

High Density Standard Cell Library

S. Whitaker, L. Miles, J. Gambles, G. Donohoe, and L. Davis
Center for Advanced Microelectronics and Biomolecular Research
University of Idaho, 721 Lochsa Street # 8
Post Falls, Idaho 83854

Abstract

A high density standard cell library based on universal logic gates (ULG) is presented. Optimization and reduction of the resulting ULG logic circuits is accomplished using a selection algebra. Reductions in area of more than 50% have been achieved for some test cases when compared with commercially available libraries. Several integrated circuits designed for NASA have been implemented using a radiation tolerant version of the ULG library.

1. Introduction

Standard cell libraries available commercially are based on boolean logic. The combinational logic is formed from logical *and*, *or* and *invert* functions. Memory elements are implemented with flip flops and latches. Normally, the libraries contain a few hundred variations of these fundamental cells. Higher level logic is constructed from combinations of these standard cells.

The universal logic gate library is based on just 14 kernel cells. The kernel cells are combined into 142 basic cells. The characteristics, properties, and operation of the higher level library cells such as adders, multipliers, ALU's, comparators, decoders, state machines, etc. are then programmed into combinations of these 156 basic plus kernel cells. The flexibility of these basic cells is introduced by using universal logic gate structures.

2. Library

The 14 kernel cells consist of three types: universal logic gates, memory and buffers. The three universal logic gates are implemented with 2 to 1, 4 to 1 and 8 to 1 multiplexors. The input to the muxes can be programmed to implement any 2-variable, 3-variable or 4-variable combinational logic function as well as many higher functions. The memory cells are D flip flops with synchronous, asynchronous and clocked set and reset options. Inverting, non-inverting, and tri-stating buffers form the last group.

One of the powerful features of this library is that only fourteen cell layouts need to be drawn. There are three U cells. The layouts are named U2 (2-1 mux), U4 (4-1 mux) and U8 (8-1 mux). There is one D flip flop cell, D1. The three resettable flip flops are DR1 (synchronous), DR2 (asynchronous) and DR3 (clockedge). The three settable flip

DB	:	D → B
DBN	:	D → BN
DZB	:	D → ZB
DRB	:	DR → B
DRBN	:	DR → BN
DRZB	:	DR → ZB
DSB	:	DS → B
DSBN	:	DS → BN
DSZB	:	DS → ZB

TABLE I
MEMORY → BUFFER CELLS

UD	:	U → D
UDB	:	U → D → B
UDBN	:	U → D → BN
UDZB	:	U → D → ZB
UDR	:	U → DR
UDRB	:	U → DR → B
UDRBN	:	U → DR → BN
UDRZB	:	U → DR → ZB
UDS	:	U → DS
UDSB	:	U → DS → B
UDSBN	:	U → DS → BN
UDSZB	:	U → DS → ZB

TABLE II
MUX → MEMORY → BUFFER CELLS

flops are DS1 (synchronous), DS2 (asynchronous) and DS3 (clockedge). The inverting buffer is called BN1. Two non-inverting buffer cells are made from one cell layout. B2 has both Q and QN outputs while B1 has only the Q output. The tri-state buffer cell is called ZB1. Another major benefit of this approach is that only a small number of SPICE simulations are required to characterize the entire library.

The layouts for the basic library cells are automatically generated by combining the 14 kernel cells. Buffers can be added to the output of the flip flop and mux cells. Mux cells can directly drive the inputs of flip flops. The kernel cell layouts are drawn to allow these combinations to be connected by abutment. The assembly of the basic cells has been automated. The resulting 142 basic cells are summarized in Tables I,II, and III. Figure 1 illustrates a basic cell consisting

