卒 業 研 究 報 告

題　　目

VHDLによるディジタル時計の設計

指 導 教 員

矢野　政顕教授

報 告 者

学籍番号：1030192
下江　毅

平成　15年　2月　10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・・・・・・・・・・・・・・・・・・・・・・1

第 2 章　順序回路の基本事項・・・・・・・・・・・・・・・・2

2.1 順序回路・・・・・・・・・・・・・・・・・・・・・・・・・2

2.2 フリップフロップ・・・・・・・・・・・・・・・・・・・・・5

2.3 ラッチ・レジスタ・・・・・・・・・・・・・・・・・・・・11

2.3.1　ラッチ・・・・・・・・・・・・・・・・・・・・・・・・12
2.3.2　レジスタ・・・・・・・・・・・・・・・・・・・・・・・12

2.4 カウンタ・・・・・・・・・・・・・・・・・・・・・・・・12

2.4.1　同期式カウンタ・・・・・・・・・・・・・・・・・・・12

2.4.2　非同期式カウンタ・・・・・・・・・・・・・・・・・・13

2.5 状態遷移図と状態遷移表・・・・・・・・・・・・・・・・13

2.5.1　状態遷移図・・・・・・・・・・・・・・・・・・・・・13
2.5.2　状態遷移表・・・・・・・・・・・・・・・・・・・・・14

2.6 デコーダ・エンコーダ・・・・・・・・・・・・・・・・・15

2.6.1　デコーダ・・・・・・・・・・・・・・・・・・・・・・15
2.6.2　エンコーダ・・・・・・・・・・・・・・・・・・・・・16

第 3 章　標準ロジック IC による回路設計・・・・17

3.1　標準ロジック IC・・・・・・・・・・・・・・・・・・・17

3.2　TTL ファミリ・・・・・・・・・・・・・・・・・・・・・17

3.3　CMOS ファミリ・・・・・・・・・・・・・・・・・・・・20

3.4　クロック分周回路・・・・・・・・・・・・・・・・・・21

3.5 水晶発振器・・・・・・・・・・・・・・・・・・・・・・・・21

3.6 60 進カウンタ・・・・・・・・・・・・・・・・・・・・・・22

3.7 7 セグメント・・・・・・・・・・・・・・・・・・・・・・・22

3.8 チャタリング防止回路・・・・・・・・・・・・・・・・・・25

3.9 ディジタル時計の設計・・・・・・・・・・・・・・・・・26

3.10 ディジタル時計の回路図と回路シミュレーション・27

第 4 章 VHDL による時計の設計・・・・・・・・・・・34

4.1 VHDL とは？・・・・・・・・・・・・・・・・・・・・・・34

4.2 VHDL の歴史・・・・・・・・・・・・・・・・・・・・・・34

4.3 VHDL 設計のメリット・・・・・・・・・・・・・・・・・35

4.4 VHDL によるディジタル時計の設計・・・・・・・37

4.4.1 7 セグメント表示器・・・・・・・・・・・・・・・・・・・・37

4.4.2 60 進カウンタ（分,秒）・・・・・・・・・・・・・・・・・40

4.4.3 24 進カウンタ（時）・・・・・・・・・・・・・・・・・・・47

4.4.4 時、分、秒の合成・・・・・・・・・・・・・・・・・・・・50

第 5 章 まとめ・・・・・・・・・・・・・・・・・・・・・・・58

謝辞・・・・・・・・・・・・・・・・・・・・・・・・・・・・59

参考文献・・・・・・・・・・・・・・・・・・・・・・・・・60

付録 1・・・・・・・・・・・・・・・・・・・・・・・・・・・61

付録 2・・・・・・・・・・・・・・・・・・・・・・・・・・・65

第 1 章　はじめに

ディジタル電子計算機を中心とするディジタル電子技術の発展は著しく、従来アナログ電子回路の分野で取り扱われていた種々の信号も、次第にディジタル電子技術により処理されるようになってきた。また、集積回路技術の進歩に伴い、ディジタル電子回路の集積度は指数関数的に増大し、ディジタルシステムも、ますます複雑化、巨大化の傾向にある。このような複雑なディジタルシステムも、基本的には数種類の論理ゲートとフリップフロップ回路の組合に過ぎず、この基本回路の動作を充分に把握しておくことは、いかなる複雑なディジタルシステムの設計、解析にも重要である。

卒業研究では、ディジタル回路設計をより理解するために、身近にあるディジタル時計をとりあげ、標準ロジック IC による設計と VHDL による設計を行った。

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、順序回路の設計に関する基本事項について述べている。第 3 章では、標準ロジック IC を使用した順序回路の設計について述べるとともにプリント基板による試作結果について報告している。第 4 章では、VHDL による設計方法を述べ、その実例を示している。最後に第 5 章では、全体をまとめている。

第2章　順序回路の基本事項

2.1　順序回路

基本論理ゲートを用いて作られる組み合わせ論理回路は、現在の入力がそのときの出力を決定し、入力が変化すると出力もそれに応じて変化する。一方、カウンタでは、入力にいくつかのパルスが入ると、出力が変化する。これは、出力が過去の入力に依存していることを意味している。このように、出力が過去の入力に依存して決定される回路を順序回路(sequential circuit)という。

順序回路では、出力が現時点での入力だけでは定まらず、過去の入力にも依存する。すなわち、過去の入力系列（input sequence）に依存する情報を記憶しており、出力は現在の入力と記憶している情報により定まる。この情報を回路の内部状態（internal state）あるいは単に状態（state）という。また、回路の入力を特に外部入力（external input)、出力を外部出力（external output）ということがある。

順序回路は、通常、図 2.1 に示すように組み合わせ回路と記憶回路を用いて構成されている。 x1、 x２・・・、 xn は入力変数、z1、z2、・・・、zm は出力変数、y1、y2、・・・yk は状態変数である。順序回路では現時点での入力変数の値と状態変数の値から、出力変数の値と次の時点での状態変数の値が決まる。〔3〕

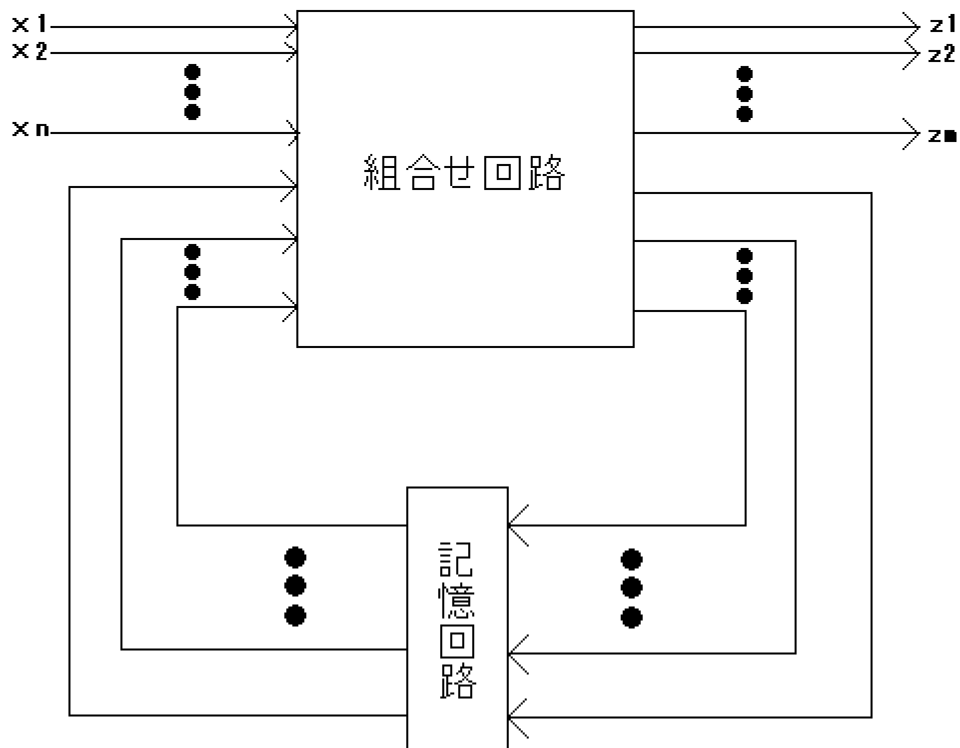


図 2.1　順序回路の構成

同期式回路・非同期式回路

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。同期式では、ある基準となるパルスにタイミングを合わせて、各部の状態やパルスの立ち上がり、立ち下りなどが決まる。図 2.2.に、同期信号と非同期信号の例を示す。クロックパルス CK の立ち下りの時刻に合わせて、信号 A および B のパルスは立ち上がったり、立ち下がったりしている。これらのパルスをクロック CK とは無関係に立ち上がり、また立ち下がっている。このようなパルスを非同期パルスという

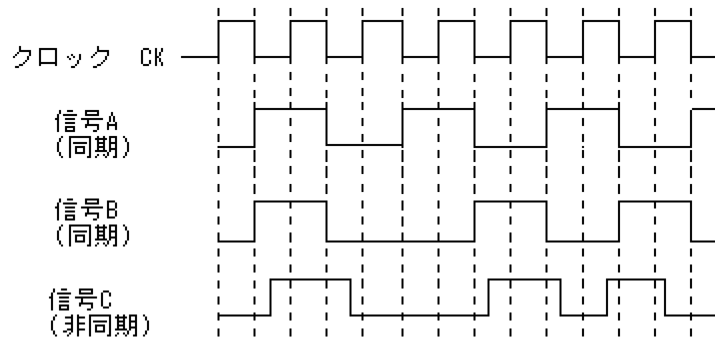


図 2.2.同期・非同期パルス

順序回路では、フリップフロップなどに記憶した状態をあるタイミングで取り出して、さらに演算を行ってから、新しい出力を決定するなどの動作を行うことが多い。そのため、多くの場合、順序回路には同期式の信号が使われる。このような順序回路を同期式順序回路という。

