修士論文

エラー検出可能な
二線式ドミノ回路の評価

平成19年2月2日 提出

指導教員 池田誠 助教授

東京大学大学院 工学系研究科
電子工学専攻

56423　石井　健

目 次

図 目 次

第1章　序論

1.1　研究の背景

近年半導体の微細化が進むにつれプロセスばらつきが問題となってきている。さらに、プロセスばらつきに加え温度、電源電圧のばらつきは回路の遅延の変動となって現れ、設計の段階での遅延の制約を満たさなくなり誤動作を起こしてしまうことになる。また電源電圧は下げられ、それとともに性能を保つためにスレショルド電圧も下げられてきたため、ノイズマージンが減り電源ノイズによるエラーやソフトエラーも起こりやすくなる。

これらの問題を解決するために、実際にエラーを検出し訂正しようとする研究がなされており、多数決回路を用いる方法が提案されている [1] [2]。この方法では、同じ回路を複数載せるために面積を数倍とることになるという欠点がある。

本研究では、DCVSL(Differential Cascode Voltage Switch Logic) という二線式の回路方式を自己同期的に使用することでこれらの問題を解決しようとしている。この方式は多重化回路に比べて面積を抑えることができる。二線式によって、エラーの検出を行い、さらに自己同期的 [3] に用いることによって遅延の制約がなくなり、遅延の変動によるエラーが起きない。

このような特徴を持つDCVSL 回路の性能評価のためシミュレーションによりスタティック CMOS(Complementary Metal Oxide Semiconductor) 回路との遅延、消費電力等の比較を行い、その優位性を明らかにする。また実際に回路で発生するエラーを検出する回路を提案し、90nm テクノロジを用いて設計した。次に DCVSL 回路を用いた CPU で様々なばらつき条件下で命令の遅延を測定し比較を行なう。そして CPU の電源電圧を制御するための回路の提案を行なう。

本論文ではその研究成果についての報告を行なう。

1.2　現在の問題点

現在の LSI(Large Scale Integrated Circuit) の問題点としてまず、誤動作が起きることが考えられる。誤動作の要因としては大きく二つに分けられる。一つ目は、製造時の不良により期待したものができていないものであり、これは出荷時に不良品として出荷されない。二つ目は、製造時には不良が起きないものの LSI を動

作させていると、正しく動作しないものである。

本研究では後者を対象とする。後者のエラーもまた二つに分けられ、ソフトエラーやノイズによって論理的に誤った値を計算してしまうものを論理エラー、遅延ばらつきが原因で起こるエラーを遅延エラーと呼ぶ。ソフトエラーは、中性子線やα線などによって引き起こされるが、半導体の微細化に伴い、電源電圧の下げられ回路中の各ノードの容量が小さくなったことはメモリ回路や組み合わせ回路のソフトエラーの耐性を弱くしている。以前は、回路に全く影響を与えないような弱いものでも現在のテクノロジに対してはソフトエラーを起こす可能性がある。

メモリにおいてはECC(Error Correction Code)やパリティを用いることである程度防げるようになってきた。また、組み合わせ回路においてはメモリなどと異なり以下の図1.1で示されるように本来ソフトエラーなどのノイズに対する耐性がある [4]。

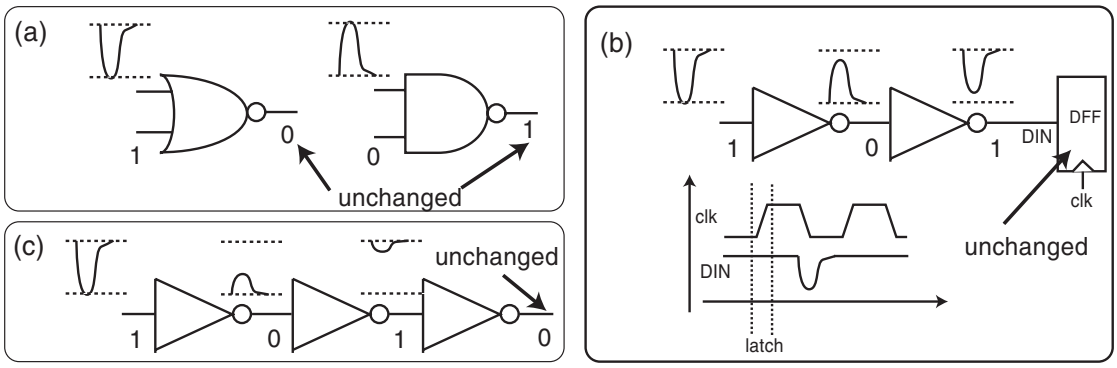


図 1.1: (a)Logical masking (b)Temporal masking (c)Electrical masking [4]

- Logical masking

  図1.1の(a)で示されるように、NORやNANDゲートでそれぞれ一方の入力が1,0であった時にもう一方の入力がノイズなどの影響で反転してしまったとしても出力には影響を与えない。

- Temporal masking

  図1.1の(b)で示されるように、反転してしまった信号がDFFまで届いてしまったとしてもクロックの立ち上がりの時点で元に戻っていればエラーにならない。

- Electrical masking

  図1.1の(c)で示されるように、CMOS回路においては、ノイズによる信号の反転がカットオフ周波数より高速で起きた場合に、次段において増幅されないために反転した信号は徐々に小さくなりエラーとならない。

このように組み合わせ回路ではもともとソフトエラーに対する耐性があるために従来あまり研究されていなかったが近年問題となってきている [13]。また回路の性能にプロセスばらつきが与える影響が大きくなっているので遅延エラーについても問題とされる。

本研究では組み合わせ回路における論理エラーと遅延エラーを対処しようとしている。

論理エラー

論理エラーは図 1.2 の (a) のようなものである。あるクロックの立ち上がりで組み合わせ回路に入力が入ってきたものの、α 線や電源ノイズなどの要因で本来期待される出力とは違うものが出力され次のクロックの立ち上がりで誤った値をラッチしてしまうことである。

遅延エラー

遅延エラーは図 1.2 の (b) のようなものである。あるクロックの立ち上がりで組み合わせ回路に入力が入ってきたものの次のクロックの立ち上がりの時点で演算が終了しておらず正しい値をラッチできずに発生するエラーである。遅延エラーはクロック周波数を遅くすることができればエラーにならない。

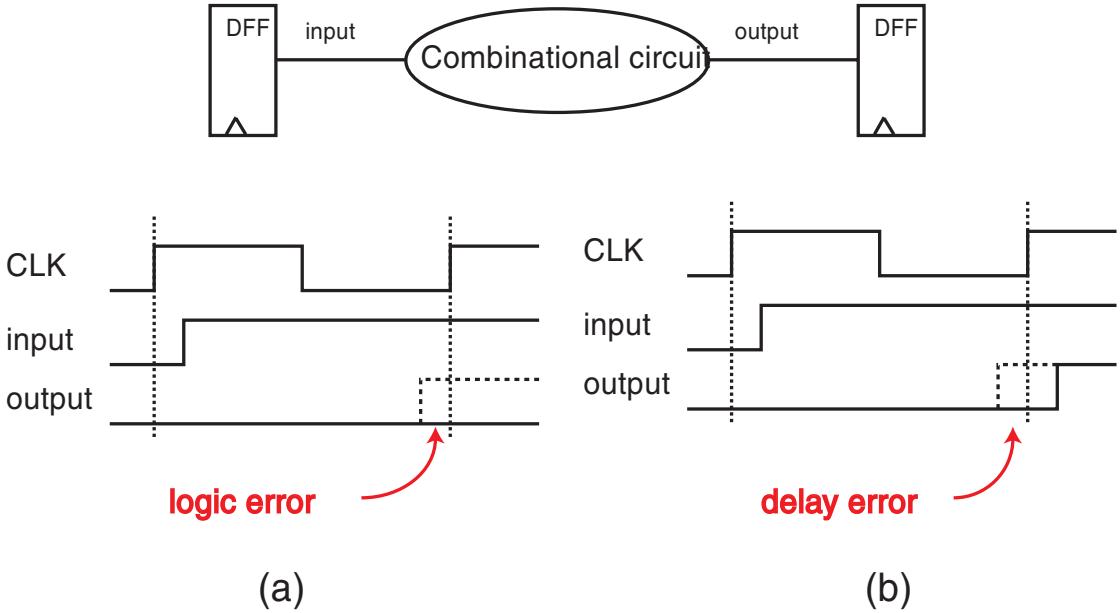


図 1.2: (a) 論理エラー (b) 遅延エラー

表 1.1: Dual-Rail Coding

	A_2	A_1
initial	0	0
0	1	0
1	0	1
illegal	1	1

1.3　関連研究

本研究で用いる二線式ドミノ回路

本研究で用いる二線式ドミノ回路は論理エラーと遅延エラーという問題を解決するために用いる。論理エラーは、信号線に二線を導入することによって検出する。遅延エラーは、自己同期的に動作させることで遅延エラー自体が起きない。自己同期回路では、終了を検出して値をラッチする手法やレプリカ遅延を持たせてクリティカルパスの遅延時間にマージンを持たせた時間が経過したら値をラッチする手法などがあるが、本研究では DCVSL [5] という終了検出が容易にできる回路を使う。また同じ二線式ドミノ回路の一種であるが、速度を改善する目的で当研究室で考案された Footless DCVSL [6] という回路も比較する。

DCVSL (Differential Cascode Voltage Switch Logic)

DCVSL はダイナミック回路の一種である。ロジックを移動度の速い NMOS のみで行なうので高速に動作するのが特長である。図 1.3 は DCVSL による NAND 回路の例である。プリチャージ信号が存在しており、それぞれプリチャージのための PMOS(M1) と貫通電流を防ぐために用いられる NMOS(M2) に入力される。

この回路は二線式であり、表 1.1 のように一つの論理を表すのに二線を用いている。

二線であることより入力信号は二本になり、A_1 が入る論理部は、通常のスタティック CMOS の NMOS 部分と同じであり、A_2 が入る論理部はその双対の関係となっている。こうすることで、入力パターンに応じて 0(1,0) または、1(0,1) が出力されることになる。

回路の動作は、プリチャージフェイズとエバリュエーションフェイズの 2 つのフェイズに分けられる。タイミングチャートは図 1.4 のようになっている。

- プリチャージフェイズ　：　充電

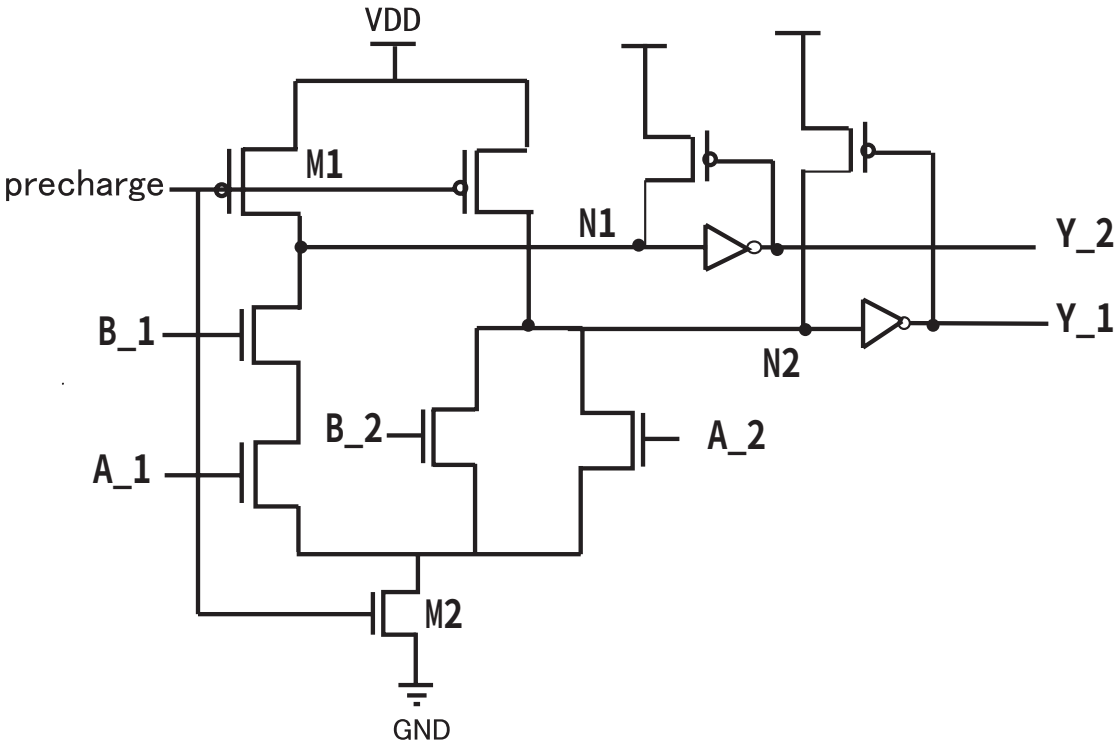


図 1.3: DCVSL の回路図

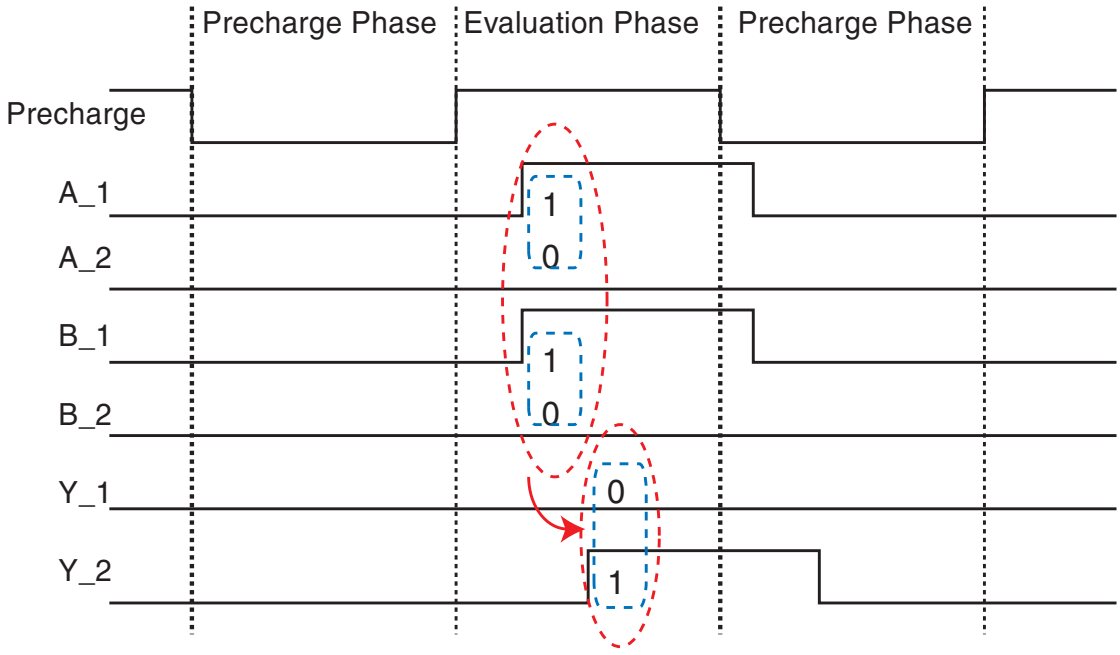


図 1.4: DCVSL の動作のタイミングチャート

このフェイズではプリチャージ信号が Low であり、PMOS が ON になりダイナミックノード (図中 N1,N2) が VDD になるように電荷がたまる。

入力信号の A_1 と A_2 は 0 のままである。

- エバリュエーションフェイズ　：　放電

  このフェイズではプリチャージ信号が High であり、PMOS が OFF になり VDD からダイナミックノードへの電荷の供給がされなくなる、さらに GND 側の NMOS(Foot トランジスタ) が ON となる。

  入力信号の A_1 と A_2 には 0(1,0) または 1(0,1) が入力され、その入力に応じてダイナミックノードに蓄積された電荷が放電される。ダイナミックノードの論理が出力段のインバータによって反転され出力される。

Footless DCVSL

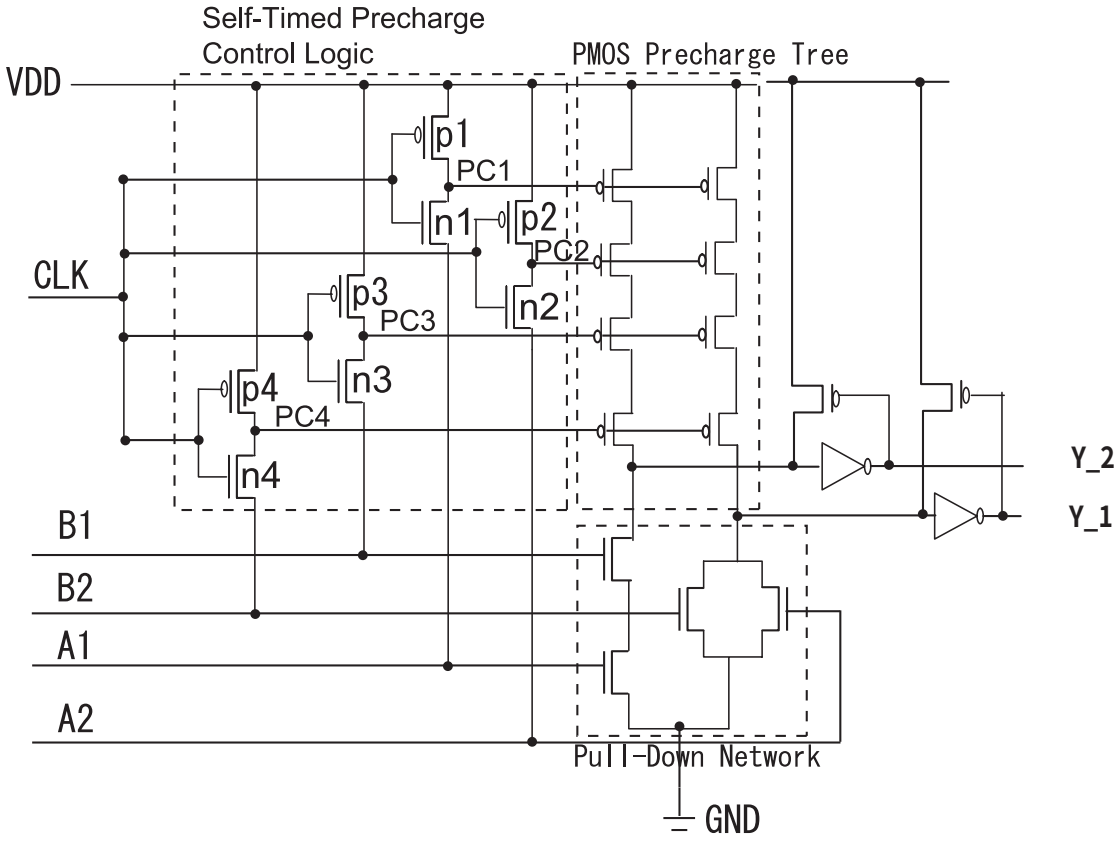


図 1.5: Footless DCVSL の回路図 [6]

Footless DCVSL [6] も基本的には DCVSL と動作原理は同じである。回路図は

図 1.5 のようになっている。DCVSL では、GND 側に Foot トランジスタと呼ばれる NMOS を配置することで、プリチャージフェイズに入力が 1 となったとしても貫通電流が流れないようになっているが、エバリュエーションフェイズにおいてこの NMOS は抵抗として働き回路の遅延が大きくなってしまう。Footless DCVSL では、この Foot トランジスタを除去することで回路を高速にすることができる。しかし、貫通電流が流れないように入力信号のタイミングをうまく制御しなければならない。

この貫通電流が流れないように Self-timed Precharge Control Logic という回路が追加されている。これはプリチャージフェイズにおいて、もし入力信号が 1 であったとすると Self-timed Precharge Control Logic 中のインバータで反転することができず、プリチャージのための PMOS を ON にすることができないということになる。

つまり、入力信号のうちどれか一つでも 1 となるものが存在しているとプリチャージができず、貫通電流が流れることもないということである。

- プリチャージフェイズ

  このフェイズではプリチャージ信号が High であり、インバータで反転した信号より PMOS が ON になりダイナミックノードが VDD になるように電荷がたまる。

  入力信号の IN_1 と IN_2 は 0 のままである。

- エヴァリュエーションフェイズ

  このフェイズではプリチャージ信号が Low であり、PMOS が OFF になり VDD からダイナミックノードへの電荷の供給がされなくなる。

  入力信号の IN_1 と IN_2 には (10) または (01) が入力され、その入力に応じてダイナミックノードに蓄積された電荷が放電される。ダイナミックノードの論理が出力段のインバータによって反転され出力される。

二線式ドミノ回路を用いたエラーの回避

論理エラーの回避

論理エラーは、表 1.1 に示すように (1,1) の禁止状態になったときに起こる。これは二線の AND をとることによって得られる。図 1.6 のような回路においては、もしデータパスの途中で論理エラーが起きたとするとその出力は (1,1) となる。すると次段の入力において本来ならば (0,1) または (1,0) でなくてはならないのに対し (1,1) が入ってくる。このとき出力が入力に影響を受けないような信号のパター

ンであればロジカルマスキングされこの入力は次の段に影響を与えない。しかしこの入力が出力に影響を与えるようであれば (1,1) は伝播していく。最終的な論理エラーの判断は出力の DFF(Delay Flip Flop) に入ってくる時点で行なう。全ての二線のペアに対し AND をとることで論理エラーの有無を判定し、それらの中に一つでも論理エラーがあれば最終的に論理エラーが発生していたと判定するので、それぞれの OR をとることでエラー信号は以下のように定義できる。

$$Error = (a_1\_1 * a_1\_2) + (b_2\_1 * b_2\_2) + ... + (z_n\_1 * z_n\_2)$$

この Error 信号が 1 であれば論理エラーを検出したことになる。論理エラーを検出することができれば、レジスタに値を書き込まずに再度計算したり、または電圧を上げることでノイズマージンを増やして再度計算するなどを行なうことで論理エラーが起きないようにして演算をすることが可能になると考えられる。

遅延エラーの回避

遅延エラーに関しては、DCVSL を用いた場合には演算の終了検出を行なった後にラッチを行なう、つまり自己同期的に動作させるため遅延エラーは存在しない。図 1.6 のような回路で、あるゲートにおいて初期状態の出力は (0,0) であり、この状態で前段から信号が伝播してくると論理エラーが起きない限り、入力に (1,0) または (0,1) が入ってくる。すると出力が (1,0) または (0,1) となりこれが次段の入力となる。このようにして伝播し最終的に DFF に入る時点でもともとは (0,0) であった全ての信号の出力が (1,0) または (0,1) になる。つまり (0,0) から (0,1) または (1,0) に変化したことを検出すればよいので、終了検出は全ての二線のペアに対し OR をとることで終了したかどうかを判定し、それらのペアが全て終了していれば全ての演算が終了したと判定できるので、それぞれの AND をとることで終了信号は以下のように定義できる。

$$Completion = (a_1\_1 + a_1\_2) * (b_2\_1 + b_2\_2) * ..... * (z_n\_1 + z_n\_2)$$

この Completion が 1 になったときに初めて回路全体の終了を検出したことになり、出力側の DFF をラッチする。終了を検出してからラッチを行なうので同期回路ではプロセスばらつきなどによりうまくラッチできずに遅延エラーとなる可能性があるものの終了検出型の自己同期回路では遅延エラーの起きる可能性はない。

