Ordered Binary Decision Diagrams in Circuit Design

Henry Fernandes

Supervisor: Carl McCrosky

Abstract

Many techniques can be applied when designing circuits. Each has advantages and disadvantages, and there are different situations for which a designer might wish to utilise a certain technique. This paper explores the suitability of a relatively new design method, Ordered Binary Decision Diagrams (OBDDs), to two more established methods, PLAs and random logic. The paper begins with a brief introduction to the concept of OBDDs and then continues on with a comparison of various designs using the three different techniques.

1 Introduction

Two goals which every circuit designer strives toward are low propogation delay and reduced area. The designer who is able to satisfy both of these goals is likely to satisfy the demands of the consumer. However, achievement of both small area and short delay is not always a realistic expectation. Oftentimes, one is forced to compromise one aim in the hopes of bettering the other.

A designer with access to place and route tools might apply random logic to implement a circuit. Unfortunately, doing so generally requires some labour and consequently designers will often turn to other methods such as programmable logic. PLAs are a good example of programmable logic. They have regular structures which are easily generated and implemented, are quite fast, and do not make very good use of silicon area. The purpose of this paper is to discover whether or not Ordered Binary Decision Diagrams (OBDDs) would be a good addition to the realm of programmable logic.

To the present, the role of OBDDs has been limited to merely representation, as finding a desirable hardware mapping has proven difficult. [5] suggested an implementation based upon multiplexors, but it has many problems. Among these problems are a long propogation delay and power dissipation. As it stands, the problems associated with this implementation are restrictive enough that it is not a suitable method for design.

The implementation of OBDDs used in this paper was originated by Andrew Hughes, Michael Kolodziejczyk, Jerry Ho, and Henry Fernandes. It was subsequently mutated by Dr. Carl McCrosky to its present form. It is similar to Matos and Oldfield's multiplexor design except that it employs NP dominoe (Zipper) logic to help cure the problems of speed and power dissipation. This method complicates the design procedure somewhat, but the design procedure is still extremely regular and thus can be generated automatically. This additional complication is invisible to the designer.

2 Background on OBDDs

A binary decision diagram (BDD) is a DAG (directed acyclic graph) which is used to form representations of functions. A truth table and its BDD are shown in Table 1 and Figure 1 respectively. Each node in the BDD corresponds to a variable and the path that is followed along a BDD corresponds to

Table 1: Truth table

x1	x2	x3	f
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

the decision made at each node according to the value of that variable. In Figure 1, all dashed lines correspond to a 0 value of the variable while all solid lines correspond to a 1 value of the variable.

An OBDD (Ordered BDD) is a BDD with certain changes applied. First of all, as the name implies, the OBDD is *ordered*. That is, the variables and thus nodes are ordered in some fashion, as is Figure 1, $x1 < x2 < x3$. In addition to this, OBDDs have the following three transformation rules [2] applied:

1. Remove duplicate terminal vertices
2. Remove duplicate non-terminal vertices
3. Remove redundant vertices

The effects of applying such rules are shown in Figure 2. One can see that the complexity has been reduced from Figure 1, which has 15 nodes, to Figure 2(c), which has 5 nodes.

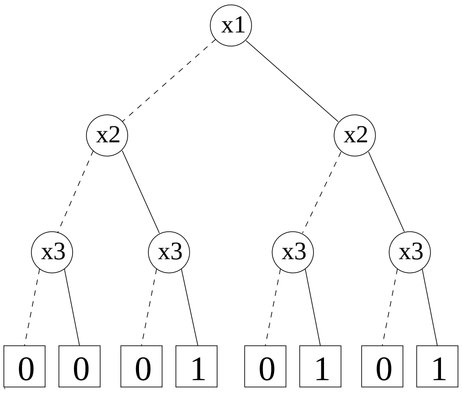


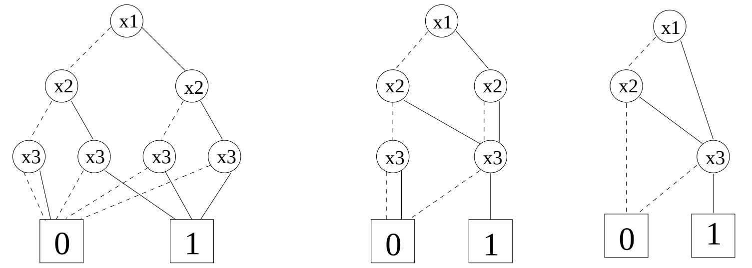
Figure 1: OBDD corresponding to above truth table