同期式順序回路、非同期式順序回路をそれぞれ図 2.3、図 2.4 に示す。

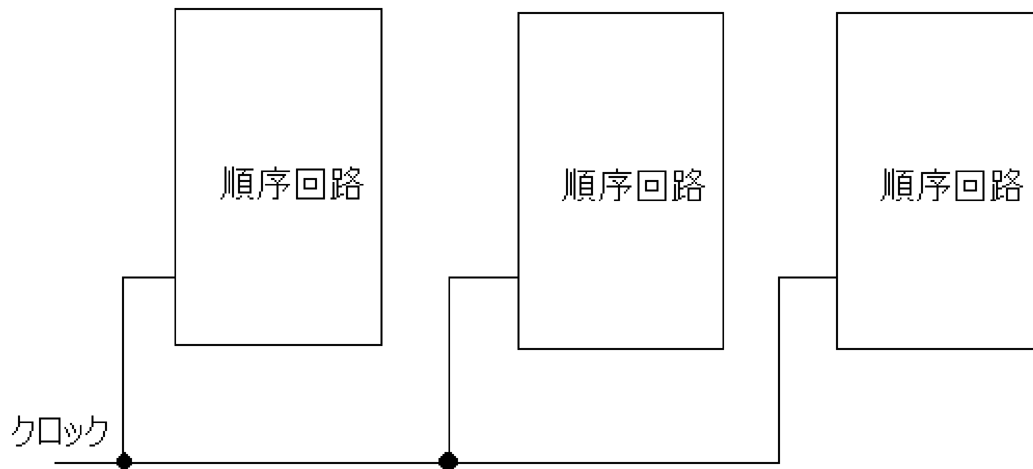


図 2.3 同期式順序回路

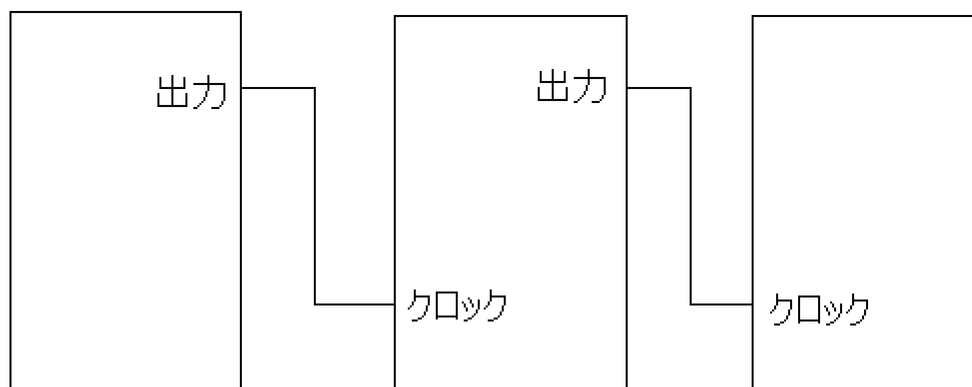


図 2.4 非同期式順序回路

2.2　フリップフロップ

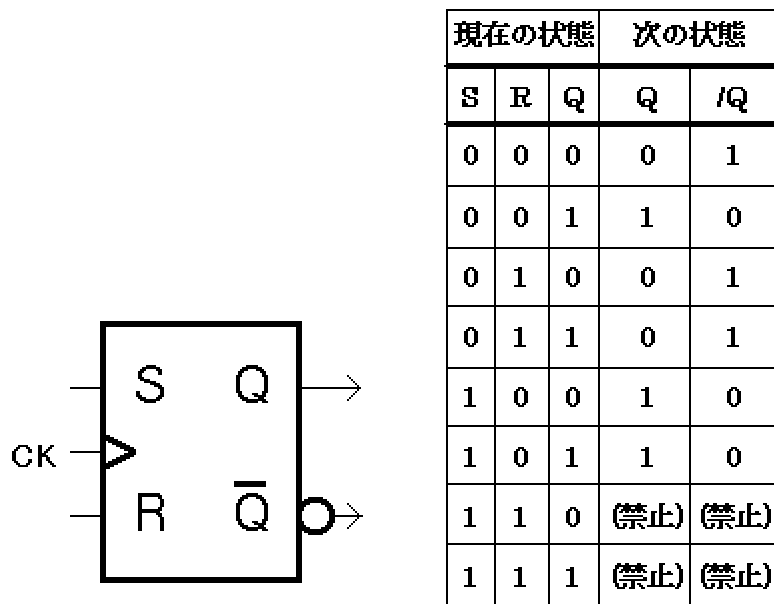
順序回路を実現するには、内部状態を記憶する記憶回路が必要である。記憶回路を構成する記憶素子として、フリップフロップ(flip-flop)がある。フリップフロップは NAND ゲートなどの論理ゲートをいくつか接続して構成される。組み合わせ回路と異なり、フィードバックループ(feedback loop)をもつ。フィードバックループは、ある論理ゲートからの出力を辿っていくと、再び元の論理ゲートに戻るような配線のループである。

フリップフロップは 2 つの状態をもち、クロックパルスが加えられない限り、状態を保持し続ける。代表的なフリップフロップとして、SR-FF、D-FF、JK-FF、T-FF がある。

SR-FF

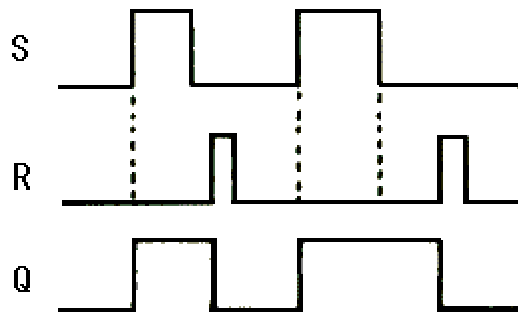
S と R はそれぞれセットとリセットを表し、S=1、R=0 ならセット、すなわち、次の時刻の状態(出力)が Q(t+1)=1 となり、S=0,R=1 ならリセット(Q(t+1)=0)、S=0、R=0 なら現状態の保持(Q(t+1)=Q(t))である。S=1,R=1 は禁止入力であり、この入力に対する状態は不定である。次状態は、$Q(t+1)=S+\overline{R}\,Q(t)$、ただし S・R＝0 である。

ＳＲ-ＦＦの回路記号と特性表、タイミングチャートを図 2.5 に示す。



現在の状態			次の状態	
S	R	Q	Q	/Q
0	0	0	0	1
0	0	1	1	0
0	1	0	0	1
0	1	1	0	1
1	0	0	1	0
1	0	1	1	0
1	1	0	(禁止)	(禁止)
1	1	1	(禁止)	(禁止)

回路記号　　　　　　　　　　特性表



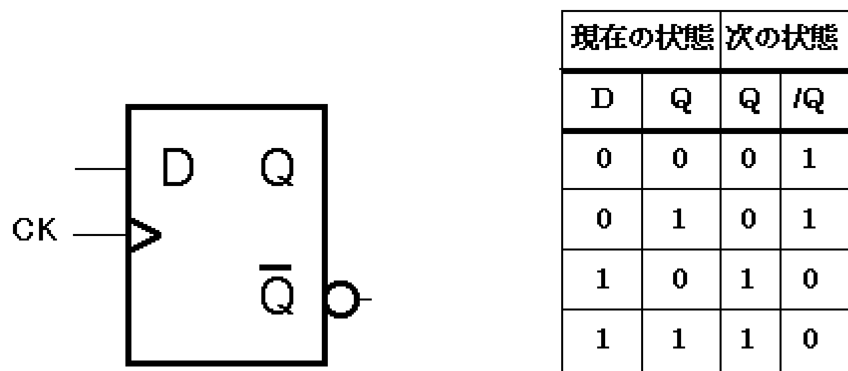
タイミングチャート

図 2.5　ＳＲ-ＦＦ

D-ＦＦ

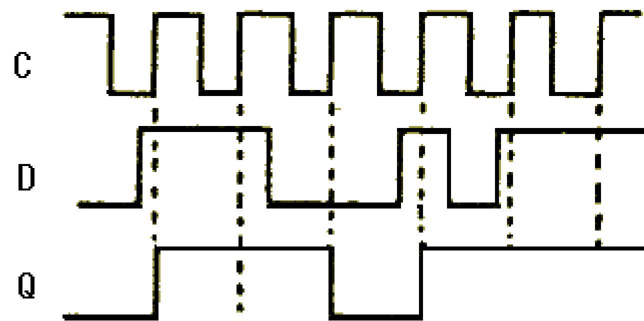
ＳＲ-ＦＦの入力をR=$\bar{S}$とすることにより構成できる。　D=1 ならＱ(ｔ+1)=1、D=0 ならＱ(ｔ+1)=Ｄとなる。すなわち、現在の入力が次の時刻の状態(出力)になる。Ｄは遅延(delay)を意味する。Q(t+1)=D である。CMOS では、NOT ゲートとトランスファゲートと呼ばれるゲートで構成された D-FF が用いられることが多い。

D-FF の回路記号と特性表、タイミングチャートを図 2.6 に示す。



現在の状態		次の状態	
D	Q	Q	/Q
0	0	0	1
0	1	0	1
1	0	1	0
1	1	1	0

回路記号　　　　　　　　　　特性表



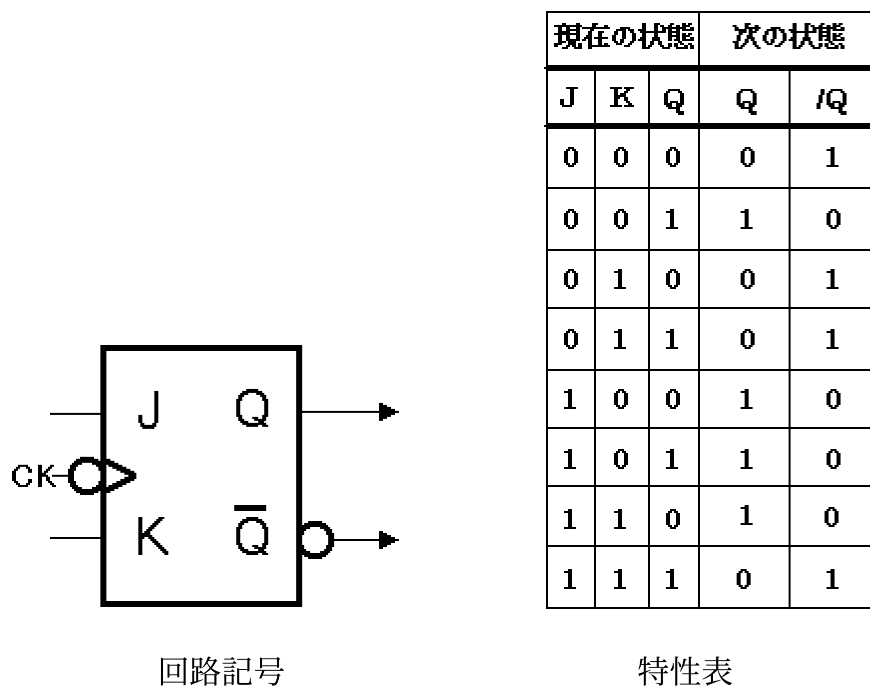
タイミングチャート

図 2.6　D-FF

JK-FF

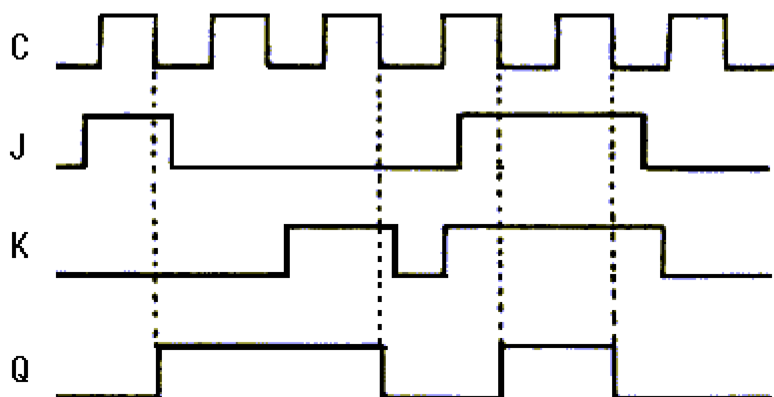
SR-FF の禁止入力をなくしたもので、J=K=1 のとき状態の反転、すなわち、$Q(t+1)=\overline{Q(t)}$ になる。$Q(t+1)=J\overline{Q}+\overline{K}Q(t)$である。

同期式 JK フリップフロップの回路記号と特性表、タイミングチャートを図 2.7 に示す。



現在の状態			次の状態	
J	K	Q	Q	/Q
0	0	0	0	1
0	0	1	1	0
0	1	0	0	1
0	1	1	0	1
1	0	0	1	0
1	0	1	1	0
1	1	0	1	0
1	1	1	0	1

