卒 業 研 究 報 告

題　　目

順序回路の設計

指 導 教 員

矢野　政顕　　教授

報 告 者

小泉義信

平成　13 年　3 月　2 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

本報告は卒業研究の成果をまとめたものである。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 を使用した順序回路の設計について述べると共に作成例を示している。

第４章では、新しい設計手法であるハードウェア記述言語による設計手法と、その実際例として設計したカウントダウン時計について報告する。

最後の５章で全体をまとめる。

第２章　順序回路

ある時間ｔの出力がその時間ｔの入力だけでなく入力の履歴にも依存する論理回路を順序回路という。

現在、順序回路はコンピュータの CPU だけでなく様々な機能を動かすために用いられている。カウンタ、レジスタ、シフトレジスタなど、現在のコンピュータになくてはならないというよりもむしろ常識的なものになっている。

2.1 基本順序回路

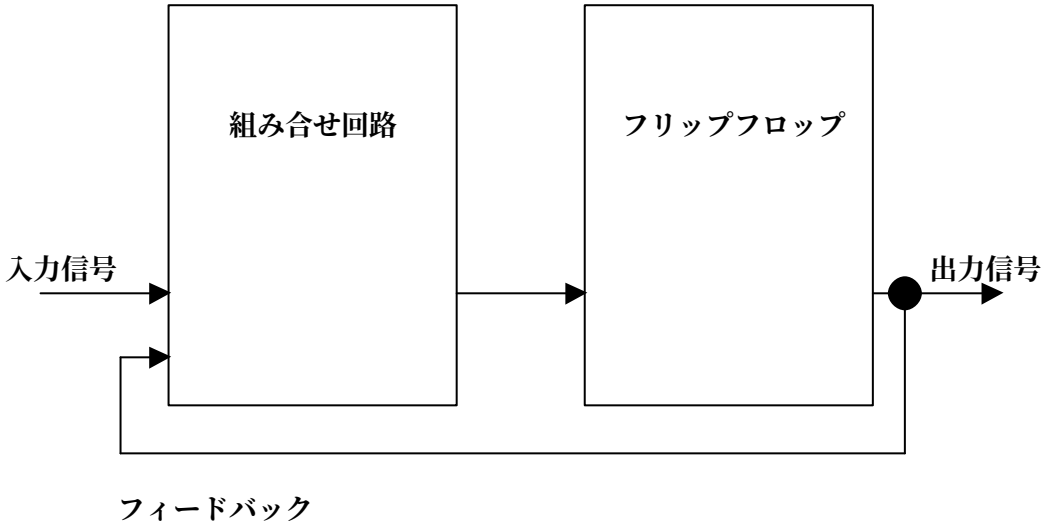


図 2.1.1 基本順序回路

2.1.1 メモリ

順序回路には、入力の履歴を保存するためにメモリが必要となる。メモリ（記憶素子）のことを遅延素子と言う。簡単なメモリは 2 値の論理値を記憶する機能だけを持てばよい。1 ビットメモリはフリップフロップで実現する。

2.1.2 基本順序回路

基本的な順序回路は組み合わせ回路とメモリとから構成される。順序回路は回路動作のタイミングによって、同期式順序回路と非同期式順序回路とに分けることができる。

2.1.3 クロック

順序回路における動作のタイミングをとるパルス信号がクロックである。クロックによって順序回路のメモリの状態を変更することができる。

2.1.4 同期式順序回路

回路動作がクロックに同期している順序回路を同期式順序回路という。同期式順序回路では、メモリは 1 クロックパルス入力に対して 1 回だけ状態を変え得る。したがって、入力が不変でも状態変化によって出力が変わり得る。

同期式順序回路の記述では、状態遷移図（表）によって表すことができる。状態遷移図においては、ノードが状態を、リンクが状態遷移を表す。

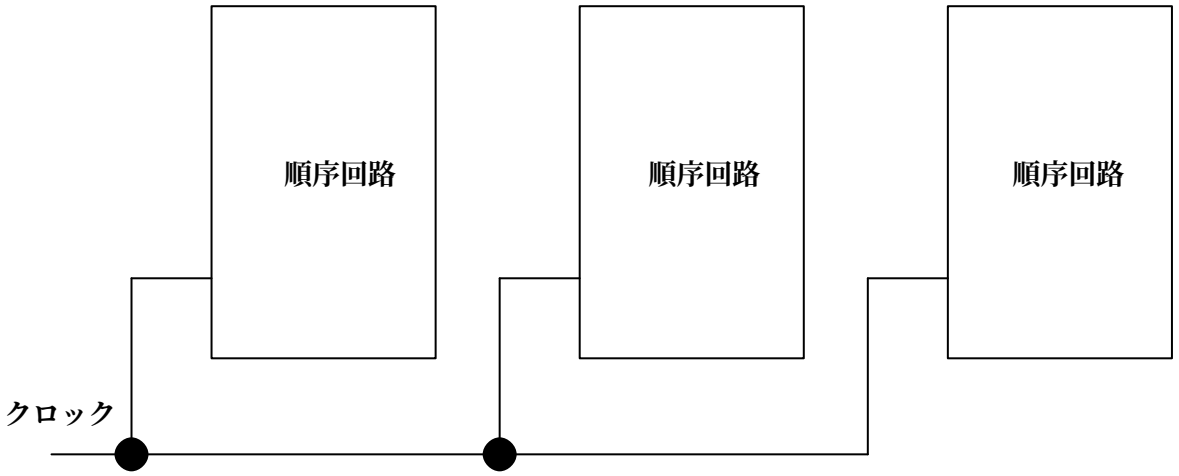


図 2.1.2 同期式順序回路

※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図を用いる。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。この移りいく状態を表にしたものを状態遷移表といい、移りゆく状態を理解しやすくするために図示したものを状態遷移図という。

2.1.5 非同期式順序回路

回路動作が任意の時刻に発生する入力変化、およびその順序だけに依存する順序回路を非同期式順序回路という。最後の入力変化の影響が落ち着くことを安定という。

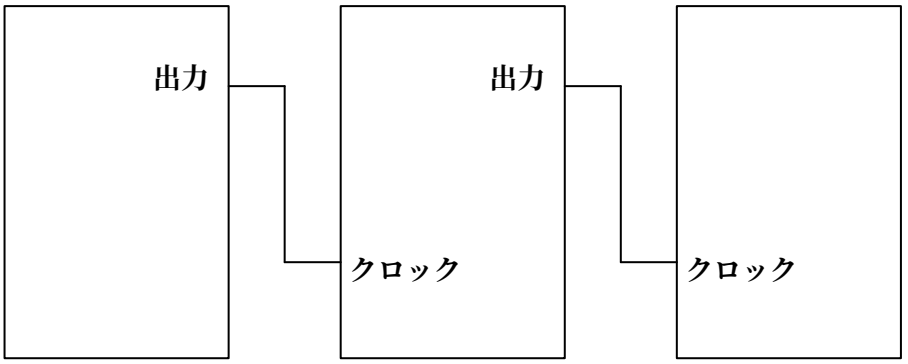


図 2.1.3　非同期式順序回路

2.2 フリップフロップ

1 ビットのメモリは 0 と 1 のいずれかの安定状態をもつもので、これをフリップフロップという。順序回路を構成する重要な部品であるメモリは多数のフリップフロップによって構成されている。

フリップフロップの状態変化は入力による。ある論理値とその論理が逆の値の両方を出力することができる。最小構成のフリップフロップは、2 個の NAND 素子で構成される。SR フリップフロップ入力は、双安定状態間の切り替えを可能とする同期信号と考えることができる。

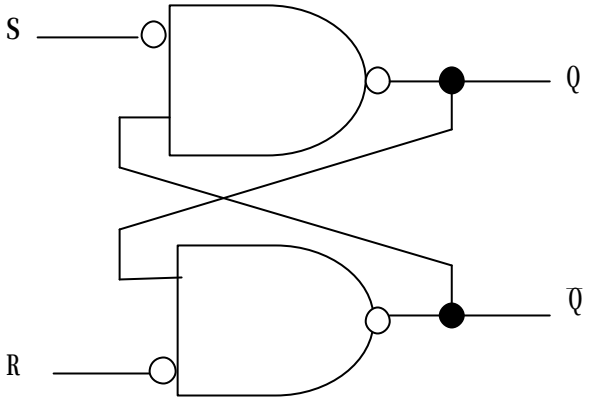


図 2.2.1　FF の基本構造（SR フリップフロップ）

2.2.1 ＳＲフリップフロップ

SR フリップフロップは文字どうり、入力としてセット S とリセット R を持つ基本的な FF である。基本構造は前ページ　図 2.2.1 に示している。

入力 S に 1 が印加された時、出力 Q を 1 とし、入力 R に 1 が印加されたとき出力 Q を 0 とする。入力 SR がともに 0 のときは以前の状態を保つ。また、入力 SR がともに 1 のときは、Q、$\overline{Q}$ともに、1 になるが、そもそも$\overline{Q}$は、Q の否定でなければならないので、この入力の組合せは禁止される。

表 2.2.1 遷移図

S	R	Q
0	0	Q
0	1	0
1	1	×
1	0	1

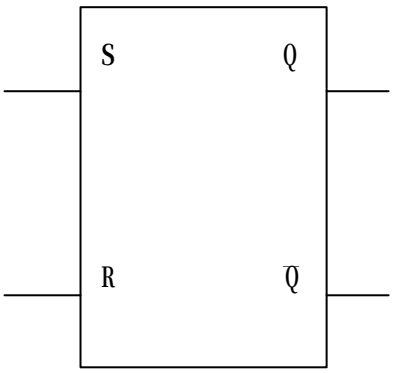


図 2.2.2 SR フリップフロップの記号

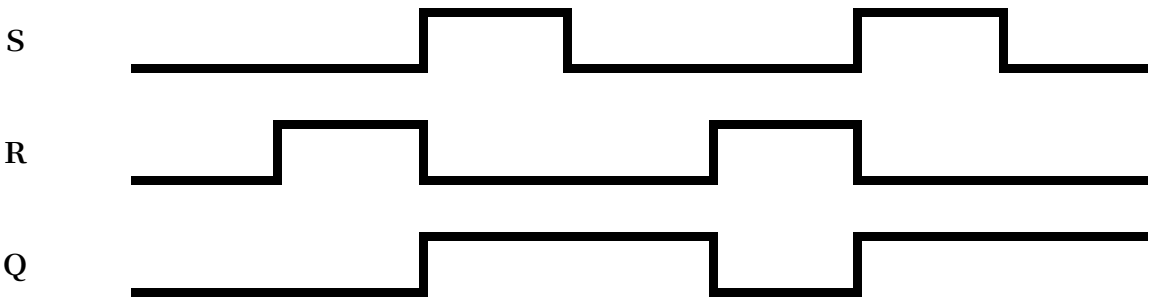


図 2.2.3 SR フリップフロップのタイムチャート

2.2.2 クロック付き SR フリップフロップ

S/R 入力とも 0 のとき、Q 出力は不変。S/R 入力とも 1 のとき、出力は不定。S/R 入力のいずれかが 1 の時、Q 出力は強制的に 1（set)、0（reset）という動作をする。

