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

田口　禎譲

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

ディジタル回路は、コンピュータの心臓部としての役割だけでなく、テレビ、ビデオ、オーディオ、携帯電話などの一般家電の制御中枢としての役割を担っており、その応用分野は、半導体技術の進歩とともに広がり続けている。

今回、同期式順序回路を設計したが、ディジタル回路の設計図 **1.1.1** のような過程がある。

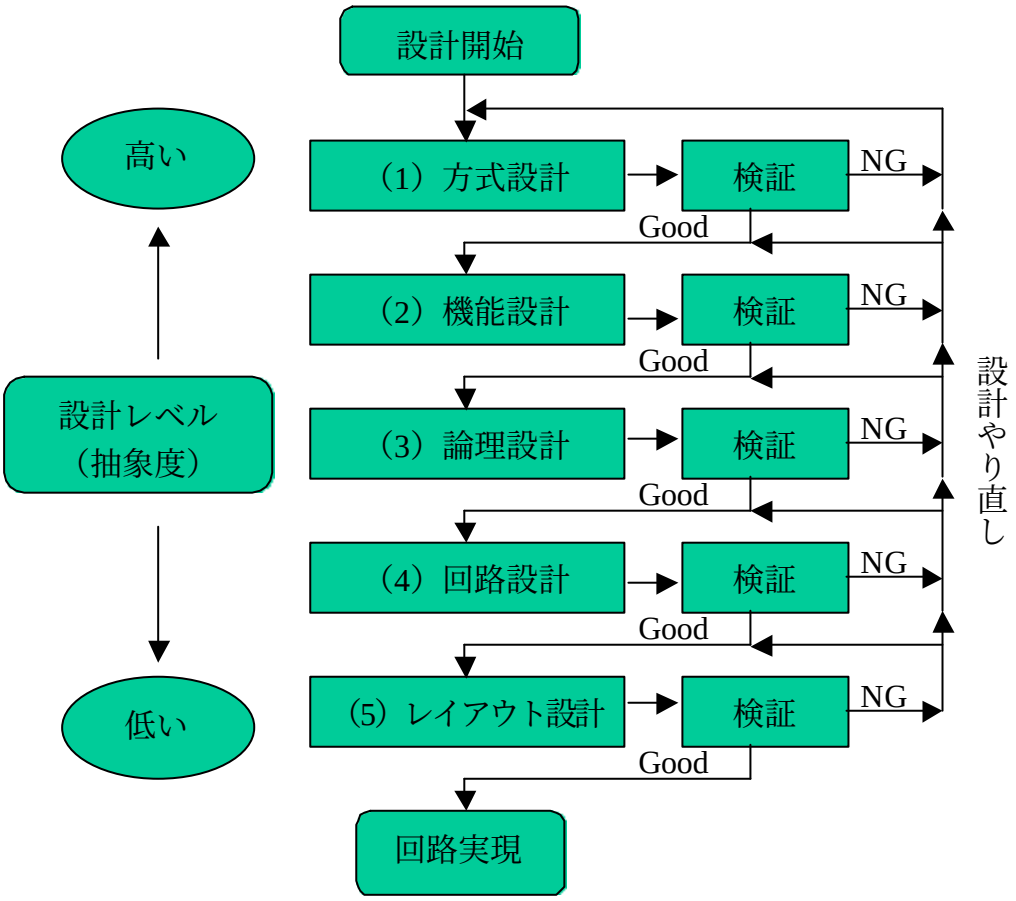

図 **1.1.1** ディジタル回路の設計過程

① 方式設計（system design）

ディジタル回路の動作やそれを実現する手順などの仕様を決定する。

② 機能設計（functional design）

必要となる構成要素や構成要素間のデータの流れを決定する。

(3)　論理設計（logic　design）
ゲート回路を用いた回路図を作成する。

(4)　回路設計（circuit　design）
トランジスタレベルでの回路図を作成する。

(5)　レイアウト設計（layout　design）
IC 上の回路の場合はフォトマスク原画となるマスクパターンを作成する。

ディジタル回路の設計は、上記の（1）～（5）の順に行われ、各段階における設計作業が終了すると、その設計作業が正しく行われたかどうかを確認する。
この確認作業を設計検証（design　verification）、または、単に検証（verification）という。検証の結果、設計作業が正しく行われていると判断された場合は、次の設計段階に進む。

一方、設計作業が正しく行われていないと判断された場合は、その設計段階または、以前の設計段階に戻って、設計作業をやり直す。この設計作業のやり直しは、検証結果がよくなるまで繰り返される。

設計の初期段階では、実際のディジタル回路に対する物理的なイメージが少なく非常に抽象的である。設計が進んでいくと、具体的な回路イメージに近づいていく。ディジタル回路設計においては、抽象度の高い設計段階を高レベル（high　level）といい、抽象度の低い設計段階を低レベル（low　level）という。上記の設計過程は、高いレベルから低いレベルへと進んでいくため、トップダウン設計（top-down　design）と呼ばれる。一方、これとは逆に低いレベルから高いレベルへと進めていく設計手法もあり、これをボトムアップ設計（bottom-up　design）と呼ぶ。

トップダウン設計とボトムアップ設計

トップダウン設計は、まずディジタル回路全体の動作や機能を決定し、徐々に具体化（回路化）していく方法である。トップダウン設計手法は、比較的容易に設計作業が行え、設計検証にかかる時間や設計期間を短縮できるなどのメリットがある。

一方、ボトムアップ設計手法は、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である。この手法は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設計において、従来採用されてきた方法である。

トップダウン設計のほうが設計手法として回路設計者の注目を集めているが、実際に

ディジタル回路を設計する場合、一から設計を始めるのではなく、過去の設計資産を再利用することが多い。そのためトップダウン設計とボトムアップ設計とが混在しているのが実情である。

今回はボトムアップ設計で同期式順序回路を図 **1.1.1** の（3）回路設計までを行った。順序回路には様々な種類がある。その中でカウンタを取り上げ、カウントダウン回路を設計、製作した。
設計には、

⑵　ディジタル IC を用いた設計及び、プリント基板上での実現
⑶　ハードウェア記述言語による設計

の 2 パターンを行った。

第２章　順序回路

２.１　順序回路

順序回路と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。これに対し、現在の入力によってのみ出力が決定される回路を組み合わせ回路という。

順序回路はコンピュータの中の重要な機能を実現するために至るところで用いられている。これには、プログラムカウンタなどに用いられる計数回路（カウンタ）、メモリのアドレスやデータあるいはメモリから取り出された命令を１マシンサイクル間保持する命令レジスタ、データの直並列変換などのためのシフトレジスタなどが含まれる。

順序回路は、組み合わせ回路と１ビットの記憶素子であるフリップフロップをつなげ、出力をフィードバックさせることで回路を実現させている。

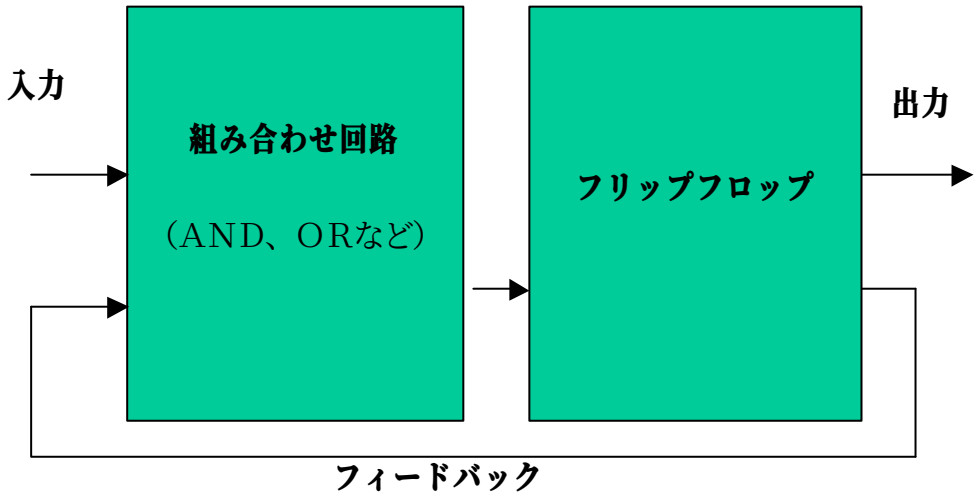

図 2.1.1　順序回路のイメージ図

順序回路には同期式順序回路と非同期式順序回路がある。

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。同期式順序回路はクロックパルスを用いて、この信号の周期で順序回路全体を動作させるものである。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。同期式順序回路は、クロックを用いるため高速に動作し得る部分もこの周期で動作することになるた

め速度的には非同期式にくらべ劣ってしまうが、回路構成は簡単になるので、コンピュータで広く使われている。

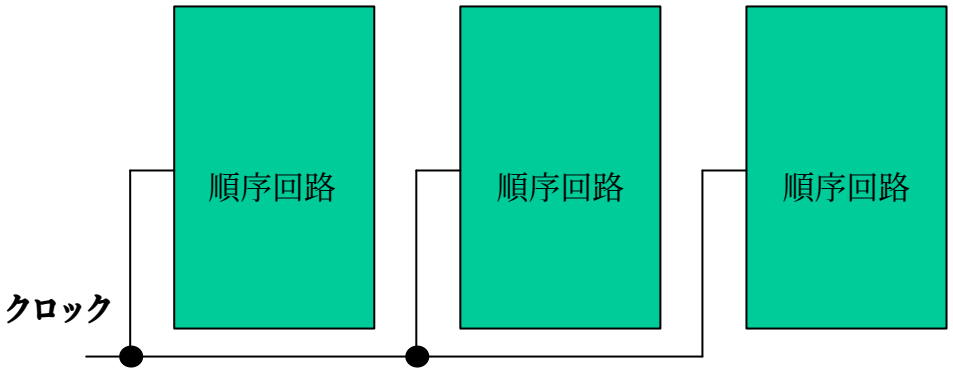

図 2.1.2　同期式順序回路のイメージ

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の変化、及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。遅延回路による素子の伝播遅延時間を利用しているため入力が加わると直ちに状態と出力が確定し、それを入力側へ伝播するため全体的に動作速度を早くすることができるという利点はあるが、一般に回路構成は複雑になってしまう。

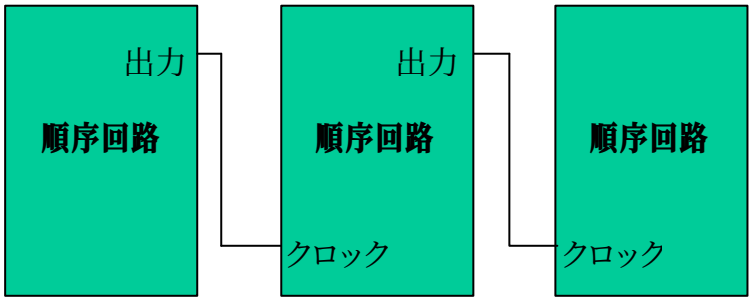

図 2.1.3　非同期式順序回路のイメージ

2.2　フリップフロップ

ラッチとフリップフロップ

クロックが変化したときに状態および出力が変化するものをフリップフロップという。信号レベルで状態および出力が変化するものをラッチという。

ラッチは情報をＯＦＦ状態のとき保持し、ON状態のときに保持した情報をスルーする。フリップフロップは OFF 状態のとき情報を保持し、エッジで情報をとりこみ、ON 状態でまたその情報を保持する。

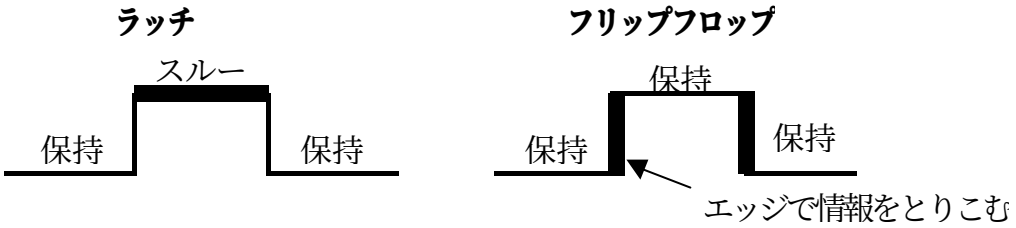

図 2.2.1　ラッチとフリップフロップ

SR ラッチ

１ビットのラッチは０と１とのいずれかの安定状態（双安定状態）をもつ。

ラッチの状態変化は入力によって決定され、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。最小構成のラッチは、２個のＮＡＮＤ素子で構成されている SR ラッチである。