回路記号　　特性表



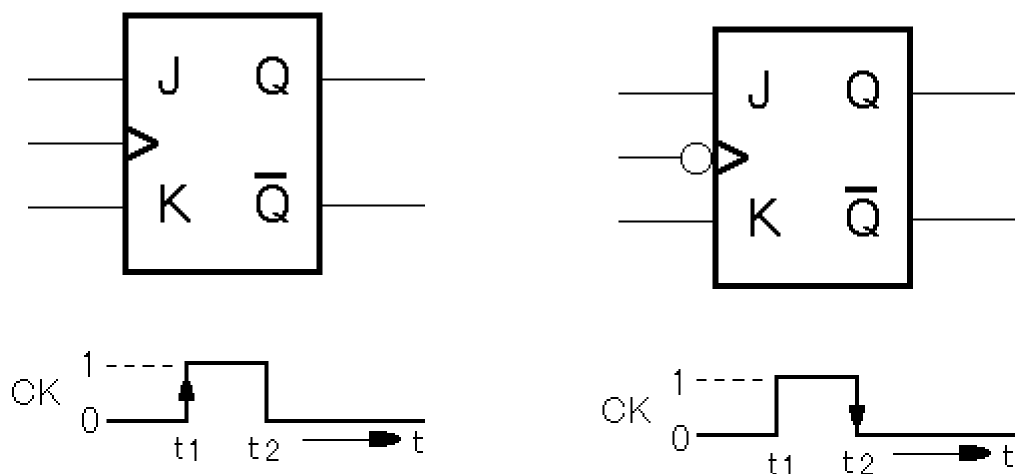
タイミングチャート

図　2.7　JK-FF

図 2.8 にエッジトリガ JK フリップフロップの記号を示す。図 2.8(a)は CK 端子のパルスの立ち上がり部分で出力が変化するもので,ポジティブエッジトリガ JK フリップフロップという。図に示すようにタイミングチャート上では、CK 入力パルスの立ち上がり部分に矢印を記入して、ポジティブエッジトリガ動作であることを表す。

図 2.8(b)は、CK 端子のパルスの立ち下がり部分で,出力が変化するネガティブエッジトリガ JK フリップフロップである。

いずれの場合も、時刻ｔ１とｔ２の間では、CK＝1 の状態が続いてはいるが、この間ではフリップフロップの出力は変化しないのが,エッジトリガフリップフロップの特徴である。



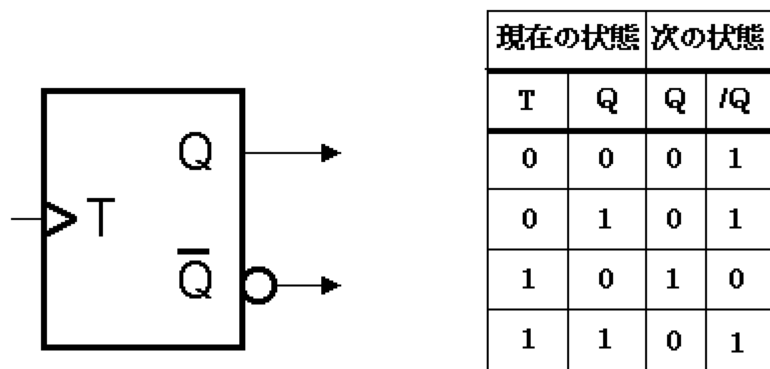
(a)　ポジティブエッジトリガ　　(b)ネガティブエッジトリガ

図 2.8　エッジトリガ JK フリップフロップ

T-FF

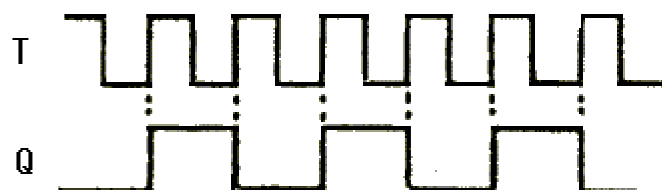
JK-FF の入力を J=K とすることにより構成できる。T=1 なら状態の反転(Q(t+1)=Q(t))、T=0 なら状態の保持(Q(t+1)=Q(t))である。Q(t+1)=TQ(t)'+T'Q(t)である。

T-FF の回路記号と特性表、タイミングチャートを図 2.9 に示す。



現在の状態		次の状態	
T	Q	Q	/Q
0	0	0	1
0	1	0	1
1	0	1	0
1	1	0	1

回路記号　　　　　　　　　　　特性表



タイミングチャート

図 2.9　T-FF

FF を順序回路の記憶回路部に用いる場合、FF の状態(出力)Q が順序回路の状態変数に対応する。次時刻に、状態変数を所望の値にする、すなわち、FF が所望の状態遷移をするようにするには、FF に適切な適切な入力を与える必要がある。各 FF について各状態遷移を起こす入力を表 2.1 に示す。表中の「ー」は 0 と 1 のどちらでもよいことを表している。順序回路の組合せ回路部では、次状態変数関数そのものではなく、FF を駆動する関数、すなわち励起関数(excitation function)を計算する。D-FF の場合は、励起関数は次状態変数関数に等しくなる。励起関数においては,FF の入力が 0 と 1 のどちらでもよい部分は、ドントケアな組になる。

表 2.1　フリップフロップの駆動条件

Q(t)	→ Q(t+1)	S	R	D	J	K	T
0	0	0	–	0	0	–	0
0	1	1	0	1	1	–	1
1	0	0	1	0	–	1	1
1	1	–	0	1	–	0	0

いったんフリップフロップの出力の状態が定まると、入力を取り除いても、出力は変化しない。したがって、出力を定めるための入力は、いつまでも入れ続ける必要はない。このような場合には、図 2.10 に示すような電圧が使用される。電圧は,時刻 t1 と t2 の間だけ、高い値(すなわち 1 の状態)を持ち、ほかの時刻では低い値(ふつう 0V)となっている。このように、時間的に短い間だけ、”1”となるような波形をパルスと呼んでいる。

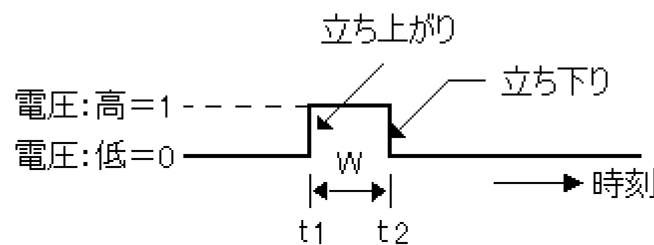


図 2.10　パルス

図 2.10 に、パルスが 0 から 1 に変化する部分、パルスの立ち上がりと立ち下がりを示している。理想的なパルスでは,立ち上がりと立ち下がりに要する時間は 0 である。すなわち、瞬間的に 0 から 1 へ、また、1 から 0 へ変化するのが、理想的なパルスなのである。t 1 から t 2 までの時間 W をパルス幅という。〔1〕

2.3 ラッチ・レジスタ

ラッチ･レジスタとはフリップフロップの集まり(複数個のフリップフロップを並べたもの)をラッチ(latch)とかレジスタ(register)という。“レジスタ長(ビット)”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(read)とフリップフロップへの書き込み(write)/状態更新とがあり、各ビットは同時(並列)に動作する。

2.3.1　ラッチ

ラッチとは掛けがね、かんぬきという意味で、クロック入力時のデータを記憶し後は受け付けないことに基づく名称である。〔2〕

2.3.2 レジスタ

1 個のフリップフロップは、1 ビットの 2 進データを記憶することができるから、これを図 2.11 のように n 個用いることにより、n ビットの 2 進データを記憶できる回路が作れる。これをメモリレジスタ、あるいは単にレジスタ(置数器)という。〔2〕

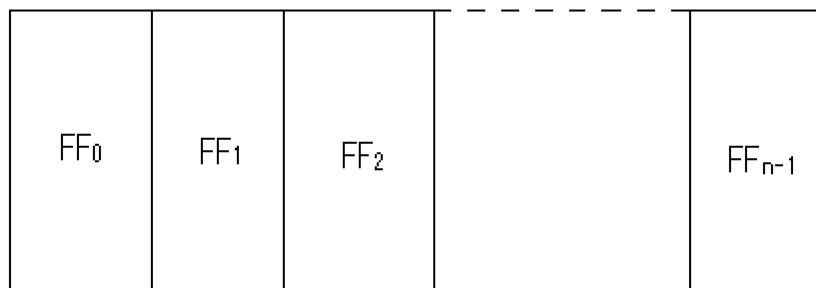


図 2.11　n ビットの記憶回路

2.4　カウンタ

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順序回路をカウンタ(counter)という。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合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全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を同期カウンタという。〔2〕

2.4.2 非同期式カウンタ

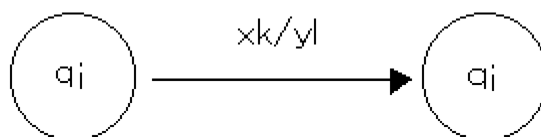
各順序回路が共通のクロックにより同期しておらず、別々のタイミングで動作しているカウンタである。クロックパルスに同期しないで前段のフリップフロップの出力が次段のフリップフロップのクロック入力に入力されて動作するのでフリップフロップの数が多いほど遅れが蓄積されてくる。〔2〕

2.5　状態遷移図と状態遷移表

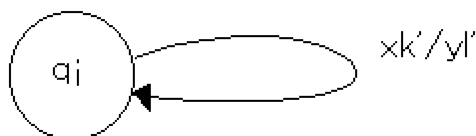
2.5.1 状態遷移図

順序回路は、記憶作用を持たない組合せ論理回路と、情報の一時記憶が可能な記憶回路あるいは遅延回路より構成される。記憶あるいは遅延回路としては、各種フリップフロップが多くの場合使用される。

いま、回路がある状態ｑｉであった時、入力 xk によって、状態が qi に変化すると同時に、yl を出力するような順序回路があったとするこの回路の一連の動作を、図 2.5.1(a)のように表す。また、xk'が入力されとき、yl'を出力するが状態は変化しないような場合は、図 2.12(b)に示すように qi から qi へのループで表す。図 2.5.1 のように状態の変化を表現した図を状態遷移図という。〔1〕



(a)



(b)

図 2.12 状態遷移図

たとえば 4 進カウンタの状態遷移図は、図 2.14 のようになる。入力が 1 とき、次の状態に移り、 1 が 4 回入力されるごとに出力を 1 にして、元の状態に復帰し、また、入力が 0 の場合は状態が変化しない様子を表している。

一般に順序回路が与えられれば、状態遷移図を書くことができる。このとき、順序回路の状態とは、一般に記憶回路または遅延回路の出力の状態をいう。図 2.13 の 4 進カウンタを 2 個のフリップフロップを用いて実現した場合は、その状態 q1〜q4 は、2 個のフリップフロップの出力値の組合せで表され、00,01,10,11 の 4 状態に対応している。

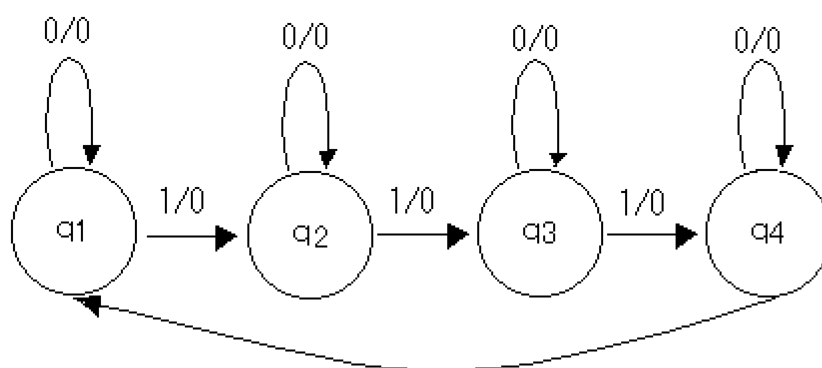


図 2.13　4 進カウンタの状態遷移図

2.5.2 状態遷移表

順序回路の解析は、状態遷移図を求めて、回路の動作を知ることである。状態遷移図は、直感的に回路の動作が理解できる利点があるが、与えられた回路より直接これを導くことは、多くの場合困難である。順序回路の解析は、状態遷移図を表にした状態遷移表を求めるところから始められる。

