# รายการอ้างอิง


1. Pedram M. Power Minimization in IC Design : Principles and Applications. ACM Transactions on Design Automation of Electronic Systems. 1 (January 1996): 3-56.
2. Yeo K. S. and Roy K. Low-Voltage, Low-Power VLSI Subsystems. USA: McGraw-Hill, 2005.
3. Usami K. and Igarashi M. Low-Power Design Methodology and Applications utilizing Dual Supply Voltages. IEEE Asia and South Pacific Design Automation Conference. (January 2000): 123-128.
4. Sundararajan V. and Parhi K. K. Synthesis of Low Power CMOS VLSI Circuits using Dual Supply Voltages. IEEE Design Automation Conference. (June 1999): 72-75.
5. Angel E. D. and Swartzlander E. E. Low Power Parallel Multipliers. IEEE VLSI Signal Processing. (November 1996): 199-208.
6. Goldovsky A. and Burns G. Design and Implementation of a 16 by 16 Low-Power Two's Complement Multiplier. IEEE International Symposium on Circuits and Systems. (May 2000): 345-348.
7. Wang Y.; Jiang Y.; and Sha E. On Area-Efficient Low Power Array Multipliers. IEEE International Conference on Electronics, Circuits and Systems. 3 (September 2001): 1429-1432.
8. Huang Z. and Ercegovac M. D. High-Performance Low-Power Left-to-Right Array Multiplier Design. IEEE Transactions on Computers. 3 (March 2005): 272-283.
9. Rabaey J. Digital Integrated Circuits: A Design Perspective. 2nd ed. USA: Prentice-Hall, 2003.
10. Goel S. and Bayoumi M. On the Design of Low-Energy Hybrid CMOS 1-Bit Full Adder Cells. IEEE International Midwest Symposium on Circuits and Systems. (May 2000): 345-348.
11. Zhuang N. and Wu H. A New Design of the CMOS Full Adder. IEEE Journal of Solid-State Circuits. 5 (May 1992): 840-844.

12. Wey I. C.; Huang C. H.; and Chow H. C. A New Low-Voltage CMOS 1-Bit Full Adder for High Performance Applications. IEEE Asia-Pacific Conference. (August 2002): 21-24.

13. Bui H. L.; Wang Y.; and Jiang Y. Design and Analysis of Low-Power 10-Transistor Full Adders Using Novel XOR-XNOR Gates. IEEE Transactions on Circuits and Systems-II: Analog and Digital Signal Processing. 1 (January 2002): 25-30.

14. Chang C. H.; Zhang M.; and Gu J. A Novel Low Power Low Voltage Full Adder Cell. IEEE Proceedings of the 3rd International Symposium on Image and Signal Processing and Analysis. (September 2003): 454-458.

15. Shama E. A. and Bayoumi M. A. A New Cell for Low Power Adders. IEEE International Symposium on Circuits and Systems. (May 1996): 49-52.

16. Shams A. M. and Bayoumi M. A. A New Full Adder Cell for Low-Power Applications. IEEE Proceedings of the 8th Great Lakes Symposium on VLSI. (February 1998): 45–49.

17. Shams A. M. and Bayoumi M. A. Performance Analysis of Low-Power 1-Bit CMOS Full Adder Cells. IEEE Transactions on Very Large Scale Integration (VLSI) Systems. 1 (February 2002): 20-29.

18. Soudris D.; Piguet C.; and Goutis C. Designing CMOS Circuits for Low Power. USA: Kluwer academic publishers, 2002.

19. Uehara T. and Vancleemput W. M. Optimal Layout Of CMOS Functional Arrays. IEEE Transactions on Computers. 5 (May 1981): 305-312.

20. Igarashi M.; Usami K.; and Hatanaka N. A Low-Power Design Method using Multiple Supply Voltages. IEEE International Symposium on Low Power Electronics and Design. (August 1997): 36-41.

21. Yeh C.; Kang Y. S.; and Wang J. S. Layout Techniques Supporting the Use of Dual Supply Voltages for Cell-Based Designs. IEEE Design Automation Conference. (June 1999): 62-67.

# ภาคผนวก

# ภาคผนวก ก.

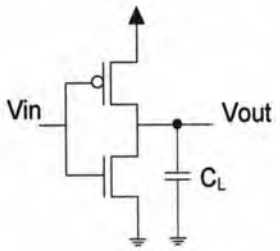
## การคำนวณค่าตัวเก็บประจุภาระ

ในการคำนวณโหลดของอินเวอร์เตอร์แสดงดังนี้ รูปที่ ก-1 คือ วงจรอินเวอร์เตอร์มี $C_L$ เป็นตัวเก็บประจุภาระ (Capacitance load) สามารถเขียนสมการของตัวเก็บประจุภาระได้ดังนี้

$$\begin{aligned} C_L &= Cgate(NMOS) + Cgate(PMOS) \\ &= (C_{GSOn} + C_{GDOn} + W_n L_n Cox) \\ &\quad + (C_{GSOp} + C_{GDOp} + W_p L_p Cox) \end{aligned} \tag{ก-1}$$

ซึ่ง $C_{GDO} = C_{GSO} = CoW$ (ก-2)

$$C_L = (2Co_n W_n + W_n L_n Cox) + (2Co_p W_p + W_p L_p Cox) \tag{ก-3}$$




*รูปที่ 3-6:วงจรอินเวอร์เตอร์*

ซึ่ง $C_{GSO}$ = ค่าเก็บประจุเกต-ซอร์สแบบซ้อนเหลื่อมต่อหน่วยความยาว (Gate-to-Source Overlap Capacitance per channel length) ขึ้นกับกระบวนการผลิต

$C_{GDO}$ = ค่าเก็บประจุเกต-เดรนแบบซ้อนเหลื่อมต่อหน่วยความยาว (Gate-to-Drain Overlap Capacitance per channel length) ขึ้นกับกระบวนการผลิต

$C_O$ = ค่าเก็บประจุแบบซ้อนเหลื่อมต่อหน่วยความยาว (Overlap Capacitance per channel length) ขึ้นกับกระบวนการผลิต

$C_{OX}$ = ค่าเก็บประจุออกไซด์ต่อหน่วยพื้นที่ (Gate oxide Capacitance per unit area) ขึ้นกับกระบวนการผลิต

โดยที่ค่าเก็บประจุออกไซด์ ($C_{ox}$) เป็นอัตราส่วนระหว่างสภาพยอมของชั้นออกไซด์ ($\varepsilon_{ox}$) และความหนาของชั้นเกต-ออกไซด์ ($t_{ox}$) ซึ่งมีค่าขึ้นกับกระบวนการผลิตเท่ากับ $5.8\times10^{-9}$ เมตร แสดงดังสมการที่ (ก-4)

$$C_{ox} = \frac{\varepsilon_{ox}}{t_{ox}} = \frac{3.97\varepsilon_0}{t_{ox}} = \frac{3.97 \times 8.85 \times 10^{-12}\ F/m}{5.8 \times 10^{-9}\ m} = 6.058 \times 10^{-3}\ \ F/m^2 \qquad \text{(ก-4)}$$

จากสมการที่ (ก-3) แทนค่าตัวแปรต่างๆในสมการซึ่งดูได้จากกระบวนการผลิต TSMC 0.25 microns ในภาคผนวก (ข) ซึ่งค่า $Co_n$ เท่ากับ $3.11 \times 10^{-10}$ F/m และ $Co_p$ มีค่าเท่ากับ $2.68 \times 10^{-10}$ F/m โดยให้ขนาดของทรานซิสเตอร์ในอินเวอร์เตอร์มีขนาดเล็กสุด (Minimum size) คือ $W_p = 2$ µm และ $W_n = 1$ µm จะได้ว่าค่าเก็บประจุภาระมีค่าเท่ากับ 6.24 fF

โดยสามารถประมาณค่า C1 และ C2 ได้ดังนี้ ซึ่ง C1 เป็นภาระของ Sum ซึ่ง Sum จะไปต่อกับอินพุท A หรือ B ของเซลล์วงจรบวกตัวถัดไป จะได้ว่า Sum จะต้องขับเกตประมาณ 4 เกต หรือประมาณเท่ากับอินเวอร์เตอร์ 2 ตัว ซึ่งแต่ละเกตมีขนาดใหญ่กว่าขนาดของเกตในอินเวอร์เตอร์ขนาดเล็กที่สุดประมาณ 3 เท่า ดังนั้น C1 = 2 x 3 x $C_L$ = 37.44 fF ส่วน C2 เป็นภาระของ Cout ซึ่ง Cout จะไปต่อกับอินพุท Cin ของเซลล์วงจรบวกตัวถัดไป การประมาณค่า C2 จะคล้ายกันกับการประมาณค่า C1 ซึ่ง C2 และค่าภาระของวงจรบวก ที่ $V_{DD}$ = 2.5 V และ $V_{DD}$ = 1.8 V แสดงในตารางที่ ก-1

*ตารางที่ ก-1 : ค่าภาระที่แรงดันต่างๆ*

| VDD (V) | C1 (fF) | C2 (fF) |
|---|---|---|
| 2.5 | 24.96 | 37.44 |
| 3.3 | 37.44 | 56.16 |
| 1.8 | 18.60 | 27.90 |

# ภาคผนวก ข.

# รายละเอียดกระบวนการผลิต TSMC 0.25 microns

**CORNER_LIB OF TYPICAL MODEL**

```
*.LIB TT
.param toxp = 5.8e-9 toxn = 5.8e-9
+dxl = 0 dxw = 0
+dvthn = 0 dvthp = 0
+cjn = 2.024128E-3    cjp = 1.931092e-3
+cjswn = 2.751528E-10  cjswp = 2.232277e-10
+cgon =  3.11E-10 cgop = 2.68e-10
+cjgaten = 2.135064E-10 cjgatep = 1.607088e-10
+hdifn = 3.1e-07 hdifp = 3.1e-7
*.lib '<ModelFile>' MOS
*.ENDL TT
```

**NMOS DEVICES MODEL**

```
.MODEL nmos              NMOS (          LEVEL  = 49
+VERSION = 3.1                  LMIN   = 2.4E-07          LMAX   = '5.1E-07-dxl'
+LEVEL   = 49                   TNOM   = 25               XL     = '3E-8 + dxl'
+XW      = '0 + dxw'            VERSION = 3.1             TOX    = toxn
+CALCACM = 1                    SFVTFLAG= 0               VFBFLAG = 1
+XJ      = 1E-07                NCH    = 2.354946E+17     LLN    = 1
+LWN     = 1                    WLN    = 1               WWN    = 1
+LINT    = 1.76E-08             WINT   = 6.75E-09        MOBMOD = 1
+BINUNIT = 2                    DWG    = 0               DWB    = 0
+VTH0    = '0.4321336+dvthn'    LVTH0  = 2.081814E-08     WVTH0  = -5.470342E-11
+PVTH0   = -6.721795E-16        K1     = 0.3281252       LK1    = 9.238362E-08
+WK1     = 2.878255E-08         PK1    = -2.426481E-14   K2     = 0.0402824
+LK2     = -3.208392E-08        WK2    = -1.154091E-08   PK2    = 9.192045E-15
+K3      = 0                    DVT0   = 0               DVT1   = 0
+DVT2    = 0                    DVT0W  = 0               DVT1W  = 0
+DVT2W   = 0                    NLX    = 0               W0     = 0
+K3B     = 0                    VSAT   = 7.586954E+04    LVSAT  = 3.094656E-03
+WVSAT   = -1.747416E-03        PVSAT  = 8.820956E-10    UA     = 8.924498E-10
+LUA     = -1.511745E-16        WUA    = -3.509821E-17   PUA    = -3.08778E-23
+UB      = 8.928832E-21         LUB    = -1.655745E-27   WUB    = -2.03282E-27
```

+PUB = 3.4578E-34
+WUC = -1.256705E-18
+PRWB = 0
+U0 = 0.06005258
+PU0 = 3.57209E-15
+WA0 = 5.512414E-07
+LKETA = 2.360844E-08
+A1 = 0.02659908
+AGS = -4.01637
+PAGS = -3.108579E-13
+VOFF = -0.1829426
+PVOFF = -2.832958E-15
+WNFACTOR= 1.501016E-07
+LCIT = -1.076934E-10
+CDSC = 0
+ETA0 = 1.215578E-04
+PETA0 = 1.529658E-17
+WETAB = -6.897268E-10
+PCLM = 3.583838
+PPCLM = -1.33176E-15
+LPDIBLC2= 7.808481E-09
+PDIBLCB = -0.1229374
+PPDIBLCB= -9.624918E-15
+LPSCBE1 = 28.64041
+PSCBE2 = 1.842585E-06
+PPSCBE2 = -1.301972E-18
+WPVAG = 3.07622E-10
+LDELTA = 1.492454E-08
+ALPHA0 = 0
+LKT1 = -1.664895E-08
+KT2 = -0.05347481
+PKT2 = -1.240924E-15
+WAT = 6.829220E-04
+LUTE = 1.662427E-07
+UA1 = -3.461758E-10
+PUA1 = 7.171442E-23
+LUC1 = -2.895726E-18
+KT1L = 0
+MJ = 0.4960069

