卒 業 研 究 報 告

題　　目

同期式順序回路の設計

指 導 教 員

矢野 政顕 教授

報 告 者

小島 大輔

平成　13 年 2 月 22 日

高知工科大学 電子・光システム工学科

目次

第1章 はじめに

最近ではパソコンに限らず身のまわりのちょっとした電気製品にまで想像以上の集積回路が組み込まれている。これらの集積回路がどのように設計されどのように動作しているかなどの興味から、ディジタル回路に対する理解を少しでも深めることができればと思い、この課題に取り組むことにした。

実際には短時間で一度に理解することができないので、ディジタル基本回路のひとつである順序回路の仕組みと設計を、カウントダウン時計の製作を通して学んだ。

本報告の第 2 章では、順序論理回路の設計に関する基本事項をまとめている。第 3 章では、標準論理ＩＣを使用した順序論理回路の設計について述べ、第 4 章では、新しい設計方法であるハードウェア記述言語による設計方法について述べている。最後に第 5 章で卒業研究の成果をまとめている。

第２章　順序論理回路
（sequential logic circuits）

2.1　順序論理回路

論理回路は時間に関係しない回路と時間に依存する回路に大別できる。時間に関係なく、与えられた入力値から出力値が一意的に決まる回路が**組み合わせ回路**である。

順序論理回路（sequential logic circuits）は、入力だけでなく過去の入力によって得られた現在の状態にも影響されて次の状態と出力が決まる回路である。順序回路の概念図を図 1.1 に示す。過去の状態をフィードバックして**記憶素子**に記憶させ、それを現在の状態（内部状態：internal state）とする。順序回路はこの**現在の状態**と**入力**を組み合わせ回路に入力して**次の状態**と**出力**を得るもので、組み合わせ回路と記憶素子で構成されるものである。順序回路を最も簡単に理解できる典型的な例がカウンタである。例えば現在の状態が３の時、次の入力１があれば次の状態が４になり、現在の状態によって次の状態が異なることが分かる。

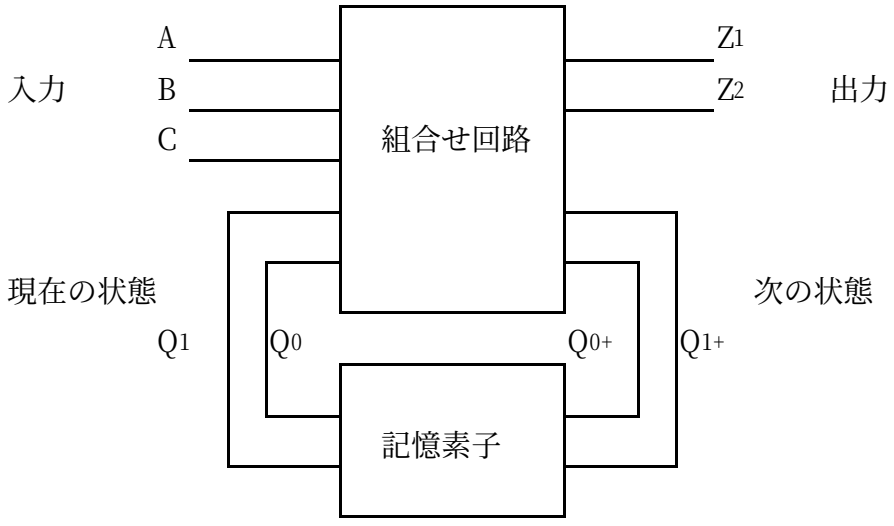


図 2.1　順序回路の概念図

図 2.1 において、入力Ａ、Ｂ、Ｃと現在の状態Q_0、Q_1の組み合わせで出力関

数Z_0、Z_1および次の状態Q_{0+}、Q_{1+} が求まる。順序回路に使用する１ビットの記憶素子を**フリップフロップ**（flip-flop：ＦＦと略記する）といい、動作の異なるいろいろな種類がある。出力Z_0、Z_1を求める方法は従来の組み合わせ回路の設計順序と同じだが、現在の状態を記憶させるために使用するＦＦの動作が異なるのでQ_{0+}、Q_{1+} が異なる。順序回路の設計とは、入力Ａ、Ｂ、Ｃと現在の状態Q_0、Q_1 を順序回路の入力変数とし、次の出力Z_0、Z_1 および同時に得られた次の状態をＦＦに正しく記憶させるためのQ_{0+}、Q_{1+} を求めることである。したがって、記憶素子として使用するＦＦの動作特性とその制御方法について理解しなければならない。

記憶素子（memory：メモリ）のことを遅延素子（delay）ともいう。この遅延時間を発生させる方法として、外部から動作時点を指定する**クロックパルス**（clock pulse：cp と略記する）を用いる**同期式**（synchronous）および素子の**伝搬遅延時間**（信号が素子に入力してから出力されるまでの時間）を利用する**非同期式**（asynchronous）がある。前者は動作速度がクロックパルスに依存するが、現在の状態が十分安定したと思われる時間間隔で動作されるので安定性がよく、コンピュータなどの設計に用いられる。後者は、原理的に前者よりは高速だが、回路が複雑になると遅延時間の推定が困難になるので、むしろ時間に左右されない比較的遅いディジタル回路設計に使われる。

2.2　状態遷移表と状態遷移図

組み合わせ回路ではその設計に真理値表を用いたが、順序回路では真理値表の代わりに**状態遷移表**（state transition table）を用いる。順序回路は安定した現在の状態に入力信号が加わると出力が出るとともに次の状態へ遷移する。この移りゆく状態を表にしたものを状態遷移表といい、理解しやすくするために図示したものを**状態遷移図**（state transition diagram）という。

表 2.1　状態遷移表

入力	現在の状態	次の状態	出力
0	S0	S0	0
1	S0	S1	0
0	S1	S1	0
1	S1	S2	0
0	S2	S2	0
1	S2	S0	1

2.3　フリップフロップ（ＦＦ）

順序回路を構成する重要な部品である記憶素子（メモリ）は、通常多数の**ラッチ**（latch）やフリップフロップと呼ばれるものによって構成されている。ラッチやＦＦは１ビットの記憶素子であり、いくつかの種類がある。ＦＦの状態変化（クロックを含む）は入力による。ある論理値とその論理が逆の値の両方（これを相補（complementary）信号と言う。）を出力することができる。ＦＦのほとんどはセットとリセットの機能が付加されており、これらはＦＦでレジスタやカウンタを構成した時に、その初期値の設定に使用される。

2.3.1　ＳＲフリップフロップ

ＳＲ－ＦＦ（set reset）は２個のＮＡＮＤ素子で構成される最小構成のＦＦである。実際のＦＦは、ＳＲ－ＦＦの機能と他の３つのＦＦを組み合わせたものがほとんどである。ＦＦとは図 2.2 に示す**２安定マルチバイブレータ**（multi vibrator）のことである。マルチバイブレータにはこの他に無安定（安定のない発信器）と１安定（安定点が１つある単発パルス発信器）があり、いずれも実用ディジタル回路ではよく使用される。

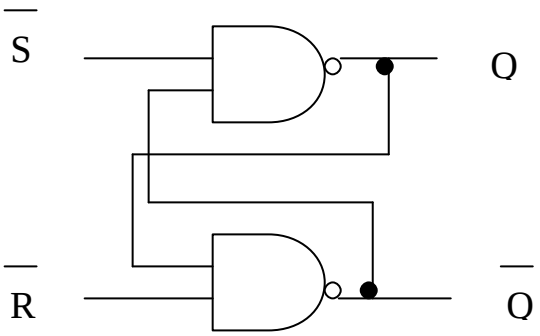


図 2.2　**２安定マルチバイブレータ**

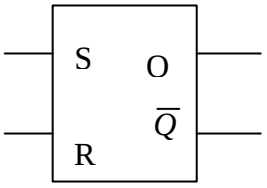


図 2.3　SR－FF **の記号**

表 2.2 SR－FF の遷移表

入力	現在の状態	次の状態
S　R	Q	Q'
0　0	0	0
0　0	1	1
0　1	0	0　(リセット)
0　1	1	0　(リセット)
1　0	0	1　(セット)
1　0	1	1　(セット)
1　1	0	×
1　1	1	×

ＳＲ－ＦＦのセットとはQ端子から 1 を出力させることで、リセットとはQ端子から 0 を出力させることである。外部からＳまたはＲに入力しない限りは永久にQから 1 または 0 のいずれかの出力状態（2 つの安定状態）を維持する。この 2 つの安定状態は、Q端子から 1 が出ている時はＦＦが１を記憶し、０が出ている時は０を記憶していることを意味している。

2.3.2　Ｔフリップフロップ

Ｔ－ＦＦ（toggle/trigger）**とは、Ｔ入力があるたびにQ出力が反転動作をする（トグル動作という）ＦＦである。これは１ビットの２進カウンタに相当する。そのタイミングチャートを図 2.4 に、また遷移表を表 2.3 に示す。**

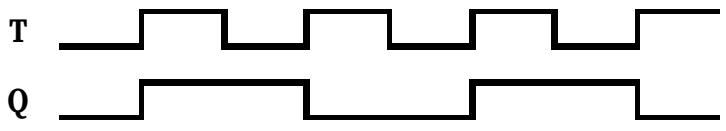


図 2.4　Ｔ－ＦＦのタイミングチャート

表 2.3　Ｔ－ＦＦの遷移表

入力　T	現在の状態　Q	次の状態　Q’
0	0	0
1	0	1 反転
0	1	1
1	1	0 反転

ＦＦは一般にＳＲ－ＦＦの機能であるセットとリセット端子を兼備している。これらは*ダイレクトセット*（S_D）または*プリセット*（ＰＲ）および*ダイレクトリセット*（R_D）または*クリア*（ＣＬＲ）などと呼ばれている。
Ｔ－ＦＦの記号を図 2.5 に示す。

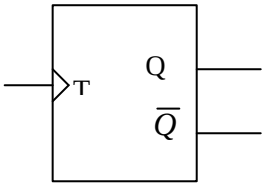


図 2.5　Ｔ－ＦＦの記号

2.3.3　ＪＫフリップフロップ

ＪＫ－ＦＦは、ＳＲ－ＦＦにｃｐを追加してＳＲ－ＦＦの禁止入力ＳＲ＝0を改善したものである。Ｊ/K入力のいずれかが“0”のとき、ＳＲフリップフロップと同じ動作をし、Ｊ/K入力ともに“1”のとき、Q出力が反転（トグル）という動作をする。ＳＲ－ＦＦは、Ｓ、Ｒに同時に入力することを禁止しているが、ＪＫ－ＦＦはＪ、Ｋに同時に入力した場合でもｃｐに同期して安定した出力が得られる。ＳＲ－ＦＦやＤ－ＦＦの機能を包含しているので、それらよりも適用範囲は広い。

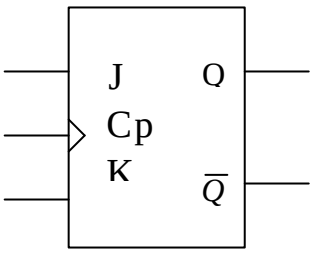


図 2.6　JK－FF の記号

表 2.4　JK－FF の遷移表

入力		現在の状態	次の状態	
J	K	Q	Q' (Cp＝1)	Q' (Cp＝0)
0	0	0	0	0
0	0	1	1	1
0	1	0	0 (リセット)	0
0	1	1	0 (リセット)	1
1	0	0	1 (セット)	0
1	0	1	1 (セット)	1
1	1	0	1 (反転)	0
1	1	1	0 (反転)	1