UB	:	U → B
UBN	:	U → BN

TABLE III
MUX → BUFFER CELLS

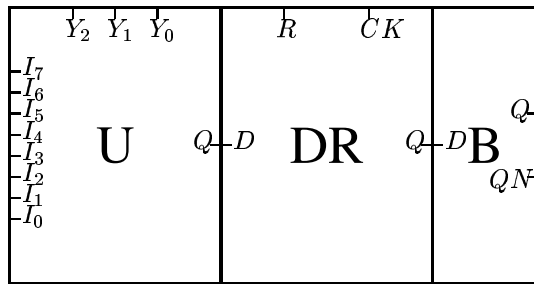


Fig. 1. Example Basic Cell

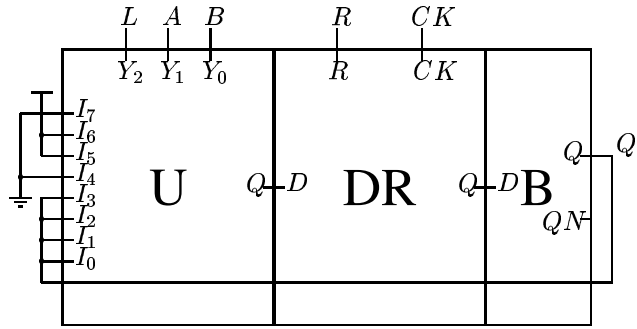


Fig. 2. Example Programmed Basic Cell

of an 8 to 1 mux driving a resettable D flip flop with a buffered output.

Basic cells are personalized by programming inputs to connect to logic high and low levels or to connect to external logic signals. The example basic cell shown in Figure 1 can be programmed to perform a 1-bit clocked resettable register which loads the *exclusive or* of two inputs A and B when a control signal is asserted. This programmed function shown in Figure 2 can be specified by the following VHDL description.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
```

```
entity xorff is
    port (R, CK, L, A, B: in std_logic;
          Q: out std_logic);
end xorff;
```

```
architecture behave of xorff is
begin
    process(R, CK, A, B)
    begin
        if RISING_EDGE(CK) then
            if R = '1' then
                Q <= '0';
            elsif L = '1' then
                Q <= A xor B;
            end if;
        end if;
    end process;
end
```

An n -bit register with the illustrated characteristics can be constructed by arraying n of these programmed 1-bit cells.

Most libraries are boolean based (*and*, *or*, *invert*) while most high level design (HDL) languages are not. The universal logic gate naturally implements non-boolean constructs such as if-then-else clauses and case statements. For example the logic described by the following VHDL code can be implemented in Figure 3.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
```

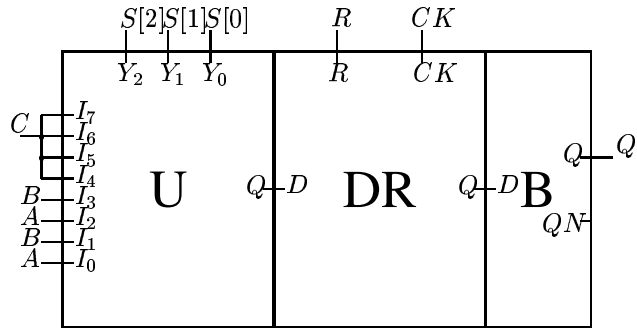


Fig. 3. Case Example

```
entity regA is
    port (R,A,B,C,CK: in std_logic;
          S: in std_logic_vector(2 downto 0);
          Q: out std_logic);
end regA;
```

```
architecture behave of regA is
begin
    process(CK)
    begin
        if RISING_EDGE(CK) then
            if R = '1' then
                Q <= '0';
            else
                case S is
                    when "000" => Q <= 'A';
                    when "001" => Q <= 'B';
                    when "010" => Q <= 'A';
                    when "011" => Q <= 'B';
                    when others => Q <= 'C';
                end case;
            end if;
        end if;
    end process;
end
```

Modern logic synthesis tools normally try to generate the most area efficient implementation of a function within a timing constraint. When programming a ULG to perform a

