卒 業 研 究 報 告

題　　目

CDMA方式における
PN符号ジェネレータの設計

指 導 教 員

矢野　政顯　教授

報 告 者

北野　克幸

平成 13 年　　2 月　　9 日

高知工科大学 電子・光システム工学科

概要

LSI 設計は 1980 年代後半から 1990 年代になって大きく変化した。それまでは主に回路図やブロック図などの図面をもとにして設計されてきたが、1980 年代後半に普及し始めたハードウェア記述言語（HDL）が一変させた。HDL によりハードウェア設計はまるでプログラミング感覚で行われるようになり、コンパイルすることで論理合成されるようになった。この変化によってハードウェア設計は、その生産性を向上させただけでなく、ソフトウェア同様に設計資産（IP）の再利用の道を開き、LSI IP なる概念の定着を促進した。

本報告は私が高知工科大学において卒業研究として行った集積回路の設計手法についてまとめたものである。第 1 章にて目的、設計ターゲットを説明し、第 2 章では回路設計の基礎である論理回路やハードウェア記述言語についてまとめた。次に第 3 章では設計ターゲットを実現するために必要な無線通信技術に触れ、第 4 章では設計ターゲットのチップ化のための FPGA についてまとめた。最後に第 5 章では実際に設計した回路について説明し、FPGA 書き込みための手順、測定結果について述べた。

目次

第 1 章　はじめに

現代の生活において、大規模な計算機から家電製品にいたるまであらゆる電気機器に集積回路（LSI）が組み込まれている。IT（Information Technology）化の波にのまれ、半導体市場は活性化し、LSI 技術は今後さらに発展を遂げるといわれている。これに応じて、企業間では技術競争が激化し、いち早く目的の機能を実現することが要求されており、設計技術者はさまざまな用途の回路を設計できるのみでなく、しかも短期間でそれを実現しなければならない。

本研究の目的は、このような LSI の基礎設計手法を習得することであった。このため 3 年次までに履修した「論理回路」、「回路理論」、「電気磁気 1,2」、「電子回路」、「情報処理概論」、「電子デバイス」、「デバイスプロセス」、「計算機アーキテクチャ」などの授業をあらためて学習しただけでなく、「HDL 設計手法」、「CMOS トランジスタ特性」などについてのセミナーに参加し、LSI 設計手法についての理解を深めた。

最近、身のまわりで無線通信を利用するシステムが普及し始めてきた。コードレス電話や携帯電話、カーナビゲーションシステムがそれである。これらの移動体通信システムに共通して使われるのが、スペクトル拡散技術である。この技術を利用したものとして、CDMA 方式の携帯電話機などがある。無線通信における CDMA 方式は次世代通信技術として大いに期待されている。このため LSI 設計手法を学ぶための研究題目として、CDMA 方式を実現するために必要な PN 符号を発生する回路（PN 符号ジェネレータ）を選択した。設計した回路をレイアウト化し、実際にチップ化したかったが、１年という短い期間では困難であるため、FPGA を使用し、オシロスコープやロジックアナライザなどの測定器を用いて、設計した回路を評価した。

第 2 章　ディジタル集積回路の基礎

この章では、卒業研究を進める上で前提となる基礎知識などをまとめた。基本的にはこれまでの授業やセミナーの内容をまとめたものである。

2.1.　LSI のしくみと設計フロー

LSI 設計はまず、どのような機能を果たすのかを決定する仕様設計から始める。設計メーカの大半はカスタマーからの依頼で目的の LSI を設計するが、大企業などでは汎用 LSI も設計している。

次に動作仕様に従って論理設計を行う。ここでは回路図、HDL 記述からの論理合成などを用いてトランジスタや論理ゲートなどの回路素子間の接続関係を表すネットリストを作成する。次にこのネットリストをもとにレイアウト設計を行う。レイアウトは、LSI 上に実際に作成されるパターンのことである。このパターンの作成過程は、レイアウトエディタなどを用いてすべての素子を手書きで作成する「フルカスタム設計」手法と、あらかじめ論理ゲートのレイアウトパターンを用意しておき、ネットリストから得られた回路の結線情報をもとにしてそれらの配置と配線を半自動的に実行する「セミカスタム設計」手法とに大別される。このように設計は、ある程度システム化されており、その設計フローは「仕様設計」、「HDL 記述設計」、「論理設計」、「回路設計」、「レイアウト設計」といった順になっている。

また LSI の製造にあたっては、設計者により作成されたレイアウトパターンに従って、実際の LSI 製造工程で使用されるマスクが製作される。このマスクパターンをリソグラフィ技術でシリコン基盤上に転写して、イオン注入、エッチング、拡散などを行うことにより、LSI が製造される。図 2.1 にシリコン（Si）基盤を用いた LSI の設計、製造フローを示す。

設計から製造までの過程はこのようになっているが、設計にあたっては HDL の採用により、設計にかかる時間が大幅に短縮できるようになった。またシミュレーション検証ツールの利用により、デバック作業もほとんどが計算機上で実現できるようになった。そしてチップ化にあたっては FPGA のようなプログラム可能なデバイスへの変換技術により、シミュレーションで検証済みの回路が一回で動作する可能性が高まってきた。

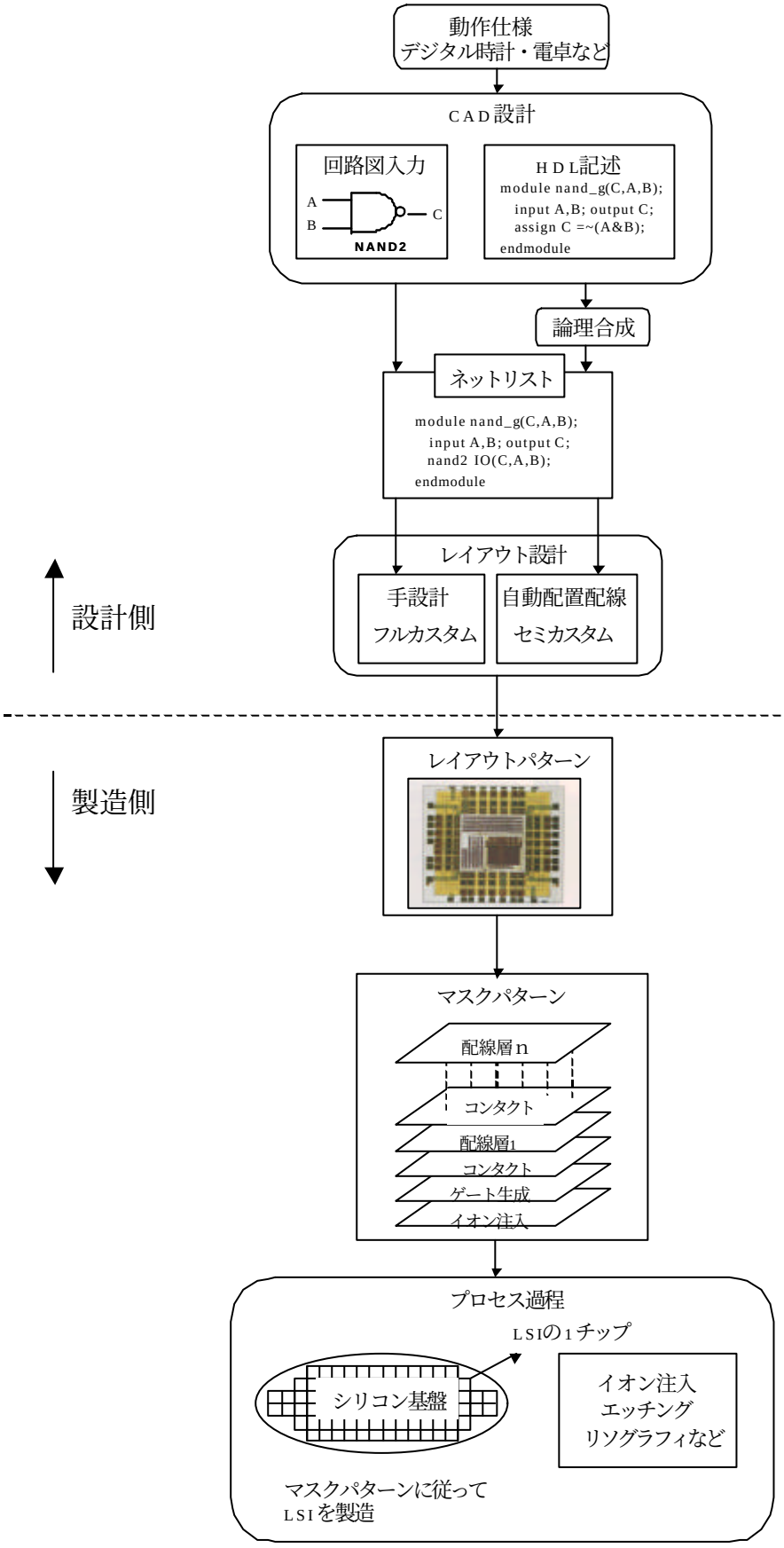


図2.1　LSIの設計、製造フロー

2.2.　単相クロック同期回路

現実のディジタル回路のほとんどは、内部に情報を保持しつつ演算などの動作を行う。このように、内部に情報を保持しつつ動作する論理回路を総称して順序回路と呼ぶが、その中で基本的な設計対象として想定するのは、設計方法や設計に必要な環境（ツール）がもっとも確立している「単相クロック同期回路」である。

単相クロック同期回路とは以下の条件を満たす回路のことである。

・記憶素子はフリップフロップ（FF）のみである。

・外部から単一のクロックが与えられている。

・このクロックの立ち上がりもしくは立ち下がりエッジのどちらか一方にすべての FF が同期して動作する。ただし、原則として立ち下がりエッジに同期する FF と立ち下がりエッジに同期する FF が混在してはならない。

単相クロック同期回路では外部から単一のクロック信号が与えられ、すべての FF がこのクロックの立ち上がりもしくは立ち下がりエッジのどちらかに同期して動作する。FF は通常 D フリップフロップ（DFF）が用いられる。DFF は、図 2.2 に示すように単一のデータ入力端子 D をもち、クロックに同期してデータを取り込み Q より出力する。単相クロック同期回路は、FF を記憶素子としてそれらの間に組み合わせ回路をはさんだ構造をとる。

単相クロック同期回路は、

1)　入出力ピン
2)　FF からなる記憶素子
3)　それらの間に存在する論理ゲートからなる組み合わせ回路

からなる。図 2.2 に単相クロック同期回路の模式図を示す。

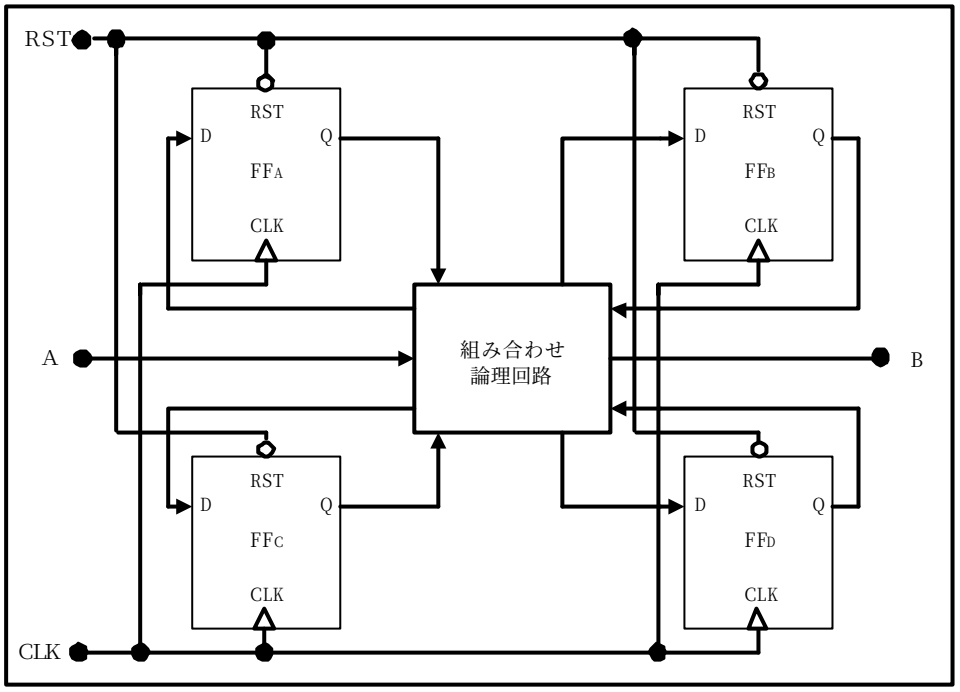


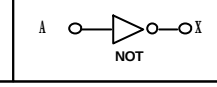
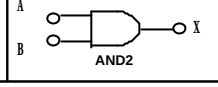
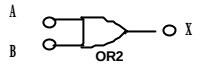
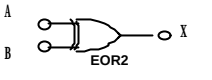
図2.2　単相クロック同期回路

これは入力ピン A、出力ピン B を備え、記憶素子として、FF_A、FF_B、FF_C、FF_Dの４個の DFF を備えた回路である。入力ピン、出力ピンは回路の入力および出力となるもので、階層的に設計している場合には、上位の回路との接続点であり、最上位の回路の場合には、LSI の外部にでる端子である。

2.2.1. 論理ゲートと組み合わせ回路

組み合わせ回路とは、現在の入力のみで出力が決定する回路のことである。組み合わせ回路は、論理回路を実現する論理ゲートから構成される。論理ゲートは、否定（NOT)、論理積（AND)、論理和（OR）の 3 つの基本演算そのもの、もしくはそれを組み合わせた論理を実現するものである。基本的な論理ゲートのシンボルと各論理関数の真理値表、ならびに VHDL、Verilog-HDL での記法を表 2.1 に示す。この他、基本論理ゲートとしては、AND、OR、XOR に NOT を組み合わせた NAND、NOR、XNOR がある。CMOS 回路などでは、AND、OR よりもトランジスタ数の少ない NAND、NOR がよく使用される。

表2.1　基本論理ゲート、真理値表、VHDL、Verilog-HDLによる記法

名前		NOT		AND				OR				XOR			
MIL 記号															
入力	A	0	1	0	0	1	1	0	0	1	1	0	0	1	1
	B	-	-	0	1	0	1	0	1	0	1	0	1	0	1
出力	X	1	0	0	0	0	1	0	1	1	1	0	1	1	0
VHDL		X <= not A		X <= A and B				X <= A or B				X<= A xor B			
Verilog-HDL		X = ~A		X = A&B				X = A|B				X = A^B			

2.2.2. D フリップフロップ

図 2.3 に示す DFF は、2.2 節で述べたとおり、クロックの立ち上がり、もしくは立ち下がりのエッジで入力 D の値を取り込み、Q にその値が出力される。立ち上がりで動作するものを、ポジティブエッジトリガ型、立ち下がりで動作するものを、ネガティブエッジトリガ型と呼ぶ。RST 信号は、クロックとは同期、もしくは非同期に与えることのできるリセット信号である。RST が 0 になると Q は 0 にクリアされ、RST が 1 になるまでこの状態が保持される。この他に、プリセット（PR）入力をもつ DFF もある。DFF は入力を１クロックサイクル遅延させて出力ピンに出力する遅延素子としての機能をもつ。たとえば、DFF を４個数珠つなぎにすれば、入力が４クロックサイクル遅れて出力に伝わる（図 2.5）。同期回路内で DFF は単なる遅延素子としてだけでなく、回路内で保存しておかなければならない値を保持する役割をもつことが多い。このためには、出力 Q を入力側に戻して、入力に２入力のセレクタを挿入したイネーブルつき DFF（図 2.4）を用いる。イネーブルつき DFF では、ENABLE が１の時に、入力 D を取り込み、ENABLE が 0 の時は現在の値を保持する。

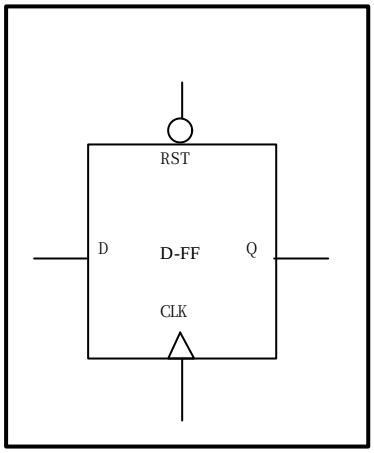


図2.3　エッジトリガ型
Dフリップフロップ

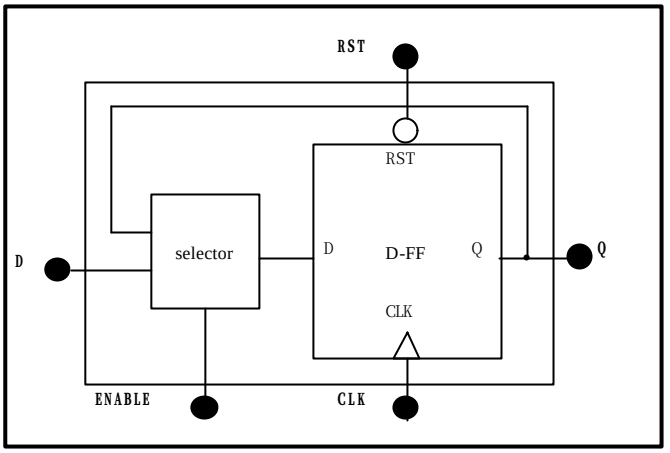


図2.4　イネーブルつきエッジトリガ型
Dフリップフロップ

FF ではクロックの前後に、セットアップタイム（t_{setup}）とホールドタイム（t_{hold}）が必要である（図 2.6 参照）。FF の D 入力は、クロック信号のエッジが与えられるより t_{setup} 前に確定し、エッジ後 t_{hold} だけその値を保持しなければならない。この条件が守られない場合、メタステーブルと呼ばれる準安定状態に移行し、発振などの誤動作を起こす場合がある。

