Radix-p Multiple Valued Logic Function Simplification using Higher Radix Representation

Zaid Al-Wardi

1Collage of Engineering, Mustansiriyah University, Baghdad, Iraq.
alwardi@uomunstansiriyah.edu.iq

Abstract. Conventional Boolean logic, which is based on radix-2 binary logic, is no longer suitable to represent complex digital systems. As digital systems continue to grow in complexity, more compact representations become essential to describe these systems. One such possibility is to use multiple-valued logic, with radices larger than two. Many non-conventional technologies are considered as candidates to replace conventional semiconductor based switching circuit technology for computational systems such. Interestingly, some of these technologies are inherently multiple valued. Multiple-valued logic design and optimization methodologies are not yet elaborated, as compared to the well-established binary logic design and optimization and testing scheme. In this paper we are proposing an algorithm, with which functions to describe digital systems are specified and simplified using Multiple valued logic. Representing a digital system with different radices come out with different equivalent specifications for that system. A designer can choose the most suitable specification for his design constrains. The influence of radix increase on the complexity of the algorithm and the resulting specification is also demonstrated and discussed in this paper.

1. Introduction
Binary logic dominates the digital systems since the beginning, because realizing two-state switching circuits is feasible. However, digital systems are exponentially growing and more compact representations and technologies become essential [1]. When signals in a circuit represents more levels (i.e. more information), interconnections on- and off-chips are reduced.

Multiple-valued logic (MVL) generalizes Binary logic and describes digital information in more compact representations [2]. MVL has many applications in both hardware and software, where MVL hardware applications enhance both time and space requirements of the circuits, such as reducing the ripple-through carry in arithmetic circuits, resulting in high-speed arithmetic operations [3]. With MVL a memory cell stores more than one bit of information, which increases the density of information per area unit [4]. However, the potential advantages and feasibility of MVL hardware depend heavily on the availability of technologies, such as promising optical, DNA and quantum computers technologies, which are multiple-valued by their nature [5]. In software there is no need for special technology to implement MVL applications, where most commercial and academic problems for image processing, data mining, machine learning, as well as digital system design, testing and verification are MVL problems [6][7][8]. MVL is used as mathematical notation in logic synthesis programs, which allows digital function minimization more efficiently [9] [10] [11].

This paper exploits higher radix logic to specify MVL functions and to minimize these functions in a straight-forward iterative algorithm. The proposed algorithm is applicable on any radix system, which can have a good impact on the design flow of digital systems.
2. Preliminaries
This section is meant to keep the paper self-contained by introducing MVL functions, as well as the main principles of discrete function minimization, where an arbitrary combinational Boolean function \( f \) over \( n \) binary inputs \((x_{n-1}, \ldots, x_0)\) maps it into \( m \) binary outputs \((f_{m-1}, \ldots, f_0)\). Boolean operators are defined over the binary set \{0, 1\}. A straightforward tabular specification of \( f \) can be achieved by enumeration its values at the entire domain points of \( f \), and called a truth-table. An input permutation in a row of a truth-table is called a minterm, which is indexed by the weighted sum of its bits \( \sum_{i=0}^{n-1} x_i \cdot 2^i \) [12], e.g. the function in Table 1(a) specifies \( f(5) = 111 \), where 5 is the index for the minterm \( i = \sum_{i=0}^{n-1} x_i \cdot 2^i \). MVL is a generalization of Binary logic to higher radices, e.g. radix-3 system is called ternary with the set \{0, 1, 2\}, and to radix-4 is called quaternary with the set \{0, 1, 2, 3\}, etc. In general, a radix-\( p \) system represents its digits using the set \{0, 1…, (p-1)\}. In this case we can see two quaternary digits representing four bits of information. This makes the basic unit used in this system more informative and causes the complexity reduction in space, time and interconnections, as mentioned in section 1. With a generalized function minimization scheme it becomes be possible to compare between different MVL systems in term of efficiency.

3. Proposed Algorithm
Unlike minterms that are indexed by their weighted sum of digits, function cubes cannot be numerically indexed because of the character '-' in the place of each input that is insignificant in these minterms. A table that specifies a function with its cubes is called a PLA table [12], see Table 1(b), in which the cube \((1--)\) replaces four minterms \{100,101,110,101\} of Table 1(a).