表 2.2.2 遷移図

S	R	Q+
0	0	保持
0	1	reset
1	0	set
1	1	×

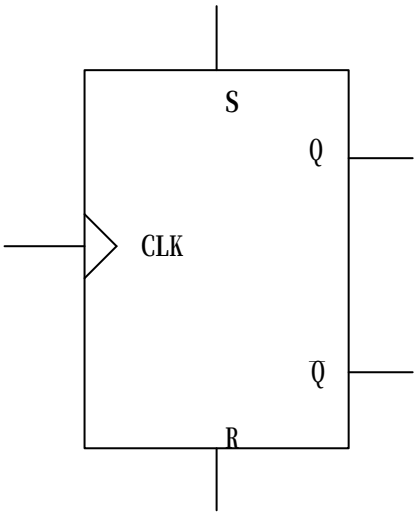


図 2.2.4　クロック付き SR フリップフロップの記号

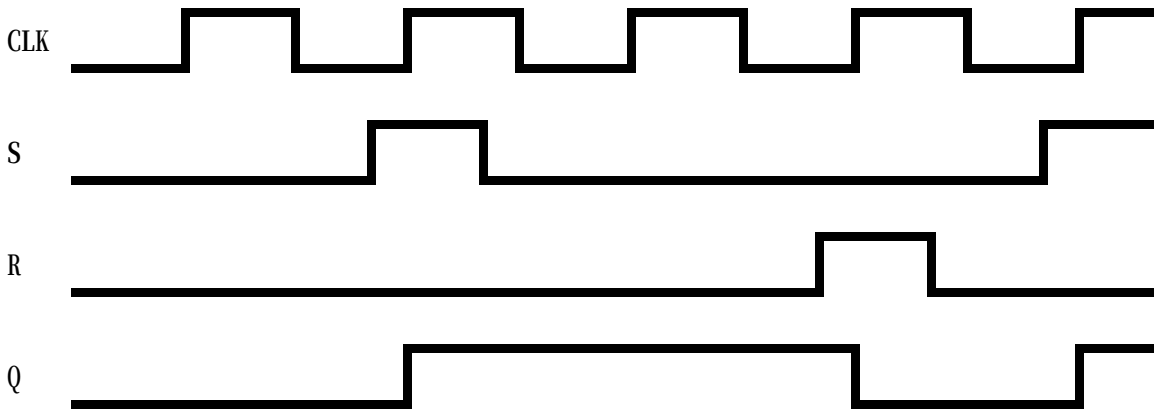


図 2.2.5 クロック付き SR フリップフロップのタイムチャート

2.2.3 D フリップフロップ

Q 出力は D 入力のみに依存し、状態信号の取り込み保存用として利用できる。

D フリップフロップは CLK 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。

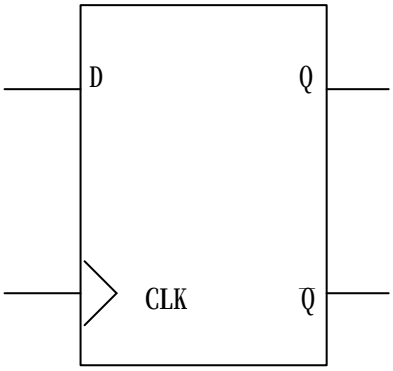


表 2.2.3 遷移図

D	Q+
0	0
1	1

図 2.2.6D フリップフロップ

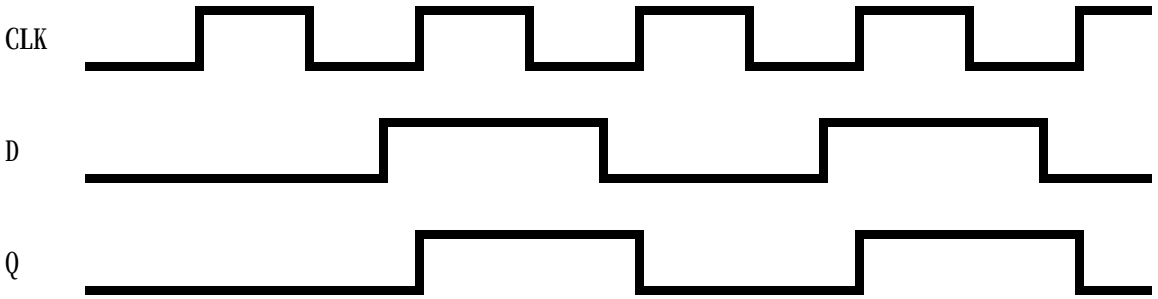


図 2.2.7 D フリップフロップのタイムチャート

2.2.4 JK フリップフロップ

SR フリップフロップの出力を入力にフィードバックし、J/K 入力を組み合わせたフリップフロップである。

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、J/K 入力ともに 1 のとき、Q 出力が反転という動作をする。SR フリップフロップ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。

JK フリップフロップは、SR フリップフロップに CLK を追加して SR フリップフロップの禁止入力 SR=0 を改善したものである。SR フリップフロップは S,R に同時に入力することを禁止しているが、JK フリップフロップは J,K に同時に入力した場合でも CLK に同期して安定した出力が得られる。すなわちこの場合は Q+の出力として現在の状態 Q の反転した状態が得られる。フリップフロップの次の状態 Q+は CLK 入力がある時のみの影響を受けるので、CLK が無いときの Q+は現在の状態のままである。JK フリップフロップの動作は、入力 J または K のそれぞれを SR フリップフロップの入力 S または R に対応させると、反転出力以外の Q+出力が SR フリップフロップの遷移表と一致するので理解しやすい。

JK フリップフロップには常に CLK=1 のとき動作する。

SR フリップフロップと異なるところは、JK フリップフロップは常に CLK に同期して使用することである。

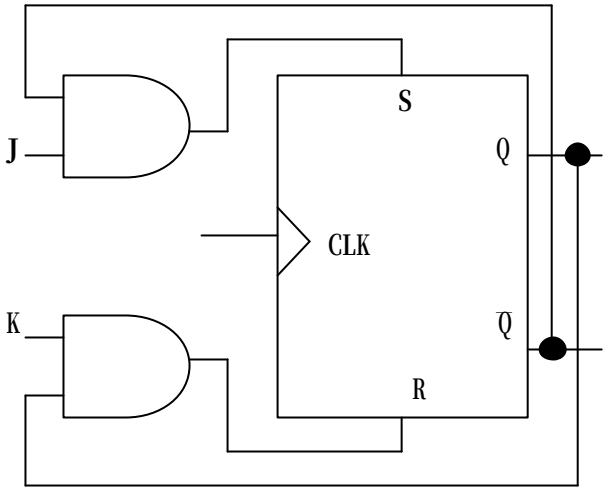


図 2.2.8 JK フリップフロップの記号

表 2.2.4 遷移図

J	K	Q+
0	0	保持
0	1	Reset
1	0	Set
1	1	$\overline{Q}$

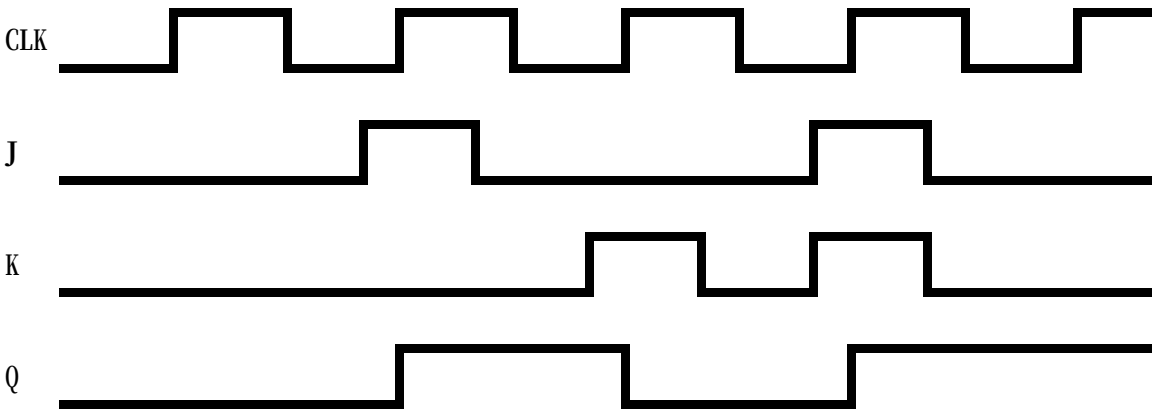


図 2.2.9 JK フリップフロップのタイムチャート

2.2.5 T フリップフロップ

JK フリップフロップの J 入力と K 入力を結合し T 入力としたフリップフロップである。

T 入力が 0 のとき、状態は不変。T 入力が 1 のとき、状態は反転という動作をする。T フリップフロップとは、T 入力があるたびに Q 出力が反転動作するフリップフロップである。

表 2.2.5 遷移図

T	0	1
Q_+	Q	$\overline{Q}$

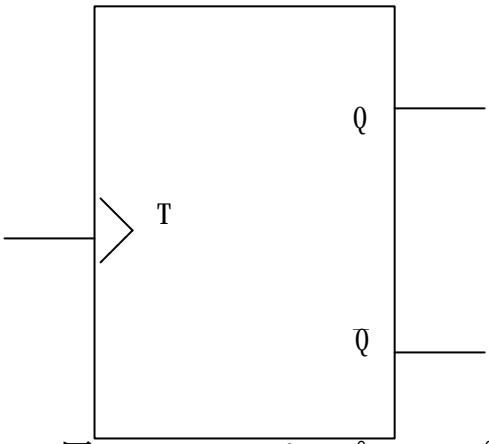
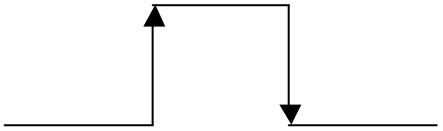


図 2.2.10 T フリップフロップの記号

図 2.2.11 T フリップフロップのタイムチャート

2.2.6 マスタースレーブ型フリップフロップ

クロックの立ち上がり（下がり）と立ち下がり（上がり）の組で 1 つの動作を行うフリップフロップをマスタースレーブ型フリップフロップという。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のものがある。



マスター　　スレーブ

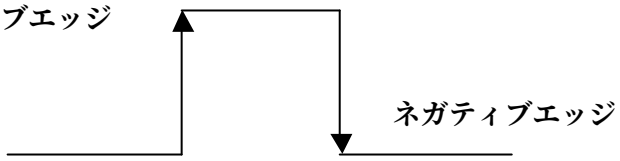
2.2.7 エッジトリガ型フリップフロップ

状態変化がクロックパルスの立ち上がり・立ち下がりによって生じるフリップフロップをエッジトリガ型フリップフロップという。

? マスタスレーブ型とエッジトリガ型

マスタスレーブ型は基本的にフリップフロップを 2 段直列に接続し、それらを互いに逆相の信号で動作させるものである。前段のフリップフロップをマスタ、後段をスレーブと呼ぶ。マスタスレーブ型は逆相の信号によって、マスタとスレーブを交互に動作させることによりフィードバックループを切断している。