The size of an OBDD may largely depend on ordering [2]. For example, Figure 3, shows two different OBDDs that can represent the function

$$f = x1 \cdot y1 + x2 \cdot y2 + x3 \cdot y3.$$

In the OBDD on the left, the ordering used is $x1, y1, x2, y2, x3, y3$. This gives a compact and minimal OBDD. The ordering on the right, however, is not so attractive. It uses the ordering $x1, x2, x3, y1, y2, y3$. Obtaining minimal orderings is an NP-hard problem [5]. Table 2 shows the best and worst case complexity for some OBDDs [2].

Though the variable ordering can have a dramatic effect on OBDD complexity, not all functions are so sensitive. Most, in fact, have exponential complexity with at least $2^n/2n$ nodes [4]. However, many functions still do generate OBDDs of attractive size. Unfortunately, this class of well-behaved functions has not as yet been identified.

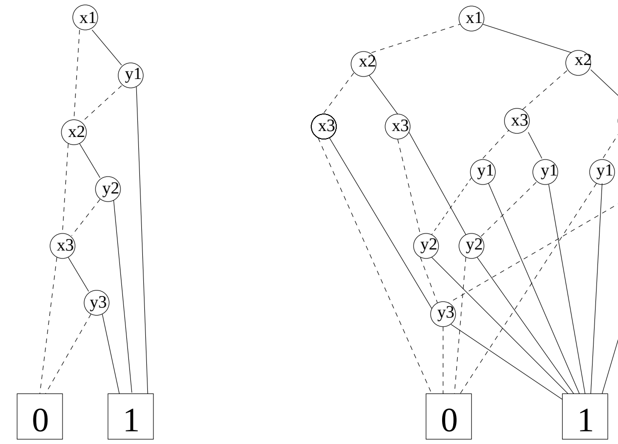


A) **Remove duplicate terminals**

B) **Remove duplicate non-terminals**

C) **Remove redundancies**

Figure 2: Changes after applying the three rules



ORDERING

x1, y1, x2, y2, x3, y3

ORDERING

x1, x2, x3, y1, y2, y3

Figure 3: Different variable orderings for identical functions

Therefore, one cannot characterize or predict the type of functions that will generate attractive OBDDs. Often, it is necessary to actually determine an OBDD layout in order to determine its complexity.

3 Implementation

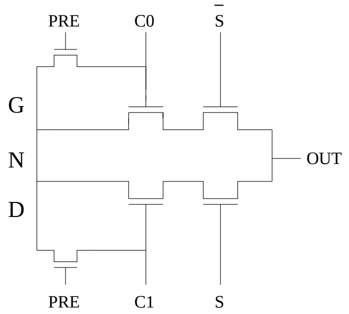
The results obtained in such a project as this are greatly affected by the design of the OBDD. Thus, care was taken to design a cell which would be fast yet small. The cell eventually arrived at is similar, as mentioned before, to Matos and Oldfield's OBDD cell, which is essentially just a multiplexor. A schematic diagram of one of these cells and its corresponding truth table covering all values of inputs is given in Figure 4.

As one can determine from the truth table, the function of the cell is not complicated. Whenever the precharge transistor is turned on, the value of the OUT wire is floating. Whenever the precharge transistor is off, OUT may be evaluated and the cell acts somewhat like a multiplexor. The main difference between this cell and a multiplexor is that this cell does not pass the value on the input of either C0 or C1 – this cell can only output a logic 0.

