN : TICIN[0:0] = "1";
COMP_OUT : SO1[0:0] =  (NOT TO0[FM_WIDTH-1:FM_WIDTH-1]) AND GNOR(TO0);

NAME : GQAD3;
GATE_COUNT : 1;
MAX_DELAY : 1;
SOURCE_FN : FM_GEQ;
TARGET_FN : FM_ADD;

LEFT_IN : TI0 = NOT SI0;
RIGHT_IN : TI1 = SI1;
CARRY_IN : TICIN[0:0] = "1";
COMP_OUT : SO1[0:0] =  TO0[FM_WIDTH-1:FM_WIDTH-1];

NAME : GQAD4;
GATE_COUNT : 1;
MAX_DELAY : 1;
SOURCE_FN : FM_GEQ;
TARGET_FN : FM_ADD;
```

```
LEFT_IN : TI0 = SI1;
RIGHT_IN : TI1 = NOT SI0;
CARRY_IN : TICIN[0:0] = "1";
COMP_OUT : SO1[0:0] =  TO0[FM_WIDTH-1:FM_WIDTH-1];
```

A.3.6 GT

```
NAME : GTGT;
GATE_COUNT : 1;
MAX_DELAY : 1;
SOURCE_FN : FM_GT;
TARGET_FN : FM_GT;

LEFT_IN : TI0 = SI0;
RIGHT_IN : TI1 = SI1;
COMP_OUT : SO1[0:0] =  TO1[0:0];

NAME : GTLQ;
GATE_COUNT : 1;
MAX_DELAY : 1;
SOURCE_FN : FM_GT;
TARGET_FN : FM_LEQ;

LEFT_IN : TI0 = SI0;
RIGHT_IN : TI1 = SI1;
COMP_OUT : SO1[0:0] =  NOT TO1[0:0];

NAME : GTSB1;
GATE_COUNT : 1;
MAX_DELAY : 1;
SOURCE_FN : FM_GT;
TARGET_FN : FM_SUB;

LEFT_IN : TI0 = SI0;
RIGHT_IN : TI1 = SI1;
CARRY_IN : TICIN[0:0] = "1";
COMP_OUT : SO1[0:0] =  NOT TO0[FM_WIDTH-1:FM_WIDTH-1];

NAME : GTSB2;
GATE_COUNT : 1;
MAX_DELAY : 1;
SOURCE_FN : FM_GT;
TARGET_FN : FM_SUB;

LEFT_IN : TI0 = SI1;
RIGHT_IN : TI1 = SI0;
CARRY_IN : TICIN[0:0] = "1";
COMP_OUT : SO1[0:0] =  TO0[FM_WIDTH-1:FM_WIDTH-1] OR GNOR(TO0);

NAME : GTRB1;
GATE_COUNT : 1;
```

```
MAX_DELAY : 1;
SOURCE_FN : FM_GT;
TARGET_FN : FM_RSUB;

LEFT_IN : TI0 = SI1;
RIGHT_IN : TI1 = SI0;
CARRY_IN : TICIN[0:0] = "1";
COMP_OUT : SO1[0:0] =  NOT TO0[FM_WIDTH-1:FM_WIDTH-1];

NAME : GTRB2;
GATE_COUNT : 1;
MAX_DELAY : 1;
SOURCE_FN : FM_GT;
TARGET_FN : FM_RSUB;

LEFT_IN : TI0 = SI0;
RIGHT_IN : TI1 = SI1;
CARRY_IN : TICIN[0:0] = "1";
COMP_OUT : SO1[0:0] =  TO0[FM_WIDTH-1:FM_WIDTH-1] OR GNOR(TO0);

NAME : GTAD1;
GATE_COUNT : 1;
MAX_DELAY : 1;
SOURCE_FN : FM_GT;
TARGET_FN : FM_ADD;

LEFT_IN : TI0 = SI0;
RIGHT_IN : TI1 = NOT SI1;
CARRY_IN : TICIN[0:0] = "1";
COMP_OUT : SO1[0:0] =  NOT TO0[FM_WIDTH-1:FM_WIDTH-1];

NAME : GTAD2;
GATE_COUNT : 1;
MAX_DELAY : 1;
SOURCE_FN : FM_GT;
TARGET_FN : FM_ADD;

LEFT_IN : TI0 = NOT SI1;
RIGHT_IN : TI1 = SI0;
CARRY_IN : TICIN[0:0] = "1";
COMP_OUT : SO1[0:0] =  NOT TO0[FM_WIDTH-1:FM_WIDTH-1];

NAME : GTAD3;
GATE_COUNT : 1;
MAX_DELAY : 1;
SOURCE_FN : FM_GT;
TARGET_FN : FM_ADD;

LEFT_IN : TI0 = NOT SI0;
RIGHT_IN : TI1 = SI1;
CARRY_IN : TICIN[0:0] = "1";
COMP_OUT : SO1[0:0] =  TO0[FM_WIDTH-1:FM_WIDTH-1] OR GNOR(TO0);

NAME : GTAD4;
```

```
GATE_COUNT : 1;
MAX_DELAY : 1;
SOURCE_FN : FM_GT;
TARGET_FN : FM_ADD;

LEFT_IN : TI0 = SI1;
RIGHT_IN : TI1 = NOT SI0;
CARRY_IN : TICIN[0:0] = "1";
COMP_OUT : SO1[0:0] =  TO0[FM_WIDTH-1:FM_WIDTH-1] OR GNOR(TO0);
```

B Detailed result

This section describes the designs produced for each of the seven examples discussed in section 6.

B.1 Example 1

```
Source component :

NAME : ALU2;
BIT_WIDTH : 32;
GATE_COUNT : 8;
MAX_DELAY : 8;

FM_ADD ::
LEFT_IN : I0 = A;
RIGHT_IN : I1 = B;
CARRY_IN : ICIN[0:0] = Cin[0:0];
MAIN_OUT : C = O0;
CARRY_OUT : Cout[0:0] = OCOUT[0:0];

FM_SUB ::
LEFT_IN : I0 = A;
RIGHT_IN : I1 = B;
CARRY_IN : ICIN[0:0] = Cin[0:0];
MAIN_OUT : C = O0;
CARRY_OUT : Cout[0:0] = OCOUT[0:0];

FM_EQ ::
LEFT_IN : I0 = A;
RIGHT_IN : I1 = B;
COMP_OUT : Comp[0:0] = O1[0:0];

FM_LT ::
LEFT_IN : I0 = A;
RIGHT_IN : I1 = B;
COMP_OUT : Comp[0:0] = O1[0:0];



Target component :

NAME : VDP1ASB001;
BIT_WIDTH : 32;
GATE_COUNT : 448;
MAX_DELAY : 27;

FM_ADD ::
LEFT_IN : I0 = A;
RIGHT_IN : I1 = B;
```

```
CARRY_IN : ICIN[0:0] = INST_CIN[0:0];
MAIN_OUT : SO = OO;
CARRY_OUT : INST_COUT[0:0] = OCOUT[0:0];
OVF_OUT : INST_OVR[0:0] = OVF[0:0];

FM_SUB ::
LEFT_IN : I0 = A;
RIGHT_IN : I1 = B;
CARRY_IN : ICIN[0:0] = INST_CIN[0:0];
MAIN_OUT : SO = OO;
CARRY_OUT : INST_COUT[0:0] = OCOUT[0:0];
OVF_OUT : INST_OVR[0:0] = OVF[0:0];


Solutions are :


Cost Function :  Gate_Count
Algorithm Type :  1-greedy
Time : 7.1 real         4.5 user         0.6 sys

NAME : CANONICAL;
TARGET COMPONENT : VDP1ASB001;
BIT_WIDTH : 32;
GATE_COUNT : 37;
MAX_DELAY :  56.40;

FUNCTION-RULE LIST : (FM_ADD, AA1), (FM_SUB, SS), (FM_EQ, EQSB1), (FM_LT, LTSB1), ;

LEFT_IN :: TI0[31:0] = A[31:0] (FM_ADD, FM_SUB, FM_EQ, FM_LT, ), ;
RIGHT_IN :: TI1[31:0] = B[31:0] (FM_ADD, FM_SUB, FM_EQ, FM_LT, ), ;
MAIN_OUT :: SO0[31:0] = O0[31:0] (FM_ADD, FM_SUB, ), ;
CARRY_IN :: TICIN[0:0] = Cin[0:0] (FM_ADD, FM_SUB, ), "1" (FM_EQ, FM_LT, ), ;
CARRY_OUT :: SOCOUT[0:0] = OCOUT[0:0] (FM_ADD, FM_SUB, ), ;
COMP_OUT :: SO1[0:0] = (TO0[31:0] GNOR ) (FM_EQ, ), TO0[31:31] (FM_LT, ), ;


Cost Function :  Max_Delay
Algorithm Type :  1-Greedy
Time : 5.1 real         4.5 user         0.3 sys


NAME : CANONICAL;
TARGET COMPONENT : VDP1ASB001;
BIT_WIDTH : 32;
GATE_COUNT : 37;
MAX_DELAY :  56.40;

FUNCTION-RULE LIST : (FM_ADD, AA1), (FM_SUB, SS), (FM_EQ, EQSB1), (FM_LT, LTSB1), ;

LEFT_IN :: TI0[31:0] = A[31:0] (FM_ADD, FM_SUB, FM_EQ, FM_LT, ), ;
RIGHT_IN :: TI1[31:0] = B[31:0] (FM_ADD, FM_SUB, FM_EQ, FM_LT, ), ;
MAIN_OUT :: SO0[31:0] = O0[31:0] (FM_ADD, FM_SUB, ), ;
CARRY_IN :: TICIN[0:0] = Cin[0:0] (FM_ADD, FM_SUB, ), "1" (FM_EQ, FM_LT, ), ;
```

```
CARRY_OUT :: SOCOUT[0:0] = OCOUT[0:0] (FM_ADD, FM_SUB, ), ;
COMP_OUT :: SO1[0:0] = (TOO[31:0] GNOR ) (FM_EQ, ), TOO[31:31] (FM_LT, ), ;


Cost Function :  Gate-Count
Algorithm :   Dynamic Programming (1)
Time :   5.1 real         4.6 user         0.3 sys

NAME : CANONICAL;
TARGET COMPONENT : VDP1ASB001;
BIT_WIDTH : 32;
GATE_COUNT : 37;
MAX_DELAY :  56.40;

FUNCTION-RULE LIST : (FM_ADD, AA1), (FM_SUB, SS), (FM_EQ, EQSB1), (FM_LT, LTSB1), ;

LEFT_IN :: TI0[31:0] = A[31:0] (FM_ADD, FM_SUB, FM_EQ, FM_LT, ), ;
RIGHT_IN :: TI1[31:0] = B[31:0] (FM_ADD, FM_SUB, FM_EQ, FM_LT, ), ;
MAIN_OUT :: SO0[31:0] = O0[31:0] (FM_ADD, FM_SUB, ), ;
CARRY_IN :: TICIN[0:0] = Cin[0:0] (FM_ADD, FM_SUB, ), "1" (FM_EQ, FM_LT, ), ;
CARRY_OUT :: SOCOUT[0:0] = OCOUT[0:0] (FM_ADD, FM_SUB, ), ;
COMP_OUT :: SO1[0:0] = (TOO[31:0] GNOR ) (FM_EQ, ), TOO[31:31] (FM_LT, ), ;


Cost Function :  Gate-Count
Algorithm :   Dynamic Programming (2)
Time :   5.8 real         5.2 user         0.3 sys


NAME : CANONICAL;
TARGET COMPONENT : VDP1ASB001;
BIT_WIDTH : 32;
GATE_COUNT : 37;
MAX_DELAY :  56.40;

FUNCTION-RULE LIST : (FM_ADD, AA1), (FM_SUB, SS), (FM_EQ, EQSB1), (FM_LT, LTSB1), ;

LEFT_IN :: TI0[31:0] = A[31:0] (FM_ADD, FM_SUB, FM_EQ, FM_LT, ), ;
RIGHT_IN :: TI1[31:0] = B[31:0] (FM_ADD, FM_SUB, FM_EQ, FM_LT, ), ;
MAIN_OUT :: SO0[31:0] = O0[31:0] (FM_ADD, FM_SUB, ), ;
CARRY_IN :: TICIN[0:0] = Cin[0:0] (FM_ADD, FM_SUB, ), "1" (FM_EQ, FM_LT, ), ;
CARRY_OUT :: SOCOUT[0:0] = OCOUT[0:0] (FM_ADD, FM_SUB, ), ;
COMP_OUT :: SO1[0:0] = (TOO[31:0] GNOR ) (FM_EQ, ), TOO[31:31] (FM_LT, ), ;


Cost Function :  Gate-Count
Algorithm :   Dynamic Programming (4)
Time :   8.5 REAL         7.8 USER         0.3 sys


NAME : CANONICAL;
TARGET COMPONENT : VDP1ASB001;
BIT_WIDTH : 32;
GATE_COUNT : 37;
```

```
MAX_DELAY :  56.40;

FUNCTION-RULE LIST : (FM_ADD, AA1), (FM_SUB, SS), (FM_EQ, EQSB1), (FM_LT, LTSB1), ;

LEFT_IN :: TI0[31:0] = A[31:0] (FM_ADD, FM_SUB, FM_EQ, FM_LT, ), ;
RIGHT_IN :: TI1[31:0] = B[31:0] (FM_ADD, FM_SUB, FM_EQ, FM_LT, ), ;
MAIN_OUT :: SO0[31:0] = O0[31:0] (FM_ADD, FM_SUB, ), ;
CARRY_IN :: TICIN[0:0] = Cin[0:0] (FM_ADD, FM_SUB, ), "1" (FM_EQ, FM_LT, ), ;
CARRY_OUT :: SOCOUT[0:0] = OCOUT[0:0] (FM_ADD, FM_SUB, ), ;
COMP_OUT :: SO1[0:0] = (TO0[31:0] GNOR ) (FM_EQ, ), TO0[31:31] (FM_LT, ), ;


Cost Function :  Max_Delay
Algorithm :   Dynamic Programming (1)
Time :   5.5 real         4.5 user         0.5 sys


NAME : CANONICAL;
TARGET COMPONENT : VDP1ASB001;
BIT_WIDTH : 32;
GATE_COUNT : 37;
MAX_DELAY :  56.40;

FUNCTION-RULE LIST : (FM_ADD, AA1), (FM_SUB, SS), (FM_EQ, EQSB1), (FM_LT, LTSB1), ;

LEFT_IN :: TI0[31:0] = A[31:0] (FM_ADD, FM_SUB, FM_EQ, FM_LT, ), ;
RIGHT_IN :: TI1[31:0] = B[31:0] (FM_ADD, FM_SUB, FM_EQ, FM_LT, ), ;
MAIN_OUT :: SO0[31:0] = O0[31:0] (FM_ADD, FM_SUB, ), ;
CARRY_IN :: TICIN[0:0] = Cin[0:0] (FM_ADD, FM_SUB, ), "1" (FM_EQ, FM_LT, ), ;
CARRY_OUT :: SOCOUT[0:0] = OCOUT[0:0] (FM_ADD, FM_SUB, ), ;
COMP_OUT :: SO1[0:0] = (TO0[31:0] GNOR ) (FM_EQ, ), TO0[31:31] (FM_LT, ), ;


Cost Function :  Max_Delay
Algorithm :   Dynamic Programming (2)
Time :   5.7 real         5.1 user         0.3 sys


NAME : CANONICAL;
TARGET COMPONENT : VDP1ASB001;
BIT_WIDTH : 32;
GATE_COUNT : 37;
MAX_DELAY :  56.40;

FUNCTION-RULE LIST : (FM_ADD, AA1), (FM_SUB, SS), (FM_EQ, EQSB1), (FM_LT, LTSB1), ;

LEFT_IN :: TI0[31:0] = A[31:0] (FM_ADD, FM_SUB, FM_EQ, FM_LT, ), ;
RIGHT_IN :: TI1[31:0] = B[31:0] (FM_ADD, FM_SUB, FM_EQ, FM_LT, ), ;
MAIN_OUT :: SO0[31:0] = O0[31:0] (FM_ADD, FM_SUB, ), ;
CARRY_IN :: TICIN[0:0] = Cin[0:0] (FM_ADD, FM_SUB, ), "1" (FM_EQ, FM_LT, ), ;
CARRY_OUT :: SOCOUT[0:0] = OCOUT[0:0] (FM_ADD, FM_SUB, ), ;
COMP_OUT :: SO1[0:0] = (TO0[31:0] GNOR ) (FM_EQ, ), TO0[31:31] (FM_LT, ), ;
```

```
Cost Function :  Max_Delay
Algorithm :   Dynamic Programming (4)
Time :   8.3 real         7.7 user         0.4 sys
```

B.2 Example 2

```
Source component :


NAME : ALU1;
BIT_WIDTH : 32;
GATE_COUNT : 8;
MAX_DELAY : 8;

FM_ADD ::
LEFT_IN : I0 = A;
RIGHT_IN : I1 = B;
CARRY_IN : ICIN[0:0] = Cin[0:0];
MAIN_OUT : C = O0;
CARRY_OUT : Cout[0:0] = OCOUT[0:0];

FM_SUB ::
LEFT_IN : I0 = A;
RIGHT_IN : I1 = B;
CARRY_IN : ICIN[0:0] = Cin[0:0];
MAIN_OUT : C = O0;
CARRY_OUT : Cout[0:0] = OCOUT[0:0];

FM_RSUB ::
LEFT_IN : I0 = A;
RIGHT_IN : I1 = B;
CARRY_IN : ICIN[0:0] = Cin[0:0];
MAIN_OUT : C = O0;
CARRY_OUT : Cout[0:0] = OCOUT[0:0];

FM_INC ::
LEFT_IN : I0 = A;
MAIN_OUT : C = O0;

FM_DEC ::
LEFT_IN : I0 = A;
MAIN_OUT : C = O0;



Target component :


NAME : ADDER;
BIT_WIDTH : 32;
GATE_COUNT : 672;
MAX_DELAY : 17.6;
```

```
FM_ADD ::
LEFT_IN : I0 = A;
RIGHT_IN : I1 = B;
CARRY_IN : ICIN[0:0] = INST_CIN[0:0];
MAIN_OUT : S0 = O0;
CARRY_OUT : INST_COUT[0:0] = OCOUT[0:0];
OVF_OUT : INST_OVR[0:0] = OVF[0:0];



Solution is :


Cost Function :  Gate_Count
Algorithm Type :  1-greedy
Time : 5.2 real         3.8 user         0.3 sys

NAME : CANONICAL;
TARGET COMPONENT : ADDER;
BIT_WIDTH : 32;
GATE_COUNT : 387;
MAX_DELAY :  24.50;

FUNCTION-RULE LIST : (FM_ADD, AA1), (FM_SUB, SA1), (FM_RSUB, RA1), (FM_INC, IA1)
, (FM_DEC, DA3), ;

LEFT_IN :: TI0[31:0] = A[31:0] (FM_ADD, FM_SUB, FM_INC, FM_DEC, ), (A[31:0] NOT
) (FM_RSUB, ), ;
RIGHT_IN :: TI1[31:0] = B[31:0] (FM_ADD, FM_RSUB, ), (B[31:0] NOT ) (FM_SUB, ),
("0" REPEAT 32) (FM_INC, ), (("1" REPEAT 31) CONCAT "0") (FM_DEC, ), ;
MAIN_OUT :: SO0[31:0] = O0[31:0] (FM_ADD, FM_SUB, FM_RSUB, FM_INC, FM_DEC, ), ;
CARRY_IN :: TICIN[0:0] = Cin[0:0] (FM_ADD, FM_SUB, FM_RSUB, ), "1" (FM_INC, FM_D
EC, ), ;
CARRY_OUT :: SOCOUT[0:0] = OCOUT[0:0] (FM_ADD, FM_SUB, FM_RSUB, ), ;



Cost Function :  Max_Delay
Algorithm Type :  1-Greedy
Time : 4.5 real         3.7 user         0.5 sys

NAME : CANONICAL;
TARGET COMPONENT : ADDER;
BIT_WIDTH : 32;
GATE_COUNT : 387;
MAX_DELAY :  24.50;

FUNCTION-RULE LIST : (FM_ADD, AA1), (FM_SUB, SA1), (FM_RSUB, RA1), (FM_INC, IA1)
, (FM_DEC, DA3), ;

LEFT_IN :: TI0[31:0] = A[31:0] (FM_ADD, FM_SUB, FM_INC, FM_DEC, ), (A[31:0] NOT
) (FM_RSUB, ), ;
RIGHT_IN :: TI1[31:0] = B[31:0] (FM_ADD, FM_RSUB, ), (B[31:0] NOT ) (FM_SUB, ),
```

```
("0" REPEAT 32) (FM_INC, ), (("1" REPEAT 31) CONCAT "0") (FM_DEC, ), ;
MAIN_OUT :: SO0[31:0] = O0[31:0] (FM_ADD, FM_SUB, FM_RSUB, FM_INC, FM_DEC, ), ;
CARRY_IN :: TICIN[0:0] = Cin[0:0] (FM_ADD, FM_SUB, FM_RSUB, ), "1" (FM_INC, FM_D
EC, ), ;
CARRY_OUT :: SOCOUT[0:0] = OCOUT[0:0] (FM_ADD, FM_SUB, FM_RSUB, ), ;


Cost Function :  Gate-Count
Algorithm :   Dynamic Programming (1)
Time :   4.8 real         4.0 user        0.3 sys


NAME : CANONICAL;
TARGET COMPONENT : ADDER;
BIT_WIDTH : 32;
GATE_COUNT : 389;
MAX_DELAY :  24.50;

FUNCTION-RULE LIST : (FM_ADD, AA1), (FM_SUB, SA1), (FM_RSUB, RA1), (FM_INC, IA1)
, (FM_DEC, DA1), ;