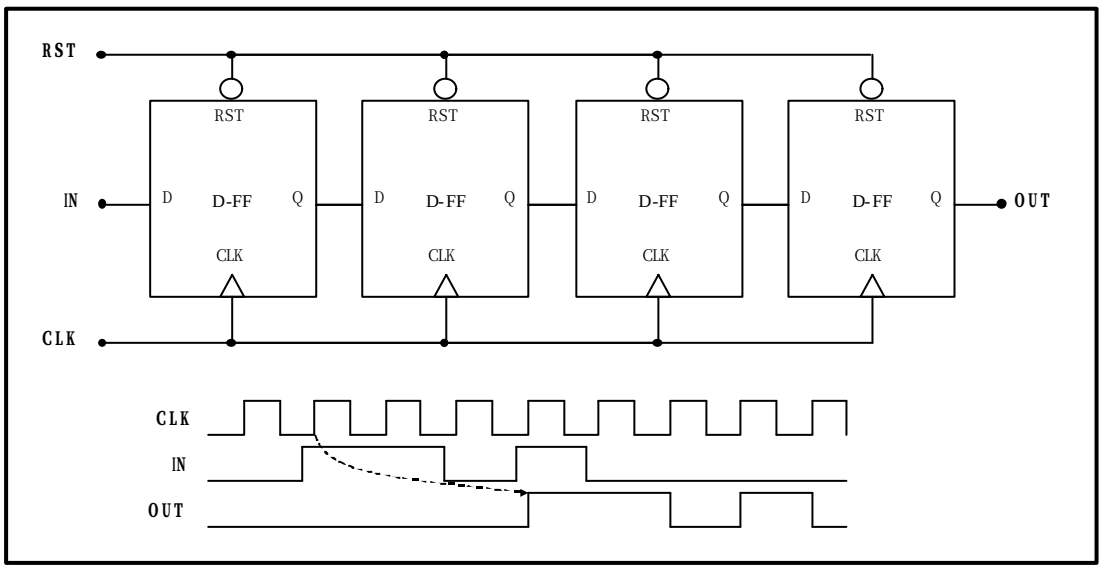


図2.5　DFFは、遅延素子として利用することができる

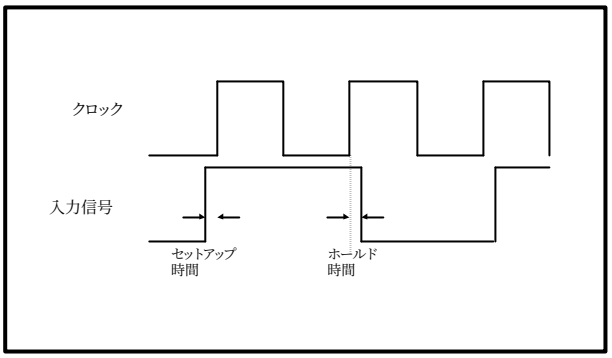


図2.6　ポジティブエッジトリガ型フリップフロップのセットアップ時間とホールド時間

2.2.3. 単相クロック同期回路の性能

単相クロック同期回路の性能は、クロック周波数により決定される。クロックの周波数が高いほど同じ時間で多くの処理が可能となる。クロックの周波数を決定するのは、入力値が確定するまでの間の時間（ディレイ）である。回路の中でディレイがもっとも大きいパスを、クリティカルパスと呼ぶ。このクリティカルパスの遅延を小さくすることが回路の性能をあげる上での大きな鍵となる。クリティカルパスは論理ゲートの段階などに比例して大きくなっていく。遅延を小さくするには FF 間の論理ゲートの段階をできるだけ少なくすることが望ましい。

2.3. ハードウェア記述言語

2.3.1. ハードウェア記述言語の歴史

第１章で述べたとおり、IC や LSI といった半導体集積回路の論理設計は、トランジスタや論理ゲートのシンボルを用いて、所望の機能を実現するように回路設計する方法が主流であった。このような方法では、回路自体を人手により最適化できるというメリットはあるものの、所望の機能を人間の頭を使って回路図に落としこむという作業が必要になる。現在の 100 万ゲートにまで達する規模の回路を人間の頭だけで設計するのは、ほぼ不可能に近い。ソフトウェアにたとえると、回路図による設計はアセンブリ言語により大規模なプログラミ

ングを行っているようなものである。

100 万ゲートにもおよぶ大規模な回路を効率よく設計するために生まれたのがハードウェア記述言語（HDL：Hardware Description Language）である。

HDL 採用の目的は

> 目的とする回路の機能を人間のわかりやすいようにテキストで記述して、その記述から自動的に回路記述を生成する

ことである。HDL から回路記述を得ることを一般に論理合成と呼ぶ。現在のソフトウェア開発が、アセンブリ言語から C や C++といった高級言語に主流を移しているのと同様に、ハードウェアの開発も、HDL が主流となってきている。さらに大規模な回路を効率よく設計するために、C や C++の記述から直接ハードウェアを合成する設計手法も実用化の段階にはいってきている。

2.3.2. Verilog-HDL と VHDL

現在、LSI の世界では Verilog-HDL と VHDL という２種類の HDL が標準化され、用いられている。Verilog-HDL は、元々Cadence 社の開発した Verilog-XL というシミュレータ用の入力を記述する言語から派生した HDL であり、その文法は C 言語に似ている。VHDL は、Ada という言語をもとに、ハードウェアの仕様を書くことを目的に開発された言語である。両者とも、自動的に回路を生成（合成）することを目的として開発された言語ではないので、文法的には間違っていないが、合成できないということが生じる。

Verilog-HDL は、抽象度が低く、回路がわかっている電気系技術者向きの言語である。VHDL は抽象度が高く、回路がわかっていなくても書ける情報系技術者向きの言語である。両者の違いがもっとも現れるのが状態の FF へのマッピングの仕方である。Verilog-HDL では、設計者自身が状態の FF へのマッピングを定義しなければならない。これに対して、VHDL では状態の名前だけを定義すればよく、FF へのマッピングは合成する側に任せることができる。

例えば、S0、S1、S2 という３つの状態をもつ回路を考えると、Verilog-HDL では、

```
‘define S0 2’b00;
‘define S1 2’b01;
‘define S2 2’b10;
```

と、S0、S1、S2 を２ビットの FF で表現するというように、明示的に書かなければならない。

それに対して VHDL では、

```
type state is (S0,S1,S2);
```

と、S0、S1、S2 をもつ型 state を新たに定義するだけでよい。

2.3.3. HDL を使うメリット

HDL を使う一番大きなメリットは、

> HDL そのものが回路の仕様に近いために、仕様変更、バグに対応することが容易である

ということである。所望の機能をいったん回路図に落とした後に、仕様が変更になったり、バグが見つかった場合、もう一度回路設計をやり直さなければならない。HDL の場合は HDL の記述を修正するだけで、後は論理合成を行うツールを用いて自動的に回路を生成するだけである。回路設計に使用される論理ゲートの種類（ライブラリ）は、各半導体ベンダにより異なる。あるベンダ（メーカ）のライブラリを用いて回路設計を行った回路を、別のベンダのライブラリで再設計するには非常に手間がかかる。HDL 自体は回路の機能そのものを表し、ライブラリには依存しない。論理合成ツールに与えるライブラリを変えるだけで、それぞれのライブラリ用に最適化された回路が生成される。HDL を使うメリットを図 2.7 に示す。

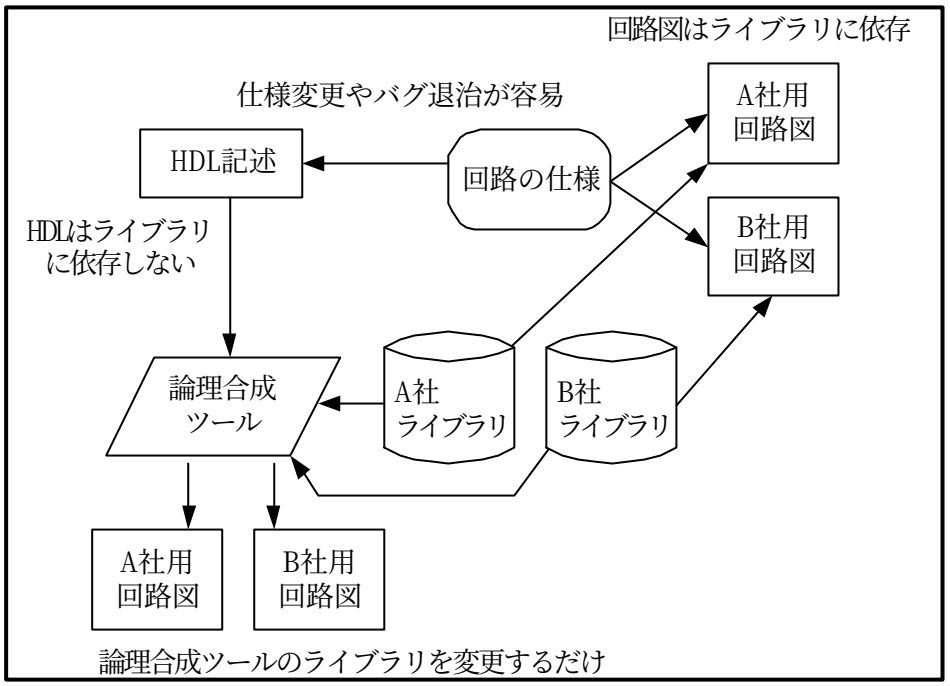


図2.7　HDL設計のメリットのイメージ図

2.3.4. HDL を書くための準備

HDL の文法を知っているだけでは、最適な回路は設計できない。HDL からどのような回路が生成されるかをある程度知った上で記述しないと意図しない回路が生成される場合がある。小規模な回路の場合は、回路の機能を決めた上で、いきなり HDL のコーディングにはいることもできるが、通常は、

1. 回路中のフリップフロップ、レジスタなどの記憶素子の構成を決める。

2. それらのフリップフロップ、レジスタをどのように接続するかを考え、ブロック図を書く。レジスタ間に存在する組み合わせ回路の詳細は考える必要はない。

という設計過程を踏む。ブロック図は回路図を書く前にも必要であるが、HDL を書く前にも必要である。

HDL にて書く上で一番重要なのは、

どのような回路を意図するか？

である。HDL から合成した回路が意図した通りになっているかどうかを確かめるのも重要な作業である。またブロック図を書いて、フリップフロップ、レジスタをどのように配置するかを考えるのも、意図した以外のフリップフロップ、レジスタが挿入されるのを防ぐという意味で重要な作業である。

2.3.5. HDL の使用例

VHDL、VerilogHDL について簡単に解説したが、ここで HDL の簡単な使用例を載せておく。今回は全体を通して VHDL を使用しているので、VHDL 言語での 1 ビットコンパレータを設計ターゲットとした。

~条件~

・入力を A、B、出力を C と定義する。
・出力 C は A=B ならば C=1、A≠B ならば C=0 とする。

条件から真理値表は次のようになる。

表2.2　コンパレータの真理値表

A	B	C
0	0	1
0	1	0
1	0	0
1	1	1

条件および真理値表から VHDL 記述を行う。図 2.8 に VHDL 記述を示す。

```
 library ieee;
use ieee.std_logic_1164.all;
entity COMPARE is
port (
 Ain: in std_logic;
 Bin: in std_logic;
 Cout:out std_logic);
end  COMPARE;
  architecture DATAFLOW of COMPARE is
begin
    Cout<= not(Ain xor Bin);
   end DATAFLOW;
```

図2.8　コンパレータのVHDL記述

エンティティ文にて信号 A、B、C、を定義し、アーキテクチャ文にて回路の動作仕様を示している。

次に VHDL にて記述したコンパレータをシミュレーションするために、テストベンチを記述する（図 2.9 参照）。

```
library ieee;
use ieee.std_logic_1164.all;
use std.textio.all;
use work.COMPARE;
entity TESTBNCH is
end TESTBNCH;
architecture stimulus of TESTBNCH is
component COMPARE is
port (
Ain, Bin: in std_logic;
Cout: out std_logic);
end component;
signal Ain, Bin: std_logic;
signal Cout: std_logic;
begin
DUT: COMPARE port map (
Ain, Bin,Cout);
```

```
STIMULUS1: process
Begin
Ain <= '0'; Bin <= '0'; wait for 100 ns;
Ain <= '0'; Bin <= '1'; wait for 100 ns;
Ain <= '1'; Bin <= '0'; wait for 100 ns;
Ain <= '0'; Bin <= 'X'; wait for 100 ns;
Ain <= '1'; Bin <= '1'; wait for 100 ns;
wait;
end process STIMULUS1;
end stimulus;
```

図2.9　コンパレータのテストベンチ

測定信号を A、B、C を定義し、プロセス文について次のように設定する。

A=0、B=0 のときの出力 C
A=0、B=1 のときの出力 C
A=1、B=0 のときの出力 C
A=0、B=X のときの出力 C
A=1、B=1 のときの出力 C

上の状態変化を 100ns ごとに測定する。シミュレーション結果を図 2.9 に示す。

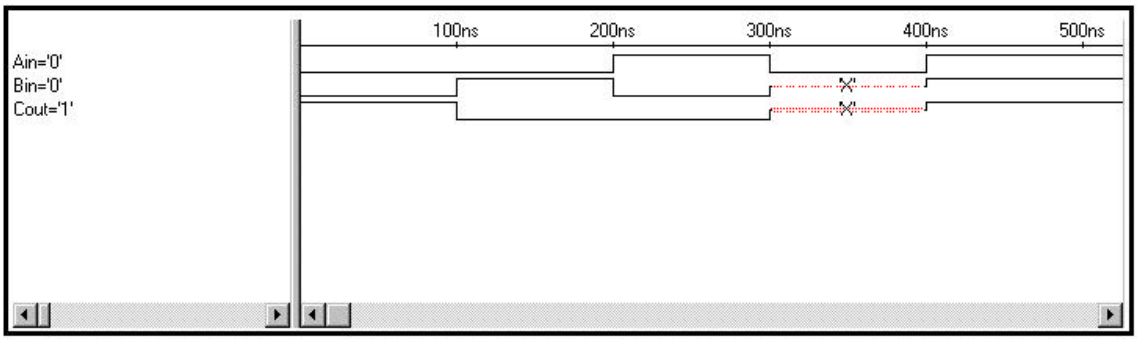


図2.10　コンパレータのタイムチャート図

表 2.2 の真理値表と見比べると予測通りの出力結果が得られていることがわかる。

VHDL にて記述した後、本来ならば論理合成して、ゲートレベルでの回路図を確認するのだが、今回は論理合成用ツールが利用できないので、真理値表および VHDL から、回路図を図 2.11 に設計した。

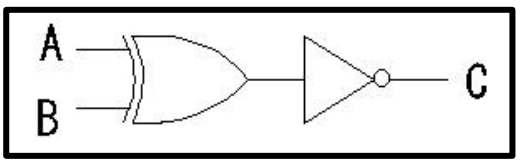


図2.11　回路図

なお今回 VHDL 記述にて使用したソフトは、multiSIM、VHDL Simulator（Electronics WORKBENCH 社）の製品である。今後、本報告で用いる VHDL 記述はすべて multiSIM、VHDL Simulator によるものである。

第 3 章　CDMA 方式の基礎

この章では無線通信技術におけるスペクトル拡散技術を解説し、CDMA 方式の基本的原理について説明する。

3.1. 背景

近年の無線通信技術の発展は目覚しく、1980 年代はじめに携帯電話が登場して以来、移動体通信市場は爆発的な成長を遂げている。複数組の通信を同じ地域（場所）で使用するとき、両者が互いに混信を起こさずに、それぞれの相手と通信できるようにすることを多元接続、あるいは多重という。多元接続の方式は大きく分けて 3 つの世代に分かれる。

第 1 世代　：　FDMA（Frequency Division Multiple Access）
　　周波数分割多元接続方式
第 2 世代　：　TDMA（Time Division Multiple Access）
　　時分割多元接続方式
第 3 世代　：　CDMA（Code Division Multiple Access）
　　符号分割多元接続方式

第 1 世代である FDMA 方式はユーザごとに使用周波数を変える方式であり、第 2 世代の TDMA 方式はユーザごとに時間を区切って同じ周波数を共有する方式である。CDMA 方式はスペクトル拡散技術（スペクトル拡散技術については 3.2 節にて説明）を用いて、多くのユーザが同一の周波数と時間を共有する方式である。（図 3.1 参照）

現在は第 3 世代に入っているが、それぞれの世代の特徴を簡単にまとめる。

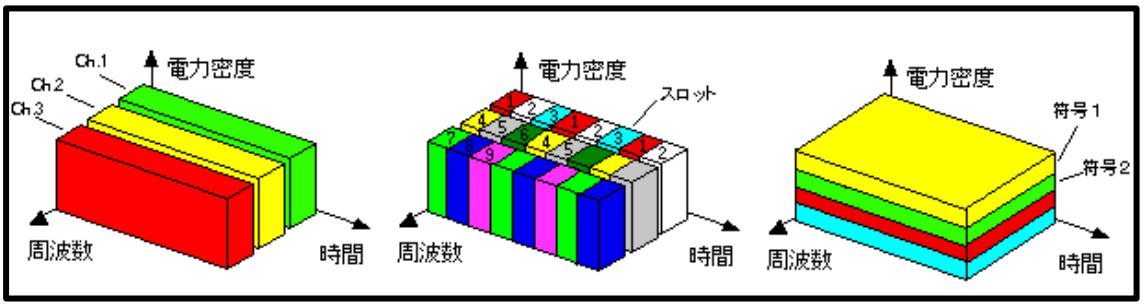


図3.1　FDMA、TDMA、CDMA

3.1.1. FDMA 方式について

FDMA 方式は、従来のアナログ方式による携帯・自動車電話システムで広く採用されてきた方式である。この方式では、各ユーザが使用する無線チャネルを、ユーザごとに異なる搬送波周波数に設定して使用する。送受信周波数の設定は周波数シンセサイザによって行う。

3.1.2. TDMA 方式について

TDMA 方式は、１つの無線キャリアを複数のユーザで時間的に分割して使用する方法である。一定の時間周期で多数のタイムスロットと呼ばれる単位に分割するが、複数のユーザがそれぞれ異なるタイムスロットを使用するため、通信チャネルを有効に活用できる。この方式は、ディジタル方式の携帯・自動車電話や移動衛星通信の分野で広く使用されている。TDMA 方式の中で PDC 方式では上り／下りの通信周波数が異なる FDD （Frequency Division Duplex）方式を使用し、PHS 方式では上り／下りの通信周波数が等しい TDD （Time Division Duplex）方式を用いる。

