High Performance and Low Leakage Design Using Cell Replacement and Hybrid V_t Standard Cell Libraries

Liang-Ying Lu, Kuang-Yao Chen, and Tsung-Yi Wu

Abstract—**In recent years, the chip leakage power may be larger than the chip dynamic power because the semiconductor process technology progresses quickly. Therefore, leakage power reduction becomes an important issue for low power circuit designers. In this paper, we propose a heuristic cell replacement algorithm to reduce the leakage power of a logic design. The algorithm contains two procedures. The first one uses a conventional optimization approach and the second one uses a new optimization approach. In the first procedure, the algorithm uses high V_t, normal V_t, and low V_t cells to do cell replacement. In the second procedure, the algorithm employs hybrid threshold voltage standard cell libraries (HTVSCLs) to do cell replacement. The experimental results show that our technique can further reduce the leakage power up to 14.186% by using the second procedure after invoking the first procedure.**

Index Terms—**Low Power Design, Leakage Current Reduction, MTCMOS, Hybrid Threshold Voltage Standard Cell Library**

I. Introduction

Currently, in an advanced nanometer CMOS chip, the leakage power becomes larger than the dynamic power. Therefore, if the leakage power of an advanced nanometer CMOS chip used in a portable electronic device is not properly controlled, the chip will extremely shorten the operating time of the device's battery. Many techniques are proposed for leakage power reduction, for example, V_t (threshold voltage) assignments, power gating techniques, reverse body bias, etc.

In this paper, we use *MTCMOS* (*Multi-Threshold CMOS*) cell replacement technique to do leakage power reduction. Our work employs three kinds of V_t that are low V_t, normal V_t and high V_t. A low V_t transistor has high leakage power and short propagation delay while a high V_t transistor has low leakage power and long propagation delay. In a circuit, high V_t transistors are preferably placed on non-critical paths to reduce the leakage current of the circuit and low V_t transistors are placed on critical paths to keep the circuit performance.

Our algorithm contains two procedures. The first one uses a traditional optimization approach and the second one uses a new optimization approach. In the first procedure, our algorithm uses high V_t, normal V_t, and low V_t standard cells to do cell replacement. In the second procedure, it employs *hybrid threshold voltage standard cell libraries* (HTVSCLs) to do cell replacement. The cell replacement actions can reduce the leakage power of the processed circuit and cannot bring any timing violation.

The rest of the paper is organized as follows. Section II describes the first procedure of our algorithm. Section III presents the second procedure of our algorithm and the characteristics of our HTVSCLs. Section IV shows the experimental results. Finally, we conclude this paper in Section V.

II. The First Procedure of our Algorithm

MTCMOS is a well-known technique. Previous works [1]–[11] used dual or triple threshold voltage standard cell libraries to synthesize low power circuits. The first procedure of our algorithm uses high V_t, normal V_t, and low V_t standard cell libraries to do cell replacement for reducing the leakage power of the input circuit.

The input circuit of the first procedure of our algorithm is a gate level circuit that is synthesized only by a normal V_t standard cell library. The output circuit of the procedure may contain high V_t, normal V_t, and low V_t gates. Fig. 1 shows the pseudo code of the procedure. In the code, $g_{N\text{-}Vt}$ denotes a normal V_t gate. Let p_v be a path that has the worst timing violation, and let g_c be a gate that has the smallest cost among all evaluated gates. The cost function used to decide which gate is g_c will be introduced in the next subsection.

A. Cost Function

The cost of the candidate gate that is chosen for being replaced by other V_t gate is defined as follows:

$$Cost = \frac{1}{Paths_{Total} + \Delta Power_{Leakage} / 1000} \tag{1}$$

where $Paths_{Total}$ is the amount of the paths through the gate, and $\triangle Power_{Leakage}$ is leakage power reduction caused by the cell replacement. Fig. 2 shows the function *Calculate_Replacement_Costs_of_All_Gates* that is called by our algorithm in order to calculate the cell replacement cost (1) of each gate.

Manuscript received January 7, 2008. This work is supported by the National Science Council of R.O.C. under contract no. NSC 96-2221-E-018-023.