UC = -1.364265E-11
PUC = -6.249644E-24
PRWG = 0
LU0 = -6.31976E-09
A0 = -1.468837
PA0 = -9.222928E-14
WKETA = 1.560385E-08
LA1 = -6.511454E-09
LAGS = 1.090294E-06
B0 = 0
LVOFF = 9.941631E-09
NFACTOR = 0.6790636
PNFACTOR= -2.955591E-14
WCIT = -3.286884E-10
CDSCB = 0
LETA0 = -1.037758E-11
ETAB = 3.548681E-03
PETAB = 3.481742E-16
LPCLM = -6.874146E-07
PDIBLC1 = 0
WPDIBLC2= 5.516945E-10
LPDIBLCB= 4.956215E-08
DROUT = 0
WPSCBE1 = 15.7154
LPSCBE2 = 2.871008E-12
PVAG = -2.015254E-03
PPVAG = -1.55418E-16
WDELTA = -6.71663E-09
BETA0 = 30
WKT1 = -1.633463E-08
LKT2 = 8.244731E-09
AT = -1.132632E+04
PAT = -4.154249E-10
WUTE = 1.244801E-07
LUA1 = 1.747495E-16
UB1 = 0
WUC1 = -1.990052E-17
PRT = -1E-18
PB = 0.9173808

LUC = 1.170473E-17
RDSW = 447.8871
WR = 0.99
WU0 = -8.819531E-09
LA0 = 6.419548E-07
KETA = -0.04922795
PKETA = -5.98377E-15
A2 = 1
WAGS = 1.162021E-06
B1 = 0
WVOFF = 1.568082E-08
LNFACTOR= 3.454948E-08
CIT = 2.218499E-04
PCIT = 1.658928E-16
CDSCD = 0
WETA0 = -3.030225E-11
LETAB = -1.791374E-09
DSUB = 0
WPCLM = 5.664574E-08
PDIBLC2 = 5.379674E-03
PPDIBLC2= -2.784957E-16
WPDIBLCB= 3.299946E-08
PSCBE1 = 4.472639E+08
PPSCBE1 = -7.933138E-06
WPSCBE2 = 2.579183E-12
LPVAG = 1.017757E-09
DELTA = -0.02862256
PDELTA = 3.407521E-15
KT1 = -0.2579945
PKT1 = 3.755864E-15
WKT2 = 1.13705E-09
LAT = 6.469047E-03
UTE = -2.309089
PUTE = -5.627924E-14
WUA1 = -1.42065E-16
UC1 = -2.38157E-12
PUC1 = 1.004497E-23
CJ = cjn
CJSW = cjswn

```
+MJSW   = 0.443145      PBSW   = 0.9173808      CJSWG  = cjgaten
+MJSWG  = 0.443145      PBSWG  = 0.9173808      HDIF   = hdifn
+RS     = 0             RD     = 0
+ACM    = 12            LDIF   = 1.2E-07        RSH    = 4.5
+CTA    = 7.707813E-04  CTP    = 5.512283E-04  PTA    = 1.167715E-03
+PTP    = 1.167715E-03  N      = 1             XTI    = 3
+CGDO   = 'cgon'        CGSO   = 'cgon'        CAPMOD = 0
+NQSMOD = 0             XPART  = 1             CF     = 0
+TLEV   = 1             TLEVC  = 1             JS     = 1E-06
+JSW    = 5E-11         )
```

**PMOS DEVICES MODEL**

```
.MODEL pmos     PMOS (          LEVEL  = 49
+VERSION = 3.1  LMIN   = 2.4E-7         LMAX   = '5.0E-7-dxl'
+XL     = '3e-8+dxl'
+XW     = '0+dxw'       TNOM   = 25             TOX    = toxp
+CALCACM = 1            SFVTFLAG= 0             VFBFLAG = 1
+XJ     = 1E-7          NCH    = 4.1589E17
+LLN    = 1             LWN    = 1              WLN    = 1
+WWN    = 1             LINT   = 1.2365E-8      WINT   = 7.8E-9
+MOBMOD = 1             BINUNIT = 2             DWG    = 0
+DWB    = 0             VTH0   = 'dvthp-0.6236538'  LVTH0  = 2.649834E-8
+WVTH0  = 3.214189E-8   PVTH0  = -3.22268E-15   K1     = 0.4198155
+LK1    = 5.770498E-8   WK1    = 5.577151E-8   PK1    = -2.81684E-14
+K2     = 0.0429467     LK2    = -2.296405E-8  WK2    = -1.355302E-8
+PK2    = 6.848271E-15  K3     = 0             DVT0   = 0
+DVT1   = 0             DVT2   = 0             DVT0W  = 0
+DVT1W  = 0             DVT2W  = 0             NLX    = 0
+W0     = 0             K3B    = 0             VSAT   = 1.443912E5
+LVSAT  = -7.688012E-4  WVSAT  = -6.083648E-3  PVSAT  = 2.186471E-10
+UA     = 1.846811E-9   LUA    = -3.27694E-16  WUA    = -2.82106E-16
+PUA    = 7.180233E-23  UB     = -7.84535E-19  LUB    = 4.772849E-25
+WUB    = 2.599205E-25  PUB    = -1.46530E-31  UC     = -1.75560E-10
+LUC    = 3.360832E-17  WUC    = 1.504425E-17  PUC    = -1.30556E-23
+RDSW   = 1.03E3        PRWB   = 0             PRWG   = 0
+WR     = 1             U0     = 0.0136443     LU0    = -7.22084E-10
+WU0    = -1.088554E-9  PU0    = 2.730854E-16  A0     = 0.1071803
+LA0    = 4.64252E-7    WA0    = 5.383179E-7   PA0    = -1.32033E-13
+KETA   = -4.943762E-3  LKETA  = -3.565304E-9  WKETA  = -5.226247E-9
```

```
+PKETA  = 2.640665E-15
+AGS    = 0.1664005
+PAGS   = -2.67304E-14
+VOFF   = -0.0592623
+PVOFF  = 7.510321E-15
+WNFACTOR= 1.210664E-8
+LCIT   = 2.916437E-10
+CDSC   = 0
+ETA0   = -3.819468E-3
+PETA0  = -4.16037E-16
+WETAB  = 5.284657E-11
+PCLM   = 0.1098002
+PPCLM  = -1.97766E-13
+LPDIBLC2= -1.81964E-9
+PDIBLCB = 0.1921199
+PSCBE2 = 1E-20
+ALPHA0 = 0
+LKT1   = -1.160393E-8
+KT2    = -0.0367632
+UA1    = 3.992421E-10
+WUB1   = -6.13069E-26
+KT1L   = 0
+MJ     = 0.4812153
+MJSW   = 0.3237595
+MJSWG  = 0.3237595
+LDIF   = 1.2E-7
+RD     = 0
+CTP    = 4.30175E-4
+CGDO   = cgop
+CAPMOD = 0
+CF     = 0
+TLEV   = 1
+JSW    = 5E-12        )
*.ENDL MOS
```

```
A1    = 0
LAGS  = 1.19106E-7
B0    = 0
LVOFF = -1.96686E-8
NFACTOR = 0.8588103
PNFACTOR= -6.11712E-15
WCIT  = -3.11284E-11
CDSCB = 0
LETA0 = 2.155422E-9
ETAB  = 1.334637E-3
PETAB = -2.68353E-17
LPCLM = 6.874263E-7
PDIBLC1 = 0
WPDIBLC2= -5.853396E-9
DROUT = 0
PVAG  = 0
BETA0 = 30
WKT1  = 4.153356E-8
AT    = 1E4
UB1   = -9.23294E-19
PUB1  = 1.503674E-32
PRT   = 0
PB    = 0.9134669
PBSW  = 0.9134669
PBSWG = 0.9134669
ACM   = 12
RSH   = 3.5
PTA   = 1.3004E-3
CGSO  = cgop
NQSMOD = 0
N     = 1
TLEVC = 1
```

```
A2    = 0.4
WAGS  = 5.29237E-8
B1    = 0
WVOFF = -1.486398E-8
LNFACTOR= -1.158881E-7
CIT   = 6.439495E-5
PCIT  = 1.572825E-17
CDSCD = 0
WETA0 = 8.235612E-10
LETAB = -7.93631E-10
DSUB  = 0
WPCLM = 6.724724E-7
PDIBLC2 = 5.801323E-3
PPDIBLC2= 2.957545E-15
PSCBE1 = 7.19E8
DELTA = 0.01
KT1   = -0.3248987
PKT1  = -4.62347E-15
UTE   = -1.04
LUB1  = -5.28718E-26
UC1   = -1.00699E-12
CJ    = cjp
CJSW  = cjswp
CJSWG = cjgatep
HDIF  = hdifp
RS    = 0
CTA   = 8.3043E-4
PTP   = 1.3004E-3

XPART = 1
XTI   = 3
JS    = 3E-7
```

# ภาคผนวก ค.

# การหาวิถีวิกฤติของวงจรคูณ

**1. การหาวิถีวิกฤติของวงจรคูณแบบโครงสร้างต้นไม้ ขนาด 4 X 4 บิต ด้วยภาษา VHDL ซึ่งแบ่งเป็น 5 ส่วน คือ**

**1.1 สัญญาณด้านเข้า**

ส่วนของสัญญาณด้านเข้านี้เป็นการเขียนโปรแกรมเพื่อสร้างรูปแบบสัญญาณด้านเข้า (input pattern) ที่เป็นไปได้ให้กับวงจรคูณแบบโครงสร้างต้นไม้ ขนาด 4 X 4 บิต

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use std.textio.ALL;                    -- TextIO package

entity tree4_pattern is                -- input pattern of tree multiplier 4 bits
end tree4_pattern;

architecture Behavioral of tree4_pattern is
signal abus,bbus,abus1,bbus1,abus2,bbus2  : std_logic_vector (3 downto 0) := (others => '0');
signal z1,z2   : std_logic_vector (7 downto 0) := (others => '0');
signal sel, z3   : std_logic := '0';
constant period : time := 40 ns ;
signal t1,t2,t3 :time := 0 ps;
signal temp1,temp2,temp3,temp4 : bit_vector (3 downto 0) := (others => '0');
signal temp5 : bit_vector (7 downto 0) := (others => '0');
signal temp6 : time;
FILE in_out : text is out "c:\tree4_pattern.txt";   -- keep output file in tree4_pattern.txt
        component tree4_delay          -- component for assign delay in tree multiplier 4 bits
                Port ( x, y : in std_logic_vector(3 downto 0);
                   z1, z2 : out std_logic_vector(7 downto 0));
        end component;
begin
stage0 : tree4_delay port map (abus, bbus, z1, z2);
sel <= not sel after period/2;
abus <= abus1 when sel <= '0' else abus2;
bbus <= bbus1 when sel <= '0' else bbus2;
z3 <= not((z1(0)or z1(1))or (z1(2)or z1(3))or (z1(4)or z1(5))or (z1(6)or z1(7)));

process                -- process for write output file
variable L :Line;      -- stores a complete line
begin
        write(L,string'("abus1"));
        write(L,' ');
        write(L,string'("bbus1"));
        write(L,' ');
        write(L,string'("abus2"));
        write(L,' ');
        write(L,string'("bbus2"));
        write(L,string'("   "));
        write(L,string'("z2"));
        write(L,string'("   "));
        write(L,string'("T1"));
        write(L,string'("   "));
        write(L,string'("T2"));
        write(L,string'("   "));
        write(L,string'("T3"));
        writeline(in_out,L);
        for i in 0 to 16 loop          -- create input pattern for tree multiplier 4 bits
        abus1 <= conv_std_logic_vector(i, abus1'length);
```

```
for j in 0 to 16 loop
bbus1 <= conv_std_logic_vector(j, abus1'length);
        for m in 0 to 16 loop
        abus2 <= conv_std_logic_vector(m, abus2'length);
                for n in 0 to 16 loop
                bbus2 <= conv_std_logic_vector(n, bbus2'length);

                                write(L,temp1);
                                write(L,string'("  "));
                                write(L,temp2);
                                write(L,string'("  "));
                                write(L,temp3);
                                write(L,string'("  "));
                                write(L,temp4);
                                write(L,string'("  "));
                                write(L,temp5);
                                write(L,string'("  "));
                                write(L,temp6);
                                write(L,string'("  "));
                                write(L,t2);
                                write(L,string'("  "));
                                write(L,t3);
                                writeline(in_out,L);
                                wait for 40 ns;
                        end loop;
                end loop;
                wait for 40 ns;
        end loop;
end loop;
```

```
end process;

process(sel)            -- wait signal sel
begin
        if sel = '1' then
                t2 <= t1;
                temp1 <=To_bitvector(abus1);
                temp2 <=To_bitvector(bbus1);
                temp3 <=To_bitvector(abus2);
                temp4 <=To_bitvector(bbus2);
        end if;
end process ;

process(z3)             -- wait signal z3
begin
        if sel = '0' then
                t3 <= 0 ns;
        elsif z3'event and z3 = '1' then
                t3 <= t1-t2; -- difference of time
        end if;
        temp6 <= t1;
        temp5 <=To_bitvector(z2);
end process;

process
begin
        while TRUE loop
                t1 <= t1 + 100 ps;
                wait for 100 ps;
        end loop;
```

