Systems Synthesis with Multi-Value Logic (MVL) Quaternary Logic Synthesis

João Miguel Tavares Severino
Instituto Superior Técnico
Lisboa, Portugal
Email: joao.severino@tecnico.ulisboa.pt

***Abstract*—The utilization of Multiple-Valued Logic (MVL) in logic circuits has the potential to reduce the number of logic elements and interconnections that connect different parts of the circuit. With the reduction of the interconnections, delays, area and energy consumption can be reduced. In this thesis we propose a technology mapping tool that implements circuits using recently proposed 2-input Quaternary Lookup Tables (QLUTs), taking advantage of the benefits of MVL to produce more efficient circuits. The mapping functionality was implemented using MVSIS as a base platform. MVSIS reads the circuit specification from a file and creates a network representation, which is then used by the tool we developed to perform the decomposition of the network and mapping into the target technology. Overall, the results show a reduction in the number of interconnections, but this is offset by the increase in occupied area by the Lookup Tables (LUTs), due to the fact that a QLUT requires more transistors to be implemented than a binary LUT. The conclusion taken from this work is that, although the mapping tool produces circuits that, in some cases, are more efficient than their binary equivalent, there is still room for further optimizations in both the mapping tool and the implementation of the QLUT.**

Keywords: Multiple-valued logic (MVL), system synthesis, technology mapping, Quaternary Lookup Table (QLUT)

I. Introduction

Digital circuits are prevalent in current technology and nearly all of these are based on binary logic for its simplicity of interpretation and implementation.

Although binary logic presents several advantages that make it appealing to be used in circuits, there are situations where the nature of the problem benefits from a different representation.

Examples of situations such as this are a constant presence in the real world and a few are presented below.

- **Traffic light** red, yellow, green
- **Playing card suits** clubs, diamonds, hearts, spades
- **DNA nucleotides** Adenine, Cytosine, Guanine, Thymine
- **Days of the week** Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday

As seen in the previous examples, with MVL more information can be represented with the same amount of signals, i.e. just 1 ternary signal is required to represent the color of a traffic light.

Another benefit of using MVL is being able to reduce the number of logic gates and interconnections required to implement a given functionality. With this reduction some limitations of the ever shrinking manufacturing process, such as energy dissipation and signal crosstalk, can be attenuated.

To be able to leverage from MVL benefits both hardware, logic gates, and software, mapping tool, have to become available.

Although several MVL devices have been proposed and developed[1, 2, 3, 4], no viable general purpose gate had been proposed until recently when a voltage mode QLUT structure was presented [5], that several limitations were overcome and a viable gate has become available to be used in MVL logic circuits.

To automate the process of creating circuits using the QLUT technology, we propose the development of a mapping tool, described in Section III, that converts a logic level circuit specification into the required QLUTs and their respective memory configurations and interconnections.

To test the developed mapping tool, benchmarks were used to compare the implementation of the same in both binary and quaternary form and evaluate the performance of the tool. The results of these benchmarks are presented in Section IV.

II. State of the Art

In the beginning of the 20^{th} century, mathematicians, logicians and philosophers, needing to better represent varying degrees of truthfulness, started investigating systems that allowed for more than just "true" or "false" [6]. These needs led to the concept and development of several forms of MVL to use in the representability of problems.

Engineers from various areas also started looking into MVL [7, 8, 9] as a source for improvements in technology.

One of the technological improvements that can benefit from MVL, is in the area of electronic devices and circuit design where interconnection area and delay has become a major concern. Higher-radix systems, not only provide a better relation between circuits and natural representation, but also provide a way of reducing the number of interconnections required to transmit information. As illustrated by Figure 1, to transmit the decimal number "219" the amount of interconnections reduces as the radix increases, from 8 in binary to 4 in quaternary to 3 in decimal.

Although MVL presents numerous advantages, the feasibility of such systems depends upon two factors, namely, the availability of reliable gate implementations and adequate synthesis tools and techniques.

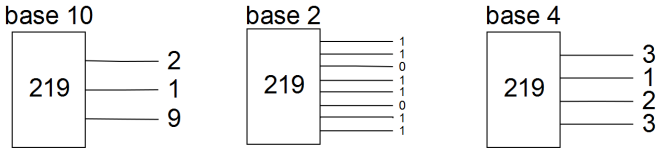


Fig. 1: Simple signal representation for different radixes.

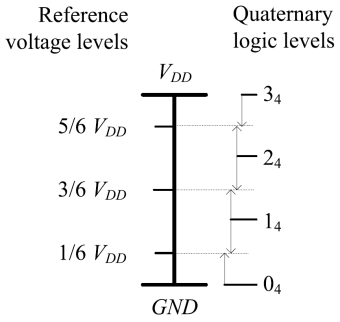


Fig. 2: Quaternary logic and reference voltages levels.

A. MVL Gates and QLUT

Devices have been proposed and implemented using MVL for several years, namely, memories [1], combinational circuits [2, 3] and programmable devices [4]. Mainly in memories, manufacturers have used MVL to increase storage density and have developed commercially available products, first Intel with the StrataFlash™ Memory Technology [10] and then other manufacturers [11] with Solid-State Drives (SSDs).

Although commercially available exist with memories, when it comes to standard gates, several implementations have been proposed to be used in MVL circuits but until now few represent a viable alternative to regular binary Complementary Metal-Oxide-Semiconductor (CMOS) gates. These implementations of MVL circuits can be fundamentally categorized as Current-mode or Voltage-mode operation.