Liang-Ying Lu is with Department of Electronic Engineering, National Changhua University of Education, Changhua, Taiwan (e-mail: m95642003@mail.ncue.edu.tw).

Kuang-Yao Chen is with Department of Electronic Engineering, National Changhua University of Education, Changhua, Taiwan (e-mail: muscle9839@gmail.com).

Tsung-Yi Wu is with Department of Electronic Engineering, National Changhua University of Education, Changhua, Taiwan (phone: 886-4-7232105 EXT.7131; fax: 886-4-7211283; e-mail: tywu@cc.ncue.edu.tw).

```
The_First_Procedure()
{
  /* The input circuit only has normal $V_t$ gates; */
  For each normal $V_t$ gate $g_{N\text{-}Vt}$
  {
    Resynthesize $g_{N\text{-}Vt}$ by a high $V_t$ gate;
  }
  Do static timing analysis (STA);
  While ( there exists a timing violation path )
  {
    Call Calculate_Replacement_Costs_of_All_Gates();
    Find $p_v$ that is the worst timing violation path;
    If ( there exists a high $V_t$ gate in $p_v$ )
    {
      Decide which gate is $g_c$ from among all high $V_t$ gates in $p_v$;
      /* $g_c$ is the gate that has the smallest cost among all evaluated
         gates. */
      Resynthesize $g_c$ by a normal $V_t$ gate;
    }
    Else If ( there exists a normal $V_t$ gate in $p_v$ )
    {
      Decide which gate is $g_c$ from among all normal $V_t$ gates in $p_v$;
      Resynthesize $g_c$ by a low $V_t$ gate;
    }
    Else
      Break;
    Do incremental STA;
  }
}
```

Fig. 1 : The first procedure of our algorithm.

```
Calculate_Replacement_Costs_of_All_Gates()
{
  Let $\triangle P_{Leakage}(g_i)$ be leakage power reduction caused by replacing $g_i$ by
  a new $V_t$ gate;
  For each gate $g_i$
  {
    Calculate $\#enter_i$ that is the number of paths entering $g_i$;
    Calculate $\#leave_i$ that is the number of paths leaving $g_i$;
  }
  For each gate $g_i$
  {
    If ( $g_i$ is a combinational gate )
    {
      /* $C(g_i)$ is defined as the cell replacement cost of gate $g_i$; */

        $C(g_i) = \frac{1}{\#enter_i \times \#leave_i + \Delta P_{Leakage}(g_i) / 1000}$ ;

    }
    Else If ( $g_i$ is a flip-flop )
    {

        $C(g_i) = \frac{1}{\#enter_i + \#leave_i + \Delta P_{Leakage}(g_i) / 1000}$ ;

    }
    If ( the caller is HYVT1_Sub-Procedure() )
    {
      $C(g_i) = -1 * C(g_i)$;
      If ( $g_i$ is a fixed gate ) $C(g_i) = \infty$;
    }
  }
}
```

Fig. 2 : The function of calculating the cell replacement cost of each gate.

III. The Second Procedure of our Algorithm

We proposed two kinds of HTVSCL cells. They are $HYVT_1$ cells and $HYVT_2$ cells. Fig. 3 and Fig. 4 show $HYVT_1$ NAND2 cells and $HYVT_2$ NAND2 cells, respectively. There are 6 different structures for both $HYVT_1$ NAND2 and $HYVT_2$ NAND2. For convenience sake, we use different patterns to represent the symbols of logic gates that have the different V_t structures. Fig. 5(a), Fig. 5(b) and Fig. 5(c) show the symbols of a normal V_t NAND2, an $HYVT_1$ NAND2, and an $HYVT_2$ NAND2, respectively. We assume that the corresponding circuit of the symbol shown in Fig. 5(b) is the one shown in Fig. 3(f). Moreover, we assume that the corresponding circuit of the symbol shown in Fig. 5(c) is the one shown in Fig. 4(f).