```
        end process ;
end Behavioral;
```

### 1.2 ค่าความหน่วง

ในส่วนนี้เป็นการเขียนโปรแกรมเพื่อกำหนดค่าความหน่วงให้กับวงจรบวกภายในวงจรคูณ

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity tree4_delay is                    -- assign delay in tree multiplier 4 bits
  Port ( x, y : in std_logic_vector(3 downto 0);
       z1, z2 : out std_logic_vector(7 downto 0));
end tree4_delay;

architecture Behavioral of tree4_delay is
  component tree4                        -- component of tree multiplier 4 bits
  generic (unit_delay : time );
           port ( x, y : in std_logic_vector(3 downto 0);
                  z    : out std_logic_vector(7 downto 0));
  end component ;

  signal z3,z4 : std_logic_vector(7 downto 0);

begin
        stage0 : tree4
                 generic map (unit_delay => 1 ns)    -- assign first delay
                 port map ( x, y, z3);
        stage1 : tree4
                 generic map (unit_delay => 0 ns)    -- assign second delay
                 port map (x, y, z4);
                 z1 <= z3 xor z4;                    -- compare output signal of tree multiplier 4 bits
                 z2 <= z4;
end Behavioral;
```

### 1.3 วงจรคูณแบบโครงสร้างต้นไม้ ขนาด 4 X 4 บิต

ในส่วนนี้เป็นการเขียนโปรแกรมเพื่อสร้างวงจรคูณจากผลคูณย่อย (partial product) ที่เชื่อมต่อกับวงจรบวก

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity tree4 is                  -- structure of tree multiplier 4 bits
        generic (unit_delay : time );
  Port (  x,y : in std_logic_vector(3 downto 0);
          z : out std_logic_vector(7 downto 0));
end tree4;

architecture Behavioral of tree4 is
        signal c : std_logic_vector(12 downto 1) := (others => '0');
        signal s : std_logic_vector (12 downto 0) := (others => '0');
        signal p0,p1,p2,p3 : std_logic_vector(3 downto 0);
        signal Cin : std_logic := '0';
        component full_adder             -- component of full adder
                generic ( unit_delay : time);
                port (Cin,a,b  : in std_logic;
                sum, Cout   : out std_logic);
        end component ;
```

```
component and_gate                    -- component of AND gate
        generic ( unit_delay : time);
        port (x ,y : in std_logic;
              pp   : out std_logic );
end component ;

begin

-- generate partial product
g0 : for i in 0 to 3 generate
        stage : and_gate  generic map (unit_delay => unit_delay)
                          port map (x(i), y(0), p0(i));
end generate g0;
g1 : for i in 0 to 3 generate
        stage : and_gate  generic map (unit_delay => unit_delay)
                          port map (x(i), y(1), p1(i));
end generate g1;
g2 : for i in 0 to 3 generate
        stage : and_gate  generic map (unit_delay => unit_delay)
                          port map (x(i), y(2), p2(i));
end generate g2;
g3 : for i in 0 to 3 generate
        stage : and_gate  generic map (unit_delay => unit_delay)
                          port map (x(i), y(3), p3(i));
end generate g3;

-- connect full adder cells
stage1 : full_adder  generic map (unit_delay => unit_delay)
                     port map (Cin, p0(3), p1(2), s(1), c(1));
stage2 : full_adder  generic map (unit_delay => unit_delay)
                     port map (Cin, p2(2), p3(1), s(2), c(2));
stage3 : full_adder  generic map (unit_delay => unit_delay)
                     port map (p0(2), p1(1), p2(0), s(3), c(3));
stage4 : full_adder  generic map (unit_delay => unit_delay)
                     port map (p2(1), p3(0), s(1), s(4), c(4));
stage5 : full_adder  generic map (unit_delay => unit_delay)
                     port map (p1(3), s(2), c(1), s(5), c(5));
stage6 : full_adder  generic map (unit_delay => unit_delay)
                     port map (p2(3), p3(2), c(2), s(6), c(6));
stage7 : full_adder  generic map (unit_delay => unit_delay)
                     port map (Cin, p0(1), p1(0), s(7), c(7));
stage8 : full_adder  generic map (unit_delay => unit_delay)
                     port map (Cin, s(3), c(7), s(8), c(8));
stage9 : full_adder  generic map (unit_delay => unit_delay)
                     port map (c(8), s(4), c(3), s(9), c(9));
stage10 : full_adder  generic map (unit_delay => unit_delay)
                     port map (c(9), s(5), c(4), s(10), c(10));
stage11 : full_adder  generic map (unit_delay => unit_delay)
                     port map (c(10), s(6), c(5), s(11), c(11));
stage12 : full_adder  generic map (unit_delay => unit_delay)
                     port map (p3(3), c(11), c(6), s(12), c(12));

z(0)<= p0(0);
z(1)<= s(7);
z(2)<= s(8);
z(3)<= s(9);
z(4)<= s(10);
z(5)<= s(11);
z(6)<= s(12);
```

```
        z(7)<= c(12);
end Behavioral;
```

### 1.4 วงจรบวก

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity full_adder is                    -- structure of full adder
        generic (unit_delay : time );
    Port ( a, b, Cin : in std_logic;
        sum, Cout  : out std_logic);
end full_adder;

architecture Behavioral of full_adder is
begin
        sum <= a xor b xor Cin after unit_delay;
        Cout <= (a and b) or (a and Cin) or (b and Cin)after unit_delay;
end Behavioral;
```

### 1.5 AND Gate

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity and_gate is                      -- structure of  AND gate
        generic (unit_delay : time );
    Port ( x, y : in std_logic ;
        pp : out std_logic);
end and_gate ;

architecture Behavioral of and_gate is
begin
        pp <= x and y after unit_delay;
end Behavioral;
```

**2. การหาวิถีวิกฤติของวงจรคูณแบบโครงสร้างต้นไม้ ขนาด 8 X 8 บิต ด้วยภาษา VHDL ซึ่งแบ่งเป็น 5 ส่วน คือ**

**2.1 สัญญาณด้านเข้า**

ส่วนของสัญญาณด้านเข้านี้เป็นการเขียนโปรแกรมเพื่อสร้างรูปแบบสัญญาณด้านเข้า (input pattern) ที่เป็นไปได้ให้กับวงจรคูณแบบโครงสร้างต้นไม้ ขนาด 4 X 4 บิต

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use std.textio.ALL;                    -- TextIO package

entity tree8_pattern is                -- input pattern of tree multiplier 8 bits
end tree8_pattern;

architecture Behavioral of tree8_pattern is
signal abus,bbus,abus1,bbus1,abus2,bbus2  : std_logic_vector (7 downto 0) := (others => '0');
signal z1,z2   : std_logic_vector (15 downto 0) := (others => '0');
signal sel, z3   : std_logic := '0';
constant period : time := 100 ns ;
signal t1,t2,t3 :time := 0 ps;
signal temp1,temp2,temp3,temp4 : bit_vector (7 downto 0) := (others => '0');
signal temp5 : bit_vector (15 downto 0) := (others => '0');
signal temp6 : time;
FILE in_out : text is out "c:\tree8_pattern.txt";   -- keep output file in tree8_pattern.txt
        component tree8_delay              -- component for assign delay in tree multiplier 8 bits
                Port ( x, y : in std_logic_vector(7 downto 0);
                  z1, z2 : out std_logic_vector(15 downto 0));
end component;
begin
stage0 : tree8_delay port map (abus, bbus, z1, z2);
sel <= not sel after period/2;
abus <= abus1 when sel <= '0' else abus2;
bbus <= bbus1 when sel <= '0' else bbus2;
z3 <= not((z1(0)or z1(1))or (z1(2)or z1(3))or (z1(4)or z1(5))or (z1(6)or z1(7))or (z1(8)or
      z1(9))or (z1(10)or z1(11))or (z1(12)or z1(13))or (z1(14)or z1(15)));

process                -- process for write output file
variable L :Line;      -- stores a complete line
begin
        write(L,string'("abus1"));
        write(L,' ');
        write(L,string'("bbus1"));
        write(L,' ');
        write(L,string'("abus2"));
        write(L,' ');
        write(L,string'("bbus2"));
        write(L,string'("   "));
        write(L,string'("z2"));
        write(L,string'("   "));
        write(L,string'("T1"));
        write(L,string'("   "));
        write(L,string'("T2"));
        write(L,string'("   "));
        write(L,string'("T3"));
        writeline(in_out,L);
        for i in 0 to 255 loop     -- create input pattern for tree multiplier 8 bits
```

```
abus1 <= conv_std_logic_vector(i, abus1'length);
for j in 0 to 255 loop
        bbus1 <= conv_std_logic_vector(j, abus1'length);
        for m in 0 to 255 loop
            abus2 <= conv_std_logic_vector(m, abus2'length);
            for n in 0 to 255 loop
                bbus2 <= conv_std_logic_vector(n, bbus2'length);
                                write(L,temp1);
                                write(L,string'("  "));
                                write(L,temp2);
                                write(L,string'("  "));
                                write(L,temp3);
                                write(L,string'("  "));
                                write(L,temp4);
                                write(L,string'("  "));
                                write(L,temp5);
                                write(L,string'("  "));
                                write(L,temp6);
                                write(L,string'("  "));
                                write(L,t2);
                                write(L,string'("  "));
                                write(L,t3);
                                writeline(in_out,L);
                                wait for 100 ns;
                        end loop;
                end loop;
                wait for 100 ns;
        end loop;
```

```
        end loop;
end process;

process(sel)            -- wait signal sel
begin
        if sel = '1' then
                t2 <= t1;
                temp1 <=To_bitvector(abus1);
                temp2 <=To_bitvector(bbus1);
                temp3 <=To_bitvector(abus2);
                temp4 <=To_bitvector(bbus2);
        end if;
end process ;

process(z3)             -- wait signal z3
begin
        if sel = '0' then
                t3 <= 0 ns;
        elsif z3'event and z3 = '1' then
                t3 <= t1-t2;            -- difference of time
        end if;
        temp6 <= t1;
        temp5 <=To_bitvector(z2);
end process;

process
begin
        while TRUE loop
                t1 <= t1 + 100 ps;
                wait for 100 ps;
```

```
                end loop;
        end process ;
end Behavioral;
```

## 2.2 ค่าความหน่วง

ในส่วนนี้เป็นการเขียนโปรแกรมเพื่อกำหนดค่าความหน่วงให้กับวงจรบวกภายในวงจรคูณ

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity tree8_delay is                   -- assign delay in tree multiplier 8 bits
  Port ( x, y : in std_logic_vector(7 downto 0);
       z1, z2 : out std_logic_vector(15 downto 0));
end tree8_delay;

architecture Behavioral of tree8_delay is
  component tree8                       -- component of tree multiplier 8 bits
  generic (unit_delay,unit_delay1 : time );
          port ( x, y : in std_logic_vector(7 downto 0);
                z  : out std_logic_vector(15 downto 0));
  end component ;

  signal z3,z4 : std_logic_vector(15 downto 0);