また図 1.7 の左図で示されるように、同期回路の場合 CPU などで命令を連続実行させたときにはその遅延は演算の命令やデータに依存するために実行サイクルごとに異なる。そのため同期回路では確実に動作させるには遅延が最大になる場合にマージンを加えた時間からクロックの周波数を決めなくてはならない。これは設計の段階で動作速度を非常に悲観的に見積もっていることになる。一方自己

同期回路の場合には、各命令が終了したということを検出するために各サイクルにおいては余分に時間がかかるものの終了し次第次の命令を実行できるために動作速度は平均の値で評価できる。そのために自己同期型の回路の方が速度という面で見たときに優れていると言える。

1.4　本論文の構成

本論文の構成は次のようになっている。まず第二章で DCVSL とスタティック CMOS の性能を遅延や消費エネルギーの観点から比較する。続いて第三章では DCVSL とスタティック CMOS のエラーレートを比較するための回路を提案する。第四章においては、DCVSL を用いた CPU を用いていくつかの命令の遅延を温度、電圧、チップを変えて測定した結果を示す。さらに、第五章では CPU の電源電圧制御を行なうために設計した DC-DC Converter について論じる。最後に第六章では、結論を述べる。

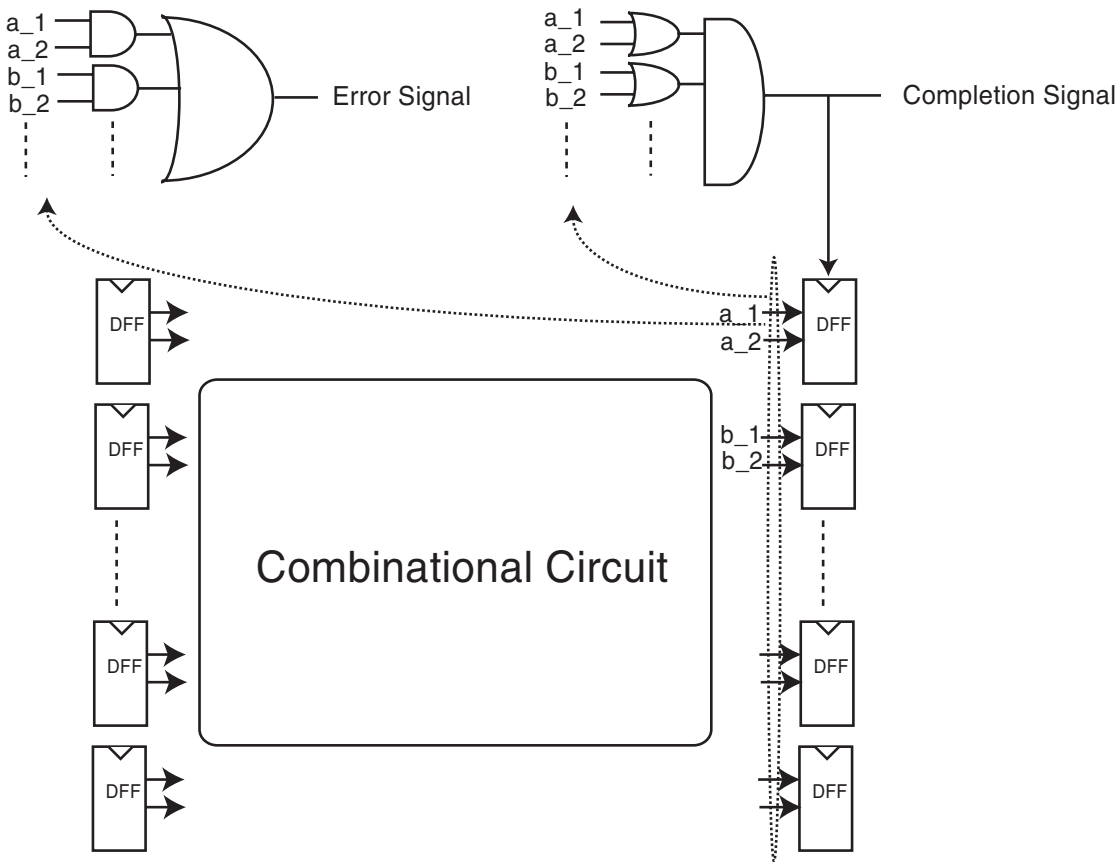


図 1.6: 論理エラーと遅延エラーの検出方法

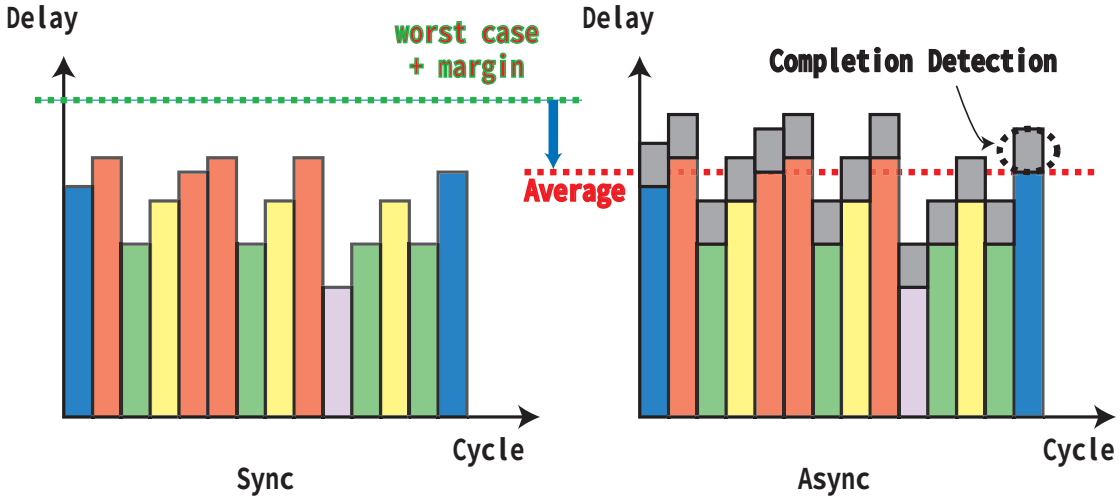


図 1.7: 同期回路と自己同期回路の動作速度

第2章　DCVSLとスタティックCMOSの性能比較

2.1　比較条件

現在LSIの設計において主流に用いられているのはスタティックCMOSである。そのためDCVSL、Footless DCVSLを用いたときにどの程度の性能の差があるかを知ることは重要である。DCVSLとFootless DCVSLの性能を評価するためにそれぞれNAND回路を用いてシミュレーションを行った。シミュレーションするための回路は図2.1のようになっている。ファンアウトを現実的な値である4とした。シミュレーションは90nmのテクノロジーで行なった。入力のスルーレートは予め多段にしてシミュレーションを行なうことで得た値を用いた。

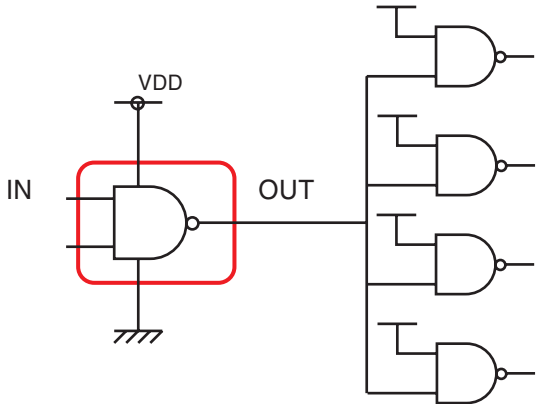


図 2.1: DCVSLとCMOSの性能比較のための回路

スルーレートの決定

回路の遅延は、入力のスルーレートと負荷容量によって決定される。そのためシミュレーションを行っていく上でのスルーレートを決める必要があり、図2.2のようにファンアウトを4としたNAND回路を直列に6段接続し、その5段目の出力のスルーレートを測定した。

スルーレートの定義は、信号が 0 → VDD(VDD → 0) に変化するときの 0.1VDD → 0.9VDD(0.9VDD → 0.1VDD) までにかかる時間である。入力信号は、図 2.2 の下図のようになっている。スタティック CMOS においては一方の入力を High にしておいて他方の入力を Low → High と変化させたときの 5 段目の出力のスルーを測定。DCVSL, Footless DCVSL においては、入力信号はプリチャージフェイズでは初期状態の (0,0) にしておいてエヴァリュエーションフェイズに入ったのちに (1,0) とした。このスルーレートのシミュレーションは電源電圧を下げながら行った。その結果は図 2.3 のようになっている。

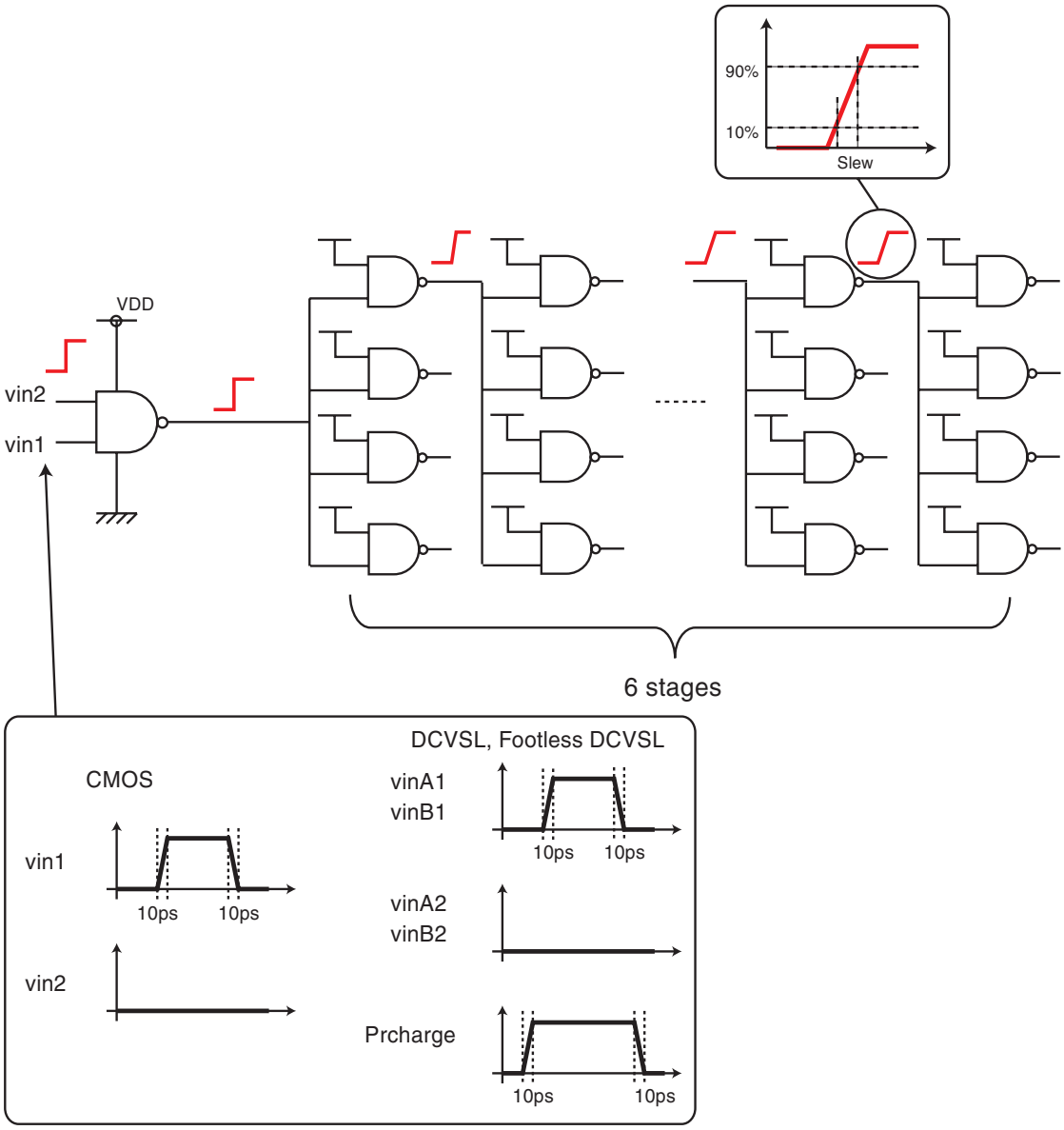


図 2.2: スルーレート決定のための回路

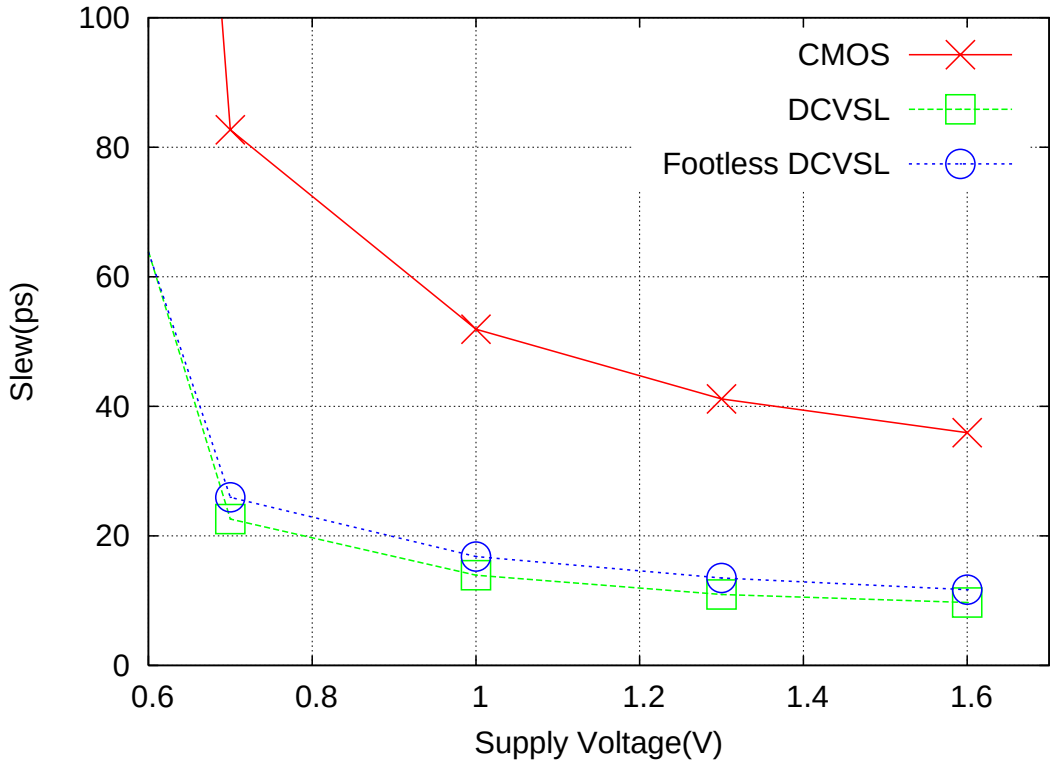


図 2.3: スルーレート

2.2　シミュレーションによる比較結果

DCVSL、Footless DCVSL のノイズ耐性

エラーには論理エラーと遅延エラーがあることを第一章で述べた。遅延エラーとは、DFF のラッチの際に値が定まっていないために正しい値が取り込まれずにエラーとなることで、この遅延エラーはクロックの周波数を落とせば正しく動作する。これはスタティック CMOS、DCVSL、Footless DCVSL に関わらず起きる。

論理エラーは、遅延エラーとは違い周波数を落としても正しく動作しないようなエラーである。スタティック CMOS では、ロジック部分が pull up と pull down により成り立っているので時間が経てば必ず正しい論理が出力される。一方 DCVSL と Footless DCVSL ではロジック部分が pull down のみによっているために、誤って pull down された場合 pull up をすることができない。

DCVSL の例を用いて具体的に説明する。図 2.4 においてプリチャージ終了時にはダイナミックノードの寄生容量に電荷が蓄積され電位は VDD となっている。エヴァリュエーションフェイズに入り、本来は入力の A_1、B_1 は 0 と 0 であったとすると直列側のダイナミックノードの電位は VDD に保たれるはずである。しかし、

ここで A_1、B_1 にノイズがのってしまったとすると本来 OFF であるべき NMOS が ON してしまい保持されるべき電荷が放電されダイナミックノードの電位が 0 になってしまうことがある。ダイナミック回路であるため一度放電されてしまうと二度と pull up できないために論理エラーとなる。

ノイズ耐性を調べるために、ノイズを持った電圧を入力に入れ、どの程度でロジックエラーが起きるかを調べた。ノイズとして入力信号に sin 波をのせたものを用いた。

$$Vnoise = Vac * \sin(2\pi ft)$$

として、電源電圧とノイズの振幅を変えたときにロジックエラーの起きる状態を調べると図 2.5、図 2.6 のようになる。さらに周波数を変えエラーの起きる振幅も調べた。

DCVSL、Footless DCVSL ともに電源電圧が下がるとともにエラーの起きるノイズの電圧も下がっていく。また、周波数が低くなっていくにつれ、エラーを起こすノイズの電圧の振幅も小さくなる。これは、周波数が低いときには NMOS の閾値を超える時間が一定期間ありその間にダイナミックノードに蓄積された電荷が放電されるためである。

DCVSL と Footless DCVSL を比較すると、Footless DCVSL の方がノイズの電圧が小さくてもエラーが起きている。これは、Foot トランジスタを取り除いてしまったために電流が流れやすくなってしまったためだと考えられる。

1V の電源電圧に対して、入力に 0.4V 程のノイズが載らない限りエラーは起きないが、ダイナミック回路においては、ハザードなどが発生しないので、ロジックエラーは起き難いといえる。

図 2.5、図 2.6 から、1V の電源電圧に対して 0.4V 以上のノイズがのった場合にロジックエラーが起こる可能性が高いことが分かる。しかしながら、入力に 0.4 v 程度のノイズがのることはまれであり実用上十分な論理エラー耐性があることが示された。

回路の遅延

次に CMOS, DCVSL, Footles DCVSL 回路の遅延のシミュレーションを行った。遅延の定義は、入力信号が 0.5VDD になったときから出力信号が 0.5VDD になるまでの時間とした。さらに電源の定格電圧は 1V であるが、0.6V から 1.5V まで変化させた。この遅延のシミュレーションの結果は図 2.7 のようになっている。CMOS においては立ち上がり遅延と立下り遅延の平均をとり、DCVSL、Footless DCVSL においてはロジック部分の並列側と直列側の平均をとった。1V において、DCVSL と Footless DCVSL は CMOS に比べそれぞれ 28 %、38 %速く動作する。ただし、

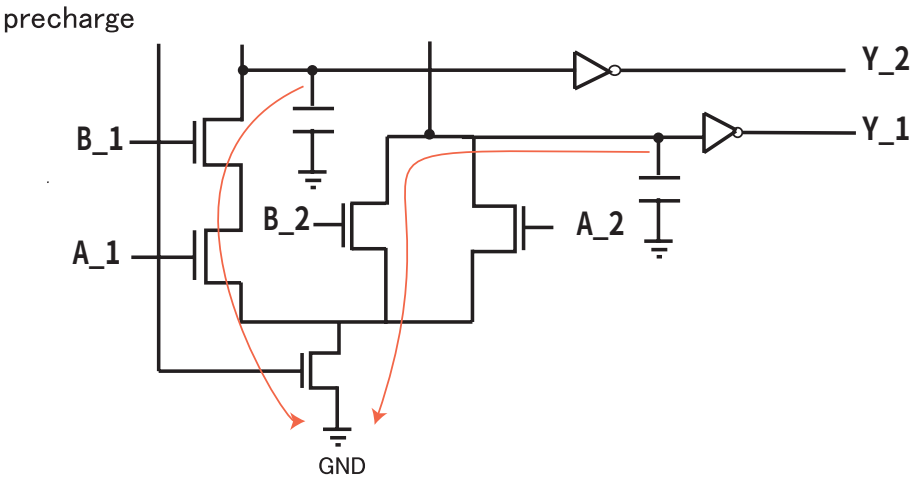


図 2.4: 論理エラー

後者はプリチャージの時間を考慮しない結果であり実際の使用においては同等の高速化を実現することはできない、しかし多段で使用したり、プリチャージを自己同期的に行なう手法 [7] などを用いることでプリチャージ時間の全体の遅延に占める割合が小さくすることができる。以上のことより DCVSL 方式が CMOS と比較して十分な遅延性能を有していることが示された。

消費エネルギー

それぞれの回路の消費エネルギーを比較すると図 2.8 のようになった。

算出方法として、CMOS は立ち上がり時と立下り時の平均を用いた。一般に CMOS 回路の消費電力は遷移確率を α ゲートの出力によって駆動されるキャパシタの合計を C_{total}、電源電圧を V_{DD}、動作周波数 f として以下の式で表すことができる。

$$Power \propto \alpha C_{total} {V_{DD}}^2 f$$

このように周波数と遷移確率に依存するが、消費エネルギーで評価するので、今回の実験では遷移確率を 1 として一度遷移するときの消費エネルギーを求めた。つまり、毎クロック遷移が起きるときの消費エネルギーとなっている。DCVSL と Footless DCVSL は図 1.3 の Y1、Y2 で消費されるエネルギーの平均をとった。CMOS の時とは違い、毎クロック必ず Precharge と Evaluation を行うので遷移確率などをかける必要はない。

消費電力の評価では CMOS、DCVSL、Footless DCVSL の順になる。1V において、DCVSL、Footless DCVSL の消費電力は CMOS のそれに比べてそれぞれ 2、4