2.3.4　Dフリップフロップ

D－FF（delay/data latch）は**cp**に同期して**cp**が入力する直前のD入力の状態と同じ状態をQから出力するものである。タイミングチャートを図 2.7 に、遷移表を表 2.5 に示す。常に**cp＝1**の時動作するので、この表では**cp**を省いている。表からD－FFの特性方程式は次式になる。

$$Q' = DQ + D\overline{Q} = D$$

また、D－FFの記号を図 2.8 に示す。

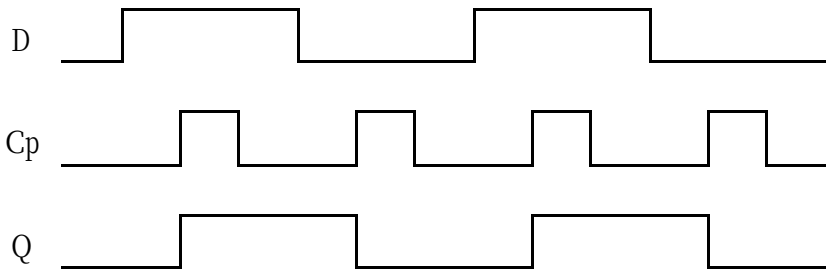


図 2.7　D－FF のタイミングチャート

表 2.5　D－FF の遷移表

D	Q	Q'
0	0	0
0	1	0
1	0	1
1	1	1

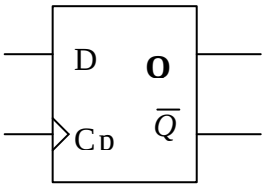


図 2.8　D－FF の記号

2.3.5　マスタースレーブ型フリップフロップ

状態変化がクロックパルスの立ち上がり（ポジティブエッジ）／立下り（ネガティブエッジ）によって生じるＦＦを**エッジトリガ型フリップフロップ**という。クロックの立ち上（下）がり（マスタ）と立ち下（上）がり（スレーブ）の組で１つの動作（２段階動作）を行うＦＦを**ＭＳ型ＦＦ**という。動作タイミングが異なるだけで、種類はエッジトリガ型フリップフロップと同様なものがある。

2.4　カウンタ

カウンタ（counter）は、単に計数回路としてだけではなく、タイマ回路や分周回路、さらには計算機の制御、モーターの回転数の計算、時計など非常に広く用いられています。

カウンタはパルスの数を計数し、記憶する回路で、その動作方式から非同期式（asynchronous）と同期式（synchronous）に大別される。基本となる回路素子として、ＦＦが用いられる。

2.4.1　同期式カウンタ

非同期式カウンタの問題点は、カウント数が大きくなれば遅延時間が蓄積されていくことである。この欠点を解決するのが同期式カウンタで、すべてのＦＦのクロック端子を共通のクロックで一斉に制御できるようになっている。そ

の結果、伝搬遅延時間はＦＦの段数に関係なく最も応答の遅いＦＦ１個分の遅延ですむ。しかしＦＦの段数が多くなるとクロックのファンアウト数が増え、クロック入力のインピーダンスが減少するので波形ひずみの原因になりやすい。

すべてのＦＦがクロックに同期して動作する同期式カウンタはすべてのＦＦが並列動作することから並列カウンタともいう。

参考として図 2.9　にＪＫ－ＦＦによる同期式 4 ビット２進カウンタの例を示す。

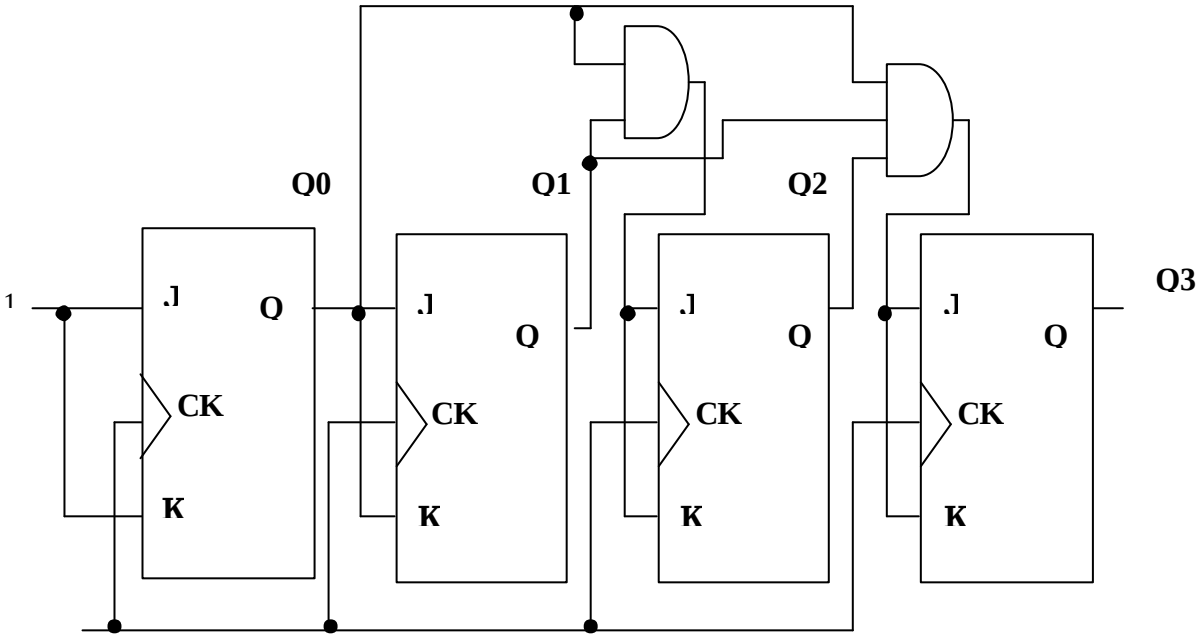


図 2.9　ＪＫ－ＦＦによる同期式 4 ビット２進カウンタ

2.4.2　同期式と非同期式

どちらの方式を選択するかは用途によって選択する必要がある。一般的には同期式カウンタの方が問題は少ない。非同期式カウンタは回路構成が簡単である利点はあるが、前段のフリップフロップ出力が、後段のクロック入力となって次々に伝えられるため、伝搬遅延時間を考慮しなければならない。一方、同期式カウンタは入力パルスを全フリップフロップのクロックに共通に接続する形式であるため伝搬遅延時間は少ない。

また、非同期式カウンタでは、各フリップフロップの出力をデコードするとスパイク状のノイズが発生することがある。このノイズは回路の誤動作になることが多い。一方、同期式カウンタでは、各フリップフロップが同時に変化するため、出力をデコードしてもノイズは発生しにくい。

2.4.3　可逆カウンタ

カウントアップ（カウント数の増加、インクリメント）のカウントダウン（カウント数の減少、デクリメント）の両方が可能なカウンタを可逆カウンタ（アップダウンカウンタ）という。

第３章 標準論理ＩＣによる回路設計

3.1　標準論理ＩＣ

初期の IC は超小型の電子部品をモジュールとして組み合わせたもので、1970 年代以降は、コンデンサやトランジスタ回路などを 1 つのチップ上に形成する半導体 IC が主流となった。半導体 IC は、形成されるトランジスタのタイプの違いによりバイポーラ型と MOS 型とに分類される。さらに最近ではバイポーラ型とＭＯＳ型とを組み合わせたＩＣも開発されている。

3.1.1　バイポーラ型 IC

バイポーラとは「2 つの極性」という意味で、電子と正孔（ホール）の 2 つが IC 内部の電流の運び手（キャリア）になっているため、こう呼ばれる。

トランジスタが動作状態にあるとき、素子の中を電流が流れるメカミズムに着目し、電子または正孔のどちらか一方だけがキャリアとして動作しているものをユニポーラ（単極性）と呼び、MOS 型トランジスタがこれにあたる。電子と正孔の両方がキャリアとして動作しているものをバイポーラ型トランジスタと呼ぶ。

3.1.2　MOS 型 IC

MOS 型とは、Metal （金属）・Oxide （酸化膜）・Semi-conductor（半導体）の頭文字の略である。この略号そのものが構造を表し、半導体基板の表面に薄いシリコン酸化膜を作り、その上に金属電極（ゲート）をつけたもので、このゲートにかける電圧によって半導体の機能が制御される。

MOS 型は、構造が簡単で高集積化に適しており、小さな電力で動作するのが特長である。特に CMOS(Complementary＝相補形 MOS)タイプは、消費電力が低いため、現在製造されている MOS 型 IC のなかの主流となっている。

MOS 型 IC を機能面で分類すると、メモリ（記憶素子）とロジック（論理素子）の 2 つに分かれる。

メモリ（記憶素子）

膨大な量の計算結果やその途中のデータなどの情報を記憶する。メモリはさらに、RAM（書き込み読み出しメモリ）と ROM（読み出し専用メモリ）とに大別できる。

ロジック（論理素子）

演算処理を行う役目の IC **で、各種の計算をしたり、情報と情報を比べたり、他の** IC **などのコントロールを行う。ロジックは、さらにマイクロプロセッサや周辺** LSI **などのマイコンと、専用ロジックに分類できる。**

このほかにバイポーラ型とユニポーラ型の両方の利点を活かすため、これらを混在させたＢｉＣＭＯＳディジタルＩＣも開発されたが、製造プロセスが複雑であること、CMOS **に比べ性能面の利点が少ないということで現在では殆ど作られていない。**

3.1.3　標準論理ＩＣファミリ

標準論理ＩＣは、メーカーが違っても電気的な特性が同じように作られた一連の仲間を持っており、それらをＩＣファミリと呼んでいる。

汎用ロジックＩＣのファミリにはいくつかの種類がるが、一般的に、ロー・パワー・ショットキーＴＴＬ（ＬＳ　ＴＴＬ）とＣＭＯＳの 74**ＨＣシリーズおよび** 4000/4500 **シリーズが多く使われる。**

ファミリ化されたＩＣは、そのＩＣに刻印されている型名を見れば製造したメーカーと機能が分かるようになっている。

例えば、74 **シリーズのロー・パワー・ショットキーＩＣである** 74**ＬＳの場合は、**

英文字　　数字
□□74LS×××

と刻印されていれば、□□の英文字がメーカー名を表し、ＬＳがロー・パワー・ショットキー・ファミリであることを表し、×××の数字がＩＣの機能を表している。×××が同じ番号のものであれば、型名表示（この場合ＬＳと書かれた部分）によらず、同じ機能をもつように作られている。このため型名表示の部分を省略して、単に 74×××**と呼ぶことが多い。**

英文字からわかるメーカー名の代表的なものを表 3.1 **に、型名表示のファミリの例を表** 3.2 **に示す。**

表 3.1　標準論理ＩＣ製造メーカー名

SN	TI (テキサスインツルメント)
MC1	モトローラのCMOS製品
CD	RCAのCMOS製品
MM	NSのCMOS製品
μPD	日本電気のMOS製品
TC	東芝のCMOS製品
MB	富士通のデジタル IC
HD	日立のデジタル IC
DN	松下のデジタル IC

表 3.2　標準論理ＩＣ型名表示

L	ローパワーTTL
S	ショットキーTTL
LS	ローパワー・ショットキーTTL
H	高速TTL
AS	アドバンスト・ショットキーTTL
ALS	アドバンスト・ローパワー・ショットキーTTL
HC	高速CMOS