エッジトリガ型は、クロックのエッジの時点で動作を行うようにしたもので、入力の立ち上がりまたは立ち下がりで出力を決定し、それ以降の入力は出力に影響しないようにする。



2.3 ラッチとレジスタ

フリップフロップの集まりをラッチとかレジスタという。レジスタ長とは、並べたフリップフロップ個数のことであり、各フリップフロップは同期して動作する。すなわち、ラッチやレジスタの動作にはフリップフロップ出力の読み出しとフリップフロップへの書き込み/状態更新とがあり、各ビットは同時に動作する。

2.3.1 レジスタ

レジスタとは置数器のことで、1 バイトまたは 1 ワードの大きさでデータの一時記憶として使用する。例えば演算装置の汎用レジスタ、メモリのアドレスを指定するアドレスレジスタおよびメモリのデータを入出力するときの一時記憶に使用されるメモリレジスタなどがある。これらは並列に入力したデータをそのまま出力すればよいので D フリップフロップの機能があればよい。

2.3.2 シフトレジスタ

シフトとは、桁移動のことである。10 進数を 1 桁左シフトすると元の１０倍、右シフトすると 10 分の 1 になるのと同様に 2 進数を 1 桁左シフトすると 2 倍、右シフトすると 2 分の 1 になる。このシフト機能をもつレジスタをシフトレジスタといい、CP に同期して全体のビットが左または右にシフトする。この機能は乗算や除算に応用できるので、演算装置の汎用レジスタやアキュームレータ（累算器）に付加して使用する。

2.4 シフタ

あるフリップフロップ出力を別のフリップフロップ入力とし、それらの相互に連結したフリップフロップに共通のクロックを入力することによって同期して動作するフリップフロップ群をシフタという。ラッチやレジスタは共通クロック入力をもつだけでビットが相互に独立しているのに対して、シフタではビットが連結されて相互に影響しあう点が特徴的である。

シフタ全体としての機能は、シフト方向と、シフトビット数によって決まる。複数ビットへの並列入出力が可能であり、直列⇔並列の相互変換や固定小数点乗除算などに応用される。

2.4 カウンタ

カウンタには、クロックパルスに同期してあらかじめ決められた順序で状態変化を起こす同期式順序回路と、クロックパルスに同期することなく動作している非同期式順序回路がある。カウンタはシフタの一種であるが、隣接するフリップフロップ間を直結するのではなく簡単な組み合わせ回路を挿入することによって決められたビットパターンとしての状態変化を生起させる点が異なる。

2.4.1 同期式カウンタ

カウンタを構成するすべてのフリップフロップに同一のクロックが入力され、これらすべてのフリップフロップが同時に状態変化するカウンタを同期式カウンタという。

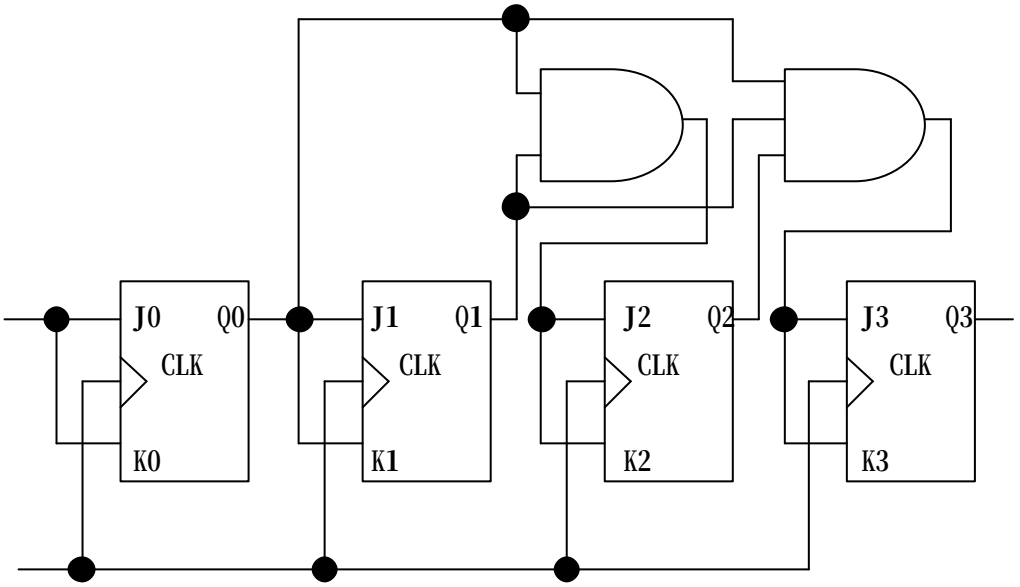


図 2.4.1ＪＫフリップフロップによる同期式カウンタ（4 ビット 2 進カウンタ）

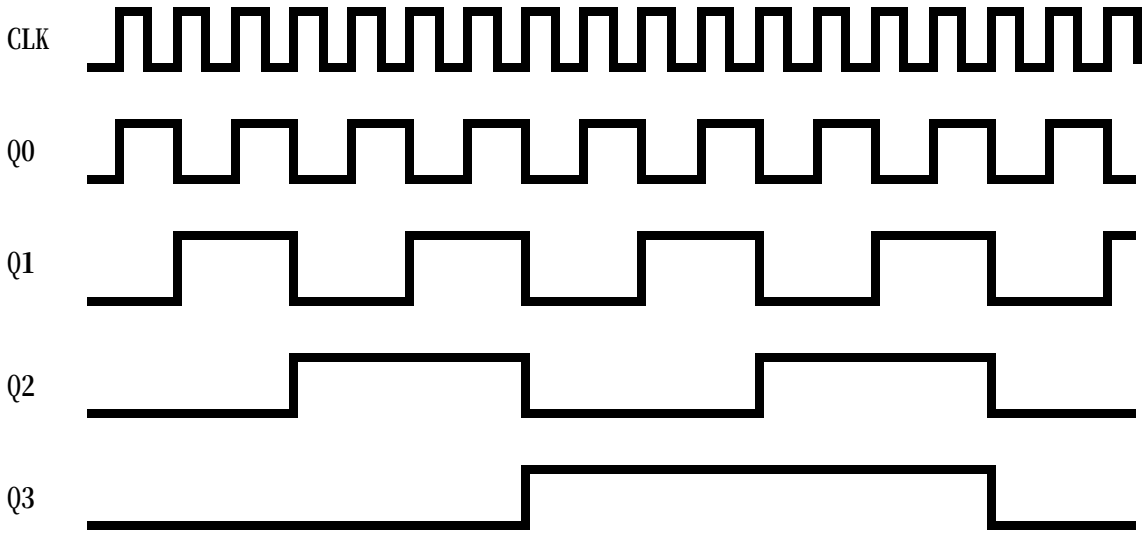


図 2.4.2 JK フリップフロップによる同期式カウンターのタイムチャート

2.4.2 非同期式カウンタ

各順序回路が共通のクロックに同期しておらず、別々のタイミングで動作しているカウンタである。最下位段以外の各フリップフロップは、前段の出力によってトリガされる。

カウントパルスが最下位段から最上位段まで順番に伝搬する。各フリップフロップの伝搬遅延時間がカウンタの動作を左右しフリップフロップは同期しないので、非同期式カウンタをリプルカウンタということがある。

リプルカウンタでは、カウントパルスが最下位段から最上位段まで順番に伝播する。桁上げ信号（キャリー）が前段（下位の桁）から後段（上位の桁）のフリップフロップへ順番に伝わっていく構造のため、後段のフリップフロップの状態が確定するまでに時間がかかる。

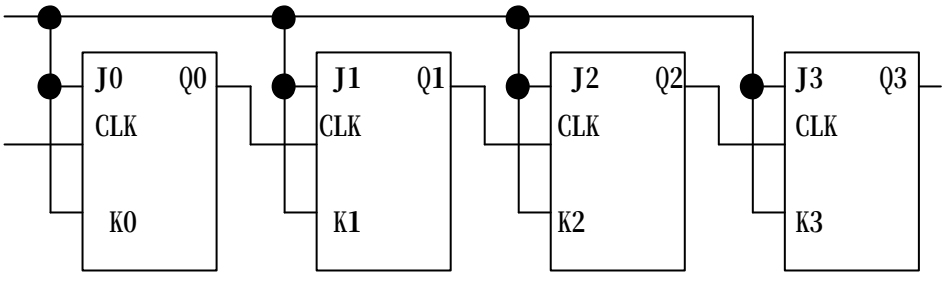


図 2.4.3 ＪＫフリップフロップによる非同期の記号

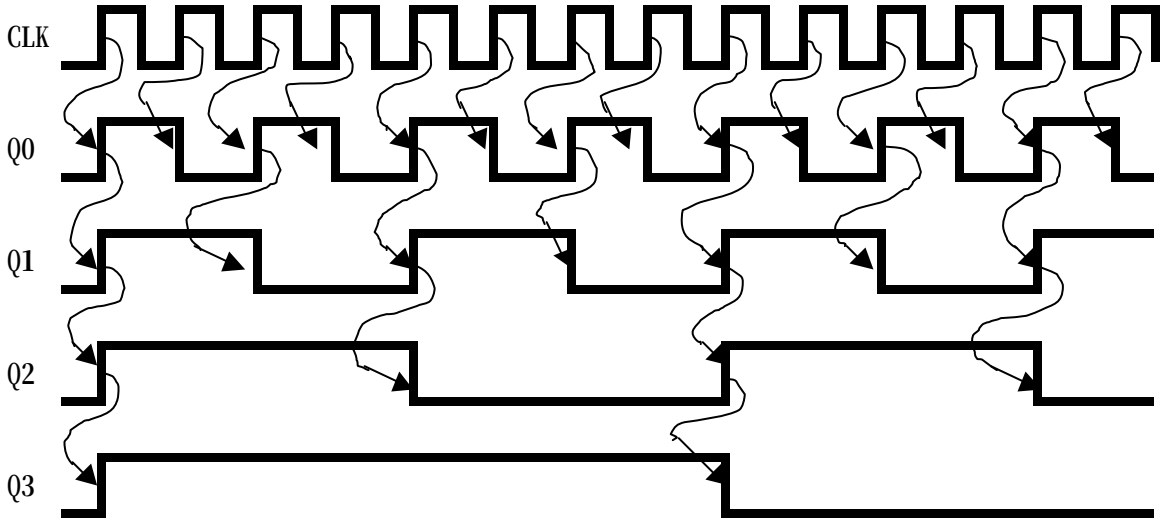


図 2.4.4 ＪＫフリップフロップによる非同期式のタイムチャート

2.5 フリップフロップを用いた種々の順序回路

実際にコンピュータのハードウェア機構を構成する基本論理回路として利用され、フリップフロップやカウンタを用いて構成できる順序回路について述べる。

2.5.1 クロック生成回路

同期式順序回路の状態変化を直接生成させるクロックを発生する。クロックの生成回路は固有の発信周波数をもつ水晶発振子などを中心に構成される。

2.5.2 タイミング生成回路

クロック生成回路より得られるマスタクロックから位相のずれたタイミングパルスを生成する回路をタイミング生成回路という。タイミングパルスとは、クロック周波数を分周した周波数をもつが、パルス幅はマスタクロックと同じパルスのことをいう。