Table 1. Truth-table and PLA Table Representations of a Function.

| (a) | Binary Truth table | (b) | PLA table |
|-----|-------------------|-----|-----------|
| \(i\) | \(x_2\) | \(x_1\) | \(x_0\) | \(f_2\) | \(f_1\) | \(f_0\) | \(i\) | \(v_1\) | \(v_0\) | \(g_i\) | \(g_0\) |
| 0   | 0    | 0    | 0       | 0     | 0     | 0     | 0   | 0    | 0    | 0     |
| 1   | 0    | 0    | 1       | 0     | 0     | 1     | 1   | 0    | 0    | 1     |
| 2   | 0    | 1    | 0       | 0     | 1     | 0     | 1   | 0    | 1    | 0     |
| 3   | 0    | 1    | 1       | 0     | 0     | 1     | 1   | 0    | 1    | 1     |
| 4   | 1    | 0    | 0       | 1     | 1     | 0     | 1   | 0    | 2    | 1     |
| 5   | 1    | 0    | 1       | 1     | 1     | 1     | 2   | 1    | 1    | 2     |
| 6   | 1    | 1    | 0       | 1     | 0     | 1     | 3   | 1    | 2    | 1     |
| 7   | 1    | 1    | 1       | 0     | 0     | 1     | 3   | 1    | 3    | 1     |

Based on the properties of Boolean logic, a group of minterms can be replaced in the function by one term, called a cube, which contains a '-' in the place of each input that is insignificant in these minterms. A table that specifies a function with its cubes is called a PLA table [12], see Table 1(b), in which the cube \((1--)\) replaces four minterms \{100,101,110,101\} of Table 1(a).

MVL is a generalization of Binary logic to higher radices, e.g. radix-3 system is called ternary with the set \{0, 1, 2\}, and to radix-4 is called quaternary with the set \{0, 1, 2, 3\}, etc. In general, a radix-\( p \) system represents its digits using the set \{0, 1…, (p-1)\}. Higher radix means less digits to represent binary information, and computed from Eq.

\[
d = \left[\log_p(2^n)\right],
\]

where \( p \) is the radix of the MVL system \( d \) is the number of MVL digits and \( n \) is the number of bits, e.g. two bits are represented using one quaternary digit. The index in a radix-\( p \) minterms is computed form \( i = \sum_{k=0}^{n-1} v_k \cdot p^k \) [2]. MVL truth tables are translations of Boolean truth tables, e.g. Table 1(c) represents the same information of Table 1(a) quaternary coded. In this case we can see two quaternary digits representing four bits of information. This makes the basic unit used in this system more informative and causes the complexity reduction in space, time and interconnections, as mentioned in section 1. With a generalized function minimization scheme it becomes be possible to compare between different MVL systems in term of efficiency.
the other hand, cube 1-1 has the index j computed with the digit ‘-’ replaced by 2, i.e. j = (121)\(_3\) = (1×3^2+2×3^1+1×3^0) = 16. No minterm index can be computed for this cube, because it is not a minterm.

This cube indexing motivates a simple minimization algorithm of radix-(q+1) cubes to a radix-q MVL function. The idea is to compute the function covered with cube j that contains a digit ‘-’ as following: \(f(j) = \text{minimum} \{f(j-w.k)\}; \forall k \in \{1, \ldots, q\}\), where w is the weight of the ‘-’ digit. In other words \(f(j)\) is the minimum function value of any minterm covered by the cube j. For simplicity of the algorithm in this section considers single output functions. The same algorithm is applicable with some extra details when implemented for multiple-output functions, e.g. the minimum is picked for each output alone.