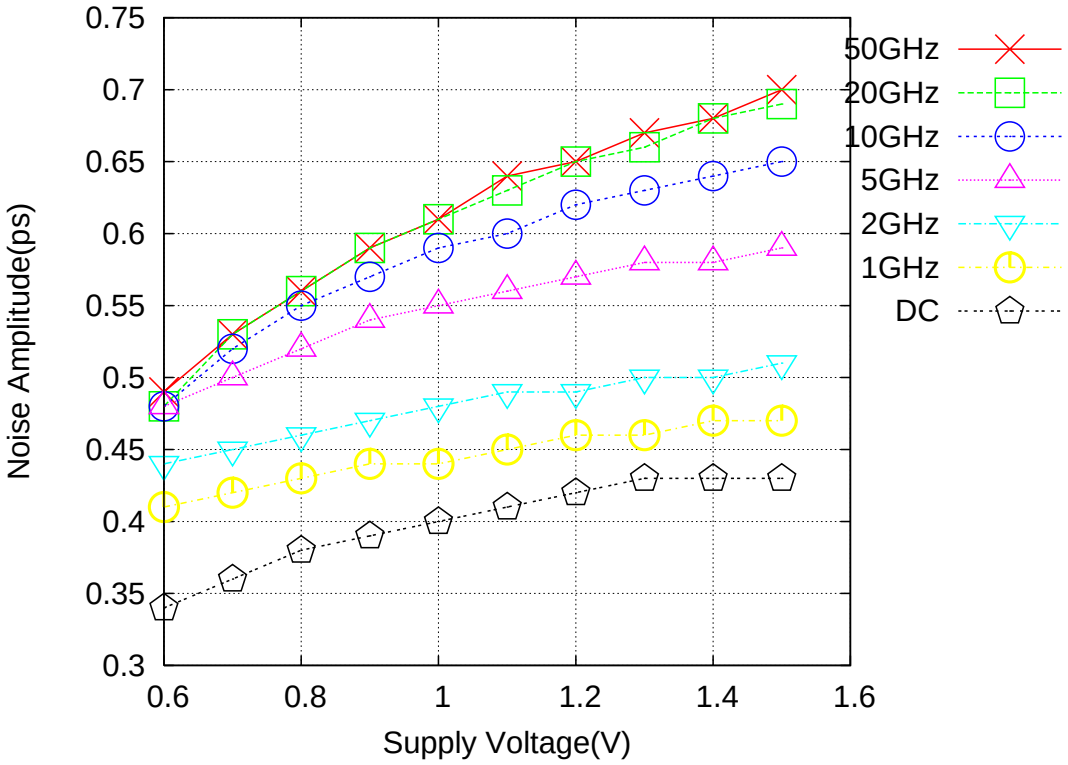


図 2.5: DCVSL：電源電圧 - ノイズの振幅

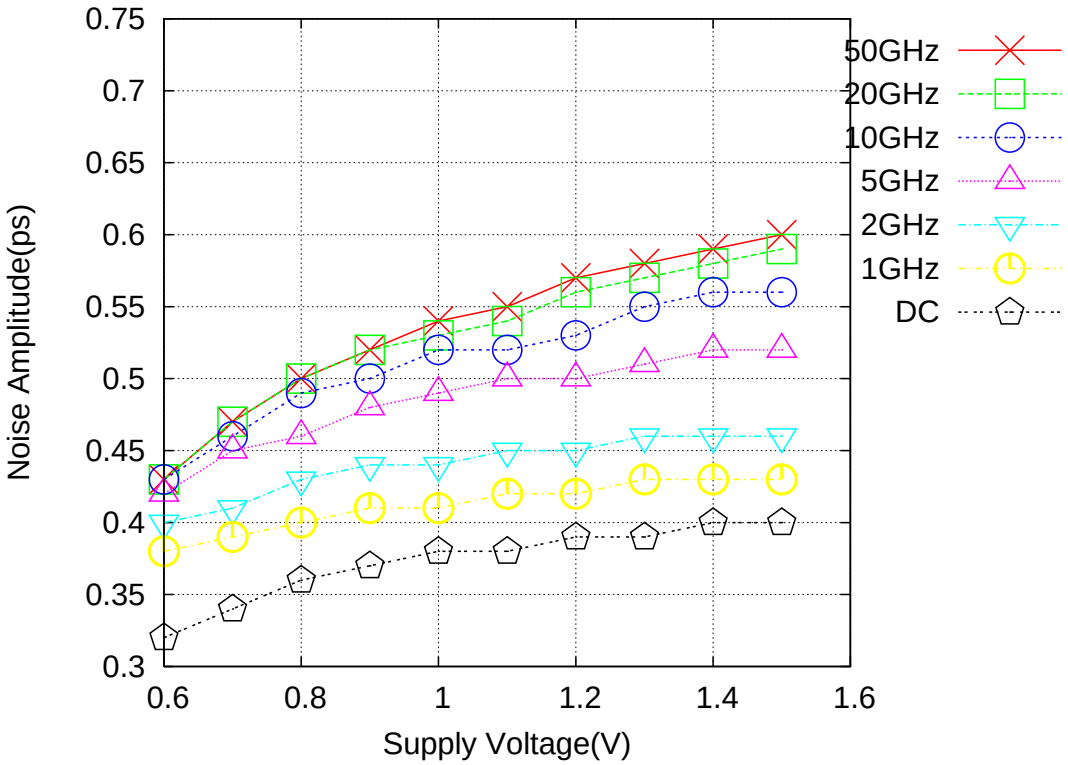


図 2.6: Footless DCVSL：電源電圧 - ノイズの振幅

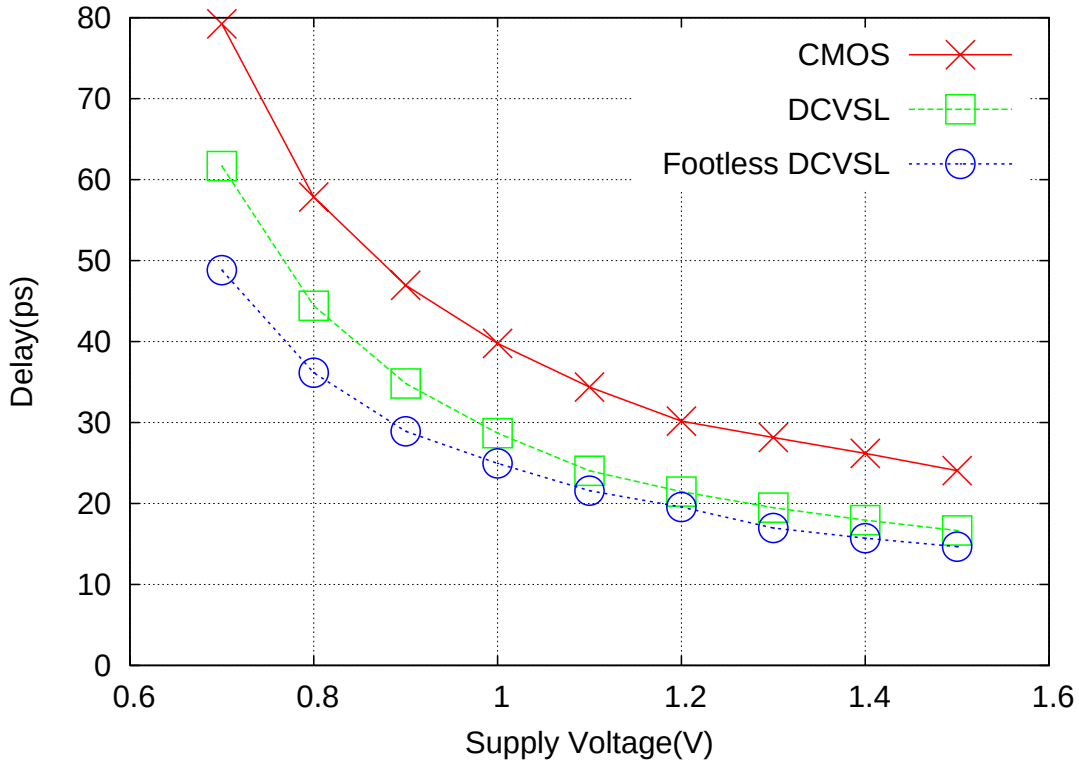


図 2.7: 電源電圧 - 遅延のグラフ

倍となった。この原因として DCVSL、Footless DCVSL はスタティック CMOS と比較して駆動しなければならないトランジスタ数が多いことが考えられる。消費電力においては CMOS と比較して大きな値となったが、CMOS がエラー訂正方式として多数決回路を採用した場合には同等程度の消費電力になると考えられる。

電源電圧の変動に伴う遅延の解析

電源電圧に sin 波をのせてそのときの遅延の変動をシミュレーションした。

$$V_{DD} = Vdc + 0.1\sin(2\pi ft + \theta)$$

Vdc を変化させたときの遅延の変化を示したグラフは、図 2.9 となった。ノイズがのったときの遅延の変動の標準偏差をエラーバーとして記している。Vdc を下げるにつれて、電源電圧 V_{DD} に占める振幅の割合が大きくなるために遅延の変動は大きくなる。1V での動作においては、DCVSL、Footless DCVSL の標準偏差はともに CMOS の半分程度であるが、これは回路の遅延が CMOS に比べて小さかったからに起因する。さらに実際に回路として使用するときには CMOS と DCVSL では、それぞれが放出するノイズの大きさも違うと考えられるので、これから DCVSL の

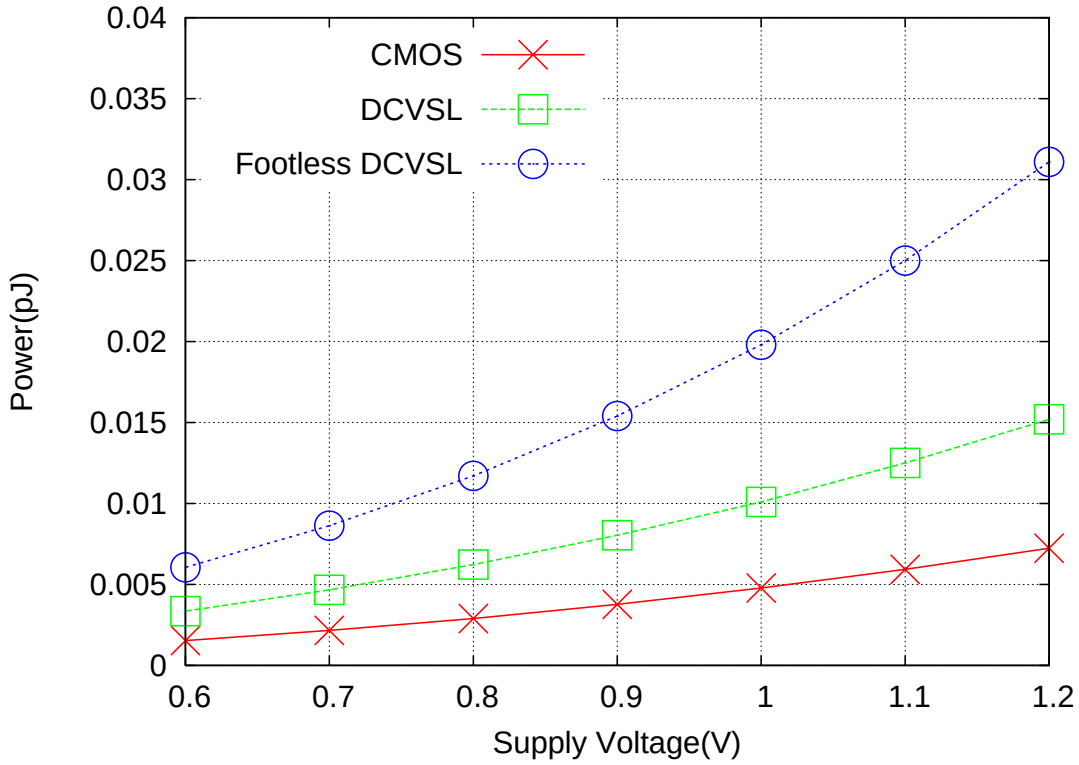


図 2.8: 電源電圧 - 消費エネルギーのグラフ

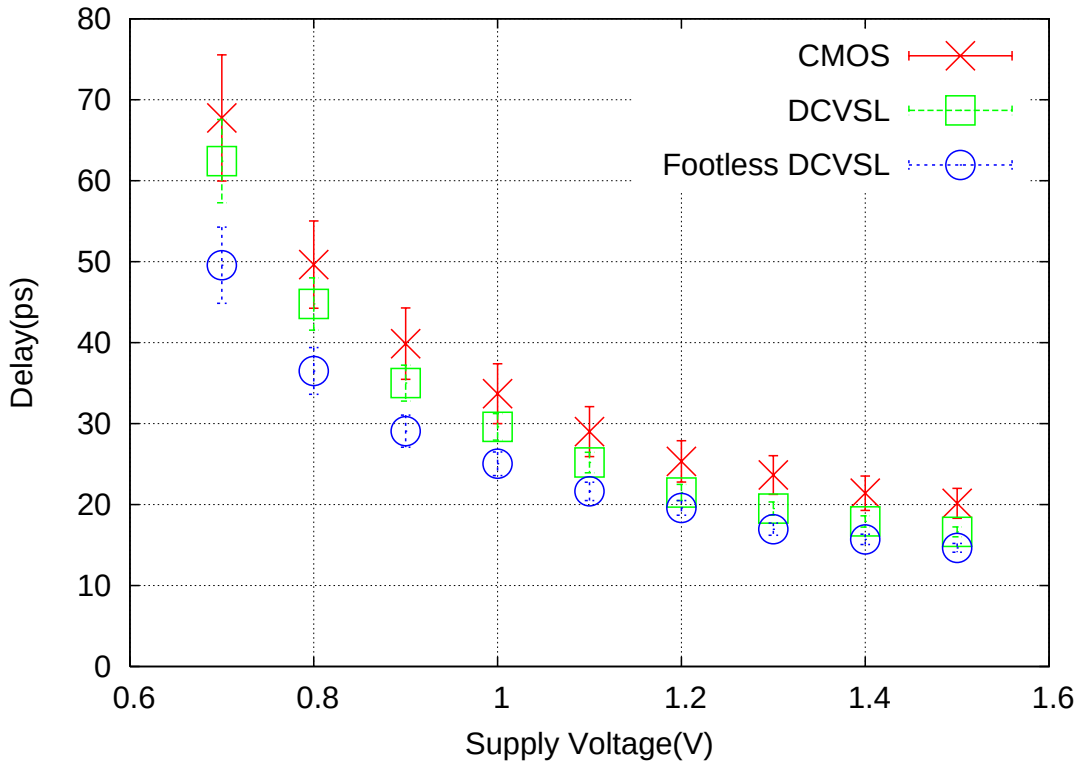


図 2.9: 電源電圧変動時の遅延のばらつき

方が遅延の変動が小さくなるとは導くことはできないが、自己同期方式では動作時に放出するノイズも小さくなると予想されるので、より DCVSL が有効であると考えられる。

基板バイアスの変化に伴う遅延の解析

基板のバイアスを変えることで回路の速度を変えることができ、基板バイアスを変えて回路の遅延を制御し歩留まりを上げる研究などもなされている [8][9]。そこで DCVSL とスタティック CMOS においてどの程度速度を変えることができるか知るためにこのシミュレーションを行なった。

今までのシミュレーションは通常の CMOS と同じで PMOS の基板は VDD、NMOS であれば GND に接続して行なった。ここでは、基板バイアス効果によってそれぞれの回路の遅延がどの程度変化するかを調べるために、基板にバイアスを与えてシミュレーションを行った。

シミュレーションを行う上で、PMOS の基板の電圧を Vbp とし、NMOS の基板の電圧を Vbn とする。そして以下の範囲で Vbp、Vbn の値を変える。

$$-0.5 < Vbn < 1$$

$$0 < Vbp < 1.5$$

基板バイアスを変えたときの遅延は、CMOS、DCVSL、Footless DCVSL のときにそれぞれ図 2.10 の (a)、(b)、(c) のようになる。

CMOS の場合は立ち上がり遅延と立下り遅延の平均、DCVSL、Footless DCVSL の場合は直列部と並列部の遅延の平均をとっている。グラフから基板バイアスが 0.1V 変動したとき CMOS、DCVSL、Footless DCVSL のどの方式においても遅延が 4 %程度ばらつくことが分かる。

シミュレーション結果のまとめ

以上のシミュレーション結果をまとめると表 2.1 のようになる。DCVSL、Footless DCVSL はスタティック CMOS に比べて高速に動作し、さらに電源や、基板バイアスが変動したときの遅延に与える影響も DCVSL、Footless DCVSL がスタティック CMOS に比べて小さい。

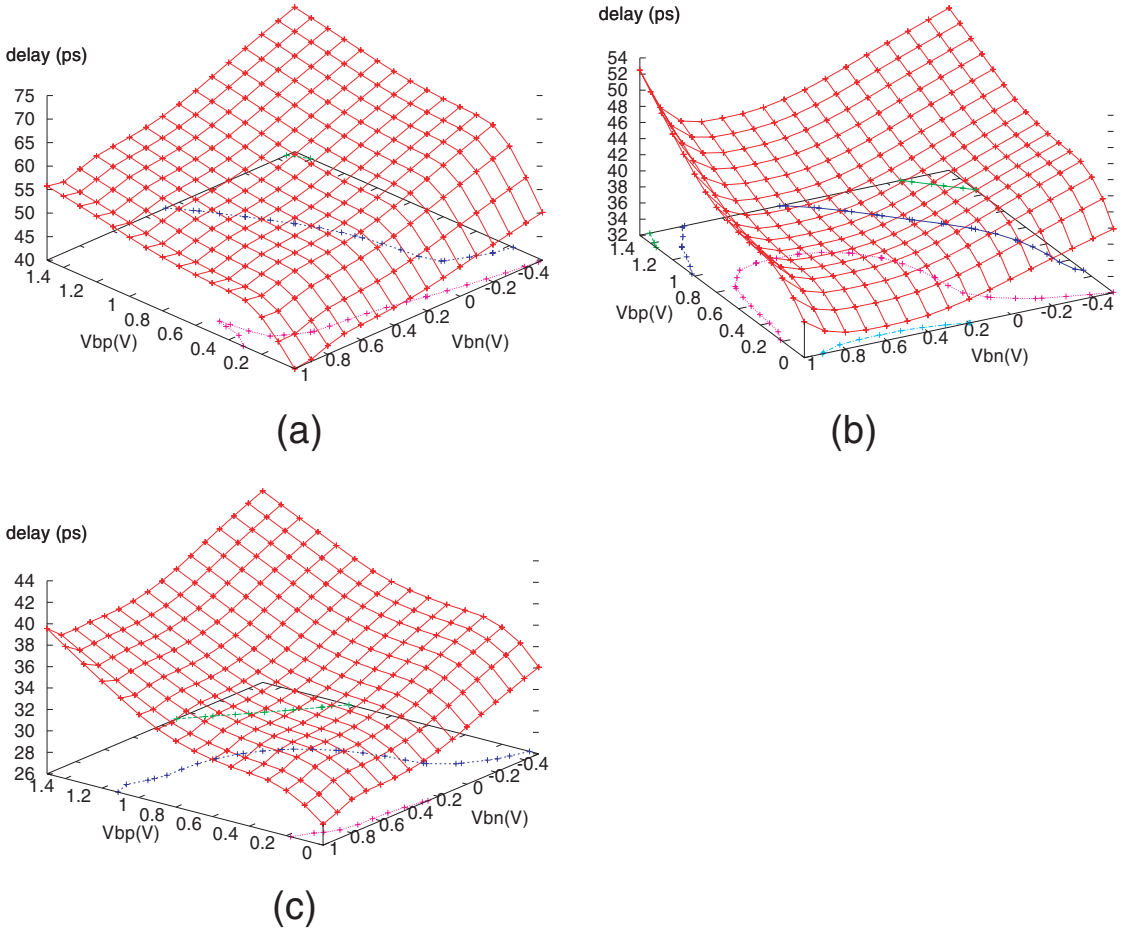


図 2.10: 基板バイアス - 遅延　(a)CMOS, (b)DCVSL,(c)Footless DCVSL

表 2.1: シミュレーション結果のまとめ

	CMOS	DCVSL	Footless DCVSL
入力ノイズ耐性		0.4V	0.38V
遅延	34.7 ps	28.6 ps	24.9 ps
電源ノイズ遅延ばらつき	11 %	5 %	5 %
基板バイアス遅延ばらつき	4 %	4 %	4 %
消費エネルギー	4.7 (fJ)	10.1(fJ)	19.8(fJ)

第3章　DCVSLとスタティックCMOSのエラーレート比較回路

DCVSL の性能がスタティック CMOS と比較して同等であることを二章で示した。しかしながら回路が実際にエラーが起きるかということは、シミュレーションではノイズやプロセスばらつきなどを考慮して行なわなければならないために難しい。そこで本章では実際のチップを用いて両者のエラーレートを比較する回路を提案する。

3.1　エラーレート比較方法

同期回路、終了検出型自己同期回路のエラーレートを比較を行なうことのできる回路を実装し、それぞれのエラーレートを測定する。

同期回路ではスタティック CMOS を用い、終了検出型自己同期回路では二線式ドミノ回路である DCVSL を用いる。

エラーレートの比較は図 3.1 のようにして行い、被加算数 A,B とその和 S を予め用意しておき実際に A+B を計算したものと、S を比較することで行なう。Adder の部分をそれぞれの回路方式によって変えて実装した。

１つの演算には４サイクルを使う。それぞれのサイクルは以下のようになっている。

- PC

  ROM のためのアドレスを生成する。

- ROM

  被加算数 A,B とその和 S を得る。

- ADDER

  被加算数 A,B の和を計算する。

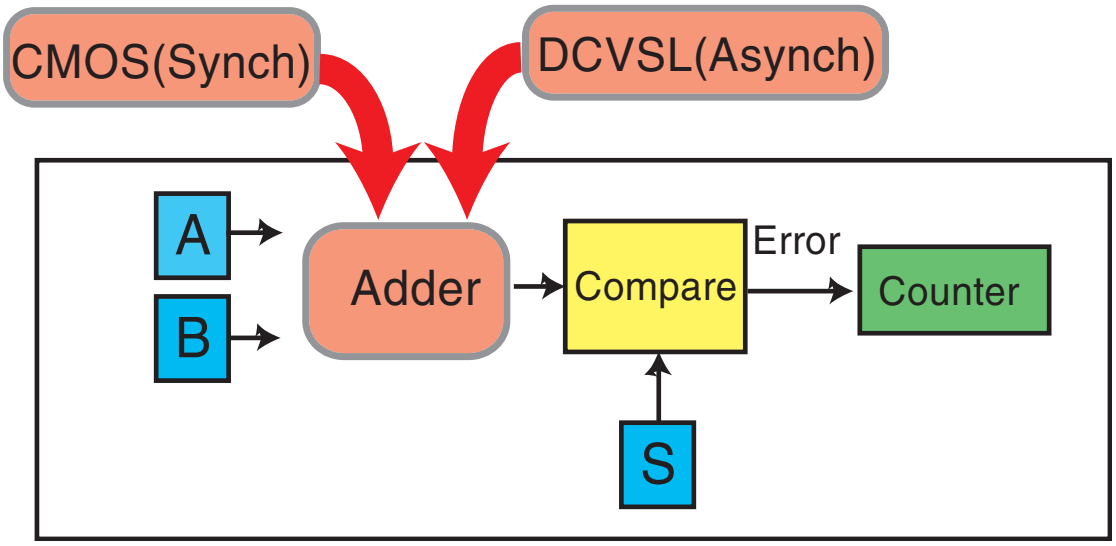


図 3.1: エラーレート測定回路の概念図

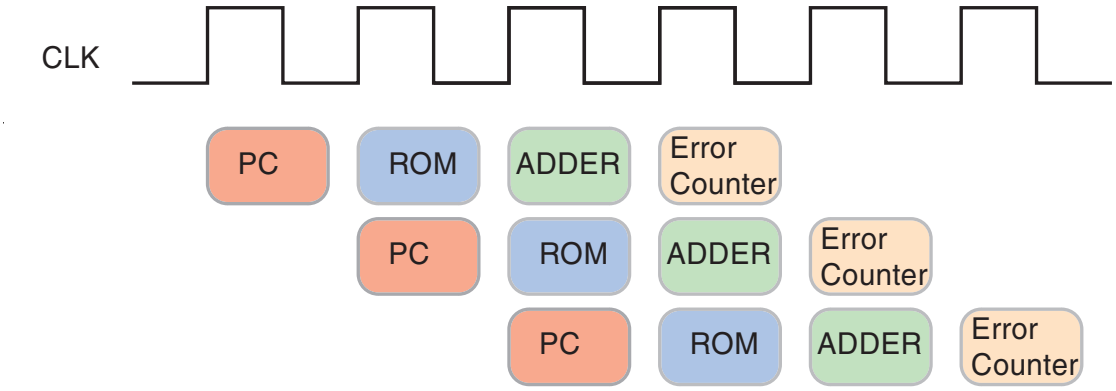


図 3.2: エラーレート測定回路のタイミングチャート

- Error Counter

  ADDER で計算された和と ROM に格納されていた和を比較する。

これらの 4 つのことをタイミングチャート図 3.2 で示すように並列で行なう。初めのクロックでプログラムカウンタでアドレスを生成し、次のクロックで ROM から値を取り出す。その次のクロックで加算器で和を計算し、最後に計算結果の是非を比較する、という手順で行なう。