Current-mode logic gates work on the principle of Kirchhoff's current law where logic levels are represented by different currents that can be easily added and subtracted to get the desired output, making arithmetic operations very simple to implement, as incoming currents add up to the outgoing current.

The main disadvantage of this approach is a higher power consumption inherent to the current constantly flowing through the circuit.

Voltage-mode gates work by having several stable intermediate voltages levels between fully discharged and fully charged, usually requiring non-standard technologies, but benefit from having a lower power consumption when compared with current-mode logic gates.

1) QLUT: In [5], a new approach on a voltage mode QLUT is presented, that overcomes the drawback of voltage mode gates previously presented by using single supply voltage of $1.2V$ and standard CMOS technology and presents a viable way of implementing MVL circuits.

The proposed 2-input 1-output QLUT structure, shown in Figure 3, in order to guarantee equal noise margins for all four logic levels, uses three different reference voltage values are required, $\frac{1}{6}VDD$, $\frac{3}{6}VDD$, and $\frac{5}{6}VDD$, to determine a quaternary value, shown in Figure 2.

In the same manner as in a regular LUT, the QLUT behaves as an array or a multiplexer, where the input signals select which logic value is connected to the output, based on the configuration memory. To program the QLUT memory configuration 16 quaternary values (V_1 through V_{16} in the Figure 3 are required.

This multiplexer action and programmable memory, allows the QLUT to implement any desired logic function making it perfect for use in MVL applications.

B. Logic Synthesis

In order to optimize and automate the process of designing logic circuits, Computer Aided Design (CAD) software is commonly used. Logic synthesis, an important component of Electronic Design Automation (EDA), translates a high level description of a circuit design, such as an Hardware Description Language (HDL), into a gate-level representation. Typically, logic synthesis is divided in two separate phases, logic optimization and technology mapping.

MVL synthesis tools and techniques have been a subject of great interest for years [12, 13, 14, 15, 16] and resulted in tools that have been evolving over time, namely from Synopsys and Cadence.

However, these are proprietary software and as such most details and information are not public. Therefore, we focused on academic open-source alternatives, from several universities, that have made research in the field of logic synthesis tools.

SIS, MVSIS, ABC from University of California, Berkeley
BOLD from University of Colorado, Boulder
VIS from University of California, Berkeley; University of Colorado, Boulder and University of Texas, Austin.
RASP from University of California, Los Angeles

Although several tools are available, as shown in the previous example, our interest is in tools capable of handling MVL circuits, i.e. ABC and MVSIS.

1) BLIF-MV: Both ABC and MVSIS use BLIF-MV [24, 25] file format as an intermediate language between a high level HDL, such as VHDL or Verilog, and the synthesis and verification systems.

The BLIF-MV file format is an extension of the Berkeley Logic Interchange Format (BLIF) with the main difference between the two being that BLIF-MV supports MVL signals.

2) ABC: ABC [23] is a software system, currently the most advanced open source package in sequential synthesis and verification tools for logic circuits, developed based on previous systems such as SIS, VIS and MVSIS.

It combines logic optimization with delay aware technology mapping for binary LUTs and standard cells, along with optimized algorithms for sequential synthesis and verification.

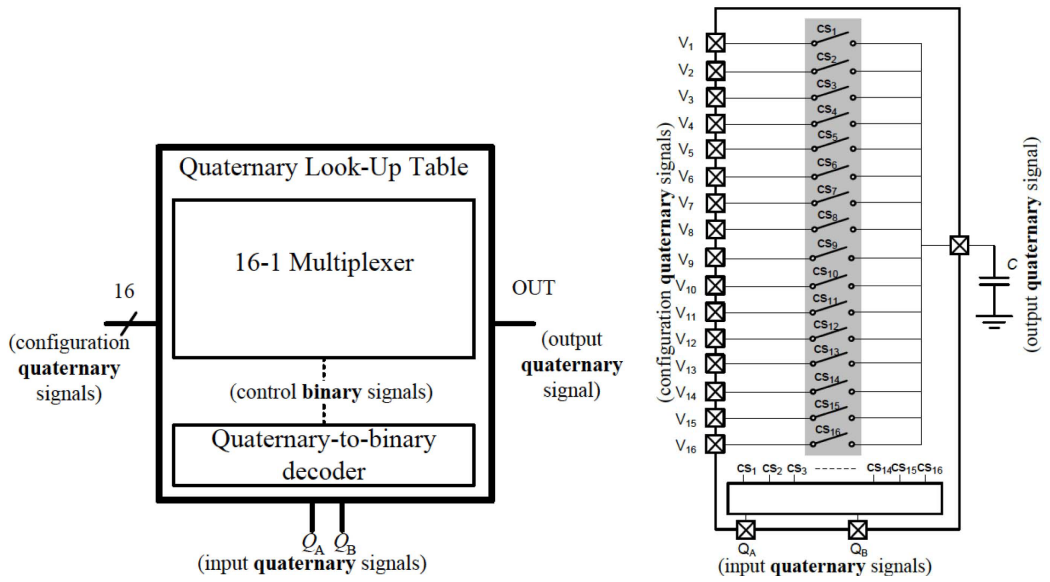


Fig. 3: Quaternary Lookup Table (QLUT).

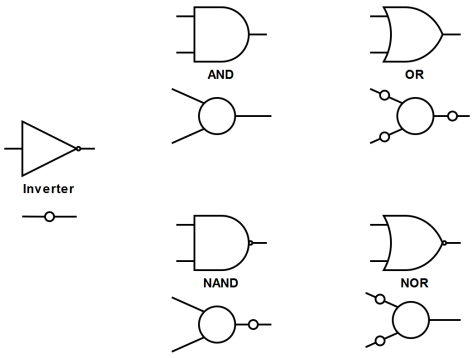


Fig. 4: Basic Logic Gates and the correspondent AIG representation.