下図は、SR ラッチの回路構成図とタイムチャートである。

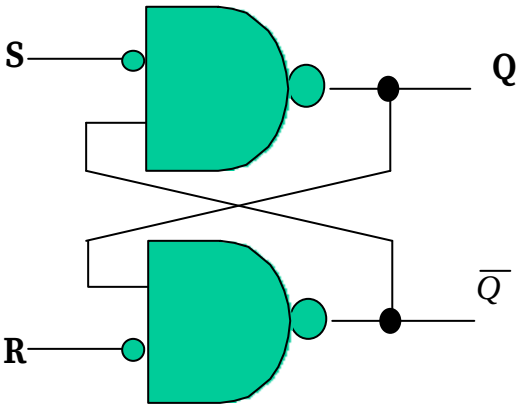

図2.2.2　SRラッチ

表 2.2.1　SR ラッチの状態表

S	R	Q
0	0	前の状態を保持
0	1	0（リセット）
1	0	1（セット）
1	1	×（不定）（禁止）

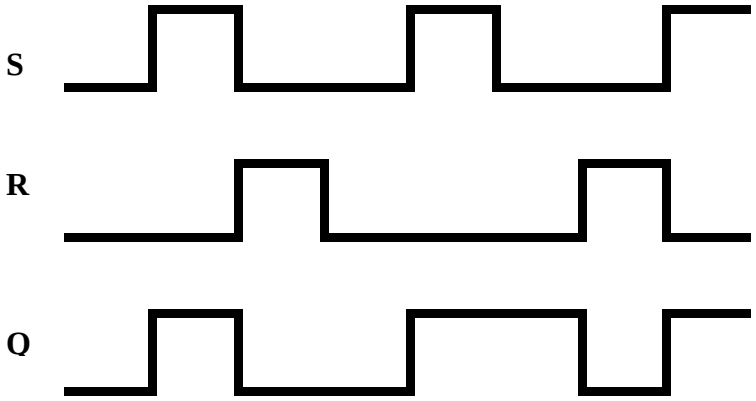

図 2.2.3　SR ラッチのタイムチャート

ＳＲフリップフロップ

Ｓ，Ｒとも０のとき、Q出力は不変。Ｓ，Ｒとも１のとき出力は不定。Ｓ，Ｒのいずれかが１のときＱ出力は強制的に１（セット）、０（リセット）という動作をする。

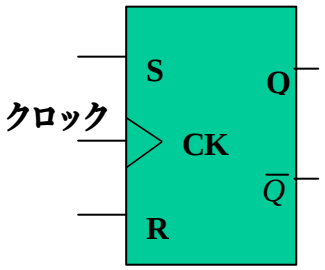

図2.2.4　SRフリップフロップ

表 2.2.2　SR フリップフロップの状態表

S	R	Q+
0	0	前の状態を保持
0	1	0（リセット）
1	0	1（セット）
1	1	×（不定）（禁止）

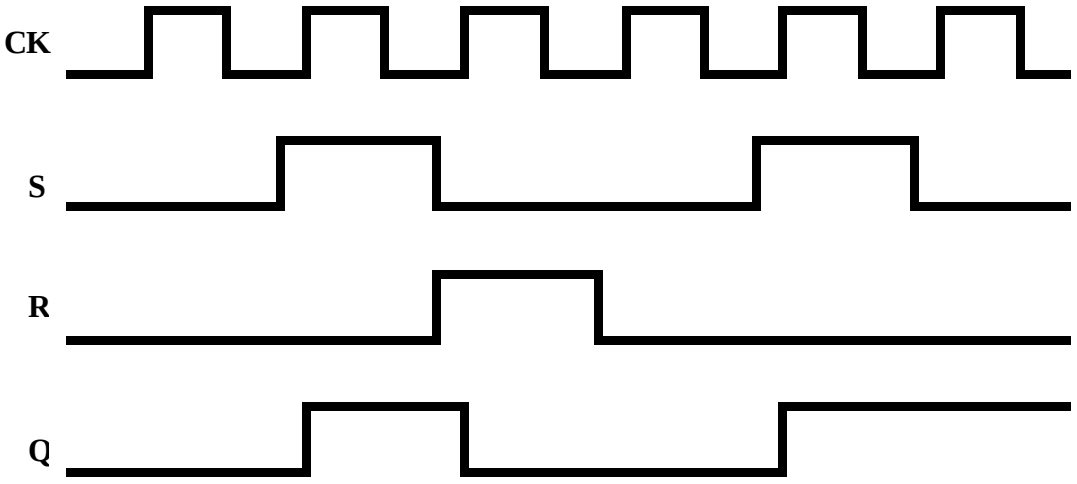

図 2.2.5　SR フリップフロップのタイムチャート

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（ラッチ）用として利用できる。

表 2.2.3　D フリップフロップの状態表

D	Q +
0	0
1	1

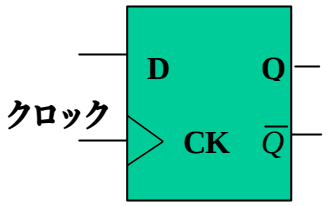

図2.2.6　Dフリップフロップ

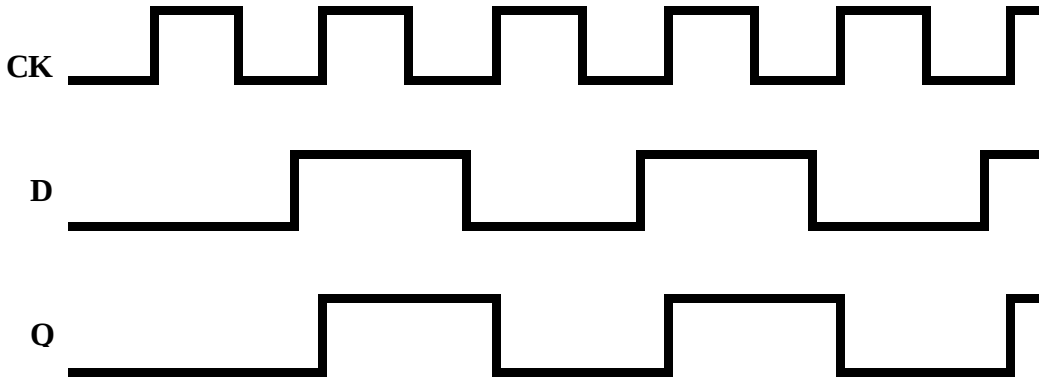

図 2.2.7　クロックつきD フリップフロップのタイムチャート

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入力と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。J、K入力とも１のとき、Q出力が反転（トグル）という動作をする。

表 2.2.4　JK フリップフロップの状態表

J	K	Q+
0	0	前の状態を保持
0	1	0（リセット）
1	0	1（セット）
1	1	NOT(Q)（トグル）

$$Q+ = \overline{K}Q + J\overline{Q}$$

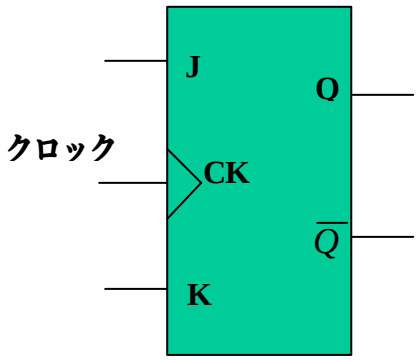

図2.2.8　JKフリップフロップ

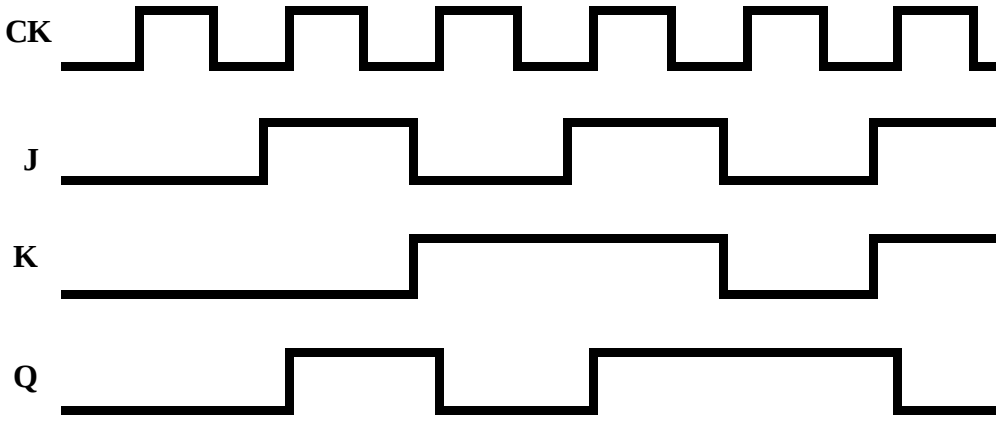

図 2.2.9　JK フリップフロップのタイムチャート

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（Q出力）は不変。Ｔ入力が1のとき、状態（Q出力）を反転（トグル）する

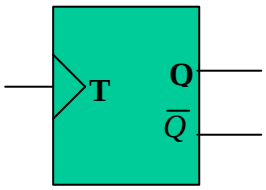

図2.2.10　Tフリップフロップ

表 2.2.5　T フリップフロップの状態表

T	0	1
Q	Q(不変)	NOT(Q)　(トグル)

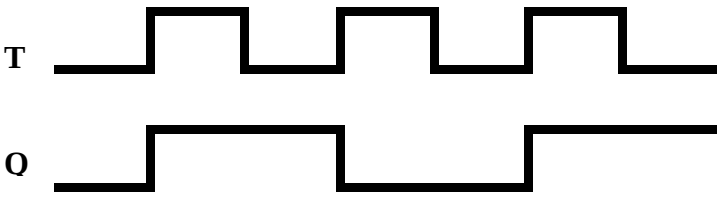

図 2.2.11　T フリップフロップのタイミングチャート

マスタスレーブ型フリップフロップ

同一時刻に出発したいくつかの信号が途中異なる経路を通過してひとつのゲートの入力端子に到着した場合、経路ごとに異なる伝搬遅延時間の影響を受けて到着時間

に差が生ずることが多い。組み合わせ回路ではこの時間差が原因となってハザードを発生する場合がある。順序回路も同様の問題が発生する場合がある。これをレースという。レースはゲート遅延時間の差によって生ずるハザードの一種でもある。

レースの原因の一つは入力端子から出力端子に向かう直通のルートが内部に存在することである。そこでフリップフロップの内部を 2 段階構成とし 1 段ずつ作動させて直通ルートを断つと、レースの起こりにくいフリップフロップを作ることができる。この構成をしたフリップフロップをマスタスレーブ型フリップフロップという。

クロックの立ち上がり（マスタ）と立ち下がり（スレーブ）の組で一つの動作（2段階動作）を行う。

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。

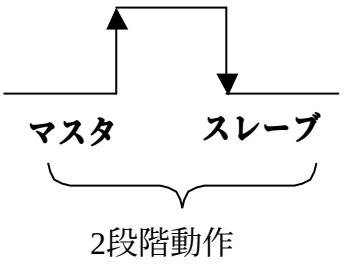

図2.2.12　マスタスレーブ型フリップフロップのクロック入力

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）、立ち下がり（ネガティブエッジ）で生じるフリップフロップのことをいう。