さらに標準論理ＩＣ型名表示についての特徴を表 3.3 に、ＣＯＭＳ論理ＩＣの型名表示についての特徴を表 3.4 に示す。

この他にＢｉＣＭＯＳファミリとして、74ＢＣシリーズがある。

表 3.3　ＴＴＬファミリとその特徴

型番	タイプ	特徴
74	スタンダード	最初に開発されたTTL。
74Ｓ	ショットキー	ショットキートランジスタを用いて高速化したもの。
74AS	アドバンスト･ショットキー	74Ｓタイプを低消費電力化したもの。
74LS	ローパワー ･ショットキー	スタンダードタイプに比べ低消費電力で、高速動作が可能。
74ALS	アドバンスト･ローパワー ･ショットキー	ASタイプをさらに低電力消費化したもの。
74F	FAST	ＬＳタイプに比べ低消費電力で高速化したもの。

表 3.4　ＣＭＯＳファミリとその特徴

型番	タイプ	特徴
4000/4500	スタンダード	**最初に開発されたＣＭＯＳ　ＩＣ、74 シリーズとは異なった**論理があり、低速だが電源電圧範囲が広い。
74HC	ハイスピード	低消費電力でTTLのＬＳタイプに匹敵する速度とファンアウトをもつ。CMOＳとして標準的に用いられている。
74HCT		HCタイプを、ＴＴＬとインタフェース可能にしたもの。
74AC	FACT	低消費電力でTTLのＦタイプに匹敵する速度とファンアウトをもつ。
74ACT		ACタイプを、ＴＴＬとインタフェース可能にしたもの。

3.2　カウンタ用ＩＣ

順序回路の設計にはカウンタＩＣがよく用いられる。TTL ファミリーの代表的なカウンタＩＣを表 3.5 に、また CMOS ファミリーの代表的なカウンタＩＣを表 3.6 に示す。

表 3.5　ＴＴＬファミリーの４ビットカウンタ

型番	特徴
SN74160	4ビット、同期プリセット、非同期クリア
SN74161	4ビット、同期プリセット、非同期クリア
SN74162	4ビット、同期プリセット、同期クリア
SN74163	4ビット、同期プリセット、同期クリア
SN74168	４ビットBCD、同期プリセット、クリアなし、同期式可逆カウンタ
SN74169	４ビットバイナリー、同期プリセット、クリアなし、同期式可逆カウンタ
SN74190	４ビットBCD、非同期プリセット、クリアなし、同期式可逆カウンタ
SN74191	４ビットバイナリー、非同期プリセット、クリアなし、可逆カウンタ
SN74192	４ビットBCD、非同期プリセット、非同期クリア、 同期式可逆カウンタ、並列入力
SN74193	４ビットバイナリー、非同期プリセット、非同期クリア、 同期式可逆カウンタ、並列入力
SN74568	４ビットBCD、同期プリセット、同期／非同期クリア、 同期式可逆カウンタ、３ステート出力
SN74569	４ビットバイナリー、同期プリセット、同期／非同期クリア、 同期式可逆カウンタ、３ステート出力
SN74390	非同期式、SN7490 ×　2、プリセットなし
SN74393	非同期式、SN7493 ×　2、

表 3.6　ＣＭＯＳファミリーの４ビットカウンタ

型番	特徴
40102	同期式 4ビットBCD　DOWN ×　2、カスケード接続で拡張可能
40160	同期式 4ビット10進、非同期クリア端子付き
40161	同期式、40160 のバイナリー型
40162	同期式、40160 の同期クリア端子付き
40163	同期式、40160 のバイナリー型で同期式クリア端子付き
40192	4ビットBCD　UP／DOWNの独立したクロック入力、非同期リセット、プリセット機能
40193	4ビットバイナリー　UP／DOWNの独立したクロック入力、非同期リセット、プリセット機能
4526	同期式プログラマブル4ビットバイナリー、非同期プリセット／クリア

カウンタには、パルスの入力違いにより、同期式と非同期式とがある。同期式は入力クロックに同期して各ビットの出力が変化するものであり、非同期式は入力クロックに同期しないで各ビット出力が変わるものである。

また、これらのファミリーの中には、カウント・アップとカウント・ダウンの切り替えが出来るアップ／ダウン・カウンタ（可逆カウンタ）も含まれている。

さらに計数値により、2 進、4 進､8 進、16 進・・・などの 2ｎ進カウンタであるのがバイナリ―カウンタ、ＢＣＤ（Binary Coded Decimal number）、つまり 2 進化 10 進数を扱うＢＣＤカウンタに分類できる。また計数の初期値を外部から設定できるのがプログラマブル・カウンタである。

カウンタの内部記憶回路であるフリップフロップやラッチは、ＪＫまたはＤフリップフロップであり、一般にクリア端子（リセット：ＣＬＲ）とプリセット端子（セット：ＰＲ）を備えている。これらの入力信号は、クロック信号とは無関係に、強制的に信号をＨレベル（プリセット）またはＬレベル（リセット）にするもの（非同期式プリセットまた非同期リセット）とクロック信号に同期して出力信号をＨレベル（プリセット）またはＬレベル（リセット）にするもの（同期式プリセットまた同期式リセット）がある。

非同期式ＰＲまたはＣＬＫは、

①ＰＲにＬレベルを入力する（ピン配置図のＰＲ端子に○がついている）と、出力Ｑ＝Ｈ、$\overline{Q}$ ＝Ｌとなる。（アクティブロー：）

②ＣＬＲにＬレベルを入力する（アクティブロー：ピン配置図のＣＬＲ端子に○がついている）と、Ｑ＝Ｌ、$\overline{Q}$ ＝Ｈとなる。

これら、①、②の設定期間中はクロック入力を無視し、ＰＲ，ＣＬＲの入力を解除（Ｈレベルにする）した後もその状態を保持する。

③ＰＲ，ＣＬＲの両方を同時にＬレベルにすると、Ｑ、$\overline{Q}$ ともＨレベルを出力する。ただし、一方を先に解除すれば、出力の状態は、ＰＲがＬレベルのままであるか、ＣＬＲがＬレベルのままであるかによって①または②に従って動作する。また同時に解除した場合は、素子のばらつきや、不可の大小によって、Ｑ、$\overline{Q}$ のいずれかがＨレベルになるかは不定ある。

ＰＲ，ＣＬＲが両方とも解除（両方ともＨレベル）となっていれば、クロック信号に従ってフリップフロップ本来の動作をする。

74160 と 74161 はクリア入力がクロックに非同期で、クロックに無関係でクリアできる例であり、74162 と 74163 はクリア入力がクロックに同期して、クリア端子（ピン配置図のＣＬＲ端子に○がついている）がＬレベルのときにクロックの立ち上がりでクリアできる例である。

3.3　ＰＳｐｉｃｅによる回路設計

SPICE は Simulation Program with Integrated Circuit Emphasis の頭文字をとったものでカリフォルニア大学バークレー校(UCB)で開発された回路解析プログラムです。その後いろいろな会社から発表され、代表的なものとしては、HSPICE (Meta Software)や PSpice (Cadence)、SPICE-Plus (ADT)、DSPICE (Daisy)、MSPICE (Mentor)などがあり、今回使用したのは Cadence 社の PSpice A/Dである。

PSpice は主に 3 つの別々のソフトウェアからなり，構成は図 3.1 のようになっている。

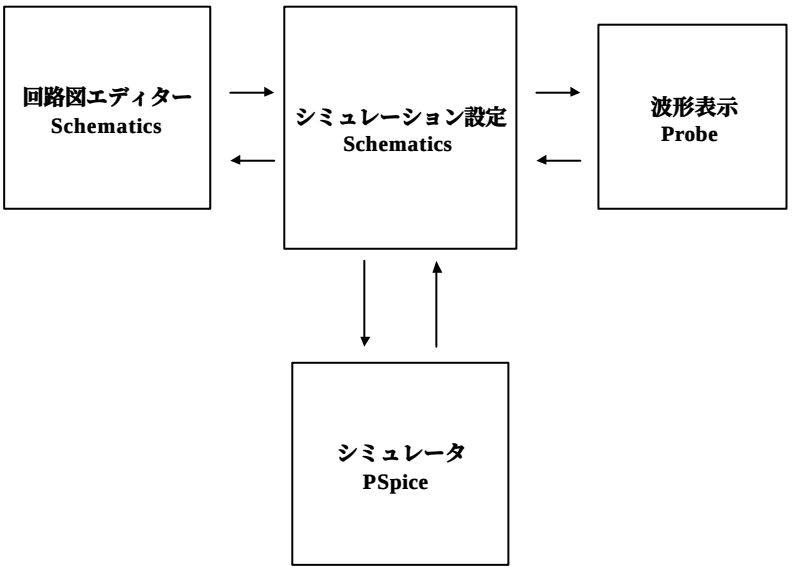


図 3.1　PSpice のソフトウェア構成

Capture

シミュレーションを行う元になる回路図を作成するプログラムで、シミュレーションの設定や、PSpice、Probe を有機的に結び付ける。

PSpice

シミュレータ本体、回路データ（ネットリスト）、部品のデータ（モデルパラメータ）、シミュレーション命令をアスキーファイルで入力しシミュレーションを行うプログラムである。

Probe

シミュレーション結果をグラフにして表示するプログラムである。

回路設計のシミュレーションの流れはだいたい次のようになっている。

①Capture で回路を設計する。
②Capture でシミュレーションの内容，素子のパラメータを設定する。
③PSpice でシミュレーションを実行する。
④Probe で結果をグラフに表示する。
⑤出力された結果が予定と違っていた場合はもう一度①に戻り回路の設計をやり直しシミュレーションし直す。

PSpice によるシミュレーションの種類

PSpice で実行できるシミュレーションは大きく分けて３種類になる。PSpice はこれらのシミュレーションを行う前に必ずバイアスポイント解析という初期値の計算を行う。

? **バイアスポイント解析**

PSpice でシミュレーションを実行する場合は前処理として必ず実行される。バイアスポイント解析は以下のように回路をとらえる。

・キャパシタは解放する
・インダクタは短絡する
・時間に関する導関数を持つ素子の値は０にする
・半導体等はモデルを用いて線形化する

このような処理を行うことによって、どのような回路でも抵抗及び電圧源、電流源のみで構成されるようになる。これによって、オームの法則、キルヒホッフの法則により回路中全てのノードの電流、電圧が計算できるようになる。

? **ＤＣ解析**

ＤＣ解析はバイアスポイント解析と同様に回路をとらえる。しかし、バイアスポイント解析は 1 回だけ値を算出するのに対し、ＤＣ解析では回路上の電圧値、電流値及び素子の値のうちどれか一つを変動させその変動による電圧値、電流値を算出する。

※ 出力は横軸が電圧、電流、素子値いずれかの静特性のグラフとなる。

? **ＡＣ解析**

正弦波交流信号を入力として抵抗、キャパシタ、インダクタを複素数で展開されたインピーダンスとして取り扱う。入力正弦波信号の周波数を変化させて回路上の電圧、電流を伝達関数的に絶対値、デシベル、実数分、虚数分、位相の形で出力する。
ただし、線形領域の解析になるので入力信号の振幅の変化によるサチレーションの計算はできない。

※ 横軸が周波数のグラフとなり、イメージ的にはスペクトラムアナライザー、ネットワークアナライザーと同じになる。

? **過渡解析**（Transient analysis）

過渡解析は、時刻ごとのノード電圧や電流を求める。バイアスポイント解析により初期状態を算出し、過渡的な回路動作を時間レンジで求める。過渡解析は AC 解析と違い非線形領域もカバーしているのでサチレーションの観測も可能である。

※ 横軸が時間のグラフとなり、オシロスコープによる観測になる。

このほかに温度変化の影響を調べるシミュレーション、素子の値のばらつきによる影響を調べるシミュレーション等も行える。

3.4　設計例

－カウントダウン時計－

今回は水晶発振器から発振された信号をデバイダ－に通すことによって 1Hz のクロックを生成し、ＳＮ74ＬＳ169Ｂを用いてカウントし７セグメントにて表示させるカウントダウン回路を設計した。この回路は最大 99 日 23 時間 59 分 59 秒までカウントダウンすることができる。全体のブロック図を図 3.2 に示す。