表 2.2 は、図 2.13 の 4 進カウンタの状態変化を表にした状態遷移表である。現在の状態から、入力 x により移るべき次状態が示されると同時に、出力値も合わせて示されている。出力値を表す側の表を、特に出力表と呼ぶことがある。

〔1〕

表 2.2　状態遷移表

現状態 q	次状態 q' 入力 x 0	次状態 q' 入力 x 1	出力 y 入力 x 0	出力 y 入力 x 1
q1	q1	q2	0	0
q2	q2	q3	0	0
q3	q3	q4	0	0
q4	q4	q1	0	1

2.6　デコーダ・エンコーダ

2.6.1 デコーダ

デコード(decode)とは、入力の組合せ(2 進数パターン、コード)にしたがって対応する単一出力を生成することであり、コードの解読(復号)を行う機能である。デコード機能をもつ組合せ回路をデコーダ(decoder)という。図 2.14 に示すように、n 入力と 2^n 出力をもつデコーダを " n*m デコーダ " という。デコーダの出力は相互に排他的であり、入力パターンに対応する出力だけ 1 とし、残りすべては 0 となる。〔1〕

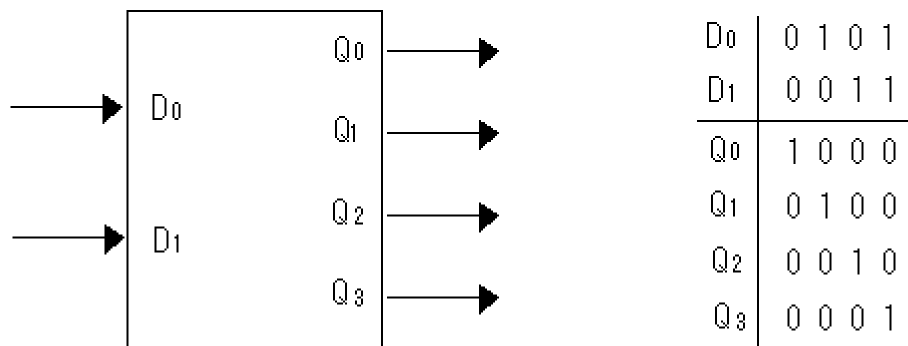


図 2.14　2*4 デコーダ

2.6.2 エンコーダ

エンコード(encode)とは、デコーダの反対の機能(すなわち符号化機能)であり、エンコーダ(encoder)はエンコードを行う組合せ回路である。エンコーダでは、複数入力のうち一つだけが 1 となり､1 になった入力に対応する出力パターン(コード)を生成する。　図 2.15 に示すように、2^n 入力と出力をもつエンコーダを“ m*n エンコーダ ”という。

単純なエンコーダでは、複数入力が 1 となると出力パターンから一意に入力を識別できない。例えば図 2.15 において、D1＝D2＝１、D0=D3=0 の場合、Q=11 となり、入力が D3 だけであると誤認してしまう。そこで、入力にプライオリティ(priority)という優先順位づけをしてこの識別を可能にしたエンコーダをプライオリティエンコーダという。プライオリティエンコーダでは、優先順位の最も高い入力に対応する出力コードを生成する。上記の例では、D0→D3 とプライオリティが高くなるというようあらかじめ設定しておくと、“ D1=D2=1 の場合でも Q=10 で優先順位の高い入力が D2 である ”と識別可能となる。

プライオリティエンコーダの応用としては、割り込み検出回路などがある。

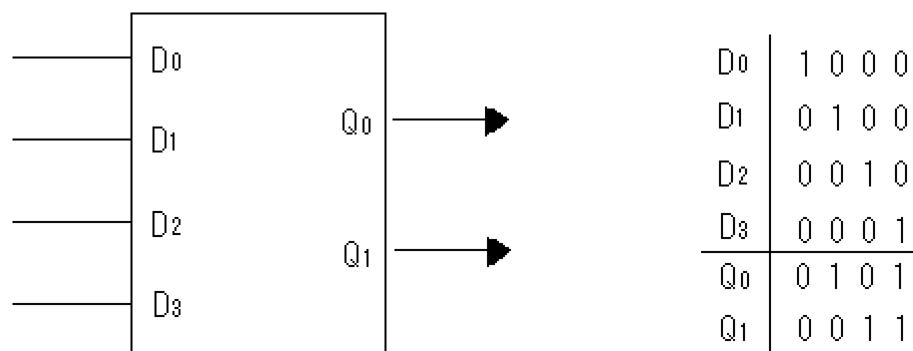


図 2.15　4*2 エンコーダ

第３章　標準ロジック IC による回路設計

3.1　標準ロジック IC

バイポーラトランジスタを用いた TTL(transistor　transistor　logic)IC は、集積度はあまり高くないが、比較的高速に動作する。このため TTL は、組み合わせてさまざまな回路を構成する標準ロジック（ランダムロジック）IC として用いられていられる。TTL IC には、速度の向上や消費電力の低下を求めて改良したさまざまな種類があり、次に述べる CMOS IC とともに多く用いられている。

バイポーラトランジスタを用いた TTL 以外のディジタル IC としては、ECL（emitter　coupled　logic）がある。ECL は内部で用いるトランジスタが非飽和で動作するとめ、TTL に比べて高速動作が可能である。高速動作を必要とする特殊な用途に向けて、各種論理の IC が用意されている。これに対して、ユニポーラトランジスタ（unipolar　transistor）を用いた MOS IC は、集積度が高く消費電力が小さいため、TTL と同様に標準ロジックとして用いられている。当初 TTL と比較して低速であった MOS 標準ロジックも、半導体製造技術の進歩によって、高速な IC が開発されるようになった。現在用いられている MOS 系の標準ロジック IC は、CMOS で構成されている。CMOS は P チャネルの MOSFET と､N チャネルの MOSFET を相補的に用いたものである｡現在では、ディジタル回路の設計には CMOS ロジックが多く用いられている。〔2〕

3.2　TTL ファミリ

標準ロジック IC を図 3．1 に示す。標準ロジック IC はその機能や回路構成によって、このように多くの種類がある。これらは、その構成回路別に XX シリーズと呼ばれる。

例えば TTL IC で、最初に開発されたスタンダード TTL は、74 シリーズと呼ばれる。シリーズと呼ばれるのは、各種機能を持った一連の IC が用意されていることからきている。

74 シリーズの IC は、その論理機能によって一連の番号が付けられている。例えば 1 個の IC 内に、2 入力の NAND が 4 個組み込まれたものは 7400、インバータ（NOT）が 6 個組み込まれたものは 7404、という番号が付けられている。この TTL の型名は、テキサスインスツルメント社によって名付けられたものが、標準

として用いられている。

74 シリーズには NAND、NOR などのゲート回路から、デコーダなどの組み合わせ回路、各種フリップフロップ、カウンタ、マイクロプロセッサのインターフェース用の IC まで、多くの種類が用意されている。現在設計されるディジタル回路で実際にスタンダードの 74 シリーズが用いられることは少ないがこの機能ごとに付けられた番号は、TTL の他のシリーズだけでなく、74 系の CMOS 標準ロジック IC にまで共通に用いられている。

TTL の 74 系シリーズでは、回路構成や動作速度などが異なったシリーズが開発されてきたが、回路動作のために当然電源が必要となる。TTL の 74 系シリーズでは、電源電圧は５V 一定になっている。

TTL には、74 シリーズの他にも速度や消費電力などの性能により多くのシリーズがある。表 3.1 に TTL ファミリのタイプと特徴を示す。ショートキートランジスタを用いてさらに低電力化した 74LS シリーズ、74LS シリーズをさらに高速化したアドバンスドショットキー74ALS シリーズ、さらに高速化した 74F シリーズなどがある。〔2〕

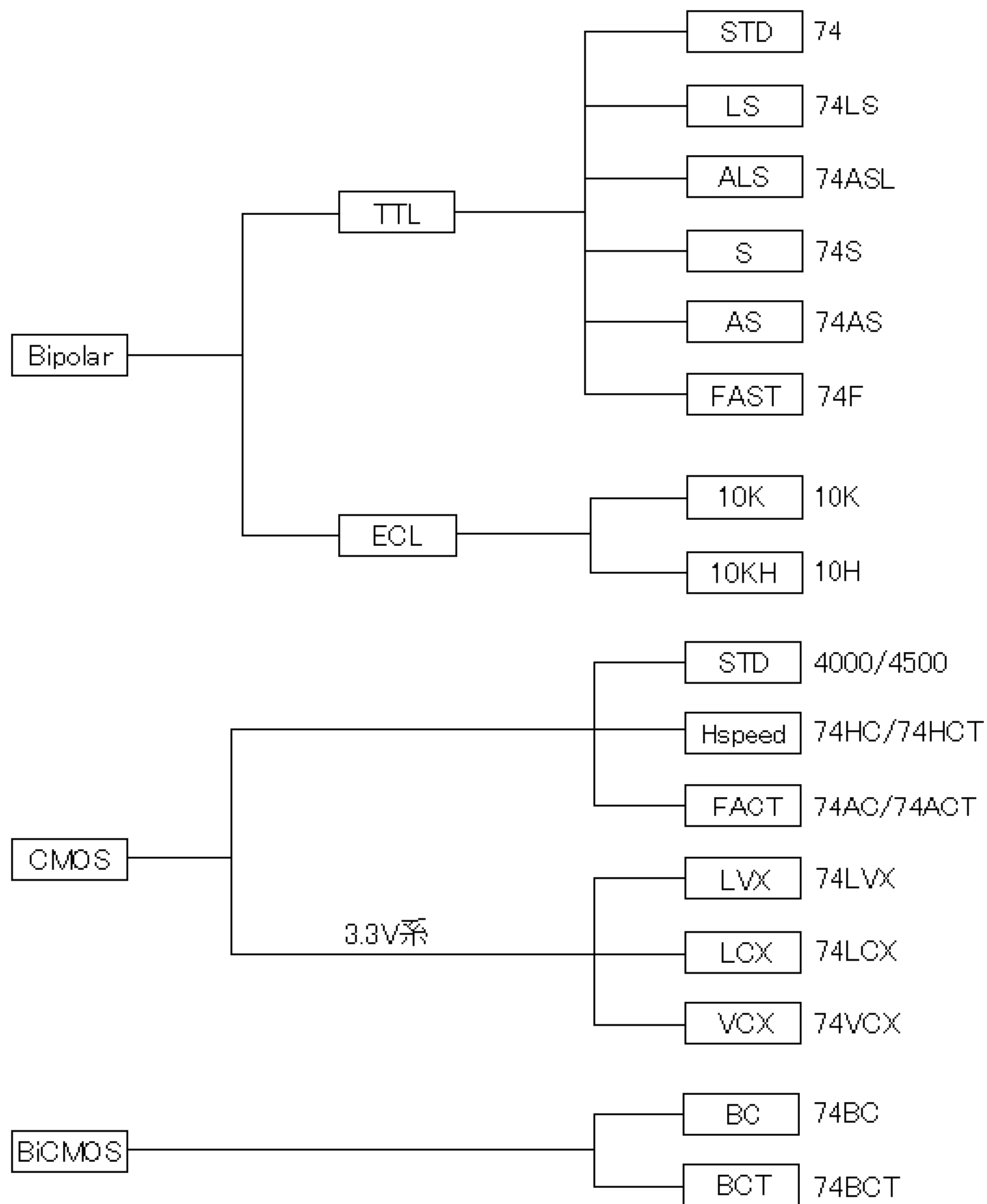


図 3.1　標準ロジック IC

表3．1　TTLファミリと特徴

タイプ	型番	特徴
スタンダード	74	最初に開発されたTTL.
ショットキー	74S	ショットキートランジスタを用いて高速化したもの.
アドバンスド・ショットキー	74AS	74LSタイプを低消費電力化したもの.
ローパワー・ショットキー	74LS	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、TTLとして標準的に用いられている.
アドバンスド・ローパワー・ショットキー	74ALS	AS、ALSタイプをさらに低消費電力化したもの.
FAST	74F	LSタイプに比べ低消費電力で高速化したもの.