高速のディジタル伝送を行う広帯域の TDMA 方式では、マルチパスによる周波数選択性フェージングの影響を受けるようになる。このため、周波数選択性フェージングが問題にならない程度に帯域幅を抑えるか、あるいは適応等化などの改善技術を適用して選択性フェージングを克服することが必要となる。また、移動機側には間欠的なバースト信号送信を行うために、高精度の同期回路が必要になる。

3.1.3. CDMA 方式について

CDMA 方式ではユーザごとにそれぞれ異なる拡散コードを設定し、複数のユーザが同一の広帯域無線チャネルを共有する。CDMA 方式はスペクトラム拡散技術に基づく方式で、変調後の信号の帯域幅を大きく拡散させることによってノイズや干渉の影響を受けにくくする技術である。また、広帯域であるためマルチパスを RAKE 合成することによってフェージング耐性を獲得することができる。スペクトラム拡散変調信号は、第三者に対しての秘匿性が優れているというのも大きなメリットである。さらにスペクトラム拡散変調信号でつくられた信号を用いると、信号到着時刻の相対値を正確に算出でき、この特徴を生かして移動体の現在位置を正確に測定できる。この特徴を応用したものが GPS（Global Positioning System）システムや車載型ナビゲーションシステムなどである。

3.2. スペクトル拡散変調

アナログ通信、ディジタル通信を問わず、ある情報を無線伝送によって相手方に伝えようとするとき、伝送したい情報波形（ベースバンド波形）を変調操作によって搬送波に重畳させる。このとき変調された信号は、ベースバンド波形と変調方式から決まるある周波数幅を有する信号になる。つまり変調された信号には帯域幅が存在している。

変調された信号の占有する帯域幅はベースバンド波形の変化速度や変調方式により大幅に変化するが、通常の通信目的に用いられる狭帯域変調方式では、ベースバンド波形の有する帯域幅の数倍程度以内に納まることがほとんどである。とりわけ最近では、前節にて述べたように、周波数の有効利用が叫ばれていて、同じ情報を伝送するなら、なるべく狭い帯域幅ですむような変調方式が好まれている。

移動体通信での無線機器では、利用に適した周波数帯が VHF 帯から準マイクロ波帯に限られている。それ以上の周波数帯では電波の直進性が強くなりすぎて建造物の陰に電波がまわり込みにくくなるために移動体通信には適していないし、それ以下の周波数帯ではアンテナが大きくなりすぎたり、ユーザの割り当てに十分な帯域幅がないために適していない（図 3.2 参照）。

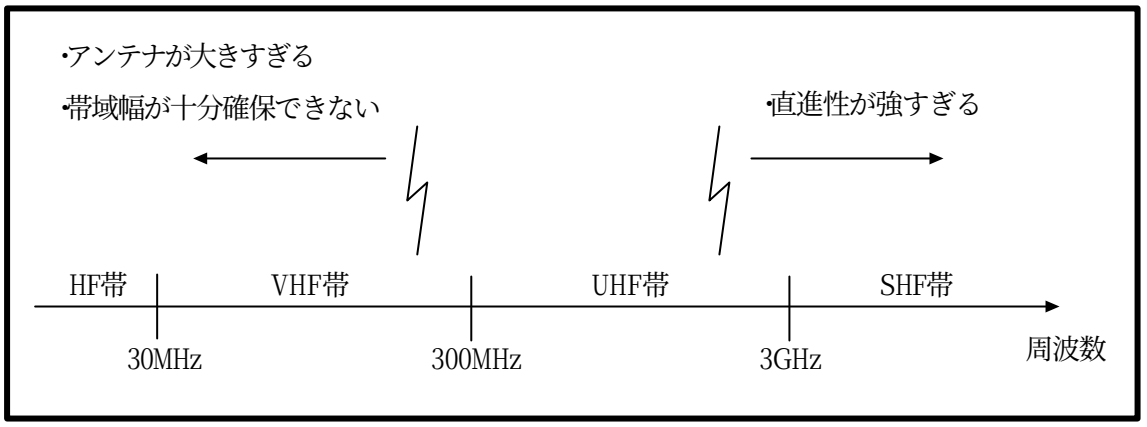


図3.2　移動体通信に適した周波数帯

いずれにせよ限られた周波数帯域幅しかないところに爆発的な需要が発生しているために、伝送信号の狭帯域化が移動体通信の課題となっている。

狭帯域な変調方式が占有する帯域幅は、例えばディジタル通信の場合であれば伝送したいディジタル情報のビットレートを Hz（ヘルツ）に読み替えた程度の帯域幅、アナログ通信では伝送したいベースバンド信号の占有帯域幅と同程度になる。

表3.1　占有帯域幅の対比

変調方式	占有帯域幅
狭帯域変調	ビットレートと同程度のオーダ
スペクトル拡散変調	ビットレートの数十～数千倍

そこで登場したのがスペクトル拡散（Spread Spectrum：SS）通信方式である。SS 方式は通常の狭帯域変調方式とは異なり、変調された後の信号の帯域幅を狭帯域変調によるそれに比べてはるかに広くさせる変調方式の総称である。変調信号のスペクトル成分を広範な周波数に拡げているためにスペクトル拡散という名称がつけられている。

スペクトル拡散技術には、代表的なものとして次の 2 方式がある。

DS（Direct Sequence）**方式：直接拡散方式**
FH（Frequency Hopping）**方式：周波数ホッピング変換方式**

次節でそれぞれを解説する。

3.2.1. DS 方式

携帯や PHS はもちろん、ＴＶ・ラジオなどの無線を使う通信では、音声などのデータを送りやすいように変調してから送信する。受信側では、送信側で使っている変調方法に応じた適当な方法で復調し、もとの信号をとりだすが、DS 方式では、一度変調した信号にスペクトル拡散という操作をしてから送信し、受信側では復調する前に逆拡散という動作を施す。スペクトル拡散、逆拡散は、ＰＮ符号という特殊な符号を掛け合わせるという単純な動作である（図 3.3 参照）。

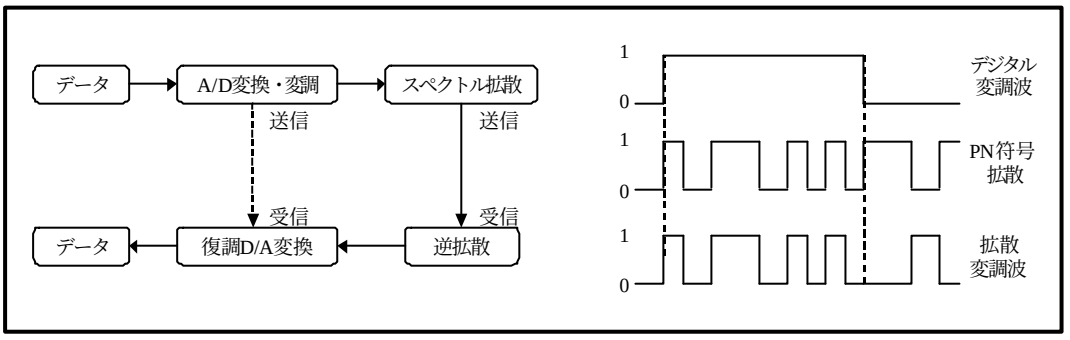


図3.3　DS方式の送受信とPN符号

ＰＮ符号は、ディジタル変調波と同じく２つの値をとるが、その変化の周期はディジタル変調波のものよりも一定以上すばやく変化するようにしてある。ここで、ディジタル変調波の変化のスピードをビットレート、ＰＮ符号のスピードをチップレートと呼んでいる。また、ビットレートとチップレートとの比を拡散率と呼ぶ。

図 3.3 のように信号を掛け合わせると、信号の電力密度スペクトル（周波数ごとの電力分布の様子）が大きく変化する。普通の信号では、電力密度スペクトルの分布は割合狭い周波数帯の中に収まっているが、スペクトル拡散を行った信号は電力密度スペクトルが薄く広く分布している。スペクトル拡散の前後における信号の周波数帯域の比は拡散率にほぼ一致する。
つまり次のような式が成りたつ。

$$\text{拡散率} = \frac{\text{チップレート}}{\text{ビットレート}} = \frac{\text{拡散後の帯域幅}}{\text{拡散前の帯域幅}}$$

この信号は適当なタイミングで、かつ送信側で使用したものと同じＰＮ符号を使って逆拡散したときのみ、スペクトル拡散する前の信号に戻る。

3.2.2. ＦＨ 方式について

直接拡散方式では、１次変調した信号にＰＮ符号を直接掛け合わせてスペクトル拡散を行うが、周波数ホッピング方式では、１次変調した信号を、適当なタイミングで周波数変換をかけることによってスペクトル拡散を行う。

周波数変換のパターンは「ホッピングパターン」といい、このパターンに従って周波数変換を行うと、外から見たとき搬送波周波数がランダムかつ広い範囲で飛び回るようになる。周波数ホッピングを行うと、周波数スペクトルは図3.4 のようになる。

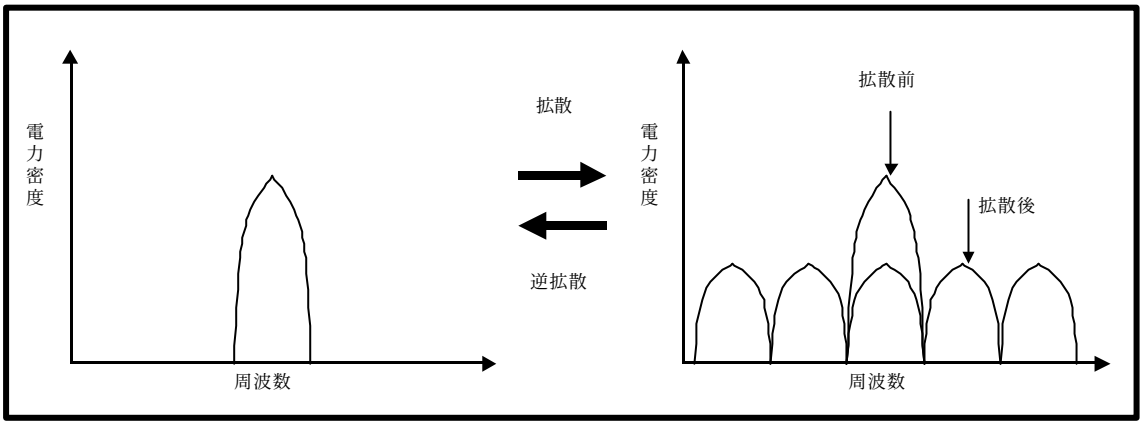


図3.4　FH方式の拡散と逆拡散

なお、この図は周波数スペクトルを長い時間観測したときの平均をとったもので、それぞれの山は、別々のタイミングで発生したものである。信号スペクトルの形は直接拡散のものと多少違うが、元の信号を周波数帯域が広く、電力密度の薄い信号に変換するという点ではよく似ている。動作だけをとってみれば、直接拡散方式のＰＮ符号がホッピングパターンに置き換わったようなものとい

える。逆拡散も直接拡散方式と似たようなことを行う。元の信号を取り出すには、受信した信号を送信したときと同じホッピングパターンを使って周波数変換をかける。この場合、ホッピングパターンと周波数変換のタイミングが一致したときのみ元の信号が取り出せる。

周波数ホッピング方式は、１次変調した信号速度(ビットレート)とホッピングの速度の割合によって、SFH(Slow FH)と FFH(Fast FH)の２つに分かれる。SFH はビットレートがホッピング速度よりも十分速いケースを指し、FFH はホッピング速度がビットレートとほぼ同じか、速いケースを指す。周波数ホッピングで周波数変換を行う「ホッピングシンセサイザー」は、安価かつ高速で動作するものが作りにくいものの、広帯域に信号を拡散させやすいため、今までは主に軍事用の通信機器で使用されていた。しかし最近になって FFH も 1 つのチップで実現できるようになり、「bluetooth(パソコンなどとの無線接続システム)」で採用されている。その一方 SFH 用のホッピングシンセサイザーは比較的作りやすいため、携帯電話の事実上の世界標準規格 GSM（Global System Mobile）で採用されている。周波数ホッピングは周波数ダイバーシティを行っているのと似たような効果が得られるため、GSM ではフェージング対策として用いている。

3.3.　PN 符号について

3.3.1.　PN 符号とは

ＰＮ符号のＰＮとは、疑似ランダム雑音（Pseudorandom Noise）の頭文字である。つまりＰＮ符号とは、ランダム雑音を模擬するビット列を表し、「疑似」という言葉どおりＰＮ符号は真のランダム性をもたない。つまりＰＮ符号はあらかじめ決められたビット列を周期的に繰り返している。広帯域 CDMA システムにおいてＰＮ符号は次の目的で使われる。

- 信号を広い周波数帯域に拡散する。
- ある有限の周波数帯域に、複数の異なる情報を付加できるよう個々の信号を独立して符号化する。
- 基地局と端末の間で、同期を確立する。

3.3.2. 自己相関と相互相関

スペクトル拡散に適したＰＮ符号は、良好な自己相関関数と相互相関関数をもつ。
ここでいう相関とは、信号の類似性を表す尺度である。

（1）自己相関：信号 $f(t)$と、これを δだけずらした信号 $f(t-\delta)$の相関を表す。
（2）相互相関：信号 $f(t)$と、これと異なる信号 $g(t-\delta)$との相関を表す。

ここでビット列「1110010」の自己相関をみてみる。図 3.5 に示すように、ビット列「1110010」と、これを 1 ビットずつ右に巡回シフト（バレルシフト）したビット列の間の相違を比較する。図 3.6 は元のＰＮ符号とこれを時間方向に 2 単位シフトした波形である。

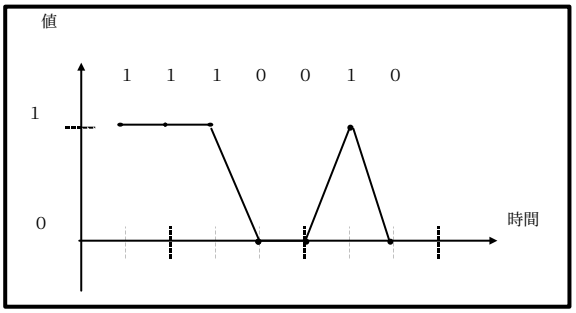


図3.5　あるPN符号の時間波形

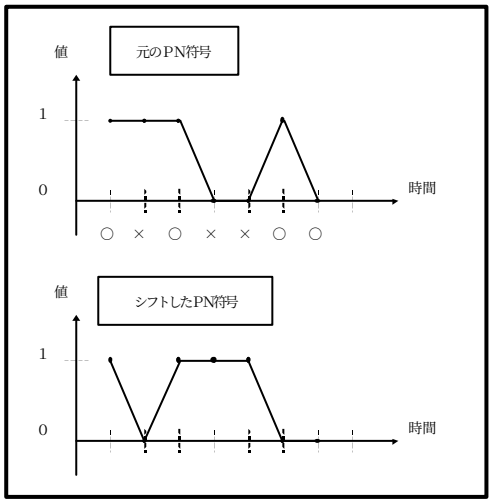


図3.6　2つの波形の比

この 2 つの波形の似ている程度を評価するのが「相関（correlation）」である。相関は以下の式で定義される。

相関値　＝　一致した符合の数　－　不一致の符号の数

この式を用いれば、図 3.6 の２波形の相関値は－１となり、0 に近い小さな値となる。これはこの２波形が似ていない（相関が小さい）ことを意味する。表 3.2 にこの相関特性をまとめておく。元の符号とまったく同じ波形である時間シフト 0 もしくは 7 の符号だけが大きな相関値 7 となり、ほかはすべて－１という小さな値になる。つまりこの場合、シフトの回数が 7、14、21、・・・、7ｎ目のみに大きな相関が現われ、これ以外では相関が低いことがわかる。

表3.2　自己相関関係が強い信号

PN符号	シフト量	元のコードと一致する数	不一致の数	相関
1110010	0	7	0	7
0111001	1	3	4	-1
1011100	2	3	4	-1
0101110	3	3	4	-1
0010111	4	3	4	-1
1001011	5	3	4	-1
1100101	6	3	4	-1
1110010	7	7	0	7

明確な起状がある

同様の考察をビット列「1111000」についておこなったものを表 3.3 に示す。表 3.2 と異なり、相関に明確な起状はみられない。

表3.3　自己相関関数が弱い信号

PN符号	シフト量	元のコードと一致する数	不一致の数	相関
1111000	0	7	0	7
0111100	1	5	2	3
0011110	2	3	4	-1
0001111	3	1	6	-5
1000111	4	1	6	-5
1100011	5	3	4	-1
1110001	6	5	2	3
1111000	7	7	0	7

明確な起状がない

先ほど述べたスペクトル拡散に適した自己相関特性をもつＰＮ符号とは、周期的に相関値のピークが現われるもの、だと言える。したがって、これら 2 つの例を比較すると、表 3.2 で示したビット列「1110010」は、表 3.3 示したビット列「1111000」よりも良好な自己相関関数をもつ符号ということがわかる。

次に図 3.5 について元の PN 系列とその PN 系列の 0 と 1 を反転させた系列との相関を考える。図 3.7 に元の PN 系列と反転した PN 系列を示す。

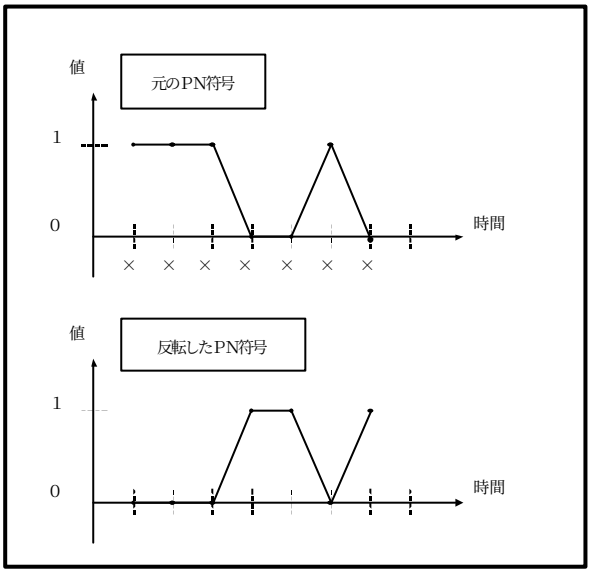


図3.7　反転波形との比較

この２つの波形はすべての符号が一致しないので、相関値は－７という負の大きな値となる。

以上の性質により、１ビットの情報すなわち、１もしくは０を送信する場合、「０」の意味で元のＰＮ符号（ＰＮ１）を送信し、「１」の意味で反転させたＰＮ符号を送信する。

同時に同じ伝送路に元のＰＮ符号をシフトした別のＰＮ符号（ＰＮ２）を用いて、「０」の意味でそのＰＮ符号（ＰＮ２）を送信し、「１」の意味で反転させたＰＮ符号を送信する。