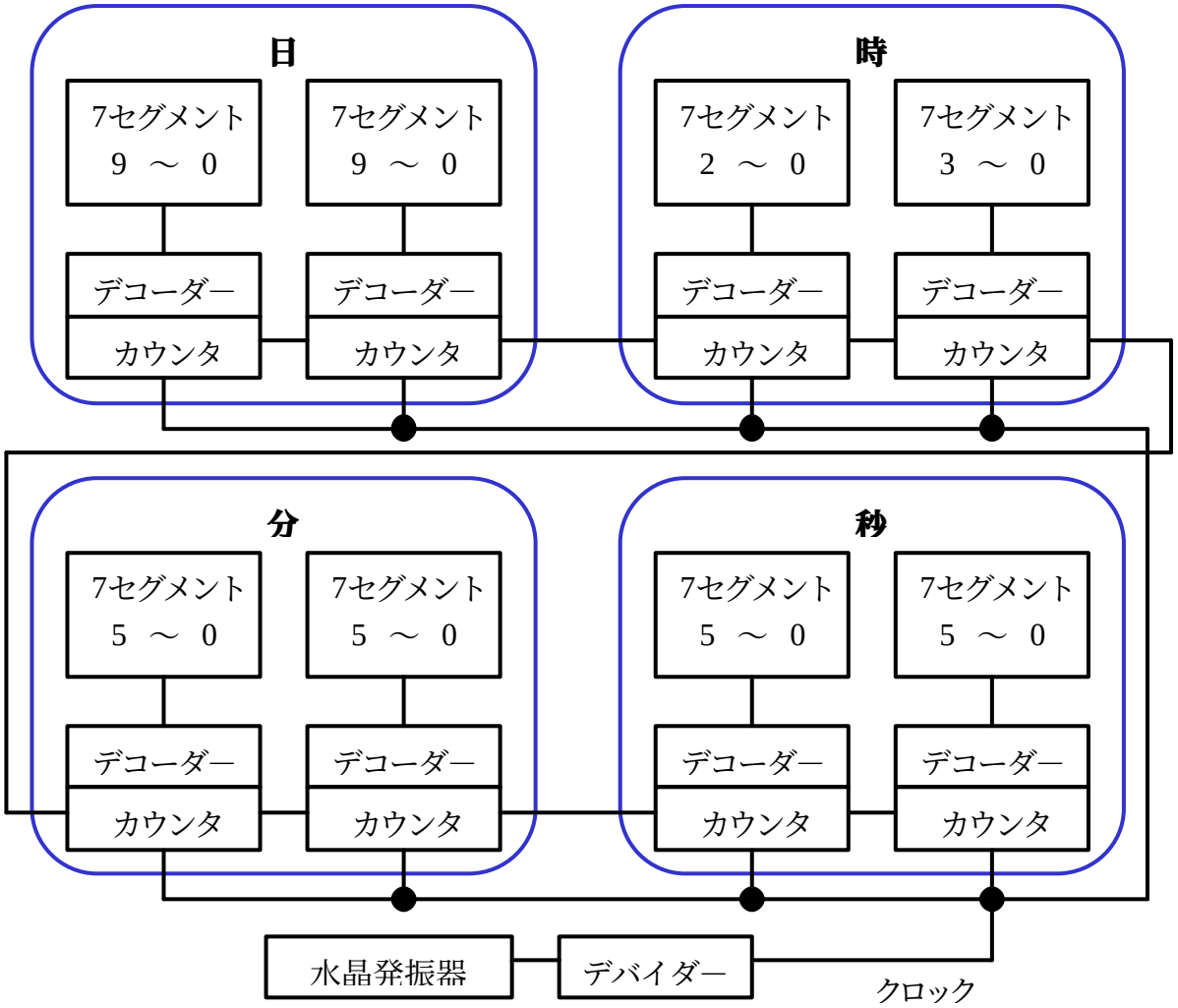


図 3.2　カウントダウン時計のブロック図

今回使用した主な材料

- **抵抗**
  10KΩ　**スイッチ用**　4 **本**
  220Ω　**セグメント用**　56 **本**
- IC
  SN74LS247P　**セグメント用**　8 **個**
  　4 **ビットの信号を** 7 **ビットの信号に変換するために使用した。**
  SN74LS169BN　**カウントダウン用**　8 **個**
  SN74LS08N　AND **回路用**　4 **個**
  SN74LS00N　NAND **回路用**　1 **個**

SN74LS292N　プログラマブルデバイダ　1 個

クロックを FF に何回通過させるかを外部から設定できる。クロックの周波数を 1 Hz にまで落とすために使用した。

・ 水晶（TOYOCOM　TCO－707F　4.194304MHz）

FF を通過するたびに周波数が２分の 1 される。このため周波数は 2^n である必要があり、そこで 4.194304MHz（2^{22} Hz）を使用した。

・ セグメント　8 個
・ 電解コンデンサ（470μF）電源用　2 個
・ セラミックコンデンサ（0.1μF）スイッチ用　4 個
・ プッシュスイッチ　４個

カウントダウン時計の製作に用いるそれぞれの標準論理IC の動作特性と回路構成を以下に表や図としてまとめる。

SN74LS00とSN74LS08

SN74LS00 は NAND 回路の IC であり、SN74LS08 は AND 回路の IC である。

表 3.7　SN74LS00 と SN74LS08 の動作特性

	SN74LS00			SN74LS08		
	MIN	NOM	MAX	MIN	NOM	MAX
V_{CC} Suppy voltage (V)	4.75	5	5.25	4.75	5	5.25
V_{IH} High-level input voltage (V)	2			2		
V_{IL} Low-level input voltage (V)			0.8			0.8
I_{OH} High-level output current (mA)			-0.4			-0.8
I_{OL} Low-level output current (mA)			16			16
T_A Operating free-air temperature (℃)	0		70	0		70

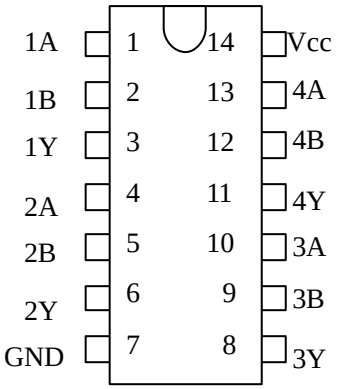


図 3.3　N74LS00 **と** SN74LS08 **の** IC **のピン配置図**

表 3.8　SN74LS00 **と** SN74LS08 **の状態表**

SN74LS00		
入力		**出力**
A	**B**	**Y**
L	L	H
L	H	L
H	L	L
H	H	L

SN74LS08		
入力		**出力**
A	**B**	**Y**
L	L	H
L	H	L
H	L	L
H	H	L

SN74LS169B

SN74LS169B **は同期式** 4 **ビット　アップ/ダウン　バイナリーカウンタの** IC **である。**

表 3.9　74LS169B の動作特性

		SN74LS169B		
		MIN	NOM	MAX
VCC Suppy voltage (V)		4.5	5	5.5
VIH High-level input voltage (V)		2		
VIL Low-level input voltage (V)				0.8
IOH High-level output current (mA)				-0.4
IOL Low-level output current (mA)				8
Fclock Clock frequency (MHz)		0		40
tw Pulse duration, CLK high or Low (ns)		12.5		
tsu Setup time sefore CLK (ns)	A, S, C or D	15		
	$\overline{ENP}$ or $\overline{ENT}$	15		
	$\overline{LOAD}$	15		
	U/$\overline{D}$	15		
th Hold time,data after CLK		0		
TA Operating free-air temperature (℃)		0		70

SN74LS169B はカウントをアップするかダウンさせるか選択することができる。カウントアップしたい場合は、下の図 3.4 ピン番号 1 番に H レベルの信号を入れる。逆にカウントダウンしたい場合は、L レベルの信号を入れる。
今回はカウントダウンする回路を製作するのでピン番号 1 番に L レベルの信号を入れた。

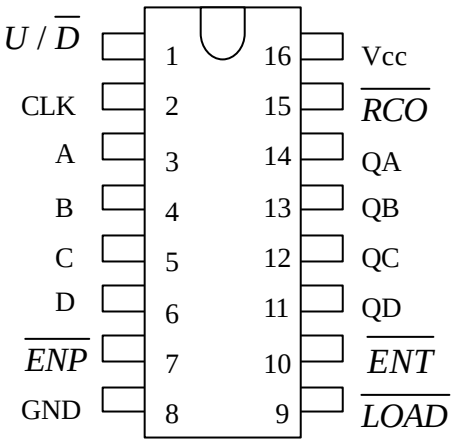


図 3.4　74LS169B の IC のピン配置図

SN74LS169B を論理回路で表すと図 3.5 のようになっている。

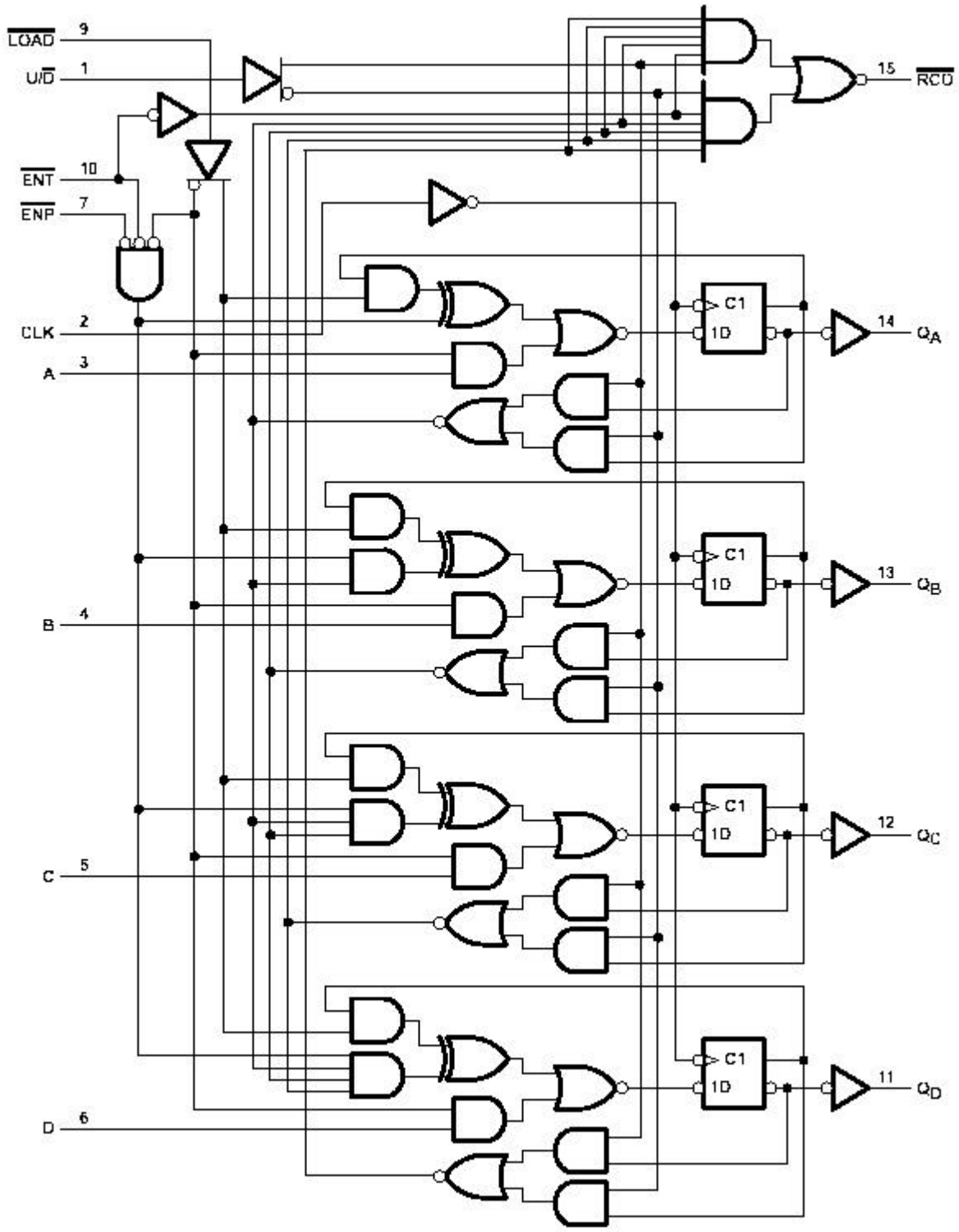


図 3.5　N74LS169B の論理回路図

(参考：http://www.tij.co.jp **)**

SN74LS169B は$\overline{ENP}$と$\overline{ENT}$の AND を取った信号が L レベルの時にだけ動作する。$\overline{RCO}$は次の桁に桁上げないし、桁下げの信号を出力する。$\overline{LOAD}$はカウントアップ/ダウン時にクロックに同期して信号を割り込ませてカウントを 1 つ先に進めることができる。