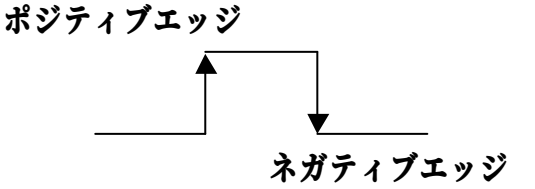

図 2.2.13　ポジティブとネガティブエッジ

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。

マスタスレーブ型とエッジトリガ型の構成や動作を T フリップフロップを用いて説明すると以下のようになる。

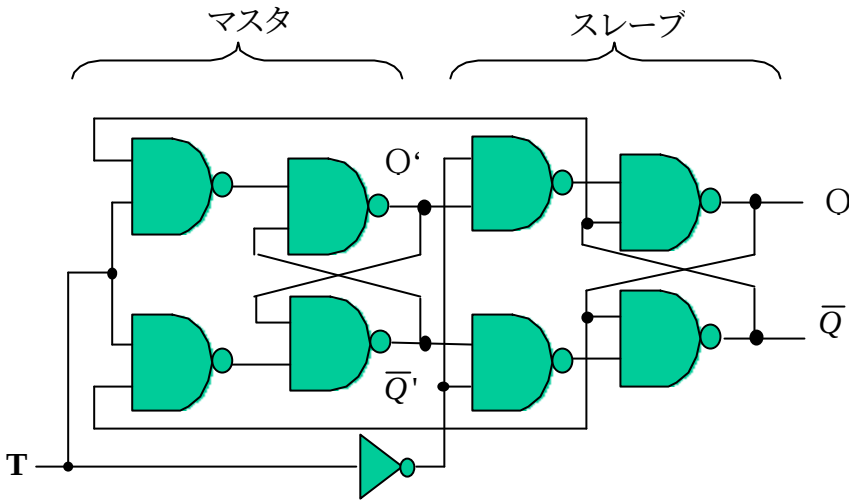

図 2.2.14　マスタスレーブ型 T フリップフロップの論理回路図

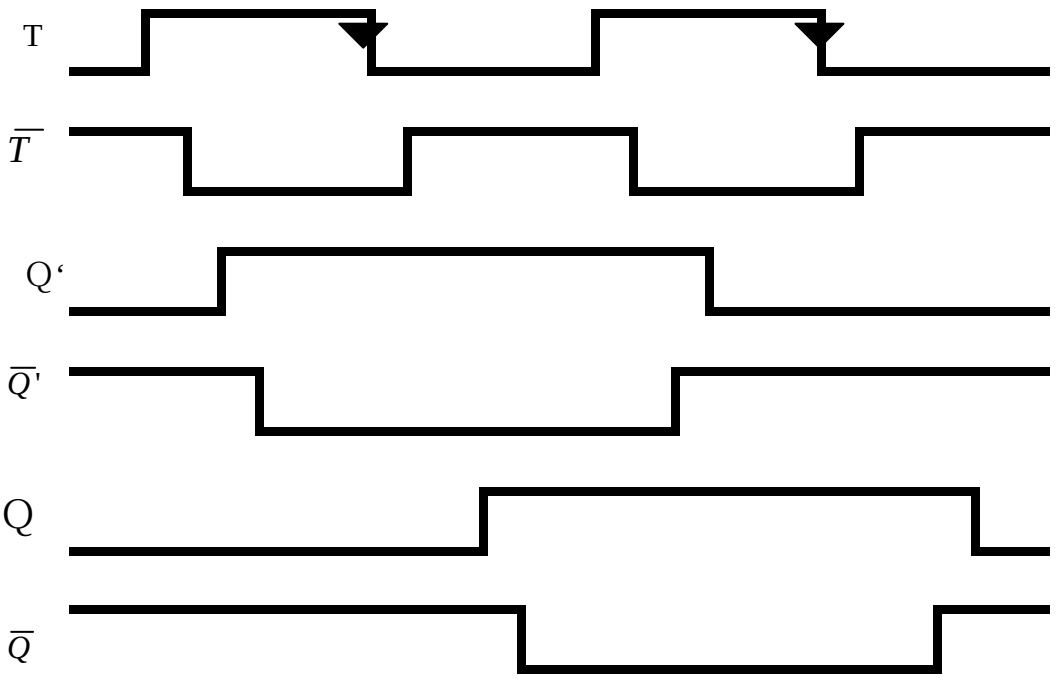

図 2.2.15　マスタスレーブ型 T フリップフロップのタイムチャート

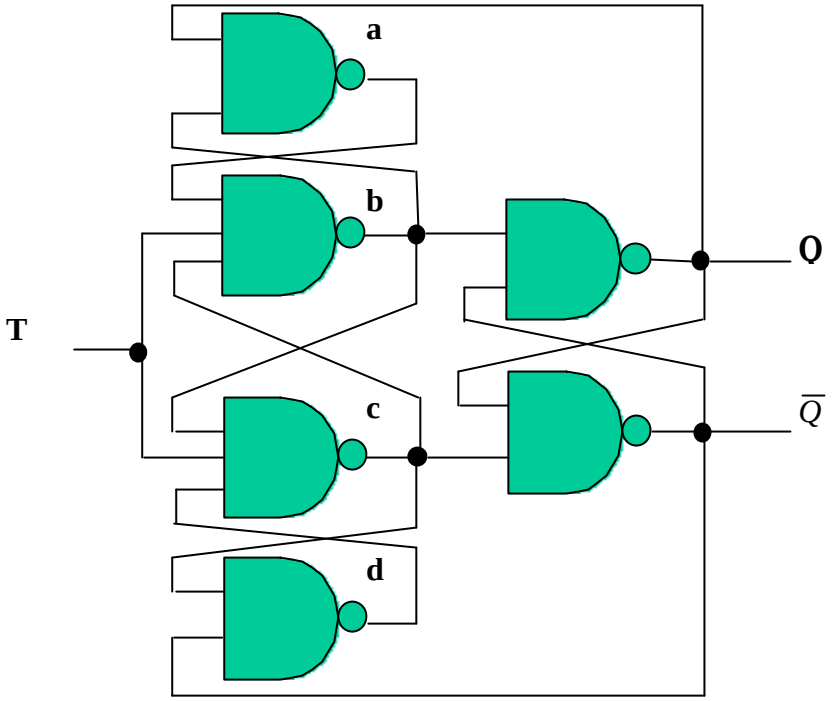

図 2.2.16　エッジトリガ型Ｔフリップフロップの論理回路図

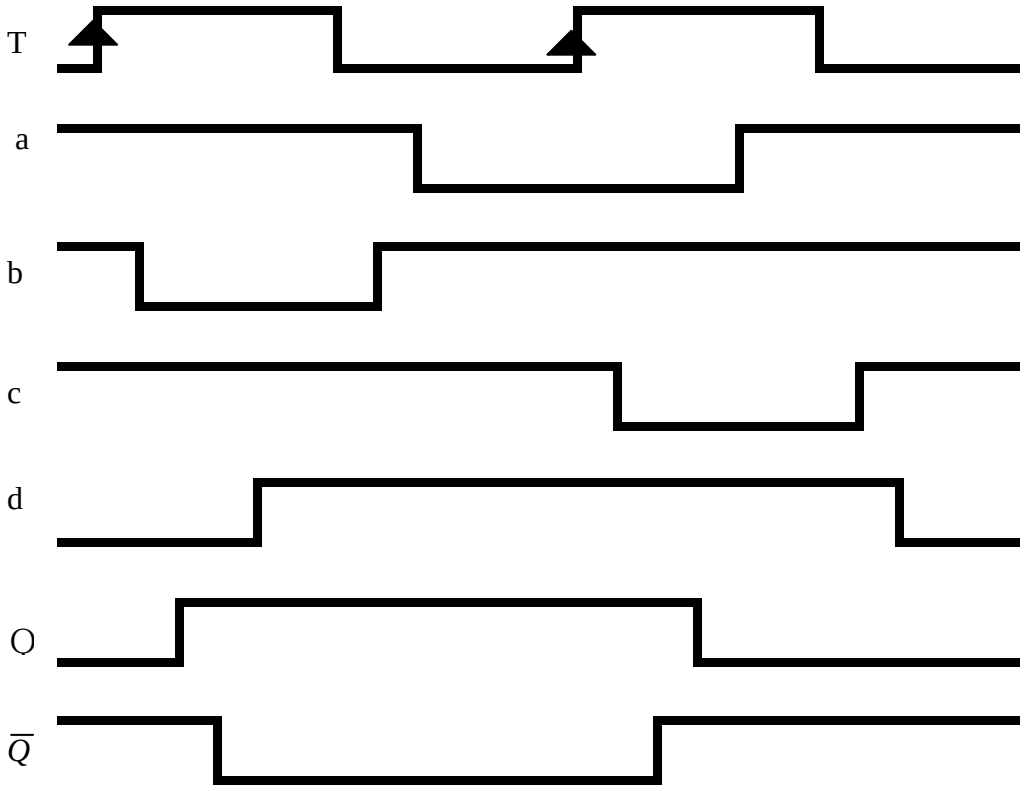

図 2.2.17　エッジトリガ型Ｔフリップフロップのタイムチャート

2.3　カウンタ

あらかじめ決められた順序で状態変化を起こす順序回路で入力されたパルスの個数を計数し、それを保持する回路のことをいう。カウンタには同期式と非同期式とがある。カウンタは隣接するフリップフロップ間に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。

非同期式カウンタ

各順序回路が共通のクロックにより同期しておらず、別々のタイミングで動作しているカウンタである。

下図は最下位段以外の各フリップフロップは、前段の出力（リプルキャリ）によってトリガされるのでリプルカウンタともいう。

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。

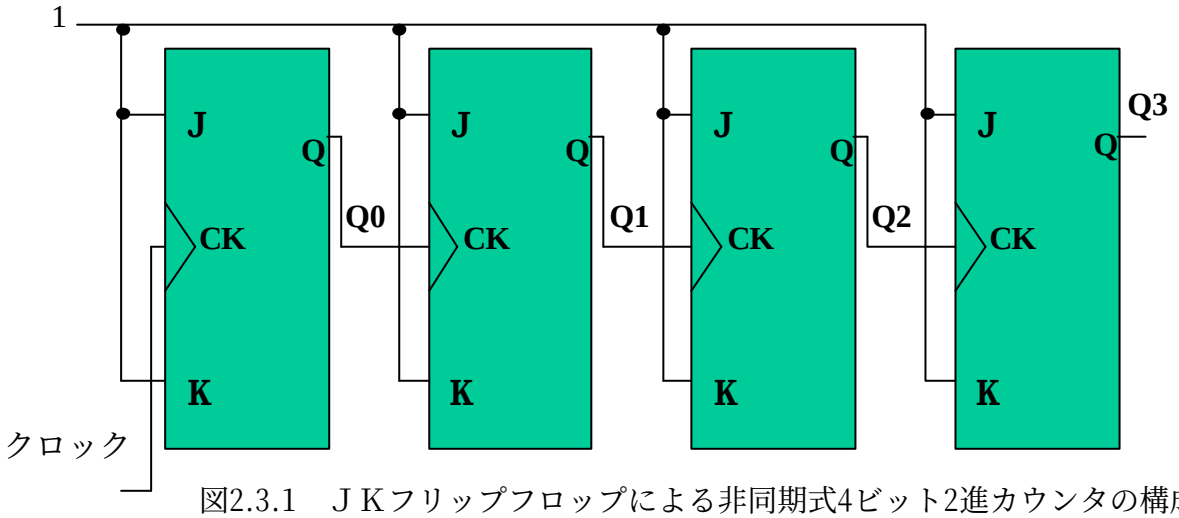

図2.3.1　ＪＫフリップフロップによる非同期式4ビット2進カウンタの構成

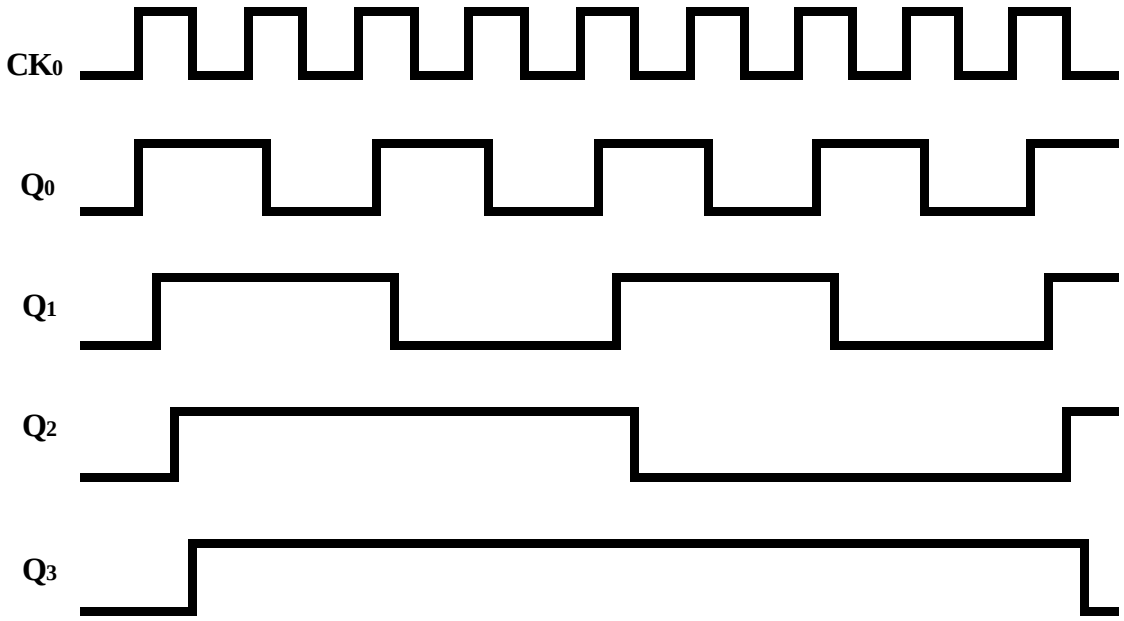

図 2.3.2　ＪＫフリップフロップによる非同期式
4 ビット 2 進カウンタのタイムチャート

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、それらすべてのフリップフロップが同時に（同期して）状態変化するカウンタのことである。