送信路では上記複数の波形が加算され、まじってしまうが、受信波形（ＰＮ１とＰＮ２の加算）とたとえばＰＮ１との相関をとれば、ＰＮ１とＰＮ２の相関は小さいのでＰＮ１とＰＮ１の相関がでてくる。その相関値が大きい値であ

れば、「0」の情報であり、負の大きな値であれば「1」の情報であることがわかる。

送信側と受信側で同じ系列が同時に作れるためには、次々に発生される系列がある状態変数に応じて決まる関数で表せる必要がある。このような系列発生器としてよく紹介されるものにM系列（Maximum Length Code ）発生器がある。

3.4.　M系列

M系列はDS方式によるスペクトル拡散の説明をするときに必ず登場する符号系列である。M系列はシフトレジスタと加算器（Exclusive OR）で簡単に作ることができる。図3.8にM系列発生回路の一例を示す。ただし図3.8でシフトレジスタ中間部分から取り出されているタップ（帰還タップ）はレジスタのどの位置から取り出してもよいわけではない。特定の少数組み合わせの帰還タップを使った場合にのみ取り出されるデータ系列をM系列と呼ぶ。

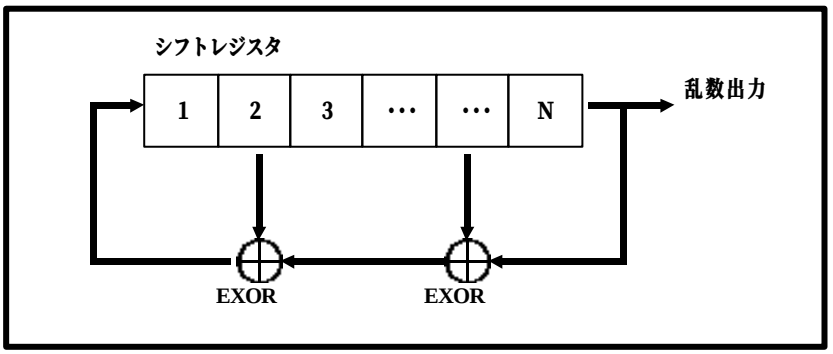


図3.8　7ビットM系列発生回路

図3.8のシフトレジスタは基本的には7ビットのデータを記憶する記憶素子でしかない。7ビットの記憶素子を使うと128通りの状態が記憶でき、またレジスタの内容は1回シフトするたびに128通りの状態のどれかに必ず移行する。そのためレジスタの初期状態がどうなっていても、最大128回レジスタをシフトすれば、それまでにレジスタの内容が必ず初期状態にもどるはずである。こ

れが発生回路の周期になり、それ以後同じデータが繰り返し出力される。

以上のことから図 3.8 の回路では周期が 128 を越える系列は絶対に作れない。しかし実際にはレジスタの初期状態をオールゼロとしてスタートさせるとレジスタの初段に帰還される値がゼロになって、以後レジスタの内容はオールゼロを繰り返す。そのためレジスタ状態としてオールゼロを除くと図 3.8 で実用可能な最大周期の系列は 127 であることになる。

3.4.1. M 系列の性質

M 系列が DS 方式の解説で必ず登場するわけは、M 系列の性質がよく解析され、数学的に美しい特徴をもっているからである。その特徴をいくつか列挙してみる。

（1）どんな周期の M 系列でも 1 周期中に含まれる「0」と「1」の数の割合が一定で、k 段のシフトを使ったとすれば、1 の数が 2^{k-1} 個、0 の数が（2^{k-1}）－1 になる。すなわち 0 の数がひとつだけ少ない。これはレジスタの状態としてオールゼロを除いたことから容易に推測できるだろう。0 と 1 の数のバランスがとれているので 0 を－1 に置き換える PN 系列では直流分がきわめて少なくなって好都合になる。

(2)系列中に 0 または 1 が連続して現れるとき、その連続の長さを run と呼ぶ。系列 1 周期中の長さ m の run の発生頻度は、長さ m＋1 の run の頻度のちょうど 2 倍になっている。これは（擬似乱数ではない）真の乱数がもつ性質と同じである。

（3）発生されたある M 系列と、レジスタの初期値だけが違う同じ M 系列のデータを加算すると、もとの M 系列を時間的にずらせただけの同一の系列ができる（シフト加法性）。ただしここでいう加算とは桁上げのない特別な加算で、排他的論理和（Exclusive OR）と同じと考えておけばよい。

他にも様々な表現で M 系列の性質をあげることができるが、ここではこれくらいにする。いずれにせよ真の乱数にきわめて近い性質をもっていることだけは覚えておく必要がある。

第 4 章　FPGA の構造と仕様

第 5 章にて PN 符号発生器を設計するにあたって、それを評価する際、FPGA を使用する。この章では FPGA の構造と仕様についてまとめた。

FPGA を理解する上でまず PLD について説明することで、プログラム可能な論理素子の歴史的背景にふれてみたい。

4.1.　PLD について

設計者が自由に動作仕様を変更できる LSI を、PLD（プログラマブルロジックアレイ）と呼ぶ。ASIC（特定用途向け IC）などの GA（ゲートアレイ）は、製造時点で論理ゲート間のマスクパターンを決定するので、一種の PLD と言えるだろう。このように製造時にマスクパターンを変更する LSI を MPD（マスクプログラマブルデバイス）という。これとは別に、製造後に機能を変更できる LSI を FPD（フィールドプログラマブルデバイス）という（図 4.1 参照）。

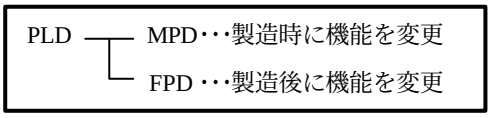


図4.1　PLDの分類

PLD は PLA（プログラマブルロジックアレイ）をベースにしたものとして誕生し、その後改良が重ねられた。PLA はプログラム可能な AND アレイと OR アレイからなり、任意の AND-OR 回路を設計できる。MPD タイプの PLA は複雑な組み合わせ回路をコンパクトに実現するための回路要素としてしばしば使われていた。一方 FPD タイプの PLA には、AND アレイのみをプログラム可能（OR アレイは固定）とした PAL（プログラムアレイロジック：AMD 社）がある（図 4.2 参照）。

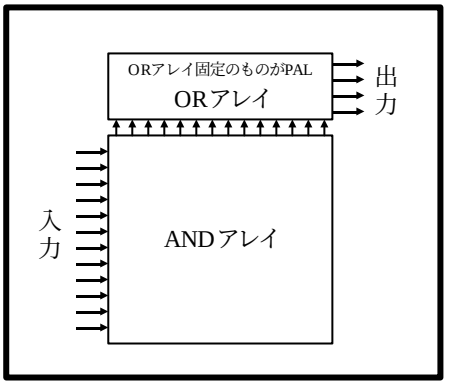


図4.2　PLA、PALの構造

4.2.　FPGA について

集積回路技術の向上にともない、製造した後に記憶素子などを用いて動作仕様を自由に変更できる大規模な集積回路が登場してきた。これは、論理ゲートをアレイ上に敷き詰めて、その間の結線を自由に変更できるものであり、製造時にマスクパターンを変更する GA に対し、FPGA（フィールドプログラマブルゲートアレイ）と呼ぶ。FPGA と区別するため以降、通常の GA を MPGA（マスクプログラマブルゲートアレイ）という。FPGA はその場（フィールド）でプログラム可能であり、素子によっては何度もその機能を書き換えることが可能である。最近の FPGA の集積度における伸びは著しく、最先端の技術では 100 万ゲートほどの大規模な回路を実現できる素子が登場している。MPGA は論理ゲートレベルで自動配置配線を行うセミカスタム LSI と FPGA の増加により、MPGA の利用率が年々減少している。

4.2.1.　FPGA の構造について

図 4.3 に示すように FPGA は大きく分けて、

- 組み替え可能な論理ブロック
- 論理ブロック間を接続する組み替え可能な配線

からなっている。論理ブロックとは組み替え可能な論理ゲートとフリップフロップを含む回路ブロックである。

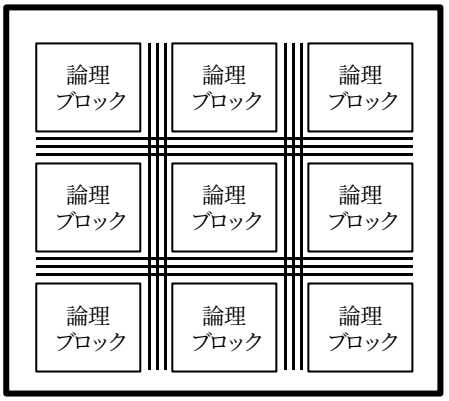


図4.3　FPGAの構造

4.3.　FPGA のプログラム方式

論理ブロックや配線にプログラムを書き込む方式で FPGA を分類すると、以下の 3 種類に分類される。

・SRAM などの揮発性メモリに書き込む
・EPROM、EEPROM などの不揮発性メモリに書き込む
・電圧をかけて、アンチフューズを短絡させる。

EEPROM（Electric Erasable Programmable ROM）は、何度でも書き換え可能であり、電源をきってもその内容を保持する（不揮発性）。EPROM（Electric Programmable ROM）は不揮発性であるが、書き換え可能なものとそうでないものがある。SRAM（Static RAM）は何度でも書き込み可能であるが、電源をきると書き込んだ内容を失ってしまう（揮発性）。アンチヒューズとはヒューズの逆で、電流を流すことで、電気的に短絡するヒューズのことである。一度短絡したヒューズは二度と元に戻らないので、一回しか書き込みできない。それぞれの特性は次のとおりである（表 4.1 参照）。

表4.1　プログラム方式によるFPGA の特性分

プログラム方式	再書込	不揮発性	動作速度	冗長度
SRAM	○	×	遅い	大
EPROM	△	○	中	中
EEPROM	○	○	中	中
アンチヒューズ	×	○	速い	小

EPROM や EEPROM は、前述の PAL を再書き込み可能にするために登場したプログラム可能な ROM である。不揮発性という大きなメリットがあるが、製造に特殊なプロセスを要求するため、通常の LSI と同じプロセスで製造できる SRAM タイプに次第に移行している。

アンチヒューズタイプは再書き込みできないものの、配線をヒューズのみで構成できる。SRAM 方式では配線の接続変更はトランジスタで構成されたスイッチにより行われる。しかしトランジスタによるスイッチは、アンチヒューズに比べてスピードが遅く、動作速度をあげる上での妨げとなる。

表 4.1 の冗長度とは、プログラムするために必要とする回路の大きさのことである。SRAM は 1 ビットの情報を格納するのに、6 個のトランジスタを必要とし冗長度が大きい。EPROM、EEPROM は 1 個のトランジスタでよく、アンチヒューズタイプはトランジスタを必要としないのでもっとも冗長度が小さい。

4.4.　FPGA の設計手法

前節で述べたとおり、FPGA には、AND や OR などの論理ゲートそのものが内蔵されているわけではない。FPGA を直接設計しようとすると、LUT（Look up Table）の設定値や論理ブロック間の配線方法を通常の LSI の設計と全く異なる手法で考えなければならない。そこで、各 FPGA ベンダは、LSI での回路設計手法を用いて FPGA が設計できるように、論理ゲートレベルの回路設計を FPGA 用にマッピングするツールを開発して提供している。一般的な FPGA の設計の流れを図 4.4 に示す。回路図エントリでは、LUT や配線構造を意識することなく、通常の AND や OR、フリップフロップで回路を記述できる。HDL での設計でも、いったん通常の論理ゲートでの合成を行う。これら通常の論理ゲートの回路を各 FPGA アーキテクチャ向きに最適化した後、論理ブロックの物理位置や配線の接続を決定する。SRAM 型 FPGA の場合は、最後に SRAM へのコンフィグレーションデータを作成し、PC（Personal Computer）などを用いて FPGA 内の SRAM にダウンロードする。

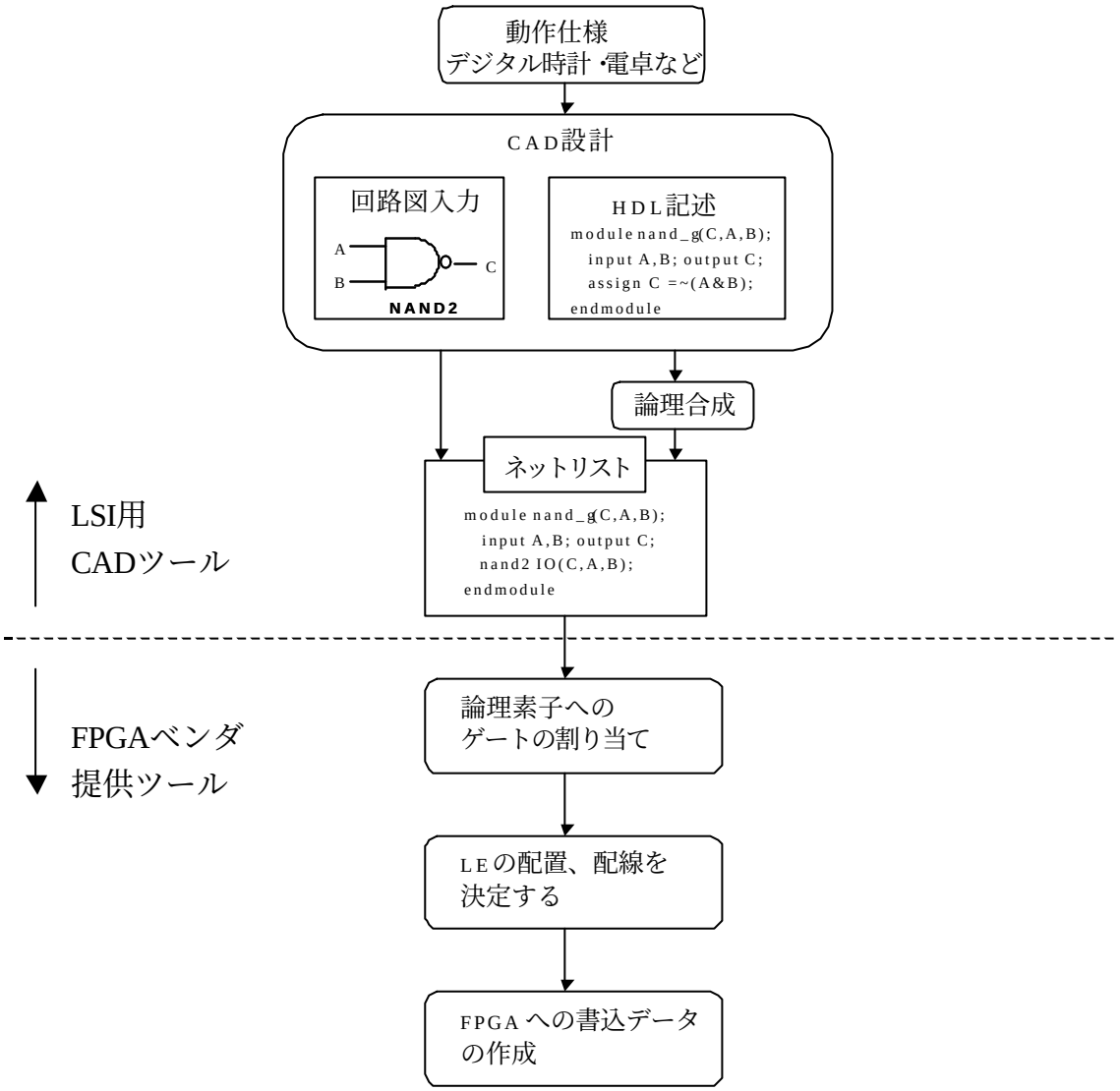

動作仕様
デジタル時計・電卓など
CAD設計
回路図入力
A
B
C
NAND2
HDL記述
module nand_g(C,A,B);
input A,B; output C;
assign C =~(A&B);
endmodule
論理合成
ネットリスト
module nand_g(C,A,B);
input A,B; output C;
nand2 IO(C,A,B);
endmodule
LSI用
CADツール
FPGAベンダ
提供ツール
論理素子への
ゲートの割り当て
LEの配置、配線を
決定する
FPGAへの書込データ
の作成

図4.4　FPGA設計法

第 5 章　VHDL による PN 符号ジェネレータの設計

この章では実際に設計した PN 符号ジェネレータと PN 符号ジェネレータを使用した CDMA 方式送信機の一例について述べる。
設計するにあたって、ハードウェア記述言語の VHDL、ゲートレベルでの回路図エディタは Electronics WORKBENCH 社のシミュレータ multiSIM を使用した。

5.1.　設計仕様

今回設計した回路は、大きく分けて２つある。

（1）PN 符号ジェネレータ
（2）CDMA 方式における送信機

それぞれについてどのような手順で設計したかについてまとめた。

5.1.1.　PN 符号ジェネレータ

今回設計した PN 符号ジェネレータは M 系列を使用した。出力される乱数は 7 ビットで表現される。設計手順は次の通りである。

①シフトレジスタの設計
②PN 符号ジェネレータの設計

①について、まず PN 符号ジェネレータを実現するためにシフトレジスタを設計した。出力は PN 符号ジェネレータと同様に 7 ビットで表せ、MSB からの

入力が右にシフトされる回路である（図 5.1 参照）。

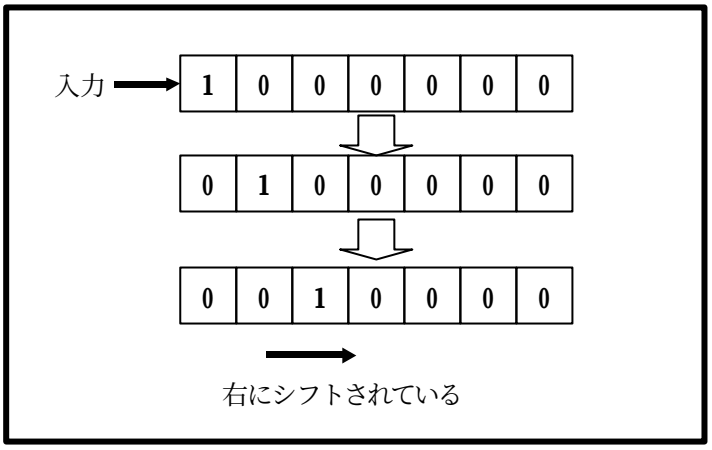


図5.1　シフトレジスタの動作

②は、①を利用して実現した。出力される乱数は擬似ランダムで表現され、127 周期をもつ回路である。帰還タップは MSB に取り付けた。
図 5.2 にブロック図を示す。