**Table 2.** Cube Minimization of the Function Specified by the Truth-table in Table 1(a).

| Cube | Step 1 | Step 2 |
|------|--------|--------|
| 0    | 0 0 0 0| 0 0 0 0|
| 1    | 0 1 0 1| 1 0 0 0|
| 2    | 0 0 - 0| 0 0 0 0|
| 3    | 0 1 0 1| 1 0 0 0|
| 4    | 0 1 - 0| 0 0 0 0|
| 5    | 0 1 - 0| 0 0 0 0|
| 6    | 0 0 - 0| 0 0 0 0|
| 7    | 0 - - 0| 0 0 0 0|

Example 2: In Table 3, a quaternary system radix-4, the value of the function at cube (-0). In this case the cube computation is j=(-0)(4+1) = (4×51+0×50) = 20, is computed simply from:

\[ f(20) = \text{minimum} \{f(15), f(10), f(5), f(0)\} \]

After iterative computation of the function value that covers all possible cubes in the function domain, the minimization is carried out within the same iteration. The iteration starts from cube j=0 until the universal cube j=(q+1)d-1. The minimization is performed by resetting to 0 any cube that is already covered by a larger cube with the same value in the function, after computing the value of the function at cube j, by assigning 0 to all minterms and sub-cubes covered by \(f(j)\) using the relation:

\[f(j-w.k) = f(j)\]  if  \(f(j-w.k) = f(j)\)

**Table 3.** Cube Minimization of the Function Specified by the Truth-table in Table 1(c).

| Cube | Step 1 | Step 2 |
|------|--------|--------|
| 0    | 0 0 0 0| 0 0 0 0|
| 1    | 1 0 1 0| 1 0 0 0|
| 2    | 0 2 0 3| 0 3 0 3|
| 3    | 3 0 2 0| 0 2 0 2|
| 4    | 4 0 - 0| 0 0 0 0|
| 5    | 5 1 0 1| 1 0 0 0|
| 6    | 6 1 3 1| 3 1 0 3|
| 7    | 7 2 1 1| 1 1 0 0|
| 8    | 8 3 1 0| 1 0 0 0|

Example 3: Table 2 shows the cube computation of the function in Table 1(a). The left most two columns show the transform of the minterm index i to the corresponding cube index j. The cube inputs’ values are shown in the third column. The computation of the cubes is shown in the column labeled (Step 1). In this table the function is binary (radix-2), then the function of each non minterm cube is computed from the minimum (bitwise AND) of its forming two sub-cubes, \(f(j)=f(j-w) & f(j-2w)\), e.g. \(f(11)\) contains a ‘-’ in the rightmost bit with w=1, then \(f(11)= f(11-1) & f(11-2) = f(10) & f(9) = 111 & 110 = 110\).

The minimization is carried out by resetting bits to 0, as shown in (Step 2). The iteration of computation and minimization result specifies the function using the five underlined cubes, similar to the PLA table of Table 1(b). The cubes computation through this array takes 27 iterations. Table 3, on the other hand, shows the cube computation of the function in Table 1(c), which is the same function above but coded in radix-4 system. In this case the algorithm specifies the function using seven cubes, but with less digits. In this case the cubes computation takes 25 iterations.
The difference in the number of cubes and the algorithm computational complexity seems insignificant, but this example considers a small function with only three-bit input. Larger functions show larger difference as shown in the next section.

Table 4. Time and Space Complexity Expansion of the Proposed Algorithm.

| i/p bits | Memory usage |
|----------|--------------|
|          | Binary $p=2$ | Ternary $p=3$ | Quaternary $p=4$ | Octal $p=8$ | Decimal $p=10$ | Hexadecimal $p=16$ |
| 1        | 3            | 4             | 5             | 9             | 11            | 17               |
| 2        | 9            | 16            | 5             | 9             | 11            | 17               |
| 3        | 27           | 16            | 25            | 9             | 11            | 17               |
| 4        | 81           | 64            | 25            | 81            | 121           | 17, 289          |
| 5        | 243          | 256           | 125           | 81            | 121           | 289              |
| 6        | 729          | 4096          | 625           | 729           | 1331          | 289              |
| 7        | 2187         | 1024          | 125           | 81            | 121           | 289              |
| 8        | 6561         | 4096          | 625           | 729           | 1331          | 289              |
| 10       | 59049        | 16384         | 3125          | 6561          | 14641         | 4913             |
| 16       | 43046721     | 4194304       | 390625        | 531441        | 161051        | 83521            |
| 20       | 3.49E+09     | 67108864      | 9765625       | 4782969       | 19487171      | 1419857          |
| 32       | 1.85E+15     | 4.39E+12      | 1.53E+11      | 3.14E+10      | 2.59E+10      | 6.98E+09         |