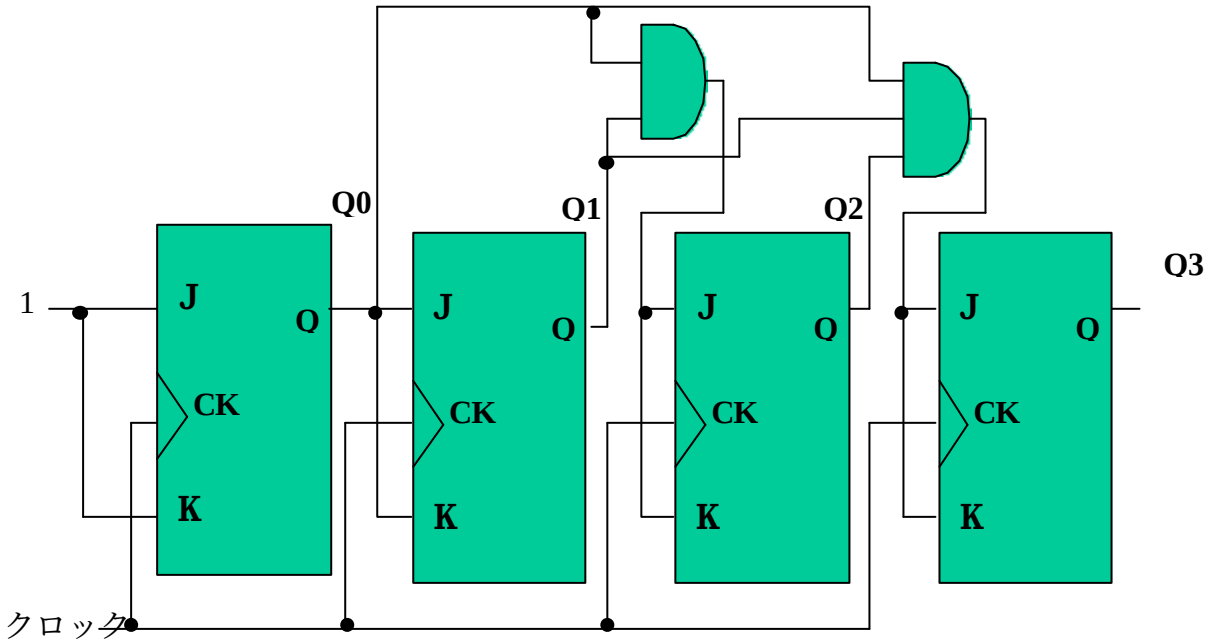

図2.3.3ＪＫフリップフロップによる同期式カウンタ（4ビット2進カウンタ）

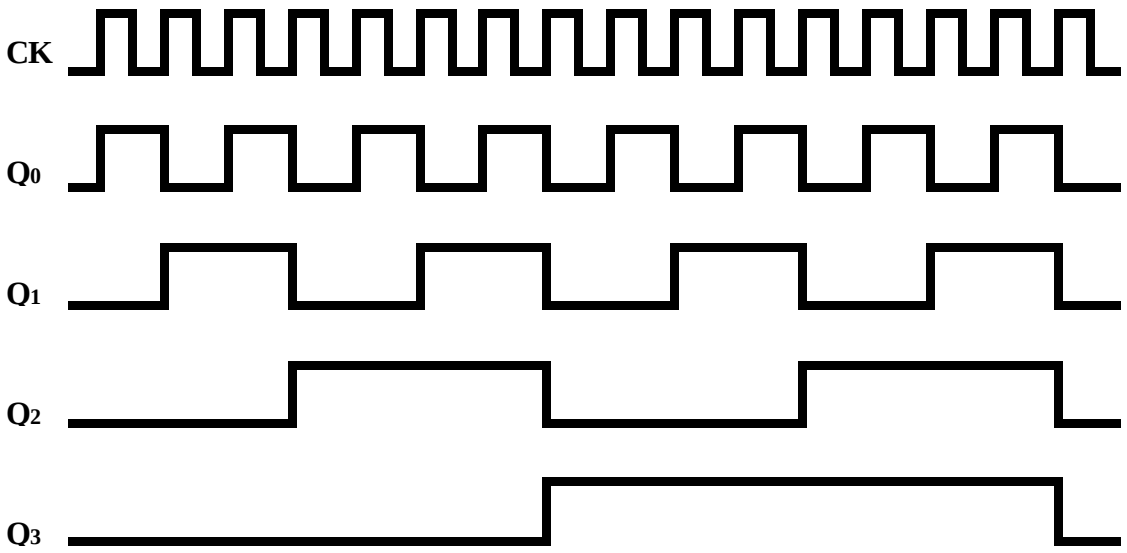

図 2.3.4　ＪＫフリップフロップによる同期式
4 ビット 2 進カウンタのタイムチャート

カウンタの設計方法

カウンタ設計の際は、以下のことなどを決定する必要がある。

①　何進カウンタであるか。

用途によって決まるが、4，8，16 進など 2^n 進カウンタは比較的簡単に設計できる。3，7，10 進カウンタは 2^n 進カウンタに比べ複雑になる。

②　同期式か、非同期式か。

ど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。

❸　使用するフリップフロップの種類。

カウンタでは、前段のフリップフロップが変化することで、後段のフリップフロップが変化してしまうレーシングという現象が起こることもある。これには、クロックの立ち上がり、または立下りだけを用いて同時に読み込みと出力を変化させるエッジトリガ型のフリップフロップや前段はクロックの立ち上がりで動作し、後段はクロックの立下りを用いて出力が変化するマスタスレーブ型を用いることでレーシングを発生しにくくすることができる。

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、バイポーラトランジスタを用いたＩＣ、ユニポーラトランジスタのMOSFETを用いたＩＣ、バイポーラトランジスタとMOSFETとが混在したＩＣがある。

表 3.1.1　標準論理 IC の回路素子による分類

回路素子	回路構成
バイポーラ	ＴＴＬ ＥＣＬ
ユニポーラ	ｐMOS ｎMOS CMOS
バイポーラ ユニポーラ　混在	ＢｉCMOS

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（emitter coupled logic）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。

このほかにバイポーラトランジスタとユニポーラトランジスタ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あまり使われていない。

ゲート、フリップフロップ、カウンタなど、標準的な論理を用意してディジタル回路の構成に用いることができるようにしたＩＣを、標準ロジックＩＣと呼ぶ。標準ロジックＩＣには、構成回路別に××シリーズまたは××ファミリと呼ばれる多くの種類がある。

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、

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。このため型名表示の部分を省略して、単に 74×××と呼ぶこともある。

英文字からわかるメーカー名の代表的なものを表 **3.1.1** に、型名表示のファミリの例を表 **3.1.2** に示す。

表 3.1.2　メーカー名

SN	TI（テキサス インツルメント）
MCI	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 3.1.3　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

TTL ファミリの型名表示についての特徴をまとめたものを表 3.1.3 に、CMOS ファミリについての特徴をまとめたものを表 3.1.4 に示す。

表 3.1.4　ＴＴＬ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。
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。

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とインター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とインターフェース可能にしたもの。

表 **3.1.5**　ＣＭＯＳファミリと特徴

上の表などで示したＴＴＬファミリとＣＭＯＳファミリのほかにＢｉＣＭＯＳファミリというものもある。ＢｉＣＭＯＳファミリには 74ＢＣシリーズと 74ＢＣＴシリーズがある。

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。

TTL ファミリの代表的なカウンタ IC の型番とその特徴を表 3.2.1 に、CMOS ファミリの代表的なカウンタ IC を表 3.2.2 に示す。

表 3.2.1　TTL（同期式）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

表 3.2.2　TTL（非同期式）の 4 ビットカウンタ用 IC の型番とその特徴

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 3.2.3　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

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り、同期式は入力クロックに同期して各ビットの出力が変わるものであり、非同期式は入力クロックに同期しないで入力に応じて各ビット出力が変わるものである。

また、これらのファミリの中に、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進カウンタであるバイナリーカウンタやBCD（Binary Coded Decimal number）、2進化10進数を扱うBCDカウンタなどもある。初期値が、外部から設定できるカウンタをプログラマブル・カウンタという。

クリアとプリセット

カウンタ内の記憶回路であるフリップフロップのほとんどはＪＫまたはDフリップフロップであり、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。これらの入力端子は、データ入力信号とは無関係に、フリップフロップをHレベル（プリセット）またはＬレベル（リセット）にするためのものである。これらの信号がＬレベル時にプリセットまたはリセットを行う場合をアクティブローという。回路図上では、ピン配置図のＰＲ端子、ＣＬＲ端子に○をつけて示す。（Hレベルの場合はアクティブハイという）

PR、CLRがアクティブ時にクロック入力に関係なくプリセットまたはクリアを行うものを非同期プリセット、非同期クリアという。

アクティブロー時に動作し、非同期プリセット、非同期クリアである場合、以下のように動作する。

① **ＰＲにＬレベルを入力すると、出力Q＝H、$\overline{Q}$ ＝Ｌとなる。**

② **ＣＬＲにＬレベルを入力すると、Q＝Ｌ、$\overline{Q}$ ＝Hとなる。**

③ **ＰＲ，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である。**

④ **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。

これに対して、PR、CLR がアクティブのときクロック入力が入力されて初めてプリセット、クリアを行うものを同期プリセット、同期クリアという。

3.3 設計例 ― カウントダウン時計 ―

今回設計したカウントダウン時計は水晶から発生する信号をデバイダーに通して１秒に１回信号を発生させたものを各カウンタにクロックとして同期させ、日、時間、分、秒を７セグメントで表示する。

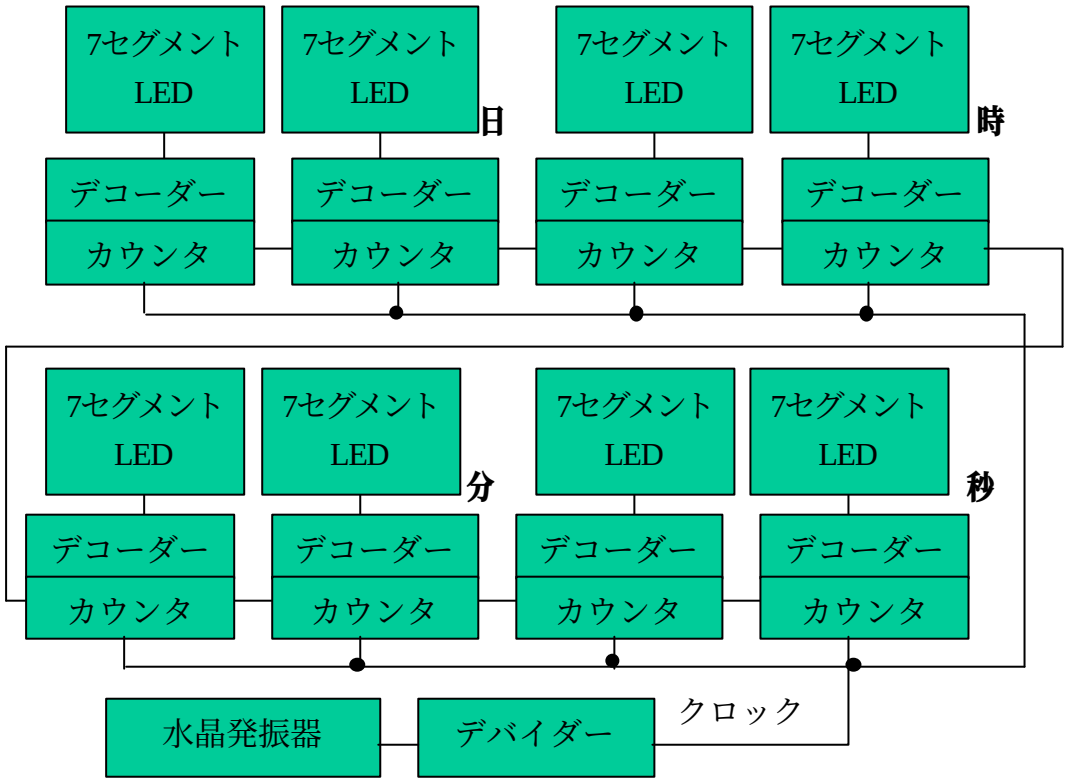

図 3.3.1　カウントダウン時計のブロック図

材料

- 抵抗
  - スイッチ用・・・・・・・・・・・・・10ｋΩ　4 本
  - セグメント用・・・・・・・・・・・220Ω　56 本
- コンデンサ
  - 電解コンデンサ（電源用）・・・・・・・470μF　2 個
  - セラミックコンデンサ（スイッチ用）・・0.1μF　4 個
- 標準論理 IC
  - デコーダ・・・・・・・・・・SN74LS247P　8 個

カウンタ・・・・・・・・・・SN74LS169BN　　8 個
AND 回路・・・・・・・・・SN74LS08N　　4 個
NAND 回路・・・・・・・・SN74LS00N　　1 個
カウントデバイダー・・・・・SN74LS292N　　1 個

- 水晶発振器
  TOYOCOM　TCO-707F　　4.194304MHz　　1 個
- 7 セグメント LED　　8 個
- スイッチ　　4 個
- プリント基板　　2 枚

使用したソフト

- PSpice（Schematics・PSpiceA/D・Probe）

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下に表や図をまとめる。

(1)SN74LS00 と SN74LS08

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

表 3.3.1　SN74LS00 と SN74LS08 の電気特性

	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

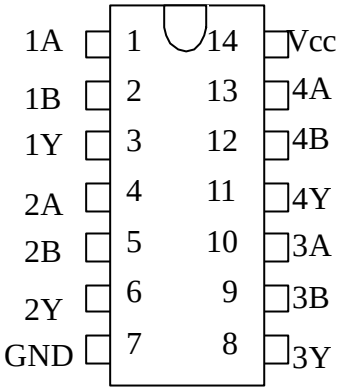

図 3.3.2　SN74LS00 と SN74LS08
の IC のピン配置図

表 3.3.2　SN74LS00 と SN74LS08 の状態表

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

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

（2）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 3.3.3　SN74LS169B の電気特性

		SN74LS169B		
		MIN	NOM	MAX
V_{CC} Suppy voltage (V)		4.5	5	5.5
V_{IH} High-level input voltage (V)		2		
V_{IL} Low-level input voltage (V)				0.8
I_{OH} High-level output current (mA)				-0.4
I_{OL} 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 before 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		
T_A Operating free-air temperature (°C)		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 3.3.2 のピン番号 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。