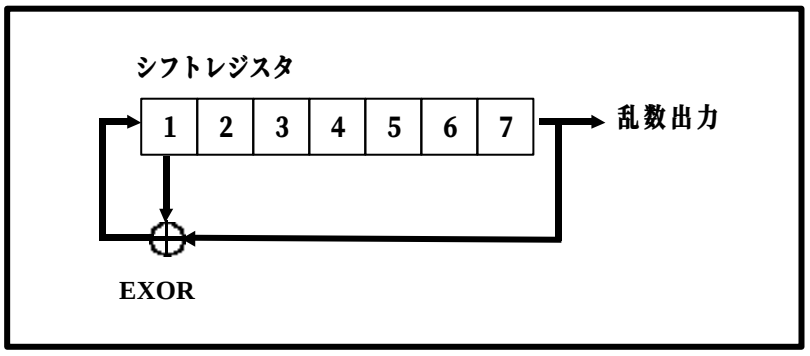


図5.2　PN符号ジェネレータブロック図

5.1.2. CDMA 方式送信機

PN 符号ジェネレータを使用して CDMA 方式における送信機の一例を設計した。それによって PN 符号がどのような役割をもっているかを確認した。例として系列長 7 の PN 符号 3 種類を用いて、データを送信する。

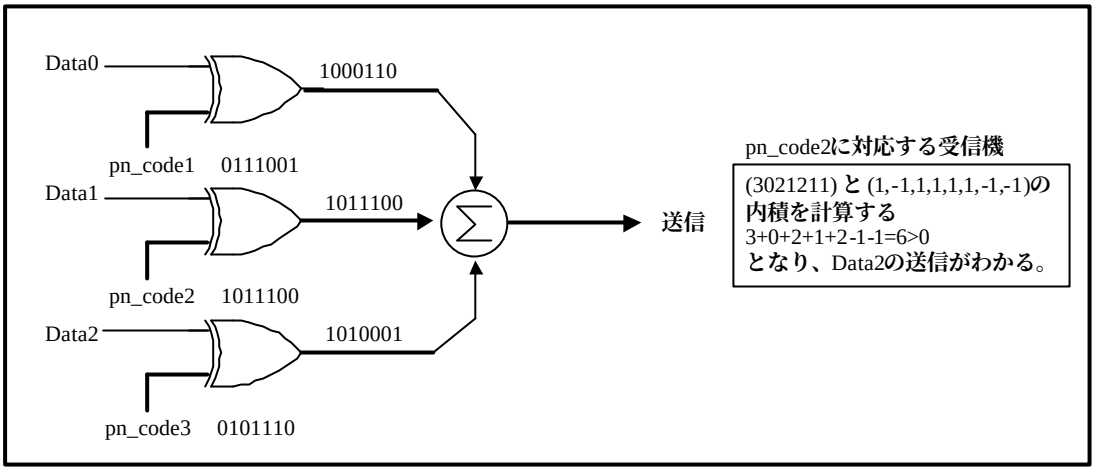


図5.3　CDMA通信の簡単な例

図 5.3 のような構成のとき、仮に

pn code1 = {0111001}
pn code2 = {1011100}
pn code3 = {0101110}

の 3 種類の符号を用いて、それぞれの符号に対して Data0 に「1」、Data2 に「0」、Data3 に「1」を送信する場合を考える。各 pn code は Data と EXOR（排他的論理和）され、Data が「1」のときには各 pn 符号は反転する。

これらの符号を加算し、結果的に送信機には｛3021211｝なる信号が送信される。受信機では｛3021211｝なる信号が受信され、それと pn code2 との相関をとる。具体的には pn code の中の「0」を「-1」と読み直して、系列をベクトルと考え、内積をとる。この結果が、正の大きめの値であれば、pn code2 で送られた送信データは「0」、負の大きめの値であれば、送信データは「1」と判断される。このようにして CDMA 方式では同一の周波数間においてチャネルの振り分けを行っている。

図 5.3 の通信例を設計するにあたって、まず 3 入力の加算、つまり全加算器を設計した。全加算器を設計する前に半加算器から設計した。また PN 符号ジェネレータを使用し、他に 2 つの異なる符合を割り当てる PN 符号ジェネレータを設計した。それぞれのブロック図を図 5.4 に示す。

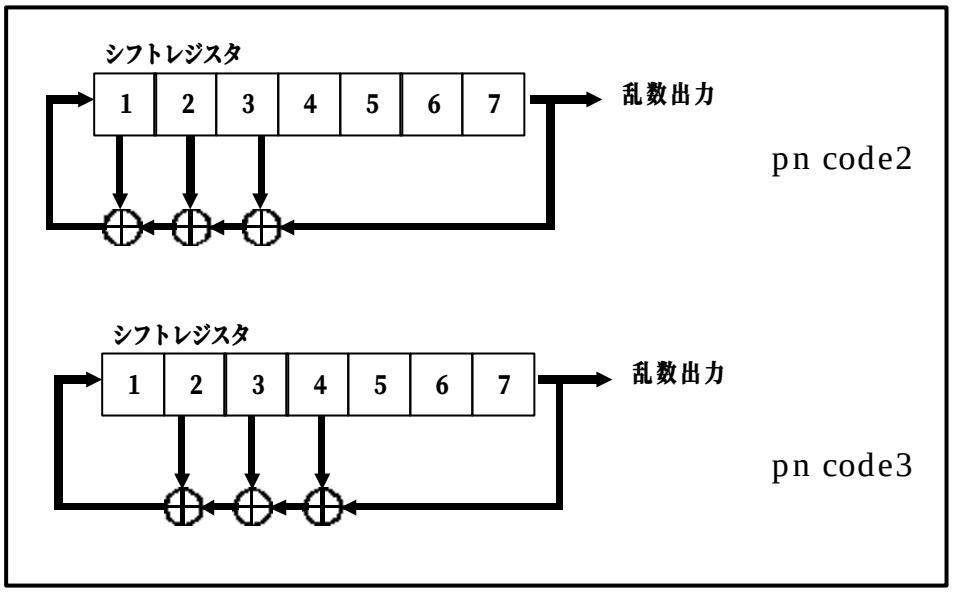


図5.4　pn code2、pn code3

帰還タップの位置は pn code2 では図の第 1、2、3 ビットに、pn code3 では第 2、3、4 ビットの部分に設定した。

5.2. PN 符号ジェネレータの設計

5.2.1. シフトレジスタの設計

設計仕様に基づいてシフトレジスタを設計した。VHDL 言語にて設計するにあたって、まずそれぞれの信号を定義した（表 5.1 参照）。また使用ツール（コンポーネント）は D フリップフロップを用いた。

表5.1　信号の定義

信号	動作仕様
1	MSBに「1」を入力
CLK	クロック
OUT1	出力7ビット目（MSB）
OUT2	出力6ビット目
OUT3	出力5ビット目
OUT4	出力4ビット目
OUT5	出力3ビット目
OUT6	出力2ビット目
OUT7	出力1ビット目（LSB）

まず入出力信号から VHDL のソースコードを生成した（図 5.5）。

```
library ieee;
USE ieee.std_logic_1164.all;
entity shifter is
port (
   1 : in std_logic ;
   CLK : in std_logic ;
  OUT1 : out std_logic ;
  OUT2 : out std_logic ;
  OUT3 : out std_logic ;
  OUT4 : out std_logic ;
  OUT5 : out std_logic ;
  OUT6 : out std_logic ;
  OUT7 : out std_logic );
end shifter;
architecture structural of shifter is
component FD
port (
   Q : out std_logic ;
   C : in std_logic ;
   D : in std_logic );
end component ;
signal a_1 : std_logic;
signal a_CLK : std_logic;
signal a_OUT1 : std_logic;
signal a_OUT2 : std_logic;
signal a_OUT3 : std_logic;
signal a_OUT4 : std_logic;
signal a_OUT5 : std_logic;
signal a_OUT6 : std_logic;
signal a_OUT7 : std_logic;
begin
  I0 : FD
     port map(
       Q => a_OUT1, C => a_CLK, D => a_1);
  I1 : FD
     port map(
       Q => a_OUT7, C => a_CLK, D => a_OUT6);
  I2 : FD
     port map(
       Q => a_OUT6, C => a_CLK, D => a_OUT5);
  I3 : FD
     port map(
       Q => a_OUT5, C => a_CLK, D => a_OUT4);
  I4 : FD
     port map(
       Q => a_OUT4, C => a_CLK, D => a_OUT3);
  I5 : FD
     port map(
       Q => a_OUT3, C => a_CLK, D => a_OUT2);
  I6 : FD
     port map(
       Q => a_OUT2, C => a_CLK, D => a_OUT1);
  a_1 <= 1;
  a_CLK <= CLK;
  OUT1 <= a_OUT1;
  OUT2 <= a_OUT2;
  OUT3 <= a_OUT3;
  OUT4 <= a_OUT4;
  OUT5 <= a_OUT5;
  OUT6 <= a_OUT6;
  OUT7 <= a_OUT7;
  end structural;
```

図5.5　シフトレジスタのVHDLソース

次に VHDL から回路（ゲートレベル）設計を行った。ゲートレベルで使用したツールを表 5.2 に示す。

表5.2　使用ツール

Component	Manufacture List	ID List
ExOR	Texas Instruments	L1-7486
D/FF	Texas Instruments	L1-7474
CLK	generic	L0-CLK
DC-Voltage Source	generic	L0-VDC
Ground	generic	L0-GRD

回路図、シミュレーション結果を図 5.6、5.7 に示す。

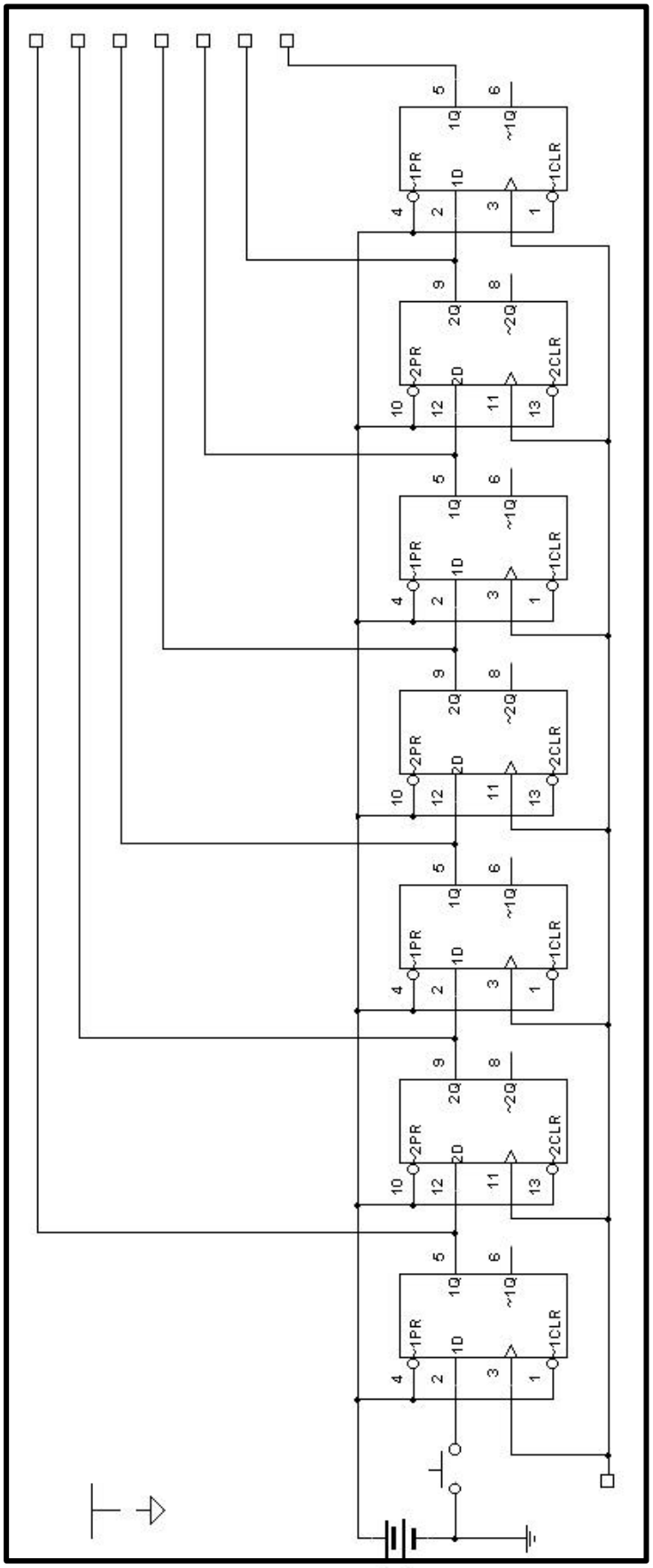


図5.6　シフトレジスタの回路図

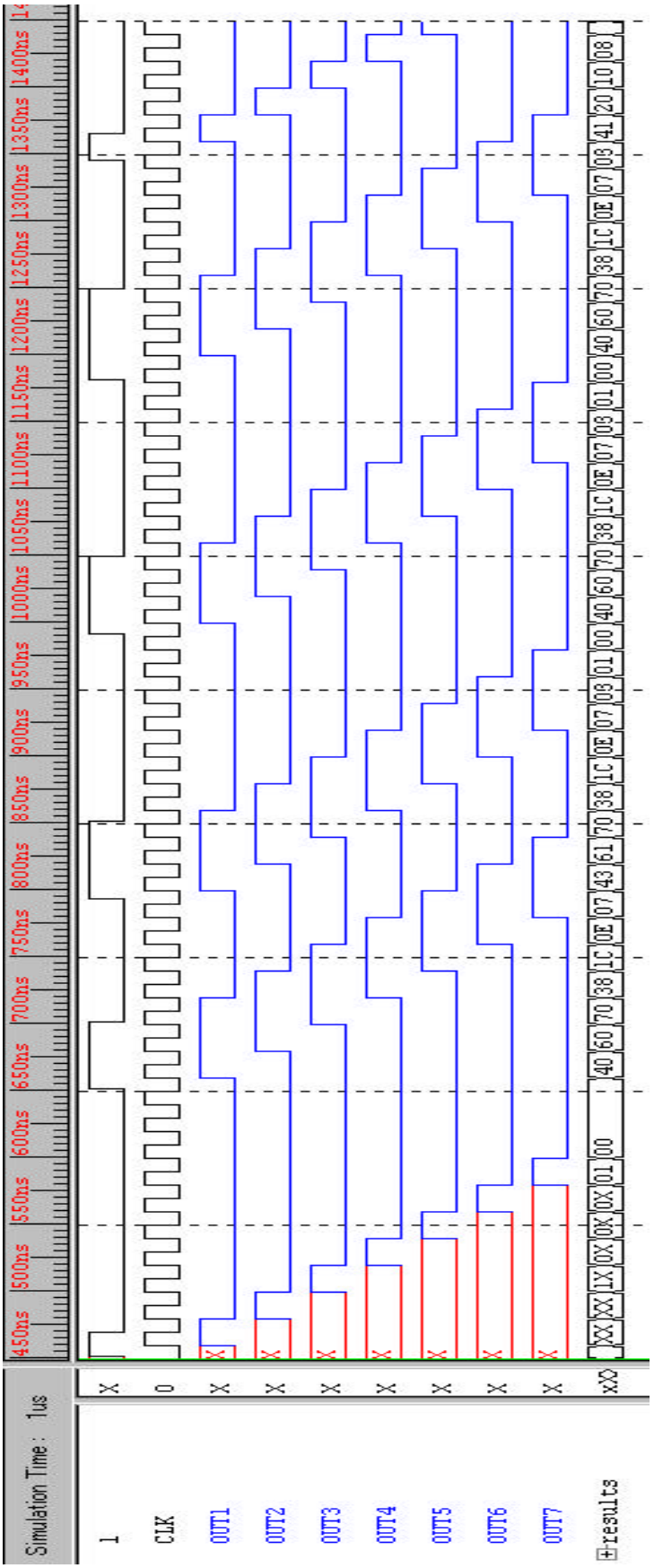


図5.7　シフトレジスタのシミュレーション結果

図 5.7 のシミュレーション結果から、入力データがシフトされている様子が確認できた。

5.2.2. PN 符号ジェネレータの設計

前節にて使用したシフトレジスタを使用して PN 符号ジェネレータを設計した。各信号の定義を表 5.3 に示す。また VHDL のソースを図 5.8 に示す。

表5.3　信号の定義

信号	動作仕様
CLK	クロック
OUT1	出力7ビット目（MSB）
OUT2	出力6ビット目
OUT3	出力5ビット目
OUT4	出力4ビット目
OUT5	出力3ビット目
OUT6	出力2ビット目
OUT7	出力1ビット目（LSB）