LEFT_IN :: TI0[31:0] = A[31:0] (FM_ADD, FM_SUB, FM_INC, FM_DEC, ), (A[31:0] NOT
) (FM_RSUB, ), ;
RIGHT_IN :: TI1[31:0] = B[31:0] (FM_ADD, FM_RSUB, ), (B[31:0] NOT ) (FM_SUB, ),
("0" REPEAT 32) (FM_INC, ), ("1" REPEAT 32) (FM_DEC, ), ;
MAIN_OUT :: SO0[31:0] = O0[31:0] (FM_ADD, FM_SUB, FM_RSUB, FM_INC, FM_DEC, ), ;
CARRY_IN :: TICIN[0:0] = Cin[0:0] (FM_ADD, FM_SUB, FM_RSUB, ), "1" (FM_INC, ), "
0" (FM_DEC, ), ;
CARRY_OUT :: SOCOUT[0:0] = OCOUT[0:0] (FM_ADD, FM_SUB, FM_RSUB, ), ;


Cost Function :  Gate-Count
Algorithm :   Dynamic Programming (2)
Time :   5.7 real         5.2 user        0.3 sys

NAME : CANONICAL;
TARGET COMPONENT : ADDER;
BIT_WIDTH : 32;
GATE_COUNT : 389;
MAX_DELAY :  24.50;

FUNCTION-RULE LIST : (FM_ADD, AA1), (FM_SUB, SA1), (FM_RSUB, RA1), (FM_INC, IA1)
, (FM_DEC, DA1), ;

LEFT_IN :: TI0[31:0] = A[31:0] (FM_ADD, FM_SUB, FM_INC, FM_DEC, ), (A[31:0] NOT
) (FM_RSUB, ), ;
RIGHT_IN :: TI1[31:0] = B[31:0] (FM_ADD, FM_RSUB, ), (B[31:0] NOT ) (FM_SUB, ),
("0" REPEAT 32) (FM_INC, ), ("1" REPEAT 32) (FM_DEC, ), ;
MAIN_OUT :: SO0[31:0] = O0[31:0] (FM_ADD, FM_SUB, FM_RSUB, FM_INC, FM_DEC, ), ;
CARRY_IN :: TICIN[0:0] = Cin[0:0] (FM_ADD, FM_SUB, FM_RSUB, ), "1" (FM_INC, ), "
0" (FM_DEC, ), ;
CARRY_OUT :: SOCOUT[0:0] = OCOUT[0:0] (FM_ADD, FM_SUB, FM_RSUB, ), ;

Cost Function :  Gate-Count
```

```
Algorithm :   Dynamic Programming (4)
Time :   10.3 real         9.5 user        0.4 sys

NAME : CANONICAL;
TARGET COMPONENT : ADDER;
BIT_WIDTH : 32;
GATE_COUNT : 387;
MAX_DELAY :  24.50;

FUNCTION-RULE LIST : (FM_ADD, AA1), (FM_SUB, SA1), (FM_RSUB, RA1), (FM_INC, IA1)
, (FM_DEC, DA3), ;

LEFT_IN :: TI0[31:0] = A[31:0] (FM_ADD, FM_SUB, FM_INC, FM_DEC, ), (A[31:0] NOT
) (FM_RSUB, ), ;
RIGHT_IN :: TI1[31:0] = B[31:0] (FM_ADD, FM_RSUB, ), (B[31:0] NOT ) (FM_SUB, ),
("0" REPEAT 32) (FM_INC, ), (("1" REPEAT 31) CONCAT "0") (FM_DEC, ), ;
MAIN_OUT :: SO0[31:0] = O0[31:0] (FM_ADD, FM_SUB, FM_RSUB, FM_INC, FM_DEC, ), ;
CARRY_IN :: TICIN[0:0] = Cin[0:0] (FM_ADD, FM_SUB, FM_RSUB, ), "1" (FM_INC, FM_D
EC, ), ;
CARRY_OUT :: SOCOUT[0:0] = OCOUT[0:0] (FM_ADD, FM_SUB, FM_RSUB, ), ;


Cost Function :  Max_Delay
Algorithm :   Dynamic Programming (1)
Time :   4.7 real         4.1 user        0.3 sys

NAME : CANONICAL;
TARGET COMPONENT : ADDER;
BIT_WIDTH : 32;
GATE_COUNT : 389;
MAX_DELAY :  24.50;

FUNCTION-RULE LIST : (FM_ADD, AA1), (FM_SUB, SA1), (FM_RSUB, RA1), (FM_INC, IA1)
, (FM_DEC, DA1), ;

LEFT_IN :: TI0[31:0] = A[31:0] (FM_ADD, FM_SUB, FM_INC, FM_DEC, ), (A[31:0] NOT
) (FM_RSUB, ), ;
RIGHT_IN :: TI1[31:0] = B[31:0] (FM_ADD, FM_RSUB, ), (B[31:0] NOT ) (FM_SUB, ),
("0" REPEAT 32) (FM_INC, ), ("1" REPEAT 32) (FM_DEC, ), ;
MAIN_OUT :: SO0[31:0] = O0[31:0] (FM_ADD, FM_SUB, FM_RSUB, FM_INC, FM_DEC, ), ;
CARRY_IN :: TICIN[0:0] = Cin[0:0] (FM_ADD, FM_SUB, FM_RSUB, ), "1" (FM_INC, ), "
0" (FM_DEC, ), ;
CARRY_OUT :: SOCOUT[0:0] = OCOUT[0:0] (FM_ADD, FM_SUB, FM_RSUB, ), ;


Cost Function :  Max_Delay
Algorithm :   Dynamic Programming (2)
Time :   6.9 real         5.2 user        0.4 sys

NAME : CANONICAL;
TARGET COMPONENT : ADDER;
BIT_WIDTH : 32;
GATE_COUNT : 389;
MAX_DELAY :  24.50;
```

```
FUNCTION-RULE LIST : (FM_ADD, AA1), (FM_SUB, SA1), (FM_RSUB, RA1), (FM_INC, IA1)
, (FM_DEC, DA1), ;

LEFT_IN :: TI0[31:0] = A[31:0] (FM_ADD, FM_SUB, FM_INC, FM_DEC, ), (A[31:0] NOT
) (FM_RSUB, ), ;
RIGHT_IN :: TI1[31:0] = B[31:0] (FM_ADD, FM_RSUB, ), (B[31:0] NOT ) (FM_SUB, ),
("0" REPEAT 32) (FM_INC, ), ("1" REPEAT 32) (FM_DEC, ), ;
MAIN_OUT :: SO0[31:0] = O0[31:0] (FM_ADD, FM_SUB, FM_RSUB, FM_INC, FM_DEC, ), ;
CARRY_IN :: TICIN[0:0] = Cin[0:0] (FM_ADD, FM_SUB, FM_RSUB, ), "1" (FM_INC, ), "
0" (FM_DEC, ), ;
CARRY_OUT :: SOCOUT[0:0] = OCOUT[0:0] (FM_ADD, FM_SUB, FM_RSUB, ), ;


Cost Function :  Max_Delay
Algorithm :   Dynamic Programming (4)
Time :   11.0 real         9.5 user         0.3 sys

NAME : CANONICAL;
TARGET COMPONENT : ADDER;
BIT_WIDTH : 32;
GATE_COUNT : 387;
MAX_DELAY :  24.50;

FUNCTION-RULE LIST : (FM_ADD, AA1), (FM_SUB, SA1), (FM_RSUB, RA1), (FM_INC, IA1)
, (FM_DEC, DA3), ;

LEFT_IN :: TI0[31:0] = A[31:0] (FM_ADD, FM_SUB, FM_INC, FM_DEC, ), (A[31:0] NOT
) (FM_RSUB, ), ;
RIGHT_IN :: TI1[31:0] = B[31:0] (FM_ADD, FM_RSUB, ), (B[31:0] NOT ) (FM_SUB, ),
("0" REPEAT 32) (FM_INC, ), (("1" REPEAT 31) CONCAT "0") (FM_DEC, ), ;
MAIN_OUT :: SO0[31:0] = O0[31:0] (FM_ADD, FM_SUB, FM_RSUB, FM_INC, FM_DEC, ), ;
CARRY_IN :: TICIN[0:0] = Cin[0:0] (FM_ADD, FM_SUB, FM_RSUB, ), "1" (FM_INC, FM_D
EC, ), ;
```

B.3 Example 3

```
Source component :


NAME : VDP5ALU001;
BIT_WIDTH : 48;
GATE_COUNT : 912;
MAX_DELAY : 36.3;

FM_ADD ::
LEFT_IN : I0 = I0;
RIGHT_IN : I1 = I1;
CARRY_IN : ICIN[0:0] = ICIN[0:0];
MAIN_OUT : O0 = O0;
CARRY_OUT : OCOUT[0:0] = OCOUT[0:0];
OVF_OUT : OVF[0:0] = OVF[0:0];
```

```
FM_SUB ::
LEFT_IN : I0 = I0;
RIGHT_IN : I1 = I1;
CARRY_IN : ICIN[0:0] = ICIN[0:0];
MAIN_OUT : O0 = O0;
CARRY_OUT : OCOUT[0:0] = OCOUT[0:0];
OVF_OUT : OVF[0:0] = OVF[0:0];

FM_AND ::
LEFT_IN : I0 =  I0;
RIGHT_IN : I1 = I1;
MAIN_OUT : O0 = O0;

FM_NAND ::
LEFT_IN : I0 =  I0;
RIGHT_IN : I1 = I1;
MAIN_OUT : O0 = O0;

FM_OR ::
LEFT_IN : I0 =  I0;
RIGHT_IN : I1 = I1;
MAIN_OUT : O0 = O0;

FM_NOR ::
LEFT_IN : I0 =  I0;
RIGHT_IN : I1 = I1;
MAIN_OUT : O0 = O0;

FM_XOR ::
LEFT_IN : I0 =  I0;
RIGHT_IN : I1 = I1;
MAIN_OUT : O0 = O0;

FM_XNOR ::
LEFT_IN : I0 =  I0;
RIGHT_IN : I1 = I1;
MAIN_OUT : O0 = O0;

FM_LID ::
LEFT_IN : I0 =  I0;
RIGHT_IN : I1 = I1;
MAIN_OUT : O0 = O0;

FM_RID ::
LEFT_IN : I0 =  I0;
RIGHT_IN : I1 = I1;
MAIN_OUT : O0 = O0;

FM_LNOT ::
LEFT_IN : I0 =  I0;
RIGHT_IN : I1 = I1;
MAIN_OUT : O0 = O0;
```

```
FM_RNOT ::
LEFT_IN : I0 =  I0;
RIGHT_IN : I1 = I1;
MAIN_OUT : O0 = O0;

FM_ONE ::
MAIN_OUT : O0 = O0;

FM_NEQ ::
LEFT_IN : I0 = I0;
RIGHT_IN : I1 = I1;
COMP_OUT : O1 = O1;

FM_GT ::
LEFT_IN : I0 = I0;
RIGHT_IN : I1 = I1;
COMP_OUT : O1 = O1;


Target component :


NAME : VDP5ALU001;
BIT_WIDTH : 48;
GATE_COUNT : 912;
MAX_DELAY : 36.3;

FM_ADD ::
LEFT_IN : I0 = A;
RIGHT_IN : I1 = B;
CARRY_IN : ICIN[0:0] = INST_CIN[0:0];
MAIN_OUT : SO = O0;
CARRY_OUT : INST_COUT[0:0] = OCOUT[0:0];
OVF_OUT : INST_OVR[0:0] = OVF[0:0];

FM_SUB ::
LEFT_IN : I0 = A;
RIGHT_IN : I1 = B;
CARRY_IN : ICIN[0:0] = INST_CIN[0:0];
MAIN_OUT : SO = O0;
CARRY_OUT : INST_COUT[0:0] = OCOUT[0:0];
OVF_OUT : INST_OVR[0:0] = OVF[0:0];

FM_RSUB ::
LEFT_IN : I0 = A;
RIGHT_IN : I1 = B;
CARRY_IN : ICIN[0:0] = INST_CIN[0:0];
MAIN_OUT : SO = O0;
CARRY_OUT : INST_COUT[0:0] = OCOUT[0:0];
OVF_OUT : INST_OVR[0:0] = OVF[0:0];

FM_SUB ::
LEFT_IN : I0 = NOT A;
RIGHT_IN : I1 = B;
```

```
CARRY_IN : ICIN[0:0] = INST_CIN[0:0];
MAIN_OUT : SO = OO;
CARRY_OUT : INST_COUT[0:0] = OCOUT[0:0];
OVF_OUT : INST_OVR[0:0] = OVF[0:0];


FM_AND ::
LEFT_IN : I0 =  A;
RIGHT_IN : I1 = B;
MAIN_OUT : SO = OO;

FM_NAND ::
LEFT_IN : I0 =  A;
RIGHT_IN : I1 = B;
MAIN_OUT : SO = OO;

FM_OR ::
LEFT_IN : I0 =  A;
RIGHT_IN : I1 = B;
MAIN_OUT : SO = OO;

FM_NOR ::
LEFT_IN : I0 =  A;
RIGHT_IN : I1 = B;
MAIN_OUT : SO = OO;

FM_XOR ::
LEFT_IN : I0 =  A;
RIGHT_IN : I1 = B;
MAIN_OUT : SO = OO;

FM_XNOR ::
LEFT_IN : I0 =  A;
RIGHT_IN : I1 = B;
MAIN_OUT : SO = OO;

FM_RIMPL ::
LEFT_IN : I0 =  A;
RIGHT_IN : I1 = B;
MAIN_OUT : SO = OO;

FM_LIMPL ::
LEFT_IN : I0 =  A;
RIGHT_IN : I1 = B;
MAIN_OUT : SO = OO;

FM_LINHI ::
LEFT_IN : I0 =  A;
RIGHT_IN : I1 = B;
MAIN_OUT : SO = OO;

FM_RINHI ::
LEFT_IN : I0 =  A;
RIGHT_IN : I1 = B;
```

```
MAIN_OUT : SO = OO;


Solution is :


Cost Function :  Gate_Count
Algorithm Type :  1-greedy
Time : 17.1 real         7.1 user         1.4 sys

NAME : CANONICAL;
TARGET COMPONENT : VDP5ALU001;
BIT_WIDTH : 48;
GATE_COUNT : 630;
MAX_DELAY :  72.10;

FUNCTION-RULE LIST : (FM_ADD, AA1), (FM_SUB, SS), (FM_AND, ADAD1), (FM_NAND, NDN
D1), (FM_OR, OROR1), (FM_NOR, NRNR1), (FM_XOR, XRXR1), (FM_XNOR, XNXN1), (FM_LID
, LDAD1), (FM_RID, RDAD1), (FM_LNOT, LNAD1), (FM_RNOT, RNAD1), (FM_ONE, OEOR2),
(FM_NEQ, NQXR1), (FM_GT, GTSB1), ;

LEFT_IN :: TI0[47:0] = I0[47:0] (FM_ADD, FM_SUB, FM_AND, FM_NAND, FM_OR, FM_NOR,
 FM_XOR, FM_XNOR, FM_LID, FM_NEQ, FM_GT, ), I1[47:0] (FM_RID, ), (I0[47:0] NOT )
 (FM_LNOT, ), (I1[47:0] NOT ) (FM_RNOT, ), ;
RIGHT_IN :: TI1[47:0] = I1[47:0] (FM_ADD, FM_SUB, FM_AND, FM_NAND, FM_OR, FM_NOR
, FM_XOR, FM_XNOR, FM_NEQ, FM_GT, ), ("1" REPEAT 48) (FM_LID, FM_RID, FM_LNOT, F
M_RNOT, FM_ONE, ), ;
MAIN_OUT :: SO0[47:0] = O0[47:0] (FM_ADD, FM_SUB, FM_AND, FM_NAND, FM_OR, FM_NOR
, FM_XOR, FM_XNOR, FM_LID, FM_RID, FM_LNOT, FM_RNOT, FM_ONE, ), ;
CARRY_IN :: TICIN[0:0] = ICIN[0:0] (FM_ADD, FM_SUB, ), "1" (FM_GT, ), ;
CARRY_OUT :: SOCOUT[0:0] = OCOUT[0:0] (FM_ADD, FM_SUB, ), ;
OVF :: SOVF[0:0] = OVF[0:0] (FM_ADD, FM_SUB, ), ;
COMP_OUT :: SO1[0:0] = (TO0[47:0] GOR ) (FM_NEQ, ), (TO0[47:47] NOT ) (FM_GT, ),
 ;



Cost Function :  Max_Delay
Algorithm Type :  1-Greedy
Time : 23.3 real         7.1 user         1.2 sys

NAME : CANONICAL;
TARGET COMPONENT : VDP5ALU001;
BIT_WIDTH : 48;
GATE_COUNT : 630;
MAX_DELAY :  72.10;

FUNCTION-RULE LIST : (FM_ADD, AA1), (FM_SUB, SS), (FM_AND, ADAD1), (FM_NAND, NDN
D1), (FM_OR, OROR1), (FM_NOR, NRNR1), (FM_XOR, XRXR1), (FM_XNOR, XNXN1), (FM_LID
, LDAD1), (FM_RID, RDAD1), (FM_LNOT, LNAD1), (FM_RNOT, RNAD1), (FM_ONE, OEOR2),
(FM_NEQ, NQXR1), (FM_GT, GTSB1), ;

LEFT_IN :: TI0[47:0] = I0[47:0] (FM_ADD, FM_SUB, FM_AND, FM_NAND, FM_OR, FM_NOR,
 FM_XOR, FM_XNOR, FM_LID, FM_NEQ, FM_GT, ), I1[47:0] (FM_RID, ), (I0[47:0] NOT )
```

```
 (FM_LNOT, ), (I1[47:0] NOT ) (FM_RNOT, ), ;
RIGHT_IN :: TI1[47:0] = I1[47:0] (FM_ADD, FM_SUB, FM_AND, FM_NAND, FM_OR, FM_NOR
, FM_XOR, FM_XNOR, FM_NEQ, FM_GT, ), ("1" REPEAT 48) (FM_LID, FM_RID, FM_LNOT, F
M_RNOT, FM_ONE, ), ;
MAIN_OUT :: SO0[47:0] = O0[47:0] (FM_ADD, FM_SUB, FM_AND, FM_NAND, FM_OR, FM_NOR
, FM_XOR, FM_XNOR, FM_LID, FM_RID, FM_LNOT, FM_RNOT, FM_ONE, ), ;
CARRY_IN :: TICIN[0:0] = ICIN[0:0] (FM_ADD, FM_SUB, ), "1" (FM_GT, ), ;
CARRY_OUT :: SOCOUT[0:0] = OCOUT[0:0] (FM_ADD, FM_SUB, ), ;
OVF :: SOVF[0:0] = OVF[0:0] (FM_ADD, FM_SUB, ), ;
COMP_OUT :: SO1[0:0] = (TO0[47:0] GOR ) (FM_NEQ, ), (TO0[47:47] NOT ) (FM_GT, ),
 ;


Cost Function :  Gate-Count
Algorithm :   Dynamic Programming (1)
Time :   62.0 real        21.6 user         1.6 sys


NAME : CANONICAL;
TARGET COMPONENT : VDP5ALU001;
BIT_WIDTH : 48;
GATE_COUNT : 726;
MAX_DELAY :  75.30;

FUNCTION-RULE LIST : (FM_ADD, AA1), (FM_SUB, SS), (FM_AND, ADAD1), (FM_NAND, NDN
D1), (FM_OR, OROR1), (FM_NOR, NRNR1), (FM_XOR, XRXR1), (FM_XNOR, XNXN1), (FM_LID
, LDAD1), (FM_RID, RDAD1), (FM_LNOT, LNAD1), (FM_RNOT, RNAD1), (FM_ONE, OEOR1),
(FM_NEQ, NQSB1), (FM_GT, GTSB1), ;

LEFT_IN :: TI0[47:0] = I0[47:0] (FM_ADD, FM_SUB, FM_AND, FM_NAND, FM_OR, FM_NOR,
 FM_XOR, FM_XNOR, FM_LID, FM_NEQ, FM_GT, ), I1[47:0] (FM_RID, ), (I0[47:0] NOT )
 (FM_LNOT, ), (I1[47:0] NOT ) (FM_RNOT, ), ("1" REPEAT 48) (FM_ONE, ), ;
RIGHT_IN :: TI1[47:0] = I1[47:0] (FM_ADD, FM_SUB, FM_AND, FM_NAND, FM_OR, FM_NOR
, FM_XOR, FM_XNOR, FM_NEQ, FM_GT, ), ("1" REPEAT 48) (FM_LID, FM_RID, FM_LNOT, F
M_RNOT, ), ;
MAIN_OUT :: SO0[47:0] = O0[47:0] (FM_ADD, FM_SUB, FM_AND, FM_NAND, FM_OR, FM_NOR
, FM_XOR, FM_XNOR, FM_LID, FM_RID, FM_LNOT, FM_RNOT, FM_ONE, ), ;
CARRY_IN :: TICIN[0:0] = ICIN[0:0] (FM_ADD, FM_SUB, ), "1" (FM_NEQ, FM_GT, ), ;
CARRY_OUT :: SOCOUT[0:0] = OCOUT[0:0] (FM_ADD, FM_SUB, ), ;
OVF :: SOVF[0:0] = OVF[0:0] (FM_ADD, FM_SUB, ), ;
COMP_OUT :: SO1[0:0] = (TO0[47:0] GOR ) (FM_NEQ, ), (TO0[47:47] NOT ) (FM_GT, ),
 ;

Cost Function :  Gate-Count
Algorithm :   Dynamic Programming (2)
Time : 289.7 real        75.9 user         4.7 sys

NAME : CANONICAL;
TARGET COMPONENT : VDP5ALU001;
BIT_WIDTH : 48;
GATE_COUNT : 630;
MAX_DELAY :  72.10;

FUNCTION-RULE LIST : (FM_ADD, AA1), (FM_SUB, SS), (FM_AND, ADAD1), (FM_NAND, NDN
```

```
D1), (FM_OR, OROR1), (FM_NOR, NRNR1), (FM_XOR, XRXR1), (FM_XNOR, XNXN1), (FM_LID
, LDAD1), (FM_RID, RDAD1), (FM_LNOT, LNAD1), (FM_RNOT, RNAD1), (FM_ONE, OEOR2),
(FM_NEQ, NQXR1), (FM_GT, GTSB1), ;