In ABC, the internal representation of circuits and logic functions is done with a data structure named And-Inverter Graphs (AIGs). AIGs are multi-level logic networks of two-input AND gates and inverters, with this representation it is possible to have a simple and flexible data structure that can represent any logic function/gate by simply modifying the incoming and outgoing edges of the AND gate, through the placement or removal of inverters. In Figure 4 there is an AIG representation of the basic binary gates.

Despite being able to process MVL circuits, through BLIF-MV files, all of the information is converted to the AIG data structure. Although a very efficient structure to process binary circuits, its advantages in binary logic do not apply very well when the main goal of this work is to synthesize and map MVL logic into the QLUT logic gates overviewed in Section II-A1.

3) MVSIS: MVSIS (Multi-Valued Sequential Interactive Synthesis) [22] is a logic synthesis and verification tool for Very Large Scale Integration (VLSI) design, developed for implementing new synthesis algorithms for logic optimization, targeting both MVL and binary networks.

In MVSIS, variables of the logic network are binary by default but can be multi-valued, each with its own range. An MVL network can be read to MVSIS as a netlist of MVL nodes using the BLIF-MV format. After a design specification is read, it is converted into a MVL network and stored in MVSIS internal representation.

The internal representation consists of a network of nodes, each with its own range, where in each node, i-sets are used to store the node functionality that represents the relation of the inputs to a single output.

The functions associated with each of the i-sets are stored as Sum of Products (SOP) or Multi-Valued Decisions Diagrams (MDD) form. They are a binary-output function which evaluate "true" if the output can assume the value i.

An example of i-sets is here presented for the binary function $F = a\ AND\ b$, which has two i-sets, 0-set and 1-set with the following definition

0-set $F^{(0)} = a^{(0)} + b^{(0)}$
1-set $F^{(1)} = a^{(1)}b^{(1)}$

where $a^{(0)}$ means that when the input signal a assumes the logic value 0 the function F takes on the logic value of 0 that corresponds to the 0-set.

A similar approach is used to deal with MVL where an i-set is used for each logic level, i.e. a quaternary node would have a 0-set, 1-set, 2-set and 3-set.

This representation presents an advantage over the one use in ABC because it allows all the operations performed over the network to be done directly in MVL without requiring any conversions or signal pairing to get the final result. This makes MVSIS the ideal base platform to expand upon with the MVL mapping functionality.

III. Developed Work

To develop the mapping functionality, capable of converting a MVL design specification into a set of QLUTs, the command `mapMVL` was created inside the MVSIS framework.

The `mapMVL` command works similarly to the existing MVSIS `fpga` command, but instead of working on binary networks, it converts quaternary networks into a mapped form, using the new QLUT technology proposed in [5] and overviewed in Section II-A1.

The command is divided in several steps which are described in the following sections.

A. Initial Process

Before any mapping occurs, a few verifications and configurations are done, in preparation for the next steps.

After issuing the command `mapMVL` from the MVSIS main menu, the mapping function receives the MVSIS framework as well as any additional parameters/options specified by the user. With this, we first retrieve the network from the MVSIS framework and parse the input options.

Also verifications are done to ensure that the mapping process encounters no problems. The verifications consist of having a non-empty valid network, all nodes inside the network must be quaternary and have a default value associated and finally the output file, if specified, must be accessible and writable.

Finally, the naming mode inside MVSIS is set to long naming scheme to guarantee consistency between the input file and the names of the generated QLUTs. And if verbose mode has been selected, a few general informations about the network are printed to the terminal.

B. Network Processing

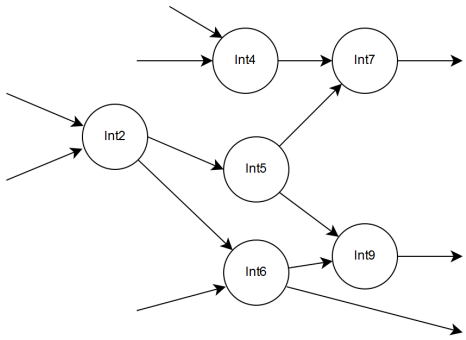
Now with all verifications and configurations done, we have a valid network ready to be processed. This happens, for each individual node, in two stages, the first one consisting of node refactoring and the second comprised of node decomposition and optimization.

In both stages the main goal is to adapt the network to be compatible with the target technology of 2-input QLUT.

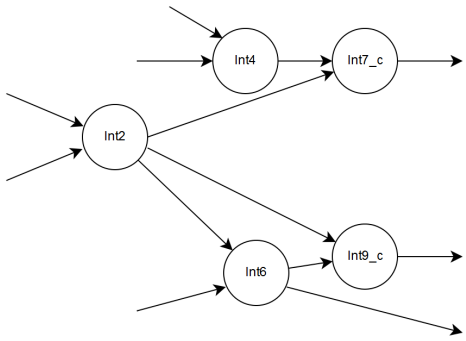
1) Node Refactoring: In this stage, we make use of MVSIS existing functions to manipulate the network. Currently, the function associated with the command `collapse` is being used to attempt collapsing 1-input nodes into its fanouts.