SN74LS292N

SN74LS292N はプログラマブルデバイダであり、水晶発振器からクロックをこの IC に通して 1Hz にする。

表 3.10　　SN74LS292N の動作特性

	SN74LS292N		
	MIN	NOM	MAX
V_{CC} Suppy voltage (V)	4.7	5	5.25
V_{IH} High-level input voltage (V)	2		
V_{IL} Low-level input voltage (V)			0.8
I_{OH} High-level output current (mA)			-1.2
I_{OL} Low-level output current (mA)			24
Fclock Clock frequency (MHz)	0		30
tw Pulse duration, CLK high or Low (ns)	16		
tsu Setup time sefore CLK (ns)	55		
th Hold time,data after CLK	15		
T_A Operating free-air temperature (℃)	0		70

SN74LS247P

SN74LS247P はデコーダーであり、SN74LS169B から出力された 4 ビットの BCD コードの出力を 8 ビットにデコードし 7 セグメントへ出力する。

表 3.11　SN74LS247P の動作特性

		SN74LS247P		
		MIN	NOM	MAX
V_{CC} Suppy voltage (V)		4.75	5	5.25
V_O Off-state output voltage	a thru g			15
I_O On-state output current	a thru g			40
I_{OH} High-level output current (μA)	$\overline{BI}$ / $\overline{RBO}$			-200
I_{OL} Low-level output current (mA)	$\overline{BI}$ / $\overline{RBO}$			8
T_A Operating free-air temperature (℃)		0		70

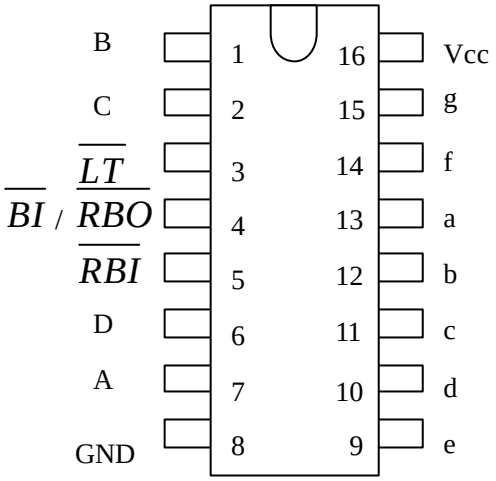


図 3.6　SN74LS247P のピン配置図

SN74LS247P は$\overline{BI}$ / $\overline{RBO}$ が H レベルのとき動作する。LT、RBI が H レベルのとき入力をデコードし出力する。どちらかが L レベルになると出力が全て L レベル（OFF 状態）になる。

表 3.12　SN74LS247P の状態表

入力				出力						
A	B	C	D	a	b	c	d	e	f	g
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	1	0	1	1

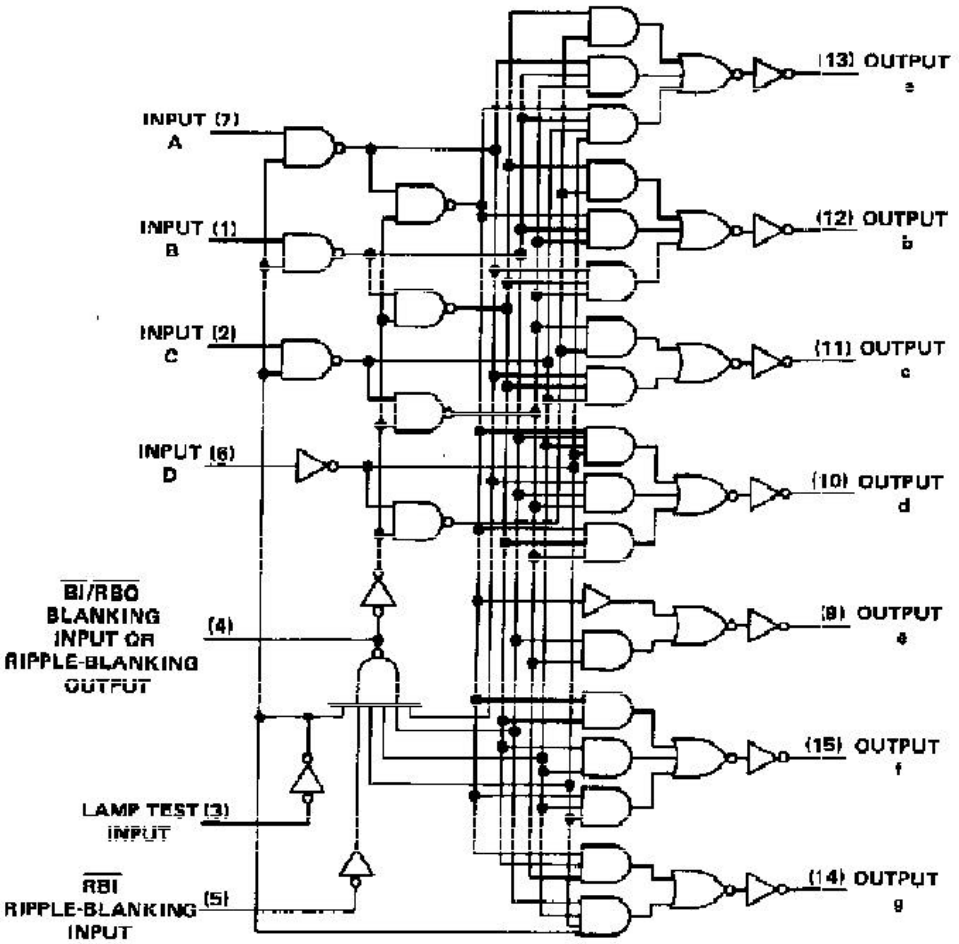


図 3.7　SN74LS247P の論理回路図

(参考：http://www.tij.co.jp)

Orcad の Capture で実際に組み上げた回路が図 3.8 であり、ＰＳpice でシミュレーションのため桁上げ信号をカットし、日、時間、分、秒が同時に作動するようにしている。これを ＰＳpice A/D でシミュレーションした結果が図 3.9 である。