LEFT_IN :: TI0[47:0] = I0[47:0] (FM_ADD, FM_SUB, FM_AND, FM_NAND, FM_OR, FM_NOR,
 FM_XOR, FM_XNOR, FM_LID, FM_NEQ, FM_GT, ), I1[47:0] (FM_RID, ), (I0[47:0] NOT )
 (FM_LNOT, ), (I1[47:0] NOT ) (FM_RNOT, ), ;
RIGHT_IN :: TI1[47:0] = I1[47:0] (FM_ADD, FM_SUB, FM_AND, FM_NAND, FM_OR, FM_NOR
, FM_XOR, FM_XNOR, FM_NEQ, FM_GT, ), ("1" REPEAT 48) (FM_LID, FM_RID, FM_LNOT, F
M_RNOT, FM_ONE, ), ;
MAIN_OUT :: SO0[47:0] = O0[47:0] (FM_ADD, FM_SUB, FM_AND, FM_NAND, FM_OR, FM_NOR
, FM_XOR, FM_XNOR, FM_LID, FM_RID, FM_LNOT, FM_RNOT, FM_ONE, ), ;
CARRY_IN :: TICIN[0:0] = ICIN[0:0] (FM_ADD, FM_SUB, ), "1" (FM_GT, ), ;
CARRY_OUT :: SOCOUT[0:0] = OCOUT[0:0] (FM_ADD, FM_SUB, ), ;
OVF :: SOVF[0:0] = OVF[0:0] (FM_ADD, FM_SUB, ), ;
COMP_OUT :: SO1[0:0] = (TO0[47:0] GOR ) (FM_NEQ, ), (TO0[47:47] NOT ) (FM_GT, ),
 ;
Cost Function :  Gate-Count
Algorithm :   Dynamic Programming (3)
Time :   257.5 real       241.4 user         4.1 sys

NAME : CANONICAL;
TARGET COMPONENT : VDP5ALU001;
BIT_WIDTH : 48;
GATE_COUNT : 630;
MAX_DELAY :  72.10;

FUNCTION-RULE LIST : (FM_ADD, AA1), (FM_SUB, SS), (FM_AND, ADAD1), (FM_NAND, NDN
D1), (FM_OR, OROR1), (FM_NOR, NRNR1), (FM_XOR, XRXR1), (FM_XNOR, XNXN1), (FM_LID
, LDAD1), (FM_RID, RDAD2), (FM_LNOT, LNAD2), (FM_RNOT, RNAD2), (FM_ONE, OEOR1),
(FM_NEQ, NQSB1), (FM_GT, GTSB1), ;

LEFT_IN :: TI0[47:0] = I0[47:0] (FM_ADD, FM_SUB, FM_AND, FM_NAND, FM_OR, FM_NOR,
 FM_XOR, FM_XNOR, FM_LID, FM_NEQ, FM_GT, ), ("1" REPEAT 48) (FM_RID, FM_LNOT, FM
_RNOT, FM_ONE, ), ;
RIGHT_IN :: TI1[47:0] = I1[47:0] (FM_ADD, FM_SUB, FM_AND, FM_NAND, FM_OR, FM_NOR
, FM_XOR, FM_XNOR, FM_RID, FM_NEQ, FM_GT, ), ("1" REPEAT 48) (FM_LID, ), (I0[47:
0] NOT ) (FM_LNOT, ), (I1[47:0] NOT ) (FM_RNOT, ), ;
MAIN_OUT :: SO0[47:0] = O0[47:0] (FM_ADD, FM_SUB, FM_AND, FM_NAND, FM_OR, FM_NOR
, FM_XOR, FM_XNOR, FM_LID, FM_RID, FM_LNOT, FM_RNOT, FM_ONE, ), ;
CARRY_IN :: TICIN[0:0] = ICIN[0:0] (FM_ADD, FM_SUB, ), "1" (FM_NEQ, FM_GT, ), ;
CARRY_OUT :: SOCOUT[0:0] = OCOUT[0:0] (FM_ADD, FM_SUB, ), ;
OVF :: SOVF[0:0] = OVF[0:0] (FM_ADD, FM_SUB, ), ;
COMP_OUT :: SO1[0:0] = (TO0[47:0] GOR ) (FM_NEQ, ), (TO0[47:47] NOT ) (FM_GT, ),
 ;



Cost Function :  Gate-Count
Algorithm :   Dynamic Programming (4)
Time : 1676.6 real       640.8 user        26.3 sys

NAME : CANONICAL;
```

```
TARGET COMPONENT : VDP5ALU001;
BIT_WIDTH : 48;
GATE_COUNT : 630;
MAX_DELAY :  72.10;

FUNCTION-RULE LIST : (FM_ADD, AA1), (FM_SUB, SS), (FM_AND, ADAD1), (FM_NAND, NDN
D1), (FM_OR, OROR1), (FM_NOR, NRNR1), (FM_XOR, XRXR1), (FM_XNOR, XNXN1), (FM_LID
, LDAD1), (FM_RID, RDAD2), (FM_LNOT, LNAD2), (FM_RNOT, RNAD2), (FM_ONE, OEOR1),
(FM_NEQ, NQSB1), (FM_GT, GTSB1), ;

LEFT_IN :: TI0[47:0] = I0[47:0] (FM_ADD, FM_SUB, FM_AND, FM_NAND, FM_OR, FM_NOR,
 FM_XOR, FM_XNOR, FM_LID, FM_NEQ, FM_GT, ), ("1" REPEAT 48) (FM_RID, FM_LNOT, FM
_RNOT, FM_ONE, ), ;
RIGHT_IN :: TI1[47:0] = I1[47:0] (FM_ADD, FM_SUB, FM_AND, FM_NAND, FM_OR, FM_NOR
, FM_XOR, FM_XNOR, FM_RID, FM_NEQ, FM_GT, ), ("1" REPEAT 48) (FM_LID, ), (I0[47:
0] NOT ) (FM_LNOT, ), (I1[47:0] NOT ) (FM_RNOT, ), ;
MAIN_OUT :: SO0[47:0] = O0[47:0] (FM_ADD, FM_SUB, FM_AND, FM_NAND, FM_OR, FM_NOR
, FM_XOR, FM_XNOR, FM_LID, FM_RID, FM_LNOT, FM_RNOT, FM_ONE, ), ;
CARRY_IN :: TICIN[0:0] = ICIN[0:0] (FM_ADD, FM_SUB, ), "1" (FM_NEQ, FM_GT, ), ;
CARRY_OUT :: SOCOUT[0:0] = OCOUT[0:0] (FM_ADD, FM_SUB, ), ;
OVF :: SOVF[0:0] = OVF[0:0] (FM_ADD, FM_SUB, ), ;
COMP_OUT :: SO1[0:0] = (TO0[47:0] GOR ) (FM_NEQ, ), (TO0[47:47] NOT ) (FM_GT, ),
 ;


Cost Function :  Max_Delay
Algorithm :   Dynamic Programming (1)
Time :   23.9 real        21.1 user        0.7 sys
Memory :


NAME : CANONICAL;
TARGET COMPONENT : VDP5ALU001;
BIT_WIDTH : 48;
GATE_COUNT : 1206;
MAX_DELAY :  71.30;

FUNCTION-RULE LIST : (FM_ADD, AA1), (FM_SUB, SS), (FM_AND, ADAD1), (FM_NAND, NDN
D1), (FM_OR, OROR1), (FM_NOR, NRNR1), (FM_XOR, XRXR1), (FM_XNOR, XNXN1), (FM_LID
, LDAD1), (FM_RID, RDAD1), (FM_LNOT, LNNE), (FM_RNOT, RNNE), (FM_ONE, OEOR1), (F
M_NEQ, NQSB1), (FM_GT, GTSB1), ;

LEFT_IN :: TI0[47:0] = I0[47:0] (FM_ADD, FM_SUB, FM_AND, FM_NAND, FM_OR, FM_NOR,
 FM_XOR, FM_XNOR, FM_LID, FM_NEQ, FM_GT, ), I1[47:0] (FM_RID, ), ("1" REPEAT 48)
 (FM_ONE, ), ;
RIGHT_IN :: TI1[47:0] = I1[47:0] (FM_ADD, FM_SUB, FM_AND, FM_NAND, FM_OR, FM_NOR
, FM_XOR, FM_XNOR, FM_NEQ, FM_GT, ), ("1" REPEAT 48) (FM_LID, FM_RID, ), ;
LOGIC_LEFT_IN :: TLI0[47:0] = I0[47:0] (FM_LNOT, FM_RNOT, ), ;
LOGIC_RIGHT_IN :: TLI1[47:0] = I1[47:0] (FM_LNOT, FM_RNOT, ), ;
MAIN_OUT :: SO0[47:0] = O0[47:0] (FM_ADD, FM_SUB, FM_AND, FM_NAND, FM_OR, FM_NOR
, FM_XOR, FM_XNOR, FM_LID, FM_RID, FM_ONE, ), LO0[47:0] (FM_LNOT, FM_RNOT, ), ;
CARRY_IN :: TICIN[0:0] = ICIN[0:0] (FM_ADD, FM_SUB, ), "1" (FM_NEQ, FM_GT, ), ;
CARRY_OUT :: SOCOUT[0:0] = OCOUT[0:0] (FM_ADD, FM_SUB, ), ;
OVF :: SOVF[0:0] = OVF[0:0] (FM_ADD, FM_SUB, ), ;
```

```
COMP_OUT :: SO1[0:0] = (TO0[47:0] GOR ) (FM_NEQ, ), (TO0[47:47] NOT ) (FM_GT, ),
 ;
LOGIC_OUT :: LO[47:0] = 1,1,0,0 (FM_LNOT, ), 1,0,1,0 (FM_RNOT, ), ;



Cost Function :  Max_Delay
Algorithm :   Dynamic Programming (2)
Time : 74.7 real        72.5 user         1.1 sys


NAME : CANONICAL;
TARGET COMPONENT : VDP5ALU001;
BIT_WIDTH : 48;
GATE_COUNT : 1110;
MAX_DELAY :  68.10;

FUNCTION-RULE LIST : (FM_ADD, AA1), (FM_SUB, SS), (FM_AND, ADAD1), (FM_NAND, NDN
D1), (FM_OR, OROR1), (FM_NOR, NRNR1), (FM_XOR, XRXR1), (FM_XNOR, XNXN1), (FM_LID
, LDAD1), (FM_RID, RDAD2), (FM_LNOT, LNNE), (FM_RNOT, RNNE), (FM_ONE, OEOR1), (F
M_NEQ, NQSB1), (FM_GT, GTSB1), ;

LEFT_IN :: TI0[47:0] = I0[47:0] (FM_ADD, FM_SUB, FM_AND, FM_NAND, FM_OR, FM_NOR,
 FM_XOR, FM_XNOR, FM_LID, FM_NEQ, FM_GT, ), ("1" REPEAT 48) (FM_RID, FM_ONE, ),
;
RIGHT_IN :: TI1[47:0] = I1[47:0] (FM_ADD, FM_SUB, FM_AND, FM_NAND, FM_OR, FM_NOR
, FM_XOR, FM_XNOR, FM_RID, FM_NEQ, FM_GT, ), ("1" REPEAT 48) (FM_LID, ), ;
LOGIC_LEFT_IN :: TLI0[47:0] = I0[47:0] (FM_LNOT, FM_RNOT, ), ;
LOGIC_RIGHT_IN :: TLI1[47:0] = I1[47:0] (FM_LNOT, FM_RNOT, ), ;
MAIN_OUT :: SO0[47:0] = O0[47:0] (FM_ADD, FM_SUB, FM_AND, FM_NAND, FM_OR, FM_NOR
, FM_XOR, FM_XNOR, FM_LID, FM_RID, FM_ONE, ), LO0[47:0] (FM_LNOT, FM_RNOT, ), ;
CARRY_IN :: TICIN[0:0] = ICIN[0:0] (FM_ADD, FM_SUB, ), "1" (FM_NEQ, FM_GT, ), ;
CARRY_OUT :: SOCOUT[0:0] = OCOUT[0:0] (FM_ADD, FM_SUB, ), ;
OVF :: SOVF[0:0] = OVF[0:0] (FM_ADD, FM_SUB, ), ;
COMP_OUT :: SO1[0:0] = (TO0[47:0] GOR ) (FM_NEQ, ), (TO0[47:47] NOT ) (FM_GT, ),
 ;
LOGIC_OUT :: LO[47:0] = 1,1,0,0 (FM_LNOT, ), 1,0,1,0 (FM_RNOT, ), ;


Cost Function :  Max_Delay
Algorithm :   Dynamic Programming (3)
Time :   264.0 real       255.4 user         2.8 sys


NAME : CANONICAL;
TARGET COMPONENT : VDP5ALU001;
BIT_WIDTH : 48;
GATE_COUNT : 1110;
MAX_DELAY :  68.10;

FUNCTION-RULE LIST : (FM_ADD, AA1), (FM_SUB, SS), (FM_AND, ADAD1), (FM_NAND, NDN
D1), (FM_OR, OROR1), (FM_NOR, NRNR1), (FM_XOR, XRXR1), (FM_XNOR, XNXN1), (FM_LID
, LDAD1), (FM_RID, RDAD2), (FM_LNOT, LNNE), (FM_RNOT, RNNE), (FM_ONE, OEOR1), (F
M_NEQ, NQSB1), (FM_GT, GTSB1), ;
```

```
LEFT_IN :: TI0[47:0] = I0[47:0] (FM_ADD, FM_SUB, FM_AND, FM_NAND, FM_OR, FM_NOR,
 FM_XOR, FM_XNOR, FM_LID, FM_NEQ, FM_GT, ), ("1" REPEAT 48) (FM_RID, FM_ONE, ),
;
RIGHT_IN :: TI1[47:0] = I1[47:0] (FM_ADD, FM_SUB, FM_AND, FM_NAND, FM_OR, FM_NOR
, FM_XOR, FM_XNOR, FM_RID, FM_NEQ, FM_GT, ), ("1" REPEAT 48) (FM_LID, ), ;
LOGIC_LEFT_IN :: TLI0[47:0] = I0[47:0] (FM_LNOT, FM_RNOT, ), ;
LOGIC_RIGHT_IN :: TLI1[47:0] = I1[47:0] (FM_LNOT, FM_RNOT, ), ;
MAIN_OUT :: SO0[47:0] = O0[47:0] (FM_ADD, FM_SUB, FM_AND, FM_NAND, FM_OR, FM_NOR
, FM_XOR, FM_XNOR, FM_LID, FM_RID, FM_ONE, ), LO0[47:0] (FM_LNOT, FM_RNOT, ), ;
CARRY_IN :: TICIN[0:0] = ICIN[0:0] (FM_ADD, FM_SUB, ), "1" (FM_NEQ, FM_GT, ), ;
CARRY_OUT :: SOCOUT[0:0] = OCOUT[0:0] (FM_ADD, FM_SUB, ), ;
OVF :: SOVF[0:0] = OVF[0:0] (FM_ADD, FM_SUB, ), ;
COMP_OUT :: SO1[0:0] = (TO0[47:0] GOR ) (FM_NEQ, ), (TO0[47:47] NOT ) (FM_GT, ),
 ;
LOGIC_OUT :: LO[47:0] = 1,1,0,0 (FM_LNOT, ), 1,0,1,0 (FM_RNOT, ), ;


Cost Function :  Max_Delay
Algorithm :   Dynamic Programming (4)
Time :   621.0 real       594.1 user         9.2 sys


NAME : CANONICAL;
TARGET COMPONENT : VDP5ALU001;
BIT_WIDTH : 48;
GATE_COUNT : 1110;
MAX_DELAY :  68.10;

FUNCTION-RULE LIST : (FM_ADD, AA1), (FM_SUB, SS), (FM_AND, ADAD1), (FM_NAND, NDN
D1), (FM_OR, OROR1), (FM_NOR, NRNR1), (FM_XOR, XRXR1), (FM_XNOR, XNXN1), (FM_LID
, LDAD1), (FM_RID, RDAD2), (FM_LNOT, LNNE), (FM_RNOT, RNNE), (FM_ONE, OEOR1), (F
M_NEQ, NQSB1), (FM_GT, GTSB1), ;

LEFT_IN :: TI0[47:0] = I0[47:0] (FM_ADD, FM_SUB, FM_AND, FM_NAND, FM_OR, FM_NOR,
 FM_XOR, FM_XNOR, FM_LID, FM_NEQ, FM_GT, ), ("1" REPEAT 48) (FM_RID, FM_ONE, ),
;
RIGHT_IN :: TI1[47:0] = I1[47:0] (FM_ADD, FM_SUB, FM_AND, FM_NAND, FM_OR, FM_NOR
, FM_XOR, FM_XNOR, FM_RID, FM_NEQ, FM_GT, ), ("1" REPEAT 48) (FM_LID, ), ;
LOGIC_LEFT_IN :: TLI0[47:0] = I0[47:0] (FM_LNOT, FM_RNOT, ), ;
LOGIC_RIGHT_IN :: TLI1[47:0] = I1[47:0] (FM_LNOT, FM_RNOT, ), ;
MAIN_OUT :: SO0[47:0] = O0[47:0] (FM_ADD, FM_SUB, FM_AND, FM_NAND, FM_OR, FM_NOR
, FM_XOR, FM_XNOR, FM_LID, FM_RID, FM_ONE, ), LO0[47:0] (FM_LNOT, FM_RNOT, ), ;
CARRY_IN :: TICIN[0:0] = ICIN[0:0] (FM_ADD, FM_SUB, ), "1" (FM_NEQ, FM_GT, ), ;
CARRY_OUT :: SOCOUT[0:0] = OCOUT[0:0] (FM_ADD, FM_SUB, ), ;
CARRY_OUT :: SOCOUT[0:0] = OCOUT[0:0] (FM_ADD, FM_SUB, ), ;
OVF :: SOVF[0:0] = OVF[0:0] (FM_ADD, FM_SUB, ), ;
COMP_OUT :: SO1[0:0] = (TO0[47:0] GOR ) (FM_NEQ, ), (TO0[47:47] NOT ) (FM_GT, ),
 ;
LOGIC_OUT :: LO[47:0] = 1,1,0,0 (FM_LNOT, ), 1,0,1,0 (FM_RNOT, ), ;
```

B.4 Example 4

```
Source component :


NAME : ALU1;
BIT_WIDTH : 16;
GATE_COUNT : 8;
MAX_DELAY : 8;

FM_ADD ::
LEFT_IN : I0 = R;
RIGHT_IN : I1 = S;
CARRY_IN : ICIN[0:0] = Cn[0:0];
MAIN_OUT : F = O0;
CARRY_OUT : Cn4[0:0] = OCOUT[0:0];
OVF_OUT : OVR[0:0] = OVF[0:0];

FM_SUB ::
LEFT_IN : I0 = R;
RIGHT_IN : I1 = S;
CARRY_IN : ICIN[0:0] = Cn[0:0];
MAIN_OUT : F = O0;
CARRY_OUT : Cn4[0:0] = OCOUT[0:0];
OVF_OUT : OVR[0:0] = OVF[0:0];

FM_RSUB ::
LEFT_IN : I0 = R;
RIGHT_IN : I1 = S;
CARRY_IN : ICIN[0:0] = Cn[0:0];
MAIN_OUT : F = O0;
CARRY_OUT : Cn4[0:0] = OCOUT[0:0];
OVF_OUT : OVR[0:0] = OVF[0:0];

FM_OR ::
LEFT_IN : I0 = R;
RIGHT_IN : I1 = S;
MAIN_OUT : F = O0;
FM_AND ::
LEFT_IN : I0 = R;
RIGHT_IN : I1 = S;
MAIN_OUT : F = O0;
FM_LINHI ::
LEFT_IN : I0 = R;
RIGHT_IN : I1 = S;
MAIN_OUT : F = O0;
FM_XOR ::
LEFT_IN : I0 = R;
RIGHT_IN : I1 = S;
MAIN_OUT : F = O0;
FM_XNOR ::
LEFT_IN : I0 = R;
RIGHT_IN : I1 = S;
MAIN_OUT : F = O0;
```

```
Target component :


NAME : VDP1ASB001;
BIT_WIDTH : 16;
GATE_COUNT : 224;
MAX_DELAY : 19.8;

FM_ADD ::
LEFT_IN : I0 = A;
RIGHT_IN : I1 = B;
CARRY_IN : ICIN[0:0] = INST_CIN[0:0];
MAIN_OUT : SO = O0;
CARRY_OUT : INST_COUT[0:0] = OCOUT[0:0];
OVF_OUT : INST_OVR[0:0] = OVF[0:0];

FM_SUB ::
LEFT_IN : I0 = A;
RIGHT_IN : I1 = B;
CARRY_IN : ICIN[0:0] = INST_CIN[0:0];
MAIN_OUT : SO = O0;
CARRY_OUT : INST_COUT[0:0] = OCOUT[0:0];
OVF_OUT : INST_OVR[0:0] = OVF[0:0];

Solution is :


Cost Function :  Gate_Count
Algorithm Type :  1-greedy
Time : 4.9 real         4.1 user         0.3 sys

NAME : CANONICAL;
TARGET COMPONENT : VDP1ASB001;
BIT_WIDTH : 16;
GATE_COUNT : 432;
MAX_DELAY :  33.20;

FUNCTION-RULE LIST : (FM_ADD, AA1), (FM_SUB, SS), (FM_RSUB, RA1),
(FM_OR, NRNE), (FM_AND, ADNE), (FM_LINHI, LINE), (FM_XOR, XRNE),
(FM_XNOR, XNNE), ;

LEFT_IN :: TI0[15:0] = R[15:0] (FM_ADD, FM_SUB, ), (R[15:0] NOT ) (FM_RSUB, ), ;
RIGHT_IN :: TI1[15:0] = S[15:0] (FM_ADD, FM_SUB, FM_RSUB, ), ;
LOGIC_LEFT_IN :: TLI0[15:0] = R[15:0] (FM_OR, FM_AND, FM_LINHI, FM_XOR, FM_XNOR, ), ;
LOGIC_RIGHT_IN :: TLI1[15:0] = S[15:0] (FM_OR, FM_AND, FM_LINHI, FM_XOR, FM_XNOR, ), ;
MAIN_OUT :: SO0[15:0] = O0[15:0] (FM_ADD, FM_SUB, FM_RSUB, ),
LO0[15:0] (FM_OR, FM_AND, FM_LINHI, FM_XOR, FM_XNOR, ), ;
CARRY_IN :: TICIN[0:0] = Cn[0:0] (FM_ADD, FM_SUB, FM_RSUB, ), ;
CARRY_OUT :: SOCOUT[0:0] = OCOUT[0:0] (FM_ADD, FM_SUB, FM_RSUB, ), ;
OVF :: SOVF[0:0] = OVF[0:0] (FM_ADD, FM_SUB, FM_RSUB, ), ;
```