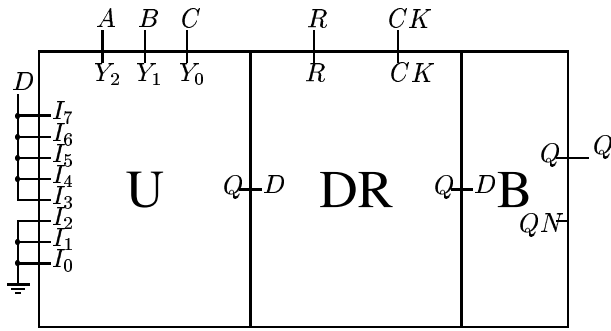


Fig. 4. Trade-off Example 1

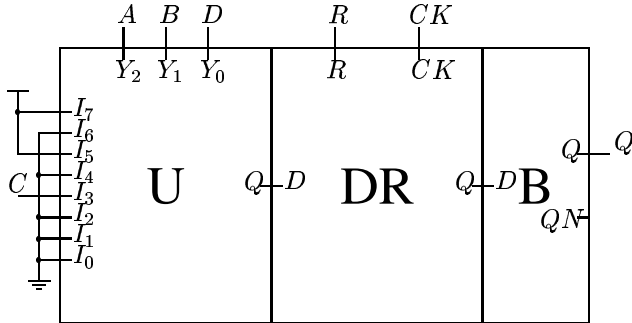


Fig. 5. Trade-off Example 2

given function, there are often several functionally equivalent implementations requiring the same minimum area. These implementations can be differentiated by characteristics such as their power, interconnect, fan-in load requirements as well as their speed. This leads to the ability to choose efficient circuits based on more extensive criteria than the traditional area/speed trade-off possible with current libraries. For example, Figure 4 and Figure 5 both implement the same logical function using two different permutations of the next state table. Both are implemented with the same basic cell and operate at the same speed. The second implementation has, however, distinct advantages. The fan-out load seen by D is considerably lower thus reducing power in the overall circuit for the same system speed of operation.

3. Algebra

In summary, the entire library is based on just 14 kernel cells. These layouts are programmatically arrayed to produce 142 basic cells. The 142 basic cells plus 14 kernel cells can be assembled and routed to generate the higher level functions. The universal logic gates in the basic cells are implemented with 2 to 1, 4 to 1, and 8 to 1 multiplexers. The inputs and control of the mux must be programmed to personalize the operation of the cell. An algebra has been developed to allow these programmings to be described, permuted, and optimized.

3.1 Syntax

All programmed basic cells and their functionality can be described by the following syntax:

$$Q < QN > .xxxx < E > Y[n-1:0] I_{2^n-1} I_{2^n-2} \dots I_0 \\ < R/S > < CK > (Param=Opt) ;$$

where:

- Q - non-inverting output
- < QN > - optional inverting output
- .xxxx - basic cell name
- < E > - optional tri-state enable
- $Y[n-1:0]$ - control variables / state variables
- I_x - truth table state / next state
- < R/S > - optional reset or set input
- < CK > - optional clock input
- (Param=Opt) - selects an option
- ;
- sentence terminator

The kernel cell names are:

- U - Universal Logic Gate
- D - D Flip Flop
- DR - Resettable D Flip Flop
- DS - Settable D Flip Flop
- B - Non-inverting Buffer
- BN - Inverting Buffer
- ZB - Tri-state Buffer

The basic cell name is a concatenation of kernel cell names. For example:

- DB - $D \rightarrow B$
- DRBN - $DR \rightarrow BN$
- UDS - $U \rightarrow DS$
- UDZB - $U \rightarrow D \rightarrow ZB$

The number of control inputs to the universal logic gate is n . The algebra works for any n . For the present library, if $n \leq 3$, there is a direct mapping to a single basic cell. If an 8 to 1 mux is used for the ULG, $n = 3$ control inputs select which of the $2^n = 8$ inputs are passed to the output. If all three control inputs are high, the state on I_7 is passed to the mux output. If $Y[2] = 0$, $Y[1] = 1$, and $Y[0] = 0$, then I_2 is passed to the output of the mux. $I_{2^n-1} I_{2^n-2} \dots I_0$ represents the truth table for a combinational logic device, or the next state of a sequential circuit. $Y[n-1:0]$ can represent the control variables for a combinational circuit or the present state variables for a sequential circuit. Parameters such as the type of reset (asynchronous, synchronous or clock-edge) are assigned and enclosed in parenthesis at the end of the statement. The truth table states, $I_{2^n-1} I_{2^n-2} \dots I_0$, can be assigned to logical 1's or 0's as well as map entered variables [1]. For the basic cell, this corresponds to a connection to VDD for a logic 1, a connection to VSS for a logic 0, and a connection to a signal for a map entered variable.