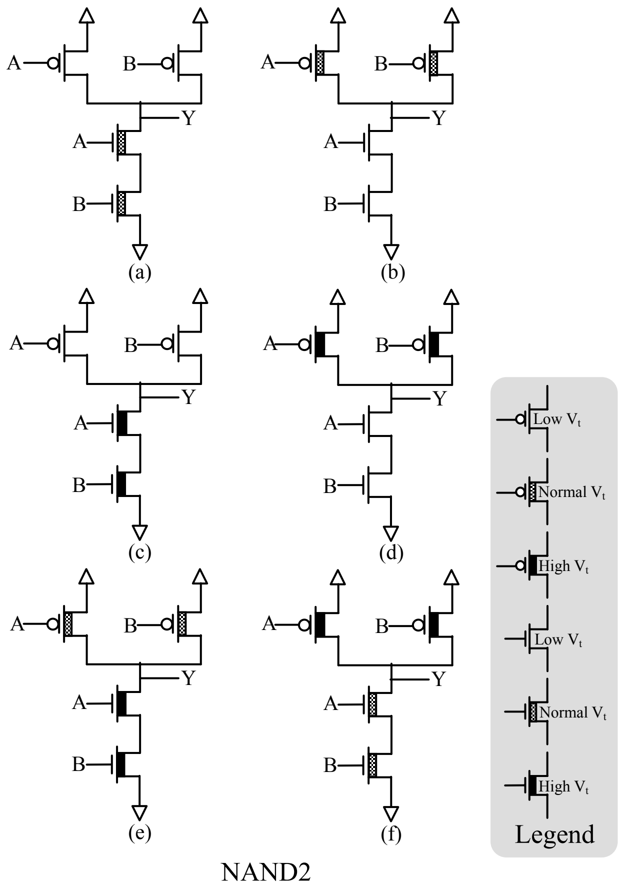

Fig. 3 : $HYVT_1$ cells.

The second procedure of our algorithm consists of two sub-procedures. One is HYVT1 sub-procedure, and the other is HYVT2 sub-procedure. If a *path* has *asymmetric rising* and *falling delays*, HYVT1 sub-procedure can utilize the asymmetric property and uses $HYVT_1$ cells to do cell replacement. If a *gate* in a path has *asymmetric slacks* on its input pins, HYVT2 sub-procedure can utilize the asymmetric slacks and uses $HYVT_2$ cells to do cell replacement.

We use the circuit shown in Fig. 6 to explain the main idea of HYVT1 sub-procedure. We assume that the wire delays of n_3 and n_5 of the circuit are both 0 ns. In the circuit shown in Fig. 6, the rising path delay and the falling path delay ending in the net n_5 are 9 ns and 10 ns, respectively. The difference (t_1) between the two delays is 1 ns (= 10 − 9). The difference (t_2) between the rising delay associated

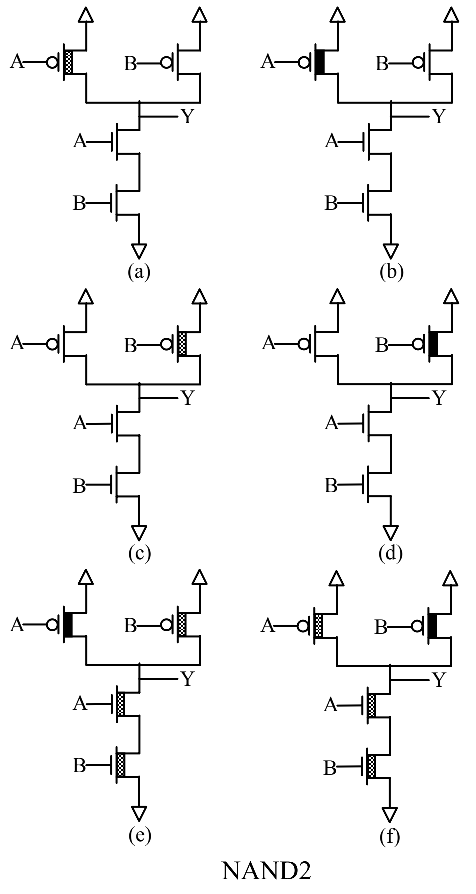

Fig. 4 : HYVT$_2$ cells.