次にゲートベルでの設計を行った。使用したツールは表 5.2 と同じである。回路図とシミュレーション結果を図 5.9、5.10 に示す。

```
library ieee;
USE ieee.std_logic_1164.all;
entity pn_code1 is
port (
  CLK : in std_logic ;
  Q1 : out std_logic ;
  Q2 : out std_logic ;
  Q3 : out std_logic ;
  Q4 : out std_logic ;
  Q5 : out std_logic ;
  Q6 : out std_logic ;
  Q7 : out std_logic );
end pn_code1;
architecture structural of pn_code1 is
component IFD
 port (
  D : in std_logic ;
  Q : out std_logic ;
  C : in std_logic );
end component ;
component XOR2
 port (
  O : out std_logic ;
  I1 : in std_logic ;
  I0 : in std_logic );
end component ;
signal a_CLK : std_logic;
signal a_Q1 : std_logic;
signal a_Q2 : std_logic;
signal a_Q3 : std_logic;
signal a_Q4 : std_logic;
signal a_Q5 : std_logic;
signal a_Q6 : std_logic;
signal a_Q7 : std_logic;
signal I7_O : std_logic;
begin
  I0 : IFD
  port map(
   D => a_Q3, Q => a_Q4, C => a_CLK);
  I1 : IFD
  port map(
   D => a_Q2, Q => a_Q3, C => a_CLK);
  I2 : IFD
  port map(
   D => a_Q1, Q => a_Q2, C => a_CLK);
  I3 : IFD
  port map(
   D => a_Q4, Q => a_Q5, C => a_CLK);
  I4 : IFD
  port map(
   D => I7_O, Q => a_Q1, C => a_CLK);
  I5 : IFD
  port map(
   D => a_Q5, Q => a_Q6, C => a_CLK);
  I6 : IFD
  port map(
   D => a_Q6, Q => a_Q7, C => a_CLK);
  I7 : XOR2
  port map(
   O => I7_O, I1 => a_Q3, I0 => a_Q7);
a_CLK <= CLK;
Q1 <= a_Q1;
Q2 <= a_Q2;
Q3 <= a_Q3;
Q4 <= a_Q4;
Q5 <= a_Q5;
Q6 <= a_Q6;
Q7 <= a_Q7;
end structural;
```

図5.8　PN符号ジェネレータのVHDLソース

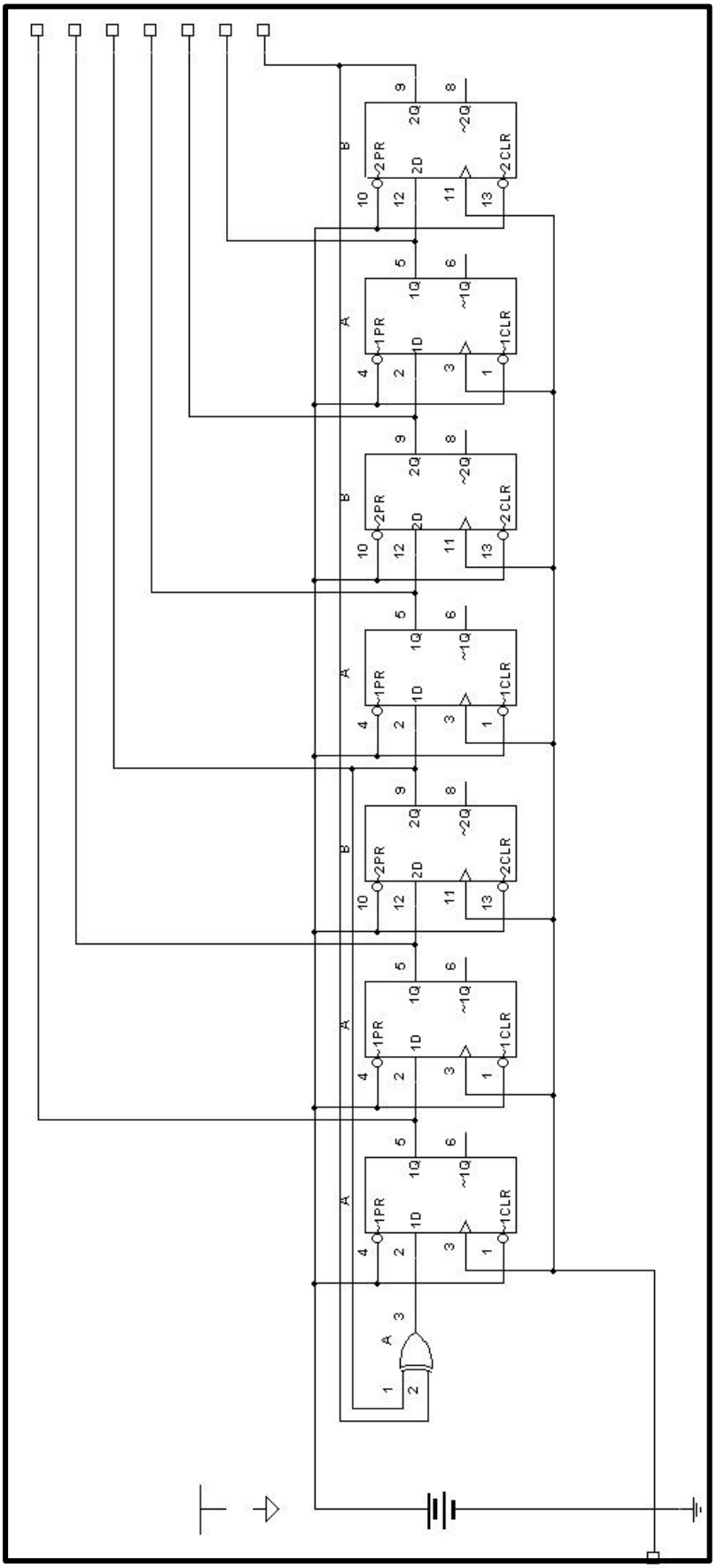


図5.9　PN符号ジェネレータの回路図

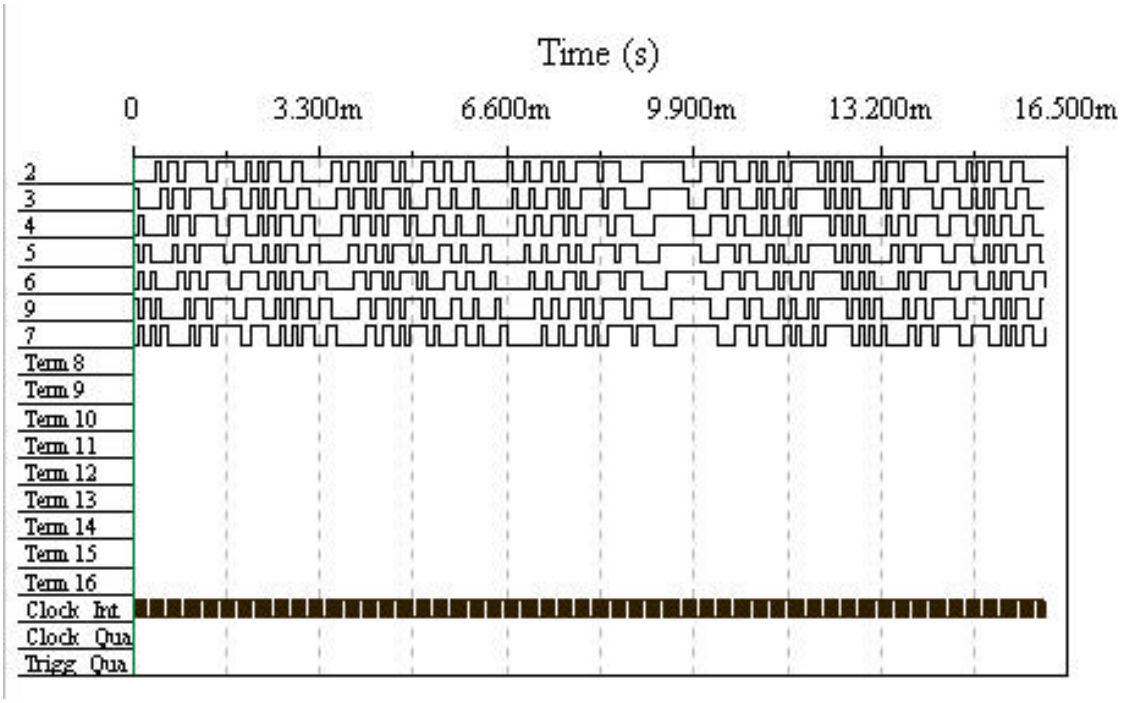


図5.10　PN符号ジェネレータのシミュレーション結果

図 5.10 における信号波形 2、3、4、5･･･7 は VHDL 記述で定義した OUT1~OUT7 に対応している。

このシミュレーション結果から、擬似ランダムにデータが出力されていることがわかる。

5.2.3. PN 符号ジェネレータの FPGA 化

前節にて PC 上で設計した回路を、実際にチップ化して回路の出力波形を確認した。チップ化にあたっては FPGA を使用し、出力の確認はオシロスコープを使用した。

使用器具の型番、メーカ名を表 5.4 に示す。

表5.4　使用器具の型番、メーカ名

製品名	型番	メーカ名
ISPプログラマ&チェッカ	ISP-01	aritec
MAX7000シリーズFPGAチップ	EPM7064SLC44-10	Altera
オシロスコープ	54645D	Agilent Technologies

FPGA への書き込みは Altera 社のチップを使用し（表 5.5 参照）、ソフトウェアとして Max+PlusⅡを使用した。Max+PlusⅡの書き込み作業をまとめると、

回路図エディタ（ゲートレベル）による回路設計
設計回路のシミュレーション
FPGA 書き込みのためのピン配置
コンパイル
FPGA への書き込み（プログラム）

となっている。

表5.5　7064SLCのピン番号、信号

ピン番号	信号名	ピン番号	信号名
1	INPUT/GCLRn	23	VcclNT
2	INP/OE2GCLK2	24	I/O
3	VcclNT	25	I/O
4	I/O	26	I/O
5	I/O	27	I/O
6	I/O	28	I/O
7	TDI	29	I/O
8	I/O	30	GND
9	I/O	31	I/O
10	GND	32	TCK
11	I/O	33	I/O
12	I/O	34	I/O
13	TMS	35	VcclNT
14	I/O	36	I/O
15	VcclNT	37	I/O
16	I/O	38	TDO
17	I/O	39	I/O
18	I/O	40	I/O
19	I/O	41	I/O
20	I/O	42	GND
21	I/O	43	INPUT/GCLK1
22	GND	44	INP/OE1

ゲートレベルでの回路設計は、図 5.9 の回路をそのまま採用した。シミュレーション結果については 1 クロックの周期を 100ns（10MHz）として、1us までの測定した。測定結果を図 5.11 に示す。

図5.11　Max+PlusⅡによるシミュレーション結果

測定範囲が狭いために、ランダムに出力されているかどうかは確認できないが、出力がシフトされているのは測定結果から確認できる（Max+PlusⅡではここまでしか測定できない。）。

次に設計した回路の入出力に対してピン配置を行った。FPGA 書き込み後の動作確認のために出力OUT1~OUT7をLEDに接続された出力ピンに設定した。具体的なピンの振り分けを図 5.12 に示す。

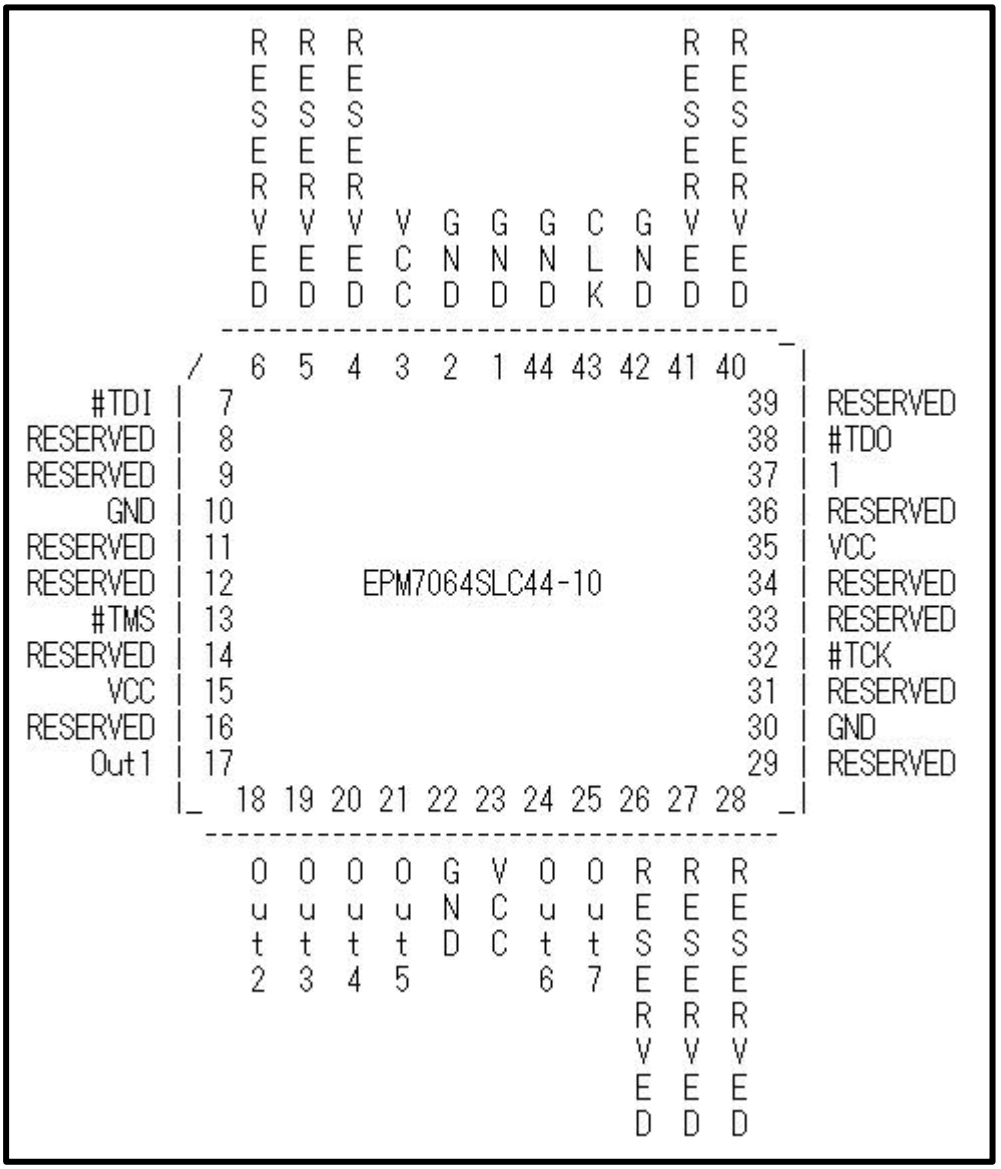


図5.12　FPGAチップへのピン配置

FPGA へのピンを定義した上で、設計した回路のデータを全てコンパイルし、評価用ボードのチップにダウンロードした。評価用のボードとチップを図 5.13 に示す。

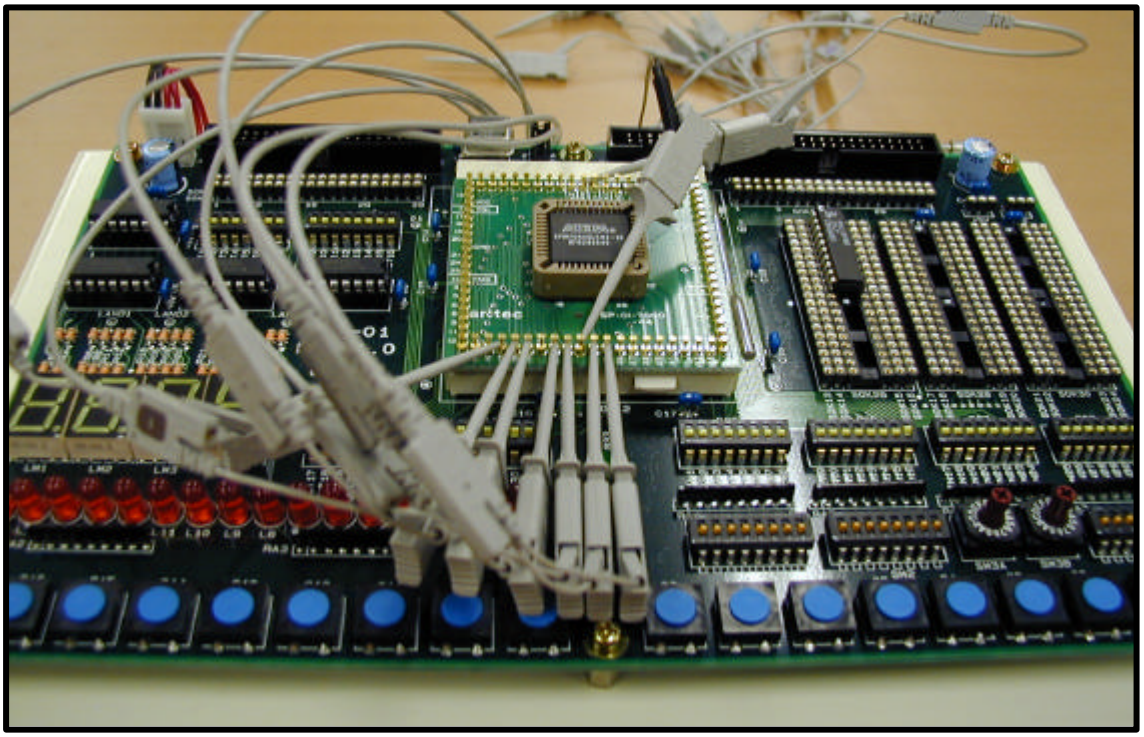


図5.13　評価用ボード(ISP-01)

図 5.12 のボードのクロック周波数は 8MHz と高い周波数をなっているので、常に LED が点灯したままの状態となっており、ランダムな出力を肉眼で確認できない。このためオシロスコープのプローブにクロックおよび OUT1~OUT7 を接続し、8 チャンネルの同期測定を行った。オシロスコープの出力画面を図 5.14 に示す。