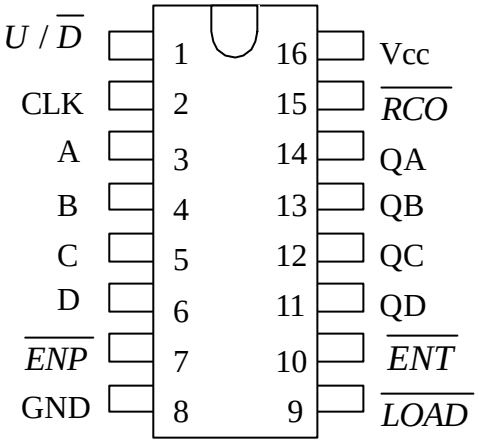

図 3.3.3　SN74LS169B の IC のピン配置図

SN74LS169B を論理回路で表すと図 3.3.4 のようになっている。

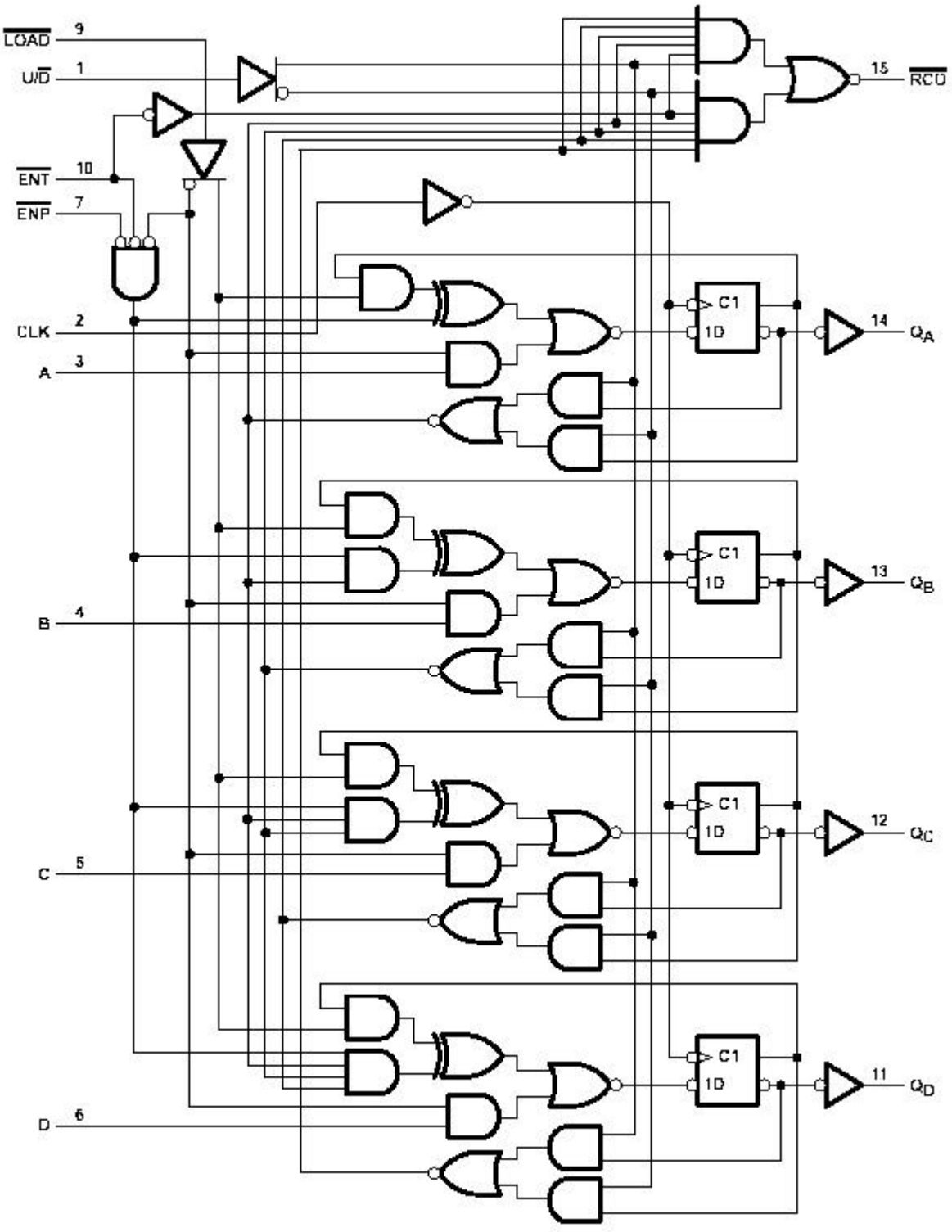

図 3.3.4　SN74LS169B の論理回路図

（http://www.tij.co.jp より参照）

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外部からのデータを取り込む。。

（3）SN74LS292N

SN74LS292N はカウントデバイダーであり、今回、水晶発振器とこのデバイダーを用い１Ｈｚのクロックを発生させる。デバイダーの構造はフリップフロップになっており、フリップフロップを通ることで 1/2 分周期される。

例えば、100Ｈｚの信号をフリップフロップに一回通すと出力では 1/2 分周期された 50Ｈｚの信号が取り出せる。これを繰り返すことで望みの周波数を取り出すことが可能である。

今回用いた SN74LS292N は外部からｎ回通す、すなわち、$1/2^n$分周期させるかを設定できる。

表 3.3.4　SN74LS292N の設定表

PROGRAMMING INPUT					Q	PROGRAMMING INPUT					Q
E	D	C	B	A	BINARY	E	D	C	B	A	BINARY
L	L	L	L	L	Inhibit	H	L	L	L	L	2^{16}
L	L	L	L	H	Inhibit	H	L	L	L	H	2^{17}
L	L	L	H	L	2^{2}	H	L	L	H	L	2^{18}
L	L	L	H	H	2^{3}	H	L	L	H	H	2^{19}
L	L	H	L	L	2^{4}	H	L	H	L	L	2^{20}
L	L	H	L	H	2^{5}	H	L	H	L	H	2^{21}
L	L	H	H	L	2^{6}	H	L	H	H	L	2^{22}
L	L	H	H	H	2^{7}	H	L	H	H	H	2^{23}
L	H	L	L	L	2^{8}	H	H	L	L	L	2^{24}
L	H	L	L	H	2^{9}	H	H	L	L	H	2^{25}
L	H	L	H	L	2^{10}	H	H	L	H	L	2^{26}
L	H	L	H	H	2^{11}	H	H	L	H	H	2^{27}
L	H	H	L	L	2^{12}	H	H	H	L	L	2^{28}
L	H	H	L	H	2^{13}	H	H	H	L	H	2^{29}
L	H	H	H	L	2^{14}	H	H	H	H	L	2^{30}
L	H	H	H	H	2^{15}	H	H	H	H	H	2^{31}

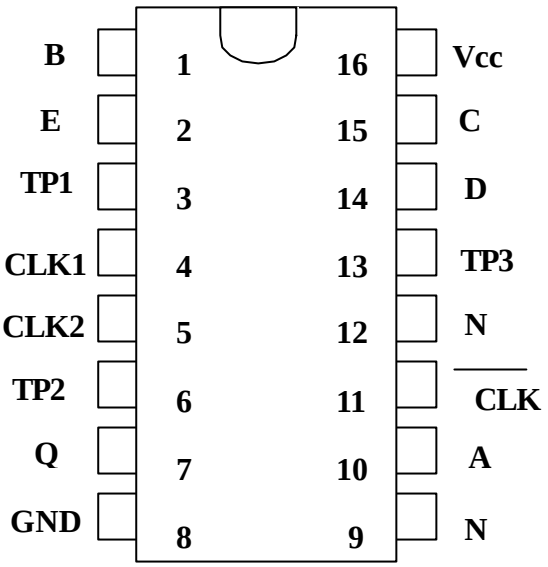

図 3.3.5　SN74LS292N の IC のピン配置図

今回使用した水晶発振器は 4.194304MHz であるが、4194304 は 2^{22} なので A、D に L レベルの信号を入力し、B、C、E に H レベルを入力することで 1Hz の信号にしている。

表 3.3.5　SN74LS292N の電気特性

(4) SN74LS247P

	SN74LS292N		
	MIN	NOM	MAX
Vcc Suppy voltage (V)	4.7	5	5.25
VIH High-level input voltage (V)	2		
VIL Low-level input voltage (V)			0.8
IOH High-level output current (mA)			-1.2
IOL Low-level output current (mA)			24
Fclock Clock frequency (MHz)	0		30
tw Pulse duration, CLK high or Low (ns)	16		
tsu Setup time before CLK (ns)	55		
th Hold time,data after CLK	15		
TA Operating free-air temperature (℃)	0		70

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

表 3.3.6　SN74LS247P の電気特性

		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

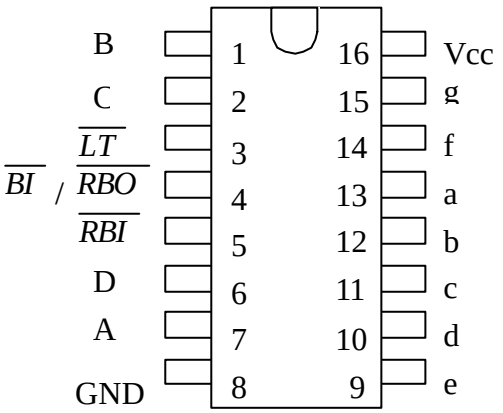

図 3.3.6　SN74LS247P のピン配置図

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 3.3.7　SN74LS247P の状態表

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

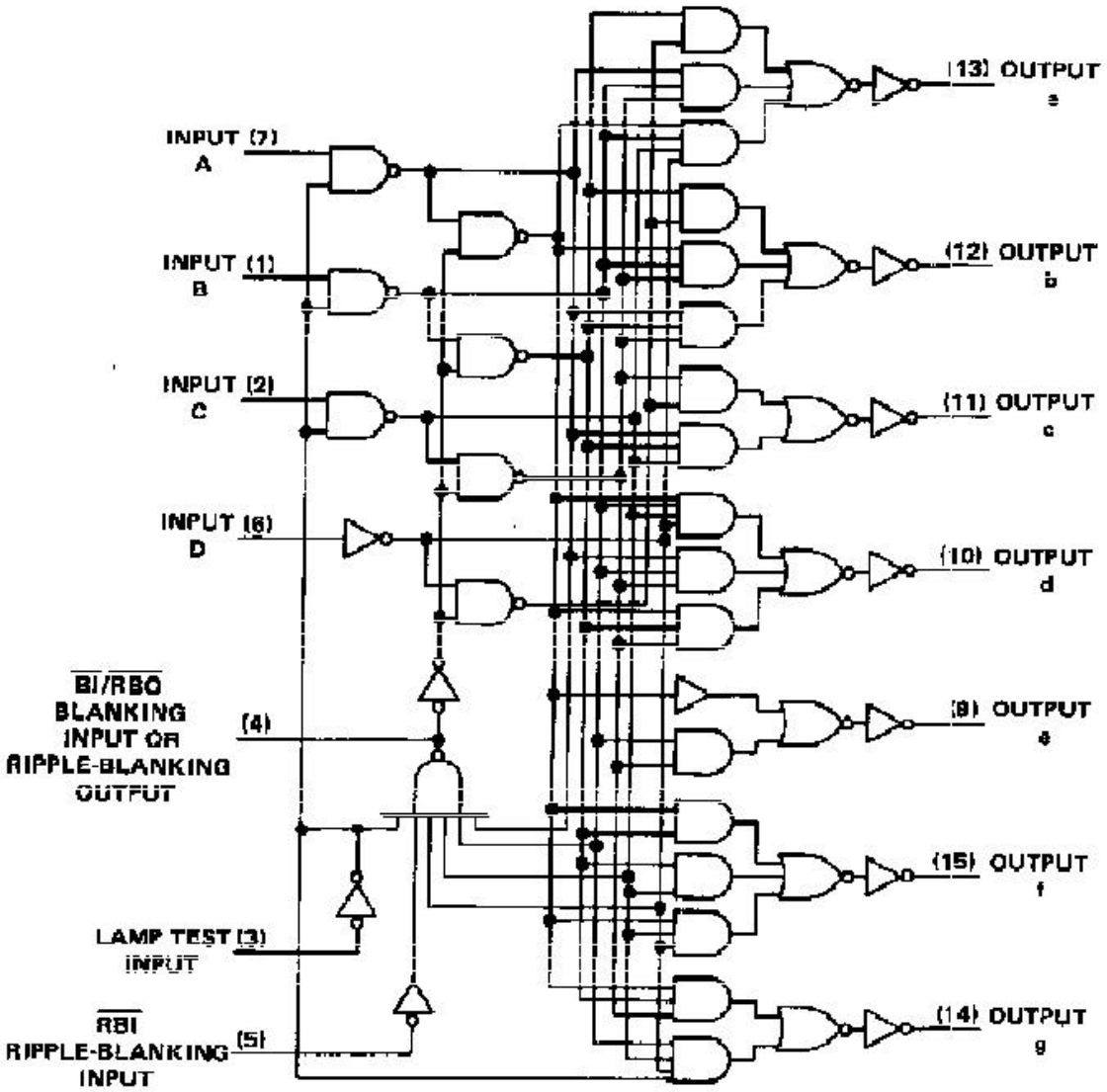

図 3.3.7　SN74LS247P の論理回路図（http://www.tij.co.jp より参照）

SN74LS169B と SN74LS247P を用いて設計したカウントダウン時計を図 3.3.8 に示す。実際、プリント基板上に実現したものは図 3.3.8 の回路に水晶発振器とデバイダー、７セグメント LED、スイッチを取り付けたものである。