そのエラーレートは以下のように条件を変えて比較する。

- それぞれの回路の電源電圧を下げる
- それぞれの回路の電源にノイズをのせる

終了検出型自己同期回路ではエラーリカバリーができるということで、この回路方式が優れていることを示す。設計は 90nm のプロセスを用いて行った。

3.2　比較する回路方式

スタティック CMOS：同期回路

同期回路を用いたエラー検出回路は図 3.3 のようになっている。まず、sync_clkgen によって回路中のクロックが生成される。クロックは PC に入り、その立ち上がりごとに出力が 1 ずつインクリメントされる。カウンタの出力はメモリのアドレスとなり、メモリに入力される。つまり、1 クロックごとにメモリの番地に次々アクセスしていくことになる。

メモリの出力は 24bit であり。それぞれ A[7:0]、B[7:0]、S[7:0] とすると、$A+B=S$ の関係になるように A、B には被加算数、S にはその結果が入っている。

メモリの出力のうち A、B は DFF に入る、そして実際に加算器によって $A+B$ の演算がなされ出力される。この出力された結果とメモリからの値 S が、比較され等しくない場合にはエラーカウンタによってエラーがカウントされる。この加算器はスタティック CMOS によって構成されている。

カウンタの出力は 16bit 幅であるので、それをパラレルシリアル変換によってシリアルに変換され外部に出力される。

電源は、2 系統用意しておく 1 つは定格電圧の 1V(VDD) で、もう一つは電圧を変えたりノイズをのせたりできるもので、VDD_noise である。この VDD_noise は図 3.3 中に記しており、8bit CLA とその前後の DFF に接続されている。また、VDD_noise は外部から調整可能であるが、より実際に近いノイズを作るために内部にノイズ源を実装している。

また比較のために、クロックに遅延線を持たせたレプリカ同期回路も CMOS で実装した。

DCVSL：自己同期回路

終了検出回路は図 3.4 のようになっている。この回路も同期回路とほぼ同じであるが、回路を自己同期的に使用するために CLA とクロックの生成部が異なる。

8bit CLA については、DCVSL を用いて設計した。DCVSL は 2 線式であるので、入力側の DFF のところで 1 線式から 2 線式への変換を行い、その後に演算を行う。

終了検出回路には論理エラー検出、タイムアウト検出も備えている。

終了信号は全てのビットにおいて 2 線のうちのどちらかが $0 \rightarrow 1$ に遷移するこ

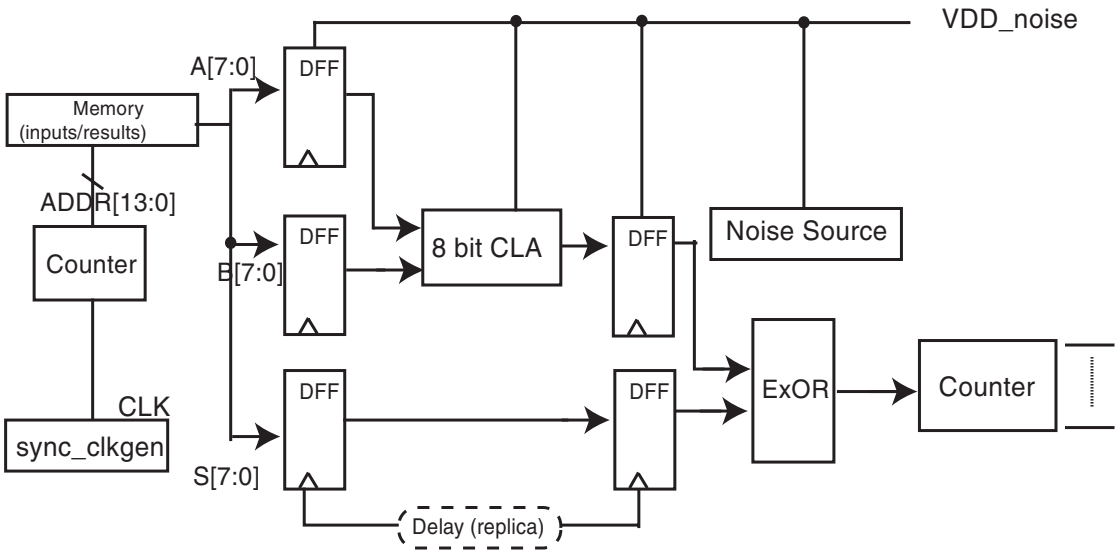


図 3.3: スタティック CMOS を用いた同期回路の回路図

と、つまり (0,0) から (0,1) または (1,0) になったことで検出できるので第一章で説明したように次の式で表される。

$$Comp = (S_1\_1 + S_1\_2) * (S_2\_1 + S_2\_2) * ..... * (S_7\_1 + S_7\_2)$$

エラー検出は、8bit のうち少なくとも 1bit で演算結果が (1,1) になったときであるので以下の式のように表される。

$$Error = (S_1\_1 * S_1\_2) + (S_2\_1 * S_2\_2) + ... + (S_7\_1 * S_7\_2)$$

タイムタイムアウト検出は、演算の出力が (00) のまま変化しないときであり、これはクロックの遷移をカウンタでカウントし、4 周期分経っても計算が終わらないときにはタイムアウトとなるようにした。

$$TimeOut = 4 * T_{clk}$$

この終了信号またはエラー信号またはタイムアウト信号が出力されると、図 3.4 のように、その信号が Async_clkgen に入り次のクロックが生成される。同期回路やレプリカ同期回路と異なり、自らの終了信号によって次のクロックが自己同期的に作られる。

また、クロックの制御信号を追加することで、終了信号に関係なく同期回路のようにクロックを利用することもできるようにした。

また、エラー、タイムアウトが起きたときには 16bit のカウンタによってエラーの起きた数をカウントしている。

このエラーの検出の容易さが DCVSL の特長である。

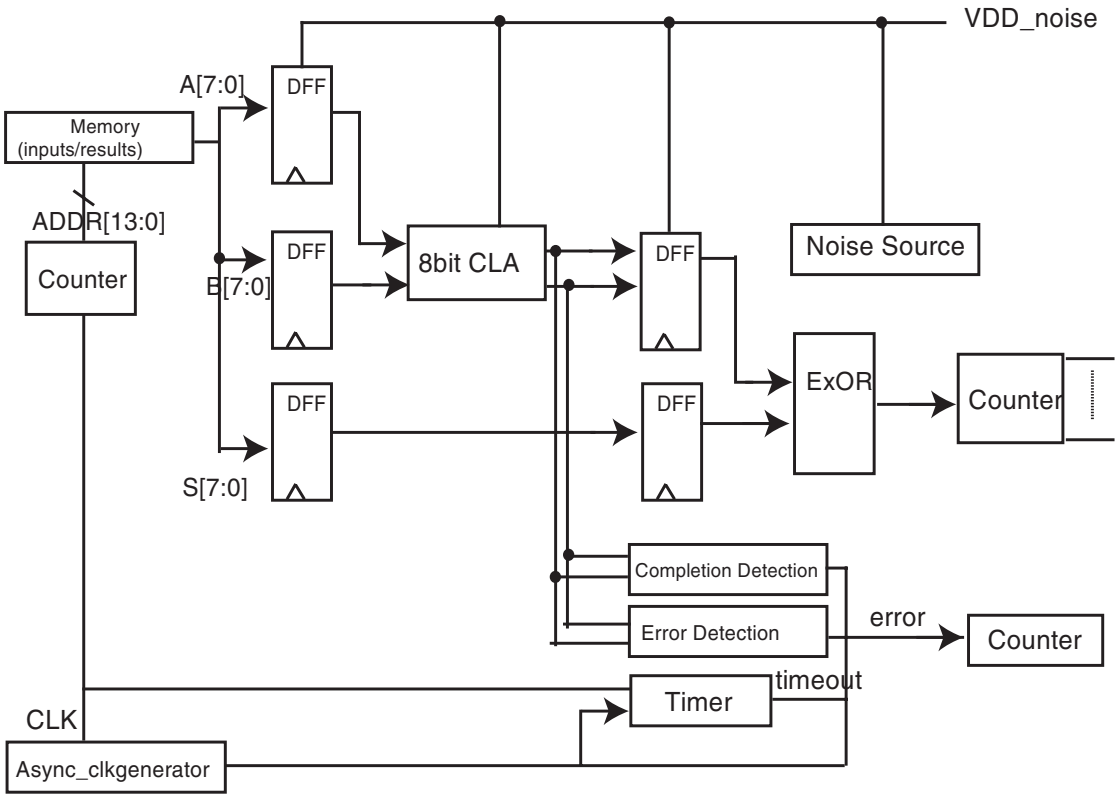


図 3.4: DCVSL を用いた自己同期回路の回路図

3.3　回路構成

VCO (Voltage Controlled Oscillator)

回路内のクロックを作り出すために VCO を設計した。まず、図 3.5 のようにインバータを 7 段接続しリングオシレータを作った。ただし、インバータの遅延をバイアス電圧で制御できるように、通常のインバータの PMOS と VDD の間に PMOS を挿入し、バイアス電圧を与えられるようにした。また、入力のバイアス電圧はカレントミラーとしている。

このようにして作成されたクロックを図 3.5 のようにして分周して回路の内部クロックとして出力できるようにした。VCO によって作成されたクロックを clk、$\frac{clk}{2}$、$\frac{clk}{4}$、他に外部からクロックを直接入れており、セレクタによって以上の 4 つのクロックから、回路の内部で使用するクロックを選択できるようにしている。

これらを通じて、160MHz〜2GHz の内部クロックを得ることができる。

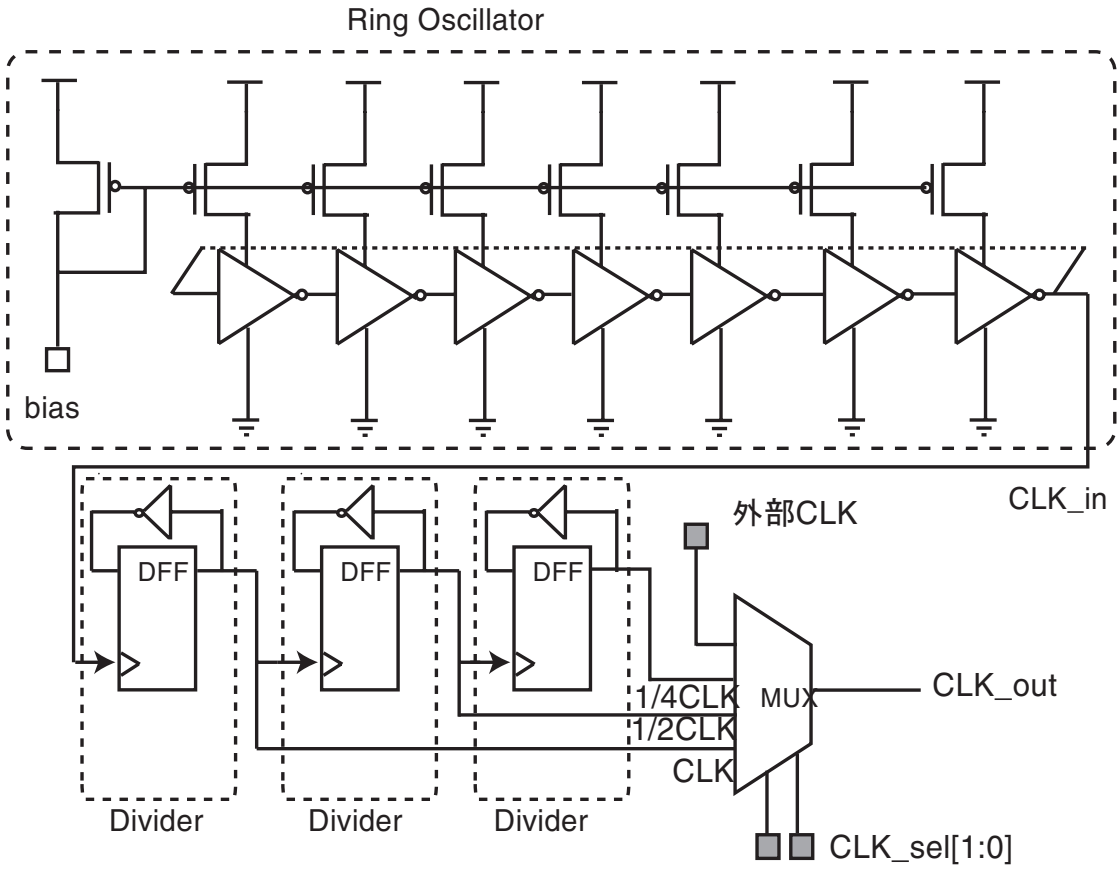


図 3.5: リングオシレータと分周器

ノイズ源

デジタル回路で発生するノイズを仮定し以前当研究室で設計されたノイズ源 [1] と、図 3.6 のようにして擬似的にランダムなノイズが発生するようなノイズ源を載せた。

- 当研究室で設計されたノイズ源

  DFF を 10 個直列に接続し、VCO で作られたクロックを分周した信号によって DFF の出力につながれているインバーターチェーンを ON するか OFF するかを決定することでランダムなノイズを発生させる仕組みになっている。

  それぞれの DFF の出力には、インバータが 30 個あるいは 60 個直列に接続され、Noise_select 信号によってノイズの大きさを選択できるようになっている。また、イネーブル信号によってノイズ源を使用するか否も選択できる。

  ノイズの大きさは 0.2V 程度になるようにした。

- PRBS を利用したノイズ源

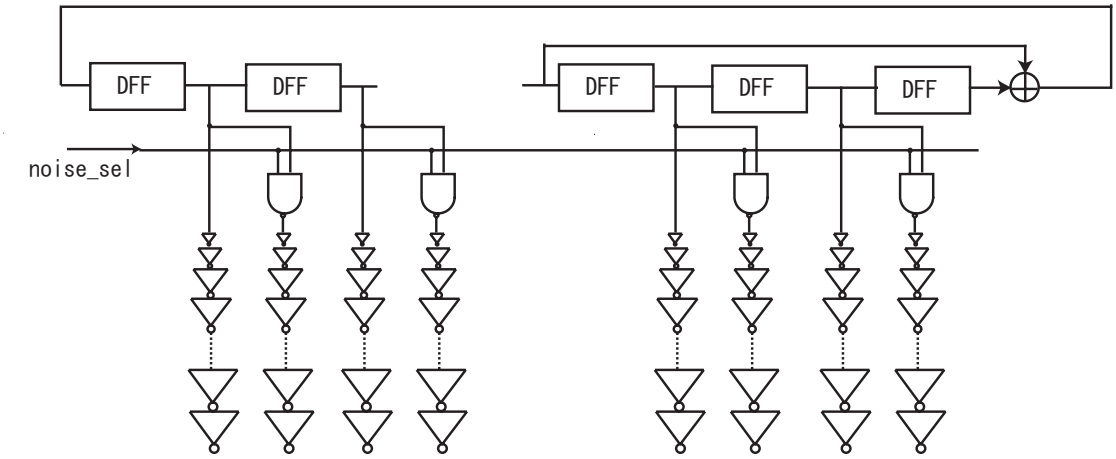


図 3.6: ノイズ源の回路図

PRBS(Pseudo-Random Bit Sequence) [11] を応用して、ランダムなノイズが作成されるようにした。

DFF を 10 個直列に接続し、それぞれの DFF の出力には、インバータが 30 個あるいは 60 個直列に接続され、Noise_select 信号によってノイズの大きさを選択できるようになっている。その 7 個目の出力と 10 個目の出力の EXOR をとりその出力を 1 つ目の DFF にフィードバックしている。

このようにすることで擬似的にランダムなパターンを作ることができる。この擬似的なノイズのパターンは $2^{10}-1$ 周期で繰り返される。

8bit CLA　(Carry Look-ahead Adder)

加算器としては 8bit の carry lookahead adder を設計した。

8bit の carry lookahead adder については、図 3.7 のような構成にした。「Creation of P and G」「Dot operator」「Sum generation」の 3 つの要素から成り立っており、以下それぞれの役割を説明する。

- Creation of P and G

  P(propagation) と G(generation) を算出する部分である。各ビットについて、P と G を計算する。図 3.7 の下の部分に相当する。以下のように計算できる。

$$P_i = A_i \oplus B_i \quad (0 \leq i \leq 7)$$

$$G_i = A_i * B_i \quad (0 \leq i \leq 7)$$

- Dot operator

  「Creation of P and G」で作った P と G をもとにして、キャリーを計算していく。

$$P_{i:j} = \begin{cases} P_i & (if\ i = j) \\ P_i * P_{i-1:j} & (if\ i > j) \end{cases}$$

$$G_{i:j} = \begin{cases} G_i & (if\ i = j) \\ G_i + P_i * G_{i-1:j} & (if\ i > j) \end{cases}$$

  ただし $P_{i:i} = P_i$, $G_{i:i} = G_i$

  また、キャリーは

$$C_i = G_{i-1:0}$$

  のようになる。

  この式にそって再帰的に求めていけばよいのだが、Dot operator(Boolean operator) 「$\circ$」を以下のように定義することで簡単に計算ができる。

$$(P, G) \circ (\widetilde{P}, \widetilde{G}) = (P * \widetilde{P}, G + P * \widetilde{G})$$

  この Dot operator を図 3.7 のように配置することで、視覚的にも分かりやすく Propagation と Generation を計算できる。

- Sum generation

  各 bit の和 S_i は、それぞれの Propagation とキャリーの Exor をとることで算出でき、図 3.7 の最終段に配置される。

$$S_i = P_{i:0} \oplus C_i = P_{i:0} \oplus G_{i-1:0}$$

ROM と PC

　今回は一定時間連続で回路を動作させその間に起きたエラーを測定するというのが目的であるので、予め計算に使用するデータを ROM に入れておいて使用することにした。

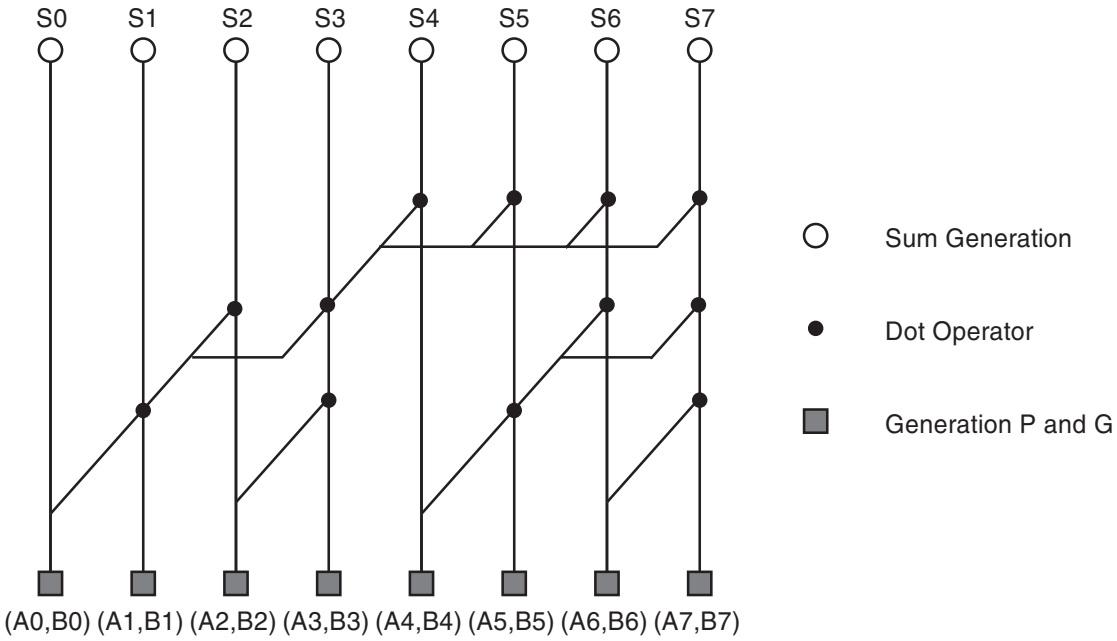


図 3.7: 8bit Carry Look-ahead Adder の構成

ROM は 1word 24bit とし、2^{14} word の大きさのものを用意した。

1word には被加算の値 A[7:0], B[7:0] とその加算の結果 S[7:0] の計 24bit が入っている。入力は、clk、A[13:0](アドレス)、CEN(イネーブル信号) で、出力は Q[23:0] となってる。

ROM の動作は、図 3.8 のようになっており、CEN が 0 になっている状態で、クロックの立ち上がりエッジを感知すると、そのアドレスに入っている値が読み出され Q[23:0] から出力される。この ROM の最大動作周波数は 500MHz である。ROM の 1word の 24bit には、A[7:0]、B[7:0]、S[7:0] が $A + B = S$ の関係を満たすように入っている。

また ROM のアドレスを作る PC を設計した。これは 14bit のカウンタであり、CLK_mem が来るたびに出力の値が 1 ずつインクリメントされる。つまり ROM のアドレスが 1 ずつインクリメントされアクセスされる番地をずらしていく。

エラーカウンタ、パラレルシリアル変換

エラー信号を入力として、クロックの立ち上がりエッジでエラー信号が 1 であればエラーをカウントするように 16bit のエラーカウンタを設計した。図 3.9 の左のようになっており、クロックが 1GHz 程度まではカウントでき ROM の仕様から 500MHz 以上で動作させることは無いので十分な性能のものとなっている。

エラーをカウントしたカウンタの出力は 16bit が並列になっているが、ピンの数

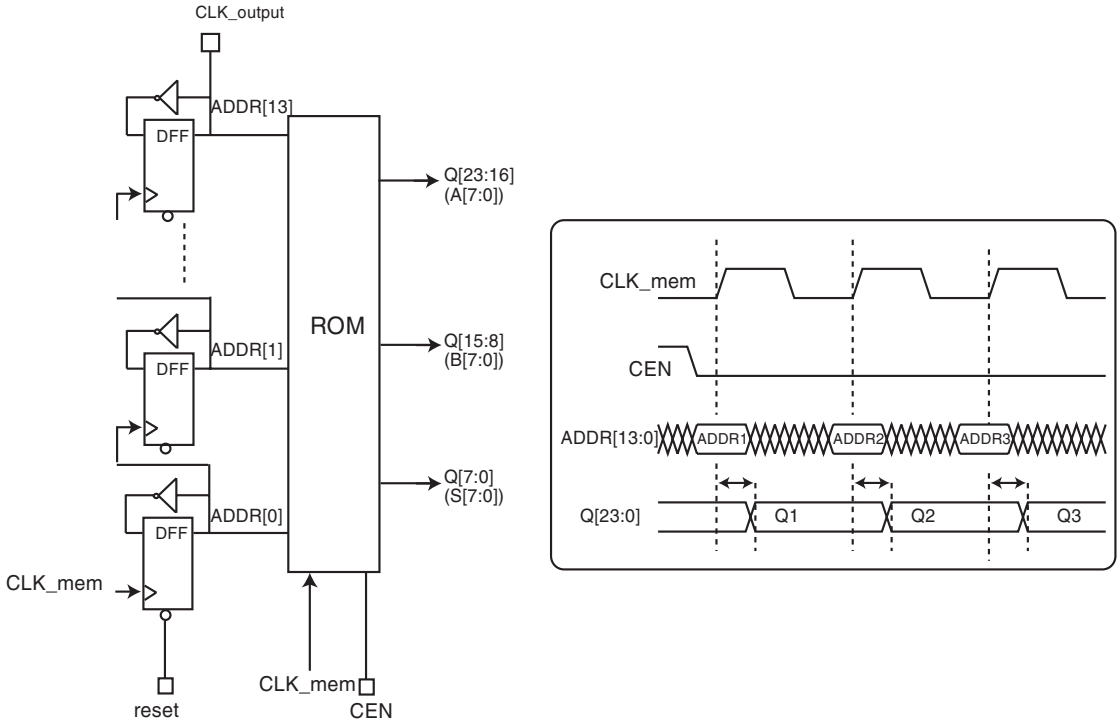


図 3.8: PC、ROM の回路図とタイミングチャート

が限られているので直列に変更して出力した。

パラレルシリアル変換は、スキャンレジスタを図 3.9 の接続して使用する。SMC 信号が Low の時は、通常のレジスタのように動作する。High になると、シフトレジスタとして働く。SHIFT 信号が立ち上がる度に 1bit ずつ信号が上位 bit へシフトしていく。