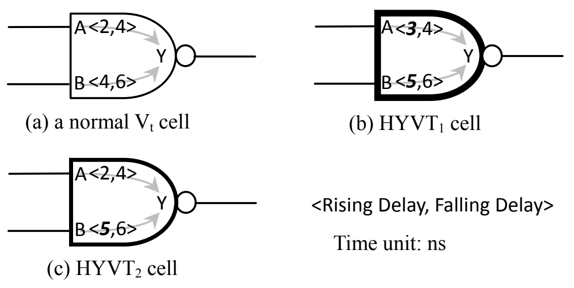

Fig. 5 : The time arc data for NAND cells.

with *A-to-Y* timing arc of the cell shown in Fig. 5(b) and the rising delay associated with *A-to-Y* timing arc of the cell shown in Fig. 5(a) is 1 ns (= 3 − 2). The difference (t_3) between the rising delays associated with *B-to-Y* timing arcs of the two cells shown in Fig. 5(b) and Fig. 5(a), respectively, is also 1 ns (= 5 − 4). Because $t_1 \geq t_2$ and $t_1 \geq t_3$, our algorithm can use the HYVT$_1$ cell to substitute the normal V$_t$ cell g_2 of the circuit shown in Fig. 6 and the substitution does not change the maximum (falling) delay of the critical path. The substitution result is shown in Fig. 7. HYVT1 sub-procedure is shown in Fig. 8.

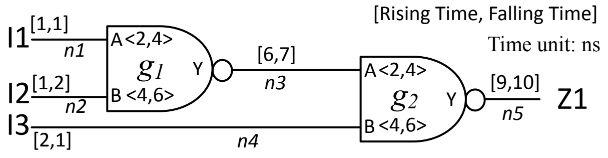

Fig. 6 : An illustration for HYVT1 sub-procedure.

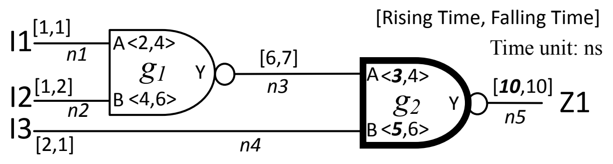

Fig. 7 : The new circuit using an HYVT$_1$ cell.

```
HYVT1_Sub-Procedure()
{
  For each gate $g$
  {
    Calculate the slack of $g$;
    Initialize $g$ as an unfixed gate;
  }
  Call Calculate_Replacement_Costs_of_All_Gates();
  /* $g_c$ is the gate that has the smallest cost. */
  Decide which gate is $g_c$;
  While ( $g_c$ has the positive slack && $g_c$ is an unfixed gate )
  {
    If ( $g_c$ is not implemented by a high $V_t$ gate )
    {
      Re-implement $g_c$ by an $HYVT_1$ gate;
      Do incremental static timing analysis;
      If ( the action of re-implementing $g_c$ induces timing violation )
      {
        Return $g_c$ to its original gate implementation;
        Do incremental static timing analysis;
      }
      For each gate $g$
      {
        Calculate the slack of $g$;
      }
    }
    Set $g_c$ to be a fixed gate;
    Call Calculate_Replacement_Costs_of_All_Gates();
    Update $g_c$;
  }
}
```

Fig. 8 : HYVT1 Sub-Procedure.

We use the example shown in Fig. 6 again to explain the main idea of HYVT2 sub-procedure. In the circuit shown in Fig. 6, the rising delay and the falling delay of net n_2 are assumed as 1 ns and 2 ns, respectively. The maximum rising delay of the maximum rising delay critical path ($n_2 \rightarrow g_1 \rightarrow n_3 \rightarrow g_2 \rightarrow n_5$) is 9 ns (= 1 + 6 + 0 + 2 + 0). The rising delay and the falling delay of net n_4 are assumed as 2 ns and 1 ns, respectively. The rising propagation delay and the falling propagation delay of the g_2's *A-to-Y* timing arc are 2 ns and 4 ns, respectively. The maximum rising delay of the path ($n_4 \rightarrow g_2 \rightarrow n_5$) that is through pin *B* of g_2 to pin *Y* of g_2 is 1 + 4 = 5 ns.