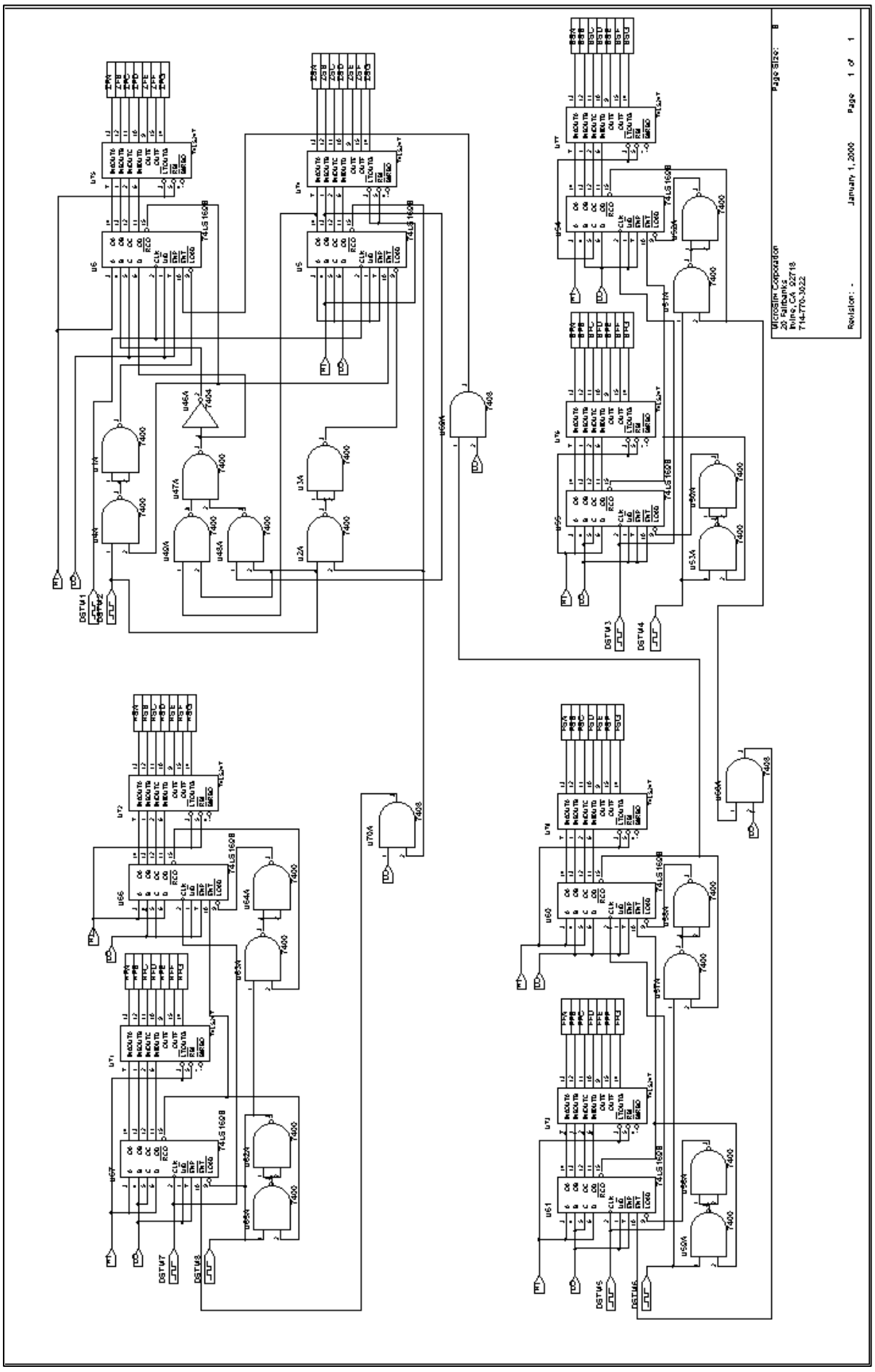

図 3.3.8　カウントダウン時計の回路図

下図 3.3.9 は図 3.3.8 の SN74LS169B から出力された信号をシミュレーションしたものである。図3.3.8 はシミュレーションのため桁下げ信号が常に発生している状態にし、秒、分、時間、日が同時に動作するようにしてある。下の図 3.3.7 にシミュレーション結果を示す。

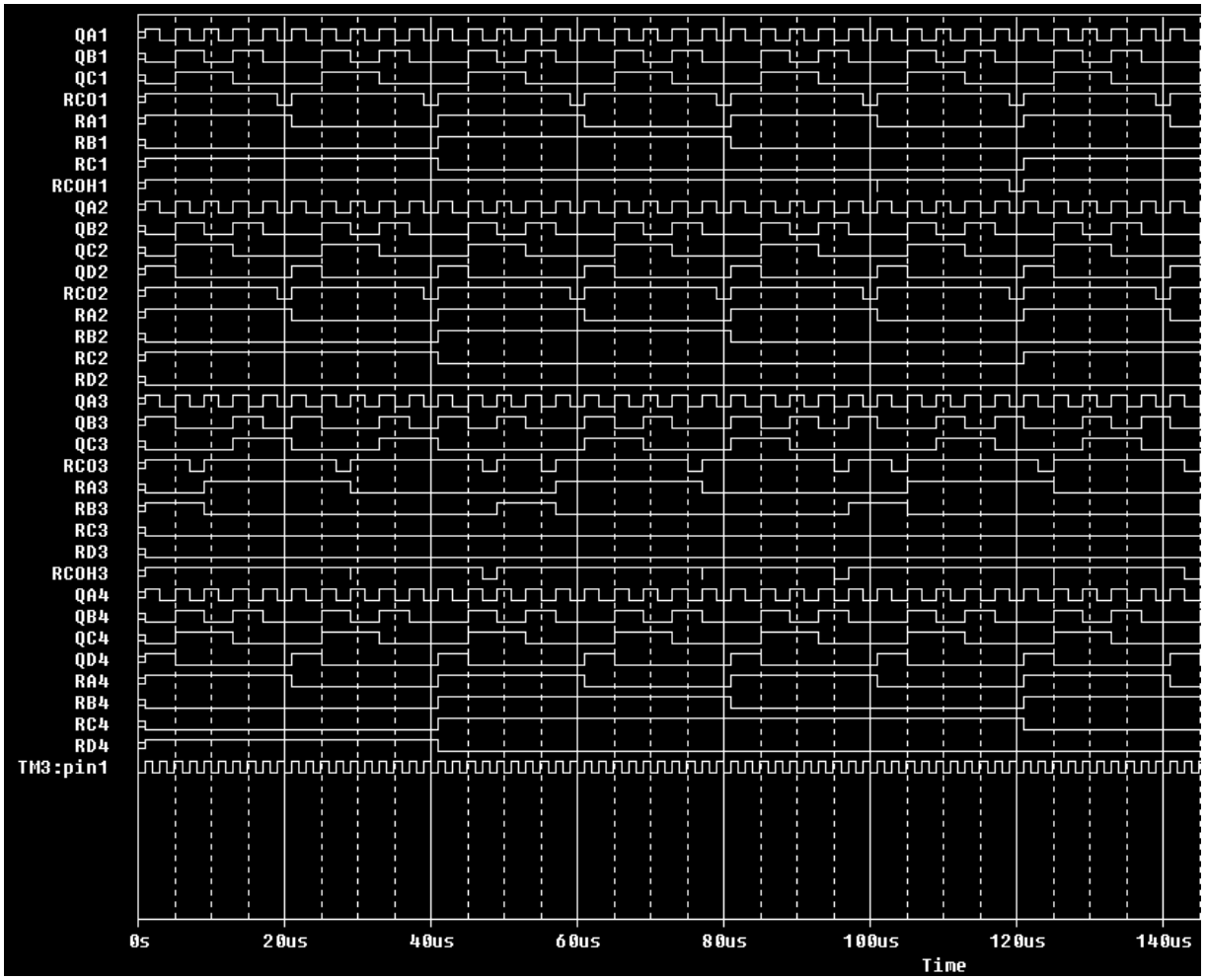

図 3.3.9　図 3.3.8 のシミュレーション結果

スイッチの構造を図 3.3.10 に示す。このスイッチは初期値を設定できるよう取り付けたもので、スイッチを押すことで桁下げの信号を発生させ、SN74LS169B がクロックに同期してカウントダウンしていくものである。

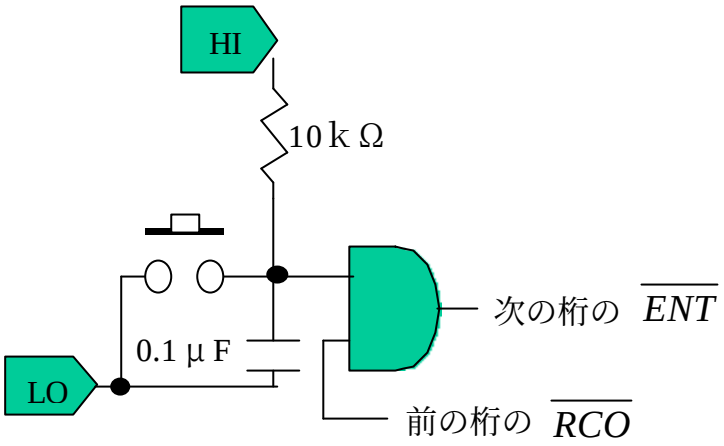

図3.3.10　スイッチの構造

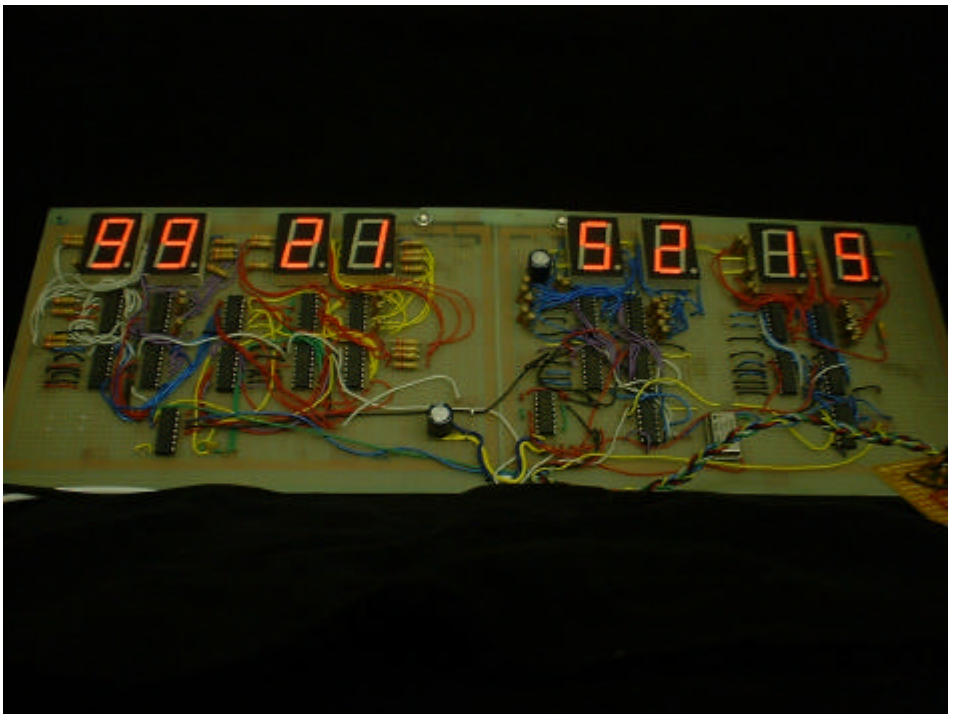

図 3.3.11　カウントダウン時計

図 3.3.11 は図 3.3.8 と図 3.3.10 をもとに作製したものである。

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　ハードウェア記述言語（HDL）

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）の試みが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。

従来、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（Hardware Description Language：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 C 言語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、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（同時処理）や時間、タイミングの概念などを記述できる特徴をもつ。

HDL はその長い歴史において、非常に多種多様な言語が開発・提案されてきた。その中で、標準化が進められ、現在広く用いられている言語に、VHDL と Verilog-HDL がある。