2.5.3 パルスエッジ検出回路

クロックと非同期のパルスのエッジを検出する回路をパルスエッジ検出回路といい、非同期信号の同期化などのコンピュータの基本回路として特に制御回路の構成には、不可欠なものである。信号や変数間に演算を施したものなどである。

第３章　標準論理 IC による
順序回路の設計

3.1　標準論理ＩＣ

ディジタルＩＣを回路素子で見ると、
（1）バイポーラトランジスタを用いたＩＣ
（2）ユニポーラトランジスタのＭＯＳＦＥＴを用いたＩＣ
（3）バイポーラトランジスタとＭＯＳＦＥＴとが混在したＩＣがある。

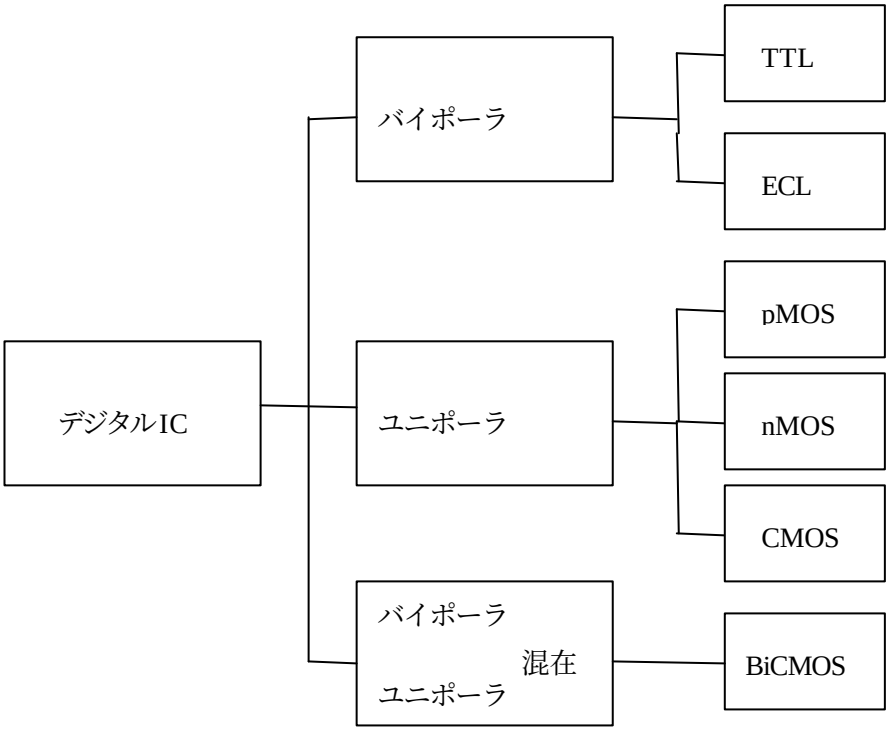


図 3.1.1　デジタル IC の回路素子別の分類

バイポーラトランジスタを用いた TTL IC は、集積度は低いが、比較的高速に動作する。このため TTL は、組み合わせて様々な回路を構成する標準ロジック IC として広く用いられてきた。TTL には、速度の向上や消費電力の低下を求めて、開発された多くの品種が存在する。

バイポーラトランジスタを用いた IC には ECL IC もある。ECL はバイポーラトランジスタを非飽和で使用するため、高速に動作させることができる IC であるが、一般に消費電力が大きいため用途が限られてくる。

これに対してユニポーラトランジスタである MOSFET を用いた IC は、集積度が高く消費電力が少ないという特長がある。pMOS か nMOS のいずれかで構成させた MOS 系ディジタル IC もあったが、現在では、CMOS で構成されているものが多い。CMOS は p チャンネルの MOSFET と、n チャンネルの MOSFET を相補的に用いたもので、回路構成が簡単で、消費電力が極めて小さい。当初、TTL と比較して低速であった CMOS 標準ロジック IC も、現在では TTL に匹敵する高速なものが開発されている。

IC ファミリ

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 ファミリと呼ぶ。汎用ロジック IC のファミリにはいくつかの種類があるが、一般的に、ロー・パワー・ショットキーTTL（LS　TTL）と CMOS の 74HC 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 74LS の場合は、

□□74LS×××（□：英数字×：数字）

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 の機能を表します。×××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。

表 3.1.1 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	富士通のディジタルＩＣ
HD	日立のディジタルＩＣ

表 3.1.2　型名

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

表 3.1.3　TTL ファミリと特徴

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

表 3.1.4　CMOS ファミリと特徴

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
74 ＡＬＳ	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

3.2　カウンタ用ＩＣ

順序回路の設計にはカウンタ IC が良く用いられる。ここでは 4 ビットカウンタについて取り上げる。

表 3.2.1　TTL の 4 ビットカウンタ用 IC の型番とその特徴

TTL(同期式)の型番	特徴
SN74160	同期プリセット、非同期クリア
SN74161	同期プリセット、非同期クリア
SN74162	同期プリセット、同期クリア
SN74163	同期プリセット、同期クリア
SN74168	BCD、同期プリセット、クリアなし、同期式可逆カウンタ
SN74169	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	BCD、非同期プリセット、クリアなし、同期式可逆カウンタ
SN74191	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	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	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	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	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
SN74668	バイナリー/デコードカウンタ
SN74669	バイナリー/デコードカウンタ
TTL(非同期式)の型番	
SN7490	10 進、非同期プリセット/クリア
SN7493	16 進、非同期クリア
SN74390	7490×2、プリセットなし
SN74393	7493×2
SN74490	7490×2

表 3.2.2　CMOS の 4 ビットカウンタ用 IC の型番とその特徴

CMOS の型番	特徴
CD40102	同期式 4 ビット BCD　DOWN×2、カスケード接続で拡張可能
CD40160	同期式 4 ビット、10 進、非同期クリア端子つき
CD40161	同期式、40160 のバイナリ―型
CD40162	同期式、40160 の同期クリア端子つき
CD40163	同期式、40160 のバイナリ―型で同期クリア端子つき
CD40192	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
CD40193	4 ビットバイナリ―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
CD4526	同期式プログラマブル 4 ビットバイナリ―、非同期プリセット/クリア
CD7493	4 ビットバイナリ―リプルカウンタ、非同期式マスターリセットつき
CD74161	同期式、バイナリ―カウンタ、非同期式リセット
CD74163	同期式、バイナリ―カウンタ、同期式リセット
CD74190	同期式、UP/DOWN、BCD デコードカウンタ
CD74191	同期式、UP/DOWN、バイナリ―カウンタ
CD74192	同期式、UP/DOWN、バイナリ―カウンタ
CD74193	同期式、UP/DOWN、バイナリ―カウンタ

3.3 設計例　―　カウントダウン時計　―

設計したカウントダウン時計は、指定した日数、時間、分、秒をどんどんカウントダウンしていく時計である。

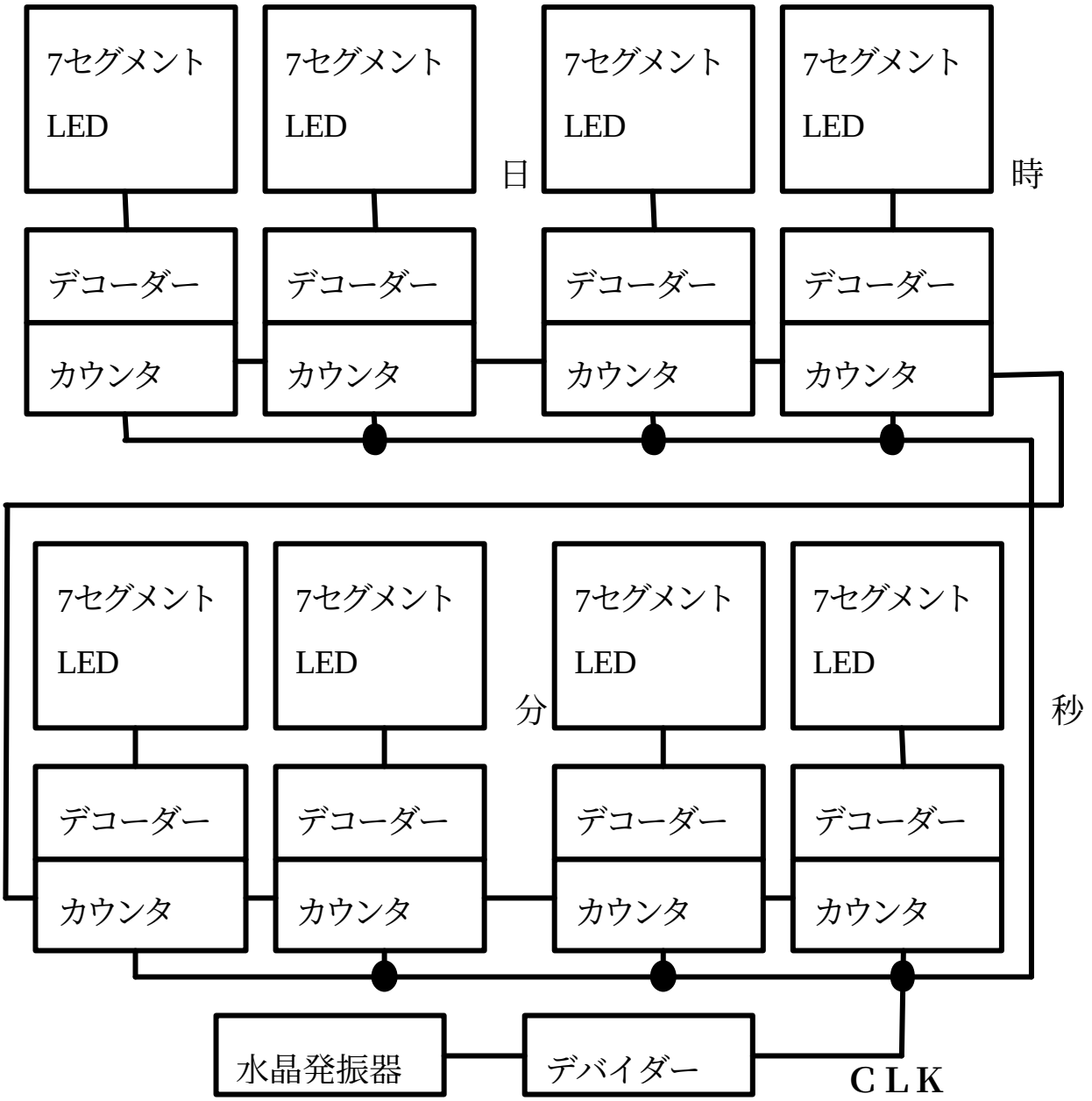


図 3.3.1　カウントダウン時計のブロック図

使用したもの。

（１）抵抗

10kΩ（スイッチ）		4本
220Ω（セグメント用）		56本

（２）ＩＣ

HD74LS247P	（セグメント用）	8個
SN74LS169BN	（カウントダウン用）	8個
SN74LS08N	（AND回路用）	4個
SN74LS292N	（クロック用）	1個
SN74LS00N	（NAND回路用）	1個