The overall OBDD design is formed by connecting together these cells much in the same fashion as the OBDD diagrams showed earlier. However, since the design uses Dominoe NP logic, it must

Table 2: OBDD Complexity for Some Functions

Function Name	Best Case	Worst Case
parity	linear	quadratic
addition	quadratic	exponential
multiplication	exponential	exponential



PRE	C0	C1	S	$\overline{S}$	OUT
1	Z	Z	1	0	Z
1	Z	Z	0	1	Z
Z	1	0	1	0	Z
Z	0	1	0	1	Z
Z	0	1	1	0	0
Z	1	0	0	1	0

Figure 4: Schematic Representation of an ncell and its Truth Table

conform to Dominoe NP design rules. Thus, two adjacent levels in the OBDD are composed of different cell types – either n or p. The schematic for the pcell is much the same as that of the ncell and is not shown for purpose of space requirements. In order to conform to Dominoe NP rules, the pcell outputs only a logic 1 or float. The essence of Dominoe NP logic is explained in [6].

The design of a cell was laid out utilising Cadence tools and is shown in Figure 8. Both the n and p cells have identical dimensions and characteristics. The cell is $45.9\mu m$ in height and $31.9\mu m$ in width. All transistors in the the layout are $14.25\mu m$ in width and $1.5\mu m$ in length. Furthermore, these transistor widths directly limit the height and thus area of the layout. In other words, these transistor lengths could be adjusted at will to fit area requirements. So there is an indirect relationship between the width of the transistors (and thus area of the cell) and the propogation delay for the cell.

The sequential and combinational circuit synthesis system, SIS, was used to generate the OBDD structures for each function in the benchmarks. Unfortunately, each structure generated corresponded to only a single function in a benchmark and SIS made no attempt to produce minimal multi-function OBDDs by combining any of the structures. At this point, a program engineered by Carl McCrosky was used to determine the multi-function OBDD structures.

A CAD program was developed to read OBDDs in this format and to generate a layout in Cadence. This program does this automatically with little to no intervention on the user's part. For this reason, we can declare OBDDs programmable logic much as we can call PLA generation programmable.

One problem this program creates is that in order to ensure correctness, inverters were added to the design at certain points. The program inserts an inverter into a level in the layout if a path in the OBDD from one node to another skips over that particular level. This can increase the number of nodes in an OBDD dramatically, as shown in Figure 5.

Despite the success the CAD program demonstrates with generation of OBDDs, it suffers because of the fact that it is an interpreted program and not compiled. Therefore, it is not as fast as one would like and it sometimes crashes when running on large OBDDs. However, structures with complexities of greater than 1000 nodes have been generated. If this does not seem like ver much, then recall that 1000 nodes would require an area of $1464210\mu m$ for the cells alone!

Another problem affecting our OBDD layouts is a problem which plagues all VLSI design – wiring. For large OBDDs, the wiring dominates over the OBDD cells in terms of area and this is a problem which could be dealt with more carefully than has been. The CAD program allocates one vertical segment of metal for every node in a particular level of the OBDD. These metal segments each initially occupy one column each. The program next tries to pack the wires in somewhat by placing two or

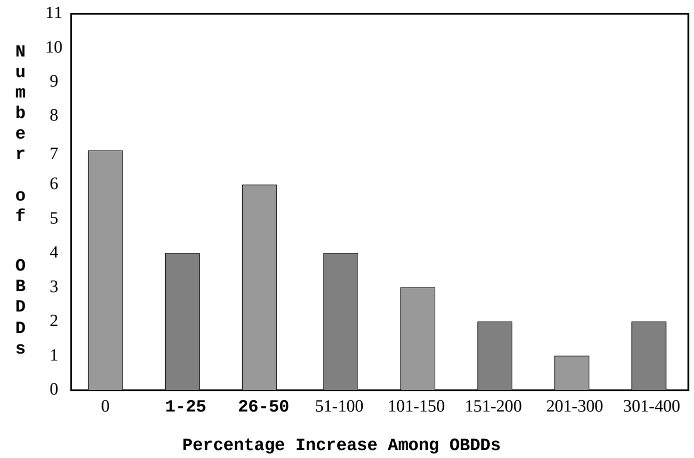


Figure 5: Number of OBDDs versus Percentage Increase in number of Nodes

more different pieces of metal into the same column if they do not cross over each other. This method greatly reduces the area of the OBDD, but the wiring problem could be even more reduced with more time and consideration spent. For example, cell ordering inside a level of the OBDD can be altered to produce a minimal sized layout.