3.3 CMOSファミリ

MOS系の標準ロジックとしては、CMOSタイプが多く利用されている。CMOSファミリの特徴を表3．2に示す。CMOSの標準ロジックICも各社が製造しているが､最初に開発されたのはRCA社/モトローラ社の4000／4500シリーズである。電源電圧は3～18Vの範囲で動作し、単純なNORゲートから、コンピュータのバスの構成に必要なロジックまで幅広く用意されている。

この他に速度を改良して、TTLと機能ごとに同番号を付けた74HCシリーズがある。74HCシリーズは、TTLも含めた標準ロジックIC全体の中で、現在最も多く利用されている。これは、低消費電力であることと、TTLに匹敵するような動作速度を得られるようになったためである。

74系CMOS ICで使用できる電源電圧は、74HCシリーズでは2～6V、74ACシリーズでは2～5.5Vの範囲である。しかし、TTLの電源電圧と合わせて5Vを用いることが多くなっている。

この74HCシリーズはTTLと番号を合わせているが、TTLと実際に接続するためには、入出力の電圧レベルを合わせる必要がある。74HCタイプの入出力にバッファを付けた74HCTシリーズは、TTLと電圧・電流レベルを合わせてTTLとの入出力インターフェースを可能にしたものである。この74HCTタイプはTTLと混在して使用できるため、TTLとCMOSのインターフェースなどの用途で、今後も使われるものと思われる。

より高機能なディジタル回路を設計するために、より高速で消費電力の小さ

なロジック IC の開発が求められてきた。また IC の高集積化からも電源電圧を下げる必要がでてきた。このため CMOS では、さらに電源電圧を 3.3V とした 74LCX シリーズなどが開発された。〔2〕

表 3.2　CMOS ファミリと特徴

タイプ	型番	特徴
スタンダード	4000/4500	最初に開発されたCMOS IC、74シリーズとは異なった特徴ある論理があり、低速だが電源電圧範囲が広い.
ハイスピード	74HC	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としての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.
FACT	74AC	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.

上節まではロジック IC の種類や特長を述べている。次節からは、基板上に実装する回路に用いられている各部分の説明を述べる。

3.4　クロック分周回路

クロック分周回路は水晶発振器から出る 4,194,304Hz を分周し、1 Hz（＝1 秒）と 2 Hz を得る回路である。4,194,304Hz は 2^{22} なので $1/2^{22}$ に分周すれば 1 Hz が得られる。したがって、IC の SN74LS292 を用いて D－FF を 22 段構成とすればよい。

3.5　水晶発振器

水晶はその共振特性が極めて安定しているため、特性の優れた発振回路を実現するためには不可欠な素子となっている。水晶発振器とは、水晶から切り取った小さい水晶の結晶である水晶振動子を使って発振回路を作り、発振させるものである。水晶振動子の両面に、圧力を加えてやると＋と－の電荷が発生する。このことを利用して、水晶振動子の両面に電極をつけ、交流電圧を加えると、圧力が水晶振動子の両面に交互に加わり水晶が伸びたり縮んだりする。これを共振現象と呼ぶ。そして、このような水晶振動子を回路に組み込むことに

より、回路の中で共振し、発振する。水晶振動子を用いた回路の発振は非常に安定で正確であるために、時計などに用いる場合は最適の振動子である。〔5〕

3.6　60 進カウンタ回路

60 進カウンタ回路は時計用ブロック回の“ 秒 ”と“ 分 ”およびアラーム用ブロックの“ 分 ”に使われる回路で図 3.2 に示すように、通常のクロックパルスと早送り調節用のパルスを切り換える時間調節回路を含む。

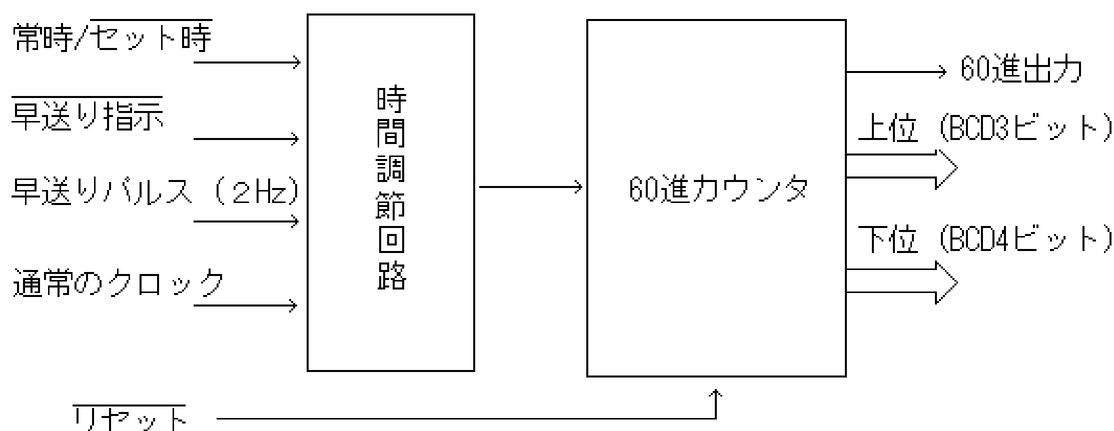


図 3.2　60 進カウンタ回路のブロック図

3.7　7 セグメント

７セグメント復号器は､図 3.4 に示すような電卓やディジタル時計などに用いられる７セグメント LED（light emitting diode）を点滅させるための回路で、４ビットの２進符号の入力に対して、図 3.3 に示す７セグメント内の適切なセグメントを点灯させるための組み合わせ回路が作りこまれている。表 3.3 は、１０進数の０～９を表示させるための、４ビットの入力と点灯させるセグメントの位置を示した真理値表である。ただし、１０進数の１０～１５までに対応する入力については、すべてのセグメントを点灯するものとした。以下、この真理値表に基づいて、目的とする７セグメント復号器の公正について考えてみる。

この表により、各セグメントに対する論理関数を求めるのであるが、表をみると、１の数より０の数のほうが非常に少ないことがわかる。つまり、出力値

が０（アクティブ－L）となる変数の組み合わせを用いた論理関数を求めたほうが簡単である。

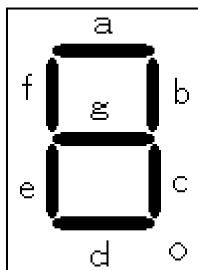


図 3.3　７セグメント LED

表 3.3　７セグメント LED の真理値表

表示	入力				出力						
	A	B	C	D	a	b	c	d	e	f	g
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	0	1	1
6	0	1	1	0	1	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	1	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	1	0	1	1
8	1	0	1	0	1	1	1	1	1	1	1
8	1	0	1	1	1	1	1	1	1	1	1
8	1	1	0	0	1	1	1	1	1	1	1
8	1	1	0	1	1	1	1	1	1	1	1
8	1	1	1	0	1	1	1	1	1	1	1
8	1	1	1	1	1	1	1	1	1	1	1

図 3.3 のように 7 個の発光ダイオードが 8 の字に配置されている。それぞれ a ～ g までのセグメントとして記号が付けられており、ここに電流を流すことで発光表示する。LED 数字表示素子は、図 3.4(a)のようにダイオードのアノードを共通にしたアノードコモン形と､図 3.4(b)のようにカソードを共通にしたカソードコモン系がある。電源や前段に配置するデコーダの種類によって、この２つを使い分けることができる。

ダイオードにはアノードとカソードの２つの端子があり、電流は、アノード

からカソード（順方向）には流れるが、逆方向には流れないようになっている。

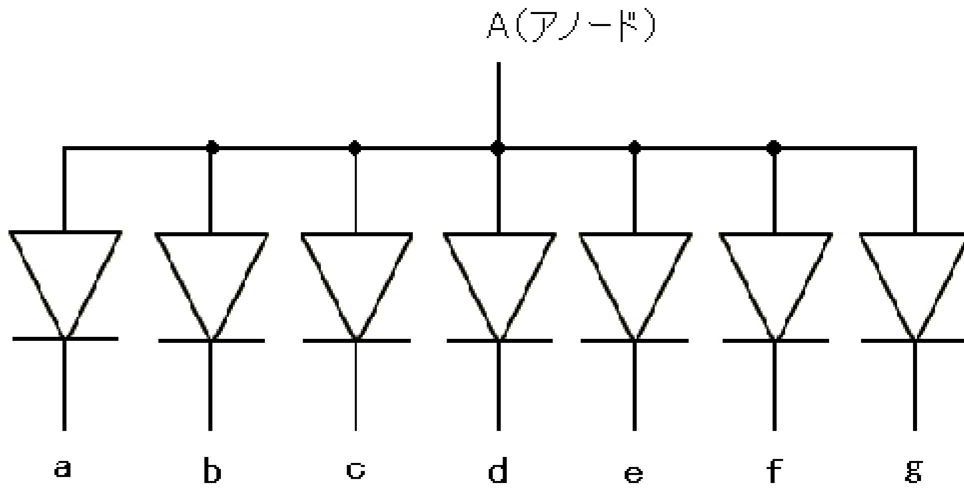
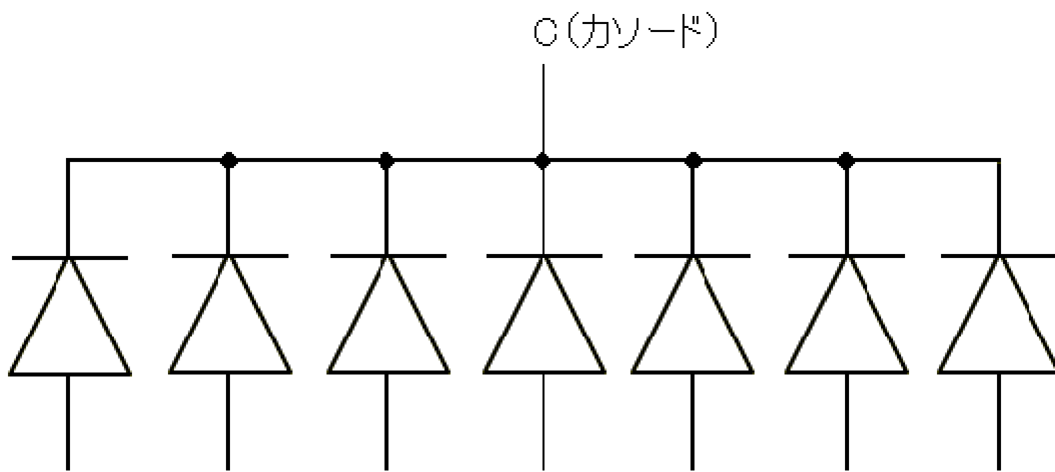