（３）水晶（TOYOCOM TCO-707F 4.194304MHz）

（４）セグメント　8個

（５）電解コンデンサ（470μＦ）　2個 電源用

（６）セラミックコンデンサ（0.1μＦ）　4個 スイッチ用

（７）プッシュスイッチ　4個

使用したソフト

P-Spice

Schematics・P-Spice A/D・Probe

使用したＩＣの説明

・HD74LS247P

表 3.3.1　HD74LS247P の電気特性

			SN74LS247P		
			MIN	NOM	MAX
VCC	Supply voltage		4.75	5	5.25
VO	Off-state output voltage	a thru g			15
IO	On-state output current	a thru g			40
IOH	High-level output current	$\overline{BI}$ ／ $\overline{RBO}$			-200
IOL	Low-level output current	$\overline{BI}$ ／ $\overline{RBO}$			8
TA	Operating free-air temprature		0		70

表 3.3.2 HD74LS247P の状態表

10 進数	入力				7 セグメントデコード出力							表示
	A	B	C	D	a	b	c	d	e	f	g	
0	0	0	0	0	0	0	0	0	0	0	1	0
1	0	0	0	1	1	0	0	1	1	1	1	1
2	0	0	1	0	0	0	1	0	0	1	0	2
3	0	0	1	1	0	0	0	0	1	1	0	3
4	0	1	0	0	1	0	0	1	1	0	0	4
5	0	1	0	1	0	1	0	0	1	0	0	5
6	0	1	1	0	0	1	0	0	0	0	0	6
7	0	1	1	1	0	0	0	1	1	1	1	7
8	1	0	0	0	0	0	0	0	0	0	0	8
9	1	0	0	1	0	0	0	0	1	0	0	9

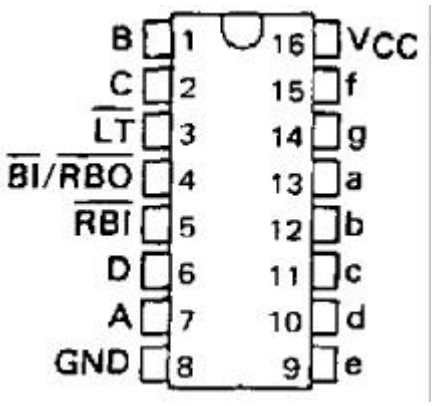


図 3.3.2 HD74LS247P の IC のピン配置図

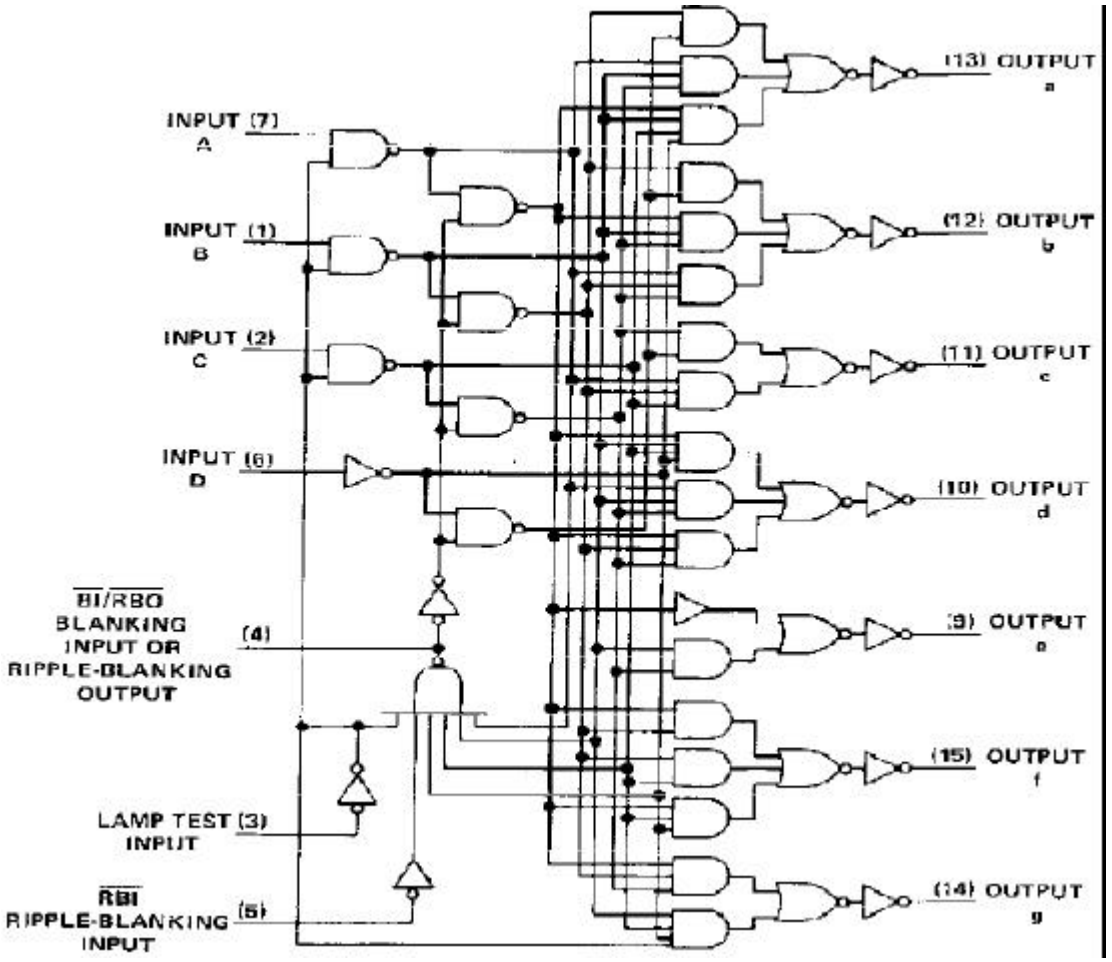


図 3.3.3 HD74LS247P の論理回路図（http://www.tij.co.jp より参照）

・SN74LS169BN

同期式４ビット　アップダウン　バイナリ―カウンター

表 3.3.3 SN74LS169BN の電気特性

SN74LS169BN						UNIT
			MIN	NOM	MAX	
VCC	Supply voltage		4.5	5	6	V
VIH	High-level input voltage		2			V
VIL	Low-level input voltage				1	V
IOH	High-level output current				-0	mA
IOL	Low-level output current				8	mA
Fclock	Clock frequency		0		40	MHz
Tw	Pulse duration,CLK high or low		12.5			ns
Tsu	Setup time before CLK↑	A,B,C,or D	15			ns
		ENP or ENT	15			
		LOAD	15			
		U/D	15			
Tsu	Hold timw,data after CLK↑		0			ns
TA	Operating free-air temprature		0		70	℃

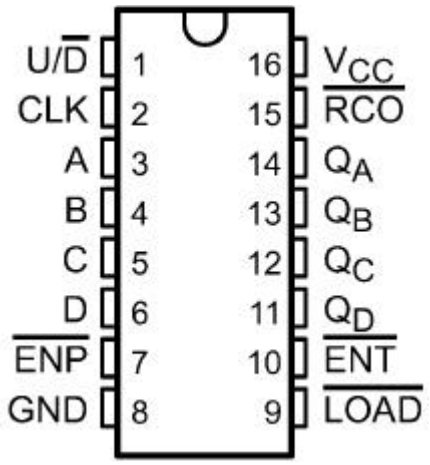


図 3.3.4 SN74LS169BN の IC のピン配置図

１番にハイレベルの入力で、カウントアップし、ローレベルの入力でカウントダウンさせることが出来る。

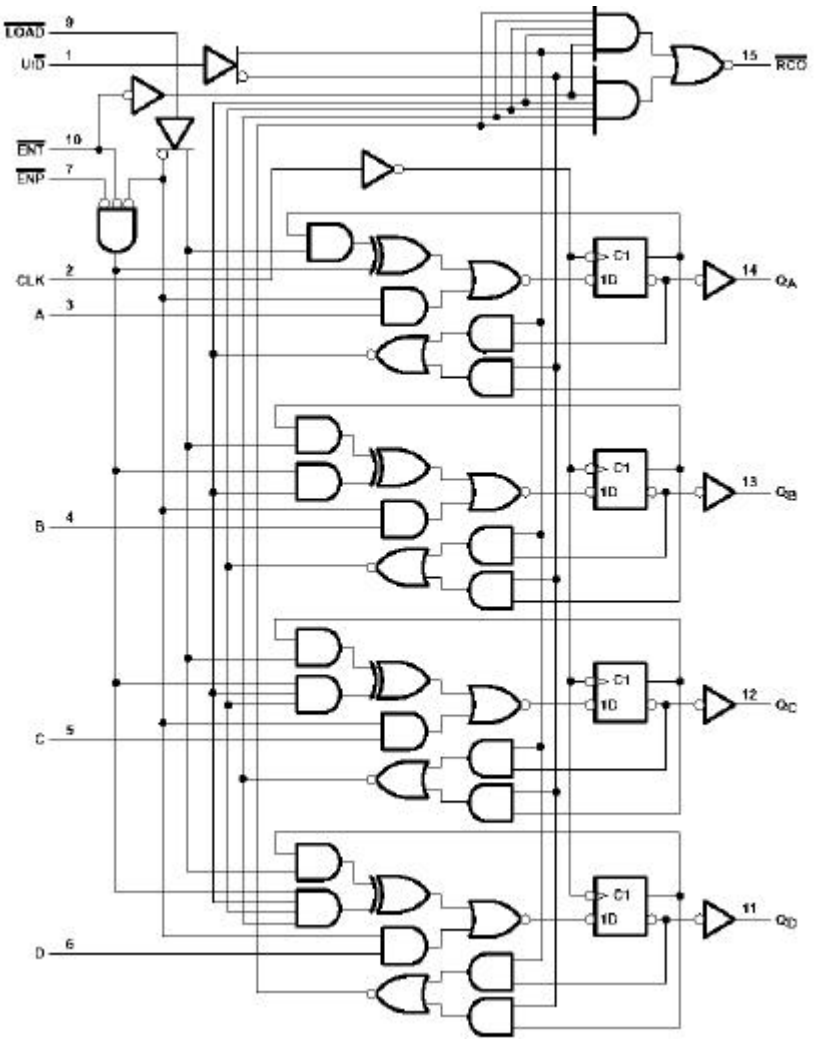


図 3.3.5 SN74LS169BN の論理回路図（http://www.tij.co.jp より参照）

・SN74LS292N（クロック用）

表 3.3.4 SN74LS292N の電気特性

		SN74LS292N		
		MIN	NOM	MAX
VCC	Supply voltage	4.75	5	5.25
VIH	High-level input voltage	2		
VIL	Low-level input voltage			0.8
IOH	High-level output current			-1.2
IOL	Low-level output current			24
Fclock	Clock frequency	0		30
Tw	Pulse duration,CLK high or low	16		
Tsu	Setup time sefore CLK	55		
Th	Hold time,data after CLK	15		
TA	Operating free-air temprature	0		70