4 Experiment

As stated previously, the complexity of most OBDDs can only be determined from actually producing their structure. It is impossible to determine which functions are applicable for mapping into hardware simply by examining a function's definition. This is a problem which greatly affected this project and the ideal solution to it would be to map out every possible function using the OBDD technique and then compare these to those generated by other design techniques. Unfortunately, this an unrealistic solution as there are 2^{2^n} possible functions for functions of n variables.

The solution to this minor hurdle is to compare the techniques using an assortment of functions. Certain benchmark functions that have been accepted as a medium for design testing are considered to be good examples of such functions. The benchmark functions used to compare the three techniques were obtained from the Microelectronics Centre of North Carolina (MCNC) FTP site [7].

The experiment involves implementing the benchmarks using the three different design techniques and comparing them to determine whether the OBDDs show significant improvements, namely in speed or size. A set of benchmarks was chosen for each of the design techniques and the results for this technique were considered the standard against which the others were measured.

The benchmarks used for implementing the random logic standards were chosen from a large set of combinational multi-level benchmarks. The PLA benchmarks were chosen from a set of benchmarks intended for two-level implementations, such as PLAs. The OBDDs presented a problem, though. No set of benchmarks has been derived for OBDD implementations. Furthermore, the program which derived the OBDD layout was interpreted and not compiled and was thus slow and inefficient. This had the effect of making the generation of certain OBDD layouts either very time consuming or impossible.

The solution to this problem was to choose benchmarks which could be represented by OBDDs that were not too complex. For this reason, the OBDD benchmarks all contain either ten or less outputs. This minimised the number of individual functions in a benchmark and so it was possible, in real-time, to derive layouts. This also made comparisons against the other techniques possible and easier. The OBDD benchmarks were chosen from a large group of combinational multi-level benchmarks.

4.1 Random Logic Details

All qualitative measurements or results relating to the random logic area and propogation delays were derived using SIS. This logic simulation program can be used to determine the number and types of gates that make up a function. It's problem, however, is that the measurement of area that it reports is equal to the sum of all the gates making up the benchmark. This is inaccurate because it does not take into account the area consumed by the wiring connecting these gates. This is quite unrealistic considering that it has been reported that interconnect in a chip may take up as much as 70 % of the area [3].

There is no accurate method to predict the area devoted to wiring in these benchmarks other than to actually lay out the circuit. Even if one were to go ahead and do this, one might not know if the layout chosen was the optimal one. So, for the purpose of achieving some qualitative results for comparison we define the area of a benchmark to be the sum of the area and the wiring. Furthermore, we will assume that wiring in our benchmarks takes up 50 % of the total area. This means that, for a given benchmark, we will consider its total area to be 2 * gate area.

The values SIS outputs foor teh delay throught the circuit similarly does not take into account the metal wires. These must be considered and so a good rule of thumb is that metal doubles the capacitance. Hence we also double the delays reported by SIS.

SIS arrived at its layout for each benchmark by determining which combination of gates would be used. It selects these gates from libraries which contain various information about the gates within them, such as rise and fall models and area. For this project, a very simple library consisting of three gates was used. An inverter, NOR, and NAND gate were all employed. Their timing and area specifics were obtained from CMC's 1.5μ Mitel 2-metal process largely due to the fact that this is the same technology that the PLAs and OBDDs were simulated in.

4.2 PLA Details

PLA structures are highly regular structures which can be characterized by the schematic shown in Figure 6.