3.4　レイアウト

以上で説明したモジュールを ASPLA90nm テクノロジを用いて実装した。設計したチップのレイアウトと仕様は図 3.10 のようになる。

3.5　検証

まず、設計したチップと同等の動作をするものを RTL で書き検証を行なった。次に設計した各モジュールに対してレイアウトから抽出したネットリストを用いシミュレーションを行った。次にフルチップシミュレーションにおいて、故意的にエラーをいくつか書き込んだ ROM を用意しエラーのカウントの検証を行った。

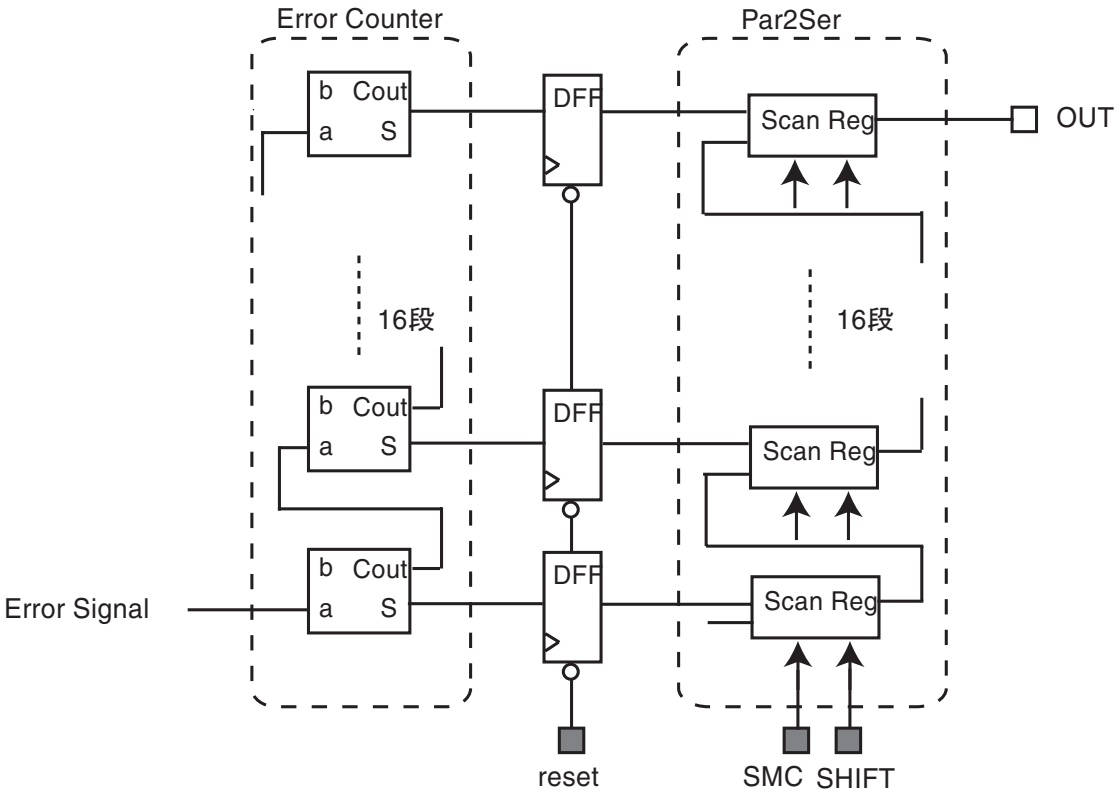
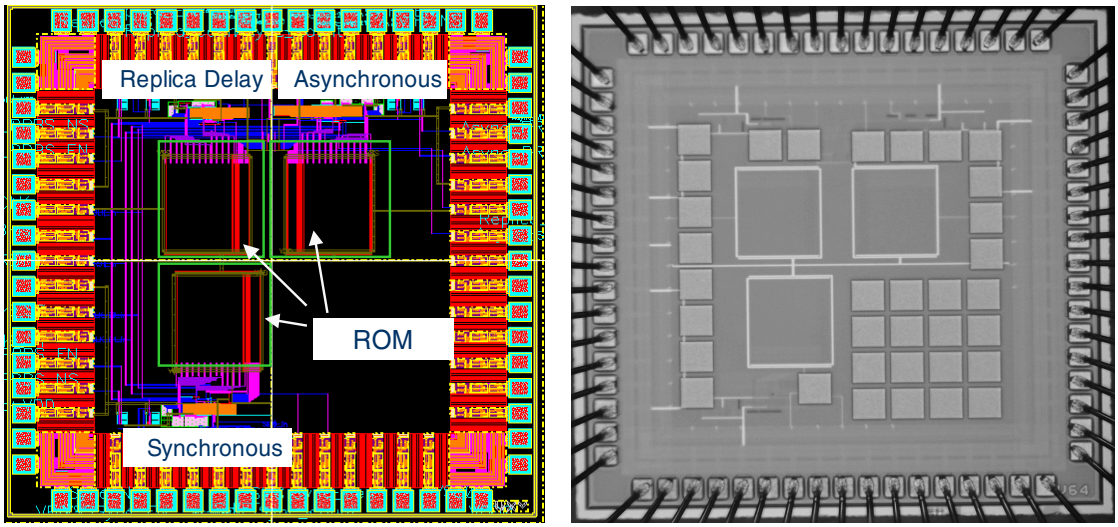


図 3.9: エラーカウンタとパラレルシリアル変換



Process	90nm standard CMOS
chip size	2.5 mm × 2.5 mm
Metal layers	6M, Cu
Nominal Vcc	1.0V

図 3.10: レイアウトと仕様

第4章　DCVSL回路における遅延ばらつきの解析

4.1　解析対象の回路

現在主に用いられているスタティックCMOSを用いた回路の場合、回路のクロック周波数は全てのパスの中で一番時間の長いクリティカルパスの遅延によって周波数が決まる。一方、DCVSLを用いた回路の場合は自己同期的に動作するので速度はそれぞれのパスが終了し次第終了するので、あらゆる命令が連続で動作するということを考えると平均的な速度で評価できる。そこで、どの程度DCVSLの自己同期方式が優れているかを示すために実際のCPUの遅延を測定した。本来はスタティックCMOSの同期方式とDCVSLの自己同期方式を比較するのが妥当であるが、DCVSLもスタティックCMOSも、ある機能をもつ回路を設計した際の構成は等しいということと、第二章で速度に関してはDCVSLはスタティックCMOSと比較しても優れていることが示したのでこのようにして比較した。

測定の対象とするCPUは当研究室で以前設計されたZ80の命令互換性をもつマイコンをDCVSLで設計したものである [14]。0.35umと90nmで設計されたものをそれぞれ測定した。

4.2　遅延の測定方法

CPUの遅延はCPUの実行開始から終了信号が外部に出力されるまでの時間をアドバンテストのT2000を用いて行なった [16]。

このT2000はデジタルテスタなので論理値の0または1を判定することしかできない。そしてその判定はデータレートごとに行なわれる、今回の測定ではデータレートを最小の2nsに設定した。対象とする遅延は20ns程度であるのに対し、この測定では解像度は2nsしかないためデータ依存などを測定するには不十分である。そのため、測定するタイミング(位相)をずらして測定するということを行い、複数の測定結果から遅延の値を得た。

具体的には図4.2のようにして行なう。終了信号の立ち上がりを2nsごとに判定しており、1回目の測定では28nsまでは0で、30ns以降は1という結果が得ら

図 4.1: アドバンテスト　T2000

れる。つまり遅延は 28ns〜30ns であるということが分かる。次に測定する位相を 0.5ns ずらして測定したところ 28.5ns まで 0 で、29.5 から 1 であるということが分かる。遅延は 28.5ns〜30.5ns ということになる。

　このように 0.5ns ずつ位相ずらすということを 4 回行なうと、遅延は以下のような式で表される。

$$28ns < T_{delay} < 30ns \quad \text{かつ}$$

$$28.5ns < T_{delay} < 30.5ns \quad \text{かつ}$$

$$29ns < T_{delay} < 31ns \quad \text{かつ}$$

$$27.5ns < T_{delay} < 29.5ns$$

　以上より

$$29ns < T_{delay} < 29.5ns$$

と解像度 0.5ns で遅延を求めることができる。

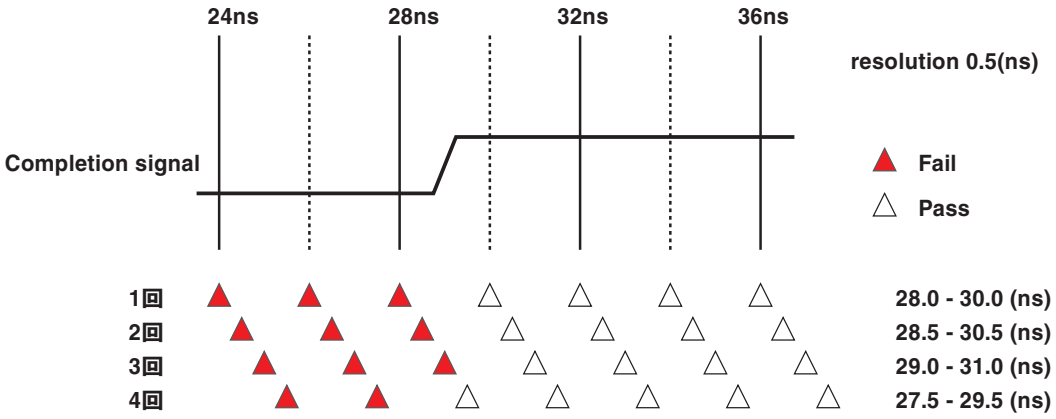


図 4.2: T2000 による遅延の測定方法

この測定では、終了信号に再現性があるために複数の測定結果から遅延の値を得ている。

実際の測定ではより解像度を上げるためにデータレートは 2ns のまま、位相を 0.25ns ずつづらし測定を行い 0.25ns の解像度を得た。

4.3　測定結果

CPU の遅延のばらつきには一般に PVT Variation である、Process Variation(プロセスばらつき)、Voltage Variation(電圧ばらつき)、Temperature Variation(温度ばらつき) の 3 つの要因が挙げられる。また CPU を考えた場合、ある命令を実行する際のデータに応じて通過するパスが異なるためデータにも依存する。そのため、まずデータ依存を測定した。そして電源電圧、温度、チップを変えたときの命令の実行時間 (遅延) について、データを変えてどの程度ばらつくかを測定した。ROHM 0.35um では内部に命令メモリもなく、プリチャージ信号も外部から直接入れているために測定した際のデータはエヴァリュエーション時間だけの遅延なので

$$T_{delay_{rohm}} = T_{evaluation}$$

となる。一方 ASPLA 90nm は内部に命令メモリがありアクセスタイムがかかる。またプリチャージ信号はクロックドライバによって自己同期的に作られる。そのため測定される時間は

$$T_{delay_{aspla}} = T_{evaluation} + T_{precharge} + T_{access}$$

となっている。両者を比較するために ASPLA 90nm ではプリチャージとアクセス時間を除いて、エヴァリュエーション時間だけで比較した。また評価する際に

は、遅延の絶対値のほかにデータを変化させたときの遅延のばらつきである標準偏差 (σ) を用いた。

データ依存

CPU で同じ命令を実行しても、その遅延は演算されるデータに依存する。そこでデータを変えてその時の遅延のばらつきを測定した。測定の対象とする命令は、ADD、LD、OR とした。

- LD n

  この命令は、A レジスタ (アキュムレータ) に即値で与えられる値を保存させる演算を行なう。n の値は 8bit であるので 256 通りのパターンについて遅延のばらつきを測定した。

- ADD A n, OR A n

  この命令は、それぞれ A レジスタの値と即値で与えられる n との ADD、OR を計算させる。レジスタ、n はそれぞれ 8bit 幅である。ROHM 0.35um では全ての組み合わせ ($2^8 \times 2^8$ 通り) の計算をさせ、そのときの遅延を求めた。ASPLA 90nm では実行の前に命令を予め RAM に入れるという仕様から全ての組み合わせの遅延を求めるのが困難であるので、速い、遅い、中くらいのパターンを選び ($2^8 \times 3$ 通り) 実行させた。実際に測定するときには、
  LD n;
  ADD A n;
  と LD n で予め A レジスタ (アキュムレータ) に入れておいて、A レジスタと n の値について計算させた。

データ依存の測定結果は ROHM 0.35um が図 4.3、ASPLA 90nm が図 4.4 のようになる。データを変えたときの遅延が X 軸で、Y 軸がその遅延になる頻度を表している。ADD 命令はデータの組み合わせによってキャリーの有無があり、演算の際に使われるパスが大きく異なるために遅延のばらつきは大きくなる。OR 命令は、各ビットの論理和をとるので ADD 命令に比べばらつきは小さい。LD 命令は A レジスタにデータ格納するためにゲートの段数も少なく、さらに遅延のばらつきはデータに依存せず ADD 命令や OR 命令より遅延のばらつきは小さくなっている。今後の測定結果において ASPLA 90nm の ADD 命令と OR 命令においては測定の都合により全てのデータの組み合わせを測定していないので全ての組み合わせを測定した場合に比べて小さい値で出ている可能性がある。またデータ依存は設計の良し悪しによりばらつきの値が変わり、ROHM 0.35um と ASPLA 90nm で

表 4.1: ROHM 0.35um の各電源電圧における遅延のデータ依存の標準偏差

	2.0 V	2.5 V	3.0 V	3.3 V	3.6 V
ADD	1.798 ns	0.991 ns	0.907 ns	0.832 ns	0.767 ns
OR	1.056 ns	0.266 ns	0.228 ns	0.177 ns	0.181 ns
LD	0.0635 ns	0.0311 ns	0.0155 ns	0.0269 ns	0.0155 ns

表 4.2: ASPLA 90nm の各電源電圧における遅延のデータ依存の標準偏差

	0.7 V	0.8 V	0.9 V	1.0 V	1.1 V
ADD	0.75 ns	0.398 ns	0.312 ns	0.257 ns	0.197 ns
OR	0.960 ns	0.122 ns	0.0948 ns	0.0896 ns	0.0873 ns
LD	0.0630 ns	0.0189 ns	0.0141 ns	0.0117 ns	0.0107 ns

は論理合成により合成された回路が違うのでここで単純に両者のデータ依存を比較することは難しい。

電源電圧依存

CPU を自己同期的に動作させると同期方式と比較して電圧を下げても同じ実行速度を得ることが出来るので ADD、LD、OR 命令の実行速度について電源電圧依存を測定した。測定結果は ROHM 0.35um が図 4.5、ASPLA 90nm が図 4.6 のようになる。電源電圧を X 軸にとり、Y 軸は遅延となっている。データを変えたときの遅延の最大と最小をエラーバーとした。ROHM 0.35um では動作電圧 3.3V が 10 %程度変動した際に遅延は 7 %程度変動する。一方 ASPLA の 90nm では動作電圧 1.0V が 10 %程度変動した際に遅延は 15〜20 %変動する。電源電圧の変動に対して ASPLA 90nm の方が大きい影響を受ける。また各電圧におけるデータを変えたときの遅延のばらつきは標準偏差として ROHM 0.35um と ASPLA 90nm でそれぞれ表 4.1、表 4.2 のように表される。

温度依存

LSI は動作させると電流が流れるために発熱する。そして発熱によりトランジスタの移動度は減少し性能は劣化する。そのため CPU を用いて温度を変化させたときの ADD、OR、LD 命令の遅延を測定した。温度は 0 ℃、25 ℃、80 ℃、120 ℃の４点で行なった。測定結果は ROHM 0.35um が図 4.7、ASPLA 90nm が図 4.8 のようになる X 軸に温度をとり、Y 軸は遅延である。データを変えたときの遅延の最