表 3.3.5 SN74LS292N の状態表

CLEAR	CLK1	CLK2	Q OUTPUT MODE
L	×	×	Cleared to L
H	↑	L	Count
H	L	↑	Count
H	H	×	Inhibit
H	×	H	Inhibit

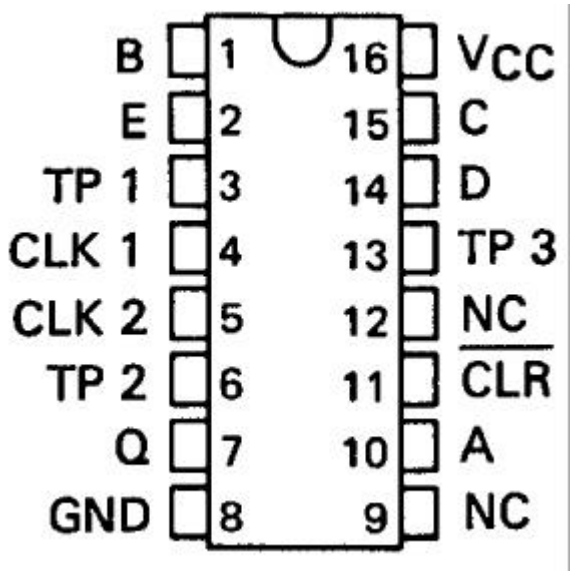


図 3.3.6 SN74LS292N の IC のピン配置図

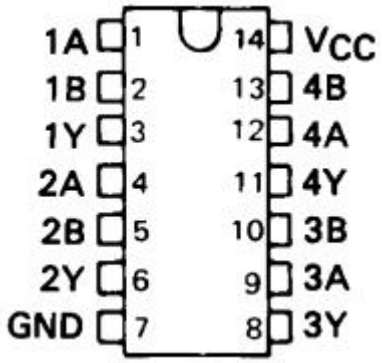
・**SN74LS00(AND)と SN74LS08(OR)**

表 3.3.6 SN74LS00(AND)と SN74LS08(OR)の電気特性

		SN74LS00			SN74LS08		
		MIN	NOM	MAX	MIN	NOM	MAX
VCC	Supply voltage	4.75	5	5.25	4.75	5	5.25
VIH	High-level input voltage	2			2		
VIL	Low-level input voltage			0.8			0.8
IOH	High-level output current			-0.4			-0.8
IOL	Low-level output current			16			16
TA	Operating free-air temprature	0		70			70

・ SN74LS08N(AND)