begin
        stage0 : tree8
                generic map (unit_delay => 1 ns)     -- assign first delay
                port map ( x, y, z3);
        stage1 : tree8
                generic map (unit_delay => 0 ns)     -- assign second delay
                port map (x, y, z4);
                z1 <= z3 xor z4;            -- compare output signal of tree multiplier 8 bits
                z2 <= z4;
end Behavioral;
```

## 2.3 วงจรคูณแบบโครงสร้างต้นไม้ ขนาด 8 X 8 บิต

ในส่วนนี้เป็นการเขียนโปรแกรมเพื่อสร้างวงจรคูณจากผลคูณย่อย (partial product) ที่เชื่อมต่อกับวงจรบวก

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity tree8 is             -- structure of tree multiplier 8 bits
        generic (unit_delay : time );
         Port (  x,y : in std_logic_vector(7 downto 0);
                 z : out std_logic_vector(15 downto 0));
end tree8;

architecture Behavioral of tree8 is
        signal c : std_logic_vector(62 downto 1) := (others => '0');
        signal s : std_logic_vector (62 downto 1) := (others => '0');
        signal p0,p1,p2,p3,p4,p5,p6,p7 : std_logic_vector(7 downto 0);
        signal Cin : std_logic := '0';

        component full_adder                -- component of full adder
                generic ( unit_delay : time);
                port (Cin,a,b  : in std_logic;
```

```
            sum, Cout   : out std_logic);
    end component ;

    component and_gate                  -- component of AND gate
            generic ( unit_delay : time);
            port (x ,y : in std_logic;
                pp    : out std_logic );
    end component ;

begin

    -- generate partial product
    g0 : for i in 0 to 7 generate
            stage : and_gate  generic map (unit_delay => unit_delay)
                    port map (x(i), y(0), p0(i));
    end generate g0;
    g1 : for i in 0 to 7 generate
            stage : and_gate  generic map (unit_delay => unit_delay)
                    port map (x(i), y(1), p1(i));
    end generate g1;
    g2 : for i in 0 to 7 generate
            stage : and_gate  generic map (unit_delay => unit_delay)
                    port map (x(i), y(2), p2(i));
    end generate g2;
    g3 : for i in 0 to 7 generate
            stage : and_gate  generic map (unit_delay => unit_delay)
                    port map (x(i), y(3), p3(i));
    end generate g3;
    g4 : for i in 0 to 7 generate
            stage : and_gate  generic map (unit_delay => unit_delay)
                    port map (x(i), y(4), p4(i));
    end generate g4;
    g5 : for i in 0 to 7 generate
            stage : and_gate  generic map (unit_delay => unit_delay)
                    port map (x(i), y(5), p5(i));
    end generate g5;
    g6 : for i in 0 to 7 generate
            stage : and_gate  generic map (unit_delay => unit_delay)
                    port map (x(i), y(6), p6(i));
    end generate g6;
    g7 : for i in 0 to 7 generate
            stage : and_gate  generic map (unit_delay => unit_delay)
                    port map (x(i), y(7), p7(i));
    end generate g7;
    -- connect full adder cells
    stage1 : full_adder  generic map (unit_delay => unit_delay)
                     port map (p1(7), p2(6), p3(5), s(1), c(1));
    stage2 : full_adder  generic map (unit_delay => unit_delay)
                    port map (p4(4), p5(3), p6(2), s(2), c(2));
    stage3 : full_adder  generic map (unit_delay => unit_delay)
                    port map (p0(6), p1(5), p2(4), s(3), c(3));
    stage4 : full_adder  generic map (unit_delay => unit_delay)
                    port map (p3(3), p4(2), p5(1), s(4), c(4));
    stage5 : full_adder  generic map (unit_delay => unit_delay)
                    port map (p0(7), p1(6), p2(5), s(5), c(5));
    stage6 : full_adder  generic map (unit_delay => unit_delay)
                    port map (p3(4), p4(3), p5(2), s(6), c(6));
    stage7 : full_adder  generic map (unit_delay => unit_delay)
```

```
                    port map (Cin, p6(1), p7(0), s(7), c(7));
stage8 : full_adder  generic map (unit_delay => unit_delay)
                    port map (Cin, s(2), s(1), s(8), c(8));
stage9 : full_adder  generic map (unit_delay => unit_delay)
                    port map (p2(7), p3(6), p4(5), s(9), c(9));
stage10 : full_adder  generic map (unit_delay => unit_delay)
                    port map (p5(4), p6(3), p7(2), s(10), c(10));
stage11 : full_adder  generic map (unit_delay => unit_delay)
                    port map (p3(7), p4(6), p5(5), s(11), c(11));
stage12 : full_adder  generic map (unit_delay => unit_delay)
                    port map (Cin, p6(4), p7(3), s(12), c(12));
stage13 : full_adder  generic map (unit_delay => unit_delay)
                    port map (p4(7), p5(6), p6(5), s(13), c(13));
stage14 : full_adder  generic map (unit_delay => unit_delay)
                    port map (p0(5), p1(4), p2(3), s(14), c(14));
stage15 : full_adder  generic map (unit_delay => unit_delay)
                    port map (p3(2), p4(1), p5(0), s(15), c(15));
stage16 : full_adder  generic map (unit_delay => unit_delay)
                    port map (Cin, s(4), s(3), s(16), c(16));
stage17 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(7), s(6), s(5), s(17), c(17));
stage18 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(8), p7(1), c(5), s(18), c(18));
stage19 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(10), s(9), c(1), s(19), c(19));
stage20 : full_adder  generic map (unit_delay => unit_delay)
                    port map (Cin, s(12), s(11), s(20), c(20));
stage21 : full_adder  generic map (unit_delay => unit_delay)
                    port map (Cin, s(13), p7(4), s(21), c(21));
stage22 : full_adder  generic map (unit_delay => unit_delay)
                    port map (p5(7), p6(6), p7(5), s(22), c(22));
stage23 : full_adder  generic map (unit_delay => unit_delay)
                    port map (p0(3), p1(2), p2(1), s(23), c(23));
stage24 : full_adder  generic map (unit_delay => unit_delay)
                    port map (p0(4), p1(3), p2(2), s(24), c(24));
stage25 : full_adder  generic map (unit_delay => unit_delay)
                    port map (Cin, p3(1), p4(0), s(25), c(25));
stage26 : full_adder  generic map (unit_delay => unit_delay)
                    port map (Cin, s(15), s(14), s(26), c(26));
stage27 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(16), p6(0), c(14), s(27), c(27));
stage28 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(17), c(3), c(4), s(28), c(28));
stage29 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(18), c(6), c(7), s(29), c(29));
stage30 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(19), c(2), c(8), s(30), c(30));
stage31 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(20), c(9), c(10), s(31), c(31));
stage32 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(21), c(11), c(12), s(32), c(32));
stage33 : full_adder  generic map (unit_delay => unit_delay)
                    port map (Cin ,s(22), c(13), s(33), c(33));
stage34 : full_adder  generic map (unit_delay => unit_delay)
                    port map (Cin, p6(7), p7(6), s(34), c(34));
stage35 : full_adder  generic map (unit_delay => unit_delay)
```

```
                    port map (p0(2), p1(1), p2(0), s(35), c(35));
stage36 : full_adder  generic map (unit_delay => unit_delay)
                    port map (Cin, s(23), p3(0), s(36), c(36));
stage37 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(25), s(24), c(23), s(37), c(37));
stage38 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(26), c(24), c(25), s(38), c(38));
stage39 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(27), c(15), c(26), s(39), c(39));
stage40 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(28), c(16), c(27), s(40), c(40));
stage41 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(29), c(17), c(28), s(41), c(41));
stage42 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(30), c(18), c(29), s(42), c(42));
stage43 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(31), c(19), c(30), s(43), c(43));
stage44 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(32), c(20), c(31), s(44), c(44));
stage45 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(33), c(21), c(32), s(45), c(45));
stage46 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(34), c(22), c(33), s(46), c(46));
stage47 : full_adder  generic map (unit_delay => unit_delay)
                    port map (Cin, p7(7), c(34), s(47), c(47));
stage48 : full_adder  generic map (unit_delay => unit_delay)
                    port map (Cin, p0(1), p1(0), s(48), c(48));
stage49 : full_adder  generic map (unit_delay => unit_delay)
                    port map (Cin, s(35), c(48), s(49), c(49));
stage50 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(36), c(35), c(49), s(50), c(50));
stage51 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(37), c(36), c(50), s(51), c(51));
stage52 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(38), c(37), c(51), s(52), c(52));
stage53 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(39), c(38), c(52), s(53), c(53));
stage54 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(40), c(39), c(53), s(54), c(54));
stage55 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(41), c(40), c(54), s(55), c(55));
stage56 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(42), c(41), c(55), s(56), c(56));
stage57 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(43), c(42), c(56), s(57), c(57));
stage58 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(44), c(43), c(57), s(58), c(58));
stage59 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(45), c(44), c(58), s(59), c(59));
stage60 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(46), c(45), c(59), s(60), c(60));
stage61 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(47), c(46), c(60), s(61), c(61));
stage62 : full_adder  generic map (unit_delay => unit_delay)
                    port map (Cin, c(47), c(61), s(62), c(62));
z(0)<= p0(0);
```

```
        z(1)<= s(48);
        z(2)<= s(49);
        z(3)<= s(50);
        z(4)<= s(51);
        z(5)<= s(52);
        z(6)<= s(53);
        z(7)<= s(54);
        z(8)<= s(55);
        z(9)<= s(56);
        z(10)<= s(57);
        z(11)<= s(58);
        z(12)<= s(59);
        z(13)<= s(60);
        z(14)<= s(61);
        z(15)<= s(62);
end Behavioral;
```

**2.4 วงจรบวก**

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity full_adder is        -- structure of full adder
        generic (unit_delay : time );
          Port ( a, b, Cin : in std_logic;
              sum, Cout : out std_logic);
end full_adder;

architecture Behavioral of full_adder is
begin
        sum <= a xor b xor Cin after unit_delay;
        Cout <= (a and b) or (a and Cin) or (b and Cin)after unit_delay;
end Behavioral;
```

**2.5 AND Gate**

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity and_gate is          -- struct of AND gate
        generic (unit_delay : time );
          Port ( x, y : in std_logic ;
              pp : out std_logic);
end and_gate ;
architecture Behavioral of and_gate is
begin
          pp <= x and y after unit_delay;
end Behavioral;
```

**3. การหาวิถีวิกฤติของวงจรคูณแบบโครงสร้างต้นไม้ ขนาด 16 X 16 บิต ด้วยภาษา VHDL ซึ่งแบ่งเป็น 5 ส่วน คือ**

**3.1 สัญญาณด้านเข้า**

ส่วนของสัญญาณด้านเข้านี้เป็นการเขียนโปรแกรมเพื่อสร้างรูปแบบสัญญาณด้านเข้า (input pattern) ที่เป็นไปได้ให้กับวงจรคูณแบบโครงสร้างต้นไม้ ขนาด 4 X 4 บิต

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use std.textio.ALL;                    -- TextIO package

entity tree16_pattern is               -- input pattern of tree multiplier 16 bits
end tree16_pattern;

architecture Behavioral of tree16_pattern is
signal abus,bbus,abus1,bbus1,abus2,bbus2  : std_logic_vector (15 downto 0) := (others => '0');
signal z1,z2   : std_logic_vector (31 downto 0) := (others => '0');
signal sel, z3   : std_logic := '0';
constant period : time := 100 ns ;
signal t1,t2,t3 :time := 0 ps;
signal temp1,temp2,temp3,temp4 : bit_vector (15 downto 0) := (others => '0');
signal temp5 : bit_vector (31 downto 0) := (others => '0');
signal temp6 : time;
FILE in_out : text is out "c:\tree16_pattern.txt";      -- keep output file in tree16_pattern.txt
        component tree16_delay                          -- component for assign delay in tree multiplier 16 bits
                Port ( x, y : in std_logic_vector(15 downto 0);
                  z1, z2 : out std_logic_vector(31 downto 0));
end component;
begin
stage0 : tree16_delay port map (abus, bbus, z1, z2);
sel <= not sel after period/2;
abus <= abus1 when sel <= '0' else abus2;
bbus <= bbus1 when sel <= '0' else bbus2;
z3 <= not((z1(0)or z1(1))or (z1(2)or z1(3))or (z1(4)or z1(5))or (z1(6)or z1(7))or (z1(8)or
      z1(9))or (z1(10)or z1(11))or (z1(12)or z1(13))or (z1(14)or z1(15))or (z1(16)or
      z1(17))or(z1(18)or z1(19))or (z1(20)or z1(21))or (z1(22)or z1(23))or (z1(24)or
      z1(25))or (z1(26)or z1(27))or (z1(28)or z1(29))or (z1(30)or z1(31)));

process                 -- process for write output file
variable L :Line;       -- stores a complete line
begin
        write(L,string'("abus1"));
        write(L,string'("       "));
        write(L,string'("bbus1"));
        write(L,string'("       "));
        write(L,string'("abus2"));
        write(L,string'("       "));
        write(L,string'("bbus2"));
        write(L,string'("              "));
        write(L,string'("z2"));
        write(L,string'("              "));
        write(L,string'("T1"));
        write(L,string'("    "));
        write(L,string'("T2"));
        write(L,string'("    "));
        write(L,string'("T3"));
```

```
writeline(in_out,L);
for i in 65535 to 65535 loop          -- create input pattern for tree multiplier 16 bits
    abus1 <= conv_std_logic_vector(i, abus1'length);
    for j in 0 to 65535 loop
        bbus1 <= conv_std_logic_vector(j, abus1'length);
        for m in 0 to 65535 loop
            abus2 <= conv_std_logic_vector(m, abus2'length);
            for n in 0 to 65535 loop
                bbus2 <= conv_std_logic_vector(n, bbus2'length);
                    write(L,temp1);
                    write(L,string'("  "));
                    write(L,temp2);
                    write(L,string'("  "));
                    write(L,temp3);
                    write(L,string'("  "));
                    write(L,temp4);
                    write(L,string'("  "));
                    write(L,temp5);
                    write(L,string'("  "));
                    write(L,temp6);
                    write(L,string'("  "));
                    write(L,t2);
                    write(L,string'("  "));
                    write(L,t3);
                    writeline(in_out,L);
                    wait for 100 ns;
                end loop;
            end loop;
                wait for 100 ns;
            end loop;
        end loop;
end process;

process(sel)              -- wait signal sel
begin
        if sel = '1' then
            t2 <= t1;
            temp1 <=To_bitvector(abus1);
            temp2 <=To_bitvector(bbus1);
            temp3 <=To_bitvector(abus2);
            temp4 <=To_bitvector(bbus2);
        end if;
end process ;

process(z3)               -- wait signal z3
begin
        if sel = '0' then
            t3 <= 0 ns;
        elsif z3'event and z3 = '1' then
            t3 <= t1-t2;          -- difference of time
        end if;
        temp6 <= t1;
        temp5 <=To_bitvector(z2);
end process;

process
begin
```

```
            while TRUE loop
                t1 <= t1 + 100 ps;
                wait for 100 ps;
            end loop;
        end process ;
end Behavioral;
```

### 3.2 ค่าความหน่วง

ในส่วนนี้เป็นการเขียนโปรแกรมเพื่อกำหนดค่าความหน่วงให้กับวงจรบวกภายในวงจรคูณ

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity tree16_delay is                  -- assign delay in tree multiplier 16 bits
  Port ( x, y : in std_logic_vector(15 downto 0);
      z1, z2 : out std_logic_vector(31 downto 0));
end tree16_delay;

architecture Behavioral of tree16_delay is
        component tree16        -- component of tree multiplier 16 bits
            generic (unit_delay : time );
            port ( x, y : in std_logic_vector(15 downto 0);
                z  : out std_logic_vector(31 downto 0));
        end component ;
        signal z3,z4 : std_logic_vector(31 downto 0);
begin
        stage0 : tree16
            generic map (unit_delay => 1 ns)            -- assign first delay
            port map ( x, y, z3);
        stage1 : tree16
            generic map (unit_delay => 0 ns)            -- assign second delay
            port map (x, y, z4);
            z1 <= z3 xor z4;            -- compare output signal of tree multiplier 16 bits
            z2 <= z4;
end Behavioral;
```