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長がある。

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。

4.2 VHDL

SR ラッチの VHDL 記述例

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

entity  RS_FF  is
    port  ( R,S     : in   std_logic;
            Q,Qnot  : out  std_logic  );
end  RS_FF ;

architecture  RTL  of  RS_FF  is

signal  S1,S2  : std_logic ;

begin
        S1      <= R nand S2;
        S2      <= S nand S1;
        Q       <= S2;
        Qnot    <= S1;
end  RTL ;
```

図4.2.1　SRラッチのVHDL記述

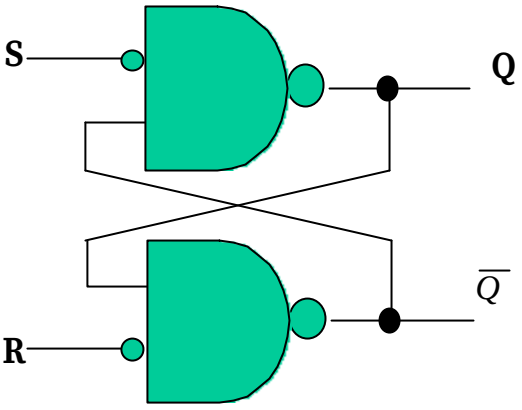

図4.2.2　SRラッチ

表4.2.1　SRラッチの状態表

S	R	Q
0	0	前の状態を保持
0	1	0（リセット）
1	0	1（セット）
1	1	×（不定）（禁止）

VHDL は「エンティティ（entity）」と「アーキテクチャ（architecture）」を必要とする。エンティティの記述によって、外部から入力される信号や外部へ出力する信号に関する情報、すなわち、インターフェースに関する情報を設定している。また、アーキテクチャの記述によって回路内部の動作や構造に関する情報を設定している。

上記において、エンティティ記述部では「RS_FF」というエンティティ名の回路が R と S という二つの入力ポートをもち、Q と Qnot の二つの出力ポートをもっていることを宣言している。

また、アーキテクチャの記述部では「RS_FF」に対して、RTL という名前のアーキテクチャを宣言し、アーキテクチャ本体として、RS フリップフロップの構造を記述している。

IEEE ライブラリパッケージには一般的に、「IEEE.std_logic_1164」を用いる。その他に、配列タイプ「std_logic_vector」があり、これを用いて算術演算や大小比較などを行う場合には算術用パッケージを使用する必要がある。

IEEE ライブラリの算術演算用パッケージ

① numeric_std パッケージ
符号ビットなし演算と符号ビットあり演算を混在させるためのパッケージ

② std_logic_unsigned パッケージ
符号ビットなしの演算を行うためのパッケージ

③ std_logic_signed パッケージ
符号ビットありの演算を行うためのパッケージ

④ std_logic_arith パッケージ
符号ビットなし演算と符号ビットあり演算を混在させるためのパッケージ

VHDL はエンティティとアーキテクチャを記述することで回路を設計しているため、論理合成回路を先に考えて VHDL 記述しなくても設計することができる。
その例として、「4 進ダウンカウンタ」の VHDL 記述による設計をあげる。

① 論理合成回路による 4 進ダウンカウンタの VHDL 記述の実現

JK フリップフロップを用いた 4 進ダウンカウンタ

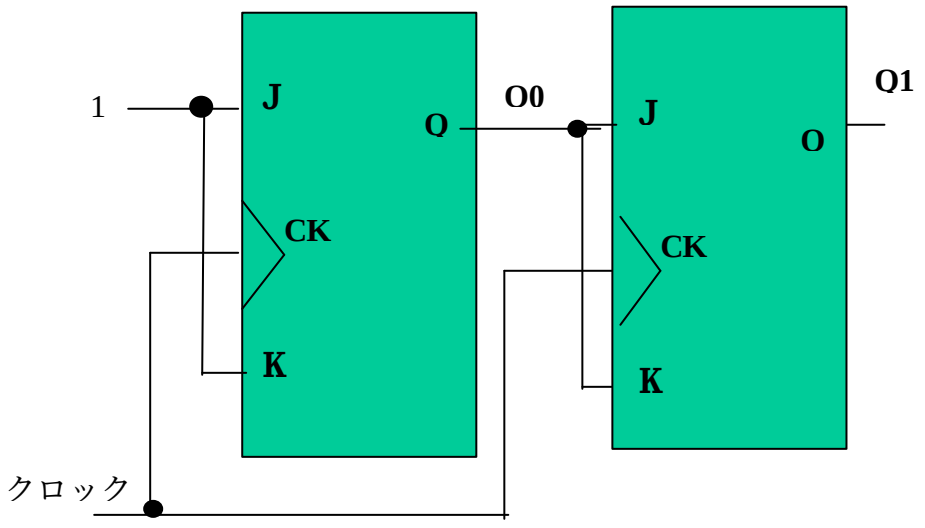

図4.2.3ＪＫフリップフロップによる同期式4進カウンタ

上の図 4.2.3 の 4 進ダウンカウンタを VHDL 記述する場合、まず JK フリップフロップ

の VHDL 記述をする。（**library** にない場合や **library** ごと自分で作る場合）

```
library    IEEE
use    IEEE.std_logic_1164.all
entity    JK_FF is
    port(   CK, J, K          : in std_logic;
            Q, Qnot           : out std_logic );
end    JK_FF ;
architecture    RTL    of    JK_FF    is
signal    INPUT       : std_logic_vector(2 downto 0);
signal    TMP         : std_logic;

begin
   INPUT <= CK    &    J    &    K;
   process    ( INPUT )    begin
     case    INPUT    is
       when    "101"    =>    TMP    <=    '0';
       when    "110"    =>    TMP    <=    '1';
       when    "111"    =>    TMP    <=    not TMP;
       when    others    =>    null;
    end    case;
  end    process;
  Q        <=    TMP;
  Qnot    <=    not TMP;
end    RTL;
```

図 4.2.4　JK フリップフロップの VHDL 記述

JK フリップフロップの VHDL 記述後、4 進カウンタを記述したものが図 4.2.5 である。

```
library    IEEE
use    IEEE.std_logic_1164.all

entity    COUNTER_4_JK_FF  is
    port(   CK, J, K    : in std_logic;
          Q, Qnot    : out std_logic );
end    COUNTER_4_JK_FF ;

architecture    RTL of COUNTER_4_JK_FF    is
component    JK_FF
port(
    CK, J, K    : in std_logic;
    Q, Qnot     : out std_logic );
end    component;

signal    J0,    K0,    J1,    K1,    Q0,    Q1,   : std_logic;
begin
   J0    =    K0    <=    '1';
   J1    =    K1    <=    Q0 ;
  COMP0   :   JK_FF    port map ( CK,    J0,    K0,    Q0 );
  COMP2   :   JK_FF    port map ( CK,    J1,    K1,    Q1 );
end    RTL;
```

図 4.2.5　JK フリップフロップを用いた 4 進カウンタの VHDL 記述

②　VHDL 記述による 4 進ダウンカウンタの実現

①では論理合成回路図を考えて VHDL 記述したが、アーキテクチャ部で **if** 文を使って、「A という信号が入力されれば B という信号を出力する。」と記述することで、論理合成回路図なしに 4 進カウンタを設計できる。
その例を図 4.2.6 に示す。

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

entity   COUNTER_4  is
    port   ( CK,   RESET,   X   : in    std_logic;
                   Y              : out std_logic   );
end   COUNTER_4 ;

architecture   RTL   of   COUNTER_4  is

signal  COUNT : std_logic_ vector(1 downto 0);

begin
    Y   <=   COUNT(0)  and  COUNT(1)
    process  (  RESET,  CK  )  begin
            if  (  RESET = ‘ 1 ’ )  then
                COUNT  <=  ”00 ”  ;
           elsif  ( CK ’ event  and  CK  = ‘ 0 ’ )  then
               if  ( X = ‘ 1 ’ )   then
                  if  ( COUNT = “ 11 ” )  then
                       COUNT  <=  “ 00 ”  ;
                  else
                       COUNT  <=  COUNT  +  “ 01 ” ; ;
                  end  if  ;
             end  if  ;
      end  if  ;
  end  process  ;
end   RTL ;
```

図 4.2.6　4 進カウンタの VHDL 記述

4.3　設計例　―　カウントダウン時計　―

VHDL による順序回路の設計例としてカウントダウン回路及び、カウントダウン時計の設計について述べる。

設計にあたって下記の 3 つの工程で進める。

① ７セグメント部の記述
② 10 進ダウンカウンタの記述
③ カウントダウン時計の設計

ここで記述したカウントダウン時計の回路は第三章で作製した回路の組み合わせ回路、カウンタ、デコーダをひとつのブラックボックスにまとめたものである。

（1）７セグメント部の記述

７セグメント部の記述にあたって、必要なものは入力が４ビットに対し出力が 7 ビットであるもの、すなわちデコーダである。図 4.3.1 は 7 セグメント部を VHDL 記述したものである。

```
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

      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;
```

図 4.3.1　7セグメント部の VHDL 記述

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

entity TESTBNCH is
end TESTBNCH;
use work.all;

architecture stimulus of TESTBNCH is
component    segment7
    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
       DUT : segment7 port map(DI, DO);
  process    begin
        DI <= "0000"; wait for 50ns;
        DI <= "0001"; wait for 50ns;
        DI <= "0010"; wait for 50ns;
        DI <= "0011"; wait for 50ns;
        DI <= "0100"; wait for 50ns;
        DI <= "0101"; wait for 50ns;
        DI <= "0110"; wait for 50ns;
        DI <= "0111"; wait for 50ns;
        DI <= "1000"; wait for 50ns;
        DI <= "1001"; wait for 50ns;

  end process;
end stimulus;
```

図 4.3.2　7セグメント部のテストベンチ

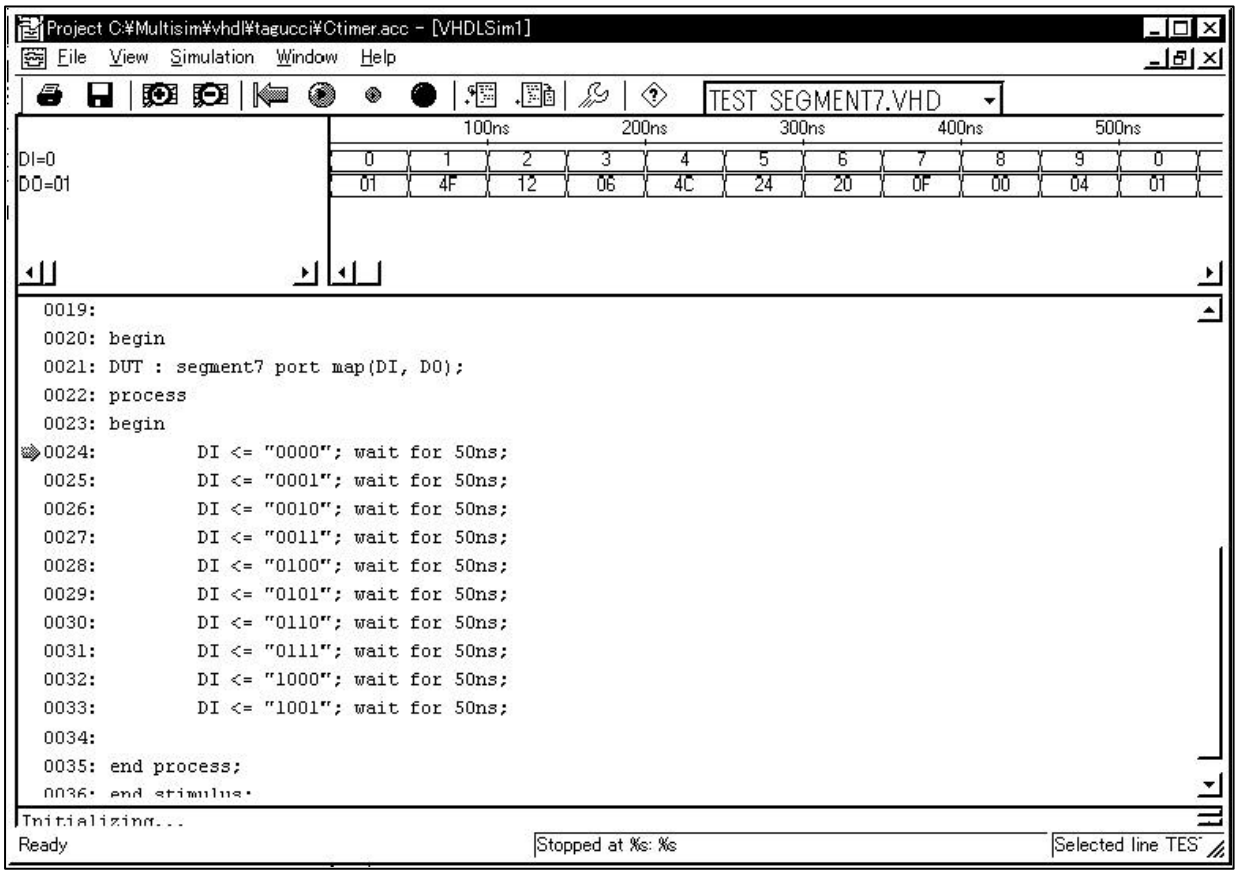

図 4.3.3　　図 4.3.2 によるシミュレーション結果

（2）10 進ダウンカウンタの記述

1 桁を表すのに 9～0 までの 10 進でよいので 10 進ダウンカウンタを記述したものが図 4.3.4 である。