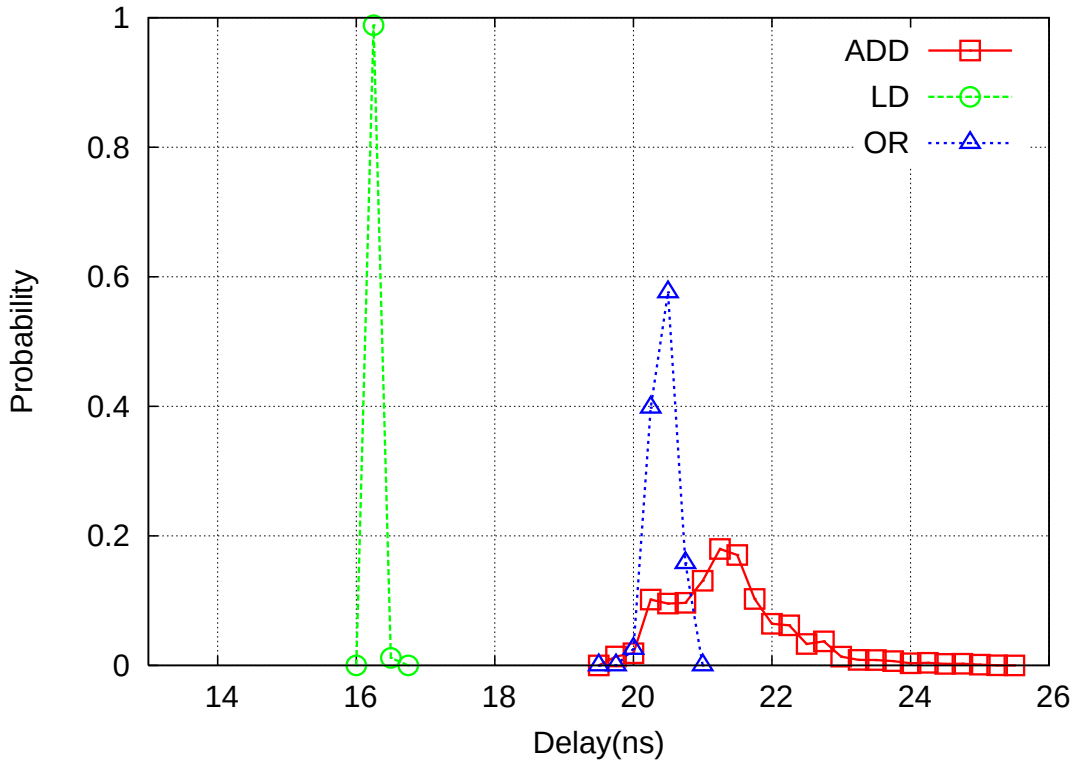


図 4.3: ROHM 0.35um のデータ依存

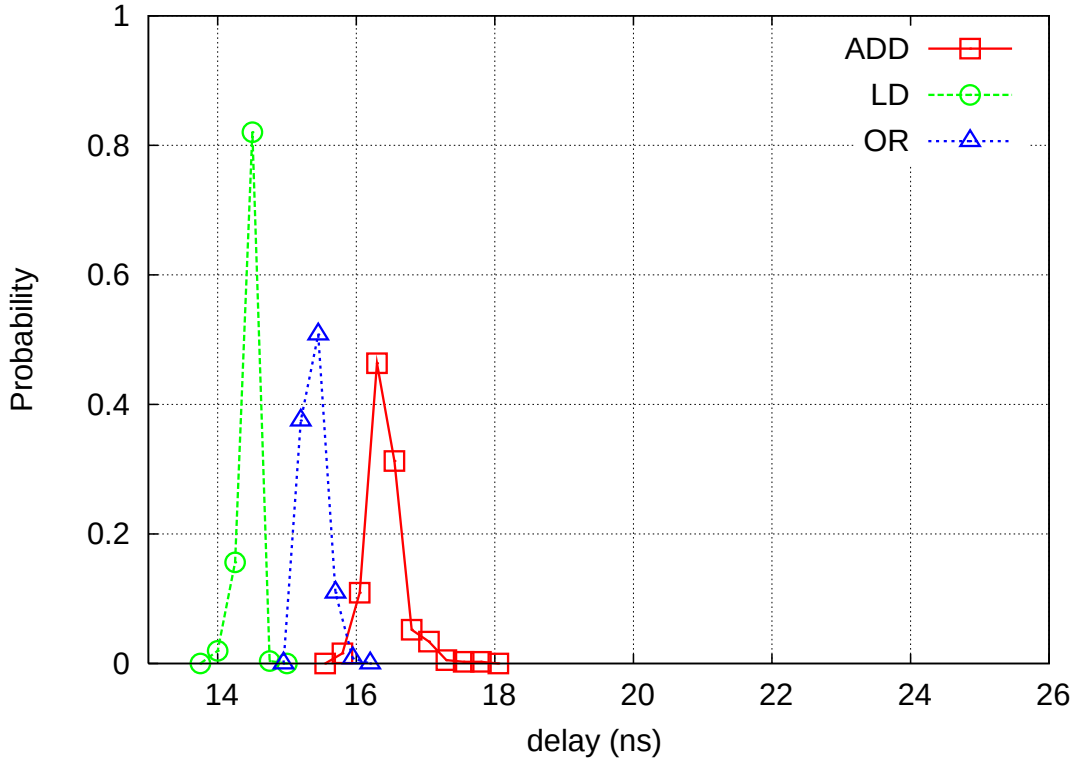


図 4.4: ASPLA 90nm のデータ依存

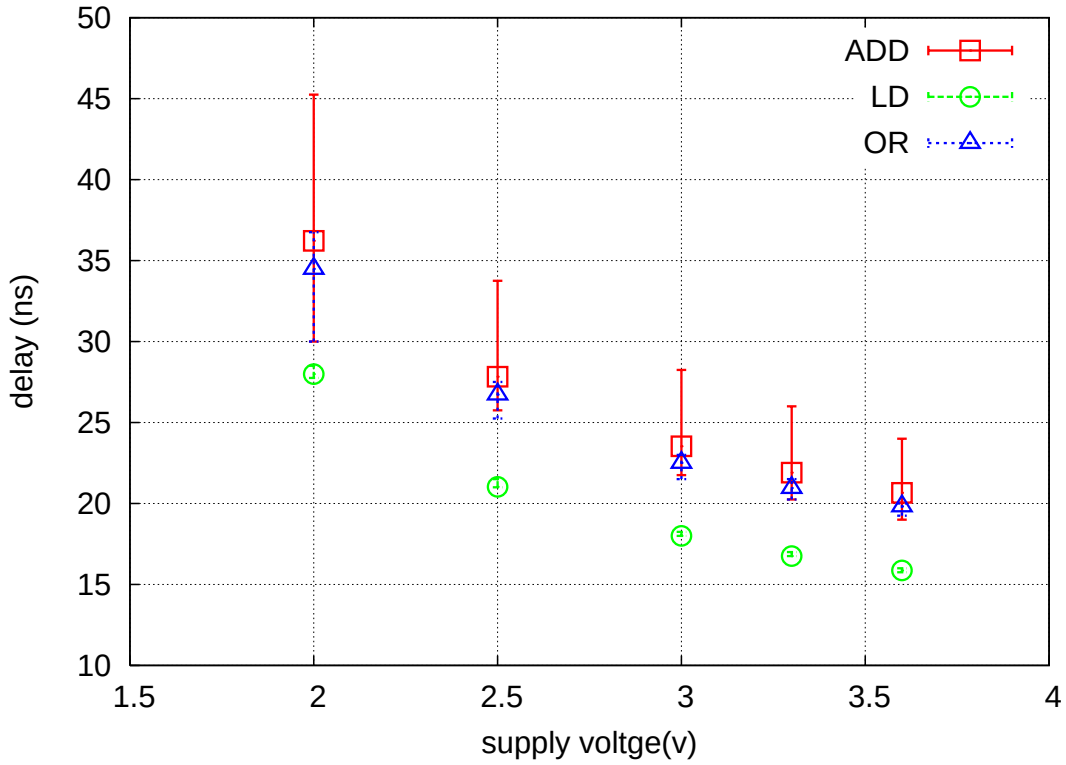


図 4.5: ROHM 0.35um の電源電圧 - 遅延

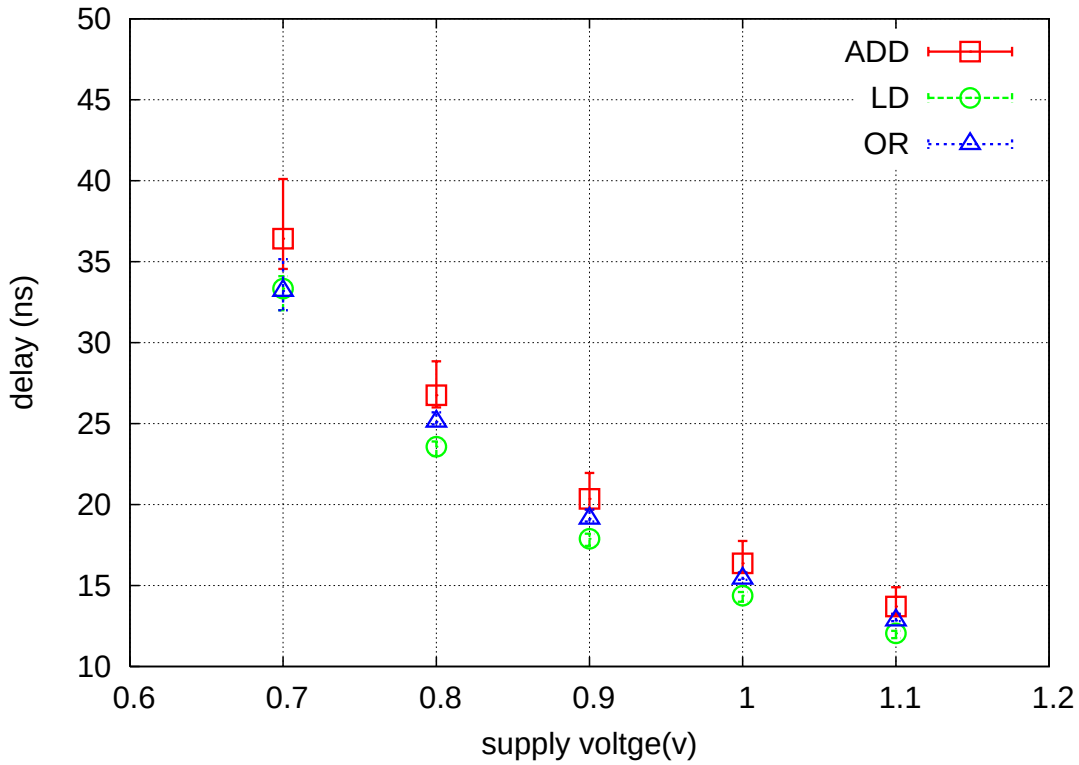


図 4.6: ASPLA 90nm の電源電圧 - 遅延

大と最小をエラーバーとした。

ROHM 0.35um では温度が 0 ℃〜80 ℃まで変動したときに室温 (25 ℃) の時に比べ遅延は 10 %程度変動する。一方 ASPLA 90nm では遅延は 3 %程度変動する。温度の変動に対して ROHM 0.35um の方が大きい影響を受ける。このことは以下のような温度を変化させたときの移動度と閾値の影響が考えられる [15]。

$$\mu = \mu_0 \left(\frac{T}{T_0}\right)^{km}, km = -1.2〜2.0$$

$$V_{th}(T) = V_{th}(T_0) + \alpha(T - T_0), \alpha = -0.5〜-3.0mV/K$$

これらの式より温度を上げたときに移動度、閾値はともに下がる。移動度が低下すると遅延は増加しようとし、閾値が低下すると遅延は減少しようとする。今温度が 100 ℃増加した時に約 100mV 閾値が下がる。このとき ROHM 0.35um において常温における閾値 0.6V 程度が 0.5V 程度まで下がる、また ASPLA 90nm においては常温における閾値 0.3V 程度が 0.2V 程度まで下がる。これより閾値の変化の割合が ROHM 0.35um より ASPLA 90nm の方が大きいために、遅延の増大の影響が ROHM 0.35um 程大きくなく測定の結果のようになったのではないかと考えられる。

また各温度におけるデータを変えたときの遅延のばらつきは標準偏差として ROHM 0.35um と ASPLA 90nm でそれぞれ表 4.3、表 4.4 のように表される。ROHM 0.35um は温度の上昇とともにデータ依存による遅延のばらつきも大きくなるが、ASPLA 90nm は温度の影響自体が小さいのでデータ依存による遅延のばらつき自体も小さくなっている。

表 4.3: ROHM 0.35um の各温度における遅延のデータ依存の標準偏差

	0　℃	25 ℃	80　℃	120　℃
ADD	0.793 ns	0.832 ns	0.926 ns	0.984 ns
OR	0.180 ns	0.177 ns	0.197 ns	0.204 ns
LD	0.0155 ns	0.0269 ns	0.0155 ns	0.1245 ns

表 4.4: ASPLA 90nm の各温度における遅延のデータ依存の標準偏差

	0　℃	25 ℃	80　℃	120　℃
ADD	0.254 ns	0.260 ns	0.261 ns	0.259 ns
OR	0.0846 ns	0.100 ns	0.0769 ns	0.0871 ns
LD	0.132 ns	0.129 ns	0.117 ns	0.122 ns

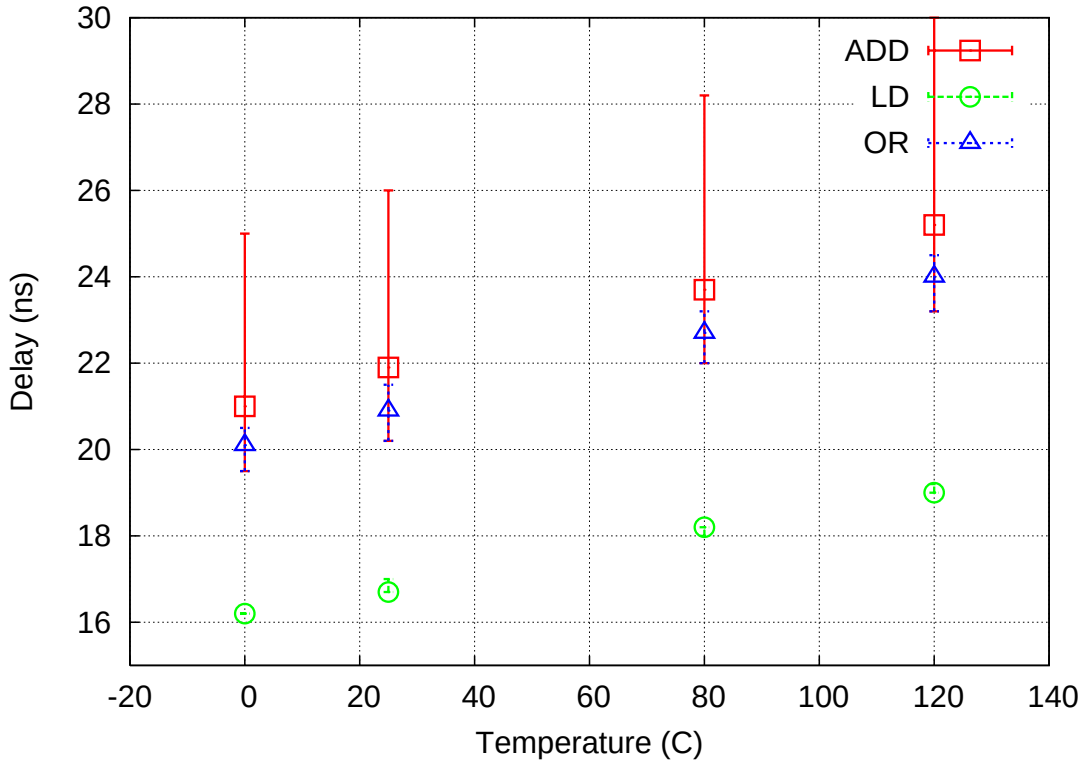


図 4.7: ROHM 0.35um の温度 - 遅延

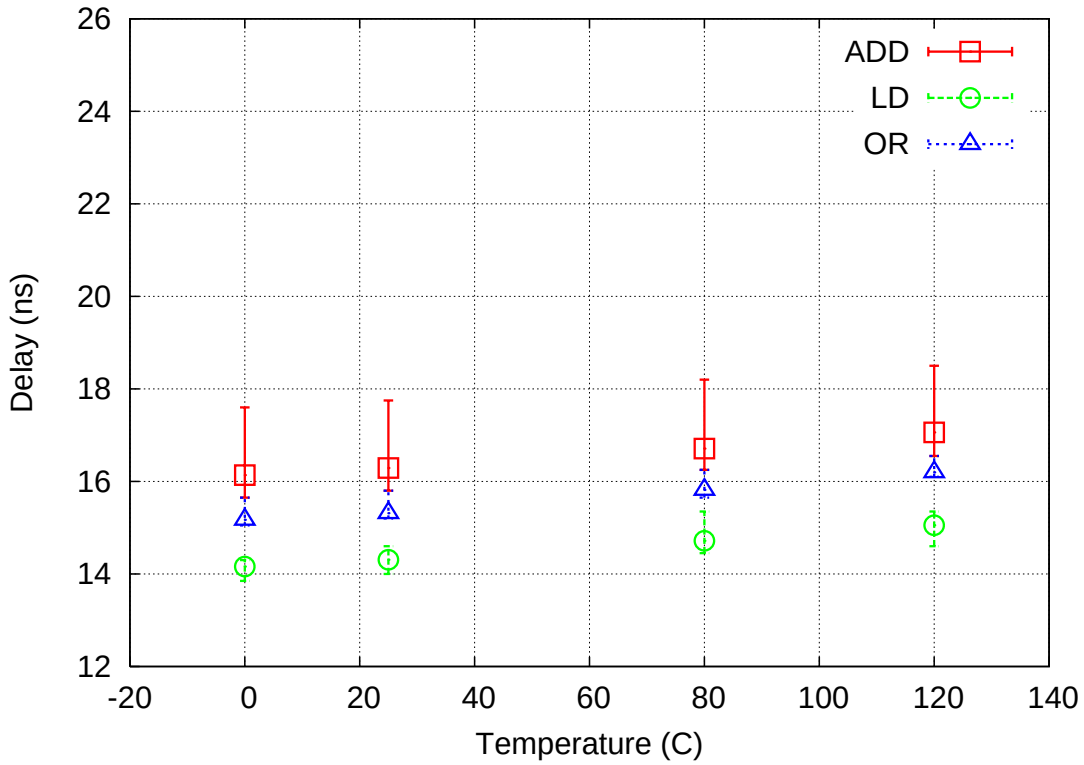


図 4.8: ASPLA 90nm の温度 - 遅延

チップ依存

同じ性能をもつ LSI を多く作ろうとしても、それぞれのチップについてゲート長やチャネルドープの量がばらつくためにトランジスタの性能がばらついてしまう。そのチップ間ばらつきの影響を調べるために ADD、OR、LD 命令の遅延を 3 つの異なるチップを用いて測定した。測定結果は ROHM 0.35um が図 4.9、ASPLA 90nm が図 4.10 のようになる。ROHM 0.35um でも ASPLA 90nm においても遅延が最大のものと最小のもので遅延の変動は 2 %程度であった。この状態ではチップ間のばらつきが小さいが、電源電圧を下げるなどしていくとチップ間の遅延にばらつきがあることが分かる。また各温度におけるデータを変えたときの遅延のばらつきは標準偏差として ROHM 0.35um と ASPLA 90nm でそれぞれ表 4.5、表 4.6 のように表される。もともとのチップ間のばらつきが小さいためにデータを変えたときのばらつきの差も小さくなっている。

表 4.5: ROHM 0.35um の各チップにおける遅延のデータ依存の標準偏差

	CHIP1	CHIP2	CHIP3
ADD	0.832 ns	0.827 ns	0.858 ns
OR	0.177 ns	0.0971 ns	0.188 ns
LD	0.0269 ns	0 ns	0.0155 ns

表 4.6: ASPLA 90nm の各チップにおける遅延のデータ依存の標準偏差

	CHIP1	CHIP2	CHIP3
ADD	0.252 ns	0.265 ns	0.256 ns
OR	0.0654 ns	0.0542 ns	0.113 ns
LD	0.119 ns	0.152 ns	0.119 ns

ばらつきの影響

前節で測定したデータを表 4.7、電圧、温度、チップ間のばらつきのうち ROHM 0.35um においてはデータと温度、ASPLA 90nm においてはデータと電圧の影響を大きく受けることが分かった。実際の同期回路ではこれら電圧、温度、チップ間ばらつき (σ_V、σ_T、σ_P) の影響を全てうける。それぞれの事象は独立であるので、全ての影響を考えたときのばらつき σ は以下の式で表される。

$$\sigma^2 = {\sigma_V}^2 + {\sigma_T}^2 + {\sigma_P}^2$$

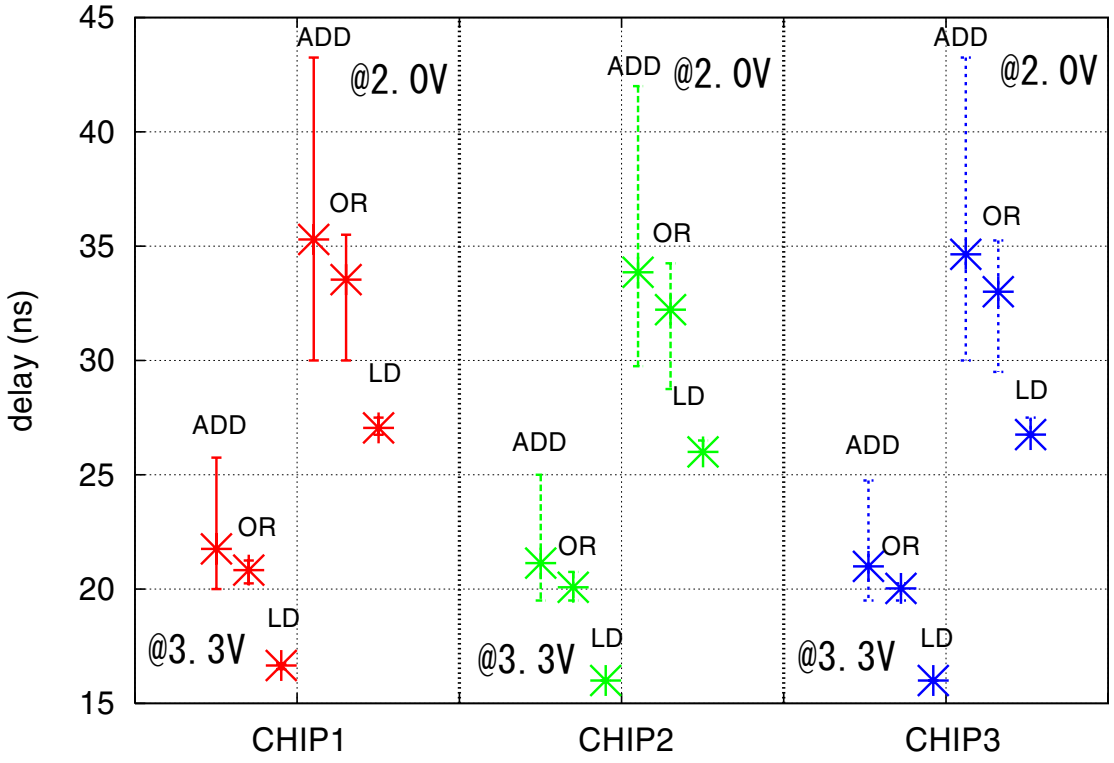


図 4.9: ROHM 0.35um のチップ間ばらつき

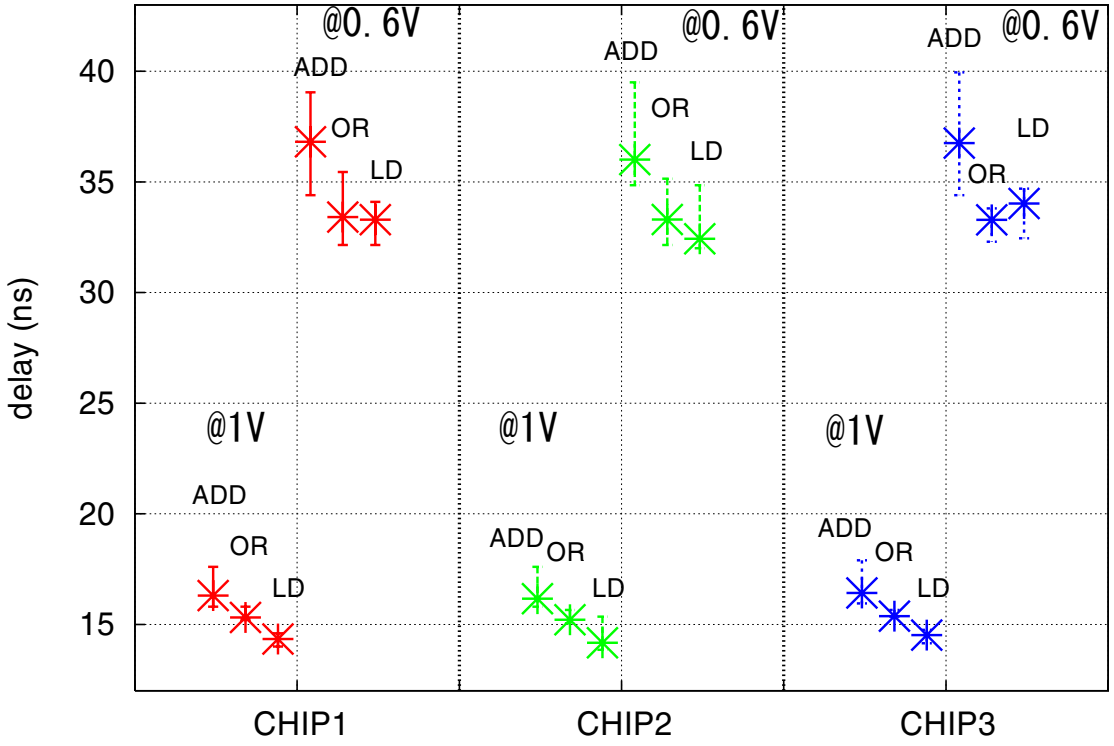


図 4.10: ASPLA 90nm のチップ間ばらつき

表 4.7: ばらつきの影響

	ROHM 0.35um	ASPLA 90nm
データ依存	18 %	10 %
電圧依存 (V)	7 %	20 %
温度依存 (T)	10 %	3 %
チップ依存 (P)	2 %	2 %
RMS(PVT)	12.3 %	20.3 %

この式から ROHM 0.35um と ASPLA 90nm のばらつきを計算すると、12.3 %、20.3 %となる (表 4.7 中の下段)。同期回路の場合は全ての命令やデータの組み合わせの中の最悪遅延からこれだけマージンをとっておく必要があり、自己同期回路であればマージンが不要であり高速に動作させることができる。

第5章　DCVSL回路のオンチップ電圧制御

この章で制御しようとしているCPUは先ほどの遅延ばらつきの測定に使用した90nmのプロセスで設計されたCPUであり、自己同期的に動作するので設計の際のマージンが同期方式に比べて小さくてすむことを第四章で述べた。その設計のマージンのため電源電圧を下げて制御することで同期回路と比較しても所望の速度を得るのに低い電圧で動作させることが可能である。そのために電源電圧を制御することで適応的に動作させることを目的として制御回路を設計した。制御にはバックコンバータを利用する方法 [17] などがあるが、インダクタを外部につける必要がある。そのためインダクタを用いずに内部にデカップリングキャパシタを載せそれに電荷を供給することで電圧を制御した。

5.1　制御方法

制御システムの全体図は図5.1のようになっている。オンチップにはCPU、予め命令やデータを入れておくメモリ(RAM)、自己同期的にクロックを作るための周辺回路、CPUに供給する電源電圧を制御するためのDC DC Converterが載っている。さらにオフチップにFPGAを用意する。DC DC Converterによって内部のCPUに供給する電源電圧が制御されるとCPUの動作速度が変化するために終了信号の出てくる速度が変化する。この終了信号を外部に出力し、遷移回数をFPGAによって数えることにより目的の動作速度と実際の動作速度を比較し、速度の上げ下げを決め、DC DC Converterの制御信号をCPUに入力するということをあるサイクルごとに行うことで、目標の速度を保ちつつ動作させるということを行なう。

DC DC Converterによる電源電圧は図5.2のような方法で制御される。まず、チップ内にキャパシタ(C_{chip})を載せておいて、そのキャパシタに電源から電荷を供給したり、放電したりすることで目的の電圧に制御する。供給はPMOS、放電はNMOSのスイッチをON、OFFすることによって行なう。そのON、OFFは図5.2中のトランジスタに入力されているようにPWM(Pulse Width Modulation)を用いる。PWMによる制御された電圧は、周期Tとスイッチが ON になっている

時間 τ のデューティー比に依存する。

電圧の制御性をあげるためには、CPU の動作周期よりも PWM の周期が短い必要がある。実際の CPU の周期は 15ns 程度なので PWM の周期を 2ns〜4ns 程度にし、そのときの波形は図 5.3 のような関係になる。Precharge 信号は 15ns 周期なので、CPU が 15ns 周期で動作し、その間に PWM は 7 周期前後入る。また、プリチャージの瞬間に電流が 0.4A 程度流れる。このときの制御性については後述する。