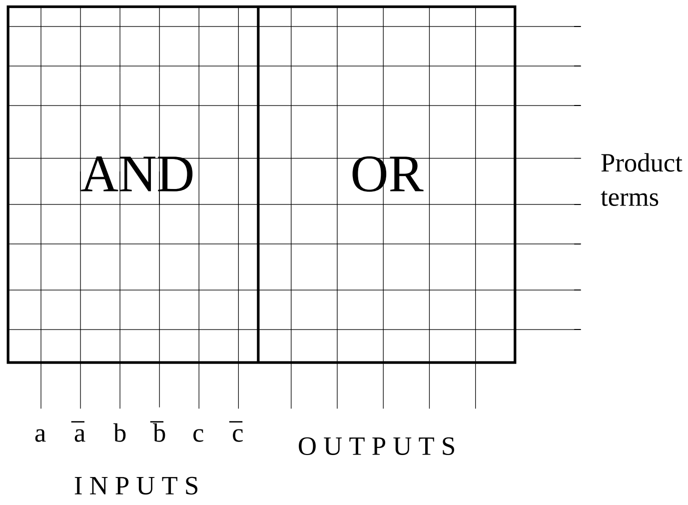


Figure 6: Schematic of a PLA

A problem arises in measuring the speed and area of a PLA. Because there are many different techniques to design PLAs, they are generally not characterized by such measurements. Instead, PLAs are often represented by three values : number of inputs, number of outputs, and number of product terms. Fortunately, we can use these qualities to measure area and to derive a delay time.

The area problem is solved with a small amount of work, as it can be generalised into the formula

$$(2in + out) * pT * area,$$

where in is the number of inputs, out the number of outputs, pT the number of product terms and $area$ the area of the cells bounded by each input/output and product term. Unfortunately, the $area$ variable is largely technology dependant and depends on the design of the PLA.

For the purpose of comparison, a sample value for the variable $area$ was determined. The layout, in mitel15 technology supplied by CMC, was created, and the area is

$$area = (9.7\mu m)w + 53.835\mu m^2$$

where w is the width of the transistor. For the purposes of this project, the transistor width was assumed to be $w = 4.45\mu m$. This gives an area of $97\mu m^2$.

Measuring the speed of a PLA is a complex task and depends on many variables. For the purporses of this project, we can assume that it is 4 * the delay through the product term transistor. This transistor was measured at $0.15ns$ fall time when the output was connected to a suitable load and the input to a pull up. Thus, we assume the speed of the PLAs to be 0.6 ns. However, in considering the metal capacitance, we must double this Figure to 1.2 ns.

The PLAs were minimized using the program Espresso.

4.3 OBDDS

The method used to determine the area of the OBDDs is debatable. It was defined by taking the length at the widest level in the OBDD and multiplying this by the height. Of course, this is an excellent method for determining the area of an OBDD like Figure 9, but it may not be so kind an estimation for an OBDD shaped in the manner of Figure 10.

Also, because of time and technology constraints, the delay through an OBDD could not be measured for each benchmark. However, a timing model was developed. For this project, an upper bound of $0.53ns$ was used for each level in the OBDD. Again, metal capacitance was taken into consideration and this so each level in the OBDD was modelled at $1.06ns$, or twice that of the measurements.

Table 3: Random Logic Benchmarks

Name	Random Logic Area (μm^2)	Random Logic Delay (ns)	PLA Area (μm^2)	PLA Delay (ns)	OBDD Area (μm^2)	OBDD Delay (ns)
C1355	773744	141.34	NP	NP	NP	NP
C1908	804836	170.34	NP	NP	NP	NP
C2670	1247562	187.98	NP	NP	NP	NP
C3540	1784710	247.92	NP	NP	NP	NP
C432	383280	193.52	NP	NP	NP	NP
C6288	3454098	523.90	NP	NP	NP	NP
C7552	4005352	461.8	NP	NP	NP	NP
b9	224402	51.04	NP	NP	NP	NP
dalu	3036258	43.40	NP	NP	NP	NP
k2	439416	349.44	NP	NP	3410000	30.74
rot	2029502	286.22	NP	NP	NP	NP
t481	2821758	367.16	NP	NP	2745000	16.92