図 3.4　(a)アノードコモン形



(b)　カソードコモン形

3.8　チャタリング防止回路

この回路設計で用いるのは機械接点のスイッチであるから、そのままの信号を用いると、チャタリングで誤動作をする恐れがある。チャタリングによる不規則なレベルの変化によって回路が動作することになるため、カウンタやタイミング回路などのトリガパルスとして使用する場合はIC化に限らず、このようなチャタリングを防止する回路が必要である。

最も確実で一般的なチャタリング防止回路を図3.5に示す。スイッチが図のように接点ｂ側に倒れている状態ではRS・FFの入力はS=0、R=1であるため出力はQ=1,$\overline{\mathrm{Q}}$＝０になっている。この状態から接点ｂが離れる瞬間は接点部分で放電が起こりチャタリングが発生するが、入力ＳとＲにはＳ＝０，Ｒ＝１(セット状態)またはＳ＝１，Ｒ＝１（ホールド状態）のレベルが与えられるため、出力はスイッチが動作する前のQ=1,$\overline{\mathrm{Q}}$＝０の状態が続く。そして、接点ａに達するとチャタリングにより入力Ｒには“１”または“０”が不規則に与えられるがＳは“１”のままである。したがって、接点ａに達した以後は最初のS=１、R=0でリセット状態となるためQ=0が続く。このようにして出力にはチャタリングが現れずチャタリングが防止される。

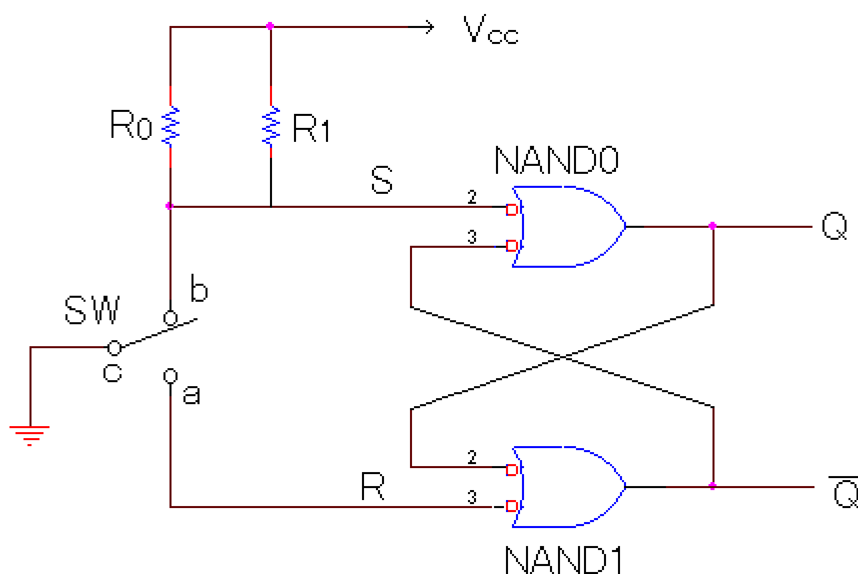


図3.5　NANDゲートを用いたチャタリング防止回路構成

3.9　ディジタル時計の設計

上記の各節の内容を組み込みながら、基板上にディジタル時計の回路設計を実装することにした。今回作成したディジタル時計は時間を用いず、分、秒のみの回路設計とした。分、秒ともに 60 進カウンタを使って 0 から 59 までを表示できるようにし、うまく桁上げができるかどうかを確認した。表示器は 7 セグメントを用い表示させるために 74HC246 のデコーダを使った。クロックには、最適と思われる水晶発振器を用い 1 Hz まで分周したものを使った。設計したディジタル時計の IC 配置図を図 3.6 に示す。

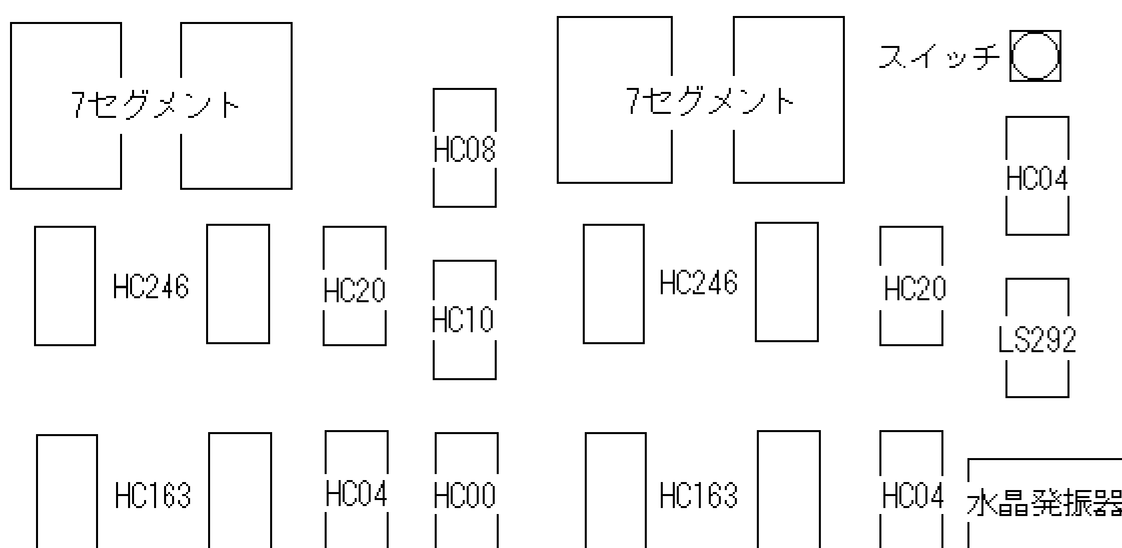
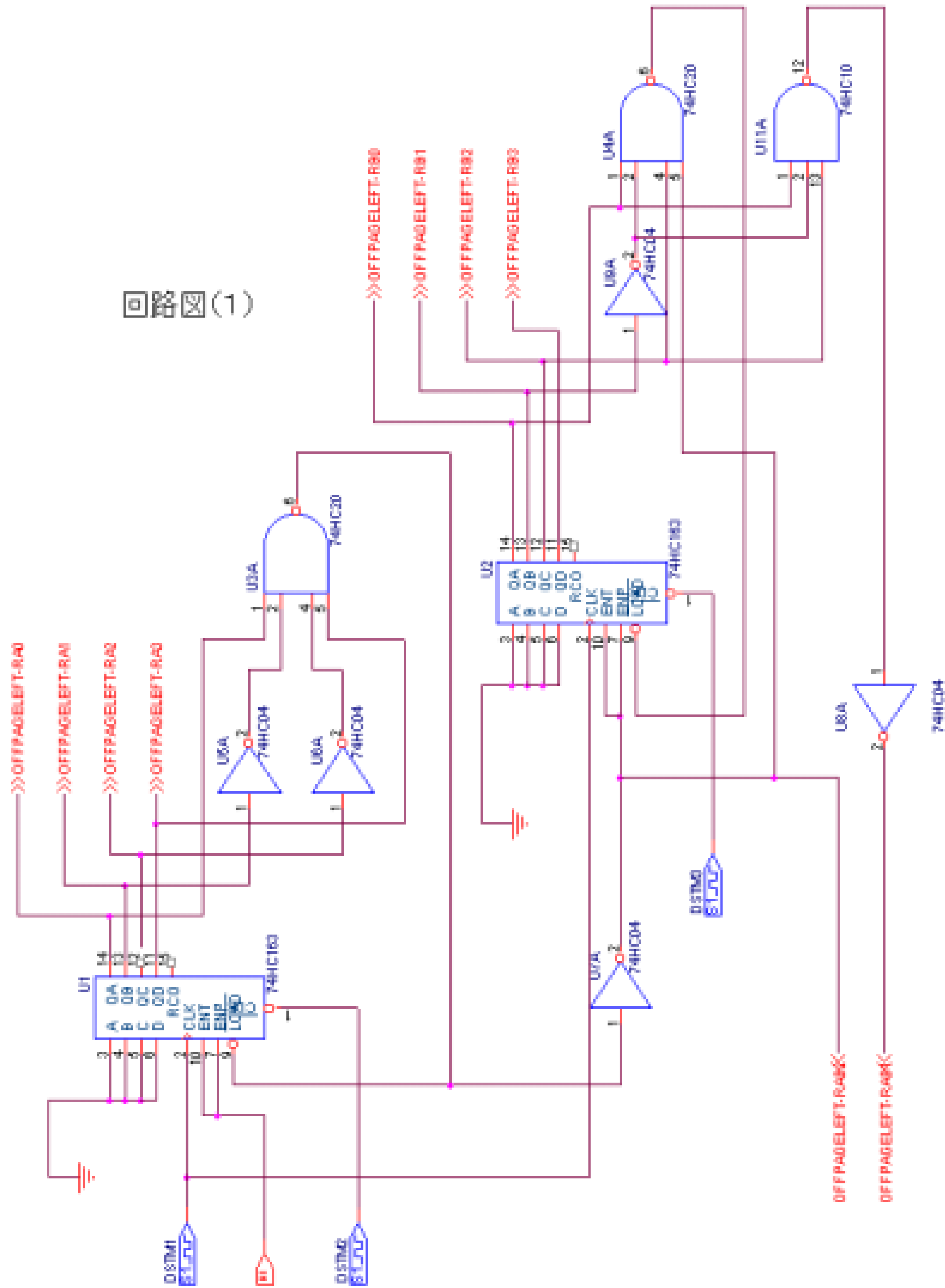
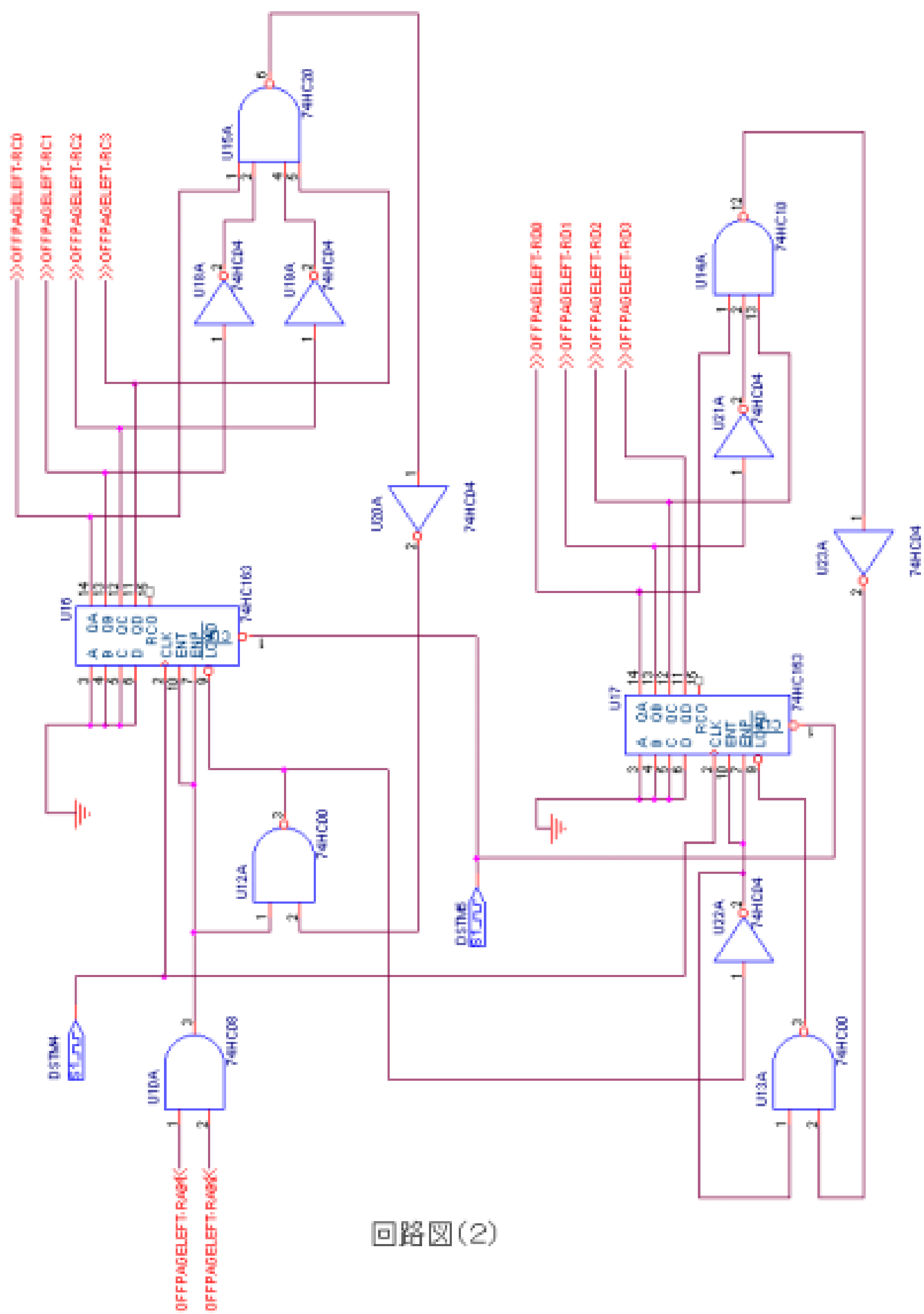