The difference between the maximum rising delay of the maximum rising delay critical path ($n_2 \rightarrow g_1 \rightarrow n_3 \rightarrow g_2 \rightarrow n_5$) and the maximum rising delay of the path ($n_4 \rightarrow g_2 \rightarrow n_5$) through g_2's pin *B* is 9 − 5 = 4 ns (t_4). The maximum difference between *B-to-Y* rising delays of the two cells shown in Fig. 5(a) and Fig. 5(c) is 5 − 4 = 1 ns (t_5). Because $t_5 < t_4$, our algorithm can use the HYVT$_2$ cell to substitute the normal V$_t$ cell g_2 and the substitution does not change the maximum rising delay of the maximum rising delay critical path. The substitution result is shown in Fig. 9. HYVT2 sub-procedure is shown in Fig. 10.

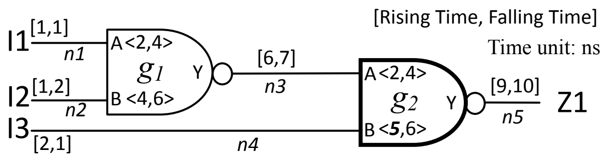

Fig. 9 : The new circuit using an $HYVT_2$ cell.

```
HYVT2_Sub-Procedure()
{
  For each low $V_t$ and normal $V_t$ gate g
  {
    If ( g is a combinational gate && ( replacing g by an $HYVT_2$ gate
        cannot induce any timing violation ) )
    {
        Replace g by an $HYVT_2$ gate;
    }
  }
}
```

Fig. 10 : HYVT2 Sub-Procedure.

IV. Experimental Results

Our algorithm is implemented in C language. Some ISCAS benchmark circuits and three benchmark circuits got from Global Unichip Corporation are used in the experiment. A 90nm standard cell library is used in the experiment for logic synthesis. All gate-level circuits used in the experiment for evaluating the capability of leakage power reduction of our algorithm are created by Synopsys *Design Compiler*.

The experimental results are shown in Table I. The Tech1 (column 4) indicates that Synopsys *Power Compiler* uses high V_t, normal V_t, and low V_t standard cells and our HTVSCLs to do leakage power optimization. The Tech2 (column 5) indicates that our program only uses the first procedure of our algorithm to do cell replacement. The Tech3 (column 6) indicates that our algorithm uses both the first procedure and the second procedure to do cell replacement. In the table, LVT indicates that a circuit is synthesized by Design Compiler only using low V_t standard cells.

In the experiment, *Power Compiler* uses high V_t, normal V_t, and low V_t standard cells and our HTVSCLs to do leakage power reduction and then reports the timing results and the leakage power data. Our program uses our algorithm and HTVSCLs to do cell replacement and then produces a gate level *Verilog* file. The *Verilog* file is read by *Design Compiler* for reporting the timing data and the leakage power data. *Power Compiler* and our program use the same timing constraints when they reduce the leakage power of each benchmark circuit. For each benchmark circuit, there is no timing violation after the cell replacement or circuit optimization in this experiment. The experimental results show that our algorithm can reduce more leakage power than *Power Compiler*. Moreover, our algorithm does not slow down the performance of the optimized circuit. The second procedure of our algorithm can averagely reduce 10.518% (shown in column 7 of Table I) leakage power of the circuit that has been optimized by the first procedure of our algorithm. Compared with *Power Compiler* (shown in column 8 of Table I) and LVT (shown in column 9 of Table I), our program can save more leakage power.

V. Conclusion

The algorithm contains two procedures. The first one is traditional and the second one is new. In the first procedure, the algorithm uses high V_t, normal V_t, and low V_t cells to do cell replacement. In the second procedure, it employs hybrid threshold voltage standard cell libraries to do cell replacement. The experimental results show that our technique can further reduce the leakage power up to 14.186% by invoking the second procedure after the optimization performed by the first procedure.

References

[1] Meeta Srivastav, S.S.S.P. Rao, and Himanshu Bhatnagat, "Power Reduction Technique Using Multi-V_t Libraries," *International Workshop on System-on-Chip for Real-Time Applications*, 2005, pp. 363-367.