```
LOGIC_OUT :: LO[15:0] = 0,1,1,1 (FM_OR, ), 0,0,0,1 (FM_AND, ),
0,1,0,0 (FM_LINHI, ), 0,1,1,0 (FM_XOR, ), 1,0,0,1 (FM_XNOR, ), ;


Cost Function :  Max_Delay
Algorithm Type :  1-Greedy
Time : 4.6 real         4.1 user         0.3 sys

NAME : CANONICAL;
TARGET COMPONENT : VDP1ASB001;
BIT_WIDTH : 16;
GATE_COUNT : 464;
MAX_DELAY :  32.40;

FUNCTION-RULE LIST : (FM_ADD, AA1), (FM_SUB, SS), (FM_RSUB, RS), (FM_OR, NRNE),
 (FM_AND, ADNE), (FM_LINHI, LINE), (FM_XOR, XRNE), (FM_XNOR, XNNE), ;

LEFT_IN :: TI0[15:0] = R[15:0] (FM_ADD, FM_SUB, ), S[15:0] (FM_RSUB, ), ;
RIGHT_IN :: TI1[15:0] = S[15:0] (FM_ADD, FM_SUB, ), R[15:0] (FM_RSUB, ), ;
LOGIC_LEFT_IN :: TLI0[15:0] = R[15:0] (FM_OR, FM_AND, FM_LINHI, FM_XOR, FM_XNOR, ), ;
LOGIC_RIGHT_IN :: TLI1[15:0] = S[15:0] (FM_OR, FM_AND, FM_LINHI, FM_XOR, FM_XNOR, ), ;
MAIN_OUT :: SO0[15:0] = O0[15:0] (FM_ADD, FM_SUB, FM_RSUB, ),
LO0[15:0] (FM_OR, FM_AND, FM_LINHI, FM_XOR, FM_XNOR, ), ;
CARRY_IN :: TICIN[0:0] = Cn[0:0] (FM_ADD, FM_SUB, FM_RSUB, ), ;
CARRY_OUT :: SOCOUT[0:0] = OCOUT[0:0] (FM_ADD, FM_SUB, FM_RSUB, ), ;
OVF :: SOVF[0:0] = OVF[0:0] (FM_ADD, FM_SUB, FM_RSUB, ), ;
LOGIC_OUT :: LO[15:0] = 0,1,1,1 (FM_OR, ), 0,0,0,1 (FM_AND, ),
0,1,0,0 (FM_LINHI, ), 0,1,1,0 (FM_XOR, ), 1,0,0,1 (FM_XNOR, ), ;


Cost Function :  Gate-Count
Algorithm :   Dynamic Programming (1)
Time :   7.2 real         6.2 user         0.3 sys

NAME : CANONICAL;
TARGET COMPONENT : VDP1ASB001;
BIT_WIDTH : 16;
GATE_COUNT : 464;
MAX_DELAY :  32.40;

FUNCTION-RULE LIST : (FM_ADD, AA1), (FM_SUB, SS), (FM_RSUB, RS), (FM_OR, NRNE),

(FM_AND, ADNE), (FM_LINHI, LINE), (FM_XOR, XRNE), (FM_XNOR, XNNE), ;

LEFT_IN :: TI0[15:0] = R[15:0] (FM_ADD, FM_SUB, ), S[15:0] (FM_RSUB, ), ;
RIGHT_IN :: TI1[15:0] = S[15:0] (FM_ADD, FM_SUB, ), R[15:0] (FM_RSUB, ), ;
LOGIC_LEFT_IN :: TLI0[15:0] = R[15:0] (FM_OR, FM_AND, FM_LINHI, FM_XOR, FM_XNOR,
 ), ;
LOGIC_RIGHT_IN :: TLI1[15:0] = S[15:0] (FM_OR, FM_AND, FM_LINHI, FM_XOR, FM_XNOR
, ), ;
MAIN_OUT :: SO0[15:0] = O0[15:0] (FM_ADD, FM_SUB, FM_RSUB, ),
LO0[15:0] (FM_OR, FM_AND, FM_LINHI, FM_XOR, FM_XNOR, ), ;
CARRY_IN :: TICIN[0:0] = Cn[0:0] (FM_ADD, FM_SUB, FM_RSUB, ), ;
CARRY_OUT :: SOCOUT[0:0] = OCOUT[0:0] (FM_ADD, FM_SUB, FM_RSUB, ), ;
```

```
OVF :: SOVF[0:0] = OVF[0:0] (FM_ADD, FM_SUB, FM_RSUB, ), ;
LOGIC_OUT :: LO[15:0] = 0,1,1,1 (FM_OR, ), 0,0,0,1 (FM_AND, ),
0,1,0,0 (FM_LINHI , ), 0,1,1,0 (FM_XOR, ), 1,0,0,1 (FM_XNOR, ), ;


Cost Function :  Gate-Count
Algorithm :   Dynamic Programming (2)
Time :   11.5 real        10.7 user         0.4 sys

NAME : CANONICAL;
TARGET COMPONENT : VDP1ASB001;
BIT_WIDTH : 16;
GATE_COUNT : 432;
MAX_DELAY :  33.20;

FUNCTION-RULE LIST : (FM_ADD, AA1), (FM_SUB, SS), (FM_RSUB, RA1), (FM_OR, NRNE),
(FM_AND, ADNE), (FM_LINHI, LINE), (FM_XOR, XRNE), (FM_XNOR, XNNE), ;

LEFT_IN :: TI0[15:0] = R[15:0] (FM_ADD, FM_SUB, ), (R[15:0] NOT ) (FM_RSUB, ), ;
RIGHT_IN :: TI1[15:0] = S[15:0] (FM_ADD, FM_SUB, FM_RSUB, ), ;
LOGIC_LEFT_IN :: TLI0[15:0] = R[15:0] (FM_OR, FM_AND, FM_LINHI, FM_XOR, FM_XNOR, ), ;
LOGIC_RIGHT_IN :: TLI1[15:0] = S[15:0] (FM_OR, FM_AND, FM_LINHI, FM_XOR, FM_XNOR, ), ;
MAIN_OUT :: SO0[15:0] = O0[15:0] (FM_ADD, FM_SUB, FM_RSUB, ),
LO0[15:0] (FM_OR, FM_AND, FM_LINHI, FM_XOR, FM_XNOR, ), ;
CARRY_IN :: TICIN[0:0] = Cn[0:0] (FM_ADD, FM_SUB, FM_RSUB, ), ;
CARRY_OUT :: SOCOUT[0:0] = OCOUT[0:0] (FM_ADD, FM_SUB, FM_RSUB, ), ;
OVF :: SOVF[0:0] = OVF[0:0] (FM_ADD, FM_SUB, FM_RSUB, ), ;
LOGIC_OUT :: LO[15:0] = 0,1,1,1 (FM_OR, ), 0,0,0,1 (FM_AND, ),
0,1,0,0 (FM_LINHI, ), 0,1,1,0 (FM_XOR, ), 1,0,0,1 (FM_XNOR, ), ;


Cost Function :  Gate-Count
Algorithm :   Dynamic Programming (4)
Time :   22.2 real        20.6 user         0.8 sys

NAME : CANONICAL;
TARGET COMPONENT : VDP1ASB001;
BIT_WIDTH : 16;
GATE_COUNT : 432;
MAX_DELAY :  33.20;

FUNCTION-RULE LIST : (FM_ADD, AA2), (FM_SUB, SA2), (FM_RSUB, RS), (FM_OR, NRNE),
(FM_AND, ADNE), (FM_LINHI, LINE), (FM_XOR, XRNE), (FM_XNOR, XNNE), ;

LEFT_IN :: TI0[15:0] = S[15:0] (FM_ADD, FM_RSUB, ), (S[15:0] NOT ) (FM_SUB, ), ;
RIGHT_IN :: TI1[15:0] = R[15:0] (FM_ADD, FM_SUB, FM_RSUB, ), ;
LOGIC_LEFT_IN :: TLI0[15:0] = R[15:0] (FM_OR, FM_AND, FM_LINHI, FM_XOR, FM_XNOR, ), ;
LOGIC_RIGHT_IN :: TLI1[15:0] = S[15:0] (FM_OR, FM_AND, FM_LINHI, FM_XOR, FM_XNOR, ), ;
MAIN_OUT :: SO0[15:0] = O0[15:0] (FM_ADD, FM_SUB, FM_RSUB, ),
LO0[15:0] (FM_OR, FM_AND, FM_LINHI, FM_XOR, FM_XNOR, ), ;
CARRY_IN :: TICIN[0:0] = Cn[0:0] (FM_ADD, FM_SUB, FM_RSUB, ), ;
CARRY_OUT :: SOCOUT[0:0] = OCOUT[0:0] (FM_ADD, FM_SUB, FM_RSUB, ), ;
OVF :: SOVF[0:0] = OVF[0:0] (FM_ADD, FM_SUB, FM_RSUB, ), ;
LOGIC_OUT :: LO[15:0] = 0,1,1,1 (FM_OR, ), 0,0,0,1 (FM_AND, ),
```

```
0,1,0,0 (FM_LINHI, ), 0,1,1,0 (FM_XOR, ), 1,0,0,1 (FM_XNOR, ), ;

Cost Function :  Max_Delay
Algorithm :   Dynamic Programming (1)
Time :   6.9 real         6.1 user         0.3 sys

NAME : CANONICAL;
TARGET COMPONENT : VDP1ASB001;
BIT_WIDTH : 16;
GATE_COUNT : 464;
MAX_DELAY :  32.40;

FUNCTION-RULE LIST : (FM_ADD, AA1), (FM_SUB, SS), (FM_RSUB, RS), (FM_OR, NRNE),
(FM_AND, ADNE), (FM_LINHI, LINE), (FM_XOR, XRNE), (FM_XNOR, XNNE), ;

LEFT_IN :: TI0[15:0] = R[15:0] (FM_ADD, FM_SUB, ), S[15:0] (FM_RSUB, ), ;
RIGHT_IN :: TI1[15:0] = S[15:0] (FM_ADD, FM_SUB, ), R[15:0] (FM_RSUB, ), ;
LOGIC_LEFT_IN :: TLI0[15:0] = R[15:0] (FM_OR, FM_AND, FM_LINHI, FM_XOR, FM_XNOR,
 ), ;
LOGIC_RIGHT_IN :: TLI1[15:0] = S[15:0] (FM_OR, FM_AND, FM_LINHI, FM_XOR, FM_XNOR
, ), ;
MAIN_OUT :: SO0[15:0] = O0[15:0] (FM_ADD, FM_SUB, FM_RSUB, ),
LO0[15:0] (FM_OR, FM_AND, FM_LINHI, FM_XOR, FM_XNOR, ), ;
CARRY_IN :: TICIN[0:0] = Cn[0:0] (FM_ADD, FM_SUB, FM_RSUB, ), ;
CARRY_OUT :: SOCOUT[0:0] = OCOUT[0:0] (FM_ADD, FM_SUB, FM_RSUB, ), ;
OVF :: SOVF[0:0] = OVF[0:0] (FM_ADD, FM_SUB, FM_RSUB, ), ;
LOGIC_OUT :: LO[15:0] = 0,1,1,1 (FM_OR, ), 0,0,0,1 (FM_AND, ),
0,1,0,0 (FM_LINHI, ), 0,1,1,0 (FM_XOR, ), 1,0,0,1 (FM_XNOR, ), ;

Cost Function :  Max_Delay
Algorithm :   Dynamic Programming (2)
Time :   11.4 real        10.7 user        0.4 sys

NAME : CANONICAL;
TARGET COMPONENT : VDP1ASB001;
BIT_WIDTH : 16;
GATE_COUNT : 464;
MAX_DELAY :  32.40;

FUNCTION-RULE LIST : (FM_ADD, AA1), (FM_SUB, SS), (FM_RSUB, RS), (FM_OR, NRNE),
(FM_AND, ADNE), (FM_LINHI, LINE), (FM_XOR, XRNE), (FM_XNOR, XNNE), ;

LEFT_IN :: TI0[15:0] = R[15:0] (FM_ADD, FM_SUB, ), S[15:0] (FM_RSUB, ), ;
RIGHT_IN :: TI1[15:0] = S[15:0] (FM_ADD, FM_SUB, ), R[15:0] (FM_RSUB, ), ;
LOGIC_LEFT_IN :: TLI0[15:0] = R[15:0] (FM_OR, FM_AND, FM_LINHI, FM_XOR, FM_XNOR, ), ;
LOGIC_RIGHT_IN :: TLI1[15:0] = S[15:0] (FM_OR, FM_AND, FM_LINHI, FM_XOR, FM_XNOR, ), ;
MAIN_OUT :: SO0[15:0] = O0[15:0] (FM_ADD, FM_SUB, FM_RSUB, ),
LO0[15:0] (FM_OR, FM_AND, FM_LINHI, FM_XOR, FM_XNOR, ), ;
CARRY_IN :: TICIN[0:0] = Cn[0:0] (FM_ADD, FM_SUB, FM_RSUB, ), ;
CARRY_OUT :: SOCOUT[0:0] = OCOUT[0:0] (FM_ADD, FM_SUB, FM_RSUB, ), ;
OVF :: SOVF[0:0] = OVF[0:0] (FM_ADD, FM_SUB, FM_RSUB, ), ;
LOGIC_OUT :: LO[15:0] = 0,1,1,1 (FM_OR, ), 0,0,0,1 (FM_AND, ),
0,1,0,0 (FM_LINHI, ), 0,1,1,0 (FM_XOR, ), 1,0,0,1 (FM_XNOR, ), ;
```

```
Cost Function :  Max_Delay
Algorithm :   Dynamic Programming (4)
Time :   23.1 real        21.3 user         0.7 sys

NAME : CANONICAL;
TARGET COMPONENT : VDP1ASB001;
BIT_WIDTH : 16;
GATE_COUNT : 464;
MAX_DELAY :  32.40;

FUNCTION-RULE LIST : (FM_ADD, AA1), (FM_SUB, SS), (FM_RSUB, RS), (FM_OR, NRNE),
(FM_AND, ADNE), (FM_LINHI, LINE), (FM_XOR, XRNE), (FM_XNOR, XNNE), ;

LEFT_IN :: TI0[15:0] = R[15:0] (FM_ADD, FM_SUB, ), S[15:0] (FM_RSUB, ), ;
RIGHT_IN :: TI1[15:0] = S[15:0] (FM_ADD, FM_SUB, ), R[15:0] (FM_RSUB, ), ;
LOGIC_LEFT_IN :: TLI0[15:0] = R[15:0] (FM_OR, FM_AND, FM_LINHI, FM_XOR, FM_XNOR, ), ;
LOGIC_RIGHT_IN :: TLI1[15:0] = S[15:0] (FM_OR, FM_AND, FM_LINHI, FM_XOR, FM_XNOR, ), ;
MAIN_OUT :: SO0[15:0] = O0[15:0] (FM_ADD, FM_SUB, FM_RSUB, ),
LO0[15:0] (FM_OR, FM_AND, FM_LINHI, FM_XOR, FM_XNOR, ), ;
CARRY_IN :: TICIN[0:0] = Cn[0:0] (FM_ADD, FM_SUB, FM_RSUB, ), ;
CARRY_OUT :: SOCOUT[0:0] = OCOUT[0:0] (FM_ADD, FM_SUB, FM_RSUB, ), ;
OVF :: SOVF[0:0] = OVF[0:0] (FM_ADD, FM_SUB, FM_RSUB, ), ;
LOGIC_OUT :: LO[15:0] = 0,1,1,1 (FM_OR, ), 0,0,0,1 (FM_AND, ),
0,1,0,0 (FM_LINHI, ), 0,1,1,0 (FM_XOR, ), 1,0,0,1 (FM_XNOR, ), ;
```

B.5 Example 5

```
Source component :


NAME : GENUS_ALU;
BIT_WIDTH : 16;
GATE_COUNT : 912;
MAX_DELAY : 36.3;

FM_ADD ::
LEFT_IN : I0 = I0;
RIGHT_IN : I1 = I1;
CARRY_IN : ICIN[0:0] = ICIN[0:0];
MAIN_OUT : O0 = O0;
CARRY_OUT : OCOUT[0:0] = OCOUT[0:0];
OVF_OUT : OVF[0:0] = OVF[0:0];

FM_SUB ::
LEFT_IN : I0 = I0;
RIGHT_IN : I1 = I1;
CARRY_IN : ICIN[0:0] = ICIN[0:0];
MAIN_OUT : O0 = O0;
CARRY_OUT : OCOUT[0:0] = OCOUT[0:0];
OVF_OUT : OVF[0:0] = OVF[0:0];

FM_AND ::
```

```
LEFT_IN : I0 =  I0;
RIGHT_IN : I1 = I1;
MAIN_OUT : O0 = O0;

FM_NAND ::
LEFT_IN : I0 =  I0;
RIGHT_IN : I1 = I1;
MAIN_OUT : O0 = O0;

FM_OR ::
LEFT_IN : I0 =  I0;
RIGHT_IN : I1 = I1;
MAIN_OUT : O0 = O0;

FM_NOR ::
LEFT_IN : I0 =  I0;
RIGHT_IN : I1 = I1;
MAIN_OUT : O0 = O0;

FM_XOR ::
LEFT_IN : I0 =  I0;
RIGHT_IN : I1 = I1;
MAIN_OUT : O0 = O0;

FM_XNOR ::
LEFT_IN : I0 =  I0;
RIGHT_IN : I1 = I1;
MAIN_OUT : O0 = O0;

FM_LID ::
LEFT_IN : I0 =  I0;
RIGHT_IN : I1 = I1;
MAIN_OUT : O0 = O0;

FM_RID ::
LEFT_IN : I0 =  I0;
RIGHT_IN : I1 = I1;
MAIN_OUT : O0 = O0;

FM_LNOT ::
LEFT_IN : I0 =  I0;
RIGHT_IN : I1 = I1;
MAIN_OUT : O0 = O0;

FM_RNOT ::
LEFT_IN : I0 =  I0;
RIGHT_IN : I1 = I1;
MAIN_OUT : O0 = O0;

FM_ONE ::
MAIN_OUT : O0 = O0;

FM_NEQ ::
LEFT_IN : I0 = I0;
```

```
RIGHT_IN : I1 = I1;
COMP_OUT : O1 = O1;

FM_GT ::
LEFT_IN : I0 = I0;
RIGHT_IN : I1 = I1;
COMP_OUT : O1 = O1;

Target component :

NAME : ALU;
BIT_WIDTH : 16;
GATE_COUNT : 912;
MAX_DELAY : 13.2;

FM_ADD ::
LEFT_IN : I0 = A;
RIGHT_IN : I1 = B;
CARRY_IN : ICIN[0:0] = C0[0:0];
MAIN_OUT : F = O0;
CARRY_OUT : COUT[0:0] = OCOUT[0:0];
OVF_OUT : OVF[0:0] = OVF[0:0];

FM_SUB ::
LEFT_IN : I0 = A;
RIGHT_IN : I1 = B;
CARRY_IN : ICIN[0:0] = C0[0:0];
MAIN_OUT : F = O0;
CARRY_OUT : COUT[0:0] = OCOUT[0:0];
OVF_OUT : OVF[0:0] = OVF[0:0];

FM_RSUB ::
LEFT_IN : I0 = A;
RIGHT_IN : I1 = B;
CARRY_IN : ICIN[0:0] = C0[0:0];
MAIN_OUT : F = O0;
CARRY_OUT : COUT[0:0] = OCOUT[0:0];
OVF_OUT : OVF[0:0] = OVF[0:0];

FM_INC ::
LEFT_IN : I0 = A;
CARRY_IN : ICIN[0:0] = C0[0:0];
MAIN_OUT : F = O0;
CARRY_OUT : COUT[0:0] = OCOUT[0:0];
OVF_OUT : OVF[0:0] = OVF[0:0];

FM_DEC ::
LEFT_IN : I0 = A;
CARRY_IN : ICIN[0:0] = C0[0:0];
MAIN_OUT : F = O0;
CARRY_OUT : COUT[0:0] = OCOUT[0:0];
OVF_OUT : OVF[0:0] = OVF[0:0];

FM_ZERO ::
```

```
MAIN_OUT : SO = OO;

FM_ONE ::
MAIN_OUT : SO = OO;

FM_AND ::
LEFT_IN : IO =  A;
RIGHT_IN : I1 = B;
MAIN_OUT : SO = OO;

FM_NAND ::
LEFT_IN : IO =  A;
RIGHT_IN : I1 = B;
MAIN_OUT : SO = OO;

FM_OR ::
LEFT_IN : IO =  A;
RIGHT_IN : I1 = B;
MAIN_OUT : SO = OO;

FM_NOR ::
LEFT_IN : IO =  A;
RIGHT_IN : I1 = B;
MAIN_OUT : SO = OO;

FM_XOR ::
LEFT_IN : IO =  A;
RIGHT_IN : I1 = B;
MAIN_OUT : SO = OO;

FM_XNOR ::
LEFT_IN : IO =  A;
RIGHT_IN : I1 = B;
MAIN_OUT : SO = OO;

FM_RIMPL ::
LEFT_IN : IO =  A;
RIGHT_IN : I1 = B;
MAIN_OUT : SO = OO;

FM_LIMPL ::
LEFT_IN : IO =  A;
RIGHT_IN : I1 = B;
MAIN_OUT : SO = OO;

FM_LINHI ::
LEFT_IN : IO =  A;
RIGHT_IN : I1 = B;
MAIN_OUT : SO = OO;

FM_RINHI ::
LEFT_IN : IO =  A;
RIGHT_IN : I1 = B;
MAIN_OUT : SO = OO;
```

```
FM_LID ::
LEFT_IN : I0 =  A;
MAIN_OUT : SO = O0;

FM_RID ::
RIGHT_IN : I1 =  B;
MAIN_OUT : SO = O0;

FM_LINHI ::
LEFT_IN : I0 =  A;
MAIN_OUT : SO = O0;

FM_RINHI ::
RIGHT_IN : I1 = B;
MAIN_OUT : SO = O0;

Solution is :