Table 4: PLA Benchmarks

Name	PLA Area (μm^2)	PLA Delay (ns)	Random Logic Area (μm^2)	Random Logic Delay (ns)	OBDD Area (μm^2)	OBDD Delay (ns)
b12	158886	1.2	171944	219.44	1706400	15.90
cordic	4255584	1.2	5002030	179.56	403200	24.38
ex1010	826440	1.2	4632228	183.54	NP	NP
cps	2467098	1.2	3008394	203.54	NP	NP
duke2	608966	1.2	1001418	101.18	NP	NP
ex4	7685892	1.2	3007746	203.76	NP	NP
misex2	184688	1.2	208386	43.02	2496000	26.50
misex3c	794430	1.2	208386	43.02	NP	NP
pdc	845064	1.2	3007746	203.44	NP	NP
rd84	494700	1.2	1520552	128.90	227250	8.48
spla	2110194	1.2	3007746	203.76	NP	NP

5 Results

The results for the three benchmarks sets are shown in Tables 3, 4, and 5. Note that a value of 'NP' means in the tables means that it was not possible to determine the information in the time givben. In other words, the benchmark was too complex to implement in the targeted design.

Analysing the results illustrated in the tables, one can see that the majority of the OBDD implementations have area which are much worse than the corresponding PLA or random logic implementations. In fact, some of the OBDD areas are prohibitive. However, one may also note that the speed of the OBDDs is faster than that of random logic, though not as quick as PLAs.

We can determine from our results that out of the 52 benchmarks dealt with, only 9 generated OBDDs with attractive mappings. The only OBDD benchmarks which have areas which are within reasonable bounds of the PLAs and random logic implementations are 9symml, parity, cm138, majority, and z4ml benchmarks. Only two PLA benchmarks had more favourable OBDD results - cordic and rd84, and t481 and k2 were the only good OBDD results for the random logic benchmarks. Figure 7 shows the timing delays for each of these benchmarks with their design techniques.

Figure 7 shows the relative timing of the three techniques and one can determine that the OBDD designs lead to relatively good delay times, though not as good as the delays for PLAs. This fact is especially useful for some of the benchmarks in the random logic style which could not generate PLAs because they were too complex.

6 Conclusions

The data derived from the experiments seems to suggest that the OBDD technique is not often very useful for implementing hardware. However, there are some cases for which it is useful and these cases cannot be ignored. The reason for this is that the complexity for most OBDDs is exponential, at least $2^n/2n$ nodes. Therefore the mappings for these OBDDs are too large for real-world use. Despite their exponential size, the propogation delay for the OBDDs remains quite attractive.

Before functions are plotted out using the OBDD method, one should ensure that the OBDD complexity is not prohibitive. This is often the case with many functions. A good example of a function type which generates attractive OBDDs are the symmetric functions. Symmetric functions are a special case of functions in which the output is determined by the number of arguments equal to 1. [2] presents a best case complexity of linear and a worst case complexity of quadratic for this. This being the case, any symmetric function is a good candidate for mapping into OBDD form.

Table 5: OBDD Benchmarks

Name	OBDD Area (μm^2)	OBDD Delay (ns)	Random Logic Area (μm^2)	Random Logic Delay (ns)	PLA Area (μm^2)	PLA Delay (ns)
9syml	105800	9.54	344076	81.90	263549	1.2
C17	57500	5.30	8683	13.16	8148	1.2
alu2	6755000	10.60	1042404	333.76	448916	1.2
b1	22750	3.18	23402	23.14	5820	1.2
cm138	89056	6.36	42864	24.44	93120	1.2
cm150	979200	22.26	139320	66.14	70907	1.2
cm151	635100	12.72	68860	51.76	42874	1.2
cm152	211700	11.66	49052	30.30	17848	1.2
cm162	725400	14.84	85730	44.30	188859	1.2
cm163	443700	16.92	86378	36.20	161505	1.2
cm82	108780	5.30	40218	27.04	29003	1.2
cm85	489100	11.66	77294	42.82	116400	1.2
cmb	127650	16.92	92166	54.36	90792	1.2
comp	1620162	33.93	261176	71.00	NP	NP
con1	118800	7.42	34369	25.04	15520	1.2
i3	6000000	139.9	188632	35.90	NP	NP
major	29240	5.30	29840	24.92	5335	1.2
mux	937500	22.26	217770	78.02	66736	1.2
parity	72900	16.92	96552	37.66	NP	NP
x2	372600	10.60	102546	36.42	81189	1.2
z4ml	250000	7.42	366238	76.12	103014	1.2