As it can be seen in Figure 5a and Figure 5b, this process, in most cases, avoids using a QLUT partially and thus reduces the total number of necessary QLUTs. In this case the QLUT Q_5 was being used only to transform the signal from Q_2, but this transformation can be performed in conjunction with any operation that comes after, in this case $Q_5 \cup Q_7 \rightarrow Q_{7\_c}$ and $Q_5 \cup Q_9 \rightarrow Q_{9\_c}$ thus reducing the number of QLUTs.

Note that this process does not guarantee that all 1-input nodes are collapsed, e.g. if the 1-input node output is directly connected to a Primary Output (PO) then it cannot be collapsed.



(a) No collapsing.



(b) Collapsing.

Fig. 5: Example of 1-input node.

2) Node Processing: As previously mentioned, we only have available 2-input QLUTs and the nodes that make up the network may have a variable number of input signals, so it is necessary to decompose each node to a compatible representation.

To process each node, information, such as the number of input signals and their identifying names, the node name and the internal functionality, is collected.

With the number of input signals it is determined which strategy to employ to process the node.

1-Input Nodes: The 1-input nodes, that were not able to be collapsed in the previous step, are here mapped into a 2-input QLUT by leaving the second input connected to a ground reference. The QLUT memory is then configured with the node functionality in the first four positions and the remaining twelve are filled with 0.

2-Input Nodes: Nodes that have two input signals require no transformation to be mapped to a 2-input QLUT and the memory configuration is programmed with the node's internal functionality.

3 or More Input Nodes: Nodes with 3 or more inputs have to be decomposed/factorized in order be implemented with the available technology. To do this, an algorithm based on principle of the Shannon decomposition was implemented, as it allows for any possible circuit specifications to be decomposed and mapped.

C. Node Decomposition

The decomposition of the nodes, with 3 or more inputs, is a recursive process, performed in Depth-First Search (DFS), that transforms a node into a tree shaped structure.

Starting from the "root" level, where the output of the node is located, every level behaves as a multiplexer controlled by a distinct input signal and that selects from one of its four "branches".

The last level, "input" level, consists of "leafs" that work the same way as if it were a 2-input node, by using the first two inputs of the node as the QLUT inputs.

1) Selector Block and Shannon Decomposition: To implement the described multiplexer, a "Selector" block, illustrated in Figure 6, was proposed that implements the principle of the Shannon decomposition, which for a quaternary system is given by Equation 1.

$$\begin{aligned} f(A,B,C,\ldots) = C^{(0)} \cdot f(A,B,0,\ldots) + C^{(1)} \cdot f(A,B,1,\ldots) \\ + C^{(2)} \cdot f(A,B,2,\ldots) + C^{(3)} \cdot f(A,B,3,\ldots) \end{aligned} \quad (1)$$

The "Selector" block is comprised of 7 QLUTs organized according to Figure 6b. The QLUTs in the selector perform the function of either a "MUX block" (Q_0, Q_1, Q_2 and Q_3) or "OR block" (Q_4, Q_5 and Q_6).

"MUX blocks" perform the selection, based on the value of C, which means that for a given value of C only the corresponding QLUT would let the input propagate to the output and all the others would put the value zero at their output, i.e., $C = 2 \rightarrow Q_0 = 0, Q_1 = 0, Q_2 = C_2, Q_3 = 0$.

Since only one of the outputs of the previous QLUTs is different from zero at any instance, the "OR blocks" just have to join the signals by performing their sum.

D. Optimization

The method used in the previous step, to perform the decomposition, generates a tree structure with all possible combinations of the inputs, which is not always the optimal solution. To achieve better results some particular cases of circuit decomposition can be explored in order to perform simplifications/optimizations.

To optimize the tree structure, two verifications are performed, a duplicate functionality verification in all "branch" levels and an input dependency verification at the "input" level.

1) Duplicate Functionality: The process to determine if two "branches", from the tree, have duplicated functionality consists in carrying out a search that compares the two, at every level, to ensure that both the structure and the functionality are exactly the same, i.e. both branches produce the same value for the same inputs.

When "branches" are determined to be duplicated, the second is deactivated and redirected to the first. This information is used, in a latter step, to rebuild the final layout of the simplified node.

2) Input Dependencies: At the input level, the verification performed consists in looking for instances where a node does not locally depend on one or both of the input signals.

This verification can return one of the following situations

Standard	$Output = f(A, B)$
fA	$Output = f(A)$
fB	$Output = f(B)$
cte	$Output = constant$

For all situations other than Standard, the node can eventually be collapsed into the output, meaning that one less QLUT is needed.

3) Reconstructing the Node: After having determined all the changes needed, the original "Selector" is adapted to account for these changes. Using the "MUX blocks" and "OR blocks" from the original, this modified version .

The process to obtain the new MVL node with all the required optimizations is described below.

1. Determine how many inputs are active in the end.
   1.1. Group all the **fA** inputs into one "MUX block".
   1.2. Group all the **fB** inputs into one "MUX block".
   1.3. One "MUX block" for each of the still active inputs that are not **fA** or **fB** or **cte**.
   1.4. Group all the **cte** inputs and add them to the first "MUX block".
      1.4.a. If all four inputs are **cte** then a "MUX block" is created similar.
2. Add the required number of "OR block" to match the number of "MUX blocks" created in the previous step.

The selectors, based on the number of inputs of which are dependent, can have one of the configurations illustrated in Figure 7 or in the case of having four input dependencies, a configuration similar to the standard selector from Figure 6b is used.

E. Generate QLUT Programming Data and New Network

After decomposing and optimizing all the nodes, the resulting trees are merged together to form a QLUT network. The QLUT network consists of a list of 2-input nodes, each representing a single QLUT in the final output.