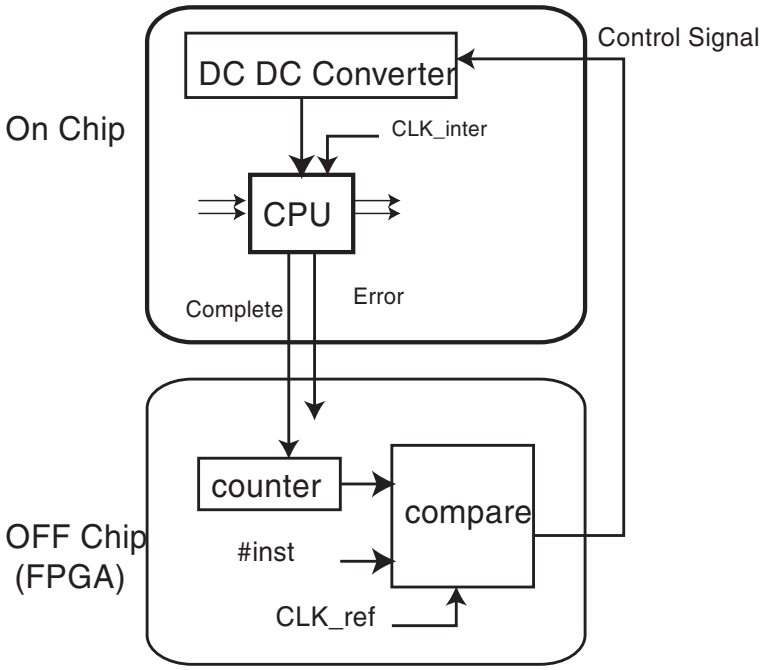


図 5.1: 制御システム全体図

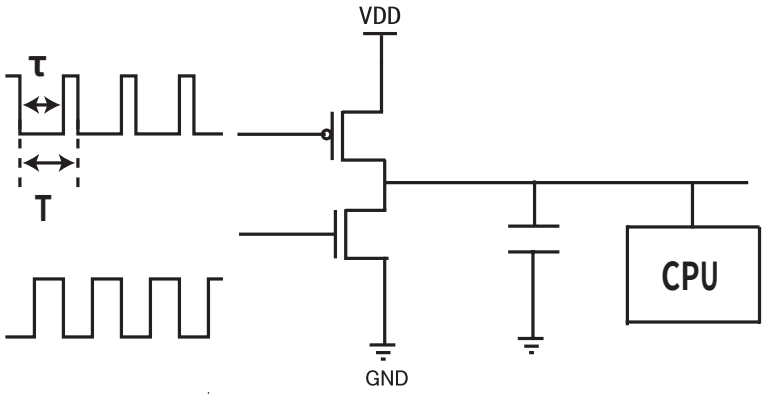


図 5.2: PWM による制御

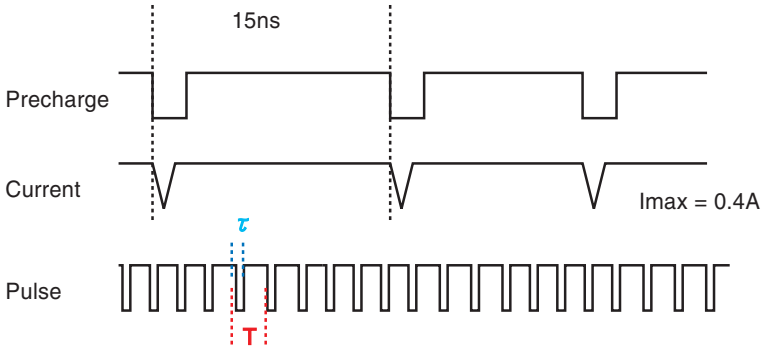


図 5.3: PWM と CPU のプリチャージ信号

DC DC Converter

電源電圧をコントロールするために DC DC Converter を設計した。設計した DC DC Converter は図 5.4 で示されるように Pulse Generator と VDD Controller から成り立っている。Pulse Generator では、電源電圧を制御するためのパルスが生成され VDD Controller に供給される。VDD Controller ではそのパルスを受け取り、そのパルスで PMOS または NMOS を ON、OFF することでチップ内のキャパシタに電源から電荷を供給したり、キャパシタから放電したりすることで CPU の動作する電圧をコントロールする。

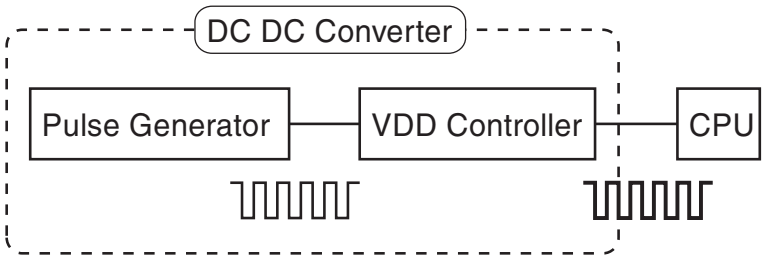


図 5.4: DC DC Converter

- Pulse Generator

  Pulse Generator の回路図は図 5.5 のようになっている。まず、VCO によって 2ns または 4ns のクロックが作成される。そのクロックは 16 段のディレイエレメントによって 0.125ns ずつずらされる、そしてディレイエレメントの出力段からはタップが出されており、そのいずれかをセレクタで選択できるようになっている。そして (0.125 × k)ns ずらされたクロックと元のクロッ

クの否定の AND をとることで、Duty 比が 0〜50 %のクロックを作成することができる (図 5.5 の左側)。また (0.125 × k)ns ずらされたクロックと元のクロックの NOR をとることで、Duty 比が 50〜100 %のクロックを作成することができる (図 5.5 の右側)

この 2 つのクロックを選択することで、最終的に VCO で 4ns のクロックを使用した場合、0.125ns 間隔で 32 段階の PWM を、VCO で 2ns のクロックを使用した場合 0.125ns 間隔で 16 段階の PWM を作成し、VDD Controller に出力することができる。

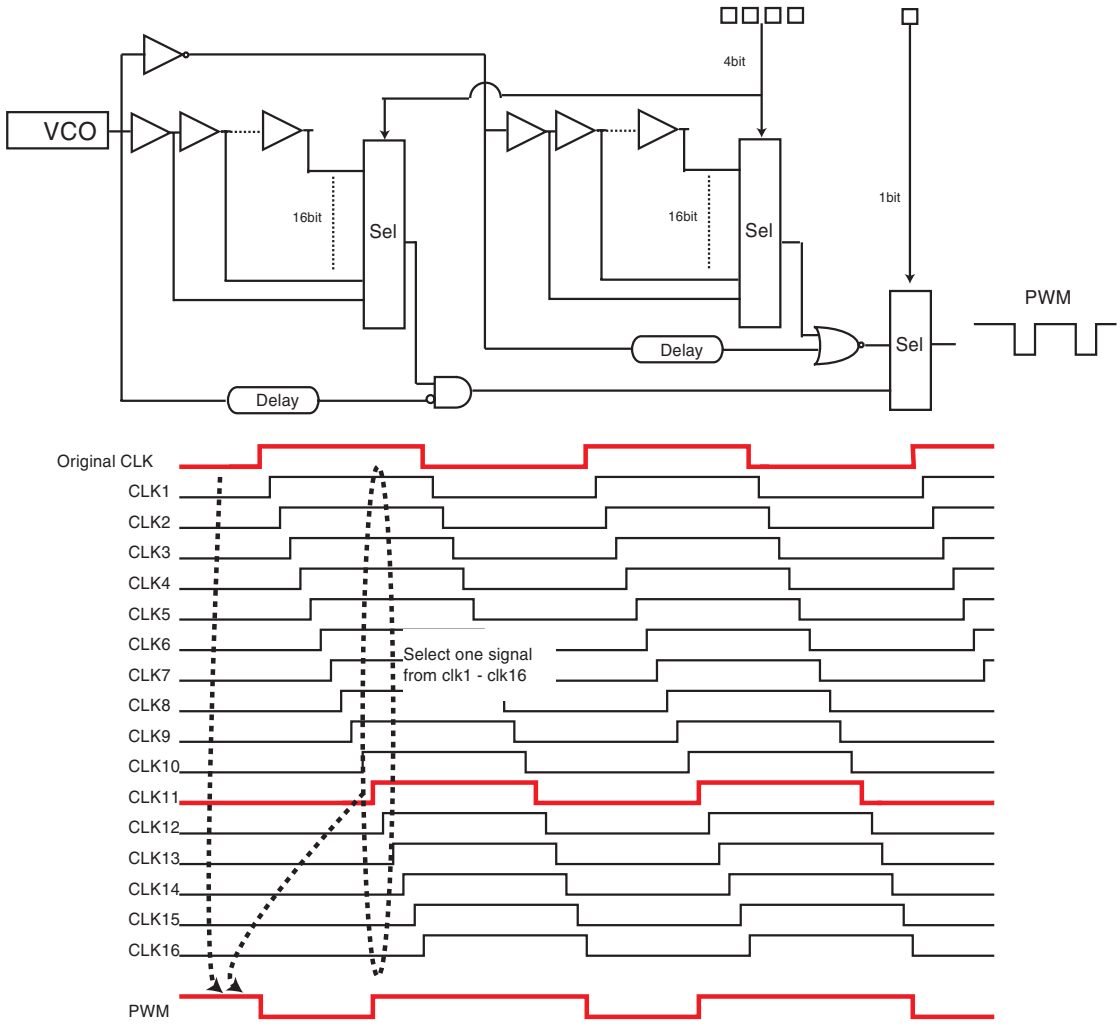


図 5.5: Pulse Generator

- VDD Controller

  VDD Controller では Pulse Generator で作成された PWM によって PMOS、

NMOS のスイッチを ON、OFF することでチップ内のキャパシタに電荷を供給する機能を持っている。その回路図は図 5.6 のようになっている。パルスを入力するトランジスタのサイズが大きくなれば流せる電流が大きいので電荷を早く供給または放電ができるためにいくつかサイズの異なるトランジスタから選択できるようにしている。その大きさ W は、100um、200um、400um の 3 種類を載せておりそれぞれは個別に選択可能である。

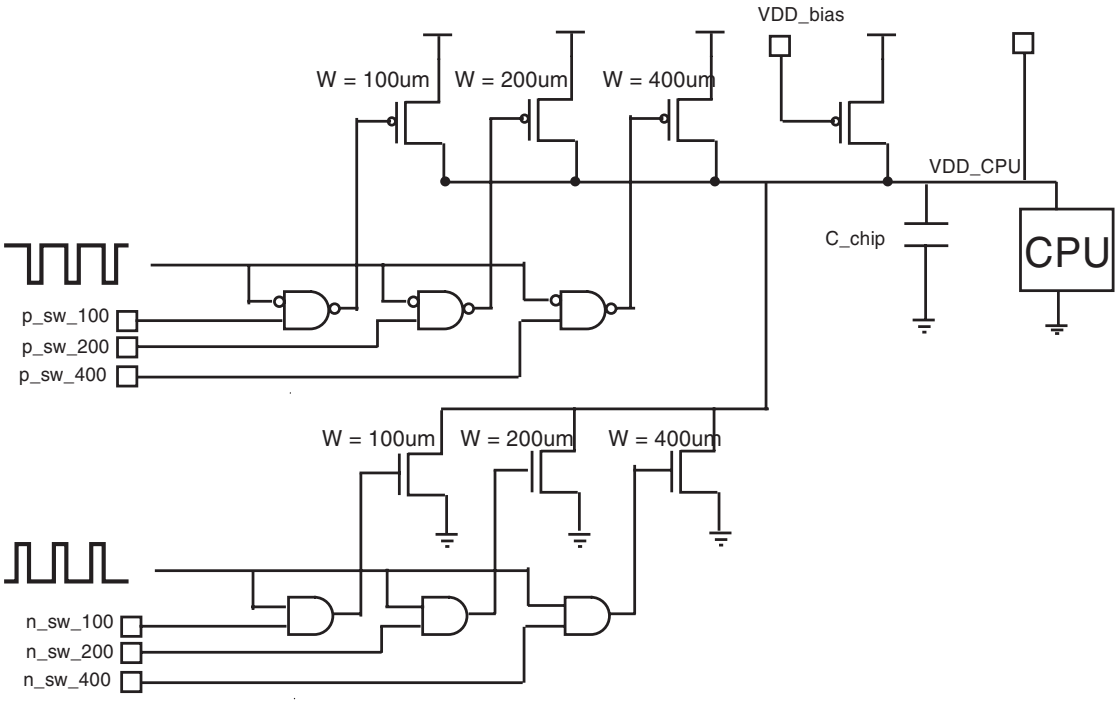


図 5.6: VDD Controller

レイアウトとチップ写真

以上で説明したモジュールを ASPLA90nm テクノロジを用いて実装した。設計したチップのレイアウトとチップ写真、またその仕様は図 5.7 のようになる。チップ上には測定対象の CPU と命令メモリ、データメモリ、周辺回路、DC DC Converter、デカップリングキャパシタを載せている。CPU のコアの面積は 600um × 600um となった。設計した DC DC Converter のレイアウトは図 5.8 のようになり、その大きさは 60um × 250um となった。

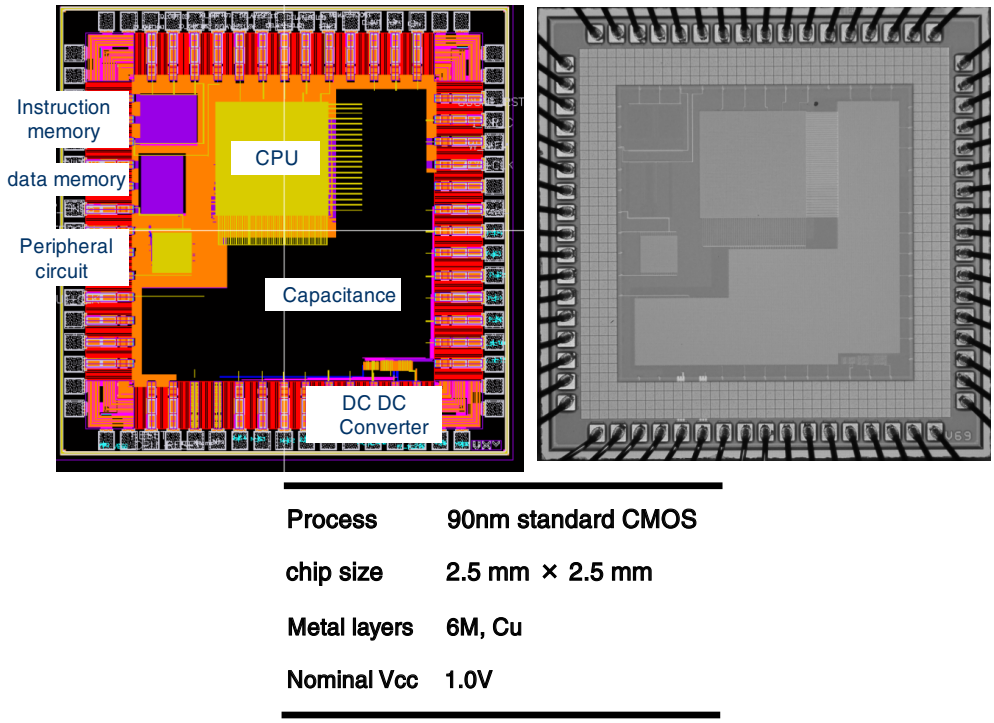


図 5.7: チップのレイアウトと写真

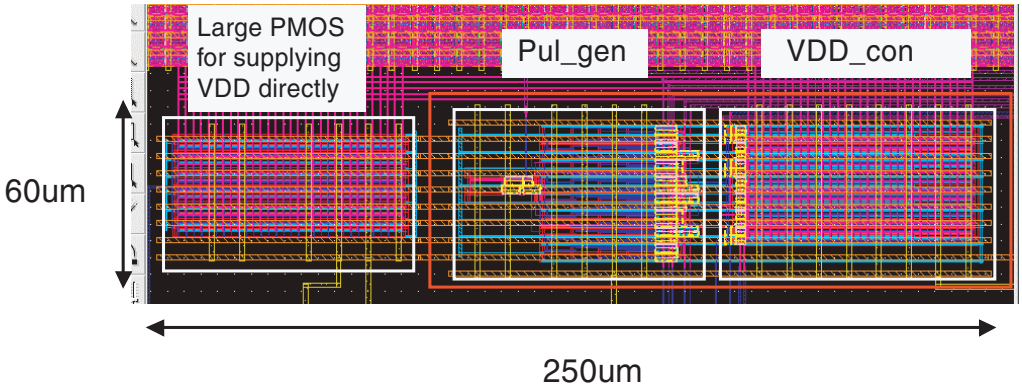


図 5.8: DC DC Converter のレイアウト

5.2　電源電圧制御回路のモデル化

CPU が動作しているときには、PWM に応じてチップ内に載せたキャパシタ (C_{chip}) に電荷が常に供給される状態になっている。プリチャージフェイズには、C_{chip} に蓄えられた電荷が CPU 内の各々の DCVSL のプリチャージノード (C_{cpu}) にためられる。そしてエバリュエーションフェイズにはプリチャージノードに蓄えられた電荷が全て放電される。

この状況をスイッチとキャパシタと抵抗でモデル化したものが図 5.9 である。制御の際の NMOS による放電は頻繁には使用されないと思われるので図には載せていない。まず、PWM を受けて sw0 を ON、OFF させて C_{chip} に電荷を供給する、これは CPU の動作とは独立に行なわれる。CPU のプリチャージフェイズにおいて sw1 が ON され C_{cpu} に C_{chip} から電荷が供給される、そしてエバリュエーションフェイズには C_{chip} にためられた電荷が放電する。

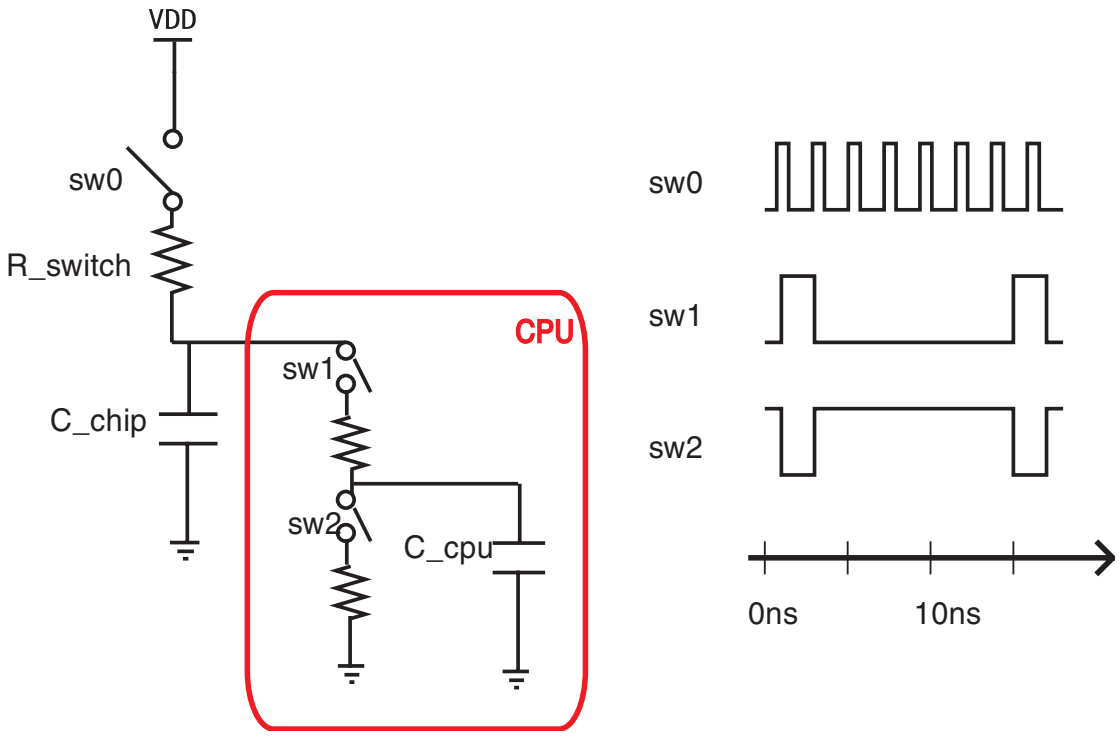


図 5.9: 制御のモデル

CPU の電圧 (V_{cpu})

C_{cpu} に電源から電荷が供給されたときには、スイッチとなる PMOS の抵抗を R_{switch}、C_{chip} にためられた電荷をとしたとき以下のような方程式で表すことがで

きる。

$$Vdd = IR_{switch} + \frac{Q}{C_{chip}}$$

これを時間 t で微分し、微分方程式を解くと CPU の電圧 V_{CPU} は時間の関数となり以下のようになる。

$$V_{CPU}(t) = Vdd\left\{1 - exp\left(-\frac{t}{R_{switch}C_{chip}}\right)\right\}$$

ある時間 t において V_{CPU} が以下の電圧であったとする。

$$V_{CPU}(t) = Vdd\left\{1 - exp\left(-\frac{t}{R_{switch}C_{chip}}\right)\right\} = Vdd_t \tag{5.1}$$

これに対して、充電を Δt だけ行なったときの CPU の電圧は、

$$V_{CPU}(t + \Delta t) = Vdd\left\{1 - exp\left(-\frac{t + \Delta t}{R_{switch}C_{chip}}\right)\right\} \tag{5.2}$$

となる。これに対して放電を行なったときの電圧は、C_{chip} と C_{cpu} の間で電荷分配が行なわれるので、

$$\frac{C_{chip}}{C_{chip} + C_{cpu}}V_{CPU}(t + \Delta t) = \frac{C_{chip}}{C_{chip} + C_{cpu}}Vdd\left\{1 - exp\left(-\frac{t + \Delta t}{R_{switch}C_{chip}}\right)\right\}$$

$$= \frac{C_{chip}}{C_{chip} + C_{cpu}}Vdd\left\{1 - \left(1 - \frac{Vdd_{t_n}}{Vdd}\right)exp\left(-\frac{\Delta t}{R_{switch}C_{chip}}\right)\right\} \tag{5.3}$$

CPU が一度動作する際には、初期状態 (5.1)、充電された状態 (5.2)、放電された状態 (5.3) を繰り返しながら動作する。

これを図 5.10 を用いて説明する。左図の X 軸は CPU が実際に動作する時の時間軸で右図の X 軸は CPU にどの程度充電されるか説明するための図である。初期状態において、V_{cpu} が Vdd_{t1} であったとする。右図においてこの電圧に対応する時間を t1 とする。充電状態においては、PMOS のスイッチが ON、OFF しているが ON のときのみ充電が行なわれるので、左図のように V_{cpu} は階段状に上昇していく。この充電状態における PMOS が ON になっている時間の総和が式 (5.2) における Δt であり、充電が終了したときの V_{cpu} が Vdd_{t2} となる。右図においてこの電圧に対応する時間を t2 とする、ここで $t2 = t1 + \Delta t$ となっている。次に放電状態では、V_{chip} にためられた電荷が V_{cpu} に分配されるので、電圧が Vdd_{t2} から Vdd_{t3} に減少する。この電圧に対応する時間を t3 とする。このように、初期状態→充電→放電→…、と繰り返すことは、左図で初期状態→ステップ状に電圧が上昇していく→電荷分配で電圧が減少する→…、となることであり、右図では t1 → t2 → t3 →…、と繰り返すことを意味する。CPU で連続実行される場合には、放電が終了した t3 における状態が初期状態となり、充電放電と繰り返す。