### 3.3 วงจรคูณแบบโครงสร้างต้นไม้ ขนาด 16 X 16 บิต

ในส่วนนี้เป็นการเขียนโปรแกรมเพื่อสร้างวงจรคูณจากผลคูณย่อย (partial product) ที่เชื่อมต่อกับวงจรบวก

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity tree16 is        -- structure of tree multiplier 16 bits
        generic (unit_delay,unit_delay1 : time );
          Port (  x,y : in std_logic_vector(15 downto 0);
                z : out std_logic_vector(31 downto 0));
end tree16;

architecture Behavioral of tree16 is
    signal c : std_logic_vector(255 downto 1) := (others => '0');
    signal s : std_logic_vector(255 downto 1) := (others => '0');
    signal p0,p1,p2,p3,p4,p5,p6,p7,p8,p9,p10,p11,p12,p13,p14,p15 : std_logic_vector(15 downto 0);
    signal Cin : std_logic := '0';
        component full_adder            -- component of full adder
            generic ( unit_delay : time);
```

```
            port (Cin,a,b  : in std_logic;
            sum, Cout   : out std_logic);
        end component ;

        component and_gate                  -- component of AND gate
            generic ( unit_delay : time);
            port (x ,y : in std_logic;
                pp    : out std_logic );
        end component ;
begin
        -- generate partial product
        g0 : for i in 0 to 15 generate
            stage : and_gate  generic map (unit_delay => unit_delay)
                    port map (x(i), y(0), p0(i));
        end generate g0;
        g1 : for i in 0 to 15 generate
            stage : and_gate  generic map (unit_delay => unit_delay)
                    port map (x(i), y(1), p1(i));
        end generate g1;
        g2 : for i in 0 to 15 generate
            stage : and_gate  generic map (unit_delay => unit_delay)
                    port map (x(i), y(2), p2(i));
        end generate g2;
        g3 : for i in 0 to 15 generate
            stage : and_gate  generic map (unit_delay => unit_delay)
                    port map (x(i), y(3), p3(i));
        end generate g3;
        g4 : for i in 0 to 15 generate
            stage : and_gate  generic map (unit_delay => unit_delay)
                    port map (x(i), y(4), p4(i));
        end generate g4;
        g5 : for i in 0 to 15 generate
            stage : and_gate  generic map (unit_delay => unit_delay)
                    port map (x(i), y(5), p5(i));
        end generate g5;
        g6 : for i in 0 to 15 generate
            stage : and_gate  generic map (unit_delay => unit_delay)
                    port map (x(i), y(6), p6(i));
        end generate g6;
        g7 : for i in 0 to 15 generate
            stage : and_gate  generic map (unit_delay => unit_delay)
                    port map (x(i), y(7), p7(i));
        end generate g7;
        g8 : for i in 0 to 15 generate
            stage : and_gate  generic map (unit_delay => unit_delay)
                    port map (x(i), y(8), p8(i));
        end generate g8;
        g9 : for i in 0 to 15 generate
            stage : and_gate  generic map (unit_delay => unit_delay)
                    port map (x(i), y(9), p9(i));
        end generate g9;
        g10 : for i in 0 to 15 generate
            stage : and_gate  generic map (unit_delay => unit_delay)
                    port map (x(i), y(10), p10(i));
        end generate g10;
        g11 : for i in 0 to 15 generate
            stage : and_gate  generic map (unit_delay => unit_delay)
```

```
                port map (x(i), y(11), p11(i));
end generate g11;
g12 : for i in 0 to 15 generate
        stage : and_gate  generic map (unit_delay => unit_delay)
                port map (x(i), y(12), p12(i));
end generate g12;
g13 : for i in 0 to 15 generate
        stage : and_gate  generic map (unit_delay => unit_delay)
                port map (x(i), y(13), p13(i));
end generate g13;
g14 : for i in 0 to 15 generate
        stage : and_gate  generic map (unit_delay => unit_delay)
                port map (x(i), y(14), p14(i));
end generate g14;
g15 : for i in 0 to 15 generate
        stage : and_gate  generic map (unit_delay => unit_delay)
                port map (x(i), y(15), p15(i));
end generate g15;
-- connect full adder cells
stage1 : full_adder  generic map (unit_delay => unit_delay)
                port map (p0(15),p1(14),p2(13),s(1),c(1));
stage2 : full_adder  generic map (unit_delay => unit_delay)
                port map (p3(12),p4(11),p5(10),s(2),c(2));
stage3 : full_adder  generic map (unit_delay => unit_delay)
                port map (p6(9), p7(8), p8(7), s(3),c(3));
stage4 : full_adder  generic map (unit_delay => unit_delay)
                port map (p1(15),p2(14),p3(13),s(4),c(4));
stage5 : full_adder  generic map (unit_delay => unit_delay)
                port map (p2(15),p3(14),p4(13),s(5),c(5));
stage6 : full_adder  generic map (unit_delay => unit_delay)
                port map (p5(12),p6(11),p7(10),s(6),c(6));
stage7 : full_adder  generic map (unit_delay => unit_delay)
                port map (p8(9), p9(8), p10(7),s(7),c(7));
stage8 : full_adder  generic map (unit_delay => unit_delay)
                port map (p3(15),p4(14),p5(13),s(8),c(8));
stage9 : full_adder  generic map (unit_delay => unit_delay)
                port map (p6(12),p7(11),p8(10),s(9),c(9));
stage10 : full_adder  generic map (unit_delay => unit_delay)
                port map (p0(14),p1(13),p2(12),s(10),c(10));
stage11 : full_adder  generic map (unit_delay => unit_delay)
                port map (p3(11),p4(10),p5(9), s(11),c(11));
stage12 : full_adder  generic map (unit_delay => unit_delay)
                port map (p6(8), p7(7), p8(6), s(12),c(12));
stage13 : full_adder  generic map (unit_delay => unit_delay)
                port map (s(3), s(2), s(1), s(13), c(13));
stage14 : full_adder  generic map (unit_delay => unit_delay)
                port map (s(4), c(1), c(2), s(14), c(14));
stage15 : full_adder  generic map (unit_delay => unit_delay)
                port map (p4(12),p5(11),p6(10), s(15),c(15));
stage16 : full_adder  generic map (unit_delay => unit_delay)
                port map (s(7), s(6), s(5), s(16), c(16));
stage17 : full_adder  generic map (unit_delay => unit_delay)
                port map (p11(6),p12(5),p13(4), s(17),c(17));
stage18 : full_adder  generic map (unit_delay => unit_delay)
                port map (Cin, s(9), s(8), s(18), c(18));
stage19 : full_adder  generic map (unit_delay => unit_delay)
```

```
                    port map (p4(15),p5(14),p6(13), s(19),c(19));
stage20 : full_adder  generic map (unit_delay => unit_delay)
                    port map (p7(12),p8(11),p9(10), s(20),c(20));
stage21 : full_adder  generic map (unit_delay => unit_delay)
                    port map (p0(12),p1(11),p2(10), s(21),c(21));
stage22 : full_adder  generic map (unit_delay => unit_delay)
                    port map (p3(9), p4(8), p5(7), s(22), c(22));
stage23 : full_adder  generic map (unit_delay => unit_delay)
                    port map (p6(6), p7(5), p8(4), s(23), c(23));
stage24 : full_adder  generic map (unit_delay => unit_delay)
                    port map (p0(13),p1(12),p2(11), s(24),c(24));
stage25 : full_adder  generic map (unit_delay => unit_delay)
                    port map (p3(10),p4(9), p5(8), s(25), c(25));
stage26 : full_adder  generic map (unit_delay => unit_delay)
                    port map (p6(7), p7(6), p8(5), s(26), c(26));
stage27 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(12), s(11), s(10), s(27), c(27));
stage28 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(13), c(10), c(11), s(28), c(28));
stage29 : full_adder  generic map (unit_delay => unit_delay)
                    port map (p9(6),p10(5), p11(4),s(29), c(29));
stage30 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(15), s(14), c(3), s(30), c(30));
stage31 : full_adder  generic map (unit_delay => unit_delay)
                    port map (p7(9), p8(8), p9(7), s(31), c(31));
stage32 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(17), s(16), c(4), s(32), c(32));
stage33 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(18) ,c(5), c(6), s(33), c(33));
stage34 : full_adder  generic map (unit_delay => unit_delay)
                    port map (Cin, s(20), s(19), s(34), c(34));
stage35 : full_adder  generic map (unit_delay => unit_delay)
                    port map (p5(15),p6(14),p7(13), s(35),c(35));
stage36 : full_adder  generic map (unit_delay => unit_delay)
                    port map (p8(12),p9(11),p10(10),s(36),c(36));
stage37 : full_adder  generic map (unit_delay => unit_delay)
                    port map (p6(15),p7(14),p8(13), s(37),c(37));
stage38 : full_adder  generic map (unit_delay => unit_delay)
                    port map (p9(12),p10(11),p11(10),s(38),c(38));
stage39 : full_adder  generic map (unit_delay => unit_delay)
                    port map (p12(9),p13(8),p14(7), s(39),c(39));
stage40 : full_adder  generic map (unit_delay => unit_delay)
                    port map (p0(11),p1(10),p2(9), s(40), c(40));
stage41 : full_adder  generic map (unit_delay => unit_delay)
                    port map (p3(8), p4(7), p5(6), s(41), c(41));
stage42 : full_adder  generic map (unit_delay => unit_delay)
                    port map (p6(5), p7(4), p8(3), s(42), c(42));
stage43 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(23), s(22), s(21), s(43), c(43));
stage44 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(26), s(25), s(24), s(44), c(44));
stage45 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(27), c(24), c(25), s(45), c(45));
stage46 : full_adder  generic map (unit_delay => unit_delay)
                    port map (p9(5), p10(4),p11(3),s(46), c(46));
stage47 : full_adder  generic map (unit_delay => unit_delay)
```

```
                    port map (s(29), s(28), c(12), s(47), c(47));
stage48 : full_adder  generic map (unit_delay => unit_delay)
                    port map (p12(3),p13(2),p14(1),s(48), c(48));
stage49 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(31), s(30), c(13), s(49), c(49));
stage50 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(32), c(14), c(15), s(50), c(50));
stage51 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(33), c(7), c(16), s(51), c(51));
stage52 : full_adder  generic map (unit_delay => unit_delay)
                    port map (p9(9),p10(8),p11(7),s(52), c(52));
stage53 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(34), c(8), c(9), s(53), c(53));
stage54 : full_adder  generic map (unit_delay => unit_delay)
                    port map (p10(9),p11(8),p12(7),s(54), c(54));
stage55 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(36), s(35), c(19), s(55), c(55));
stage56 : full_adder  generic map (unit_delay => unit_delay)
                    port map (p11(9),p12(8),p13(7),s(56), c(56));
stage57 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(39), s(38), s(37), s(57), c(57));
stage58 : full_adder  generic map (unit_delay => unit_delay)
                    port map (p7(15),p8(14),p9(13),s(58), c(58));
stage59 : full_adder  generic map (unit_delay => unit_delay)
                    port map (p10(12),p11(11),p12(10),s(59),c(59));
stage60 : full_adder  generic map (unit_delay => unit_delay)
                    port map (p13(9),p14(8),p15(7),s(60), c(60));
stage61 : full_adder  generic map (unit_delay => unit_delay)
                    port map (p0(9), p1(8), p2(7), s(61), c(61));
stage62 : full_adder  generic map (unit_delay => unit_delay)
                    port map (p3(6), p4(5), p5(4), s(62), c(62));
stage63 : full_adder  generic map (unit_delay => unit_delay)
                    port map (p6(3), p7(2), p8(1), s(63), c(63));
stage64 : full_adder  generic map (unit_delay => unit_delay)
                    port map (p0(10),p1(9),p2(8), s(64), c(64));
stage65 : full_adder  generic map (unit_delay => unit_delay)
                    port map (p3(7), p4(6), p5(5),s(65), c(65));
stage66 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(42), s(41), s(40),s(66), c(66));
stage67 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(43), c(40), c(41),s(67), c(67));
stage68 : full_adder  generic map (unit_delay => unit_delay)
                    port map (p9(3),p10(2),p11(1),s(68), c(68));
stage69 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(44), c(21), c(22), s(69), c(69));
stage70 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(46), s(45), c(26), s(70), c(70));
stage71 : full_adder  generic map (unit_delay => unit_delay)
                    port map (p12(2),p13(1),p14(0),s(71), c(71));
stage72 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(48), s(47), c(27), s(72), c(72));
stage73 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(49), c(28), c(29), s(73), c(73));
stage74 : full_adder  generic map (unit_delay => unit_delay)
                    port map (p10(6),p11(5),p12(4),s(74), c(74));
stage75 : full_adder  generic map (unit_delay => unit_delay)
```