Cost Function :  Gate_Count
Algorithm Type :  1-greedy
Time : 8.3 real         7.4 user         0.4 sys

NAME : CANONICAL;
TARGET COMPONENT : ALU;
BIT_WIDTH : 16;
GATE_COUNT : 134;
MAX_DELAY :  31.40;

FUNCTION-RULE LIST : (FM_ADD, AA1), (FM_SUB, SS), (FM_AND, ADAD1), (FM_NAND, NDND1),
(FM_OR, OROR1), (FM_NOR, NRNR1), (FM_XOR, XRXR1), (FM_XNOR, XNXN1), (FM_LID, LDLD),
(FM_RID, RDRD), (FM_LNOT, LNLD), (FM_RNOT, RNLD), (FM_ONE, OEOE), (FM_NEQ, NQXR1),
(FM_GT, GTSB1), ;

LEFT_IN :: TI0[15:0] = I0[15:0] (FM_ADD, FM_SUB, FM_AND, FM_NAND, FM_OR, FM_NOR,
 FM_XOR,
FM_XNOR, FM_LID, FM_NEQ, FM_GT, ), (I0[15:0] NOT ) (FM_LNOT, ), (I1[15:0] NOT ) (FM_RNOT, ), ;
RIGHT_IN :: TI1[15:0] = I1[15:0] (FM_ADD, FM_SUB, FM_AND, FM_NAND, FM_OR, FM_NOR
, FM_XOR, FM_XNOR, FM_RID, FM_NEQ, FM_GT, ), ;
MAIN_OUT :: SO0[15:0] = O0[15:0] (FM_ADD, FM_SUB, FM_AND, FM_NAND, FM_OR, FM_NOR
, FM_XOR, FM_XNOR, FM_LID, FM_RID, FM_LNOT, FM_RNOT, FM_ONE, ), ;
CARRY_IN :: TICIN[0:0] = ICIN[0:0] (FM_ADD, FM_SUB, ), "1" (FM_GT, ), ;
CARRY_OUT :: SOCOUT[0:0] = OCOUT[0:0] (FM_ADD, FM_SUB, ), ;
OVF :: SOVF[0:0] = OVF[0:0] (FM_ADD, FM_SUB, ), ;
COMP_OUT :: SO1[0:0] = (TO0[15:0] GOR ) (FM_NEQ, ), (TO0[15:15] NOT ) (FM_GT, ),;

Cost Function :  Max_Delay
Algorithm Type :  1-Greedy
Time : 9.7 real         7.6 user         0.5 sys

NAME : CANONICAL;
TARGET COMPONENT : ALU;
BIT_WIDTH : 16;
GATE_COUNT : 278;
```

```
MAX_DELAY :  28.20;

FUNCTION-RULE LIST : (FM_ADD, AA1), (FM_SUB, SS), (FM_AND, ADAD1), (FM_NAND, NDND1),
(FM_OR, OROR1), (FM_NOR, NRNR1), (FM_XOR, XRXR1), (FM_XNOR, XNXN1), (FM_LID, LDLD),
(FM_RID, RDRD), (FM_LNOT, LNNE), (FM_RNOT, RNNE), (FM_ONE, OEOE), (FM_NEQ, NQXR1), (FM_GT, GTSB1), ;

LEFT_IN :: TI0[15:0] = I0[15:0] (FM_ADD, FM_SUB, FM_AND, FM_NAND, FM_OR, FM_NOR, FM_XOR,
FM_XNOR, FM_LID, FM_NEQ, FM_GT, ), ;
RIGHT_IN :: TI1[15:0] = I1[15:0] (FM_ADD, FM_SUB, FM_AND, FM_NAND, FM_OR, FM_NOR, FM_XOR,
FM_XNOR, FM_RID, FM_NEQ, FM_GT, ), ;
LOGIC_LEFT_IN :: TLI0[15:0] = I0[15:0] (FM_LNOT, FM_RNOT, ), ;
LOGIC_RIGHT_IN :: TLI1[15:0] = I1[15:0] (FM_LNOT, FM_RNOT, ), ;
MAIN_OUT :: SO0[15:0] = O0[15:0] (FM_ADD, FM_SUB, FM_AND, FM_NAND, FM_OR, FM_NOR, FM_XOR,
FM_XNOR, FM_LID, FM_RID, FM_ONE, ), LO0[15:0] (FM_LNOT, FM_RNOT, ), ;
CARRY_IN :: TICIN[0:0] = ICIN[0:0] (FM_ADD, FM_SUB, ), "1" (FM_GT, ), ;
CARRY_OUT :: SOCOUT[0:0] = OCOUT[0:0] (FM_ADD, FM_SUB, ), ;
OVF :: SOVF[0:0] = OVF[0:0] (FM_ADD, FM_SUB, ), ;
COMP_OUT :: SO1[0:0] = (TO0[15:0] GOR ) (FM_NEQ, ), (TO0[15:15] NOT ) (FM_GT, ), ;
LOGIC_OUT :: LO[15:0] = 1,1,0,0 (FM_LNOT, ), 1,0,1,0 (FM_RNOT, ), ;

Cost Function :  Gate-Count
Algorithm :   Dynamic Programming (1)
Time : 21.5 real        20.4 user         0.5 sys

NAME : CANONICAL;
TARGET COMPONENT : ALU;
BIT_WIDTH : 16;
GATE_COUNT : 134;
MAX_DELAY :  31.40;

FUNCTION-RULE LIST : (FM_ADD, AA1), (FM_SUB, SS), (FM_AND, ADAD1), (FM_NAND, NDN
D1), (FM_OR, OROR1), (FM_NOR, NRNR1), (FM_XOR, XRXR1), (FM_XNOR, XNXN1), (FM_LID
, LDLD), (FM_RID, RDRD), (FM_LNOT, LNLD), (FM_RNOT, RNLD), (FM_ONE, OEOE), (FM_N
EQ, NQSB1), (FM_GT, GTSB1), ;

LEFT_IN :: TI0[15:0] = I0[15:0] (FM_ADD, FM_SUB, FM_AND, FM_NAND, FM_OR, FM_NOR, FM_XOR,
FM_XNOR, FM_LID, FM_NEQ, FM_GT, ), (I0[15:0] NOT ) (FM_LNOT, ), (I1[15:0] NOT ) (FM_RNOT, ), ;
RIGHT_IN :: TI1[15:0] = I1[15:0] (FM_ADD, FM_SUB, FM_AND, FM_NAND, FM_OR, FM_NOR
, FM_XOR, FM_XNOR, FM_RID, FM_NEQ, FM_GT, ), ;
MAIN_OUT :: SO0[15:0] = O0[15:0] (FM_ADD, FM_SUB, FM_AND, FM_NAND, FM_OR, FM_NOR
, FM_XOR, FM_XNOR, FM_LID, FM_RID, FM_LNOT, FM_RNOT, FM_ONE, ), ;
CARRY_IN :: TICIN[0:0] = ICIN[0:0] (FM_ADD, FM_SUB, ), "1" (FM_NEQ, FM_GT, ), ;
CARRY_OUT :: SOCOUT[0:0] = OCOUT[0:0] (FM_ADD, FM_SUB, ), ;
OVF :: SOVF[0:0] = OVF[0:0] (FM_ADD, FM_SUB, ), ;
COMP_OUT :: SO1[0:0] = (TO0[15:0] GOR ) (FM_NEQ, ), (TO0[15:15] NOT ) (FM_GT, ), ;

Cost Function :  Gate-Count
Algorithm :   Dynamic Programming (2)
Time :   64.7 real        62.7 user         0.9 sys

NAME : CANONICAL;
TARGET COMPONENT : ALU;
BIT_WIDTH : 16;
GATE_COUNT : 134;
```

```
MAX_DELAY :  31.40;

FUNCTION-RULE LIST : (FM_ADD, AA1), (FM_SUB, SS), (FM_AND, ADAD1), (FM_NAND, NDND1),
(FM_OR, OROR1), (FM_NOR, NRNR1), (FM_XOR, XRXR1), (FM_XNOR, XNXN1), (FM_LID, LDLD),
(FM_RID, RDRD), (FM_LNOT, LNLD), (FM_RNOT, RNLD), (FM_ONE, OEOE), (FM_NEQ, NQSB1),
(FM_GT, GTSB1), ;

LEFT_IN :: TI0[15:0] = I0[15:0] (FM_ADD, FM_SUB, FM_AND, FM_NAND, FM_OR,
FM_NOR, FM_XOR, FM_XNOR, FM_LID, FM_NEQ, FM_GT, ), (I0[15:0] NOT )
(FM_LNOT, ), (I1[15:0] NOT ) (FM_RNOT, ), ;
RIGHT_IN :: TI1[15:0] = I1[15:0] (FM_ADD, FM_SUB, FM_AND, FM_NAND, FM_OR,
FM_NOR , FM_XOR, FM_XNOR, FM_RID, FM_NEQ, FM_GT, ), ;
MAIN_OUT :: SO0[15:0] = O0[15:0] (FM_ADD, FM_SUB, FM_AND, FM_NAND, FM_OR, FM_NOR
, FM_XOR, FM_XNOR, FM_LID, FM_RID, FM_LNOT, FM_RNOT, FM_ONE, ), ;
CARRY_IN :: TICIN[0:0] = ICIN[0:0] (FM_ADD, FM_SUB, ), "1" (FM_NEQ, FM_GT, ), ;
CARRY_OUT :: SOCOUT[0:0] = OCOUT[0:0] (FM_ADD, FM_SUB, ), ;
OVF :: SOVF[0:0] = OVF[0:0] (FM_ADD, FM_SUB, ), ;
COMP_OUT :: SO1[0:0] = (TO0[15:0] GOR ) (FM_NEQ, ), (TO0[15:15] NOT ) (FM_GT, ), ;

Cost Function :  Gate-Count
Algorithm :   Dynamic Programming (4)
Time : 596.4 real       567.1 user         5.6 sys

NAME : CANONICAL;
TARGET COMPONENT : ALU;
BIT_WIDTH : 16;
GATE_COUNT : 134;
MAX_DELAY :  31.40;

FUNCTION-RULE LIST : (FM_ADD, AA1), (FM_SUB, SS), (FM_AND, ADAD1),
(FM_NAND, NDND1), (FM_OR, OROR1), (FM_NOR, NRNR1), (FM_XOR, XRXR1),
(FM_XNOR, XNXN1), (FM_LID, LDLD), (FM_RID, RDRD), (FM_LNOT, LNLD),
(FM_RNOT, RNLD), (FM_ONE, OEOE), (FM_NEQ, NQSB1), (FM_GT, GTSB1), ;

LEFT_IN :: TI0[15:0] = I0[15:0] (FM_ADD, FM_SUB, FM_AND, FM_NAND, FM_OR,
FM_NOR, FM_XOR, FM_XNOR, FM_LID, FM_NEQ, FM_GT, ), (I0[15:0] NOT )
(FM_LNOT, ), (I1[15:0] NOT ) (FM_RNOT, ), ;
RIGHT_IN :: TI1[15:0] = I1[15:0] (FM_ADD, FM_SUB, FM_AND, FM_NAND, FM_OR,
FM_NOR , FM_XOR, FM_XNOR, FM_RID, FM_NEQ, FM_GT, ), ;
MAIN_OUT :: SO0[15:0] = O0[15:0] (FM_ADD, FM_SUB, FM_AND, FM_NAND, FM_OR, FM_NOR , FM_XOR,
FM_XNOR, FM_LID, FM_RID, FM_LNOT, FM_RNOT, FM_ONE, ), ;
CARRY_IN :: TICIN[0:0] = ICIN[0:0] (FM_ADD, FM_SUB, ), "1" (FM_NEQ, FM_GT, ), ;
CARRY_OUT :: SOCOUT[0:0] = OCOUT[0:0] (FM_ADD, FM_SUB, ), ;
OVF :: SOVF[0:0] = OVF[0:0] (FM_ADD, FM_SUB, ), ;
COMP_OUT :: SO1[0:0] = (TO0[15:0] GOR ) (FM_NEQ, ), (TO0[15:15] NOT ) (FM_GT, ),
 ;

Cost Function :  Max_Delay
Algorithm :   Dynamic Programming (1)
Time :     25.2 real        21.0 user         0.7 sys

NAME : CANONICAL;
TARGET COMPONENT : ALU;
BIT_WIDTH : 16;
```

```
GATE_COUNT : 278;
MAX_DELAY :  28.20;

FUNCTION-RULE LIST : (FM_ADD, AA1), (FM_SUB, SS), (FM_AND, ADAD1),
(FM_NAND, NDN D1), (FM_OR, OROR1), (FM_NOR, NRNR1), (FM_XOR, XRXR1),
(FM_XNOR, XNXN1), (FM_LID, LDLD), (FM_RID, RDRD), (FM_LNOT, LNNE),
(FM_RNOT, RNNE), (FM_ONE, OEOE), (FM_NEQ, NQSB1), (FM_GT, GTSB1), ;

LEFT_IN :: TI0[15:0] = I0[15:0] (FM_ADD, FM_SUB, FM_AND, FM_NAND,
FM_OR, FM_NOR, FM_XOR, FM_XNOR, FM_LID, FM_NEQ, FM_GT, ), ;
RIGHT_IN :: TI1[15:0] = I1[15:0] (FM_ADD, FM_SUB, FM_AND, FM_NAND,
FM_OR, FM_NOR, FM_XOR, FM_XNOR, FM_RID, FM_NEQ, FM_GT, ), ;
LOGIC_LEFT_IN :: TLI0[15:0] = I0[15:0] (FM_LNOT, FM_RNOT, ), ;
LOGIC_RIGHT_IN :: TLI1[15:0] = I1[15:0] (FM_LNOT, FM_RNOT, ), ;
MAIN_OUT :: SO0[15:0] = O0[15:0] (FM_ADD, FM_SUB, FM_AND, FM_NAND, FM_OR, FM_NOR
, FM_XOR, FM_XNOR, FM_LID, FM_RID, FM_ONE, ), LO0[15:0] (FM_LNOT, FM_RNOT, ), ;
CARRY_IN :: TICIN[0:0] = ICIN[0:0] (FM_ADD, FM_SUB, ), "1" (FM_NEQ, FM_GT, ), ;
CARRY_OUT :: SOCOUT[0:0] = OCOUT[0:0] (FM_ADD, FM_SUB, ), ;
OVF :: SOVF[0:0] = OVF[0:0] (FM_ADD, FM_SUB, ), ;
COMP_OUT :: SO1[0:0] = (TO0[15:0] GOR ) (FM_NEQ, ), (TO0[15:15] NOT ) (FM_GT, ), ;
LOGIC_OUT :: LO[15:0] = 1,1,0,0 (FM_LNOT, ), 1,0,1,0 (FM_RNOT, ), ;

Cost Function :  Max_Delay
Algorithm :   Dynamic Programming (2)
Time :     65.6 real        63.5 user         1.1 sys

NAME : CANONICAL;
TARGET COMPONENT : ALU;
BIT_WIDTH : 16;
GATE_COUNT : 278;
MAX_DELAY :  28.20;

FUNCTION-RULE LIST : (FM_ADD, AA1), (FM_SUB, SS), (FM_AND, ADAD1),
(FM_NAND, NDND1), (FM_OR, OROR1), (FM_NOR, NRNR1), (FM_XOR, XRXR1),
(FM_XNOR, XNXN1), (FM_LID, LDLD), (FM_RID, RDRD), (FM_LNOT, LNNE),
(FM_RNOT, RNNE), (FM_ONE, OEOE), (FM_NEQ, NQSB1), (FM_GT, GTSB1), ;

LEFT_IN :: TI0[15:0] = I0[15:0] (FM_ADD, FM_SUB, FM_AND, FM_NAND,
FM_OR, FM_NOR, FM_XOR, FM_XNOR, FM_LID, FM_NEQ, FM_GT, ), ;
RIGHT_IN :: TI1[15:0] = I1[15:0] (FM_ADD, FM_SUB, FM_AND, FM_NAND,
FM_OR, FM_NOR, FM_XOR, FM_XNOR, FM_RID, FM_NEQ, FM_GT, ), ;
LOGIC_LEFT_IN :: TLI0[15:0] = I0[15:0] (FM_LNOT, FM_RNOT, ), ;
LOGIC_RIGHT_IN :: TLI1[15:0] = I1[15:0] (FM_LNOT, FM_RNOT, ), ;
MAIN_OUT :: SO0[15:0] = O0[15:0] (FM_ADD, FM_SUB, FM_AND, FM_NAND, FM_OR, FM_NOR
, FM_XOR, FM_XNOR, FM_LID, FM_RID, FM_ONE, ), LO0[15:0] (FM_LNOT, FM_RNOT, ), ;
CARRY_IN :: TICIN[0:0] = ICIN[0:0] (FM_ADD, FM_SUB, ), "1" (FM_NEQ, FM_GT, ), ;
CARRY_OUT :: SOCOUT[0:0] = OCOUT[0:0] (FM_ADD, FM_SUB, ), ;
OVF :: SOVF[0:0] = OVF[0:0] (FM_ADD, FM_SUB, ), ;
COMP_OUT :: SO1[0:0] = (TO0[15:0] GOR ) (FM_NEQ, ), (TO0[15:15] NOT ) (FM_GT, ),
 ;
LOGIC_OUT :: LO[15:0] = 1,1,0,0 (FM_LNOT, ), 1,0,1,0 (FM_RNOT, ), ;

Cost Function :  Max_Delay
Algorithm :   Dynamic Programming (4)
```

```
Time : 508.2 real       493.7 user        4.3 sys
Memory :

NAME : CANONICAL;
TARGET COMPONENT : ALU;
BIT_WIDTH : 16;
GATE_COUNT : 278;
MAX_DELAY :  28.20;

FUNCTION-RULE LIST : (FM_ADD, AA1), (FM_SUB, SS), (FM_AND, ADAD1),
(FM_NAND, NDND1), (FM_OR, OROR1), (FM_NOR, NRNR1), (FM_XOR, XRXR1),
(FM_XNOR, XNXN1), (FM_LID, LDLD), (FM_RID, RDRD), (FM_LNOT, LNNE),
(FM_RNOT, RNNE), (FM_ONE, OEOE), (FM_NEQ, NQSB1), (FM_GT, GTSB1), ;

LEFT_IN :: TI0[15:0] = I0[15:0] (FM_ADD, FM_SUB, FM_AND, FM_NAND, FM_OR, FM_NOR,
 FM_XOR, FM_XNOR, FM_LID, FM_NEQ, FM_GT, ), ;
RIGHT_IN :: TI1[15:0] = I1[15:0] (FM_ADD, FM_SUB, FM_AND, FM_NAND, FM_OR, FM_NOR
, FM_XOR, FM_XNOR, FM_RID, FM_NEQ, FM_GT, ), ;
LOGIC_LEFT_IN :: TLI0[15:0] = I0[15:0] (FM_LNOT, FM_RNOT, ), ;
LOGIC_RIGHT_IN :: TLI1[15:0] = I1[15:0] (FM_LNOT, FM_RNOT, ), ;
MAIN_OUT :: SO0[15:0] = O0[15:0] (FM_ADD, FM_SUB, FM_AND, FM_NAND, FM_OR, FM_NOR
, FM_XOR, FM_XNOR, FM_LID, FM_RID, FM_ONE, ), LO0[15:0] (FM_LNOT, FM_RNOT, ), ;
CARRY_IN :: TICIN[0:0] = ICIN[0:0] (FM_ADD, FM_SUB, ), "1" (FM_NEQ, FM_GT, ), ;
CARRY_OUT :: SOCOUT[0:0] = OCOUT[0:0] (FM_ADD, FM_SUB, ), ;
OVF :: SOVF[0:0] = OVF[0:0] (FM_ADD, FM_SUB, ), ;
OMP_OUT :: SO1[0:0] = (TO0[15:0] GOR ) (FM_NEQ, ), (TO0[15:15] NOT ) (FM_GT, ),
 ;
LOGIC_OUT :: LO[15:0] = 1,1,0,0 (FM_LNOT, ), 1,0,1,0 (FM_RNOT, ), ;
```

B.6 Example 6

```
Source component :

NAME : ALU;
BIT_WIDTH : 16;
GATE_COUNT : 912;
MAX_DELAY : 13.2;

FM_ADD (0) ::
LEFT_IN : I0 = A;
RIGHT_IN : I1 = B;
CARRY_IN : ICIN[0:0] = C0[0:0];
MAIN_OUT : F = O0;
CARRY_OUT : COUT[0:0] = OCOUT[0:0];
OVF_OUT : OVF[0:0] = OVF[0:0];

FM_ADD (1) ::
LEFT_IN : I0 = A;
RIGHT_IN : I1 = A OR (NOT B);
CARRY_IN : ICIN[0:0] = C0[0:0];
MAIN_OUT : F = O0;
CARRY_OUT : COUT[0:0] = OCOUT[0:0];
```

```
OVF_OUT : OVF[0:0] = OVF[0:0];

FM_ADD (2) ::
LEFT_IN : I0 = A AND B;
RIGHT_IN : I1 = A OR B;
CARRY_IN : ICIN[0:0] = C0[0:0];
MAIN_OUT : F = O0;
CARRY_OUT : COUT[0:0] = OCOUT[0:0];
OVF_OUT : OVF[0:0] = OVF[0:0];

FM_ADD (3) ::
LEFT_IN : I0 = A;
RIGHT_IN : I1 = A OR B;
CARRY_IN : ICIN[0:0] = C0[0:0];
MAIN_OUT : F = O0;
CARRY_OUT : COUT[0:0] = OCOUT[0:0];
OVF_OUT : OVF[0:0] = OVF[0:0];

FM_ADD (4) ::
LEFT_IN : I0 = A AND (NOT B);
RIGHT_IN : I1 = A OR B;
CARRY_IN : ICIN[0:0] = C0[0:0];
MAIN_OUT : F = O0;
CARRY_OUT : COUT[0:0] = OCOUT[0:0];
OVF_OUT : OVF[0:0] = OVF[0:0];

FM_ADD (5) ::
LEFT_IN : I0 = A;
RIGHT_IN : I1 = A;
CARRY_IN : ICIN[0:0] = C0[0:0];
MAIN_OUT : F = O0;
CARRY_OUT : COUT[0:0] = OCOUT[0:0];
OVF_OUT : OVF[0:0] = OVF[0:0];