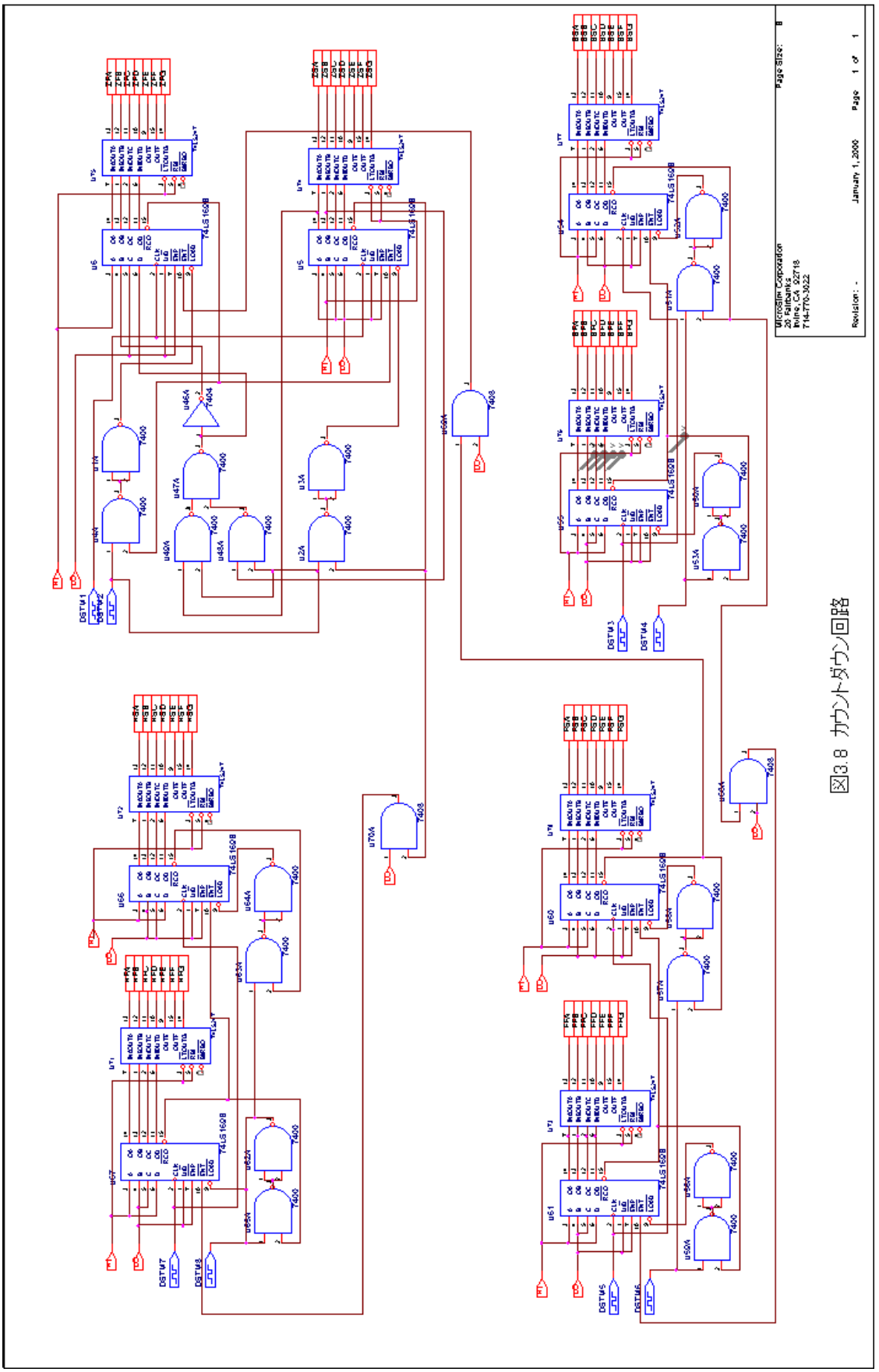


図3.8 カウント ダウン回路

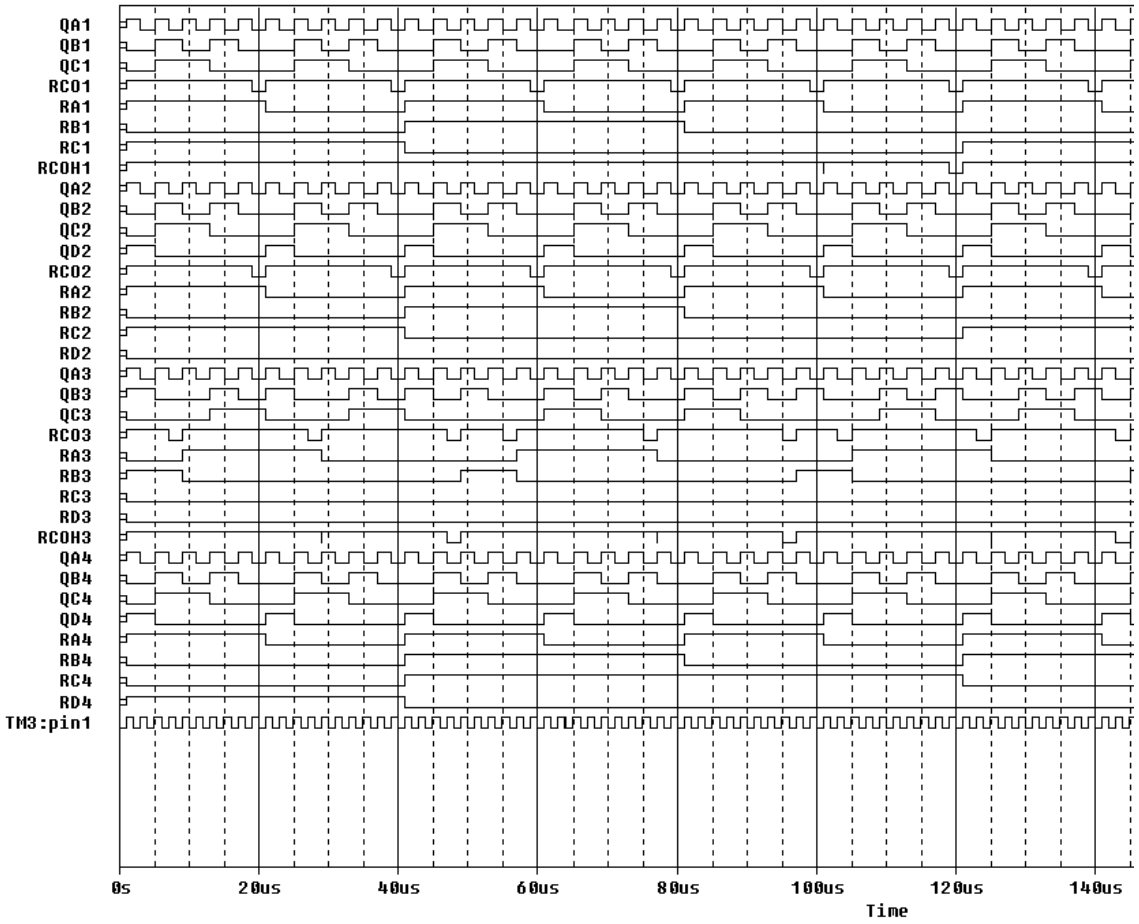
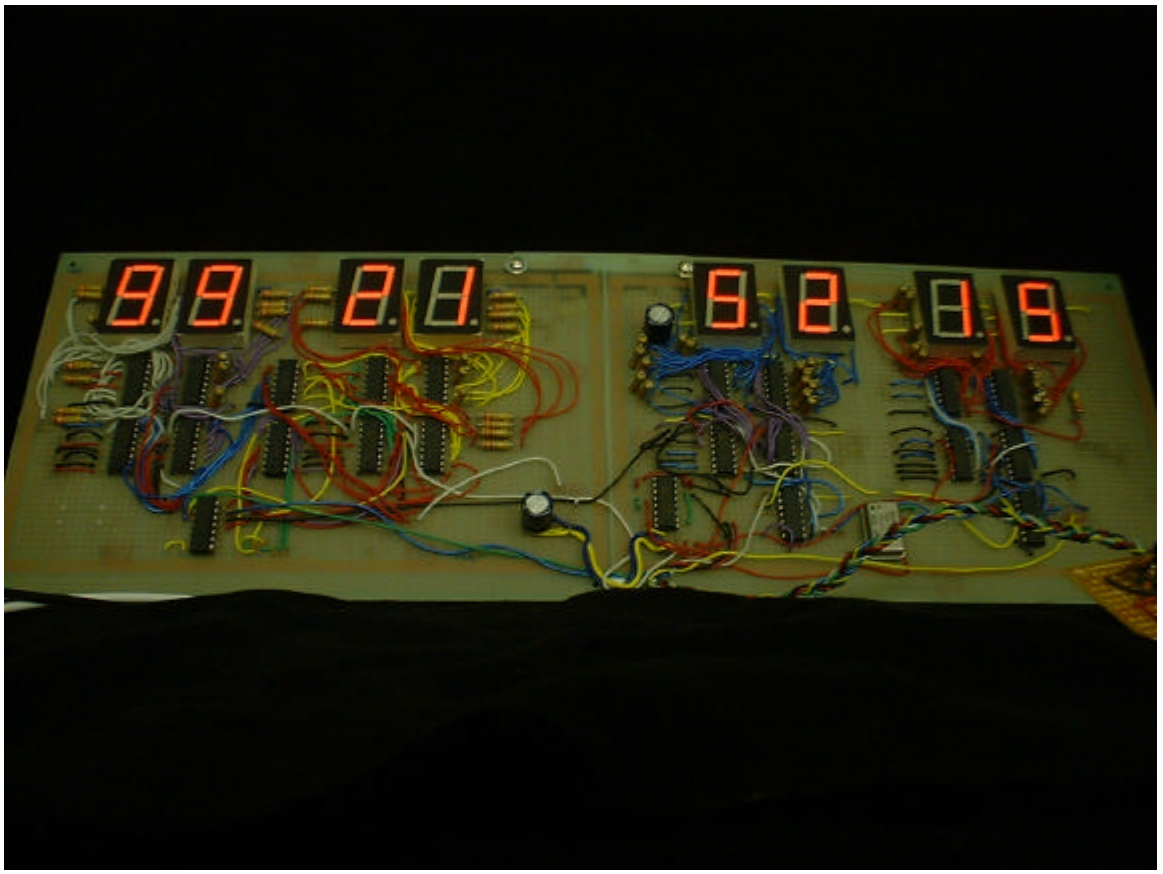


図 3.9　**シミュレーション結果**

完成図

出来上がった回路を実際に形にしてみようということで、市販のプリント基盤の上にさまざまな部品を組み上げてみた。この図 3.10 は実際に出来上がりカウントダウンしている様子をデジタルカメラにて撮影したところである。



※　９９日２１時間５２分１５秒

図 3.10　カウントダウン時計

第４章　ハードウェア記述言語による回路設計

4.1　ハードウェア記述言語

ハードウェア記述言語（ＨＤＬ：Hardware Description Language）とは、今まで人間の手作業によるディジタル回路の設計作業が、半導体集積化技術の飛躍的な向上に伴い、設計するディジタル・システムの規模は増大したため人間の手作業による回路図作成では限界となり開発された、Ｃや Pascal などのような形式言語である。ただしプログラム言語とは異なり、ハードウェアを設計するうえで欠かせない並列処理（同時処理）や時間、タイミングの概念などを記述できるものである。

ＨＤＬが出現したことによって、ＶＬＳＩ設計の手法は大きく変わってきた。従来はＲＴＬ※1（Register Transfer Level）回路の記述は仕様書をもとに行われており、計算機で動作を検証できるＲＴＬ表現も存在したが、ゲートレベルへの展開が可能な表現はなかったので、人間の手作業で回路を構築してゆくボトムアップ的設計手法※2 がとられてきた。１９８０年代後半になって、ＶＨＤＬや Verilog HDL といったハードウェア記述言語が開発されてＲＴＬ回路はＨＤＬで記述する事が一般的になり、ゲートレベルへの展開が可能になってきた。このように、ＨＤＬを用いてＲＴＬ設計をすることをＨＤＬ設計と呼んでいる。表 1 に今までに開発された主なＨＤＬをまとめてみた。

※1ＲＴＬ

仕様設計から出力された仕様書をもとに、まずＬＳＩ内部を機能ブロックに分割し、さらにブロックがレジスタと組合せ回路とで構成されるところまでブロック分割を繰り返す。そのようにして得られたブロックの中のレジスタとレジスタの間の組合せ回路を決定したうえで、クロックごとの回路動作を決定する。このときの表現をＲＴＬ記述といい、抽象度をＲＴＬという。ＲＴＬ設計は機能設計とも呼ぶ。

※2 ボトムアップ設計

まず開発対象となるディジタル回路をいくつかのブロックに分割し、設計された各ブロックを組み合わせて回路全体を構成する方法のこと。過去の設計資産を再利用しやすいので、大規模なディジタル回路の設計において採用されてきた。

表 1　HDL の歴史

言語	VHDL	Verilog HDL	UDL/I	SFL
開発開始時期	1981 年	1984 年	1987 年	1981 年
開発組織	IEEE	Cadence	日本電子工業振興会	NTT
言語仕様公表	1987 年	1985 年	1990 年	1985 年
論理シミュレータ	有	有	有	SECONDS
論理合成系	有	有	有	SFLEXP
規格の見直し	1993 年	1999 年（予定）	1992 年	予定なし

この表で、ＶＨＤＬ（Very High Speed integrated circuit Hardware Description Language）は１９８１年にスタートした米国国防総省のＶＨＳＩＣ（Very High Speed IC）プロジェクトで開発された HDL であり、１９８７年末に IEEE　Std‐1076 として、１９９３年には IEEE　Std‐1164 として規格が正式に承認され世界的に標準化された。VHDL はシステムからスイッチにいたるハードウェア全般の記述を対象としており、ハードウェアの構造と動作の両方を記述できる。VHDL では回路を階層的にとらえ、ある回路は下位のより小さな回路が接続されたものとして記述する。最下層の回路を動作的に記述し、それより上位の回路を構造的に記述する。VHDL は、ユーザ定義のタイプ、抽象度の高いデータタイプ、回路構成をコントロールするためのコンフィギュレーションなどに特徴がある。

また、Verilog HDL は米国の Cadence 社で Verilog XL というシミュレータ用に開発された HDL であり、世界的に普及している。しかし、標準化されたのは以外に遅く、1995 年に IEEE　Std‐1364 となった。Verilog HDL はシステムからスイッチにいたるハードウェア全般の記述を対象としており、ハードウェアの構造と動作の両方を記述できる。Verilog HDL でも回路は階層的に記述する。再下位層の回路は動作的に記述し、上位の回路は構造的に記述する。特徴として RTL に的を絞った簡素な記述で、テストパターン記述言語としての高い効率性などがあげられる。

SFL は日本で開発された HDL であり、制御回路の記述に特徴がある。SFL では、回路をデータパス部と制御回路に明示的に分けて記述し、制御回路の記述中からデータパスを構成する部品を手続き的に呼び出すことにより回路の動作を記述する。データパスを構成する部品間の接続やデータパスと制御回路の間の接続は合成によって行われる。制御回路の記述は単一クロックの完全同期

式ステートマシンを基本としている。SFL では階層的なステートマシンの表現が可能であり、複数のステートマシンによる並列処理、パイプライン処理、置いてきぼり処理の記述が可能である。SFL は複雑な回路を簡単に記述できる点で VHDL や Verilog HDL より優れているがあまり普及していない。なぜなら SFL の回路設計思想であるデータパスと制御回路を完全に分離して設計するスタイルが一般に理解されにくく、また日本で開発されたため外国では知名度も低くツールなどのサポートが弱いためである。