```
                    port map (s(50), c(30), c(31), s(75), c(75));
stage76 : full_adder  generic map (unit_delay => unit_delay)
                    port map (Cin, p14(3), p15(2), s(76), c(76));
stage77 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(52), s(51), c(17), s(77), c(77));
stage78 : full_adder  generic map (unit_delay => unit_delay)
                    port map (p12(6),p13(5),p14(4),s(78), c(78));
stage79 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(54), s(53), c(18), s(79), c(79));
stage80 : full_adder  generic map (unit_delay => unit_delay)
                    port map (p13(6),p14(5),p15(4),s(80), c(80));
stage81 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(56), s(55), c(20), s(81), c(81));
stage82 : full_adder  generic map (unit_delay => unit_delay)
                    port map (Cin, p14(6),p15(5), s(82), c(82));
stage83 : full_adder  generic map (unit_delay => unit_delay)
                    port map (p15(6),s(57),c(35), s(83), c(83));
stage84 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(60), s(59), s(58), s(84), c(84));
stage85 : full_adder  generic map (unit_delay => unit_delay)
                    port map (p8(15),p9(14),p10(13),s(85),c(85));
stage86 : full_adder  generic map (unit_delay => unit_delay)
                    port map (p11(12),p12(11),p13(10),s(86),c(86));
stage87 : full_adder  generic map (unit_delay => unit_delay)
                    port map (p9(15),p10(14),p11(13),s(87),c(87));
stage88 : full_adder  generic map (unit_delay => unit_delay)
                    port map (p12(12),p13(11),p14(10),s(88),c(88));
stage89 : full_adder  generic map (unit_delay => unit_delay)
                    port map (p0(8), p1(7), p2(6), s(89), c(89));
stage90 : full_adder  generic map (unit_delay => unit_delay)
                    port map (p3(5), p4(4), p5(3), s(90), c(90));
stage91 : full_adder  generic map (unit_delay => unit_delay)
                    port map (p6(2), p7(1), p8(0), s(91), c(91));
stage92 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(63), s(62), s(61), s(92), c(92));
stage93 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(65), s(64), c(61), s(93), c(93));
stage94 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(66), c(64), c(65), s(94), c(94));
stage95 : full_adder  generic map (unit_delay => unit_delay)
                    port map (p9(2),p10(1),p11(0), s(95), c(95));
stage96 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(68), s(67), c(42), s(96), c(96));
stage97 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(69), c(23), c(43), s(97), c(97));
stage98 : full_adder  generic map (unit_delay => unit_delay)
                    port map (p9(4),p10(3),p11(2), s(98), c(98));
stage99 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(71), s(70), c(44), s(99), c(99));
stage100 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(72), c(45), c(46), s(100),c(100));
stage101 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(74), s(73), c(47), s(101),c(101));
stage102 : full_adder  generic map (unit_delay => unit_delay)
                    port map (p13(3),P14(2),p15(1),s(102),c(102));
stage103 : full_adder  generic map (unit_delay => unit_delay)
```

```
                    port map (s(76), s(75), c(49), s(103),c(103));
stage104 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(78), s(77), p15(3),s(104),c(104));
stage105 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(80), s(79), c(33), s(105),c(105));
stage106 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(82), s(81), c(34), s(106),c(106));
stage107 : full_adder  generic map (unit_delay => unit_delay)
                    port map (Cin, s(83), c(36), s(107),c(107));
stage108 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(84),c(37),c(38), s(108),c(108));
stage109 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(86),s(85),c(58), s(109),c(109));
stage110 : full_adder  generic map (unit_delay => unit_delay)
                    port map (Cin, s(88), s(87), s(110), c(110));
stage111 : full_adder  generic map (unit_delay => unit_delay)
                    port map (p10(15),p11(14),p12(13),s(111),c(111));
stage112 : full_adder  generic map (unit_delay => unit_delay)
                    port map (p0(6), p1(5), p2(4),s(112),c(112));
stage113 : full_adder  generic map (unit_delay => unit_delay)
                    port map (p3(3), p4(2), p5(1),s(113),c(113));
stage114 : full_adder  generic map (unit_delay => unit_delay)
                    port map (p0(7), p1(6), p2(5),s(114),c(114));
stage115 : full_adder  generic map (unit_delay => unit_delay)
                    port map (p3(4), p4(3), p5(2),s(115),c(115));
stage116 : full_adder  generic map (unit_delay => unit_delay)
                    port map (Cin, p6(1), p7(0), s(116),c(116));
stage117 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(91), s(90), s(89),s(117),c(117));
stage118 : full_adder  generic map (unit_delay => unit_delay)
                    port map (Cin, s(92), c(89), s(118), c(118));
stage119 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(93),c(62),c(63),s(119),c(119));
stage120 : full_adder  generic map (unit_delay => unit_delay)
                    port map (p6(4),p7(3),p8(2),s(120),c(120));
stage121 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(95),s(94),c(93),s(121),c(121));
stage122 : full_adder  generic map (unit_delay => unit_delay)
                    port map (Cin, s(96), c(66),s(122),c(122));
stage123 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(98),s(97),c(67),s(123),c(123));
stage124 : full_adder  generic map (unit_delay => unit_delay)
                    port map (Cin,p12(1),p13(0),s(124),c(124));
stage125 : full_adder  generic map (unit_delay => unit_delay)
                    port map (Cin, s(99), c(69),s(125),c(125));
stage126 : full_adder  generic map (unit_delay => unit_delay)
                    port map (Cin, s(100),c(70), s(126), c(126));
stage127 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(102),s(101),c(48),s(127),c(127));
stage128 : full_adder  generic map (unit_delay => unit_delay)
                    port map (Cin, s(103), c(73), s(128),c(128));
stage129 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(104),c(32),c(50), s(129),c(129));
stage130 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(105),c(51),c(52), s(130),c(130));
stage131 : full_adder  generic map (unit_delay => unit_delay)
```

```
                    port map (s(106),c(53),c(54), s(131),c(131));
stage132 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(107),c(55),c(56), s(132),c(132));
stage133 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(108),c(39),c(57), s(133),c(133));
stage134 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(109),c(59),c(60), s(134),c(134));
stage135 : full_adder  generic map (unit_delay => unit_delay)
                    port map (Cin,p14(9), p15(8), s(135),c(135));
stage136 : full_adder  generic map (unit_delay => unit_delay)
                    port map (p15(9),s(110),c(85),s(136),c(136));
stage137 : full_adder  generic map (unit_delay => unit_delay)
                    port map (Cin, s(111), c(87), s(137),c(137));
stage138 : full_adder  generic map (unit_delay => unit_delay)
                    port map (p13(12),p14(11),p15(10),s(138),c(138));
stage139 : full_adder  generic map (unit_delay => unit_delay)
                    port map (p11(15),p12(14),p13(13),s(139),c(139));
stage140 : full_adder  generic map (unit_delay => unit_delay)
                    port map (Cin,p14(12),p15(11),s(140),c(140));
stage141 : full_adder  generic map (unit_delay => unit_delay)
                    port map (p12(15),p13(14),p14(13),s(141),c(141));
stage142 : full_adder  generic map (unit_delay => unit_delay)
                    port map (p0(5),p1(4),p2(3),s(142),c(142));
stage143 : full_adder  generic map (unit_delay => unit_delay)
                    port map (p3(2),p4(1),p5(0), s(143),c(143));
stage144 : full_adder  generic map (unit_delay => unit_delay)
                    port map (Cin, s(113),s(112),s(144),c(144));
stage145 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(116),s(115),s(114),s(145),c(145));
stage146 : full_adder  generic map (unit_delay => unit_delay)
                    port map (Cin,s(117),c(114), s(146), c(146));
stage147 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(118),c(90),c(91),s(147), c(147));
stage148 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(120),s(119),c(92),s(148),c(148));
stage149 : full_adder  generic map (unit_delay => unit_delay)
                    port map (Cin, p9(1), p10(0), s(149),c(149));
stage150 : full_adder  generic map (unit_delay => unit_delay)
                    port map (Cin, s(121),c(119), s(150),c(150));
stage151 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(122),c(94),c(95),  s(151),c(151));
stage152 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(124),s(123),c(68), s(152),c(152));
stage153 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(125),c(97), c(98), s(153),c(153));
stage154 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(126),c(71), c(99), s(154),c(154));
stage155 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(127),c(72), c(100),s(155),c(155));
stage156 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(128),c(74), c(101),s(156),c(156));
stage157 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(129),c(75), c(76), s(157),c(157));
stage158 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(130),c(77), c(78), s(158),c(158));
stage159 : full_adder  generic map (unit_delay => unit_delay)
```

```
                    port map (s(131),c(79), c(80), s(159),c(159));
stage160 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(132),c(81), c(82), s(160),c(160));
stage161 : full_adder  generic map (unit_delay => unit_delay)
                    port map (Cin, s(133), c(83),  s(161),c(161));
stage162 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(135),s(134),c(84), s(162),c(162));
stage163 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(136),c(86), c(109),s(163),c(163));
stage164 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(138),s(137),c(88), s(164),c(164));
stage165 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(140),s(139),c(111),s(165),c(165));
stage166 : full_adder  generic map (unit_delay => unit_delay)
                    port map (Cin, s(141), p15(12),s(166),c(166));
stage167 : full_adder  generic map (unit_delay => unit_delay)
                    port map (p13(15),p14(14),p15(13),s(167),c(167));
stage168 : full_adder  generic map (unit_delay => unit_delay)
                    port map (p0(3), p1(2), p2(1), s(168), c(168));
stage169 : full_adder  generic map (unit_delay => unit_delay)
                    port map (p0(4), p1(3), p2(2), s(169), c(169));
stage170 : full_adder  generic map (unit_delay => unit_delay)
                    port map (Cin, p3(1), p4(0), s(170), c(170));
stage171 : full_adder  generic map (unit_delay => unit_delay)
                    port map (Cin, s(143), s(142), s(171), c(171));
stage172 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(144), c(142), c(143), s(172), c(172));
stage173 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(145), c(112), c(113), s(173), c(173));
stage174 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(146), c(115), c(116), s(174), c(174));
stage175 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(147), p9(0), c(117), s(175), c(175));
stage176 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(149), s(148), c(118), s(176), c(176));
stage177 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(150), c(120), c(148), s(177), c(177));
stage178 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(151), p12(0), c(121), s(178), c(178));
stage179 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(152), c(96), c(122), s(179), c(179));
stage180 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(153), c(123), c(124), s(180), c(180));
stage181 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(154), c(125), p15(0), s(181), c(181));
stage182 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(155), c(126), c(154), s(182), c(182));
stage183 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(156), c(102), c(127), s(183), c(183));
stage184 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(157), c(103), c(128), s(184), c(184));
stage185 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(158), c(104), c(129), s(185), c(185));
stage186 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(159), c(105), c(130), s(186), c(186));
stage187 : full_adder  generic map (unit_delay => unit_delay)
```

```
                    port map (s(160), c(106), c(131), s(187), c(187));
stage188 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(161), c(107), c(132), s(188), c(188));
stage189 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(162), c(108), c(133), s(189), c(189));
stage190 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(163), c(134), c(135), s(190), c(190));
stage191 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(164), c(110), c(136), s(191), c(191));
stage192 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(165), c(137), c(138), s(192), c(192));
stage193 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(166), c(139), c(140), s(193), c(193));
stage194 : full_adder  generic map (unit_delay => unit_delay)
                    port map (Cin, s(167), c(141), s(194), c(194));
stage195 : full_adder  generic map (unit_delay => unit_delay)
                    port map (Cin, p14(15), p15(14), s(195), c(195));
stage196 : full_adder  generic map (unit_delay => unit_delay)
                    port map (p0(2), p1(1), p2(0), s(196), c(196));
stage197 : full_adder  generic map (unit_delay => unit_delay)
                    port map (Cin, s(168), p3(0), s(197), c(197));
stage198 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(170), s(169), c(168), s(198), c(198));
stage199 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(171), c(169), c(170), s(199), c(199));
stage200 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(172), p6(0), c(171), s(200), c(200));
stage201 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(173), c(144), c(172), s(201), c(201));
stage202 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(174), c(145), c(173), s(202), c(202));
stage203 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(175), c(146), c(174), s(203), c(203));
stage204 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(176), c(147), c(175), s(204), c(204));
stage205 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(177), c(149), c(176), s(205), c(205));
stage206 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(178), c(150), c(177), s(206), c(206));
stage207 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(179), c(151), c(178), s(207), c(207));
stage208 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(180), c(152), c(179), s(208), c(208));
stage209 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(181), c(153), c(180), s(209), c(209));
stage210 : full_adder  generic map (unit_delay => unit_delay)
                    port map (Cin, s(182), c(181), s(210), c(210));
stage211 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(183), c(155), c(182), s(211), c(211));
stage212 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(184), c(156), c(183), s(212), c(212));
stage213 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(185), c(157), c(184), s(213), c(213));
stage214 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(186), c(158), c(185), s(214), c(214));
stage215 : full_adder  generic map (unit_delay => unit_delay)
```