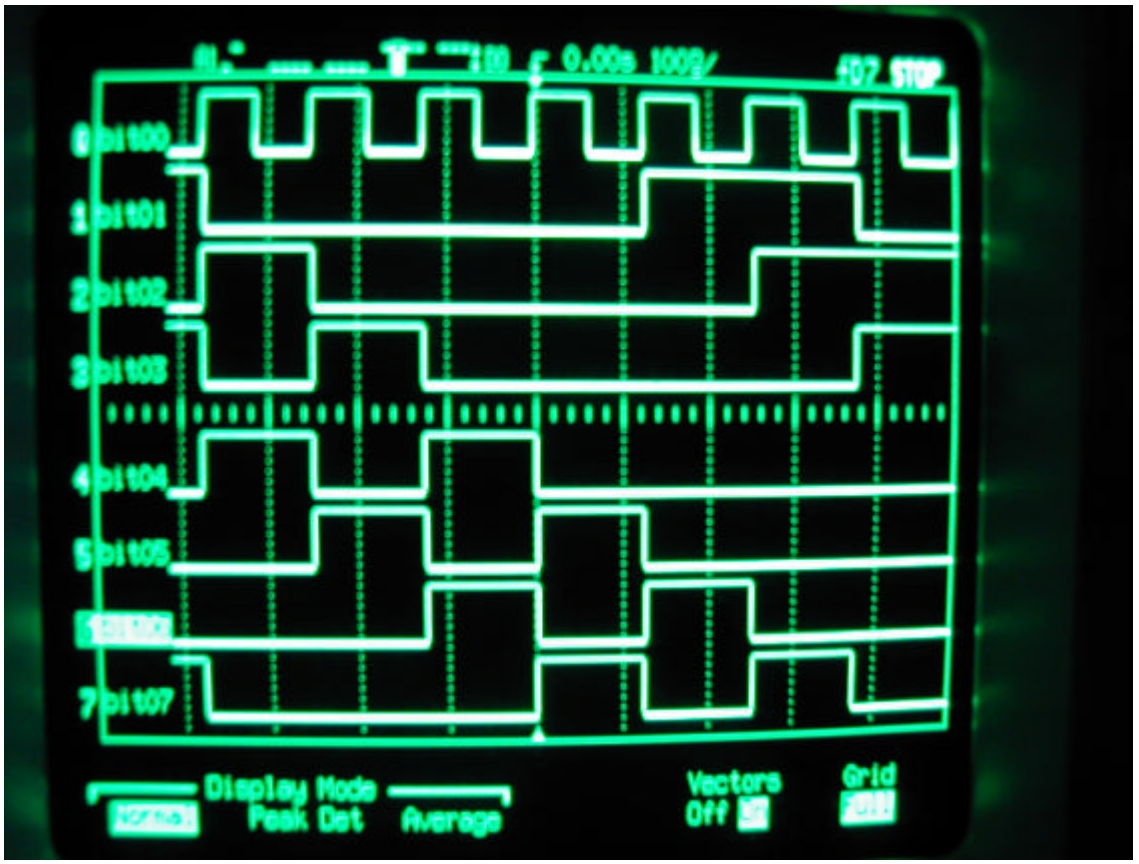


図5.14　オシロスコープによる出力波

上図の写真は出力状態が常に遷移している画面の一部分を撮影したものであるが、擬似ランダムの出力が発生している様子がうかがえる。

5.3.　CDMA 方式送信機の設計

5.1.2 節で述べた仕様に基づいて CDMA 方式送信機の一例を設計した。

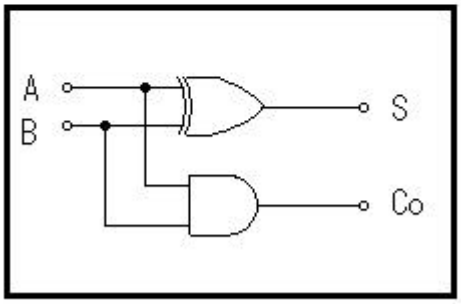
5.3.1.　半加算器、全加算器の設計

まず入力 3 つの加算、つまり全加算器を設計した。全加算器を設計する前に半加算器から設計した。半加算器の真理値表、VHDL ソースコード、回路図、

シミュレーション結果を図 5.15 に示す。

真理値表

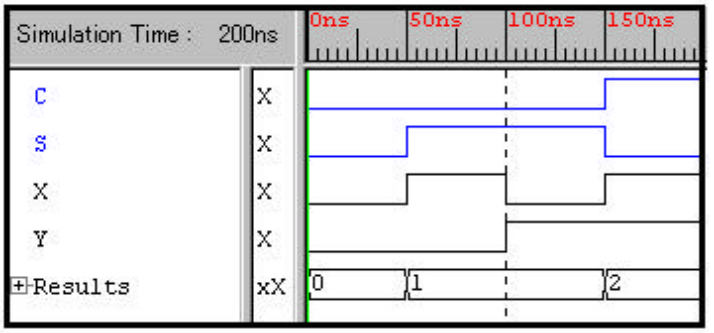
A	B	S	Co
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



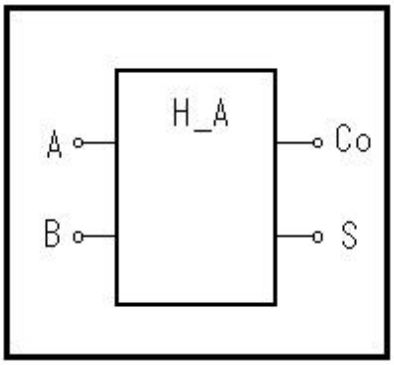
回路図

```
library ieee;
use ieee.std_logic_1164.all;
entity HALF_ADDER is
    port (
        a: in std_logic;
        b: in std_logic;
        s: out std_logic;
        co: out std_logic  );
end HALF_ADDER;
architecture BEHAVIOR of HALF_ADDER is
 signal w0,w1,w2: std_logic;
begin
 w0 <= a and b;
 w1 <= not w0;
 w2 <= a or b;
 s <= w1 and w2;
 co <= w0;
end BEHAVIOR;
```

VHDLソースコード



シミュレーション結果



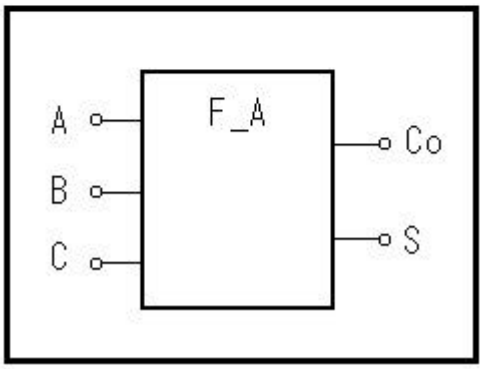
ブロック図

図5.15　半加算器の真理値表、VHDLソース、回路図、タイミング測定、ブロック図

図 5.15 から全加算器を設計する。全加算器の真理値表、VHDL ソースコード、回路図、シミュレーション結果を図 5.16 に示す。

真理値表

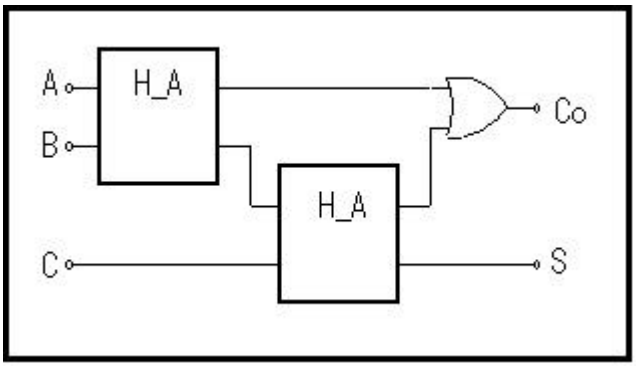
A	B	C	S	Co
0	0	0	0	0
0	0	1	1	0
0	1	1	0	1
0	1	0	1	0
1	1	0	0	1
1	1	1	1	1
1	0	1	0	1
1	0	0	1	0



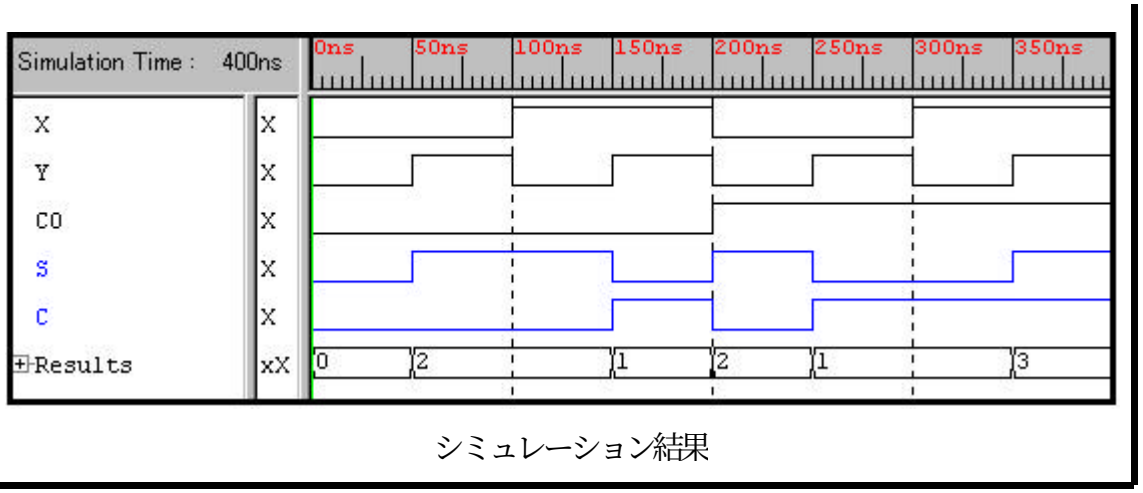
ブロック図

```
library ieee;
use ieee.std_logic_1164.all;

entity FULL_ADDER is
    port (
        a: in std_logic;
        b: in std_logic;
        ci: in std_logic;
        s: out std_logic;
        co: out std_logic
    );

end FULL_ADDER;
architecture BEHAVIOR of FULL_ADDER is
  component HALF_ADDER
    port (
      a, b: in std_logic;
      s, co: out std_logic);
  end component;
  signal w0, w1, w2: std_logic;
begin
  co <= w1 or w2;
  i0: HALF_ADDER port map (
      co=>w1,s=>w0,a=>a,b=>b);
  i1: HALF_ADDER port map (
      co=>w2,s=>s,a=>w0,b=>ci);
end BEHAVIOR;
```

VHDLソースコード



回路図



シミュレーション結果

図5.16　全加算器の真理値表、VHDLソース、回路図、タイミング測定、ブロック図

5.3.2.　PN 符号ジェネレータの設計

前節で設計した PN 符号ジェネレータを変形して、他に 2 つ異なる符合を割り当てる PN 符号ジェネレータを設計する。手順としては 5.2.2 節の内容とほぼ同じなので、簡単にまとめた。

5.1.1 節の設計仕様から pn_code２および pn_code3 を設計する。それぞれについて信号を定義するだが、pn_code２および pn_code3 は回路の配線が異なるものの、信号は同じなので、表 5.6 にまとめる。

表5.6　信号の定義

信号	動作仕様
CLK	クロック
OUT1	出力7ビット目（MSB）
OUT2	出力6ビット目
OUT3	出力5ビット目
OUT4	出力4ビット目
OUT5	出力3ビット目
OUT6	出力2ビット目
OUT7	出力1ビット目（LSB）

入出力信号から pn_code２および pn_code3 について VHDL ソースコードを記述した（図 5.17、図 5.18 参照）。

```
library ieee;
USE ieee.std_logic_1164.all;
entity pn_code2 is
port (
 CLK : in std_logic ;
 Q1 : out std_logic ;
 Q2 : out std_logic ;
 Q3 : out std_logic ;
 Q4 : out std_logic ;
 Q5 : out std_logic ;
 Q6 : out std_logic ;
 Q7 : out std_logic );
end pn_code2;
architecture structural of pn_code2 is
component IFD
 port (
 D : in std_logic ;
 Q : out std_logic ;
 C : in std_logic );
end component ;
component XOR2
 port (
  O : out std_logic ;
  I1 : in std_logic ;
  I0 : in std_logic );
end component ;
signal a_CLK : std_logic;
signal a_Q1 : std_logic;
signal a_Q2 : std_logic;
signal a_Q3 : std_logic;
signal a_Q4 : std_logic;
signal a_Q5 : std_logic;
signal a_Q6 : std_logic;
 signal a_Q7 : std_logic;
 signal I7_O : std_logic;
 signal I8_O : std_logic;
 signal I9_O : std_logic;
begin I0 : IFD
 port map(D => a_Q3, Q => a_Q4, C => a_CLK);
 I1 : IFD
 port map(D => a_Q2, Q => a_Q3, C => a_CLK);
 I2 : IFD
 port map(D => a_Q1, Q => a_Q2, C => a_CLK);
 I3 : IFD
 port map(D => a_Q4, Q => a_Q5, C => a_CLK);
 I4 : IFD
 port map(D => I7_O, Q => a_Q1, C => a_CLK);
 I5 : IFD
 port map(D => a_Q5, Q => a_Q6, C => a_CLK);
 I6 : IFD
 port map(D => a_Q6, Q => a_Q7, C => a_CLK);
 I7 : XOR2
 port map(O => I7_O, I1=> I8_O, I0=> I9_O);
 I8 : XOR2
 port map(O => I8_O, I1=> a_Q1, I0=> a_Q2);
 I9 : XOR2
 port map(O => I9_O, I1=> a_Q3, I0=> a_Q7);
a_CLK <= CLK;
Q1 <= a_Q1;
Q2 <= a_Q2;
Q3 <= a_Q3;
Q4 <= a_Q4;
Q5 <= a_Q5;
Q6 <= a_Q6;
Q7 <= a_Q7;
end structural;
```

図5.17　pn code2のVHDLソース

```
library ieee;
USE ieee.std_logic_1164.all;
entity pn_code3 is
port (
 CLK : in std_logic ;
 Q1 : out std_logic ;
 Q2 : out std_logic ;
 Q3 : out std_logic ;
 Q4 : out std_logic ;
 Q5 : out std_logic ;
 Q6 : out std_logic ;
 Q7 : out std_logic );
end pn_code3;
architecture structural of pn_code3 is
component IFD
 port (
  D : in std_logic ;
  Q : out std_logic ;
  C : in std_logic );
end component ;
component XOR2
 port (
  O : out std_logic ;
  I1 : in std_logic ;
  I0 : in std_logic );
end component ;
signal a_CLK : std_logic;
signal a_Q1 : std_logic;
signal a_Q2 : std_logic;
signal a_Q3 : std_logic;
signal a_Q4 : std_logic;
signal a_Q5 : std_logic;
signal a_Q6 : std_logic;
signal a_Q7 : std_logic;
signal I7_O : std_logic;
signal I8_O : std_logic;
signal I9_O : std_logic;
begin I0 : IFD
  port map(D => a_Q3, Q => a_Q4, C => a_CLK);
 I1 : IFD
  port map(D => a_Q2, Q => a_Q3, C => a_CLK);
 I2 : IFD
  port map(D => a_Q1, Q => a_Q2, C => a_CLK);
 I3 : IFD
  port map(D => a_Q4, Q => a_Q5, C => a_CLK);
 I4 : IFD
  port map(D => I7_O, Q => a_Q1, C => a_CLK);
 I5 : IFD
  port map(D => a_Q5, Q => a_Q6, C => a_CLK);
 I6 : IFD
  port map(D => a_Q6, Q => a_Q7, C => a_CLK);
 I7 : XOR2
  port map(O => I7_O, I1=> I8_O, I0=> I9_O);
 I8 : XOR2
  port map(O => I8_O, I1=> a_Q2, I0=> a_Q3);
 I9 : XOR2
  port map(O => I9_O, I1=> a_Q4, I0=> a_Q);
a_CLK <= CLK;
Q1 <= a_Q1;
Q2 <= a_Q2;
Q3 <= a_Q3;
Q4 <= a_Q4;
Q5 <= a_Q5;
Q6 <= a_Q6;
Q7 <= a_Q7;
end structural;
```