表 3.3.7 状態表

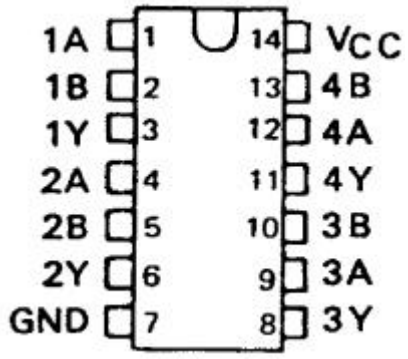


INPUTS		OUTPUT
A	B	Y
H	H	H
L	×	L
×	L	L

図 3.3.7 IC のピン配置図

・SN74LS00N（NAND）

表 3.3.8 状態表



INPUTS		OUTPUT
A	B	Y
H	H	L
L	×	H
×	L	H

図 3.3.8 IC のピン配置図

P-Spice を用いて設計したカウントダウン時計の回路です。

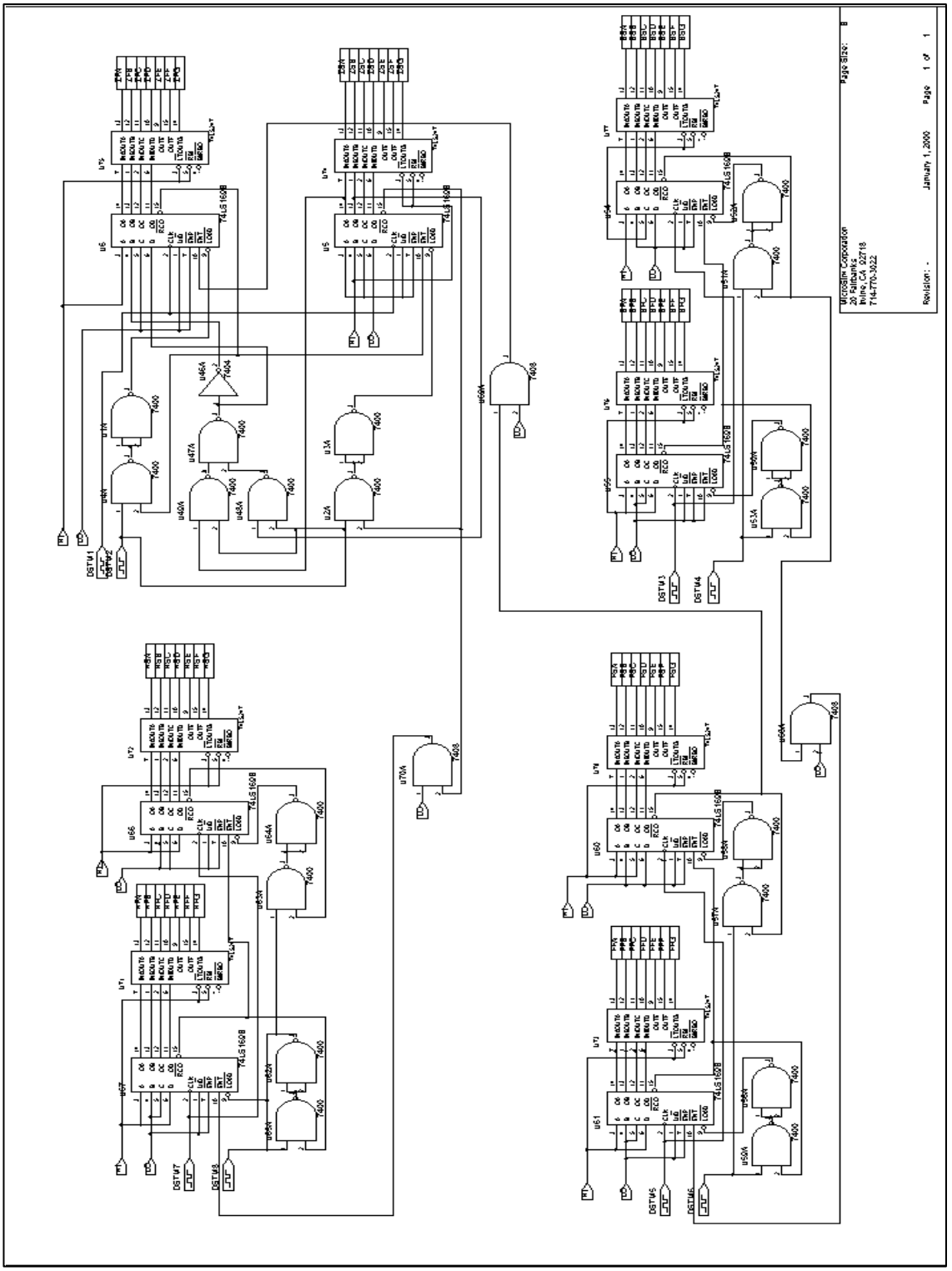


図 3.3.9 カウントダウン時計の回路図

この回路は、シミュレーションのために桁上げ信号をカットし、秒、分、時、日が同時に

動作するようになっている。

シミュレーション結果

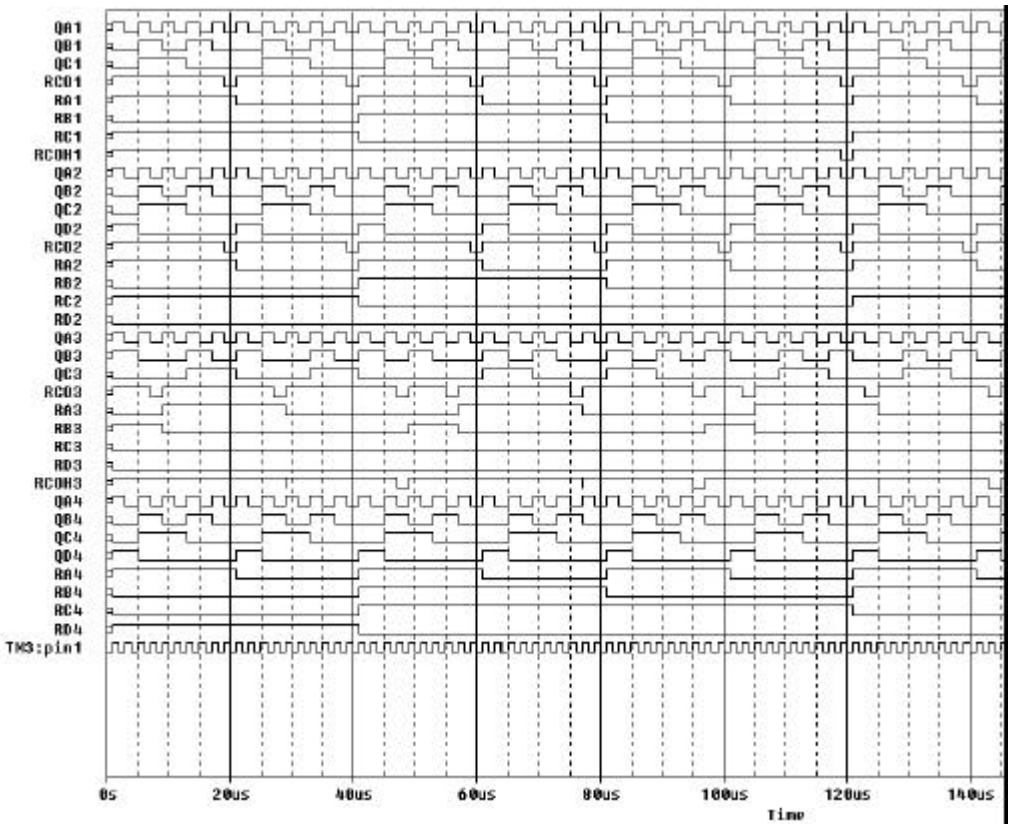


図 3.3.10　カウントダウン時計シミュレーション結果

完成した製品

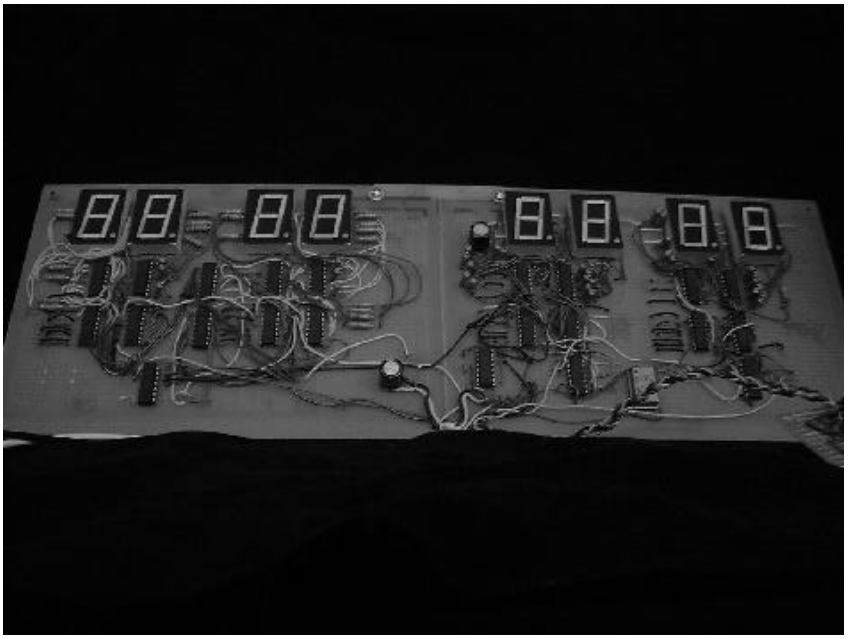


図 3.3.11　カウントダウン時計の作成した製品例

第４章　ハードウェア記述言語による順序回路の設計

4.1 順序回路ハードウェアー記述言語 HDL

ディジタル回路の設計作業の多くは、人間の手作業によるものであったが、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の一途をたどっている。そのため人間の手作業による回路図作成はすでに限界となっている。

そこで、ハードウェア記述言語（HDL）を用いたプログラミング形式のディジタル回路設計が行われるようになっている。

ＨＤＬは、プログラム言語とは異なり、ＨＤＬはハードウェアを設計するうえで欠かすことのできない並列処理や時間、タイミングの概念などを記述できる特徴をもつ。

4.1.1 設計自動化（DA）技術

HDL 技術と実際のディジタル回路とのギャップを埋めるため設計自動化（Design　Automation：DA）の試みも昔から行われてきた。この設計自動化を実現するための重要な技術の一つに論理合成（logic　Synthesis）がある。論理合成は、あるレベルの HDL 記述から、それより低いレベルの HDL 記述を自動設計する技術である。

現在では、機能設計段階の HDL 記述から回路図や LSI のマスクパターンを自動生成できるようになってきている。

このように、論合成技術の進歩によって、HDL は急速に普及し、HDL を抜きにしたディジタル回路設計は考えられなくなっている。

HDL には代表的なものに、VHDL と Verilog-HDL の 2 種類が存在している。

この 2 つは基本的な概念は一緒である。VHDL は設計仕様のドキュメント言語として開発されたために、読解性に優れ、システム仕様のあいまいさを排除した厳格な言語である。Verilog-HDL は、LSI 設計における記述性を重視した言語であり、記述しやすく、ゲートレベル・シミュレーションのために機能も充実

している

4.1.2 VHDL

VHDL は、ハードウェア記述言語（HDL）の一つであり、米国国防総省において、VHSIC（Very High Speed IC）プロジェクトの一環として、1981 年に開発が始められた。1983 年には言語に対する要求仕様書がまとめられ、1987 年には言語マニュアル（Language Reference Manual ： LRM）が発行された。1987 年末には、ＩＥＥＥにおいて標準言語として承認され、IEEE Std-1076 となり、現在、世界中に広く普及されている。

ＶＨＤＬは回路を階層的にとらえ、ある回路は下位のより小さい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。

4.1.3 Verilog-HDL

Verilog-HDL は米国ゲートウエイ社（現 Cadence 社）で Verilog-XL というシミュレータ用に開発され、1995 年に IEEE Std-1364 となり、標準化された HDL である。

Verilog-HDL は Verilog-HDL でも回路は階層的に記述する。最下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。特徴としては、RTL に的を絞った簡潔な記述、テストパターン記述言語としての高い効率性などがある。

4.1.4 設計フロー

論理合成と HDL を用いた代表的な ASIC 開発フローを下記に示します。

仕様設計（アルゴリズム設計）

システム全体に対する仕様検討を行う。仕様設計の段階では、実際どのようなハードウェアやソフトウェアとして実現するかは考慮せず、システムに要求される特性や機能について検討する。

↓

機能分割（アーキテクチャ設計）

アルゴリズムを構成する各部分をどのような構造（アーキテクチャ）に基づいて実現するかを検討する。実現方法は、CPU や DSP に対するソフトウェアとして実現する場合もあるし、汎用の ASSP を使用したり、専用の ASIC を開発したりする。

↓

機能設計

ASIC の内部機能をさらに HDL を用いて詳細に定義する。HDL の記述レベルには、論理レベル、レジスタを明確に定義した RTL、クロック数をスケジューリング可能なビヘイビア（動作）レベルなどが考えられる。記述した HDL は、論理シミュレータを用いてテストパターンを与えることで検討される。

↓

論理合成

実現するテクノロジを指定し、目標とするゲート回路の性能を設計制約条件として与え、設計制約を満足するゲート回路を自動的に合成する。合成した回路に対しては、タイミング解析を行い、要求するタイミングを満たしているか確認する。

↓

テスト設計

デバイスの動作を保証するために、故障検出率の高いテストパターンを制作する必要がある。そのため、デバイスのテストを容易にするためのテスト用意か設計を行い、さらにテストパターンの自動生成を行う。

↓

自動配置配線（レイアウト設計）

ゲート回路のネットリストにしたがって、実際にチップを製造するために必要なマスクを作成するために、セルの自動配置と自動配線を行う。

↓

サインオフ検証

論理合成の段階では、セル間を接続する配線長は仮の値を使用しているが、配線後には実際の配線長が得られるために、実配線に基づいたタイミング解析と最終動作確認のためのゲートレベル・シミュレーションを実行する。

↓

製造工程

実際のチップを製造するためのレイアウト・マスクを生成し、製造工程に入る。必要なマスク数は、ゲートアレイとスタンダードセルとでは異なる。チップ完成後、テストを行いパッケージングする。

4.2設計例　―　カウントダウン時計　―

4.2.1カウントダウン時計のＶＨＤＬによる記述

秒を作る

秒は０～9までの数字が必要になるので、カウントダウンには１０進カウンタのプログラムが必要となる。

１０進ダウンカウンタ回路のVHDLによる設計文

```
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 )
variable Q: integer range 0 to 9;
begin  --10count
     if ( RESET = '1' ) then
         TQ <= "1001";
     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;
Q <= TQ;

end process;


end RTL;
```

１０進ダウンカウンタ回路の動作確認のための
テストベンチ記述

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

use std.textio.all;
use work.COUNT10;

entity TEST_count10 is
end TEST_count10;

architecture test of TEST_count10 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;

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 TQ: std_logic_vector(3 downto 0);

begin
    S0:COUNT10 port map (
        CLK,
        RESET,
        CYIN,
        Q,
        CYOUT
        );

CLOCK: process
       begin
           CLK <= '1'; wait for 25ns;
           CLK <= '0'; wait for 25ns;
end process;

count10t: process
       begin
           RESET <= '1'; wait for 50ns;
           RESET <= '0'; CYIN <= '1';wait for 500ns;

           wait;
end process count10t;
end test;
```

１０進ダウンカウンタ回路のシミュレーション結果

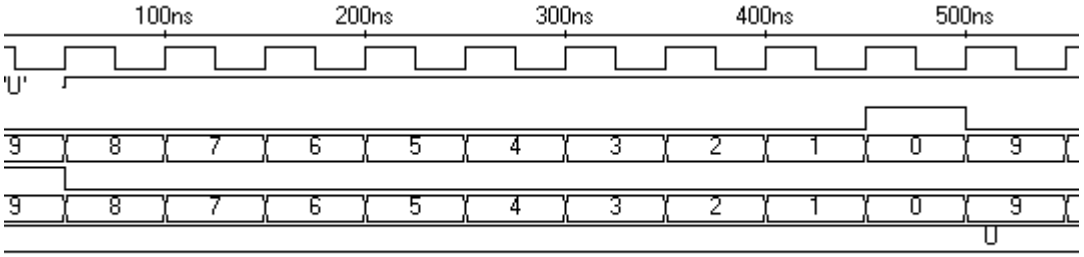


図4.2.1　１０進ダウンカウンタ回路のシミュレーション結果

時計表示のための、セグメント表示プログラムのVHDLによる設計文

```
library IEEE;
use IEEE.std_logic_1164.all;

entity SEG7 is
    port   (
        di :   in  std_logic_vector (3 downto 0);
        do :   out std_logic_vector (6 downto 0)
            ) ;
end SEG7;

architecture RTL of SEG7 is
begin
    process ( DI )
    begin
        case DI is
            when "0000" => DO <= "0000001" ;     -- 0
            when "0001" => DO <= "1001111" ;     -- 1
            when "0010" => DO <= "0010010" ;     -- 2
            when "0011" => DO <= "0000110" ;     -- 3
            when "0100" => DO <= "1001100" ;     -- 4
            when "0101" => DO <= "0100100" ;     -- 5
            when "0110" => DO <= "1100000" ;     -- 6
            when "0111" => DO <= "0001111" ;     -- 7
            when "1000" => DO <= "0000000" ;     -- 8
            when "1001" => DO <= "0001100" ;     -- 9
            when others => DO <= "1111111" ;     -- X
        end case;
    end process;
end RTL;
```

セグメント表示回路の動作確認のためのテストベンチ記述

```
library IEEE;
use IEEE.std_logic_1164.all;

use std.textio.all;
use work.SEG7;

entity TEST_SEG7 is
end TEST_SEG7;
```

```
architecture TEST of TEST_SEG7 is

component SEG7
port ( DI :  in  std_logic_vector (3 downto 0);
       DO :  out std_logic_vector (6 downto 0)
   );
end component;