このように何種類ものＨＤＬが開発されていますが、今回は現在広く普及しているＶＨＤＬを用いることにした。

4.2 ＶＨＤＬによる記述方法

実際にはどのように使われているか、ＪＫ－ＦＦのＶＨＤＬの記述を参考例として示す。

```
1   library IEEE;
2   use IEEE.std_logic_1164.all;
3
4   entity JK_FF is
5       port ( CK, J, K : in std_logic;
6              Q, Qnot : out std_logic );
7   end JK_FF;
8
9   architecture BEHAVIOR of JK_FF is
10
11  signal INPUT : std_logic_vector ( 2 downto 0 )
12  signal TMP : std_logic;
13
14  begin
15   INPUT <= CK & J & K;
16      process ( INPUT ) begin
17          case INPUT is
18            when “101” => TMP <= ‘0’;
19            when “110” => TMP <= ’1’;
20            when “111” => TMP <= not TMP;
21            when others => null;
```

```
22          end case;
23      end process;
24      Q    <= TMP;
25      Qnot <= not TMP;
26  end BEHAVIOR;
```

○データタイプ

ＶＨＤＬには、論理型のデータ･タイプとして、論理値‘0’または、‘1’しかとれない bit 型およびその配列タイプである bit_vector 型しかなかった。しかし、ディジタル回路の設計やシミュレーション、論理合成などを行う際には、不定値‘Ｘ’、ハイ･インピーダンス‘Ｚ’、ドント・ケア‘－’などを使用できることが望ましい。

１～２行目

IEEE=The Institute of Electrical and Electronics Engineers 米国電気電子技術者協会

○エンティティとアーキテクチャ

ディジタル回路は、それに入力された何らかの信号（データ）に加工を施し、その加工結果を出力する電子回路である。このようなディジタル回路を設計するためには、

① どのような信号が入力されるか？
② その信号にどのような加工を施すのか？
③ どのような信号を出力させるのか？

を明確にする必要がある。

①や③の情報は、外部から入力される信号(入力信号)や外部へ出力する信号(出力信号)に関する情報、すなわち、インターフェースに関する情報である。これに対して②の情報は、回路内部の動作や構造に関する情報である。ＶＨＤＬでは、これらの外部情報が、エンティティ（entity）4 から 7 行目、内部情報がアーキテクチャ（architecture）9～26 行目と呼ぶ。

○信号線

それが入力信号線(入力ポート)なのか、(出力ポート）なのか、あるいは入出力兼用なのかを定める必要がある。

○信号の代入と演算子

ディジタル回路の動作や構造を記述する。データの流れを記述するためには、信号代入文＜＝を用いる。信号代入文では、複数の信号間には論理演算などを施し、その結果を他の信号線に代入することもできる。15.18～21.24.25 行目

4.3　VHDL による順序回路の設計例　－カウントダウン時計－

ＶＨＤＬによる順序回路の設計として、実際に Electronics Workbench の multiSIM VHDL を用いてカウントダウン時計を設計した。このカウントダウン時計は１０進カウンタ部、７セグメントディスプレイ表示部、日、時間、分、秒の設定部から構成されている。以下に各部の設計について説明する。

4.3.1　7 セグメントディスプレイ表示部

```
Library ieee;
use ieee.std_logic_1164.all;

entity segment7 is
   port (
         DI   : in std_logic_vector(3 downto 0);
         DO  : out std_logic_vector(6 downto 0)
   );
end segment7;

architecture RTL of segment7 is
begin
  process(DI)
  begin
    case DI is
                        --gfedcba
      when "0000" => DO <= "0111111";
      when "0001" => DO <= "0000110";
      when "0010" => DO <= "1011011";
      when "0011" => DO <= "1001111";
      when "0100" => DO <= "1100110";
      when "0101" => DO <= "1111101";
      when "0111" => DO <= "0000111";
      when "1000" => DO <= "1111111";
      when "1001" => DO <= "1101111";
      when others => DO <= "1111111";
    end case;
  end process;
end RTL;
```

4.3.1の記述は実際にカウントされた数値が出力として表示される部分の設定であり、４ビットのデータから７ビットのデータへの変換や端子の入出力の設定をしている。

4.3.2　１０進カウンタ部

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

entity count10 is
   port (
     CLK,RESET    : in std_logic;
     CYin         : in std_logic;
     Q            : out std_logic_vector ( 3 downto 0 );
     CYout        : out std_logic
     );
end count10;

architecture RTL of count10 is
signal TQ : std_logic_vector ( 3 downto 0 );
begin
process ( CLK,RESET,CYin ) begin  --10count
    if ( RESET = '1' ) then
        TQ <= "0000";
    elsif ( CLK 'event and CLK = '1' and CYin ='1' ) then
       if (TQ = "0000") then
          TQ <= "1001";
      else
          TQ <= TQ - '1';
      end if;
    end if;
end process;

process ( TQ,CYin ) begin
    if (TQ = "0000" and CYin = '1') then
       CYout <= '1';
```

```
    else
       CYout <= '0';
    end if;
end process;
Q <= TQ;

end RTL;
```

4.3.2の記述は、１０進数のカウンタの設定で９からカウントダウンして０になるとまた９に戻ると言う命令を記述している。またデータと端子の入出力の設定も行っている。

4.3.3　日、時間、分、秒の設定部

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

entity CDT is
  port( CLK, RESET    : in std_logic;
        SBF, SBS      : out std_logic_vector(6 downto 0);
--byou
       SHF, SHS      : out std_logic_vector(6 downto 0);--hun
        SJF, SJS      : out std_logic_vector(6 downto
0);--jikan
        SHIF, SHIS    : out std_logic_vector(6 downto 0)
--hiniti
 );
end CDT;

architecture RTL of CDT is


component segment7
port (
        DI   : in std_logic_vector(3 downto 0);
        DO  : out std_logic_vector(6 downto 0)
```

```
  );
end component;

component COUNT10 is
port (
       CLK, RESET      : in std_logic;
       CYin            : in std_logic;
        Q              : out std_logic_vector(3 downto 0);
       CYout           : out std_logic
  );
end component;


signal TQi, TQj, TQk, TQl, TQm, TQn, TQo, TQp
                            : std_logic_vector(3 downto 0);
signal CYa, CYb, CYc, CYd, CYe, CYf, CYg  : std_logic;

begin
--byou
process (CLK, RESET) begin
    if (RESET = '1' ) then
       TQi <= "1001";
    elsif (CLK'event and CLK = '1' ) then
       if (TQi = "0000" ) then
          TQi <= "1001";
       else
          TQi <= TQi - '1';
       end if;
     end if;
end process;

process (TQi) begin
   if (TQi = "0000" ) then
      CYa <= '1';
   else
      CYa <= '0';
```

```
   end if;
end process;

process (CLK, RESET, CYa) begin
    if (RESET = '1' ) then
      TQj <= "0101";
    elsif (CLK'event and CLK = '1' and CYa ='1') then
      if (TQj = "0000" ) then
         TQj <= "0101";
      else
          TQj <= TQj - '1';
      end if;
     end if;
end process;

process (TQj,CYa) begin
   if (TQj = "0000" and CYa = '1') then
      CYb <= '1';
   else
      CYb <= '0';
   end if;
end process;

--hun
process (CLK, RESET, CYb) begin
    if (RESET = '1' ) then
       TQk <= "1001";
    elsif (CLK'event and CLK = '1' and CYb = '1' ) then
      if (TQk = "0000" ) then
         TQk <= "1001";
      else
          TQk <= TQk - '1';
      end if;
     end if;
end process;
```

```
process (TQk, CYb) begin
   if (TQk = "0000" and CYb = '1') then
      CYc <= '1';
   else
      CYc <= '0';
  end if;
end process;

process (CLK, RESET, CYc) begin
    if (RESET = '1' ) then
       TQl <= "0101";
    elsif (CLK'event and CLK = '1' and CYc ='1') then
       if (TQl = "0000" ) then
          TQl <= "0101";
       else
           TQl <= TQl - '1';
       end if;
     end if;
end process;

process (TQl,CYc) begin
   if (TQl = "0000" and CYc = '1') then
      CYd <= '1';
   else
      CYd <= '0';
  end if;
end process;

--jikan
process (CLK, RESET, CYd) begin
    if (RESET = '1' ) then
       TQm <= "0011";
    elsif (CLK'event and CLK = '1' and CYd = '1') then
       if (TQm = "0000" ) then
          TQm <= "1001";
       if (TQm = "0000" and TQn = "0000" ) then
```

```
          TQm <= "0011";
       end if; -- <= point
       else
           TQm <= TQm - '1';

       end if;
     end if;
end process;

process (TQm, CYd) begin
    if (TQm = "0000" and CYd = '1') then
       CYe <= '1';
    else
       CYe <= '0';
   end if;
end process;

process (CLK, RESET, CYe) begin
     if (RESET = '1' ) then
        TQn <= "0010";
     elsif (CLK'event and CLK = '1' and CYe ='1') then
       if (TQn = "0000" ) then
          TQn <= "0010";
       else
           TQn <= TQn - '1';
       end if;
     end if;
end process;

process (TQn,CYe) begin
    if (TQn = "0000" and CYe = '1') then
       CYf <= '1';
    else
       CYf <= '0';
   end if;
end process;
```

```
--hiniti
process (CLK, RESET) begin
    if (RESET = '1' ) then
       TQo <= "1001";
    elsif (CLK'event and CLK = '1' and CYf = '1') then
       if (TQo = "0000" ) then
          TQo <= "1001";
       else
          TQo <= TQo - '1';
       end if;
     end if;
end process;

process (TQi, CYf) begin
   if (TQi = "0000" and CYf = '1') then
      CYg <= '1';
   else
      CYg <= '0';
  end if;
end process;

process (CLK, RESET, CYg) begin
    if (RESET = '1' ) then
       TQp <= "1001";
    elsif (CLK'event and CLK = '1' and CYg ='1') then
       if (TQp = "0000" ) then
          TQp <= "1001";
       else
          TQp <= TQp - '1';
       end if;
     end if;
end process;

--7seg
U0 : segment7 port map(TQi, SBF);--byou
```

```
U1 : segment7 port map(TQj, SBS);
U2 : segment7 port map(TQk, SHF);--hun
U3 : segment7 port map(TQl, SHS);
U4 : segment7 port map(TQm, SJF);--jikan
U5 : segment7 port map(TQn, SJS);
U6 : segment7 port map(TQo, SHIF);--hiniti
U7 : segment7 port map(TQp, SHIS);



end RTL;
```

4.3.3 の記述は実際のカウントダウン時計の内部設定をしている。日の部分は９９から０まで、時間の設定は２３から０までのカウントダウンである。分、秒は５９から０までのカウントダウンとなっている。これら全てを１０進カウンタのみで書き上げている。また、データと端子の入出力の設定をし、７セグメント部や１０進カウンタ部への接続も記述している。

4.3.4　7 セグメントディスプレイのテストベンチ記述

```
--------------------------------------------------------
-- Auto-generated test bench template for: SEGMENT7
--
-- Note: comments beginning with WIZ should be left intact.
--       Other comments are informational only.
--
-- Multisim VHDL software version: 5.15a
--
library ieee;
use ieee.std_logic_1164.all;

use std.textio.all;
use work.SEGMENT7;

entity TESTBNCH is
end TESTBNCH;
```

```
architecture stimulus of TESTBNCH is

component SEGMENT7 is
   port (
   DI : in std_logic_vector (3 downto 0);
   DO : out std_logic_vector (6 downto 0)
    );

end component;

constant PERIOD: time := 100 ns;

signal DI : std_logic_vector (3 downto 0);
signal DO : std_logic_vector (6 downto 0);

begin
   DUT: SEGMENT7 port map (
       DI,
       DO
        );

 process

  begin
      DI <= "0000"; wait for 75 ns;
      DI <= "0001"; wait for 75 ns;
      DI <= "0010"; wait for 75 ns;
      DI <= "0011"; wait for 75 ns;
      DI <= "0100"; wait for 75 ns;
      DI <= "0101"; wait for 75 ns;
      DI <= "0110"; wait for 75 ns;
      DI <= "0111"; wait for 75 ns;
      DI <= "1000"; wait for 75 ns;
      DI <= "1001"; wait for 75 ns;
       DI <= "0000"; wait for 75 ns;
```

```
    wait ;
  end process;

end stimulus;
```