3.2 Operations

Several of the basic operations of the algebra can be illustrated with an example. The combinational logic function $C = Z + \bar{Y}$ can be implemented with a buffered universal logic gate.

$$C .UB Z Y VDD VDD VSS VDD ;$$

Logical operations can be performed on the function. For example, the output can be inverted by using an inverting buffer.

$$CN .UBN Z Y VDD VDD VSS VDD ;$$

The output could also be inverted by inverting all of the input states.

$$CN .UB Z Y VSS VSS VDD VSS ;$$

Different implementations are possible by permuting the control inputs. Permuting the control, rearranges the truth table. The truth table states I_3 and I_0 remain in the same position since they represent states where both control are high or both are low. States I_2 and I_1 must be interchanged.

$$C .UB Y Z VDD VSS VDD VDD ;$$

Inverting control inputs also permutes the truth table. The least significant control interchanges neighboring states. The next interchanges neighboring pairs of states.

$$C .UB Z YN VDD VDD VDD VSS ;$$

$$C .UB ZN Y VSS VDD VDD VDD ;$$

Reductions can be made by allowing control variables to become map entered variables. When Z is high, the output is high. When Z is low, the output is high when YN is high and low when YN is low. The 4 to 1 mux can therefore be reduced to a 2 to 1 mux by entering YN into the truth table.

$$C .UB Z YN VDD VDD VDD VSS ;$$

$$C .UB Z VDD YN ;$$

Permuting the truth table can also lead to map entered variable reductions of the truth table.

$$C .UB ZN Y VSS VDD VDD VDD ;$$

$$C .UB Y ZN VSS VDD VDD VDD ;$$

$$C .UB Y Z VDD ;$$

4. Reduction Rules

The mux based logic can be transformed and reduced by applying sequences of algebraic rules. These rules will be illustrated in the following sections.

4.1 Removing Inversions

Inverting X interchanges adjacent states.

$$Q .UB Z Y X A B C D E F G H ;$$

$$Q .UB Z Y XN B A D C F E H G ;$$

Inverting Y interchanges adjacent state pairs.

$$Q .UB Z Y X A B C D E F G H ;$$

$$Q .UB Z YN X C D A B G H E F ;$$

Inverting Z interchanges adjacent state quads.

$$Q .UB Z Y X A B C D E F G H ;$$

$$Q .UB ZN Y X E F G H A B C D ;$$

Inverting Z, Y, and X can be accomplished by applying each rule in sequence.

$$Q .UB Z Y X A B C D E F G H ;$$

$$Q .UB ZN YN XN H G F E D C B A ;$$

The truth table states can all be inverted by inverting the buffer.

$$Q .UB Z A B ;$$

$$Q .UBN Z AN BN ;$$

4.2 Reducing Nodes

Nodes can be reduced by entering control variables into the truth table as map entered variables.

$$Q .UB Z Y VDD VSS A A ;$$

$$Q .UB Z Y A ;$$

Redundant control can be easily identified and removed.

$$Q .UB Z Y A B A B ;$$

$$Q .UB Y A B ;$$

Permuting the control variables can result in reductions.

$$Q .UB Z Y X VDD VDD VSS VSS A B A B ;$$

$$Q .UB Z X Y VDD VSS VDD VSS A A B B ;$$

$$Q .UB Z X Y Y A B ;$$

4.3 Merging Nodes

Nodes can be merged by combining control and then defining states.

$$Q .UB Z A B ;$$

$$A .UB Y C D ;$$

$$Q .UB Z Y A A B B ;$$

$$Q .UB Z Y C D B B ;$$

The merging steps can be sequentially applied.

$$Q .UB Z A B ;$$

$$A .UB Y C D ;$$

$$B .UB X E F ;$$

$$Q .UB Z Y A A B B ;$$

$$Q .UB Z Y C D B B ;$$

$$Q .UB Z Y X C C D D B B B B ;$$

$$Q .UB Z Y X C C D D E F E F ;$$

Merging of different sized muxes.

```
Q .UB Z Y A B C D ;
A .UB X E F ;
```

```
Q .UB Z Y X E F B B C C D D ;
```

Larger into smaller.

```
Q .UB Z A B ;
A .UB Y X C D E F ;
```

```
Q .UB Z Y X A A A A B B B B ;
Q .UB Z Y X C D E F B B B B ;
```