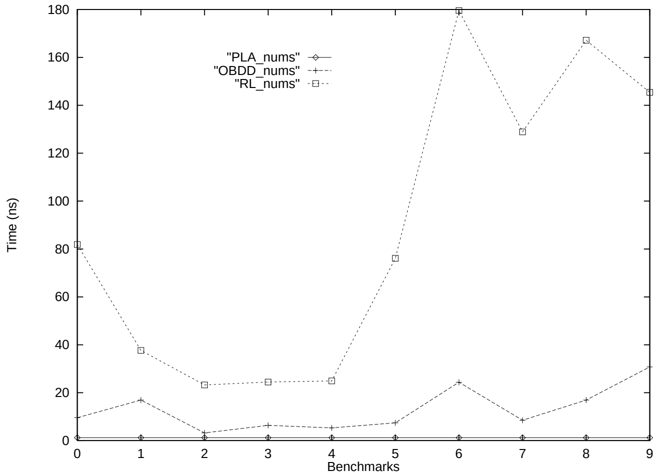


Figure 7: Delay Comparison Across the Design Techniques for Similar Areas

Future work in this area could deal with the problem of determining the class of functions which have OBDDs with complexity that is not exponential. Finding this class of well-behaved functions would eliminate much of the work involved in implementing OBDDs. That is, one would not have to generate the OBDD structure in order to know its complexity – one would already know the complexity.

As well, a performance comparison of OBDDs with different cell sizes should be run, as the cell used in this project may be deemed to large or too slow for some purposes. Finally, alternative implementations of OBDDs should be suggested. This is an open area. Many different ideas could be explored, such as different technologies or multiple valued logic (MVL).

One approach that might be taken is one similar to this one, but which miinimises or eradicates the use of inverters. Figure 5 show that in some cases, adding transistors can greatly increase the total number of nodes used. From the graph, we are made aware that most OBDD structures are increased in size by about $25 - 50\%$ by the addition of inverters.

So the answer to the question posed at the beginning of the project as to whether or not OBDDs are a suitable technique for CMOS designers to use is yes. However, one is not likely to find very many functions for which one would wish to use this method. The main problem with it lies in the OBDDs themselves – most functions have representations of prohibitive size, despite the fact that the speed of OBDDs is quite fast. However, it is this size limitation that makes OBDDs an occasionally useful, though not necessary addition to a CMOS designer's arsenal of design techniques.

References

[1] McCalla, T. *Digital Logic and Computer Design.* Merrill, Toronto, 1992.

[2] Bryant, R. Symbolic Boolean Manipulation with Ordered Binary Decision Diagrams. *ACM Computing Surveys 24,* 3 (September 1992), 293-318. September 1992.

[3] Butler, J. Preface. In *Multiple Valued Logic in VLSI*, xv-xvi, 1991.

[4] Liaw, H., and Lin, C. On the OBDD-Representation of General Boolean Functions. *IEEE Transactions on Computers 41,* 6 (June 1992), 661-664.

[5] Matos, J., and Oldfield, J. Binary Decision Diagrams : From Abstract Representations to Physical Implementations. In *Proceedings of the ACM IEEE 20th Design Automation Conference* (1983), 567-570.

[6] Lee, C., AND Szeto, E. Zipper CMOS. *IEEE Circuits and Devices Magazine 2,* 5 (May 1986), 10-16.

[7] LGSynth93.tar. Obtained via anonymous FTP from *mcnc.mcnc.org* in directory pub/benchmark/LGSynth93 (1993).

Appendix: Cadence Layouts

This appendix presents several Cadence layouts which provide examples of the work accomplished. The first figure is the layout of an ncell in Cadence. The second is the implementation of the benchmark 'cm138'. The last benchmark is also a layout for a benchmark. This benchmark illustrates 'misex2'.

One can see from Figures 9 and 10 both the best and worst aspects of an OBDD's area consumption. The OBDD 'cm138' is compact and does not waste any area, while on the other hand, 'misex2' is sparse and much space is wasted. As well, 'misex2' illustrates the limitation that wiring may dominate the area in an OBDD design.