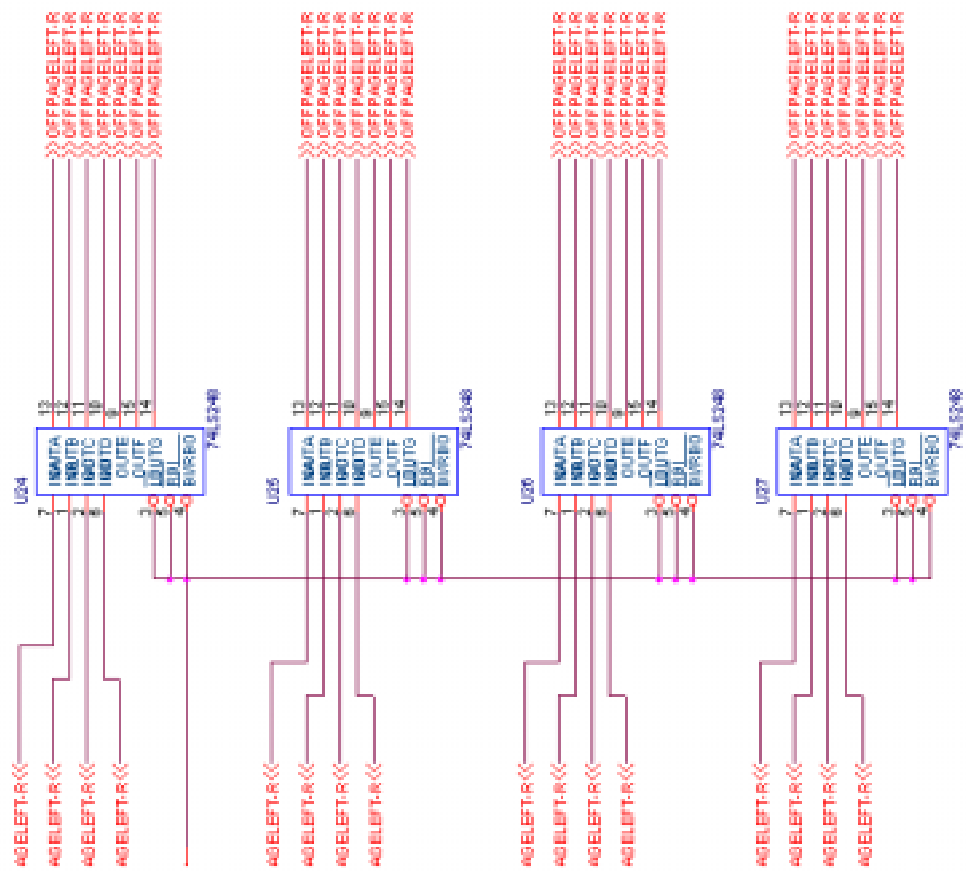
図 3.6　ディジタル時計回路 IC 配置図

3.10　ディジタル時計の回路図と回路シミュレーション

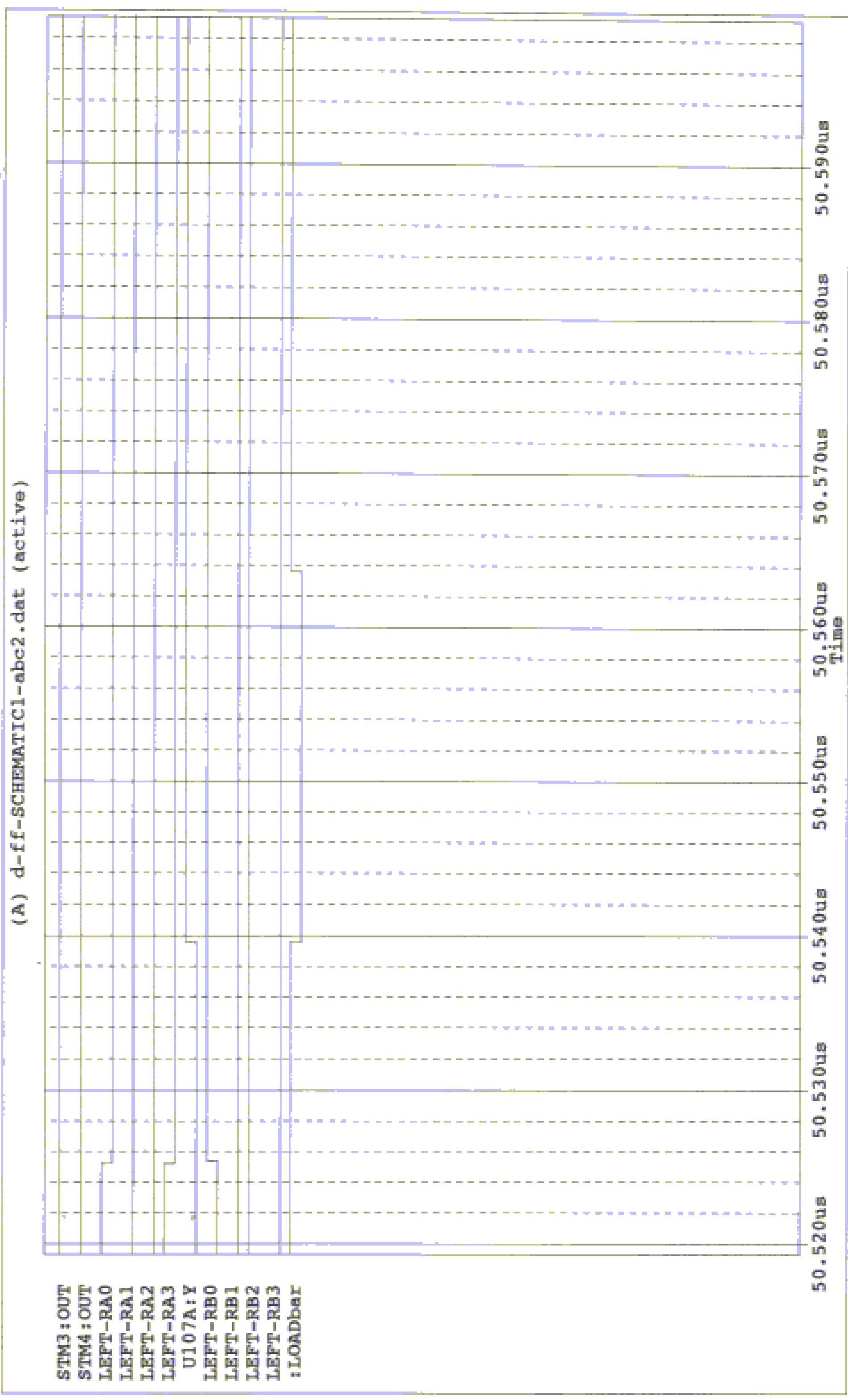




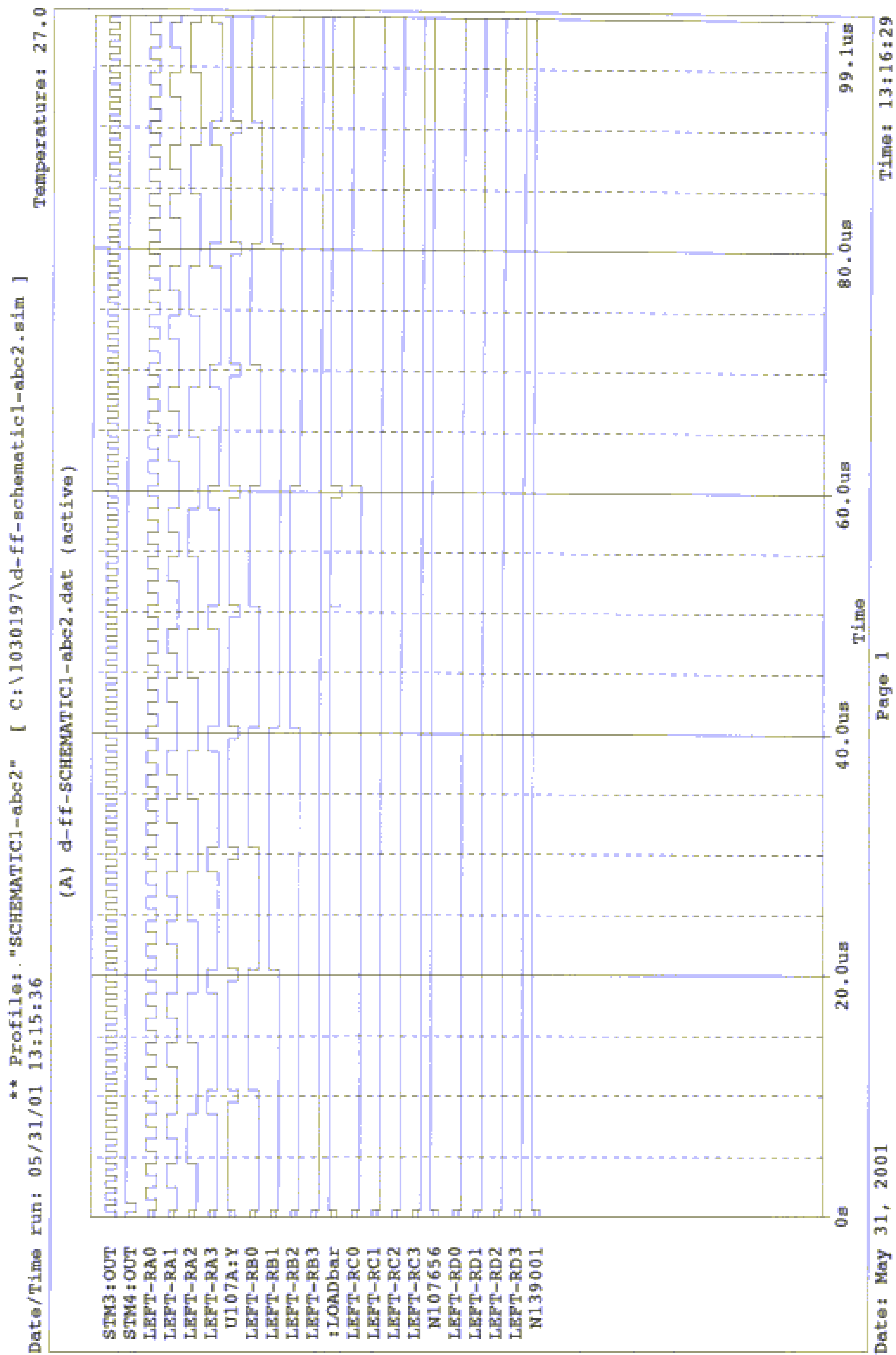
回路図(2)