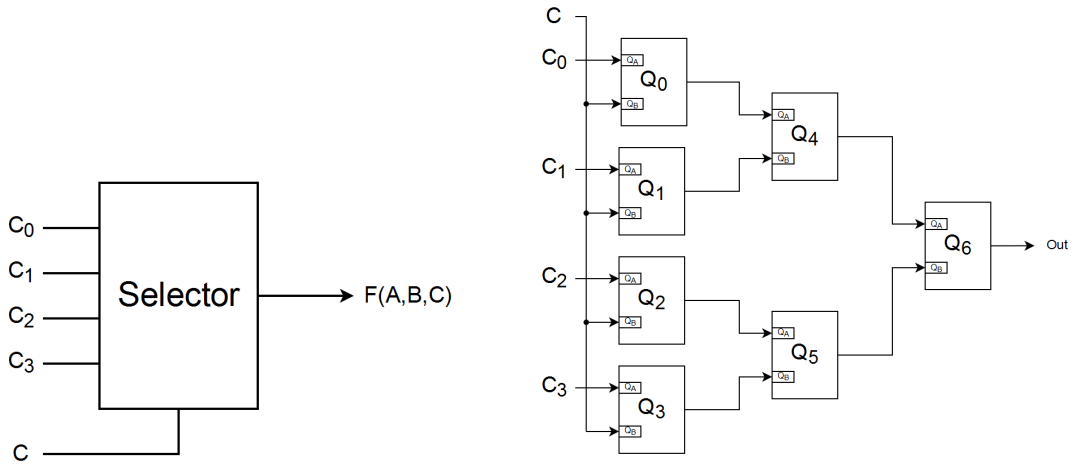
Each element of this structure stores all the informations pertaining to its functionality and relationship to the rest of the elements.

This information is then used to update the network inside the MVSIS framework as well as generating an output file with the programming information of each QLUT.

IV. Results

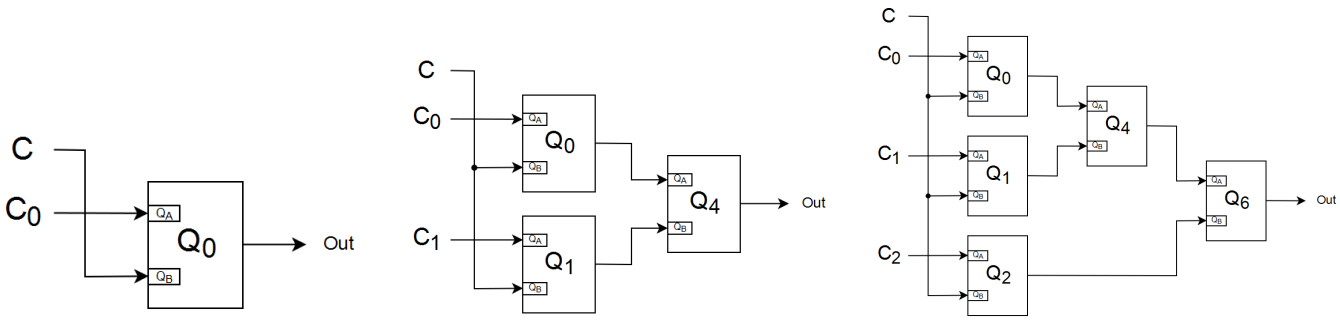
To make an assessment on the accomplishment of the goals originally proposed, a set of tests, consisting of different benchmarks, was performed and the results used in order to evaluate the developed mapping functionality.

For these tests, the mapping technologies used for comparisons are binary 2-input LUTs, binary 4-input LUTs and 2-input QLUTs.



(a) Functionality view. (b) Internal view.

Fig. 6: Selector block.



(a) One "MUX block". (1 QLUT)

(b) Two "MUX blocks". (3 QLUTs)

(c) Three "MUX blocks". (5 QLUTs)

Fig. 7: Modified node selector.

A. *Tests and Results*

The tests performed consist in an adaptation of the Microelectronics Center of North Carolina (MCNC) benchmark suite [26], from where the Finite State Machine (FSM) benchmarks were selected and converted to work with both binary and MVL mapping processes. These benchmarks were selected in order to have a broader set of examples that better represent real specifications.

1) Binary Tests: Due to a limitation in the maximum number of states supported by MVSIS when reading FSM specifications, the benchmark files had to be converted from their original FSM specifications to a binary PLA format in order to perform the binary LUT map testing.

Binary mapping tests are done in a two phase process consisting of synthesis and mapping. In the synthesis phase three different circumstances are tested consisting of:

1. No synthesis
2. Synthesis sequence "script.rugged"
3. Synthesis sequence "script.test"

Here we used two different synthesis sequences since they showed differences in the results for each benchmarks.

Followed by the `fpga` mapping stage, where two target technologies are tested, 2-input LUTs and 4-input LUTs.

The best results for all the scenarios mentioned are presented in Table I, where PI and PO represent the number of inputs and outputs of the circuit and the results in other columns represent the number of binary LUTs required to implement the circuit with that mapping target.

2) Quaternary Tests: For the quaternary tests, the files from the binary tests are converted from PLA format to quaternary encoding. This is accomplished by pairing the signals and converting them into quaternary to create a file in the BLIF-MV format.

By reading the BLIF-MV files the QLUT mapping functionality is tested without the optimizations described in Section III-D and also with the optimizations.

The results for these tests are presented in Table II, where PI and PO represent the number of inputs and outputs of the circuit and the results in remaining columns represent the number of QLUTs required to implement the circuit.