FM_ADD (6) ::
LEFT_IN : I0 = A AND B;
RIGHT_IN : I1 = A;
CARRY_IN : ICIN[0:0] = C0[0:0];
MAIN_OUT : F = O0;
CARRY_OUT : COUT[0:0] = OCOUT[0:0];
OVF_OUT : OVF[0:0] = OVF[0:0];

FM_ADD (7) ::
LEFT_IN : I0 = A AND (NOT B);
RIGHT_IN : I1 = A;
CARRY_IN : ICIN[0:0] = C0[0:0];
MAIN_OUT : F = O0;
CARRY_OUT : COUT[0:0] = OCOUT[0:0];
OVF_OUT : OVF[0:0] = OVF[0:0];

FM_ADD (8) ::
LEFT_IN : I0 = REPEAT 16 "1";
RIGHT_IN : I1 = REPEAT 16 "0";
CARRY_IN : ICIN[0:0] = C0[0:0];
```

```
MAIN_OUT : F = O0;
CARRY_OUT : COUT[0:0] = OCOUT[0:0];
OVF_OUT : OVF[0:0] = OVF[0:0];

FM_SUB ::
LEFT_IN : I0 = A;
RIGHT_IN : I1 = B;
CARRY_IN : ICIN[0:0] = C0[0:0];
MAIN_OUT : F = O0;
CARRY_OUT : COUT[0:0] = OCOUT[0:0];
OVF_OUT : OVF[0:0] = OVF[0:0];

FM_INC (0)::
LEFT_IN : I0 = A;
CARRY_IN : ICIN[0:0] = C0[0:0];
MAIN_OUT : F = O0;
CARRY_OUT : COUT[0:0] = OCOUT[0:0];
OVF_OUT : OVF[0:0] = OVF[0:0];

FM_INC (1) ::
LEFT_IN : I0 = A OR (NOT B);
CARRY_IN : ICIN[0:0] = C0[0:0];
MAIN_OUT : F = O0;
CARRY_OUT : COUT[0:0] = OCOUT[0:0];
OVF_OUT : OVF[0:0] = OVF[0:0];

FM_INC (2) ::
LEFT_IN : I0 = A OR B;
CARRY_IN : ICIN[0:0] = C0[0:0];
MAIN_OUT : F = O0;
CARRY_OUT : COUT[0:0] = OCOUT[0:0];
OVF_OUT : OVF[0:0] = OVF[0:0];

FM_DEC (0) ::
LEFT_IN : I0 = A;
CARRY_IN : ICIN[0:0] = C0[0:0];
MAIN_OUT : F = O0;
CARRY_OUT : COUT[0:0] = OCOUT[0:0];
OVF_OUT : OVF[0:0] = OVF[0:0];

FM_DEC (1) ::
LEFT_IN : I0 = A AND B;
CARRY_IN : ICIN[0:0] = C0[0:0];
MAIN_OUT : F = O0;
CARRY_OUT : COUT[0:0] = OCOUT[0:0];
OVF_OUT : OVF[0:0] = OVF[0:0];

FM_DEC (2) ::
LEFT_IN : I0 = A AND (NOT B);
CARRY_IN : ICIN[0:0] = C0[0:0];
MAIN_OUT : F = O0;
CARRY_OUT : COUT[0:0] = OCOUT[0:0];
OVF_OUT : OVF[0:0] = OVF[0:0];
```

```
FM_ZERO ::
MAIN_OUT : SO = OO;

FM_ONE ::
MAIN_OUT : SO = OO;

FM_AND ::
LEFT_IN : IO =  A;
RIGHT_IN : I1 = B;
MAIN_OUT : SO = OO;

FM_NAND ::
LEFT_IN : IO =  A;
RIGHT_IN : I1 = B;
MAIN_OUT : SO = OO;

FM_OR ::
LEFT_IN : IO =  A;
RIGHT_IN : I1 = B;
MAIN_OUT : SO = OO;

FM_NOR ::
LEFT_IN : IO =  A;
RIGHT_IN : I1 = B;
MAIN_OUT : SO = OO;

FM_XOR ::
LEFT_IN : IO =  A;
RIGHT_IN : I1 = B;
MAIN_OUT : SO = OO;

FM_XNOR ::
LEFT_IN : IO =  A;
RIGHT_IN : I1 = B;
MAIN_OUT : SO = OO;

FM_RIMPL ::
LEFT_IN : IO =  A;
RIGHT_IN : I1 = B;
MAIN_OUT : SO = OO;

FM_LIMPL ::
LEFT_IN : IO =  A;
RIGHT_IN : I1 = B;
MAIN_OUT : SO = OO;

FM_LINHI ::
LEFT_IN : IO =  A;
RIGHT_IN : I1 = B;
MAIN_OUT : SO = OO;

FM_RINHI ::
LEFT_IN : IO =  A;
RIGHT_IN : I1 = B;
```

```
MAIN_OUT : SO = OO;

FM_LID ::
LEFT_IN : IO =  A;
MAIN_OUT : SO = OO;

FM_RID ::
RIGHT_IN : I1 =  B;
MAIN_OUT : SO = OO;

FM_LNOT ::
LEFT_IN : IO =  A;
MAIN_OUT : SO = OO;

FM_RNOT ::
RIGHT_IN : I1 = B;
MAIN_OUT : SO = OO;


Target component :


NAME : VDP5ALU001;
BIT_WIDTH : 16;
GATE_COUNT : 304;
MAX_DELAY : 19.5;

FM_ADD ::
LEFT_IN : IO = A;
RIGHT_IN : I1 = B;
CARRY_IN : ICIN[0:0] = INST_CIN[0:0];
MAIN_OUT : SO = OO;
CARRY_OUT : INST_COUT[0:0] = OCOUT[0:0];
OVF_OUT : INST_OVR[0:0] = OVF[0:0];

FM_SUB ::
LEFT_IN : IO = A;
RIGHT_IN : I1 = B;
CARRY_IN : ICIN[0:0] = INST_CIN[0:0];
MAIN_OUT : SO = OO;
CARRY_OUT : INST_COUT[0:0] = OCOUT[0:0];
OVF_OUT : INST_OVR[0:0] = OVF[0:0];

FM_RSUB ::
LEFT_IN : IO = A;
RIGHT_IN : I1 = B;
CARRY_IN : ICIN[0:0] = INST_CIN[0:0];
MAIN_OUT : SO = OO;
CARRY_OUT : INST_COUT[0:0] = OCOUT[0:0];
OVF_OUT : INST_OVR[0:0] = OVF[0:0];

FM_SUB ::
LEFT_IN : IO = NOT A;
RIGHT_IN : I1 = B;
```

```
CARRY_IN : ICIN[0:0] = INST_CIN[0:0];
MAIN_OUT : SO = OO;
CARRY_OUT : INST_COUT[0:0] = OCOUT[0:0];
OVF_OUT : INST_OVR[0:0] = OVF[0:0];


FM_AND ::
LEFT_IN : I0 =  A;
RIGHT_IN : I1 = B;
MAIN_OUT : SO = OO;

FM_NAND ::
LEFT_IN : I0 =  A;
RIGHT_IN : I1 = B;
MAIN_OUT : SO = OO;

FM_OR ::
LEFT_IN : I0 =  A;
RIGHT_IN : I1 = B;
MAIN_OUT : SO = OO;

FM_NOR ::
LEFT_IN : I0 =  A;
RIGHT_IN : I1 = B;
MAIN_OUT : SO = OO;

FM_XOR ::
LEFT_IN : I0 =  A;
RIGHT_IN : I1 = B;
MAIN_OUT : SO = OO;

FM_XNOR ::
LEFT_IN : I0 =  A;
RIGHT_IN : I1 = B;
MAIN_OUT : SO = OO;

FM_RIMPL ::
LEFT_IN : I0 =  A;
RIGHT_IN : I1 = B;
MAIN_OUT : SO = OO;

FM_LIMPL ::
LEFT_IN : I0 =  A;
RIGHT_IN : I1 = B;
MAIN_OUT : SO = OO;

FM_LINHI ::
LEFT_IN : I0 =  A;
RIGHT_IN : I1 = B;
MAIN_OUT : SO = OO;

FM_RINHI ::
LEFT_IN : I0 =  A;
RIGHT_IN : I1 = B;
```

```
MAIN_OUT : SO = OO;


Solution is :


Cost Function :  Gate_Count
Algorithm Type :  1-greedy
Time : 19.3 real        13.9 user         2.4 sys

NAME : CANONICAL;
TARGET COMPONENT : VDP5ALU001;
BIT_WIDTH : 16;
GATE_COUNT : 453;
MAX_DELAY :  36.20;

FUNCTION-RULE LIST : (FM_ADD(0), AA1), (FM_ADD(1), AA1), (FM_ADD(2), AA1), (FM_A
DD(3), AA1), (FM_ADD(4), AA1), (FM_ADD(5), AA1), (FM_ADD(6), AA1), (FM_ADD(7), A
A1), (FM_ADD(8), AA1), (FM_SUB, SS), (FM_INC(0), IA1), (FM_INC(1), IR1), (FM_INC
(2), IR1), (FM_DEC(0), DA2), (FM_DEC(1), DS1), (FM_DEC(2), DS1), (FM_ZERO, ZOAD2
), (FM_ONE, OEOR1), (FM_AND, ADAD1), (FM_NAND, NDND1), (FM_OR, OROR1), (FM_NOR,
NRNR1), (FM_XOR, XRXR1), (FM_XNOR, XNXN1), (FM_RIMPL, RLRL), (FM_LIMPL, LLLL), (
FM_LINHI, LILI), (FM_RINHI, RIRI), (FM_LID, LDAD2), (FM_RID, RDAD2), (FM_LNOT, L
NAD2), (FM_RNOT, RNAD2), ;

LEFT_IN :: TI0[15:0] = A[15:0] (FM_ADD(0), FM_ADD(1), FM_ADD(3), FM_ADD(5), FM_S
UB, FM_INC(0), FM_AND, FM_NAND, FM_OR, FM_NOR, FM_XOR, FM_XNOR, FM_RIMPL, FM_LIM
PL, FM_LINHI, FM_RINHI, ), (A[15:0] AND B[15:0]) (FM_ADD(2), FM_ADD(6), FM_DEC(1
), ), (A[15:0] AND (B[15:0] NOT )) (FM_ADD(4), FM_ADD(7), FM_DEC(2), ), ("1" REP
EAT 16) (FM_ADD(8), FM_INC(1), FM_INC(2), FM_DEC(0), FM_ONE, FM_LID, FM_RID, FM_
LNOT, FM_RNOT, ), ;
RIGHT_IN :: TI1[15:0] = B[15:0] (FM_ADD(0), FM_SUB, FM_AND, FM_NAND, FM_OR, FM_N
OR, FM_XOR, FM_XNOR, FM_RIMPL, FM_LIMPL, FM_LINHI, FM_RINHI, FM_RID, ), (A[15:0]
 OR (B[15:0] NOT )) (FM_ADD(1), FM_INC(1), ), (A[15:0] OR B[15:0]) (FM_ADD(2), F
M_ADD(3), FM_ADD(4), FM_INC(2), ), A[15:0] (FM_ADD(5), FM_ADD(6), FM_ADD(7), FM_
DEC(0), FM_LID, ), ("0" REPEAT 16) (FM_ADD(8), FM_INC(0), FM_DEC(1), FM_DEC(2),
FM_ZERO, ), (A[15:0] NOT ) (FM_LNOT, ), (B[15:0] NOT ) (FM_RNOT, ), ;
MAIN_OUT :: SO0[15:0] = O0[15:0] (FM_ADD(0), FM_ADD(1), FM_ADD(2), FM_ADD(3), FM
_ADD(4), FM_ADD(5), FM_ADD(6), FM_ADD(7), FM_ADD(8), FM_SUB, FM_INC(0), FM_INC(1
), FM_INC(2), FM_DEC(0), FM_DEC(1), FM_DEC(2), FM_ZERO, FM_ONE, FM_AND, FM_NAND,
 FM_OR, FM_NOR, FM_XOR, FM_XNOR, FM_RIMPL, FM_LIMPL, FM_LINHI, FM_RINHI, FM_LID,
 FM_RID, FM_LNOT, FM_RNOT, ), ;
CARRY_IN :: TICIN[0:0] = C0[0:0] (FM_ADD(0), FM_ADD(1), FM_ADD(2), FM_ADD(3), FM
_ADD(4), FM_ADD(5), FM_ADD(6), FM_ADD(7), FM_ADD(8), FM_SUB, ), "1" (FM_INC(0),
FM_INC(1), FM_INC(2), ), "0" (FM_DEC(0), FM_DEC(1), FM_DEC(2), ), ;
CARRY_OUT :: SOCOUT[0:0] = OCOUT[0:0] (FM_ADD(0), FM_ADD(1), FM_ADD(2), FM_ADD(3
), FM_ADD(4), FM_ADD(5), FM_ADD(6), FM_ADD(7), FM_ADD(8), FM_SUB, FM_INC(0), FM_
INC(1), FM_INC(2), FM_DEC(0), FM_DEC(1), FM_DEC(2), ), ;
OVF :: SOVF[0:0] = OVF[0:0] (FM_ADD(0), FM_ADD(1), FM_ADD(2), FM_ADD(3), FM_ADD(
4), FM_ADD(5), FM_ADD(6), FM_ADD(7), FM_ADD(8), FM_SUB, FM_INC(0), FM_INC(1), FM
_INC(2), FM_DEC(0), FM_DEC(1), FM_DEC(2), ), ;

Cost Function :  Max_Delay
Algorithm Type :  1-Greedy
```

```
Time : 17.0 real        13.7 user        2.0 sys

NAME : CANONICAL;
TARGET COMPONENT : VDP5ALU001;
BIT_WIDTH : 16;
GATE_COUNT : 453;
MAX_DELAY :  36.20;

FUNCTION-RULE LIST : (FM_ADD(0), AA1), (FM_ADD(1), AA1), (FM_ADD(2), AA1), (FM_A
DD(3), AA1), (FM_ADD(4), AA1), (FM_ADD(5), AA1), (FM_ADD(6), AA1), (FM_ADD(7), A
A1), (FM_ADD(8), AA1), (FM_SUB, SS), (FM_INC(0), IA1), (FM_INC(1), IR1), (FM_INC
(2), IR1), (FM_DEC(0), DA2), (FM_DEC(1), DS1), (FM_DEC(2), DS1), (FM_ZERO, ZOAD2
), (FM_ONE, OEOR1), (FM_AND, ADAD1), (FM_NAND, NDND1), (FM_OR, OROR1), (FM_NOR,
NRNR1), (FM_XOR, XRXR1), (FM_XNOR, XNXN1), (FM_RIMPL, RLRL), (FM_LIMPL, LLLL), (
FM_LINHI, LILI), (FM_RINHI, RIRI), (FM_LID, LDAD2), (FM_RID, RDAD2), (FM_LNOT, L
NAD2), (FM_RNOT, RNAD2), ;

LEFT_IN :: TI0[15:0] = A[15:0] (FM_ADD(0), FM_ADD(1), FM_ADD(3), FM_ADD(5), FM_S
UB, FM_INC(0), FM_AND, FM_NAND, FM_OR, FM_NOR, FM_XOR, FM_XNOR, FM_RIMPL, FM_LIM
PL, FM_LINHI, FM_RINHI, ), (A[15:0] AND B[15:0]) (FM_ADD(2), FM_ADD(6), FM_DEC(1
), ), (A[15:0] AND (B[15:0] NOT )) (FM_ADD(4), FM_ADD(7), FM_DEC(2), ), ("1" REP
EAT 16) (FM_ADD(8), FM_INC(1), FM_INC(2), FM_DEC(0), FM_ONE, FM_LID, FM_RID, FM_
LNOT, FM_RNOT, ), ;
RIGHT_IN :: TI1[15:0] = B[15:0] (FM_ADD(0), FM_SUB, FM_AND, FM_NAND, FM_OR, FM_N
OR, FM_XOR, FM_XNOR, FM_RIMPL, FM_LIMPL, FM_LINHI, FM_RINHI, FM_RID, ), (A[15:0]
 OR (B[15:0] NOT )) (FM_ADD(1), FM_INC(1), ), (A[15:0] OR B[15:0]) (FM_ADD(2), F
M_ADD(3), FM_ADD(4), FM_INC(2), ), A[15:0] (FM_ADD(5), FM_ADD(6), FM_ADD(7), FM_
DEC(0), FM_LID, ), ("0" REPEAT 16) (FM_ADD(8), FM_INC(0), FM_DEC(1), FM_DEC(2),
FM_ZERO, ), (A[15:0] NOT ) (FM_LNOT, ), (B[15:0] NOT ) (FM_RNOT, ), ;
MAIN_OUT :: SO0[15:0] = O0[15:0] (FM_ADD(0), FM_ADD(1), FM_ADD(2), FM_ADD(3), FM
_ADD(4), FM_ADD(5), FM_ADD(6), FM_ADD(7), FM_ADD(8), FM_SUB, FM_INC(0), FM_INC(1
), FM_INC(2), FM_DEC(0), FM_DEC(1), FM_DEC(2), FM_ZERO, FM_ONE, FM_AND, FM_NAND,
 FM_OR, FM_NOR, FM_XOR, FM_XNOR, FM_RIMPL, FM_LIMPL, FM_LINHI, FM_RINHI, FM_LID,
 FM_RID, FM_LNOT, FM_RNOT, ), ;
CARRY_IN :: TICIN[0:0] = C0[0:0] (FM_ADD(0), FM_ADD(1), FM_ADD(2), FM_ADD(3), FM
_ADD(4), FM_ADD(5), FM_ADD(6), FM_ADD(7), FM_ADD(8), FM_SUB, ), "1" (FM_INC(0),
FM_INC(1), FM_INC(2), ), "0" (FM_DEC(0), FM_DEC(1), FM_DEC(2), ), ;
CARRY_OUT :: SOCOUT[0:0] = OCOUT[0:0] (FM_ADD(0), FM_ADD(1), FM_ADD(2), FM_ADD(3
), FM_ADD(4), FM_ADD(5), FM_ADD(6), FM_ADD(7), FM_ADD(8), FM_SUB, FM_INC(0), FM_
INC(1), FM_INC(2), FM_DEC(0), FM_DEC(1), FM_DEC(2), ), ;
OVF :: SOVF[0:0] = OVF[0:0] (FM_ADD(0), FM_ADD(1), FM_ADD(2), FM_ADD(3), FM_ADD(
4), FM_ADD(5), FM_ADD(6), FM_ADD(7), FM_ADD(8), FM_SUB, FM_INC(0), FM_INC(1), FM
_INC(2), FM_DEC(0), FM_DEC(1), FM_DEC(2), ), ;

Cost Function :  Gate-Count
Algorithm :   Dynamic Programming (1)
Time : 564.5 real        420.1 user        43.6 sys

NAME : CANONICAL;
TARGET COMPONENT : VDP5ALU001;
BIT_WIDTH : 16;
GATE_COUNT : 661;
MAX_DELAY :  39.40;
```

FUNCTION-RULE LIST : (FM_ADD(0), AA1), (FM_ADD(1), AA1), (FM_ADD(2), AA1), (FM_A
DD(3), AA1), (FM_ADD(4), AA1), (FM_ADD(5), AA1), (FM_ADD(6), AA1), (FM_ADD(7), A
A1), (FM_ADD(8), AA1), (FM_SUB, SS), (FM_INC(0), IA1), (FM_INC(1), IA1), (FM_INC
(2), IA1), (FM_DEC(0), DA1), (FM_DEC(1), DA1), (FM_DEC(2), DA1), (FM_ZERO, ZOAD1
), (FM_ONE, OEOR1), (FM_AND, ADAD1), (FM_NAND, NDND1), (FM_OR, OROR1), (FM_NOR,
NRNR1), (FM_XOR, XRXR1), (FM_XNOR, XNXN1), (FM_RIMPL, RLRL), (FM_LIMPL, LLLL), (
FM_LINHI, LILI), (FM_RINHI, RIRI), (FM_LID, LDAD1), (FM_RID, RDAD1), (FM_LNOT, L
NAD1), (FM_RNOT, RNAD1), ;

```
LEFT_IN :: TI0[15:0] = A[15:0] (FM_ADD(0), FM_ADD(1), FM_ADD(3), FM_ADD(5), FM_S
UB, FM_INC(0), FM_DEC(0), FM_AND, FM_NAND, FM_OR, FM_NOR, FM_XOR, FM_XNOR, FM_RI
MPL, FM_LIMPL, FM_LINHI, FM_RINHI, FM_LID, ), (A[15:0] AND B[15:0]) (FM_ADD(2),
FM_ADD(6), FM_DEC(1), ), (A[15:0] AND (B[15:0] NOT )) (FM_ADD(4), FM_ADD(7), FM_
DEC(2), ), ("1" REPEAT 16) (FM_ADD(8), FM_ONE, ), (A[15:0] OR (B[15:0] NOT )) (F
M_INC(1), ), (A[15:0] OR B[15:0]) (FM_INC(2), ), ("0" REPEAT 16) (FM_ZERO, ), B[
15:0] (FM_RID, ), (A[15:0] NOT ) (FM_LNOT, ), (B[15:0] NOT ) (FM_RNOT, ), ;
RIGHT_IN :: TI1[15:0] = B[15:0] (FM_ADD(0), FM_SUB, FM_AND, FM_NAND, FM_OR, FM_N
OR, FM_XOR, FM_XNOR, FM_RIMPL, FM_LIMPL, FM_LINHI, FM_RINHI, ), (A[15:0] OR (B[1
5:0] NOT )) (FM_ADD(1), ), (A[15:0] OR B[15:0]) (FM_ADD(2), FM_ADD(3), FM_ADD(4)
, ), A[15:0] (FM_ADD(5), FM_ADD(6), FM_ADD(7), ), ("0" REPEAT 16) (FM_ADD(8), FM
_INC(0), FM_INC(1), FM_INC(2), ), ("1" REPEAT 16) (FM_DEC(0), FM_DEC(1), FM_DEC(
2), FM_LID, FM_RID, FM_LNOT, FM_RNOT, ), ;
MAIN_OUT :: SO0[15:0] = O0[15:0] (FM_ADD(0), FM_ADD(1), FM_ADD(2), FM_ADD(3), FM
_ADD(4), FM_ADD(5), FM_ADD(6), FM_ADD(7), FM_ADD(8), FM_SUB, FM_INC(0), FM_INC(1
), FM_INC(2), FM_DEC(0), FM_DEC(1), FM_DEC(2), FM_ZERO, FM_ONE, FM_AND, FM_NAND,
 FM_OR, FM_NOR, FM_XOR, FM_XNOR, FM_RIMPL, FM_LIMPL, FM_LINHI, FM_RINHI, FM_LID,
 FM_RID, FM_LNOT, FM_RNOT, ), ;
CARRY_IN :: TICIN[0:0] = C0[0:0] (FM_ADD(0), FM_ADD(1), FM_ADD(2), FM_ADD(3), FM
_ADD(4), FM_ADD(5), FM_ADD(6), FM_ADD(7), FM_ADD(8), FM_SUB, ), "1" (FM_INC(0),
FM_INC(1), FM_INC(2), ), "0" (FM_DEC(0), FM_DEC(1), FM_DEC(2), ), ;
CARRY_OUT :: SOCOUT[0:0] = OCOUT[0:0] (FM_ADD(0), FM_ADD(1), FM_ADD(2), FM_ADD(3
), FM_ADD(4), FM_ADD(5), FM_ADD(6), FM_ADD(7), FM_ADD(8), FM_SUB, FM_INC(0), FM_
INC(1), FM_INC(2), FM_DEC(0), FM_DEC(1), FM_DEC(2), ), ;
OVF :: SOVF[0:0] = OVF[0:0] (FM_ADD(0), FM_ADD(1), FM_ADD(2), FM_ADD(3), FM_ADD(
4), FM_ADD(5), FM_ADD(6), FM_ADD(7), FM_ADD(8), FM_SUB, FM_INC(0), FM_INC(1), FM
_INC(2), FM_DEC(0), FM_DEC(1), FM_DEC(2), ), ;