[2] Vijay Sundararajan and Keshab K. Parhi, "Low Power Synthesis of Dual Threshold Voltage CMOS VLSI Circuits," *Symposium on Low Power Electronics and Design*, 1999, pp. 139-144.

[3] Liqiong Wei, Zhanping Chen, and Kaushik Roy, "Mixed-V_{th} (MVT) CMOS Circuit Design Methodology for Low Power Applications," *Design Automation Conference*, 1999, pp. 430-435.

[4] Mahesh Ketkar and Sachin S. Sapatnekar, "Standy Power Optimization via Transistor Sizing and Dual Threshold Voltage Assignment," *International Symposium on Low Power Electronics and Design*," 1999, pp. 139-144.

[5] Pankaj Pant, Rabindra K. Roy, and Abhijit Chatterjee, "Dual-Threshold Voltage Assignment with Transistor Sizing for Low Power CMOS Circuits," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2001, pp. 390-394.

[6] J. Jaffari and A. Afzali-Kusha, "New Dual-Threshold Voltage Assignment Technique for Low-Power Digital Circuits," *International Conference on Microelectronics*, 2004, pp. 413-416.

[7] Yen-Te Ho and Ting-Ting Hwang, "Low Power Design Using Dual Threshold Voltage," *Design Automation Conference*, 2004, pp. 205-208.

[8] Frank Sill, Frank Grassert, and Dirk Timmermann, "Low Power Gate-level Design with Mixed-V_{th} (MVT) Techniques," *Symposium on Integrated Circuits and Systems*, 2004, pp. 278-282.

[9] Frank Sill, Frank Grassert, and Dirk Timmermann, "Reducing Leakage with Mixed-V_{th} (MVT)," *VLSI Design Conference*, 2005, pp. 874-877.

[10] Yu Wang, Huazhong Yang, and Hui Wang, "Signal-Path Assignment for Dual-V_t Technique," *PhD Research in Microelectronics and Electronics Conference*, 2005, pp. 78-81.

[11] Xiaoyong Tang, Hai Zhou, and Prith Banerjee, "Leakage Power Optimization with Dual-V_{th} Library in High-Level Synthesis," *Design Automation Conference*, 2005, pp. 202-207.

Table I : Experimental results for benchmark circuits

Benchmark circuit	Total cells	Flip/ Flops	Leakage power (nW)			$\frac{\text{Tech2 - Tech3}}{\text{Tech2}}$ (%)	$\frac{\text{Tech1 - Tech3}}{\text{Tech1}}$ (%)	$\frac{\text{LVT - Tech3}}{\text{LVT}}$ (%)	CPU time (s)
			Power Compiler	Our algorithm					
			Tech1	Tech2	Tech3				
C5315	1548	0	28027.0	30304.7	26844.1	11.419	4.221	65.044	4
C6288	3216	0	179200.5	172832.1	148314.3	14.186	17.236	36.7	16
C7552	2046	0	50508.7	49714.9	44261.6	10.969	12.368	57.97	4
S386	93	6	2611.7	1777.3	1548.0	12.902	40.728	72.207	3
S420	176	16	7652.3	6879.0	6144.4	10.679	19.705	54.881	3
S838	386	32	14593.0	13973.7	12793.5	8.446	12.331	51.259	3
S1488	483	6	10042.0	11126.7	9847.9	11.493	1.933	58.525	3
S5378	1224	176	40030.7	41174.3	37635.2	8.595	5.984	65.78	3
S9234	1004	145	41813.9	40377.3	36965.1	8.451	11.596	61.063	3
S15850	3194	513	41980.2	44933.7	41698.5	7.2	0.671	84.470	5
S35932	8750	1728	435177.2	413218.0	373030.2	9.726	14.281	59.528	30
S38584	9545	1275	89070.7	87050.9	81782.7	6.052	8.182	88.028	16
GUC1	13363	992	346731.2	292565.3	254381.5	13.051	26.634	72.832	41
GUC2	17639	384	527158.8	464455.0	402883.1	13.257	23.575	59.725	269
GUC3	32286	384	634412.1	486355.8	431168.3	11.347	32.037	71.916	536
Average	–	–	–	–	–	10.518	15.432	63.995	–