Control must be entered into the map before merging (Sub-function *and*). This may require a map entered variable to become a control variable.

```
Q .UB Z Y A VDD VSS VSS ;
Z .UB X B C ;
```

```
Q .UB Y Z A VSS VDD VSS ;
Q .UB Y A Z VSS Z Z ;
Q .UB Y A X Z Z VSS VSS Z Z Z Z ;
```

```
Q .UB Y A X B C VSS VSS B C B C ;
```

Control must be entered into the map before merging (Sub-function *or*).

```
Q .UB Z Y VDD VDD A VSS ;
Z .UB X B C ;
```

```
Q .UB Y Z VDD A VDD VSS ;
Q .UB Y A VDD Z Z Z ;
Q .UB Y A X VDD VDD Z Z Z Z Z Z ;
```

```
Q .UB Y A X VDD VDD B C B C B C ;
```

4.4 Utilizing Set/Reset

If half of the truth table states are low, the control variable can be a reset input to a flip flop.

```
Q .UD Z Y X VSS VSS VSS VSS A B C D CK ;
Q .UDR Y X A B C D Z CK (RST='C') ;
```

If half of the truth table states are high, the control variable can be a set input to a flip flop.

```
Q .UD Z Y VDD VDD A B CK ;
Q .UDS Y A B Z CK (SET='C') ;
```

Permutations may result in set/reset utilization.

```
Q .UD Z Y VSS A VSS B CK ;
Q .UD Y Z VSS VSS A B CK ;
Q .UDR Y A B Z CK (RST='C') ;
```

Expansion, inversion and permutation may result in set/reset utilization.

```
Q .UD AN VSS NOT_RST CK ;
Q .UD AN NOT_RST VSS VSS VDD VSS CK ;
Q .UD A RST VSS VDD VSS VSS CK ;
Q .UD RST A VSS VSS VDD VSS CK ;
Q .UD RST VSS A CK ;
Q .DR A RST CK (RST='C') ;
```

4.5 Moving Flip Flops

Flip flops can be moved forward to facilitate reductions.

```
Q .UB Z Y A B C D ;
A .UD X E F CK ;
B .UD X G H CK ;
C .UD X I J CK ;
D .UD X K L CK ;
```

```
Q .UD Z Y A B C D CK ;
A .UB X E F ;
B .UB X G H ;
C .UB X I J ;
D .UB X K L ;
```

```
Q .UD Z Y X E F G H I J K L CK ;
```

4.6 Reduction Example

The following is an example in which many of the preceeding rules were applied to obtain a reduced implementation as an illustration of the utility of the algebra.

```
Q0 .UB Y X C VDD VDD C ;
Q1 .UB Y Q3 Q2 ;
Q2 .UB X B VDD VSS VSS VSS ;
Q3 .UB X A VDD VSS VSS VSS ;
Q .UB Z Y Q1 Q0 ;
```

```
Q0 .UB Y X C VDD VDD C ;
Q1 .UB Y Q3 Q2 ;
Q2 .UB X B VSS ;
Q3 .UB X A VSS ;
Q .UB Z Y Q1 Q0 ;
```

```
Q0 .UB Y X C VDD VDD C ;
Q1 .UB Y X A VSS B VSS ;
Q .UB Z Y Q1 Q0 ;
```

```
Q .UB Z Y X A VSS B VSS C VDD VDD C ;
```

```
Q .UB Z X Y A B VSS VSS C VDD VDD C ;
```

5. Test Results

Software to convert a netlist in a commercially available triple metal $0.35\mu\text{m}$ CMOS library to the high density library has been written. The initial conversion is a straight cell by cell translation into the high density library drawn for the same $0.35\mu\text{m}$ process. Although the result of this conversion is a logically equivalent circuit which is larger and slower than the initial design, it is then in a form to easily apply selection based algebraic reductions.

Software routines have been written to partially implement the reductions possible in the selection domain. Mux utilization and reduction routines have been implemented to work on local groups of logic, but the current software is ineffective for global reductions. Also, while some of these routines are generally applicable, many are specific to the given bench mark. Rudimentary set/reset utilization routines have also been written, and are effective in certain cases. An efficient redundant cell identification and removal routine has been implemented, but the redundant input variable routine is only marginally effective. Effective logical inversion removal routines are in place. A first effort at subfunction encoding [2] has been attempted, but it is specific and falls dramatically short of the full potential of this concept.