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

begin
S1 : SEG7 port map (DI,DO);
process
begin
       DI <= "0000"; wait for 50 ns;
       DI <= "0001"; wait for 50 ns;
       DI <= "0010"; wait for 50 ns;
       DI <= "0011"; wait for 50 ns;
       DI <= "0100"; wait for 50 ns;
       DI <= "0101"; wait for 50 ns;
       DI <= "0110"; wait for 50 ns;
       DI <= "0111"; wait for 50 ns;
       DI <= "1000"; wait for 50 ns;
       DI <= "1001"; wait for 50 ns;
end process;

end TEST;
```

セグメント表示回路シミュレーション結果

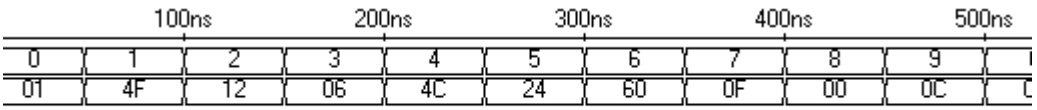


図4.2.2　セグメント表示回路シミュレーション結果

秒　分　時　日のカウントダウン回路のVHDLによる設計文

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

entity cdt is
   port (
   CLK,RESET : in std_logic;
   dhi,dlo      : out std_logic_vector ( 6 downto 0 );--day
   hhi,hlo      : out std_logic_vector ( 6 downto 0 );--hour
   mhi,mlo      : out std_logic_vector ( 6 downto 0 );--min
   shi,slo      : out std_logic_vector ( 6 downto 0 )--sec
   );
end cdt;

architecture RTL of cdt is

component count10
   port (
     CLK,RESET     : in std_logic;
     CYIN          : in std_logic;
     Q             : out std_logic_vector ( 3 downto 0 );
     CYOUT         : out std_logic
     );
end component;

component SEG7
   port   (
      di :  in  std_logic_vector (3 downto 0);
      do :  out std_logic_vector (6 downto 0)
         ) ;
end component;


signal TQa,TQb,TQc,TQd,TQe,TQf,TQg,TQh : std_logic_vector ( 3 downto 0 );
signal CYa,CYb,CYc,CYd,CYe,CYf,CYg,CYh : std_logic;

begin

process ( CLK,RESET ) begin  -- sec
     if ( RESET = '1' ) then
         TQa <= "1001";
```

```
     elsif ( CLK 'event and CLK = '1' ) then
        if (TQa = "0000") then
           TQa <= "1001";
       else
           TQa <= TQa - 1;
     end if;
    end if;
end process;

process ( TQa) begin  --sec out
     if (TQa = "0000") then
           CYa <= '1';
       else
           CYa <= '0';
     end if;
end process;

process ( CLK,RESET,CYa ) begin  --10sec
     if ( RESET = '1' ) then
         TQb <= "0101";
     elsif ( CLK 'event and CLK = '1' and CYa ='1' ) then
        if (TQb = "0000") then
           TQb <= "0101";
       else
           TQb <= TQb - '1';
       end if;
     end if;
end process;

process ( TQb,CYa ) begin  --10sec out
     if (TQb = "0000" and CYa = '1') then
           CYb <= '1';
       else
           CYb <= '0';
     end if;
end process;



process ( CLK,RESET,CYb ) begin  -- min
     if ( RESET = '1' ) then
         TQc <= "1001";
```

```
     elsif ( CLK 'event and CLK = '1' and CYb ='1'  ) then
        if (TQc = "0000") then
           TQc <= "1001";
       else
           TQc <= TQc - '1';
       end if;
    end if;
end process;

process ( TQc,CYb)  begin --min out
     if (TQc = "0000" and CYb = '1') then
           CYc <= '1';
       else
           CYc <= '0';
     end if;
end process;

process ( CLK,RESET,CYc ) begin  --10min
     if ( RESET = '1' ) then
         TQd <= "0101";
     elsif ( CLK 'event and CLK = '1' and CYc ='1' ) then
        if (TQd = "0000") then
           TQd <= "0101";
       else
           TQd <= TQd - '1';
       end if;
     end if;
end process;

process ( TQd,CYc ) begin  --10min out
     if (TQd = "0000" and CYc = '1') then
           CYd <= '1';
       else
           CYd <= '0';
     end if;
end process;


process ( CLK,RESET,CYd ) begin  -- hour
     if ( RESET = '1' ) then
         TQe <= "0011";
     elsif ( CLK 'event and CLK = '1' and CYd ='1'  ) then
```

```
        if (TQe = "0000") then
          if (TQf = "0010") then
             TQe <= "1001";
             if (TQf = "0001") then
                TQe <= "1001";
             else
                TQe <= "0011";
             end if;
          end if;
       else
           TQe <= TQe - '1';
       end if;
     end if;
end process;


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

process ( CLK,RESET,CYe ) begin  --10hour
     if ( RESET = '1' ) then
         TQf <= "0010";
     elsif ( CLK 'event and CLK = '1' and CYe ='1' ) then
        if (TQf = "0000") then
           TQf <= "0010";
       else
           TQf <= TQf - '1';
       end if;
     end if;
end process;

process ( TQf,CYe ) begin  --10hour out
     if (TQf = "0000" and CYe = '1') then
           CYf <= '1';
       else
           CYf <= '0';
     end if;
```

```
end process;



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

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

process ( CLK,RESET,CYg ) begin  -- 10day
     if ( RESET = '1' ) then
         TQh <= "1001";
     elsif ( CLK 'event and CLK = '1' and CYg ='1'  ) then
        if ( TQh = "0000") then
           TQh <= "1001";
       else
           TQh <= TQh - '1';
       end if;
     end if;
end process;

process ( TQh,CYg ) begin  --day out
     if (TQg = "0000" and CYg = '1') then
           CYh <= '1';
       else
           CYh <= '0';
```

```
    end if;
end process;

--process ( TQg,CYh ) begin  --10day out
--     if (TQg = "0000") then
--          CYh <= '1';
--       else
--          CYh <= '0';
--     end if;
--end process;

--7seg
U0 : seg7 port map(TQa, slo);--byou
U1 : seg7 port map(TQb, shi);
U2 : seg7 port map(TQc, mlo);--fun
U3 : seg7 port map(TQd, mhi);
U4 : seg7 port map(TQe, hlo);--ji
U5 : seg7 port map(TQf, hhi);
U6 : seg7 port map(TQg, dlo);--hi
U7 : seg7 port map(TQh, dhi);


end RTL;
```

秒　分　時　日のカウントダウン回路の動作確認のためのテストベンチ記述

```
library IEEE;
use IEEE.std_logic_1164.all;

use std.textio.all;
use work.cdt;

entity TEST_cdt is
end TEST_cdt;


architecture TEST of TEST_cdt is

component cdt is
   port (
   CLK,RESET : in std_logic;
   dhi,dlo      : out std_logic_vector ( 6 downto 0 );--day
```

```
    hhi,hlo      : out std_logic_vector ( 6 downto 0 );--hour
    mhi,mlo      : out std_logic_vector ( 6 downto 0 );--min
    shi,slo      : out std_logic_vector ( 6 downto 0 )--sec
    );
end component;

signal CLK,RESET : std_logic;
signal dhi,dlo,hhi,hlo,mhi,mlo,shi,slo : std_logic_vector ( 6 downto 0 );
--signal TQa,TQb,TQc,TQd,TQe,TQf,TQg,TQh: std_logic_vector ( 3 downto 0 );
--signal CYa,CYbCYc,CYd,CYe,CYf,CYg,CYh : std_logic;

begin
dut:cdt port map (
CLK,RESET,dhi,dlo,hhi,hlo,mhi,mlo,shi,slo
);

process
      begin
          CLK <= '1'; wait for 0.5sec;
          CLK <= '0'; wait for 0.5sec;
end process;


process
   begin
          reset <= '1';wait for 1sec;
          reset <= '0';wait;
end process;

end TEST;
```

秒　分　時　日のカウントダウン回路のシミュレーション結果

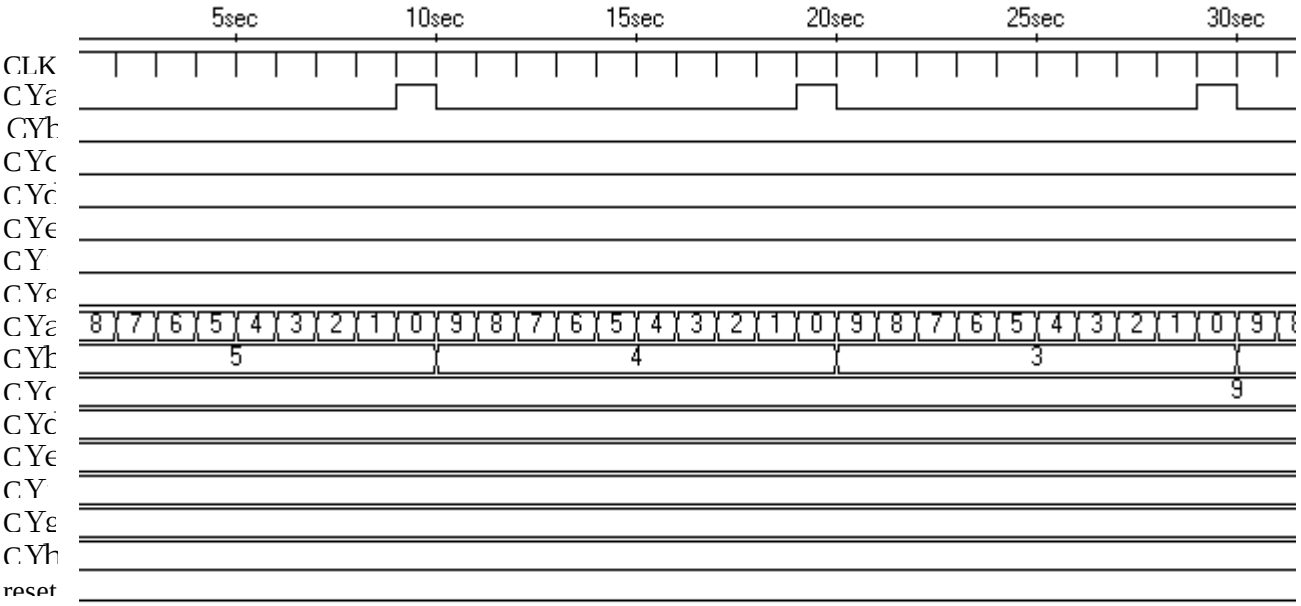


図 4.2.3　秒分時日のカウントダウン時計回路のシミュレーション結果

第５章 終わりに

テストベンチを書く時点で、カウントダウン時計のプログラムの構文上に多くの記述エラーを発見でき、実際に動かしてみる事の、重要性を感じました。

そして、ハードウェア記述言語の存在によって、容易に回路設計ができることを確認することができました。しかし、本来、回路を作る人間は順序回路などの基本的な部分を理解しておくべきであり、明確に理解できていると言えない私にとって、今以上に複雑な回路になってくると、ハードウェア記述言語に頼りっきりになってしまうことが、容易に予想できます。その辺は、今後への課題を残す結果になってしまいました。

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 矢野政顕 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 小島大輔氏、田口禎譲氏、ほか計算機応用講座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