Cost Function :  Gate-Count
Algorithm :   Dynamic Programming (2)
Time :   5716.1 real      5101.6 user       220.5 sys

NAME : CANONICAL;
TARGET COMPONENT : VDP5ALU001;
BIT_WIDTH : 16;
GATE_COUNT : 453;
MAX_DELAY :  36.10;
```

FUNCTION-RULE LIST : (FM_ADD(0), AA1), (FM_ADD(1), AA2), (FM_ADD(2), AA2), (FM_A
DD(3), AA2), (FM_ADD(4), AA2), (FM_ADD(5), AA1), (FM_ADD(6), AA2), (FM_ADD(7), A
A2), (FM_ADD(8), AA1), (FM_SUB, SS), (FM_INC(0), IA1), (FM_INC(1), IA1), (FM_INC
(2), IA1), (FM_DEC(0), DA2), (FM_DEC(1), DA2), (FM_DEC(2), DA2), (FM_ZERO, ZOAD2
), (FM_ONE, OEOR1), (FM_AND, ADAD1), (FM_NAND, NDND1), (FM_OR, OROR1), (FM_NOR,
NRNR1), (FM_XOR, XRXR1), (FM_XNOR, XNXN1), (FM_RIMPL, RLRL), (FM_LIMPL, LLLL), (
FM_LINHI, LILI), (FM_RINHI, RIRI), (FM_LID, LDAD2), (FM_RID, RDAD2), (FM_LNOT, L

```
NAD2), (FM_RNOT, RNAD2), ;

LEFT_IN :: TI0[15:0] = A[15:0] (FM_ADD(0), FM_ADD(5), FM_ADD(6), FM_ADD(7), FM_S
UB, FM_INC(0), FM_AND, FM_NAND, FM_OR, FM_NOR, FM_XOR, FM_XNOR, FM_RIMPL, FM_LIM
PL, FM_LINHI, FM_RINHI, ), (A[15:0] OR (B[15:0] NOT )) (FM_ADD(1), FM_INC(1), ),
 (A[15:0] OR B[15:0]) (FM_ADD(2), FM_ADD(3), FM_ADD(4), FM_INC(2), ), ("1" REPEA
T 16) (FM_ADD(8), FM_DEC(0), FM_DEC(1), FM_DEC(2), FM_ONE, FM_LID, FM_RID, FM_LN
OT, FM_RNOT, ), ;
RIGHT_IN :: TI1[15:0] = B[15:0] (FM_ADD(0), FM_SUB, FM_AND, FM_NAND, FM_OR, FM_N
OR, FM_XOR, FM_XNOR, FM_RIMPL, FM_LIMPL, FM_LINHI, FM_RINHI, FM_RID, ), A[15:0]
(FM_ADD(1), FM_ADD(3), FM_ADD(5), FM_DEC(0), FM_LID, ), (A[15:0] AND B[15:0]) (F
M_ADD(2), FM_ADD(6), FM_DEC(1), ), (A[15:0] AND (B[15:0] NOT )) (FM_ADD(4), FM_A
DD(7), FM_DEC(2), ), ("0" REPEAT 16) (FM_ADD(8), FM_INC(0), FM_INC(1), FM_INC(2)
, FM_ZERO, ), (A[15:0] NOT ) (FM_LNOT, ), (B[15:0] NOT ) (FM_RNOT, ), ;
MAIN_OUT :: SO0[15:0] = O0[15:0] (FM_ADD(0), FM_ADD(1), FM_ADD(2), FM_ADD(3), FM
_ADD(4), FM_ADD(5), FM_ADD(6), FM_ADD(7), FM_ADD(8), FM_SUB, FM_INC(0), FM_INC(1
), FM_INC(2), FM_DEC(0), FM_DEC(1), FM_DEC(2), FM_ZERO, FM_ONE, FM_AND, FM_NAND,
 FM_OR, FM_NOR, FM_XOR, FM_XNOR, FM_RIMPL, FM_LIMPL, FM_LINHI, FM_RINHI, FM_LID,
 FM_RID, FM_LNOT, FM_RNOT, ), ;
CARRY_IN :: TICIN[0:0] = C0[0:0] (FM_ADD(0), FM_ADD(1), FM_ADD(2), FM_ADD(3), FM
_ADD(4), FM_ADD(5), FM_ADD(6), FM_ADD(7), FM_ADD(8), FM_SUB, ), "1" (FM_INC(0),
FM_INC(1), FM_INC(2), ), "0" (FM_DEC(0), FM_DEC(1), FM_DEC(2), ), ;
CARRY_OUT :: SOCOUT[0:0] = OCOUT[0:0] (FM_ADD(0), FM_ADD(1), FM_ADD(2), FM_ADD(3
), FM_ADD(4), FM_ADD(5), FM_ADD(6), FM_ADD(7), FM_ADD(8), FM_SUB, FM_INC(0), FM_
INC(1), FM_INC(2), FM_DEC(0), FM_DEC(1), FM_DEC(2), ), ;
OVF :: SOVF[0:0] = OVF[0:0] (FM_ADD(0), FM_ADD(1), FM_ADD(2), FM_ADD(3), FM_ADD(
4), FM_ADD(5), FM_ADD(6), FM_ADD(7), FM_ADD(8), FM_SUB, FM_INC(0), FM_INC(1), FM
_INC(2), FM_DEC(0), FM_DEC(1), FM_DEC(2), ), ;

Cost Function :  Max_Delay
Algorithm :   Dynamic Programming (1)
Time : 522.3 real       462.5 user        28.7 sys

NAME : CANONICAL;
TARGET COMPONENT : VDP5ALU001;
BIT_WIDTH : 16;
GATE_COUNT : 917;
MAX_DELAY :  42.50;

FUNCTION-RULE LIST : (FM_ADD(0), AA1), (FM_ADD(1), AA1), (FM_ADD(2), AA1), (FM_A
DD(3), AA1), (FM_ADD(4), AA1), (FM_ADD(5), AA1), (FM_ADD(6), AA1), (FM_ADD(7), A
A1), (FM_ADD(8), AA1), (FM_SUB, SS), (FM_INC(0), IA1), (FM_INC(1), IA1), (FM_INC
(2), IA1), (FM_DEC(0), DA1), (FM_DEC(1), DA1), (FM_DEC(2), DA1), (FM_ZERO, ZOAD1
), (FM_ONE, OEOR1), (FM_AND, ADAD1), (FM_NAND, NDND1), (FM_OR, OROR1), (FM_NOR,
NRNR1), (FM_XOR, XRXR1), (FM_XNOR, XNXN1), (FM_RIMPL, RLRL), (FM_LIMPL, LLLL), (
FM_LINHI, LILI), (FM_RINHI, RIRI), (FM_LID, LDAD1), (FM_RID, RDAD1), (FM_LNOT, L
NNE), (FM_RNOT, RNNE), ;

LEFT_IN :: TI0[15:0] = A[15:0] (FM_ADD(0), FM_ADD(1), FM_ADD(3), FM_ADD(5), FM_S
UB, FM_INC(0), FM_DEC(0), FM_AND, FM_NAND, FM_OR, FM_NOR, FM_XOR, FM_XNOR, FM_RI
MPL, FM_LIMPL, FM_LINHI, FM_RINHI, FM_LID, ), (A[15:0] AND B[15:0]) (FM_ADD(2),
FM_ADD(6), FM_DEC(1), ), (A[15:0] AND (B[15:0] NOT )) (FM_ADD(4), FM_ADD(7), FM_
DEC(2), ), ("1" REPEAT 16) (FM_ADD(8), FM_ONE, ), (A[15:0] OR (B[15:0] NOT )) (F
M_INC(1), ), (A[15:0] OR B[15:0]) (FM_INC(2), ), ("0" REPEAT 16) (FM_ZERO, ), B[
```

```
15:0] (FM_RID, ), ;
RIGHT_IN :: TI1[15:0] = B[15:0] (FM_ADD(0), FM_SUB, FM_AND, FM_NAND, FM_OR, FM_N
OR, FM_XOR, FM_XNOR, FM_RIMPL, FM_LIMPL, FM_LINHI, FM_RINHI, ), (A[15:0] OR (B[1
5:0] NOT )) (FM_ADD(1), ), (A[15:0] OR B[15:0]) (FM_ADD(2), FM_ADD(3), FM_ADD(4)
, ), A[15:0] (FM_ADD(5), FM_ADD(6), FM_ADD(7), ), ("0" REPEAT 16) (FM_ADD(8), FM
_INC(0), FM_INC(1), FM_INC(2), ), ("1" REPEAT 16) (FM_DEC(0), FM_DEC(1), FM_DEC(
2), FM_LID, FM_RID, ), ;
LOGIC_LEFT_IN :: TLI0[15:0] = A[15:0] (FM_LNOT, ), SI0[15:0] (FM_RNOT, ), ;
LOGIC_RIGHT_IN :: TLI1[15:0] = SI1[15:0] (FM_LNOT, ), B[15:0] (FM_RNOT, ), ;
MAIN_OUT :: SO0[15:0] = O0[15:0] (FM_ADD(0), FM_ADD(1), FM_ADD(2), FM_ADD(3), FM
_ADD(4), FM_ADD(5), FM_ADD(6), FM_ADD(7), FM_ADD(8), FM_SUB, FM_INC(0), FM_INC(1
), FM_INC(2), FM_DEC(0), FM_DEC(1), FM_DEC(2), FM_ZERO, FM_ONE, FM_AND, FM_NAND,
 FM_OR, FM_NOR, FM_XOR, FM_XNOR, FM_RIMPL, FM_LIMPL, FM_LINHI, FM_RINHI, FM_LID,
 FM_RID, ), LO0[15:0] (FM_LNOT, FM_RNOT, ), ;
CARRY_IN :: TICIN[0:0] = CO[0:0] (FM_ADD(0), FM_ADD(1), FM_ADD(2), FM_ADD(3), FM
_ADD(4), FM_ADD(5), FM_ADD(6), FM_ADD(7), FM_ADD(8), FM_SUB, ), "1" (FM_INC(0),
FM_INC(1), FM_INC(2), ), "0" (FM_DEC(0), FM_DEC(1), FM_DEC(2), ), ;
CARRY_OUT :: SOCOUT[0:0] = OCOUT[0:0] (FM_ADD(0), FM_ADD(1), FM_ADD(2), FM_ADD(3
), FM_ADD(4), FM_ADD(5), FM_ADD(6), FM_ADD(7), FM_ADD(8), FM_SUB, FM_INC(0), FM_
INC(1), FM_INC(2), FM_DEC(0), FM_DEC(1), FM_DEC(2), ), ;
OVF :: SOVF[0:0] = OVF[0:0] (FM_ADD(0), FM_ADD(1), FM_ADD(2), FM_ADD(3), FM_ADD(
4), FM_ADD(5), FM_ADD(6), FM_ADD(7), FM_ADD(8), FM_SUB, FM_INC(0), FM_INC(1), FM
_INC(2), FM_DEC(0), FM_DEC(1), FM_DEC(2), ), ;
LOGIC_OUT :: LO[15:0] = 1,1,0,0 (FM_LNOT, ), 1,0,1,0 (FM_RNOT, ), ;

Cost Function :  Max_Delay
Algorithm :   Dynamic Programming (2)
Time :   4652.5 real      4452.4 user       106.0 sys

NAME : CANONICAL;
TARGET COMPONENT : VDP5ALU001;
BIT_WIDTH : 16;
GATE_COUNT : 485;
MAX_DELAY :  36.20;

FUNCTION-RULE LIST : (FM_ADD(0), AA1), (FM_ADD(1), AA2), (FM_ADD(2), AA2), (FM_A
DD(3), AA2), (FM_ADD(4), AA2), (FM_ADD(5), AA1), (FM_ADD(6), AA2), (FM_ADD(7), A
A2), (FM_ADD(8), AA1), (FM_SUB, SS), (FM_INC(0), IA1), (FM_INC(1), IA1), (FM_INC
(2), IA1), (FM_DEC(0), DA2), (FM_DEC(1), DA2), (FM_DEC(2), DA2), (FM_ZERO, ZOAD2
), (FM_ONE, OEOR1), (FM_AND, ADAD1), (FM_NAND, NDND1), (FM_OR, OROR1), (FM_NOR,
NRNR1), (FM_XOR, XRXR1), (FM_XNOR, XNXN1), (FM_RIMPL, RLRL), (FM_LIMPL, LLLL), (
FM_LINHI, LILI), (FM_RINHI, RIRI), (FM_LID, LDAD2), (FM_RID, RDAD2), (FM_LNOT, L
NAD1), (FM_RNOT, RNAD1), ;

LEFT_IN :: TI0[15:0] = A[15:0] (FM_ADD(0), FM_ADD(5), FM_ADD(6), FM_ADD(7), FM_S
UB, FM_INC(0), FM_AND, FM_NAND, FM_OR, FM_NOR, FM_XOR, FM_XNOR, FM_RIMPL, FM_LIM
PL, FM_LINHI, FM_RINHI, ), (A[15:0] OR (B[15:0] NOT )) (FM_ADD(1), FM_INC(1), ),
 (A[15:0] OR B[15:0]) (FM_ADD(2), FM_ADD(3), FM_ADD(4), FM_INC(2), ), ("1" REPEA
T 16) (FM_ADD(8), FM_DEC(0), FM_DEC(1), FM_DEC(2), FM_ONE, FM_LID, FM_RID, ), (A
[15:0] NOT ) (FM_LNOT, ), (B[15:0] NOT ) (FM_RNOT, ), ;
RIGHT_IN :: TI1[15:0] = B[15:0] (FM_ADD(0), FM_SUB, FM_AND, FM_NAND, FM_OR, FM_N
OR, FM_XOR, FM_XNOR, FM_RIMPL, FM_LIMPL, FM_LINHI, FM_RINHI, FM_RID, ), A[15:0]
(FM_ADD(1), FM_ADD(3), FM_ADD(5), FM_DEC(0), FM_LID, ), (A[15:0] AND B[15:0]) (F
M_ADD(2), FM_ADD(6), FM_DEC(1), ), (A[15:0] AND (B[15:0] NOT )) (FM_ADD(4), FM_A
```

```
DD(7), FM_DEC(2), ), ("0" REPEAT 16) (FM_ADD(8), FM_INC(0), FM_INC(1), FM_INC(2)
, FM_ZERO, ), ("1" REPEAT 16) (FM_LNOT, FM_RNOT, ), ;
MAIN_OUT :: SO0[15:0] = O0[15:0] (FM_ADD(0), FM_ADD(1), FM_ADD(2), FM_ADD(3), FM
_ADD(4), FM_ADD(5), FM_ADD(6), FM_ADD(7), FM_ADD(8), FM_SUB, FM_INC(0), FM_INC(1
), FM_INC(2), FM_DEC(0), FM_DEC(1), FM_DEC(2), FM_ZERO, FM_ONE, FM_AND, FM_NAND,
 FM_OR, FM_NOR, FM_XOR, FM_XNOR, FM_RIMPL, FM_LIMPL, FM_LINHI, FM_RINHI, FM_LID,
 FM_RID, FM_LNOT, FM_RNOT, ), ;
CARRY_IN :: TICIN[0:0] = C0[0:0] (FM_ADD(0), FM_ADD(1), FM_ADD(2), FM_ADD(3), FM
_ADD(4), FM_ADD(5), FM_ADD(6), FM_ADD(7), FM_ADD(8), FM_SUB, ), "1" (FM_INC(0),
FM_INC(1), FM_INC(2), ), "0" (FM_DEC(0), FM_DEC(1), FM_DEC(2), ), ;
CARRY_OUT :: SOCOUT[0:0] = OCOUT[0:0] (FM_ADD(0), FM_ADD(1), FM_ADD(2), FM_ADD(3
), FM_ADD(4), FM_ADD(5), FM_ADD(6), FM_ADD(7), FM_ADD(8), FM_SUB, FM_INC(0), FM_
INC(1), FM_INC(2), FM_DEC(0), FM_DEC(1), FM_DEC(2), ), ;
OVF :: SOVF[0:0] = OVF[0:0] (FM_ADD(0), FM_ADD(1), FM_ADD(2), FM_ADD(3), FM_ADD(
4), FM_ADD(5), FM_ADD(6), FM_ADD(7), FM_ADD(8), FM_SUB, FM_INC(0), FM_INC(1), FM
_INC(2), FM_DEC(0), FM_DEC(1), FM_DEC(2), ), ;
```

B.7 Example 7

```
Source component :


NAME : ALU;
BIT_WIDTH : 16;
GATE_COUNT : 912;
MAX_DELAY : 13.2;

FM_ADD (0) ::
LEFT_IN : I0 = A;
RIGHT_IN : I1 = B;
CARRY_IN : ICIN[0:0] = C0[0:0];
MAIN_OUT : F = O0;
CARRY_OUT : COUT[0:0] = OCOUT[0:0];
OVF_OUT : OVF[0:0] = OVF[0:0];

FM_ADD (1) ::
LEFT_IN : I0 = A;
RIGHT_IN : I1 = A OR (NOT B);
CARRY_IN : ICIN[0:0] = C0[0:0];
MAIN_OUT : F = O0;
CARRY_OUT : COUT[0:0] = OCOUT[0:0];
OVF_OUT : OVF[0:0] = OVF[0:0];

FM_ADD (2) ::
LEFT_IN : I0 = A AND B;
RIGHT_IN : I1 = A OR B;
CARRY_IN : ICIN[0:0] = C0[0:0];
MAIN_OUT : F = O0;
CARRY_OUT : COUT[0:0] = OCOUT[0:0];
OVF_OUT : OVF[0:0] = OVF[0:0];

FM_ADD (3) ::
```

```
LEFT_IN : I0 = A;
RIGHT_IN : I1 = A OR B;
CARRY_IN : ICIN[0:0] = C0[0:0];
MAIN_OUT : F = O0;
CARRY_OUT : COUT[0:0] = OCOUT[0:0];
OVF_OUT : OVF[0:0] = OVF[0:0];

FM_ADD (4) ::
LEFT_IN : I0 = A AND (NOT B);
RIGHT_IN : I1 = A OR B;
CARRY_IN : ICIN[0:0] = C0[0:0];
MAIN_OUT : F = O0;
CARRY_OUT : COUT[0:0] = OCOUT[0:0];
OVF_OUT : OVF[0:0] = OVF[0:0];

FM_ADD (5) ::
LEFT_IN : I0 = A;
RIGHT_IN : I1 = A;
CARRY_IN : ICIN[0:0] = C0[0:0];
MAIN_OUT : F = O0;
CARRY_OUT : COUT[0:0] = OCOUT[0:0];
OVF_OUT : OVF[0:0] = OVF[0:0];

FM_ADD (6) ::
LEFT_IN : I0 = A AND B;
RIGHT_IN : I1 = A;
CARRY_IN : ICIN[0:0] = C0[0:0];
MAIN_OUT : F = O0;
CARRY_OUT : COUT[0:0] = OCOUT[0:0];
OVF_OUT : OVF[0:0] = OVF[0:0];

FM_ADD (7) ::
LEFT_IN : I0 = A AND (NOT B);
RIGHT_IN : I1 = A;
CARRY_IN : ICIN[0:0] = C0[0:0];
MAIN_OUT : F = O0;
CARRY_OUT : COUT[0:0] = OCOUT[0:0];
OVF_OUT : OVF[0:0] = OVF[0:0];

FM_ADD (8) ::
LEFT_IN : I0 = REPEAT 16 "1";
RIGHT_IN : I1 = REPEAT 16 "0";
CARRY_IN : ICIN[0:0] = C0[0:0];
MAIN_OUT : F = O0;
CARRY_OUT : COUT[0:0] = OCOUT[0:0];
OVF_OUT : OVF[0:0] = OVF[0:0];

FM_SUB ::
LEFT_IN : I0 = A;
RIGHT_IN : I1 = B;
CARRY_IN : ICIN[0:0] = C0[0:0];
MAIN_OUT : F = O0;
CARRY_OUT : COUT[0:0] = OCOUT[0:0];
OVF_OUT : OVF[0:0] = OVF[0:0];
```

```
FM_INC (0)::
LEFT_IN : I0 = A;
CARRY_IN : ICIN[0:0] = C0[0:0];
MAIN_OUT : F = O0;
CARRY_OUT : COUT[0:0] = OCOUT[0:0];
OVF_OUT : OVF[0:0] = OVF[0:0];