Even with the short comings of this first attempt to automate synthesis based on the high density concepts, some promising results have been obtained. Several circuits from the benchmark suite originating at the 1993 International Logic Synthesis Workshop were converted and reduced. All resulted in reductions in area. The largest of these benchmarks was approximately 27K gates. This netlist was converted to the high density library and reduced using the available software routines. The resulting circuit was implemented with a triple metal $0.35\mu\text{m}$ ULG library using Avant! placement and routing software. The resulting area was $915K\mu\text{m}^2$ with an estimated speed of 117 MHz. Cell area was $842K\mu\text{m}^2$ for an impressive 92% area utilization. This circuit was then synthesized and reduced using synopsis targeting a commercial $0.35\mu\text{m}$ standard cell library and constraining it to achieve a similar 117 MHz estimated speed of operation. The resulting cell area was $1,406K\mu\text{m}^2$. The high density library was 40% more efficient in cell area. A commercially available c8051 microcontroller was then converted and optimized using the available routines. This resulted in a 55% reduction in area with a substantial increase in speed.

State-of-the-art comparisons were then attempted using highly optimized million gate netlists obtained from commercial industry. Software to convert the netlists to high density library cells drawn for the TSMC 6 metal, $0.18\mu\text{m}$ process was written. The initial conversion was again a straight cell by cell translation into the high density library followed by application of the reduction routines available. The results were more modest. One circuit achieved just a 10% cell area reduction and failed to maintain speed. Another resulted in a 40% area reduction, but also at the cost of 11% in speed. Analysis of the resulting netlists showed many areas where the

circuits could be further reduced. Software to implement these further reductions should result in smaller and faster circuits.

6. Applications

A number of chips were designed and fabricated for NASA using either radiation tolerant (RT) versions of the triple metal $0.35\mu\text{m}$ cells or the non-RT versions. These include a $t = 5$, 1 Gbit/sec Reed Solomon decoder [3], a quad $t = 8$ Reed Solomon decoder, an ultra-low-power image compressor [4], a c8051 microcontroller, and a $t = 16$ ultra-low-power Reed Solomon encoder [5]. In addition, several more designs are in progress including a bit plane encoding image compressor, a data packetizer, a low density parity check encoder and a data down link controller.

7. Future Work

While the current results are promising, much work remains to implement more of the reduction algorithms and to improve those already implemented. Optimization of the ULG library layouts is needed. Buffer strength issues need to be resolved. Selection of the basic kernel cells needs to be revisited. Cell height trade offs need to be studied. Existing mux architectures need to be compared and trade offs need to be established. Interface with existing tools such as synopsis needs to be improved. The improvements will then need to be applied to a large variety of circuits to determine the benefit and extent of applicability for the high density standard cell library.

REFERENCES

- [1] W. Fletcher, *An Engineering Approach to Digital Design*, Englewood Cliffs, NJ, Prentice-Hall, Inc., 1980, Chap. 3.
- [2] S. Whitaker, L. Miles, J. Gambles, and L. Davis, "Mux-Based ROM Using n -Bit Subfunction Encoding", *Proceedings of the 8th NASA Symposium on VLSI Design*, Albuquerque, NM, Oct. 1999, pp. 3.2.1-3.2.7.
- [3] S. Whitaker, L. Miles, and W. Smith, "A 1 Gbit/sec Reed Solomon Coder", *Proceedings of the 9th NASA Symposium on VLSI Design*, Albuquerque, NM, Nov. 2000, pp. 4.3.1-4.3.8.
- [4] L. Miles, J. Venbrux, J. Gambles, J. Hass, W. Smith, G. Maki, and S. Whitaker, "An Ultra-Low-Power, Radiation-Tolerant Data/Image Compressor for Space Applications", *Proceedings of the 10th NASA Symposium on VLSI Design*, Albuquerque, NM, Mar. 2002, pp. 9.3.1-4.3.7.
- [5] J. Gambles, L. Miles, J. Hass, W. Smith, B. Smith and S. Whitaker, "An Ultra-Low-Power, Radiation-Tolerant Reed Solomon Encoder for Space Applications", submitted for inclusion in the *Proceedings of the IEEE Custom Integrated Circuits Conference*, San Jose, CA, September, 2003.