```
                    port map (s(187), c(159), c(186), s(215), c(215));
stage216 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(188), c(160), c(187), s(216), c(216));
stage217 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(189), c(161), c(188), s(217), c(217));
stage218 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(190), c(162), c(189), s(218), c(218));
stage219 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(191), c(163), c(190), s(219), c(219));
stage220 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(192), c(164), c(191), s(220), c(220));
stage221 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(193), c(165), c(192), s(221), c(221));
stage222 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(194), c(166), c(193), s(222), c(222));
stage223 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(195), c(167), c(194), s(223), c(223));
stage224 : full_adder  generic map (unit_delay => unit_delay)
                    port map (Cin, p15(15), c(195), s(224), c(224));
stage225 : full_adder  generic map (unit_delay => unit_delay)
                    port map (Cin, p0(1), p1(0), s(225), c(225));
stage226 : full_adder  generic map (unit_delay => unit_delay)
                    port map (Cin, s(196), c(225), s(226), c(226));
stage227 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(197), c(196), c(226), s(227), c(227));
stage228 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(198), c(197), c(227), s(228), c(228));
stage229 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(199), c(198), c(228), s(229), c(229));
stage230 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(200), c(199), c(229), s(230), c(230));
stage231 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(201), c(200), c(230), s(231), c(231));
stage232 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(202), c(201), c(231), s(232), c(232));
stage233 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(203), c(202), c(232), s(233), c(233));
stage234 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(204), c(203), c(233), s(234), c(234));
stage235 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(205), c(204), c(234), s(235), c(235));
stage236 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(206), c(205), c(235), s(236), c(236));
stage237 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(207), c(206), c(236), s(237), c(237));
stage238 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(208), c(207), c(237), s(238), c(238));
stage239 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(209), c(208), c(238), s(239), c(239));
stage240 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(210), c(209), c(239), s(240), c(240));
stage241 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(211), c(210), c(240), s(241), c(241));
stage242 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(212), c(211), c(241), s(242), c(242));
stage243 : full_adder  generic map (unit_delay => unit_delay)
```

```
                    port map (s(213), c(212), c(242), s(243), c(243));
stage244 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(214), c(213), c(243), s(244), c(244));
stage245 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(215), c(214), c(244), s(245), c(245));
stage246 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(216), c(215), c(245), s(246), c(246));
stage247 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(217), c(216), c(246), s(247), c(247));
stage248 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(218), c(217), c(247), s(248), c(248));
stage249 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(219), c(218), c(248), s(249), c(249));
stage250 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(220), c(219), c(249), s(250), c(250));
stage251 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(221), c(220), c(250), s(251), c(251));
stage252 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(222), c(221), c(251), s(252), c(252));
stage253 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(223), c(222), c(252), s(253), c(253));
stage254 : full_adder  generic map (unit_delay => unit_delay)
                    port map (s(224), c(223), c(253), s(254), c(254));
stage255 : full_adder  generic map (unit_delay => unit_delay)
                    port map (Cin, c(224), c(254), s(255), c(255));
z(0)<= p0(0);
z(1)<= s(225);
z(2)<= s(226);
z(3)<= s(227);
z(4)<= s(228);
z(5)<= s(229);
z(6)<= s(230);
z(7)<= s(231);
z(8)<= s(232);
z(9)<= s(233);
z(10)<= s(234);
z(11)<= s(235);
z(12)<= s(236);
z(13)<= s(237);
z(14)<= s(238);
z(15)<= s(239);
z(16)<= s(240);
z(17)<= s(241);
z(18)<= s(242);
z(19)<= s(243);
z(20)<= s(244);
z(21)<= s(245);
z(22)<= s(246);
z(23)<= s(247);
z(24)<= s(248);
z(25)<= s(249);
z(26)<= s(250);
z(27)<= s(251);
z(28)<= s(252);
z(29)<= s(253);
z(30)<= s(254);
```

```
        z(31)<= s(255);
end Behavioral;
```

### 3.4 วงจรบวก

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity full_adder is                    -- structure of full adder
        generic (unit_delay : time );
        Port ( a, b, Cin : in std_logic;
        sum, Cout : out std_logic);
end full_adder;
architecture Behavioral of full_adder is
begin
        sum <= a xor b xor Cin after unit_delay;
        Cout <= (a and b) or (a and Cin) or (b and Cin)after unit_delay;
end Behavioral;
```

### 3.5 AND Gate

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity and_gate is                      -- structure of AND gate
        generic (unit_delay : time );
        Port ( x, y : in std_logic ;
        pp : out std_logic);
end and_gate ;

architecture Behavioral of and_gate is
begin
        pp <= x and y after unit_delay;
end Behavioral;
```

# ภาคผนวก ง.

## โปรแกรมสร้างลำดับบิตสุ่มเทียมด้วยภาษา C++

```
#include <iostream>
#include <iomanip>
#include <fstream>
#include <math.h>
//Define N shift register
#define N 31
//Define half Period of Signal
#define period 40e-9
//Define Rise/Fall Time
#define tr_f 60e-12
int main()
{
double i,j,tmp;
double test_New_line_check;
//For N = 31
int a[] = {1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1};
double bit_limit;
int a_temp;
double time_start,print_start,time_stop,print_stop,time_tmp;
double count;
//Initialize
        test_New_line_check=0;
        tmp=0;
        time_start = 0;
        print_start = 0;
        time_stop = 0;
        print_stop = 0;
        time_tmp = 0;
        count = 1;
//Create File
        std::cout << "PSRB_Sequence Software where 2^31 - 1 " << std::endl;
        std::ofstream ofile("test.txt");
        std::ofstream psrbfile("psrb_seq.txt");
        std::ofstream spfile("spice.txt");
//Total bit of PSRB
        bit_limit=pow(2,N);
//      bit_limit=10240*2;
        if (!ofile.fail())
        {
                std::cout << "Writing to file..." << std::endl;
                ofile << "These are PSRB Sequence >> 2^" << N << " -1" << std::endl;
                for (i=0; i<bit_limit; i++)
                {
                //xor N=31 bit c31,c3
                        if (a[31]==a[3])
                                a[0]=0;
                        else
                                a[0]=1;
                //shift register
                        for (j=0; j<N; j++)
                                a[(int)(N-j)]=a[(int)(N-j-1)];
                //Print to test.txt and psrb_seq.txt
```