FM_INC (1) ::
LEFT_IN : I0 = A OR (NOT B);
CARRY_IN : ICIN[0:0] = C0[0:0];
MAIN_OUT : F = O0;
CARRY_OUT : COUT[0:0] = OCOUT[0:0];
OVF_OUT : OVF[0:0] = OVF[0:0];

FM_INC (2) ::
LEFT_IN : I0 = A OR B;
CARRY_IN : ICIN[0:0] = C0[0:0];
MAIN_OUT : F = O0;
CARRY_OUT : COUT[0:0] = OCOUT[0:0];
OVF_OUT : OVF[0:0] = OVF[0:0];

FM_DEC (0) ::
LEFT_IN : I0 = A;
CARRY_IN : ICIN[0:0] = C0[0:0];
MAIN_OUT : F = O0;
CARRY_OUT : COUT[0:0] = OCOUT[0:0];
OVF_OUT : OVF[0:0] = OVF[0:0];

FM_DEC (1) ::
LEFT_IN : I0 = A AND B;
CARRY_IN : ICIN[0:0] = C0[0:0];
MAIN_OUT : F = O0;
CARRY_OUT : COUT[0:0] = OCOUT[0:0];
OVF_OUT : OVF[0:0] = OVF[0:0];

FM_DEC (2) ::
LEFT_IN : I0 = A AND (NOT B);
CARRY_IN : ICIN[0:0] = C0[0:0];
MAIN_OUT : F = O0;
CARRY_OUT : COUT[0:0] = OCOUT[0:0];
OVF_OUT : OVF[0:0] = OVF[0:0];

FM_ZERO ::
MAIN_OUT : S0 = O0;

FM_ONE ::
MAIN_OUT : S0 = O0;

FM_AND ::
LEFT_IN : I0 =  A;
RIGHT_IN : I1 = B;
MAIN_OUT : S0 = O0;
```

```
FM_NAND ::
LEFT_IN : I0 =  A;
RIGHT_IN : I1 = B;
MAIN_OUT : SO = O0;

FM_OR ::
LEFT_IN : I0 =  A;
RIGHT_IN : I1 = B;
MAIN_OUT : SO = O0;

FM_NOR ::
LEFT_IN : I0 =  A;
RIGHT_IN : I1 = B;
MAIN_OUT : SO = O0;

FM_XOR ::
LEFT_IN : I0 =  A;
RIGHT_IN : I1 = B;
MAIN_OUT : SO = O0;

FM_XNOR ::
LEFT_IN : I0 =  A;
RIGHT_IN : I1 = B;
MAIN_OUT : SO = O0;

FM_RIMPL ::
LEFT_IN : I0 =  A;
RIGHT_IN : I1 = B;
MAIN_OUT : SO = O0;

FM_LIMPL ::
LEFT_IN : I0 =  A;
RIGHT_IN : I1 = B;
MAIN_OUT : SO = O0;

FM_LINHI ::
LEFT_IN : I0 =  A;
RIGHT_IN : I1 = B;
MAIN_OUT : SO = O0;

FM_RINHI ::
LEFT_IN : I0 =  A;
RIGHT_IN : I1 = B;
MAIN_OUT : SO = O0;

FM_LID ::
LEFT_IN : I0 =  A;
MAIN_OUT : SO = O0;

FM_RID ::
RIGHT_IN : I1 =  B;
MAIN_OUT : SO = O0;

FM_LNOT ::
```

```
LEFT_IN : I0 =  A;
MAIN_OUT : S0 = O0;

FM_RNOT ::
RIGHT_IN : I1 = B;
MAIN_OUT : S0 = O0;

Target component :

NAME : ALU1;
BIT_WIDTH : 16;
GATE_COUNT : 8;
MAX_DELAY : 8;

FM_ADD ::
LEFT_IN : I0 = R;
RIGHT_IN : I1 = S;
CARRY_IN : ICIN[0:0] = Cn[0:0];
MAIN_OUT : F = O0;
CARRY_OUT : Cn4[0:0] = OCOUT[0:0];
OVF_OUT : OVR[0:0] = OVF[0:0];

FM_SUB ::
LEFT_IN : I0 = R;
RIGHT_IN : I1 = S;
CARRY_IN : ICIN[0:0] = Cn[0:0];
MAIN_OUT : F = O0;
CARRY_OUT : Cn4[0:0] = OCOUT[0:0];
OVF_OUT : OVR[0:0] = OVF[0:0];

FM_RSUB ::
LEFT_IN : I0 = R;
RIGHT_IN : I1 = S;
CARRY_IN : ICIN[0:0] = Cn[0:0];
MAIN_OUT : F = O0;
CARRY_OUT : Cn4[0:0] = OCOUT[0:0];
OVF_OUT : OVR[0:0] = OVF[0:0];

FM_OR ::
LEFT_IN : I0 = R;
RIGHT_IN : I1 = S;
MAIN_OUT : F = O0;
FM_AND ::
LEFT_IN : I0 = R;
RIGHT_IN : I1 = S;
MAIN_OUT : F = O0;
FM_LINHI ::
LEFT_IN : I0 = R;
RIGHT_IN : I1 = S;
MAIN_OUT : F = O0;
FM_XOR ::
LEFT_IN : I0 = R;
RIGHT_IN : I1 = S;
MAIN_OUT : F = O0;
```

```
FM_XNOR ::
LEFT_IN : I0 = R;
RIGHT_IN : I1 = S;
MAIN_OUT : F = O0;

Solutions are :

Cost Function :  Gate_Count
Algorithm Type :  1-Greedy
Time : 16.2 real        12.2 user         1.0 sys

NAME : CANONICAL;
TARGET COMPONENT : ALU1;
BIT_WIDTH : 16;
GATE_COUNT : 757;
MAX_DELAY :  25.60;

FUNCTION-RULE LIST : (FM_ADD(0), AA1), (FM_ADD(1), AA1), (FM_ADD(2), AA1), (FM_A
DD(3), AA1), (FM_ADD(4), AA1), (FM_ADD(5), AA1), (FM_ADD(6), AA1), (FM_ADD(7), A
A1), (FM_ADD(8), AA1), (FM_SUB, SS), (FM_INC(0), IA1), (FM_INC(1), IR1), (FM_INC
(2), IR1), (FM_DEC(0), DA2), (FM_DEC(1), DS1), (FM_DEC(2), DS1), (FM_ZERO, ZOAD2
), (FM_ONE, OEOR1), (FM_AND, ADAD1), (FM_NAND, NDAD1), (FM_OR, OROR1), (FM_NOR,
NROR1), (FM_XOR, XRXR1), (FM_XNOR, XNXN1), (FM_RIMPL, RLLI), (FM_LIMPL, LLNE), (
FM_LINHI, LILI), (FM_RINHI, RILI), (FM_LID, LDAD2), (FM_RID, RDAD2), (FM_LNOT, L
NAD2), (FM_RNOT, RNAD2), ;

LEFT_IN :: TI0[15:0] = A[15:0] (FM_ADD(0), FM_ADD(1), FM_ADD(3), FM_ADD(5), FM_S
UB, FM_INC(0), FM_AND, FM_NAND, FM_OR, FM_NOR, FM_XOR, FM_XNOR, FM_RIMPL, FM_LIN
HI, ), (A[15:0] AND B[15:0]) (FM_ADD(2), FM_ADD(6), FM_DEC(1), ), (A[15:0] AND (
B[15:0] NOT )) (FM_ADD(4), FM_ADD(7), FM_DEC(2), ), ("1" REPEAT 16) (FM_ADD(8),
FM_INC(1), FM_INC(2), FM_DEC(0), FM_ONE, FM_LID, FM_RID, FM_LNOT, FM_RNOT, ), B[
15:0] (FM_RINHI, ), ;
RIGHT_IN :: TI1[15:0] = B[15:0] (FM_ADD(0), FM_SUB, FM_AND, FM_NAND, FM_OR, FM_N
OR, FM_XOR, FM_XNOR, FM_RIMPL, FM_LINHI, FM_RID, ), (A[15:0] OR (B[15:0] NOT ))
(FM_ADD(1), FM_INC(1), ), (A[15:0] OR B[15:0]) (FM_ADD(2), FM_ADD(3), FM_ADD(4),
 FM_INC(2), ), A[15:0] (FM_ADD(5), FM_ADD(6), FM_ADD(7), FM_DEC(0), FM_RINHI, FM
_LID, ), ("0" REPEAT 16) (FM_ADD(8), FM_INC(0), FM_DEC(1), FM_DEC(2), FM_ZERO, )
, (A[15:0] NOT ) (FM_LNOT, ), (B[15:0] NOT ) (FM_RNOT, ), ;
LOGIC_LEFT_IN :: TLI0[15:0] = A[15:0] (FM_LIMPL, ), ;
LOGIC_RIGHT_IN :: TLI1[15:0] = B[15:0] (FM_LIMPL, ), ;
MAIN_OUT :: SO0[15:0] = O0[15:0] (FM_ADD(0), FM_ADD(1), FM_ADD(2), FM_ADD(3), FM
_ADD(4), FM_ADD(5), FM_ADD(6), FM_ADD(7), FM_ADD(8), FM_SUB, FM_INC(0), FM_INC(1
), FM_INC(2), FM_DEC(0), FM_DEC(1), FM_DEC(2), FM_ZERO, FM_ONE, FM_AND, FM_OR, F
M_XOR, FM_XNOR, FM_LINHI, FM_RINHI, FM_LID, FM_RID, FM_LNOT, FM_RNOT, ), (O0[15:
0] NOT ) (FM_NAND, FM_NOR, FM_RIMPL, ), LO0[15:0] (FM_LIMPL, ), ;
CARRY_IN :: TICIN[0:0] = C0[0:0] (FM_ADD(0), FM_ADD(1), FM_ADD(2), FM_ADD(3), FM
_ADD(4), FM_ADD(5), FM_ADD(6), FM_ADD(7), FM_ADD(8), FM_SUB, ), "1" (FM_INC(0),
FM_INC(1), FM_INC(2), ), "0" (FM_DEC(0), FM_DEC(1), FM_DEC(2), ), ;
CARRY_OUT :: SOCOUT[0:0] = OCOUT[0:0] (FM_ADD(0), FM_ADD(1), FM_ADD(2), FM_ADD(3
), FM_ADD(4), FM_ADD(5), FM_ADD(6), FM_ADD(7), FM_ADD(8), FM_SUB, FM_INC(0), FM_
INC(1), FM_INC(2), FM_DEC(0), FM_DEC(1), FM_DEC(2), ), ;
OVF :: SOVF[0:0] = OVF[0:0] (FM_ADD(0), FM_ADD(1), FM_ADD(2), FM_ADD(3), FM_ADD(
4), FM_ADD(5), FM_ADD(6), FM_ADD(7), FM_ADD(8), FM_SUB, FM_INC(0), FM_INC(1), FM
_INC(2), FM_DEC(0), FM_DEC(1), FM_DEC(2), ), ;
```

```
LOGIC_OUT :: LO[15:0] = 1,1,0,1 (FM_LIMPL, ), ;

Cost Function :  Gate_Count
Algorithm Type :  Dynamic Programming(1)
Time : 517.0 real       491.5 user       14.7 sys

NAME : CANONICAL;
TARGET COMPONENT : ALU1;
BIT_WIDTH : 16;
GATE_COUNT : 933;
MAX_DELAY :  38.20;

FUNCTION-RULE LIST : (FM_ADD(0), AA1), (FM_ADD(1), AA1), (FM_ADD(2), AA1),
(FM_ADD(3), AA1), (FM_ADD(4), AA1), (FM_ADD(5), AA1), (FM_ADD(6), AA1),
(FM_ADD(7), AA1), (FM_ADD(8), AA1), (FM_SUB, SS), (FM_INC(0), IA1),
(FM_INC(1), IA1), (FM_INC(2), IA1), (FM_DEC(0), DA1), (FM_DEC(1), DA1),
(FM_DEC(2), DA1), (FM_ZERO, ZOAD1), (FM_ONE, OEOR1), (FM_AND, ADAD1),
(FM_NAND, NDAD1), (FM_OR, OROR1), (FM_NOR, NROR1), (FM_XOR, XRXR1),
(FM_XNOR, XNXN1), (FM_RIMPL, RLLI), (FM_LIMPL, LLNE), (FM_LINHI, LILI),
(FM_RINHI, RILI), (FM_LID, LDAD1), (FM_RID, RDAD1), (FM_LNOT, LNAD1), (FM_RNOT, RNAD1), ;

LEFT_IN :: TI0[15:0] = A[15:0] (FM_ADD(0), FM_ADD(1), FM_ADD(3), FM_ADD(5),
FM_SUB, FM_INC(0), FM_DEC(0), FM_AND, FM_NAND, FM_OR, FM_NOR, FM_XOR, FM_XNOR,
FM_RIMPL, FM_LINHI, FM_LID, ), (A[15:0] AND B[15:0]) (FM_ADD(2), FM_ADD(6),
FM_DEC(1), ), (A[15:0] AND (B[15:0] NOT )) (FM_ADD(4), FM_ADD(7), FM_DEC(2), ),
 ("1" REPEAT 16) (FM_ADD(8), FM_ONE, ), (A[15:0] OR (B[15:0] NOT )) (FM_INC(1),
), (A[15:0] OR B[15:0]) (FM_INC(2), ), ("0" REPEAT 16) (FM_ZERO, ), B[15:0]
(FM_RINHI, FM_RID, ), (A[15:0] NOT ) (FM_LNOT, ), (B[15:0] NOT ) (FM_RNOT, ), ;
RIGHT_IN :: TI1[15:0] = B[15:0] (FM_ADD(0), FM_SUB, FM_AND, FM_NAND, FM_OR,
FM_NOR, FM_XOR, FM_XNOR, FM_RIMPL, FM_LINHI, ), (A[15:0] OR (B[15:0] NOT ))
(FM_ADD(1), ), (A[15:0] OR B[15:0]) (FM_ADD(2), FM_ADD(3), FM_ADD(4), ),
A[15:0] (FM_ADD(5), FM_ADD(6), FM_ADD(7), FM_RINHI, ), ("0" REPEAT 16)
(FM_ADD(8), FM_INC(0), FM_INC(1), FM_INC(2), ), ("1" REPEAT 16) (FM_DEC(0),
FM_DEC(1), FM_DEC(2), FM_LID, FM_RID, FM_LNOT, FM_RNOT, ), ;
LOGIC_LEFT_IN :: TLI0[15:0] = A[15:0] (FM_LIMPL, ), ;
LOGIC_RIGHT_IN :: TLI1[15:0] = B[15:0] (FM_LIMPL, ), ;
MAIN_OUT :: SO0[15:0] = O0[15:0] (FM_ADD(0), FM_ADD(1), FM_ADD(2), FM_ADD(3), FM_ADD(4),
FM_ADD(5), FM_ADD(6), FM_ADD(7), FM_ADD(8), FM_SUB, FM_INC(0), FM_INC(1), FM_INC(2),
FM_DEC(0), FM_DEC(1), FM_DEC(2), FM_ZERO, FM_ONE, FM_AND, FM_OR, FM_XOR, FM_XNOR, FM_LINHI,
FM_RINHI, FM_LID, FM_RID, FM_LNOT, FM_RNOT, ), (O0[15:0] NOT )
(FM_NAND, FM_NOR, FM_RIMPL, ), LO0[15:0] (FM_LIMPL, ), ;
CARRY_IN :: TICIN[0:0] = C0[0:0] (FM_ADD(0), FM_ADD(1), FM_ADD(2), FM_ADD(3), FM_ADD(4),
FM_ADD(5), FM_ADD(6), FM_ADD(7), FM_ADD(8), FM_SUB, ), "1" (FM_INC(0), FM_INC(1), FM_INC(2), ),
"0" (FM_DEC(0), FM_DEC(1), FM_DEC(2), ), ;
CARRY_OUT :: SOCOUT[0:0] = OCOUT[0:0] (FM_ADD(0), FM_ADD(1), FM_ADD(2), FM_ADD(3), FM_ADD(4),
FM_ADD(5), FM_ADD(6), FM_ADD(7), FM_ADD(8), FM_SUB, FM_INC(0), FM_INC(1), FM_INC(2),
FM_DEC(0), FM_DEC(1), FM_DEC(2), ), ;
OVF :: SOVF[0:0] = OVF[0:0] (FM_ADD(0), FM_ADD(1), FM_ADD(2), FM_ADD(3),
FM_ADD(4), FM_ADD(5), FM_ADD(6), FM_ADD(7), FM_ADD(8), FM_SUB, FM_INC(0),
FM_INC(1), FM_INC(2), FM_DEC(0), FM_DEC(1), FM_DEC(2), ), ;
LOGIC_OUT :: LO[15:0] = 1,1,0,1 (FM_LIMPL, ), ;

Cost Function :  Gate_Count
Algorithm Type :  Dynammic Programming (2)
```

```
Time :   5860.2 real      5330.4 user       170.3 sys

NAME : CANONICAL;
TARGET COMPONENT : ALU1;
BIT_WIDTH : 16;
GATE_COUNT : 757;
MAX_DELAY :  35.00;

FUNCTION-RULE LIST : (FM_ADD(0), AA1), (FM_ADD(1), AA1), (FM_ADD(2), AA1), (FM_ADD(3), AA1),
(FM_ADD(4), AA1), (FM_ADD(5), AA1), (FM_ADD(6), AA1), (FM_ADD(7), AA1), (FM_ADD(8), AA2),
(FM_SUB, SS), (FM_INC(0), IA2), (FM_INC(1), IA2), (FM_INC(2), IA2), (FM_DEC(0), DA1),
(FM_DEC(1), DA1), (FM_DEC(2), DA1), (FM_ZERO, ZOAD1), (FM_ONE, OEOR2), (FM_AND, ADAD1),
(FM_NAND, NDAD1), (FM_OR, OROR1), (FM_NOR, NROR1), (FM_XOR, XRXR1), (FM_XNOR, XNXN1),
(FM_RIMPL, RLLI), (FM_LIMPL, LLNE), (FM_LINHI, LILI), (FM_RINHI, RILI), (FM_LID, LDAD1),
(FM_RID, RDAD1), (FM_LNOT, LNAD1), (FM_RNOT, RNAD1), ;

LEFT_IN :: TI0[15:0] = A[15:0] (FM_ADD(0), FM_ADD(1), FM_ADD(3), FM_ADD(5), FM_SUB, FM_DEC(0),
FM_AND, FM_NAND, FM_OR, FM_NOR, FM_XOR, FM_XNOR, FM_RIMPL, FM_LINHI, FM_LID, ), (A[15:0]
AND B[15:0]) (FM_ADD(2), FM_ADD(6), FM_DEC(1), ), (A[15:0] AND (B[15:0] NOT )) (FM_ADD(4),
FM_ADD(7), FM_DEC(2), ), ("0" REPEAT 16) (FM_ADD(8), FM_INC(0), FM_INC(1), FM_INC(2), FM_ZERO, ),
B[15:0] (FM_RINHI, FM_RID, ), (A[15:0] NOT ) (FM_LNOT, ), (B[15:0] NOT ) (FM_RNOT, ), ;
RIGHT_IN :: TI1[15:0] = B[15:0] (FM_ADD(0), FM_SUB, FM_AND, FM_NAND, FM_OR, FM_NOR, FM_XOR,
FM_XNOR, FM_RIMPL, FM_LINHI, ), (A[15:0] OR (B[15:0] NOT )) (FM_ADD(1), FM_INC(1), ),
(A[15:0] OR B[15:0]) (FM_ADD(2), FM_ADD(3), FM_ADD(4), FM_INC(2), ), A[15:0] (FM_ADD(5),
FM_ADD(6), FM_ADD(7), FM_INC(0), FM_RINHI, ), ("1" REPEAT 16) (FM_ADD(8), FM_DEC(0), FM_DEC(1),
FM_DEC(2), FM_ONE, FM_LID, FM_RID, FM_LNOT, FM_RNOT, ), ;
LOGIC_LEFT_IN :: TLI0[15:0] = A[15:0] (FM_LIMPL, ), ;
LOGIC_RIGHT_IN :: TLI1[15:0] = B[15:0] (FM_LIMPL, ), ;
MAIN_OUT :: SO0[15:0] = O0[15:0] (FM_ADD(0), FM_ADD(1), FM_ADD(2), FM_ADD(3), FM_ADD(4), FM_ADD(5),
FM_ADD(6), FM_ADD(7), FM_ADD(8), FM_SUB, FM_INC(0), FM_INC(1), FM_INC(2), FM_DEC(0), FM_DEC(1),
FM_DEC(2), FM_ZERO, FM_ONE, FM_AND, FM_OR, FM_XOR, FM_XNOR, FM_LINHI, FM_RINHI, FM_LID, FM_RID,
FM_LNOT, FM_RNOT, ), (O0[15:0] NOT ) (FM_NAND, FM_NOR, FM_RIMPL, ), LO0[15:0] (FM_LIMPL, ), ;
CARRY_IN :: TICIN[0:0] = C0[0:0] (FM_ADD(0), FM_ADD(1), FM_ADD(2), FM_ADD(3), FM_ADD(4), FM_ADD(5),
FM_ADD(6), FM_ADD(7), FM_ADD(8), FM_SUB, ), "1" (FM_INC(0), FM_INC(1), FM_INC(2), ), "0" (FM_DEC(0),
FM_DEC(1), FM_DEC(2), ), ;
CARRY_OUT :: SOCOUT[0:0] = OCOUT[0:0] (FM_ADD(0), FM_ADD(1), FM_ADD(2), FM_ADD(3), FM_ADD(4),
FM_ADD(5), FM_ADD(6), FM_ADD(7), FM_ADD(8), FM_SUB, FM_INC(0), FM_INC(1), FM_INC(2), FM_DEC(0),
FM_DEC(1), FM_DEC(2), ), ;
OVF :: SOVF[0:0] = OVF[0:0] (FM_ADD(0), FM_ADD(1), FM_ADD(2), FM_ADD(3),
FM_ADD(4), FM_ADD(5), FM_ADD(6), FM_ADD(7), FM_ADD(8), FM_SUB, FM_INC(0),
FM_INC(1), FM_INC(2), FM_DEC(0), FM_DEC(1), FM_DEC(2), ), ;
LOGIC_OUT :: LO[15:0] = 1,1,0,1 (FM_LIMPL, ), ;
```