In this table we can examine the reduction, in number of used QLUTs, caused by the optimization process. The most noticeable improvements are in the benchmarks "mc", "s27"

TABLE I: Number of LUTs of binary tests for the FSM benchmarks.

benchmark	PI	PO	LUT2	LUT4
bbara	8	6	75	51
bbtas	5	5	21	8
beecount	6	7	55	32
dk14	6	8	86	57
dk15	6	8	63	42
dk17	6	7	51	32
dk27	4	5	25	5
dk512	5	7	66	31
donfile	7	6	157	104
ex5	6	6	66	39
ex7	6	6	79	54
mc	6	8	20	10
modulo12	5	5	26	8
s27	7	4	13	7
shiftreg	5	5	31	9
tav	7	7	24	15
train11	6	5	71	41

TABLE II: Number of QLUTs of quaternary tests for the FSM benchmarks.

benchmark	PI	PO	QLUTs	
			no optimization	optimization
bbara	4	3	153	88
bbtas	3	3	33	20
beecount	3	4	44	30
dk14	3	4	44	41
dk15	3	4	44	41
dk17	3	4	34	33
dk27	2	3	3	3
dk512	3	4	44	40
donfile	4	3	153	150
ex5	3	3	33	25
ex7	3	3	33	27
mc	3	4	14	6
modulo12	3	3	23	15
s27	4	2	102	33
shiftreg	3	3	23	23
tav	4	4	104	18
train11	3	3	33	24

and "tav" with reductions of 57.14%, 67.65% and 82.69%, respectively.

B. Result Analysis

To be able to make comparisons between the different implementation technologies some comparable metrics have to be established. The chosen metrics are the occupied area, based on the amount of transistors on each LUT, and the number of interconnections used in the circuit to route the signals.

According to [5], the proposed QLUT technology is composed of a total of 202 transistors. And although a binary LUTs can be implemented in different manners, an approximation is made, for comparison purposes, by considering the internal construction to be similar to the one employed in the QLUTs. With this approximation, a total of 60 and 296 transistors are used in the calculations for the 2-input and 4-input variants, respectively.

Note that, although this is not the most efficient means of implementing binary LUTs in terms of number of used transistors, it is a reasonable approximation to the internal structure of the proposed QLUT.

In Table III, the results from Tables I and II are put together, which are used to calculate the circuit area and number of interconnections, columns "area" and "wires". Equation 2 is used to determine the number of interconnections in the circuit.

$$\#wires = \#\text{LUT} \times \#\text{LUT}inputs + \#\text{PO} \qquad (2)$$

These values are then compiled into Table IV, where the QLUT implementation is analyzed against the binary for improvements/gains in number of used LUTs, circuit area and internal signals/interconnections to route, additionally, some statistics are also presented. In the table, a number greater than 1 in the LUTs columns or positive in the area and wires columns indicate improvements in the quaternary implementation over the binary implementations.

The statistics are here presented to give a more comprehensive view of the results and consist of the average, mean and trimmed mean with 20% margin (10% ceiling and 10% floor). The trimmed mean is here used in an attempt to remove the effect of discrepant results.

From Table IV the main observation that can be done is that, a few outliers do not perform well with the current MVL mapping implementation and as such the trimmed mean is going to be used when making comparisons. But overall the number of required QLUTs is approximately halved when compared to 2-input LUTs. However, as expected, is approximately the same when compared to 4-input LUTs. This is a positive result if we consider also the number of interconnections that shows an improvement of 37.52% and 38.50% when compared to 2-input and 4-input, respectively.

The number of transistors/circuit area has a contrary result to the number of LUTs and interconnections. In these tests, the area occupied by the quaternary circuit is 111.76% more than with binary 2-input LUTs and 15.02% less when compared to 4-input binary circuit.

Some conclusions should be drawn from these results, which are that while particular circuits may require other mapping algorithms and/or optimizations to be efficiently mapped into quaternary logic, e.g., "s27", others show great improvements in all metrics tested, e.g., "dk27".

V. CONCLUSIONS

In the introduction of this report, a case was made for the need for a mapping tool capable of implementing circuit specifications in MVL using the newly developed Quaternary Lookup Tables (QLUTs) from [5]. With this objective in mind, a MVL mapping functionality was added to the MVSIS tool.

The MVL mapping tool proposed and developed can be used to produce any desired circuit layouts using Quaternary Lookup Tables (QLUTs). Since it was developed inside MVSIS, it accepts any format of input that MVSIS can read and with it generate a quaternary network.

This work main problem to overcome was how to convert a node inside a network into memory configurations to program

TABLE III: Results and characteristics of the FSM benchmarks.

benchmark	binary								quaternary				
	PI	PO	LUT2	area	wires	LUT4	area	wires	PI	PO	QLUTs	area	wires
bbara	8	6	75	4500	156	51	15096	210	4	3	88	17776	179
bbtas	5	5	21	1260	47	8	2368	37	3	3	20	4040	43
beecount	6	7	55	3300	117	32	9472	135	3	4	30	6060	64
dk14	6	8	86	5160	180	57	16872	236	3	4	41	8282	86
dk15	6	8	63	3780	134	42	12432	176	3	4	41	8282	86
dk17	6	7	51	3060	109	32	9472	135	3	4	33	6666	70
dk27	4	5	25	1500	55	5	1480	25	2	3	3	606	9
dk512	5	7	66	3960	139	31	9176	131	3	4	40	8080	84
donfile	7	6	157	9420	320	104	30784	422	4	3	150	30300	303
ex5	6	6	66	3960	138	39	11544	162	3	3	25	5050	53
ex7	6	6	79	4740	164	54	15984	222	3	3	27	5454	57
mc	6	8	20	1200	48	10	2960	48	3	4	6	1212	16
modulo12	5	5	26	1560	57	8	2368	37	3	3	15	3030	33
s27	7	4	13	780	30	7	2072	32	4	2	33	6666	68
shiftreg	5	5	31	1860	67	9	2664	41	3	3	23	4646	49
tav	7	7	24	1440	55	15	4440	67	4	4	18	3636	40
train11	6	5	71	4260	147	41	12136	169	3	3	24	4848	51