```
                ofile << a[0];
                psrbfile << a[0];
            //Calculate to spice file
                if (i>0)
                    if (a_temp == a[0])
                        count++;
                    else if (a_temp != a[0])
                    {
                        time_start = time_tmp;
                        print_start = time_start+0.5*tr_f;
                        time_stop = count*period+time_start;
                        print_stop = time_stop-0.5*tr_f;
                        time_tmp = time_stop;
                        count=1;
                    //convert and print to spice.txt
                        if (a_temp==0)
                        {
                            spfile << "+"
                                << std::scientific
                                << print_start
                                << " low,";
                            spfile << std::scientific
                                << print_stop
                                << " low,";
                            spfile << std::endl;
                        }
                        else if (a_temp==1)
                        {
                            spfile << "+"
                                << std::scientific
                                << print_start
                                << " high,";
                            spfile << std::scientific
                                << print_stop
                                << " high,";
                            spfile << std::endl;
                        }
                    }
            //Set a_temp
                a_temp = a[0];
            //N bit New line
                if (test_New_line_check==N)
                {
                    test_New_line_check=0;
                    ofile <<std::endl;
                }
                else
                    test_New_line_check++;
            }
        }
        else
            std::cout <<"Cannot open";
        return 0;
}
```

# ภาคผนวก จ.
# บทความที่ได้รับการตีพิมพ์ใน
# การประชุมวิชาการทางวิศวกรรมไฟฟ้า ครั้งที่ 29

# Low-Power Multiplier Design Based on Dual Supply Voltage Technique

P.Chunak, and B.Supmonchai

IC Design and Application Research (IDAR) Laboratory

Department of Electrical Engineering, Faculty of Engineering
Chulalongkorn University, Pathumwan, Bangkok, 10330, Thailand

Phone 0-2218-6484 ext. 12, Fax 0-2218-6488, E-mail: boonchuay.s@chula.ac.th

EL20

## Abstract

A tree multiplier design approach based on dual voltage supply technique is proposed. Our design consists of two types of full-adder units, one with a higher voltage supply at 3.3V and the other at 2.5V. The 3.3V full-adder units are used exclusively in the critical path of the multiplier to guarantee its best overall performance while the 2.5V units are used in the region where the timing is not critical to reduce the power consumption. All paths are checked to ensure that the performance of the multiplier is maintained. The slower 2.5V adder units are systematically replaced with the faster 3.3V adders in the violating paths to lower the timing to be within the limit. Our proposed technique can reduce power consumption of the tree multiplier up to 42% in 16x16-bit multiplier without deteriorating its delay performance.

**Keywords:** Full-Adder, Tree Multiplier, Low-Power Technique, Dual Supply Voltage.

## 1. Introduction

Multiplier is an essential arithmetic unit commonly found in microprocessors and digital signal processing (DSP) systems. Inherently, multiplications are expensive and slow operations. The speed and power of the multiplier often determines the overall system performance.

With the increasing complexity of VLSI systems and limited amount of power available in certain scenarios such as in Personal Digital Assistants (PDA) and digital cameras, minimizing power consumption has clearly become an important design goal. Lower power consumption also leads to a saving in package cost and increase in reliability due to reduced heat dissipation. Power minimization, however, should not compromise the performance of critical components such as the multiplier.

Power dissipation in CMOS VLSI circuits is dominated by dynamic power consumption, which accounts for about 80% of the total system power [1]. Dynamic power dissipation results from the charging and discharging of the wire and transistor capacitive loads. The dynamic power dissipation of a CMOS circuit is described by [2]

$$P_{SWITCHING} = \alpha \cdot C_L \cdot V_{swing} \cdot V_{DD} \cdot f_{CLK}, \qquad (1)$$

where $\alpha$ is the switching activity, $C_L$ is the total load capacitance, $V_{swing}$ is the output voltage swing, $V_{DD}$ is the supply voltage, and $f_{clk}$ is the operating clock frequency. Consequently, power reduction can be achieved by decreasing one or more of the factors on the right-hand side.

In this paper, we propose a low-power multiplier design approach based on dual supply voltage technique. Much attention [3-7] has been devoted to power minimization of one-bit full adder which is fundamental to all multipliers. However, the overall performance of multipliers constructed from these adders is usually conceded. An effective way to simultaneously retain performance and reduce power consumption is the dual supply voltage technique [8-9]. It was successfully applied to many designs in the gate and module level [8] [10]. The technique, nevertheless, has not been fully investigated to be used in designing repetitive structure such as multiplier.

The rest of the paper is organized as follow. Section 2 provides a background on multiplier architecture and the full-adder unit. Transistor sizing of the full adder unit is also considered here. Section 3 discusses the dual supply voltage technique applied to our multiplier design. Section 4 contains simulation results for various sizes of multipliers. Section 5 gives the conclusion

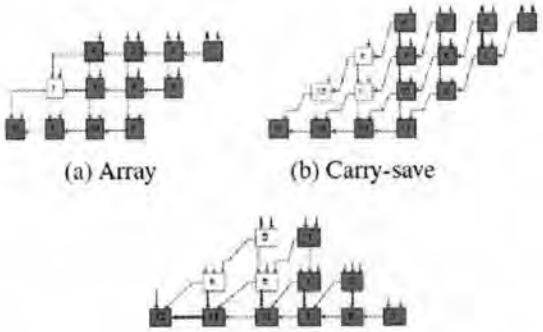
## 2. Background

Multipliers are complex adder arrays whose function is to accumulate partial products in an efficient and effective way. A partial product is the result from the logical AND between multiplicand and a multiplier bit. Since the generation of partial product is similar to all multiplier design, it will not be of our concern herein.

A straightforward implementation of the multiplier is to organize the adder array in a rhombus shape mimicking the manual multiplication as shown in figure 1(a). This arrangement, however, yields multiple critical paths all of which are of the same length. It turns out that almost every adder units in the array (darkened cell in the figure) lie in the critical paths. Therefore, optimizing just some of them will not work. The multiplier must be optimized as a whole and the benefit of transistor

sizing in this case will be minimal. Another consequence of having multiple critical paths is that worst delay of sum and carry of the adder must be approximately the same since critical paths point to every direction.

Carry-save multiplier [2] in figure 1(b) is another arrangement which could be used in our approach but it still has too many critical paths which make optimization only marginal. The carry-save multiplier also requires larger number of full adder units than the array multiplier. This means that more power is consumed in the carry-save multiplier.



(a) Array (b) Carry-save

(c) Tree

Figure 1: Architecture of Multiplier; darkened cells indicate the adder units which lie in the critical paths.

## 2.1 Tree Multiplier

Tree multiplier [2] reduces both the number of critical paths and the number of adder units in the path. The propagation delay is therefore reduced. The number of adder units in tree multiplier is slightly more than that of the array multiplier. Note that there are numerous ways to arrange the adder units into a treelike fashion. Example of an adder tree for a 4-bit by 4-bit multiplier is shown in figure 1(c). Table 1 compares the complexity of all three architectures in terms of the number of adder units used and the worst delay performance.

Table 1: Comparison of Multiplier Architecture Complexity

| | Array | Carry-Save | Tree |
|---|---|---|---|
| Number of Adder units | $O(N^2)$ | $O(N^2)$ | $O(N^2)$ |
| Worst Delay | $O(N)$ | $O(N)$ | $O(\log N)$ |

## 2.2 One-bit Full Adder

The basic building block of the multiplier is the one-bit full-adder (Figure 2). The output function of the adder can be described by the following Boolean expression [4],


Figure 2: One-bit Full Adder

$$S = (A \oplus B) \oplus C_{in} \quad (2)$$

$$C_{out} = A \cdot B + C_{in} \cdot (A \oplus B) \quad (3)$$

Figure 3 shows the transistor schematics of the full adder unit used in our multiplier. It is a modification of the circuit published in [4-5]. Similar to the original design, the proposed circuit has a good property that the worst delay for sum and carry are almost equal. The main difference is that, in our design, the inverters are inserted at the outputs to solve the voltage drop problem (which is recognized by the authors in [4]) caused by the transistor cascading effect.

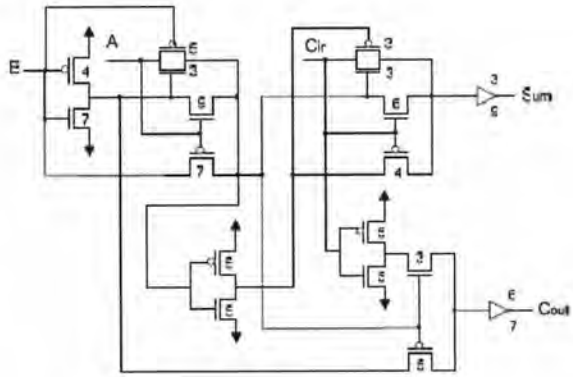



Figure 3: Transistor Schematics of Proposed Full-Adder

## 2.3 Transistor Sizing

In our multiplier design employing dual supply voltage technique, we will need two types of adder unit, each operating at different levels of supply voltage, one at 3.3V and the other at 2.5V. Transistor sizing for each type of adder unit is done through SPICE simulation following the procedure in [4] to ensure its best performance. The target technology is the TSMC 0.25 $\mu$m CMOS process. As in [4], the size of each transistor is allowed only as a multiple of the minimum transistor size in order to reduce the complexity of the optimization process. An example of the final result on transistor sizing for 3.3V adder unit is shown in figure 3. The number besides each transistor represents its relative size to the minimum sized transistor (=1). The load capacitances at Sum and Cout are assumed to be equal to the input capacitances of the next adder unit (which are approximately 37 fF for $A$ and $B$ and 56 fF for $C_{in}$). The performance of both adder units is summarized in table 2 below.

EL20

Table 2: Performance of full adders using different supply voltage

| | $V_{DD}$ = 3.3V | $V_{DD}$ = 2.5V |
|---|---|---|
| Power @ 25 MHz ($10^{-4}$ W) | 2.581 | 0.862 |
| *Sum* Worst Delay (ps) | 340.4 | 481.2 |
| *Carry* Worst Delay (ps) | 317.0 | 463.0 |
| Power-Delay ($10^{-14}$J) | 5.47 | 2.90 |

## 2.4 Level Converter

Level converters are needed whenever a module at lower supply voltage has to drive a gate at the higher voltage. In such case, the PMOS transistor in the driven gate is never turned off, resulting in static current and reduced output swing. The problems can be prevented by applying level conversion at these interfaces. In our multiplier design, level converter based on positive feedback [8], as shown in figure 4, is used.

The effect of level converters on the overall performance of the multiplier is minimal since the conversions are required only at the transitions from lower to higher supply voltage units. The size of the transistors in level converter can also be made small because of one-to-one connection between the lower and higher units. In figure 4, the sizes are given in multiple of the minimum sized transistor (=1).

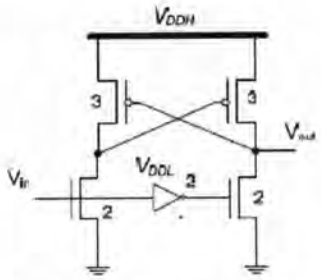



Figure 4: Level Converter with proper transistor sizing

## 3. Dual Supply Voltage Technique for Multiplier Design

We propose the following iterative steps for designing low-power tree multiplier employing dual supply voltage technique. The tree multiplier designed by our approach can achieve the same delay performance as the multiplier designed using only higher supply voltage but with significant decrease in power consumption.

*Step 1*: Construct the tree multiplier using only higher supply voltage (3.3V) adder units. Identify critical paths and mark all the adder units lying in the paths.

*Step 2*: Replace every unmarked adder units with lower supply voltage (2.5V) adder units. Check the critical paths.

*Step 3*: If critical paths do not change then stop else consider the new critical paths. Note that these new paths may contain both types of adder units.

*Step 4*: For each critical path, locate the lower supply voltage adder unit lying adjacent to a higher or a chain of higher supply voltage adder units. Replace it with a higher supply voltage adder unit. Insert level converters where necessary and remove those no longer needed. Find the critical paths. Go to step 3.

This methodology guarantees that higher supply voltage adder units are used only where necessary. For the most critical path, all adder units belong to the path are kept operating at higher supply voltage. The purpose is to not deteriorate the overall delay performance of the multiplier. The less critical path contains both types of adder but will have enough number of higher supply voltage units in the path to keep the timing below that of the most critical path. The rest of the adder units in the multiplier which is non-critical are designated to operate in lower supply voltage in order to reduce power consumption.

An example of an 8-bit by 8-bit tree multiplier resulted from the iterative steps described above is shown in figure 5. The white cells in the figure are the 2.5V (lower supply) adder units. The darkened cells are the 3.3V (higher supply) adder units that lie in the most critical paths which have a delay of approximately 5.5 ns. The hatched cells are the 3.3V adder units which replace the 2.5V adder cells later during iteration of step 3. The small squares are level converters necessarily inserted at the interface between the 3.3V and 2.5V supply units. The final design is reached after 3 iterations.

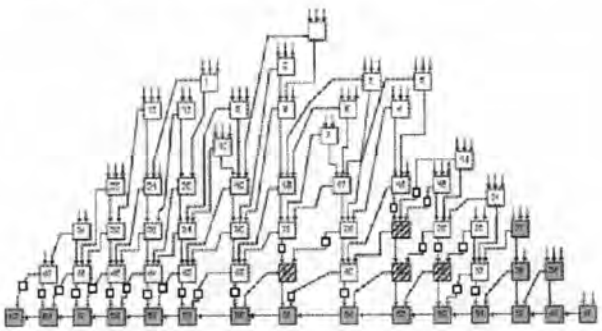


Figure 5: 8x8-bit Low-Power Tree Multiplier using dual supply voltage generated by proposed technique.

It is noted, however, that each iteration step does require a significant effort to identify the right critical paths in the multiplier. With simple searching algorithm that we have been using, an iteration loop takes a long time to be accomplished. Even worse, the amount of time increases exponentially with the size of the multiplier. Thus, finding an appropriate solution to this problem could be very challenging.

## 4. Simulation Results

We have applied the proposed dual supply voltage technique to tree multipliers of various sizes. The levels of supply voltage are chosen to be at 3.3V and 2.5V. Performance of the design achieved by our technique is compared with that of the tree multiplier with the same topology using only high supply voltage at 3.3V. The results are summarized in table 3.

Table 3: Performance comparison of tree multipliers

| Size | Power (mW) | | | Worst Delay (ns) | | |
|---|---|---|---|---|---|---|
| | Single Supply (3.3V) | Dual Supply (2.5 & 3.3V) | % dec. | Single Supply (3.3V) | Dual Supply (2.5 & 3.3V) | % inc. |
| 4x4 | 2.95 | 2.64 | 10.7 | 2.04 | 2.13 | 4.0 |
| 8x8 | 23.98 | 15.89 | 33.7 | 3.92 | 4.15 | 5.5 |
| 16x16 | 110.25 | 63.45 | 42.45 | 11.37 | 12.12 | 6.7 |

From table 3, we see that using our proposed technique, the power consumption of the tree multipliers are grossly reduced while the delay performance is slightly affected. The reduction of power consumption is as high as 42% in 16x16-bit multiplier while the delay is worsened for only 6.7%. The increased number of level converters present in the multiplier critical paths are the factor responsible for the increase in delay and reduced percentage of expected power saving. Nevertheless, we still expect an even greater power saving for larger multiplier.

## 6. Conclusions

In this paper we have presented a design approach which can be used in minimizing power consumption in a tree multiplier. The approach is based on the dual supply voltage technique. By keeping the adder units in the critical path operating at high supply voltage, the delay performance of the multiplier is guaranteed to be not deteriorated. The non-critical area of the multiplier is then replaced with the adder units at lower supply voltage to save the overall power consumption. All paths must be checked to make sure that no paths is violating the critical path timing. If violation occurs, some of the lower supply voltage units must be systematically replaced with the higher supply voltage ones to reduce the delay to be within the limit.

We have applied our approach in designing tree multipliers of various sizes. All designs are then simulated using SPICE in CMOS 0.25 um process. We find that the saving on power dissipation can be as high as 42% in 16-bit by 16-bit tree multiplier. We expect an even greater saving on larger multiplier design.

## Acknowledgement

The authors would like to thank Associate Professor Dr. Ekachai Leelarasmee for his insight and valuable suggestions during the research on this project.

## References

[1] D. Soudris, C. Piguet and C. Goutis (eds). *Designing CMOS Circuits for Low Power*, Kluwer academic publishers, 2002.

[2] J. Rabaey, *Digital Integrated Circuits: A Design Perspective*, 2nd ed., Prentice-Hall, 2003.

[3] I.C. Wey, C.H. Huang, and H.C. Chow, "A New Low-Voltage CMOS 1-Bit Full Adder for High Performance Applications", *IEEE Asia-Pacific Conference*, pp. 21-24, 2002

[4] A.M. Shams, T.K. Darwish, and M.A. Bayoumi, "Performance Analysis of Low-Power 1-Bit CMOS Full Adder Cells", *IEEE Transactions on VLSI Systems*, Vol. 10, No.1, pp. 20-29, February 2002.

[5] H. A. Mahmoud and M. A. Bayoumi, "A 10-Transistor Low-Power High-Speed Full Adder Cell" *IEEE International Symposium on Circuits and Systems*, Vol. 1, pp. 43-46, June 1999.

[6] N. Zhuang and H. Wu, "A New Design of the CMOS Full Adder", *IEEE Journal of Solid-state Circuits*, vol. 27, no. 5, pp.840-844, May 1992.

[7] S. Goel, S. Gollamudi, A. Kumar, and M. Bayoumi, "On The Design of Low-Energy Hybrid CMOS 1 -Bit Full Adder Cells", *Proceedings of the 47th IEEE International Midwest Symposium on Circuits and Systems*, vol. 2, pp. 209-212, 2004.

[8] K. Usami, and M. Igarashi, "Low-Power Design Methodology and Applications utilizing Dual Supply Voltages", *IEEE Asia and South Pacific Design Automation Conference*, pp. 123-128, January 2000.

[9] M. Pedram, "Power Minimization in IC Design," *ACM Transactions on Design Automation of Electronic Systems*, Vol.1, No. 1, pp. 3-56, January 1996.

[10] A. Chandrakasan and R. Brodersen, *Low power digital CMOS design*, Kluwer, 1995.

Patcharaporn Chunak received the B.S. degree from Suranaree University of Technology in 2002. She is currently a M.S. candidate with the IC Design and Application Research (IDAR) Laboratory at Chulalongkorn University.

Boonchuay Supmonchai is a lecturer associated with the IC Design and Application Research (IDAR) Laboratory at Chulalongkorn University. His interest is in Digital and Analog IC designs and applications, Biomedical Instrumentations.

EL20

# ประวัติผู้เขียนวิทยานิพนธ์

นางสาวภัชราภรณ์ ชูนาค สำเร็จการศึกษาระดับปริญญาวิศวกรรมศาสตรบัณฑิต สาขาวิชาวิศวกรรมไฟฟ้า จากคณะวิศวกรรมศาสตร์มหาวิทยาลัยเทคโนโลยีสุรนารี ในปีการศึกษา 2545 และเข้าศึกษาต่อในหลักสูตรวิศวกรรมศาสตรมหาบัณฑิต สาขาวิชาวิศวกรรมไฟฟ้า แขนงวิชาการออกแบบและประยุกต์วงจรรวม ที่คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย ในปีการศึกษา 2546