4.3.4 の記述は 7 セグメントディスプレイの設定をシミュレーションするためにテストベンチ記述したものである。

4.3.5　10 進カウンタのテストベンチ記述

```
-------------------------------------------------------------
-- Auto-generated test bench template for: COUNT10
--
-- Note: comments beginning with WIZ should be left intact.
--       Other comments are informational only.
--
-- Multisim VHDL software version: 5.15a
--
library ieee;
use ieee.std_logic_1164.all;
-- use ieee.numeric_std.all;     -- Note: uncomment this if you
use
                              -- IEEE standard signed or
unsigned types.
-- use ieee.std_logic_arith.all; -- Note: uncomment/modify
this if you use
                              -- Synopsys signed or unsigned
types.
use std.textio.all;
use work.COUNT10;

entity TESTBNCH is
end TESTBNCH;

architecture stimulus of TESTBNCH is
component COUNT10 is
    port (
```

```
      clk: in std_logic;
      reset: in std_logic;
      CYin: in std_logic;
      Q: out std_logic_vector(3 downto 0);
      CYout: out std_logic
   );

end component;
-- Top level signals go here...
signal clk: std_logic;
signal reset: std_logic;
signal CYin: std_logic;
signal Q: std_logic_vector(3 downto 0);
signal CYout: std_logic;
signal clock_cycle : natural :=0;

begin
   DUT: COUNT10 port map (
         clk,
         reset,
         CYin,
         Q,
         CYout
        );

   CLOCK1: process
   begin
      clock_cycle <= clock_cycle + 1;
      clk <= '1'; wait for 30ns;
      clk <= '0'; wait for 30ns;
   end process CLOCK1;

   STIMULUS1: process
   begin

      -- Sequential stimulus goes here...
```

```
        --
        -- Sample stimulus...

        RESET <= '1';  -- Reset the system
        wait for 30ns; -- Wait one clock cycle

        RESET <= '0'; -- de-assert reset

        --
        -- Enter more stimulus here...
        --
        CYin <= '1';
        wait;           -- Suspend simulation
    end process STIMULUS1;

end stimulus;
```

4.3.5 の記述は 10 進カウンタの設定をシミュレーションするためにテストベンチ記述したものである。

4.3.6 日、時間、分、秒の設定のテストベンチ記述

```
-------------------------------------------------------------
-- Auto-generated test bench template for: CDT
--
-- Note: comments beginning with WIZ should be left intact.
--       Other comments are informational only.
--
-- Multisim VHDL software version: 5.15a
--
library ieee;
use ieee.std_logic_1164.all;
use std.textio.all;
use work.CDT;

entity TESTBNCH is
end TESTBNCH;
```

```
architecture stimulus of TESTBNCH is
component CDT is
   port (
       CLK: in std_logic;
       Reset: in std_logic;
       SBF: out std_logic_vector(6 downto 0);
       SBS: out std_logic_vector(6 downto 0);
       SHF: out std_logic_vector(6 downto 0);
       SHS: out std_logic_vector(6 downto 0);
       SJF: out std_logic_vector(6 downto 0);
       SJS: out std_logic_vector(6 downto 0);
       SHIF: out std_logic_vector(6 downto 0);
       SHIS: out std_logic_vector(6 downto 0)
   );

end component;
-- Top level signals go here...
signal CLK: std_logic;
signal Reset: std_logic;
signal SBF: std_logic_vector(6 downto 0);
signal SBS: std_logic_vector(6 downto 0);
signal SHF: std_logic_vector(6 downto 0);
signal SHS: std_logic_vector(6 downto 0);
signal SJF: std_logic_vector(6 downto 0);
signal SJS: std_logic_vector(6 downto 0);
signal SHIF: std_logic_vector(6 downto 0);
signal SHIS: std_logic_vector(6 downto 0);
signal Clock_cycle: std_logic;

begin
    DUT: CDT port map (
          CLK,
          Reset,
          SBF,
          SBS,
```

```
          SHF,
          SHS,
          SJF,
          SJS,
          SHIF,
          SHIS
        );

   CLOCK1: process
     begin
      Clock_cycle <= Clock_cycle + '1';
      CLK <= '1'; wait for 10ns;
      CLK <= '0'; wait for 10ns;
     end process CLOCK1;

   STIMULUS1: process
   begin
       RESET <= '1'; wait for 15ns;
       RESET <= '0';

       wait;          -- Suspend simulation
   end process STIMULUS1;

end stimulus;
```

4.3.6 の記述は、日、時間、分、秒の設定をシミュレーションするためにテストベンチ記述したものである。

これらのテストベンチを実際にシミュレーションした結果をそれぞれ 7 セグメントが図 4.1、１０進カウンタが図 4.2、日、時間、分、秒の設定が図 4.3 となる。シミュレーションするにあたり、出力される波形を見るのが目的であるため、短時間でより多くの表示させるように 1sec = 10ns、15ns、30ns、75ns として使っている

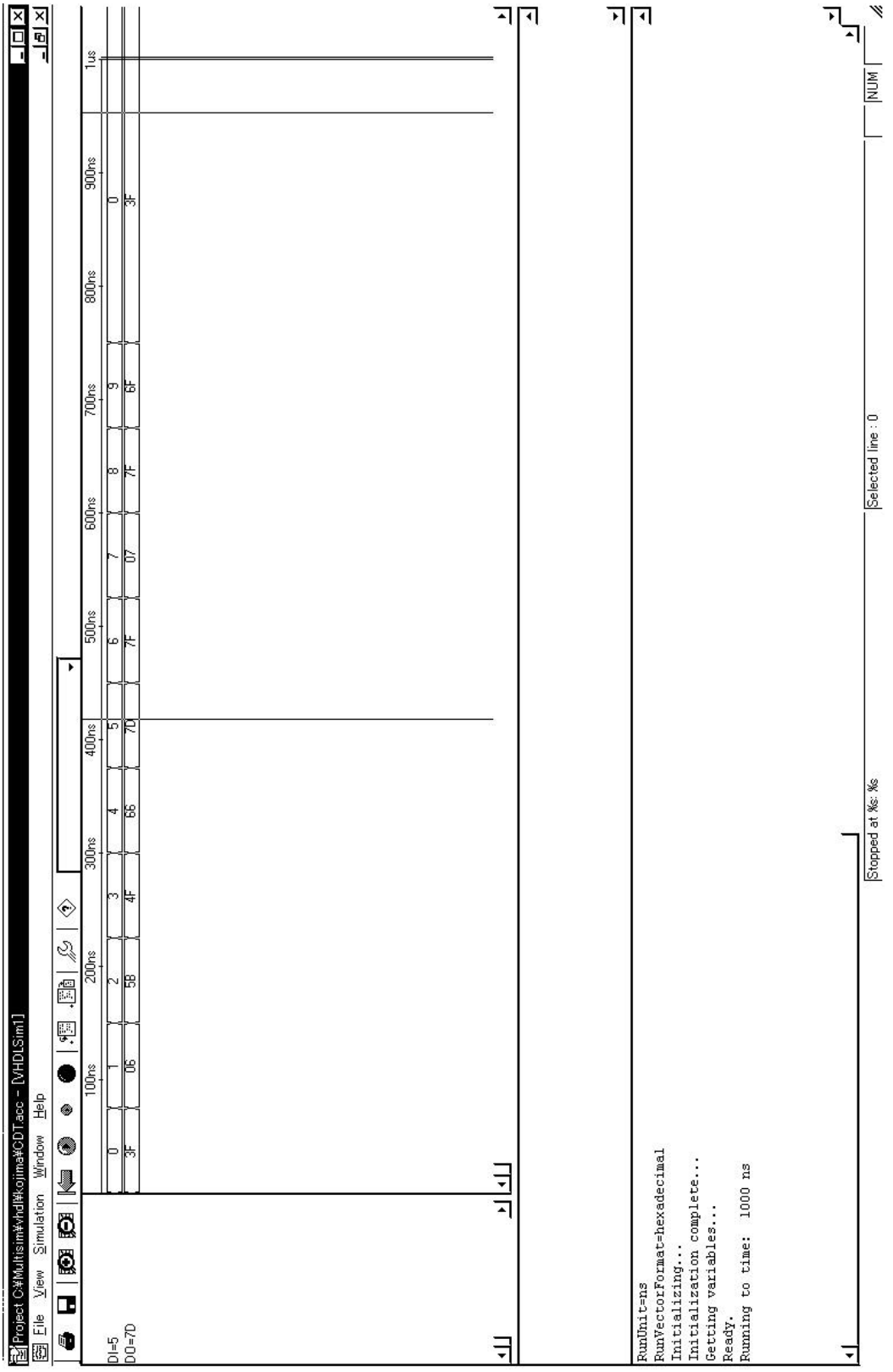


図 4.1　7 セグメントのシミュレーション結果

図 4.2　10 進カウンタのシミュレーション結果

図 4.3　日、時間、分、秒の設定のシミュレーション結果の一部

第５章 おわりに

今回の課題を取り組んだことによって、最初の目的通りディジタル回路の基本である順序回路について、かなり理解することができた。特に最初は難しいと思われた順序回路設計も、実際に一通り体験して見ることで実際の雰囲気が少しつかめた。実際には本当に基本的なことしかできていないが、VHDL、PSpice と言う代表的な回路設計の方法を使うことによって、何とか回路設計ができるようになった。

今後の課題

今回の実験を終えてみて、反省点がいくつかある。

- 限られた時間にもかかわらず、予定をたてそれをこなしていくと言うスケジュール管理がしっかりとできなかった点。
- どの項目についても中途半端な理解で、完全にマスターし使いこなせるレベルまで到達できなかった点。

これらの反省点は今回の実験だけに限らず、今後社会にでても必要とされることなので、しっかりと反省し今後の機会に生かしていきたいと思う。

謝辞

本研究を行うに際し、終始、懇切丁寧な御指導、御鞭撻を賜わりました。高知工科大学 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、懇切丁寧なご指導を賜わりました高知工科大学 電子・光システム工学科学科長 原央 教授、高知工科大学 電子・光システム工学科 河津哲 教授、高知工科大学 電子・光システム工学科 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 電子・光システム工学科専攻 北野克幸氏、小泉義信氏、田口禎譲氏、中村基継氏ほか計算機応用講座の矢野研究室はじめ隣の原研究室、河津研究室、橘研究室の皆様に心から感謝お礼申し上げます。

参考文献

笹尾　勤　著　“論理設計　スイッチング回路理論”　近代科学社
柴山　潔　著　“コンピュータアーキテクチャの基礎”　近代科学社
浜辺　隆二　著　“論理回路入門”　森北出版株式会社
高橋　寛・関根好文・作田幸憲／著　“ディジタル回路”　コロナ社
深山正幸・北川章夫・秋田純一・鈴木正國／著　“HDL による VLSI 設計”共立出版株式会社
吉田たけお・尾知　博／著　“入門！VHDL による回路設計の基礎”
Design Wave Magazine 2000 年 12 月号特集より