TABLE IV: Gains comparison of binary and quaternary circuits and statistics.

benchmark	LUT2			LUT4		
	LUTs	area	wires	LUTs	area	wires
bbara	0.8523	-2.9502	-0.1474	0.5795	-0.1775	0.1476
bbtas	1.0500	-2.2063	0.0851	0.4000	-0.7061	-0.1622
beecount	1.8333	-0.8364	0.4530	1.0667	0.3602	0.5259
dk14	2.0976	-0.6050	0.5222	1.3902	0.5091	0.6356
dk15	1.5366	-1.1910	0.3582	1.0244	0.3338	0.5114
dk17	1.5455	-1.1784	0.3578	0.9697	0.2962	0.4815
dk27	8.3333	0.5960	0.8364	1.6667	0.5905	0.6400
dk512	1.6500	-1.0404	0.3957	0.7750	0.1194	0.3588
donfile	1.0467	-2.2166	0.0531	0.6933	0.0157	0.2820
ex5	2.6400	-0.2753	0.6159	1.5600	0.5625	0.6728
ex7	2.9259	-0.1506	0.6524	2.0000	0.6588	0.7432
mc	3.3333	-0.0100	0.6667	1.6667	0.5905	0.6667
modulo12	1.7333	-0.9423	0.4211	0.5333	-0.2796	0.1081
s27	0.3939	-7.5462	-1.2667	0.2121	-2.2172	-1.1250
shiftreg	1.3478	-1.4978	0.2687	0.3913	-0.7440	-0.1951
tav	1.3333	-1.5250	0.2727	0.8333	0.1811	0.4030
train11	2.9583	-0.1380	0.6531	1.7083	0.6005	0.6982
average	2.1536	-1.3949	0.3058	1.0277	0.0408	0.3172
mean	1.6500	-1.0404	0.3957	0.9697	0.2962	0.4815
trimmed mean	1.8589	-1.1176	0.3752	1.0172	0.1502	0.3850

QLUTs with. To accomplish this, nodes have to conform to the restrictions of the available technology. In this case any node whose output value depended of more than 2 input signals have to be decomposed.

The decomposition was done using an algorithm based on the Shannon decomposition and implemented with a "Selector" block, described in Section III-C1,that performs the role of a multiplexer.

After having the nodes decomposed a new network representation is created of the new decomposed nodes and these nodes are converted into QLUT memory configurations.

The mapping functionality was tested with a set of benchmark circuits and the results compared against binary circuit implementations. Although our mapping implementation did not provided better results in all the benchmarks, a few of them showed improvements over the binary tests.

With improvements in both the hardware and software side, it is possible in the future to have this synthesis tool capable of generating MVL circuits that are smaller, faster and more energy efficient than their binary equivalents.

The overall result of the developed work is positive as it resulted in a functioning mapping tool capable of being used in a variety of scenarios to generate a circuit using QLUT technology.

Nonetheless, various enhancements can still be made in order to improve the obtained results and expand the current functionality. Bellow are a few ideas worth of exploring that can produce improvements to the results here presented. Most of these ideas consist of suggestions based on adversities encountered during the development of this tool.

- Improvement of current implementation of the QLUT since currently they use considerable more area than their binary equivalents.
- Development of a 3-input QLUTs to combat a major difficulty encountered during the development process of this tool. Having a limitation of only 2-input QLUTs

being available led to the creation of a "Selector" block, which is an inefficient and not very scalable solution.

- Process the network nodes by having a different order in the "Selector" levels. Test which permutation of the inputs order produces the most optimized result.
- Expand the current optimization process to account for multi-output circuits that may share duplicate functionality across nodes, thus creating shared fanin cones and reducing significantly the amount of QLUTs required.
- Explore mapping algorithms different from the Shannon decomposition. Although the Shannon decomposition allows to process any node, it always produces non-optimal solution, in terms of number of required QLUTs, which then has to be optimized for each particular case. If an algorithm was used such that the decomposition is done considering the function of the node instead of a general case, a custom tailored result would be obtained without having the need an optimization process.
- Explore the possibility of expanding the type of circuits that can be mapped into QLUTs by converting non-quaternary networks into quaternary form.
- Development of a power simulator to analyze the QLUTs. In the results presented, in only a few benchmarks, the circuit area was reduced by using a QLUT implementation. However QLUTs can provide improvements in power dissipation due to the smaller transitions between logic levels.

REFERENCES

[1] T.-S. Jung, Y.-J. Choi, K.-D. Suh, B.-H. Suh, J.-K. Kim, Y.-H. Lim, Y.-N. Koh, J.-W. Park, K.-J. Lee, J.-H. Park *et al.*, "A 117-mm 2 3.3-V only 128-Mb multilevel NAND flash memory for mass storage applications," *IEEE Journal of Solid-State Circuits*, vol. 31, no. 11, pp. 1575–1583, 1996.