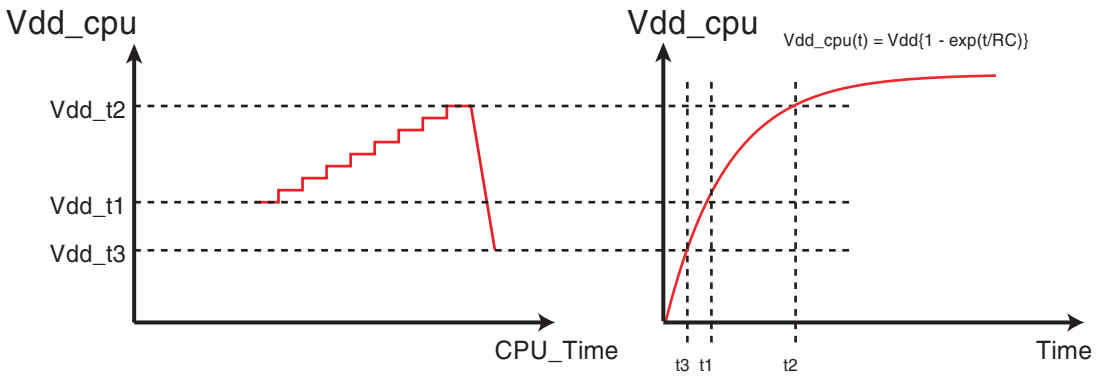


図 5.10: Vdd cpu

V_{cpu} の制御性

V_{cpu} を制御性を議論するために以下の 3 点を考える。

- 制御され安定する V_{cpu} の値

  PWM の周期 T と PMOS が ON している時間 τ が決まると電源から C_{chip} に電荷が充電され V_{cpu} は上昇するが、CPU が動作するたびに C_{chip} に蓄えられた電荷が全く電荷のたまっていない C_{cpu} に分配されるために V_{cpu} は減少する。充放電を繰り返し、この両者がつりあった時に V_{cpu} の値は安定する。これは先ほどの漸化式で表すと

$$V_{CPU}\left(t_{n+1}\right) = \frac{C_{chip}}{C_{chip}+C_{cpu}} Vdd \left\{ 1 - \left(1 - \frac{V_{cpu}(t_n)}{Vdd} \right) exp\left(-\frac{\Delta t}{R_{switch} C_{chip}} \right) \right\} \tag{5.4}$$

  となる。この式 (5.4) で初期状態の電圧を適当に入れて n を無限大に持っていくと安定する電圧を求められる。このとき、図 5.10 の左図で初期状態の電圧 (Vdd_{t1}) と電圧が降下した時の値の値 (Vdd_{t3}) が等しい状態となる。

- V_{cpu} のリプル

  V_{cpu} が安定した状態においても充放電を繰り返すために電源電圧は変動する、その変動の値を Vdd_{ripple} とするとその値は、図 5.10 で Vdd_{t1} と Vdd_{t2} の差となる。

- 制御され安定するまでの応答時間

  V_{cpu} を安定させようとすると充放電を繰り返して徐々に目的の値に漸近していく。この安定するまでの時間は以下のようにして求められる。式 (5.4) にお

いて初期値を $V_{cpu}(t_0)$ とし、この漸化式に代入して行き $V_{cpu}(t_1)$、$V_{cpu}(t_2)\cdots$ を求め目的の値になるときの n の値を算出する。この n の値が CPU のサイクルを表すので $T_{cycle}\times n$ の値によって制御が安定するまでの時間を求めることが出来る。

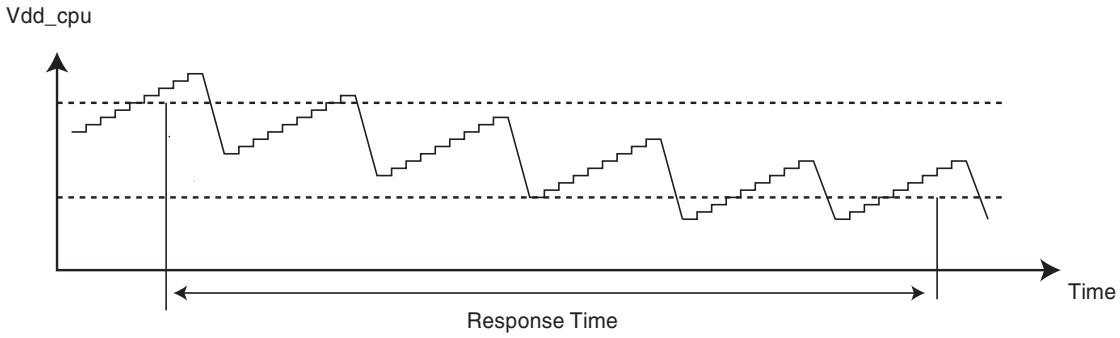


図 5.11: Vdd_{cpu} の安定状態への移行

5.3　シミュレーションと測定結果

制御信号と CPU の電圧 (Vdd_{cpu})

上で述べた V_{cpu} の制御され安定する電圧のシミュレーション、モデルによる計算結果、測定結果を図 5.12 に載せる。X 軸は制御信号、Y 軸は制御された電圧である。シミュレーションとモデルによる計算から得られた値は非常に似ているが、測定結果からはずれている。この原因として以下の三点が考えられる。

- VCO の発振周波数

  VCO の発振周波数は 500MHz になるように設計し、シミュレーションではその値が得られたが、プロセスばらつきや電源電圧のぶれは発振周波数のずれる要因となるので、これらによって期待した値と異なる値で発振していることが考えられる。発振周波数が大きい場合には制御される電源電圧は同じ制御信号でも VDD_{chip} に注入される電荷の量が多くなるので電圧は高くなる。さらにこのときコントロール信号が大きくなるにつれて制御される電圧も飽和すると考えられるのでグラフの傾きは小さくなる。逆に小さい場合には電圧が低い方にずれ傾きは小さくなると考えられる。

- CPU の動作周波数

  CPU の動作周波数が早くなれば放電される電荷の量が大きくなると考えられるので安定する電圧は下がる。遅くなった場合は逆となる。いまシミュレー

ションでは CPU の動作速度は 15ns 程度であり、計算においてもこの結果を用いて計算した。第五章の遅延の測定結果よりこの値は妥当であるが実際はデータ依存により遅延にばらつきが生じるためこの 15ns という値からはずれることが考えられ、これにより測定結果がずれたものと考えられる。

- Pulse Generator の構成

  図 5.5 の Pulse Generator の構成で、PWM を作るときにデューティー比が 50 %の前後で左側の遅延モジュールを使うか右側を使うかが異なる。そのため制御信号 16 と 17 の間の連続性が小さくなってしまうことが考えられる。

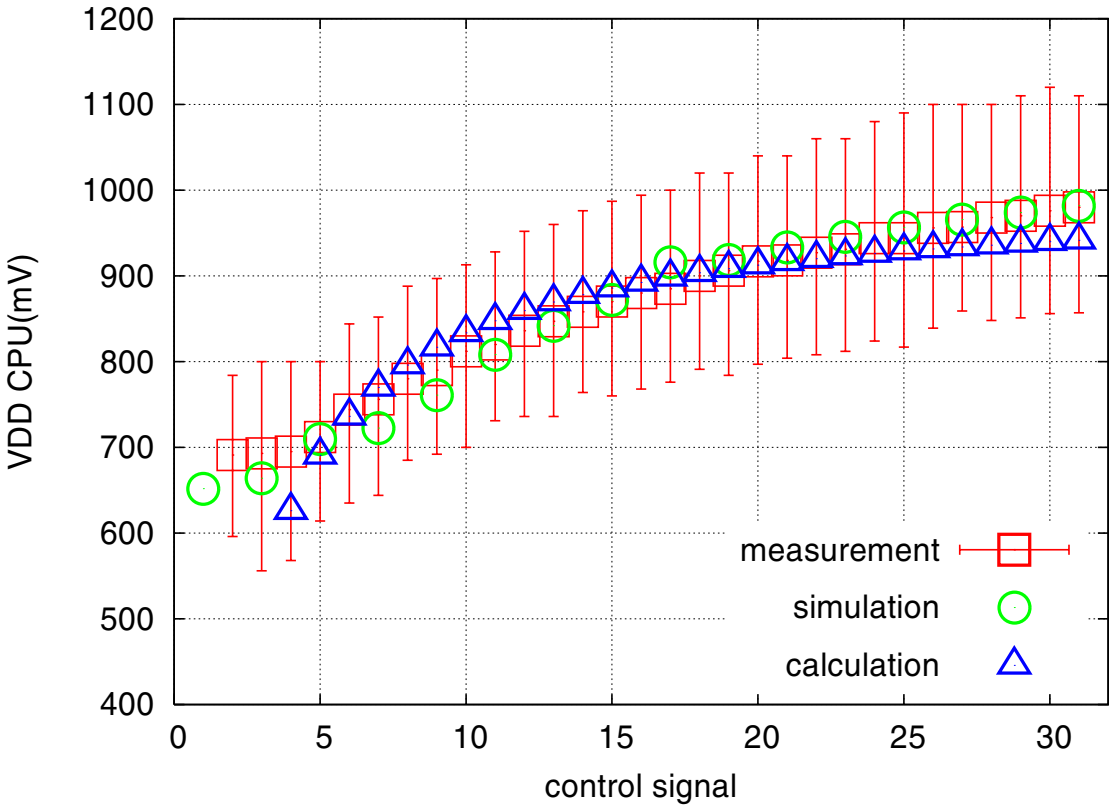


図 5.12: 制御された Vdd_{cpu} の値

CPU の実行速度

Vdd_{cpu} を変えたときの遅延の測定結果は図 5.13 のようになる。遅延はある一定時間に出力された終了信号の回数から求めた。JP 命令で連続実行させており、その遅延は LD と比較しても小さい [12] ために遅延の値が小さくなっている。

CPU の消費エネルギー

Vdd_{cpu} を変えたときの消費エネルギーの測定結果は図 5.14 のようになる。いま回路を 0.95V で動作させると、遅延は 16.5ns 程度であるので同期回路で動作させようとすると、マージンを 20 %程度とり 20ns 程度で動作させることになる。一方自己同期回路で動作させると平均遅延で評価できマージンが不要なので 20ns の遅延を 0.82V 程度で実現できる。このときの両者の消費エネルギーを比べると、同期回路が 0.95V なので 0.44nJ/transition で動作し、自己同期回路は 0.82V なので 0.34nJ/transition で動作する。つまり 20 %程度消費エネルギーを削減できる可能性があることが示された。

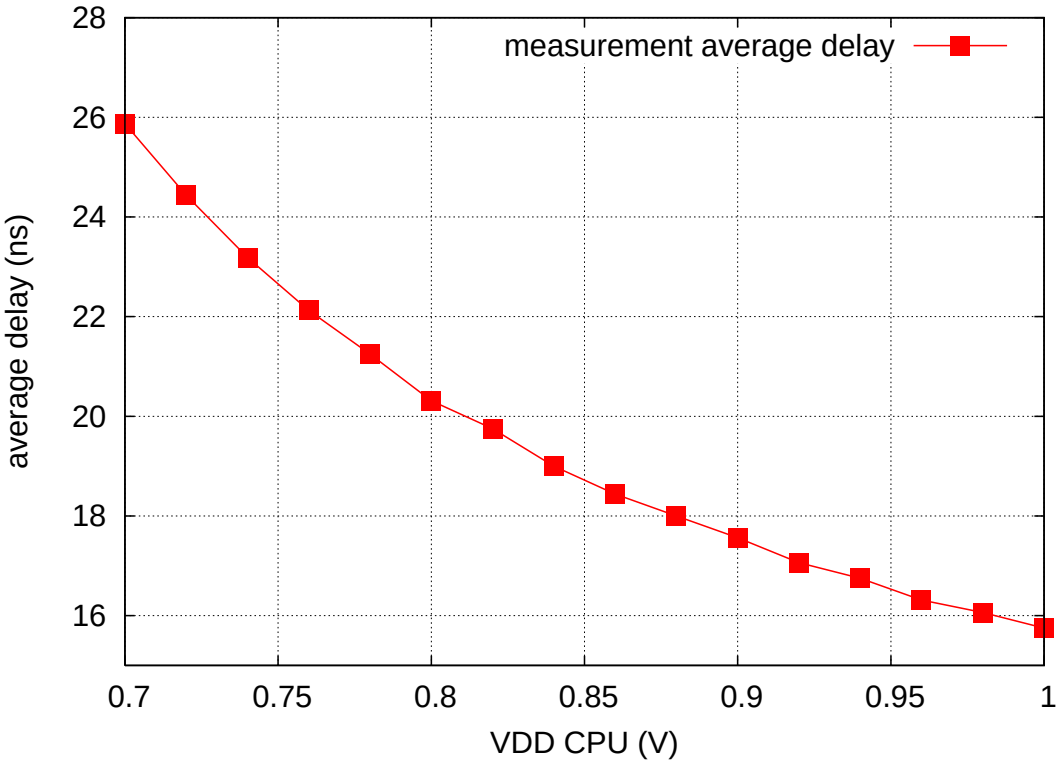


図 5.13: Vdd_{cpu} - 遅延

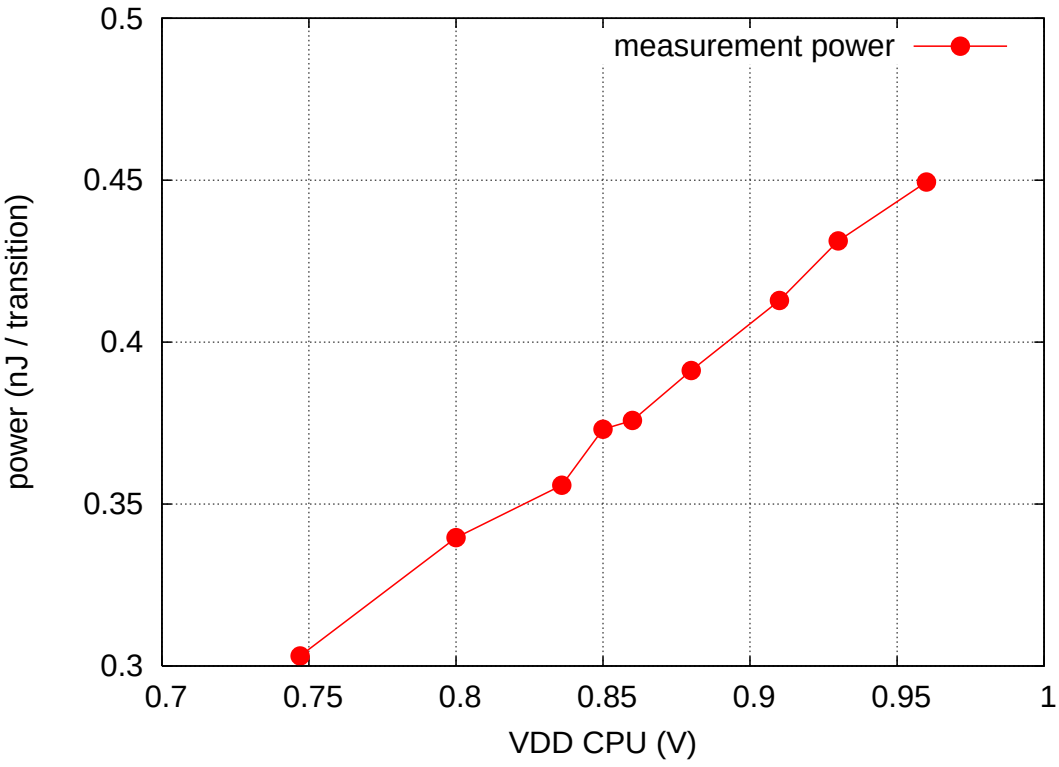


図 5.14: Vdd_{cpu} - 消費エネルギー

第6章　結論

LSI の微細化によって、電源電圧が下がるとともに閾値が下がり、α 線や電源電圧のノイズに対するノイズーマージンの低下が引き起こされ、今後論理エラーが問題になることを示した。また微細化によってプロセスのばらつきが LSI の性能に与える影響が大きくなり、遅延の見積もりが難しくなることで遅延エラーが起きることを示した。これらのエラーへの対処として、論理エラーが起きても二線を用いエラー状態を与えることで論理エラーを検出するすることができ、さらに終了検出を行い自己同期的に用いることで遅延エラーが起きない二線式ドミノ回路の DCVSL を用いることで解決できることを示した。

シミュレーションにより、DCVSL と Footless DCVSL のノイズ耐性が 0.4V、0.38V であることを示した。スタティック CMOS と比較してそれぞれ、18 %、29 %速く動作することを示した。さらに電源電圧にノイズがのった時の遅延のばらつきがスタティック CMOS が 10.6 %程度であるのに対し、DCVSL と Footless DCVSL のばらつきが 5.2 %、5.5 %と小さいことを示した。また基板バイアスが変化したときにおいてもスタティック CMOS、DCVSL も遅延はに 4 %程度がばらつくこと示した。以上より、DCVSL が現在主に用いられているスタティック CMOS と比較して高速に動作し、電源のノイズに対しての遅延ばらつきが小さいことを示した。

シミュレーションではプロセスのばらつきや電源電圧のノイズなどが存在する環境下での DCVSL とスタティック CMOS のエラーレートを求めることは困難である。そのため実際に電源電圧にノイズがのったり、電源電圧そのものが変化したときどの程度エラーが起きるかを測定するためのエラーレートの測定が出来る回路を提案した。

同期回路の動作周波数はワーストケースを考慮して決められなければならないので、早く終わる命令があったとしても次のサイクルまで待つという無駄が生じる。それに対して DCVSL を自己同期的に用いる場合には、命令が終了ししだい次の命令を実行できるので動作周波数は平均的な遅延から見積もることが出来る。実際に Z80 と命令互換をもつ DCVSL で構成された CPU(ROHM 0.35um と ASPLA 90nm) の ADD、OR、LD 命令の遅延のばらつきをデータを変えて測定した。さらに電源電圧、温度、チップを変えたときの遅延の変化を測定し、同期回路で動作させるときのマージンが ROHM 0.35um の場合 12 %、ASPLA 90nm の場合 20 %マージンが必要であることを示した。

前章で DCVSL 回路を自己同期的に使用した場合に遅延を平均速度で評価できることを示したので、電源電圧を下げ実行速度を落とすことで同期回路と同じ性能を出すのに低消費電力で行なえることを示した。そこで CPU の終了信号を外部でカウントし FPGA でオンチップの DC DC Converter を制御することで電源電圧を制御し、目的の動作速度で動作させる方法を提案した。次に設計した DC DC Converter の回路構成について述べ、その制御電圧を見積もるためにモデル化を行ない、シミュレーションの結果とモデルから計算される制御電圧の値が近いことを示した。実測の遅延と消費エネルギーのデータから自己同期回路が同期回路に比べ約 20 %消費エネルギーを減らせる可能性があることを示した。

参考文献

[1] Sterpone, L. et al., "Logic soft errors in sub-65nm technologies design and CAD challenges," in Proc. *IEEE Int. Design Automation Conference*, pp. 2-4, June 2005.

[2] Karnik, T. et al., "A new analytical approach to estimate the effects of SEUs in TMR architectures implemented through SRAM-based FPGAs," *IEEE Transactions on Nuclear Science*, vol. 52, pp 2217 - 2223, Dec. 2005.

[3] Karaki, N. et al., "A flexible 8b asynchronous microprocessor based on low-temperature poly-silicon TFT technology," *IEEE International Solid-State Circuits Conference (ISSCC)* Dig. of Tech. Papers, pp. 272-598, Feb. 2005.

[4] T. Kamik, P. Hazucha, and J. Patel, "Characterization of soft errors caused by single event upsets in CMOS processes," *IEEE Trans. Dependable and Secure Compuling*, Vol. 1, Issue 2, pp. 128-143, April-June 2004.

[5] L. Heller et al., "Cascade Voltage Switch Logic : A Differential CMOS Logic Family," in Proc. of *IEEE International Solid State Conference*, pp. 16-17, Feb. 1984.

[6] K. H. Dia et al.,"Footless Dual-Rail Domino Circuit with Self-Timed Precharge Scheme," in Proc. of *IEEE Asian Solid-State Circuits Conference (A-SSCC)*, pp. 309-312, Nov. 2005.

[7] James T. Kao et al., "Dual-Threshold Voltage Techniques for Low-Power Digital Circuits," *IEEE J. Solid-State Circuits*, vol. 35, no. 7, pp. 1109-1018, July 2000.

[8] Tschanz, J.W. et al., "Adaptive body bias for reducing impacts of die-to-die and within-die parameter variations on microprocessor frequency and leakage," *IEEE J. Solid-State Circuits,* vol. 37, no. 11, pp. 1396 - 1402, Nov. 2002.

[9] Mauro Olivieri et al., "A novel yield optimization technique for digital CMOS circuits design by means of process parameters run-time estimation and body

bias active control," *IEEE , Transactions on Very Large Scale Integration (VLSI) Systems*, pp.630 - 638, May 2005.

[10] T. Nakura et al., "Autonomous di/dt noise control scheme for margin aware operation," *IEEE , European Solid-State Circuit Conference (ESSCIRC)*, pp.467-470, Sept. 2005.

[11] Wohlmuth et al., "A low power 13-Gb/s 2/sup 7/-1 pseudo random bit sequence generator IC in 120 nm bulk CMOS," *Integrated Circuits and Systems Design(SBCCI)*, pp 233 - 236, Sept. 2004.

[12] K. H. Dia, "二線式ドミノ回路による終了検出型マイクロコントローラの設計," 修士論文, 東京大学大学院工学系研究科電子工学専攻,2006.

[13] Gill, B.S et al., "Node sensitivity analysis for soft errors in CMOS logic," in Proc of *IEEE Internationa Test Conference(ITC)*, pp 9, Nov. 2005.

[14] 額田忠之, " Z80 ファミリ・ハンドブック", CQ 出版, 1991 年.

[15] Poki Chen et al., "A Time-to-Digital-Converter-Based CMOS Smart Temperature Sensor," *IEEE J. Solid-State Circuits*, vol. 40, no. 2, pp. 451 - 461, Feb. 2005.

[16] アドバンテスト, "T2000 マニュアル,"

[17] Kuroda T. et al., "Variable supply-voltage scheme for low-power high-speed CMOS digital design," *IEEE J. Solid-State Circuits*, vol. 33, no. 3, pp. 454 - 462, March 1998.

本研究に関する発表

[1] 石井 健, 池田 誠, 浅田 邦博, “二線式ドミノ回路の評価,” 電子情報通信学会 ソサイエティ大会論文集, C-12-34, pp. 95, 2006 年 9 月.

[2] Makoto Ikeda, Taku Sogabe, Ken Ishii, Masayuki Mizuno, Toru Nakura, Koichi Nose, Kunihiro Asada, “LAGS System Using Data/Instruction Grain Power Control,” *IEEE International Solid-State Circuits Conference (ISSCC)* , Feb. 2007.(to appear)

謝辞

初めに、この修士論文を書くにあたり多大なご指導をしてくださいました池田 誠助教授、浅田 邦博教授に深く感謝いたします。

鄭 若丹彡氏、佐々木 昌浩氏、小松聡氏は、研究のアドバイスだけでなく、研究しやすい環境を提供して下さいました、深く感謝いたします。

吉田 浩章氏、飯塚 哲也氏、谷内出 悠介氏はいつも優しく接して下さり研究の助言を頂きました、深く感謝いたします。

同じ研究室として、様々な助言を下さった風間 大輔氏、Cho Yong Sung 氏、橋本 紘和氏、門馬 太平氏、梁 志成氏、に深く感謝いたします。

同じプロジェクトとして研究をともにした曽我部 拓氏にも深く感謝いたします。

他研究室の皆様方にも深く感謝いたします。

チップ試作にあたり、ご協力いただいた東京大学大規模集積システム設計教育研究センター、ローム株式会社、凸版印刷株式会社、株式会社半導体理工学研究センター、富士通株式会社、松下電器産業株式会社、NEC エレクトロニクス株式会社、株式会社ルネサステクノロジ、株式会社東芝に感謝いたします。