図5.18　pn code3のVHDLソース

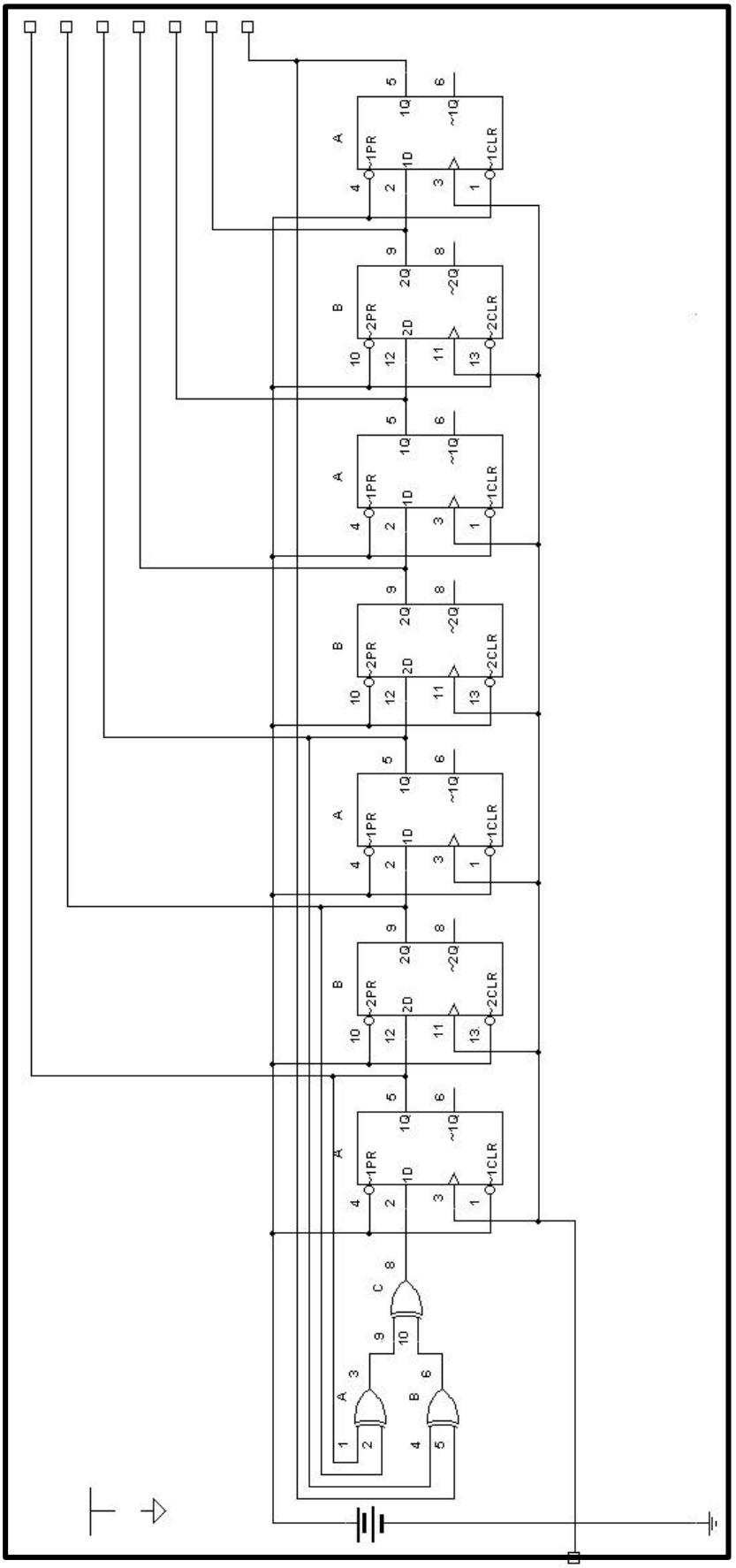


図5.19 pn_code2

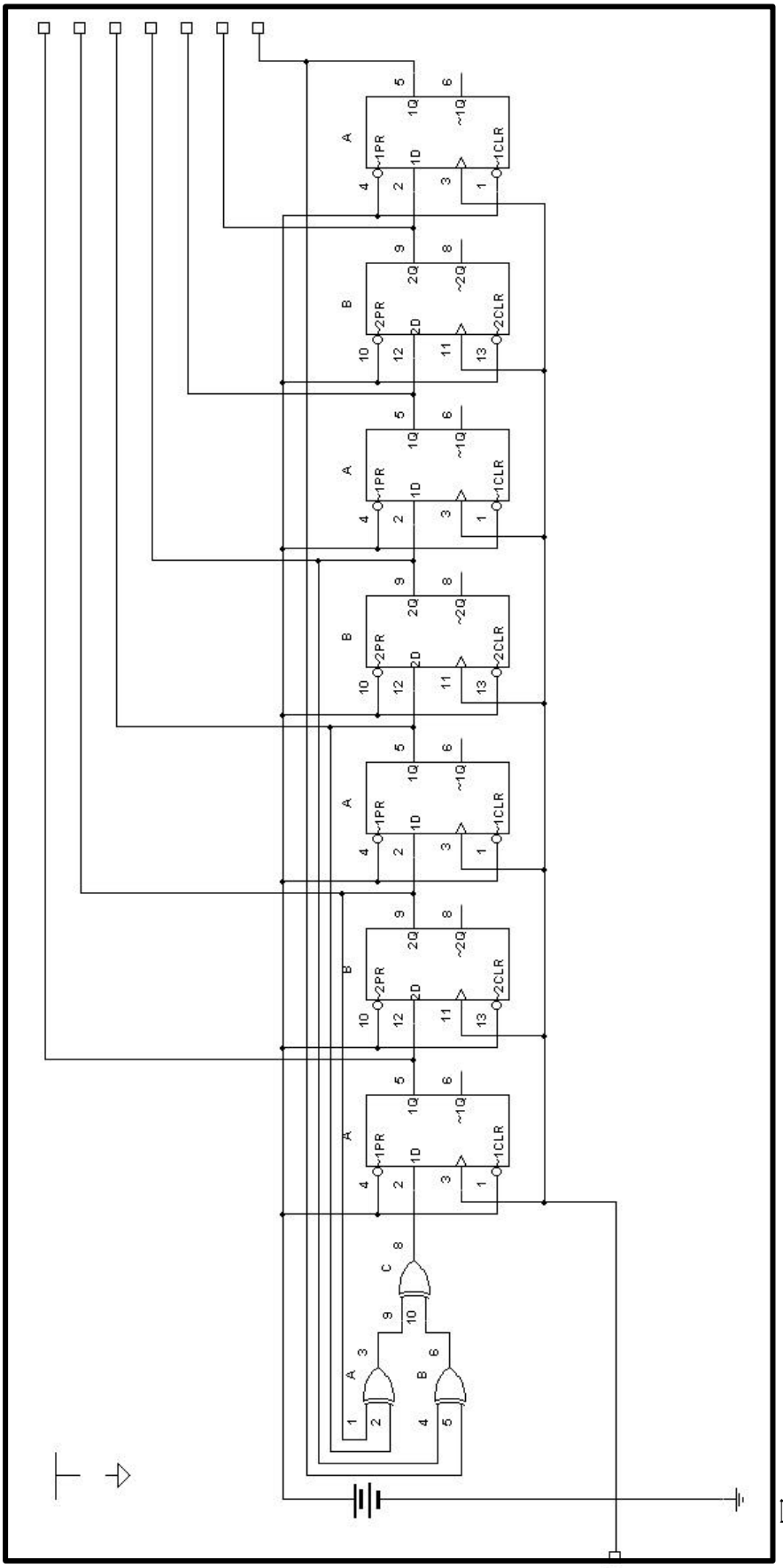


図5.20 pn_code3

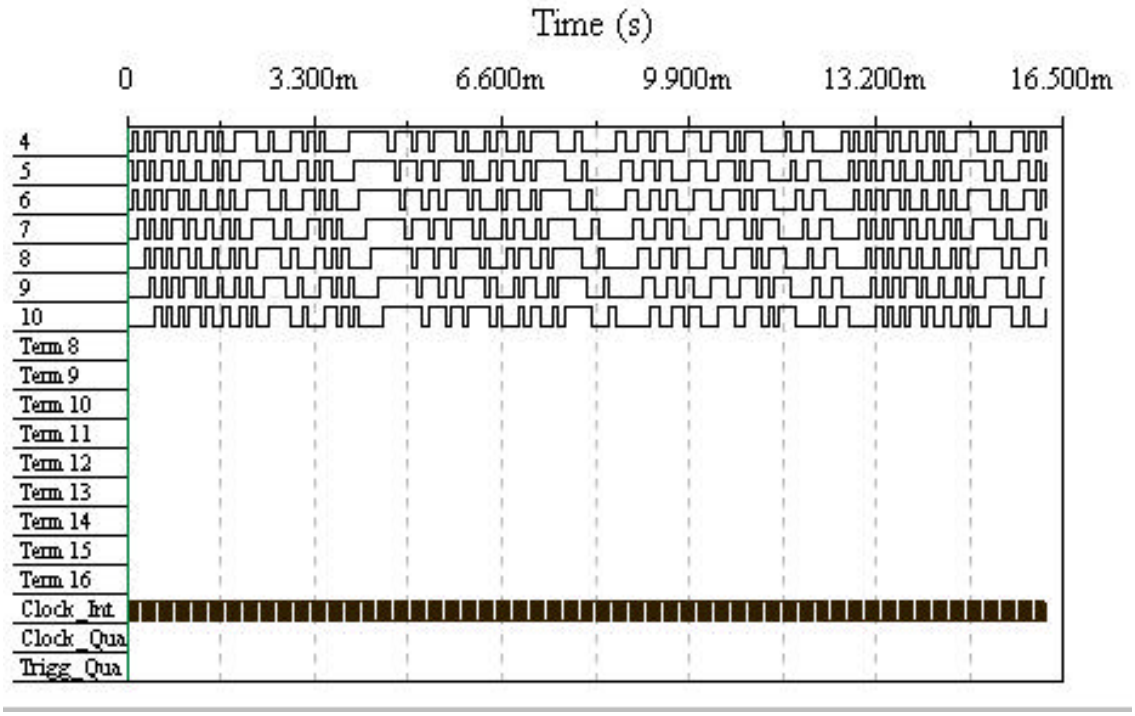


図5.21　pn code2のタイミング測定

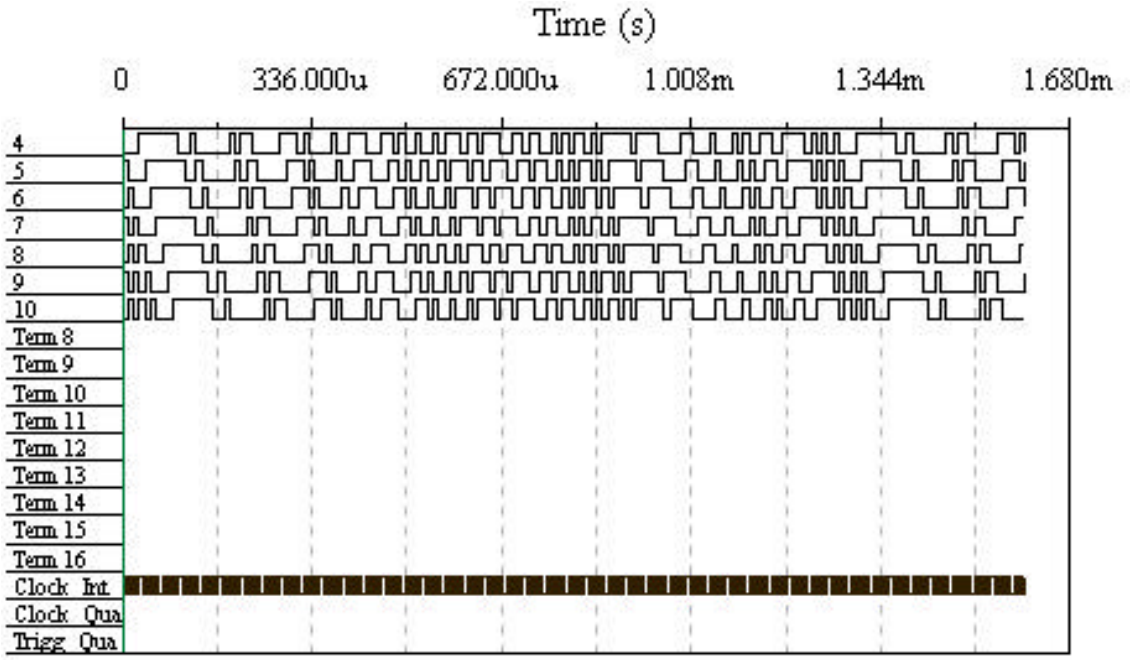


図5.22　pn code3のタイミング測定

次にゲートレベルでの設計を行った。使用したツールは表 5.2 と同じである。pn_code２および pn_code3 の回路図、シミュレーション結果を図 5.19~5.22 に示す。

5.3.3. CDMA 方式送信機の設計

前節までに設計した回路を組み合わせて、図 5.3 の通信例を実現した。その回路図を図 5.22 に示す。なお出力は、OUT1~OUT14 までの 14 ビットを出力している。本来は送信される際、1 本の線（1 ビット）で送信するのだが、出力波形を測定するため 14 ビットの出力とした。

なおシミュレーションは、5.1.2 節で述べたように Data0 に「1」、Data1 に「0」、Data2 に「1」を入力してシミュレーションした。しかし結果は得られず、出力波形は得られなかった。

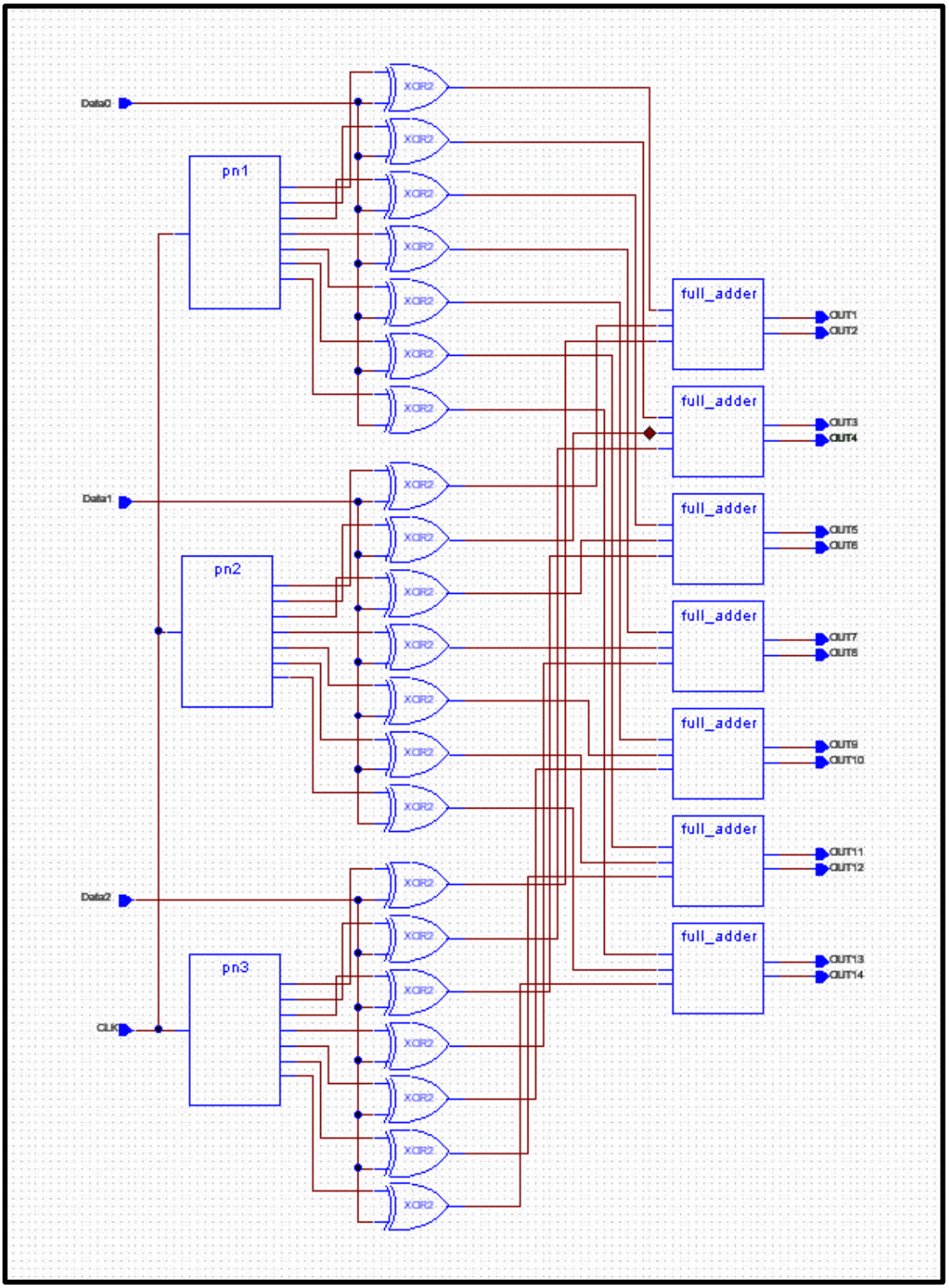


図5.23　CDMA方式の送信機部分の回路図

第 6 章　むすび

全体を通して、本卒研の目的となっている LSI の設計手順については理解を深めることができた。VHDL 設計については、さまざまな回路のモジュールを記述できるようになっただけでなく、テストベンチによる記述を用いてシミュレーションできるようになった。論理設計にあたっては順序同期回路におけるプリセット、クリアの接続方法など、実際に IC で回路を組む上での設計技術を習得できた。また設計した回路を FPGA へ書き込むことにで、FPGA による LSI の開発手順を理解できた。FPGA チップの評価では、オシロスコープやロジックアナライザなどの測定器の使用法を習得した。

無線通信技術については、CDMA 方式におけるスペクトル拡散技術のための DS 方式や PN 符号系列の基本原理について十分理解した。また CDMA 方式だけでなく、FDMA 方式や TDMA 方式など無線通信のバックグラウンドについても理解を深めることができた。

反省点

卒業研究として何とか提出できる段階にまで至ったものの、反省点はいくつかある。簡単に列挙すると、

1. スケジュール調整を怠っていた。
2. CAD ソフトを十分使いこなせなかった。
3. 論文作成にずいぶん時間がかかった。

がある。スケジュールを十分に設定していなかったために、ずいぶん無駄に時間を費やすことが多かった。また CAD ソフトを使用する際に、自分の不手際によって回路がうまく動作しないことがあった。また論文の文書作成では何度も手直しを入れたため、修正にずいぶん時間がかかった。今後社会へ出て、さまざまなプロジェクトを体験することになるが、こういった点を十分考慮反省した上で、取り組んでいきたい。

謝辞

本研究を行うに際し、終始、懇切丁寧な御指導を御鞭撻賜りました高知工科大学工学部電子･光システム工学科矢野政顯教授に心から感謝いたします。

研究中、懇切丁寧な御指導を賜りました高知工科大学工学部電子･光システム工学科学科長、原 央教授ならびに河津 哲教授、橘 昌良助教授に厚くお礼申し上げます。

また終始適切な御助言、御助力をいただきました高知工科大学大学院基盤工学専攻情報システムコース前期修士課程坂下雄一氏、高知工科大学工学部電子･光システム工学科学部生、中村基継氏、ほか矢野研究室の皆様に心から感謝し、お礼申し上げます。

参考文献

[1] 田丸啓吉：“論理回路の基礎”、工学図書株式会社、PP92～102（1988）

[2] 中村次男：“ディジタル回路設計法”、日本理工出版会、PP38～76（1990）

[3] 高橋寛、関根好文、作田幸憲：“ディジタル回路”、コロナ社、PP75～148（1996）

[4] 深山正幸、北川章夫、秋田純一、鈴木正國：“HDL による VLSI 設計”、共立出版株式会社、PP16～85（1999）

[5] 長谷川裕恭：“VHDL によるハードウェア設計入門”、CQ 出版社、PP59～79（1995）

[6] Z.ナバビ：“VHDL の基礎”、日経 BP 出版センター、PP51～70（1994）

[7] 山内雪路：“スペクトル拡散通信”、東京電機大学出版局、PP43～102（1994）

[8] 浅田邦博：“ディジタル集積回路の設計と試作”、培風館、PP7～17（2000）

[9] 柴山潔：“コンピュータアーキテクチャの基礎”、近代科学社、PP102～107（1993）

[10] John W.Carter：“Digital Designing with Programmable Logic Devices”、PRENTICE HALL、PP47～70（1997）

[11] Jim Watson、Ken Chapman、Andy Miller、白土神一、荒井雅：“FPGA で CDMA 電話機を実現する”、Design Wave Magazine、2000 年 5 月号、CQ 出版、PP35～70

[12] CDMAwow：
http://www-comm.cs.shinshu-u.ac.jp/~gfujip/cdma/index.html

[13] NTT DoCoMo R&D：
http://www.nttdocomo.co.jp/corporate/rd/homepage.html