[2] A. F. González and P. Mazumder, "Multiple-valued signed digit adder using negative differential resistance devices," *IEEE Transactions on Computers*, vol. 47, no. 9, pp. 947–959, 1998.

[3] T. Hanyu and M. Kameyama, "A 200 MHz pipelined multiplier using 1.5 v-supply multiple-valued MOS current-mode circuits with dual-rail source-coupled logic," *IEEE Journal of Solid-State Circuits*, vol. 30, no. 11, pp. 1239–1245, 1995.

[4] Z. Zilic and Z. G. Vranesic, "Multiple-valued logic in fpgas," in *Midwest Symposium on Circuits and Systems*, vol. 36, 1993, pp. 1553–1553.

[5] D. R. O. de Brito, T. Rabuske, J. M. dos Santos Ribeiro Fernandes, P. Flores, and J. Monteiro, "Quaternary Logic Lookup Table in Standard CMOS," *IEEE Transactions on Very Large Scale Integration Systems (TVLSI)*, vol. 23, no. 2, pp. 306–316.

[6] S. Gottwald, "Many-Valued Logic," in *The Stanford Encyclopedia of Philosophy*, spring 2015 ed., E. N. Zalta, Ed., 2015.

[7] Z. G. Vranesic and K. C. Smith, "Engineering aspects of multi-valued logic systems," *Computer*, vol. 7, no. 9, pp. 34–41, 1974.

[8] K. C. Smith, "The prospects for multivalued logic: A technology and applications view," *Computers, IEEE Transactions on*, vol. 100, no. 9, pp. 619–634, 1981.

[9] S. L. Hurst, "Multiple-Valued Logic its Status and its Future," *Computers, IEEE Transactions on*, vol. 100, no. 12, pp. 1160–1179, 1984.

[10] Intel StrataFlash™ Memory Technology. [Online]. Available: http://www.intel.com/content/dam/www/public/us/en/documents/research/1997-vol01-iss-4-intel-technology-journal.pdf

[11] Samsung SSD 840 Series - 3BIT/MLC NAND Flash. [Online]. Available: http://www.samsung.com/global/business/semiconductor/minisite/SSD/uk/html/about/MlcNandFlash.html

[12] C. M. Allen and D. D. Givone, "A minimization technique for multiple-valued logic systems," *IEEE Transactions on Computers*, no. 2, pp. 182–184, 1968.

[13] D. D. Givone and R. W. Snelsire, *Final report on the design of multiple-valued logic systems*. Digital Systems Laboratory of the Department of Electrical Engineering, State University of New York at Buffalo, 1968.

[14] J. Santos, H. Arango, M. Pascual, and G. Roing, "A cyclic algebra for the synthesis of ternary digital systems," *IEEE Transactions on Computers*, no. 7, pp. 651–653, 1970.

[15] R. K. Brayton and S. P. Khatri, "Multi-valued logic synthesis," in *International Conference on VLSI Design*. IEEE, 1999, p. 196.

[16] L. Scheffer, L. Lavagno, and G. Martin, Eds., *EDA for IC Implementation, Circuit Design, and Process Technology*. CRC Press, 2006, vol. 2.

[17] G. Hachtel, M. Lightner, K. Bartlett, D. Bostwick, R. Jacoby, P. Moceyunas, C. Morrison, X. Du, and E. Schwarz, "Bold: The boulder optimal logic design system," in *System Sciences, 1989. Vol.I: Architecture Track, Proceedings of the Twenty-Second Annual Hawaii International Conference on*, vol. 1, Jan 1989, pp. 59–73 vol.1.

[18] E. M. Sentovich, K. J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. R. Stephan, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Sis: A system for sequential circuit analysis," Tech. Report No. UCB/ERL M92, Tech. Rep., 1992.

[19] "VIS (Verification Interacting with Synthesis)," http://vlsi.colorado.edu/~vis/.

[20] J. Cong, J. Peck, and Y. Ding, "Rasp: A general logic synthesis system for sram-based fpgas," in *Field-Programmable Gate Arrays, 1996. FPGA '96. Proceedings of the 1996 ACM Fourth International Symposium on*, 1996, pp. 137–143.

[21] D. Chai, J.-H. Jiang, Y. Jiang, Y. Li, A. Mishchenko, and R. Brayton, "MVSIS 2.0 user's manual," *Department of Electrical Engineering and Computer Sciences*, 2003.

[22] M. Gao, J.-H. Jiang, Y. Jiang, Y. Li, A. Mishchenko, S. Sinha, T. Villa, and R. Brayton, "Optimization of multi-valued multi-level networks," in *Multiple-Valued Logic, 2002. ISMVL 2002. Proceedings 32nd IEEE International Symposium on*, 2002, pp. 168–177.

[23] Berkeley Logic Synthesis and Verification Group, "ABC: A System for Sequential Synthesis and Verification," http://www.eecs.berkeley.edu/~alanmi/abc/.

[24] R. K. Brayton, M. Chiodo, R. Hojati, T. Kam, K. Kodandapani, R. Kurshan, S. Malik, A. L. Sangiovanni-Vincentelli, E. Sentovich, T. Shiple, K. Singh, and H. Wang, "BLIF-MV: An Interchange Format for Design Verification and Synthesis," EECS Department, University of California, Berkeley, Tech. Rep. UCB/ERL M91/97, 1991.

[25] Y. Kukimoto, "BLIF-MV," *The VIS Group, University California, Berkely, May*, 1996.

[26] S. Yang, "Logic synthesis and optimization benchmarks user guide: Version 3.0," Tech. Rep., 1991.