4. Discussion

Metrics to measure the cost, performance and complexity of MVL digital systems depends highly on the technology used in implementing the system. However, the number of cubes in a PLA table may be used as rough indicators to the complexity of a certain function specification. The example function representation and minimization, as demonstrated in the previous section, shows two scenarios of specifying a digital system, based on binary and on quaternary logic. The lower radix (binary) could reach the function specification with five cubes, less than the quaternary system. On the other hand, the advantage of applying this algorithm is shown for large functions where the computation complexity shows tangible reduction in both memory usage and also the iteration time required to compute and minimize the cubes of the required function, as shown in the Table 4 below. The memory usage depends on the total number of cubes which is computed from

\[ \text{number of cubes} = (p + 1)^{\lceil \log_p(2^n) \rceil}, \]

The first few rows show disadvantages of higher-radix systems. This is gradually flipped to the opposite as the number of inputs increases, where the high radix becomes an advantage. The last row in this table shows that a in a 32-bit input system the memory usage and also the computational complexity based on ternary system is only 0.2373% as compared to the binary equivalent and reaches in the hexadecimal based system less than 0.0005%. This can make a huge difference in the system design flow where computational complexity largely restricts many design methodologies.

5. Conclusions

The algorithm proposed in this paper is based on radix-p+1 indexing of radix-p cubes. The idea of exploiting higher radix makes it possible to represent cubes numerically and to iteratively compute the covers of these cubes using a simple computation. The algorithm shows the potential to achieve different specifications of a digital system with more compact notations using multiple-valued logic. Computation complexity when solving the problem for high radices compromises the minimal solution achieved by binary system especially for large design problems.

In the future, we recommend further optimization by using recursive rather than iterative based computation. This can highly reduce the memory usage, with expected expansion in computational time. Another modification that can reduce computational complexity is by using heuristic search techniques in cube computation.
Acknowledgements The activity presented in the paper is part of the approved research plan of Electrical Engineering Department, Collage of Engineering, Mustansiriyah University.

References
[1] Moore, G. E. (1965). Cramming More Components onto Integrated Circuits, Electronics, vol. 38, no. 8, 114-117.
[2] Stancovic, R. S., Astola, J. T., and C. Moraga, (2012). Representation of Multiple-Valued Logic Functions. Morgan & Claypool Publishers.
[3] Meher, P. K., Valls, J., Juang, T., Sridharan K., and Maharatna K. (2009). 50 Years of Cordic: Algorithms, Architectures, and Applications. IEEE Transactions on Circuits and Systems: Regular Papers, vol. 56, 1893-1907.
[4] Gong, G., Zhou, H., Li, L., Tao, J., and Zeng, X. (2011). Binning Optimization for Transparently-latched Circuits, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 30, 270-283.
[5] Miller, D. M., and Thornton, M. A. (2008). Multiple Valued Logic: Concepts and Representations. Morgan & Claypool Publishers.
[6] De Micheli, G., Brayton, R. K., and Sangiovanni-Vincentelli, A. (1985). Optimal State Assignment for Finite State Machines. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 4, no. 3, 269-285.
[7] Dubrova, E., and Sack, H. (2000). Probabilistic verification of multiple-valued functions. Proceedings 30th IEEE International Symposium on Multiple-Valued Logic, Portland, Oregon, USA.
[8] Abramovici, M., Melvin, A., and Friedman, A. D. (1994). Digital Systems Testing and Testable Design. Computer Science Press, New York.
[9] Rudell, R. L., and Sangiovanni-Vincentelli, A. (1987). Multiple-Valued Minimization for PLA Optimization. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 6, no. 5, 727-750.
[10] Dubrova, E. (2002). Multiple-valued Logic Synthesis and Optimization, in Logic Synthesis and Verification, Kluwer Academic Publishers, Norwell, MA, USA, 89-114.
[11] Sasao, T. (1988) Multiple-valued logic and optimization of programmable logic arrays. Computer, vol. 21, no. 4, 71-80.
[12] Mano, M. M., Kime, C. R., Martin, T. (2015) Logic and Computer Design Fundamentals. 5th ed., Pearson.