```
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
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;                                          -- 次のページへ
```

図 4.3.4_1　10 進ダウンカウンタの VHDL 記述（その 1）

```
process (TQ,CYIN) begin
   if (TQ = "0000" and CYIN = '1') then
      CYOUT <= '1';
   else
      CYOUT <= '0';
  end if;
end process;
   Q <= TQ;

end RTL;
```

図 4.3.4_2　10 進ダウンカウンタの VHDL 記述（その 2）

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

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;                                  --次のページへ
```

図 4.3.5_1　10 進ダウンカウンタのテストベンチ（その 1）

```
signal CLK:      std_logic;
signal RESET:    std_logic;
signal CYIN:     std_logic;
signal Q:        std_logic_vector(3 downto 0);
signal CYOUT:          std_logic;

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

CLOCK: process
     begin
      CLK <= '1'; wait for 20ns;
      CLK <= '0'; wait for 20ns;
end process;

stimulus: process
     begin
      RESET <= '1'; wait for 15ns;
      RESET <= '0'; CYIN <= '1'; wait;
   end process ;

end stimulus;
```

図 4.3.5_2　10 進ダウンカウンタのテストベンチ（その 2）

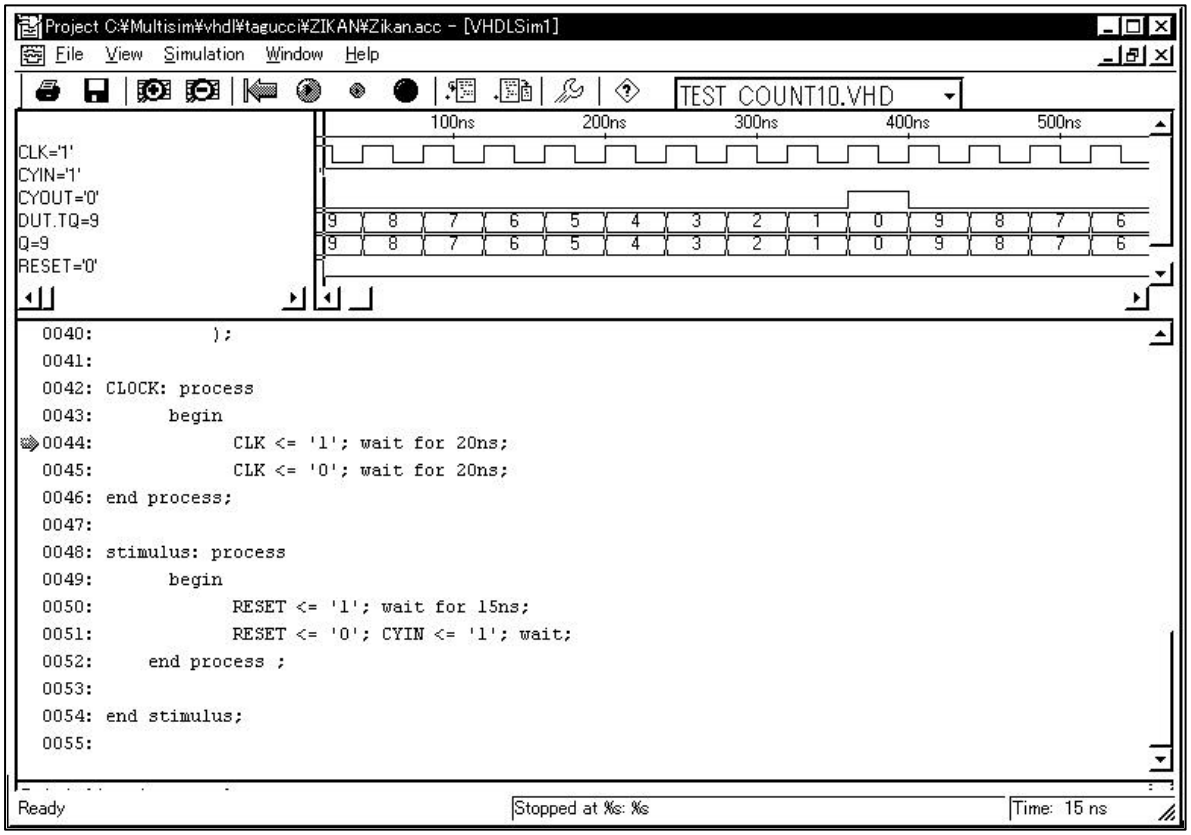

図 4.3.6　　図 4.3.5 によるシミュレーション結果

（3）カウントダウン時計の設計

①　②で記述した 7 セグメント部と 10 進ダウンカウンタをコンポーネント宣言で呼び出し、入力はクロックとクリアの 2 つ、出力は 7 ビットのセグメントを表示する信号で秒、分、時間、日を 2 桁ずつの 8 の回路を記述したものが図 4.3.7 である。

```
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
   SZF, SZS    : out std_logic_vector(6 downto 0);--zikan
   SHIF, SHIS  : out std_logic_vector(6 downto 0) --hiniti
 );
end CDT;                                          --次のページへ
```

図 4.3.7_1　カウントダウン時計の VHDL 記述（その 1）

```
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
```

--次のページへ

図 4.3.7_2　カウントダウン時計の VHDL 記述（その 2）

```
--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;
```

--次のページへ

図 4.3.7_3　カウントダウン時計の VHDL 記述（その 3）

```
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;
```

--次のページ

図 4.3.7_4　カウントダウン時計の VHDL 記述（その 4）

```
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;

--zikan
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 (TQn <= "0000" and TQm <= "0000" ) then
         TQm <= "0011";  end if;
       else
          TQm <= TQm - '1';
     end if;
   end if;
end process;                                          --次のページへ
```

図 4.3.7_5　カウントダウン時計の VHDL 記述（その 5）

```
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;
```

--次のページへ

図 4.3.7_6　カウントダウン時計の VHDL 記述（その 6）

```
--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;
```

--次のページへ

図 4.3.7_7　カウントダウン時計の VHDL 記述（その 7）

```
--7seg
U0 : segment7 port map(TQi, SBF);--byou
U1 : segment7 port map(TQj, SBS);
U2 : segment7 port map(TQk, SHF);--fun
U3 : segment7 port map(TQl, SHS);
U4 : segment7 port map(TQm, SZF);--zi
U5 : segment7 port map(TQn, SZS);
U6 : segment7 port map(TQo, SHIF);--hi
U7 : segment7 port map(TQp, SHIS);

end RTL;
```

図 4.3.7_8　カウントダウン時計の VHDL 記述（その 8）

```
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, RESET:    in std_logic;
      SBF, SBS:   out std_logic_vector(6 downto 0);
      SHF, SHS:   out std_logic_vector(6 downto 0);
      SZF, SZS:   out std_logic_vector(6 downto 0);
      SHIF, SHIS: out std_logic_vector(6 downto 0);
     );
end component;                                --次のページへ
```

図 4.3.8_1　カウントダウン時計のテストベンチ（その 1）

```
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 SZF:    std_logic_vector(6 downto 0);
signal SZS:    std_logic_vector(6 downto 0);
signal SHIF:   std_logic_vector(6 downto 0);
signal SHIS:   std_logic_vector(6 downto 0);

begin
    DUT: CDT port map (
            CLK, RESET,
           SBF, SBS,
           SHF,  SHS,
            SZF,  SZS,
           SHIF,  SHIS
         );

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

    STIMULUS1: process   begin
       RESET <= '1';  wait for 2sec;
       RESET <= '0';  wait;
    end process STIMULUS1;
end stimulus;
```

図 4.3.8_2　カウントダウン時計のテストベンチ（その 2）

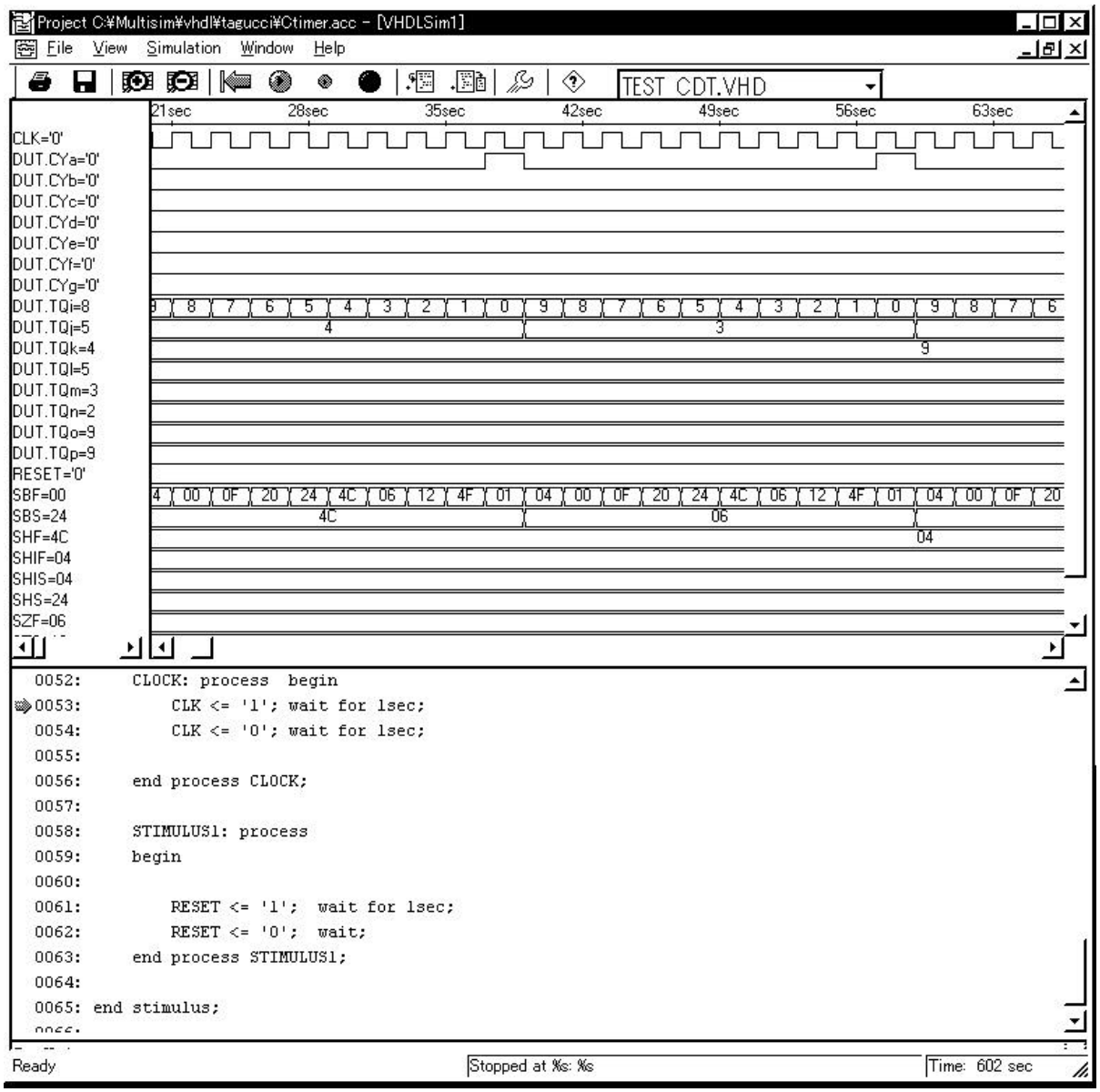

図 4.3.9　　図 4.3.8 によるシミュレーション結果

以上のように、VHDL によるカウントダウン時計回路はその動作を完全に行うことができた。

第 5 章　おわりに

今回行った標準論理 IC を用いた設計、ハードウェア記述言語を用いた設計で感じたことは、ハードウェア記述言語を用いると簡単な回路は論理回路図を考え、設計するよりも格段に早く設計することができるが、すこし回路が複雑になると記述はできるが、その記述が完全なものであるかの信頼性は低くなる。これについては、まだ自分のハードウェア記述言語に対する理解がまだ浅いためである。

今回の VHDL を用いた設計でも自分なりの記述はできたのだが、テストベンチを正しく記述するのに苦労した。テストベンチの記述を試行錯誤していくうちにコツをつかむことができ、以外に簡単に記述できることがわかった。

テストベンチは記述できたとしても、シミュレーションがうまくいくとは限らず、このうまくいっていないシミュレーション結果から VHDL 記述及び、テストベンチ記述の誤りを見つけ修正させるのに苦労した。しかし、この苦労がよい勉強になった。

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 北野克幸氏、小泉義信氏、小島大輔氏、幾井崇博氏、中村基継氏、小松輝将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。