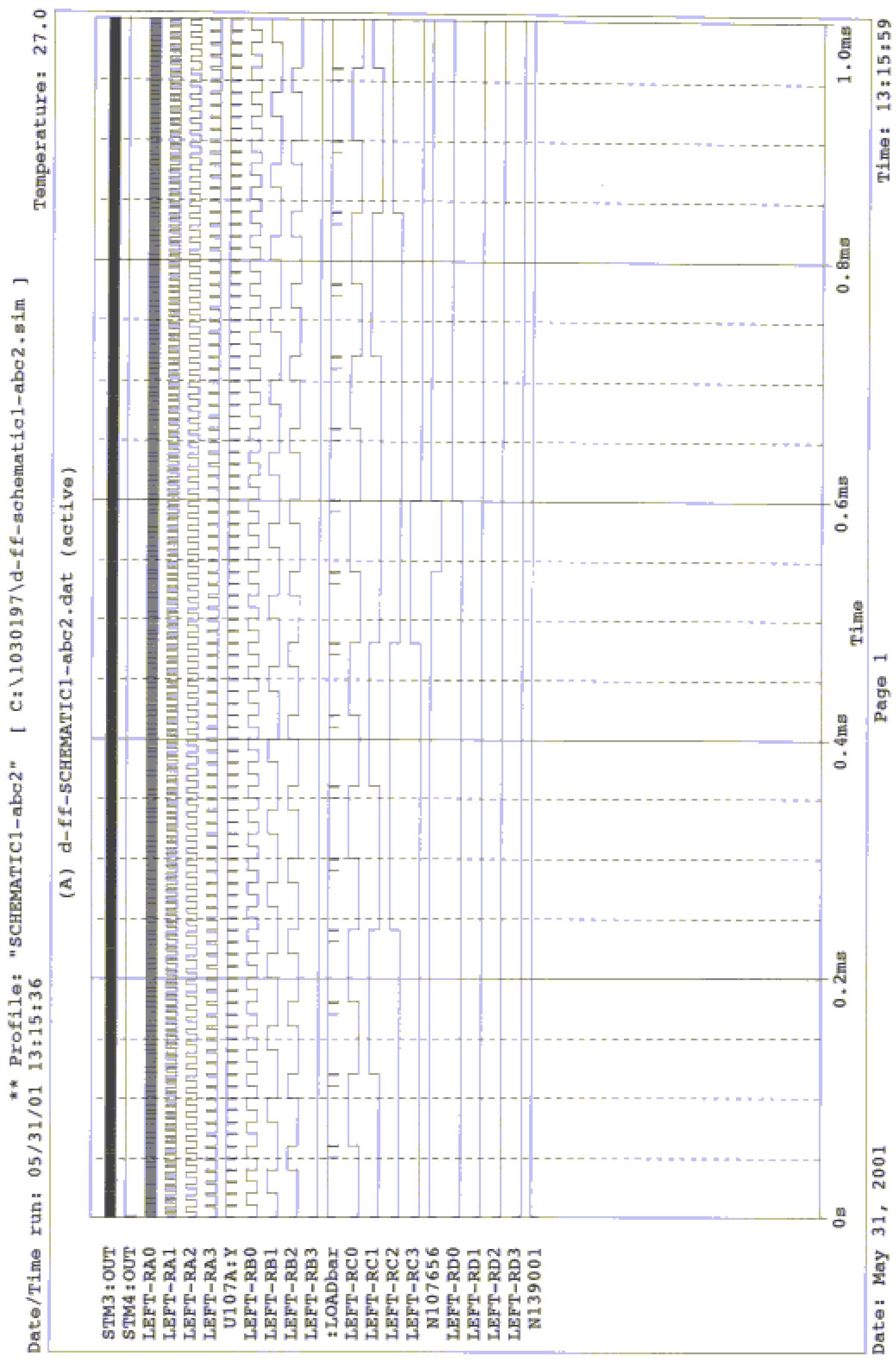
回路図（3）



シミュレーション図（1）



シミュレーション図（2）



シミュレーション図（3）

実装したディジタル時計の写真を図 3.7 に示す

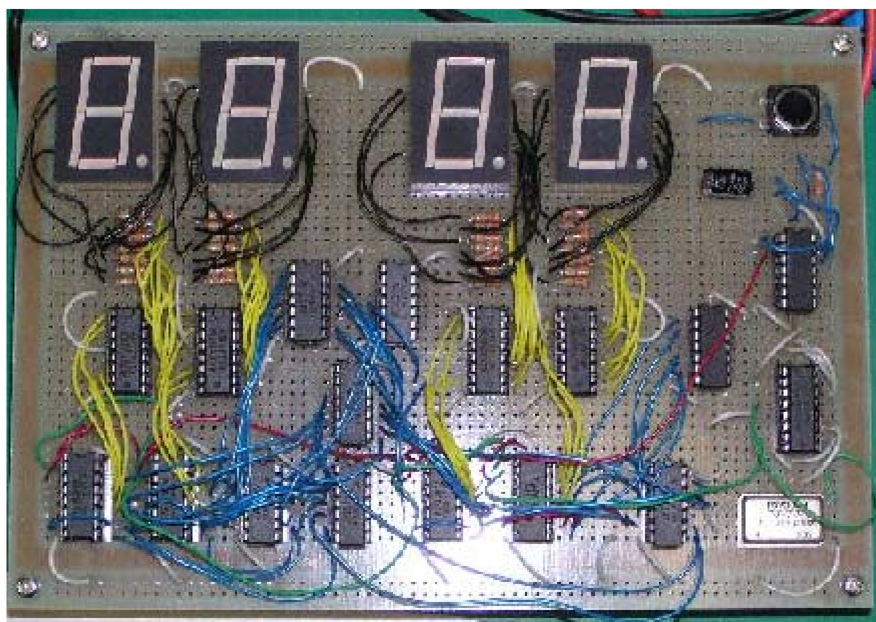


図 3.7　ディジタル時計の実装写真

第４章　VHDL による時計の設計

４.１　VHDL とは？

VHDL は“ Very　High Speed　Integrated　Circuit（VHSIC）　Hardware Description　Language ”の略で、もともとは“ HDL ”（ハードウェア記述言語）の１種です。

さまざまな記述方法ができるようになっており、一般的には下記 3 種類の表現方法があるといわれています。

(1)Behavior（振る舞い）記述

機能レベルでの振る舞いや動作内容だけを記述するレベルで、抽象的な機能の検証に使う。しかし、直接論理合成が出来ないので、ASIC や FPGA の設計に使われる。

(2)RTL 記述（Register　Transfer　Level）

直接論理合成可能な、クロックをベースにしたレジスタや、組み合わせ回路をまとまった論理式で記述するレベルで、通常の ASIC や FPGA の設計に使われます。

(3)ゲートレベル記述

ネットリストと呼ばれる、ASIC 内部のゲートやセルベースの接続関係で表現したリストで記述するレベルである。RTL では表現が出来ない高速性や高密度を要求する場合になどに使う。〔4〕

４.２　VHDL の歴史

VHDL は、米国国防省の VHSIC（Very　High　Speed　Integrated　Circuit）委員会で 1981 年に提唱された。大規模 IC の開発には、より上位レベルでの検証が求められていた。また当時、国防省向け ASIC の開発は長いもので 3 年から 4 年もかかっていた。その間、半導体のプロセスは進歩し、開発当初の時点では一番スピードが速い ASIC を使用していたのが、開発が完了する時点では時代遅れになってしまうという問題が生じていた。そこで直接ロジック・ゲートを回路図で入力するのではなく、ハードウェア記述言語（HDL）で設計することによって、開発終了時に一番スピードの速い ASIC を選択できるようにす

る必要があった。

こうして、1983 年に VHDL の仕様作成が始まり、1985 年に作業が完了した。1986 年にはマニュアルにまとめられ、バージョン 7.2 として公開された。現在では、米国国防省が調達するすべての ASIC は、VHDL 記述付きで納入するように義務づけられている。

その後、1986 年には IEEE（米国電機電子技術者協会）での標準化作業が、VASG（VHDL Analysis & Standardization Group）委員会で始まりまった。1987 年 5 月には LRM（言語仕様書 Language Reference Manual）が作成され、12 月に IEEE Std 1076－1987 として承認されている。

IEEE は、米国での技術者の集まりという位置づけですが、ここで承認されたものが世界の標準として認められる権威ある団体で、VHDL も全世界の標準 HDL として広く普及している。

1989 年には、VHDL シミュレータや VHDL 記述からロジック回路を生成するソフトウェア（論理合成ツール）が EDA ベンダから販売されるようになり、実際にロジック回路設計に用いられるようになった。〔4〕

4.3 HDL 設計のメリット

HDL（ハードウェア記述言語）による設計手法は、すでに ASIC などの大規模集積回路の設計で盛んに利用されている。HDL による設計は ASIC に限らず、FPGA（Filed Programmable Gate Array）や PLD（Programmable Logic Device）などを使用した比較的小規模な設計にもさまざまなメリットをもたらす。表 4.1 に回路図入力による設計と HDL による設計の比較を示す。

HDL による設計は、より抽象度の高いレベルで設計することにより、難しい論理式から設計者を解放し、設計期間を短縮することができる。また、抽象度の高い記述であるということは、それだけ設計の変更が容易になるということで、設計者はより完成度の高いシステムを構築することができる。

ハードウェア記述言語には VHDL（VHSIC HDL）、Verilog－HDL、UDL/I（Unified Design Language for Integrate Circuit）、SFL（Structured Function description Language）などがある。それぞれの特徴を表 4.2 に示す。〔4〕

表 4.1 回路図入力と HDL 入力による設計

	回路図入力	HDL 設計	HDL 設計での利点
1	回路図入力に時間がかかる	テキストで簡単に入力	設計期間の短縮 1/2～1/3
2	論理式(ブール代数)を考える必要がある	論理式を考える必要がない	
3	回路変更が大変	回路変更が容易	より完成度の高いシステムの構築
4	設計者以外では、内容を理解しづらい	誰にでも内容を理解しやすい	設計の再利用が容易
5	特定の半導体メーカのライブラリを使用して回路図入力をする	半導体メーカのライブラリを使用しない。どのメーカでも作成可能	

表 4.2 各種 HDL の比較

言語名	開発元	特徴
VHDL	米国国防省が中心となって開発	幅広い分野の記述が可能。高い記述能力
Verilog-HDL	シミュレータ Verilog の言語として開発	幅広い分野の記述が可能だが VHDL ほど記述能力は高くない
SFL	PARTHENON システムの言語として開発	RTL での記述のみ可能。完全同期式の回路の限定している。単純で分かりやすい記述
UDL/I	日本電子工業振興協会において開発	RTL での記述のみ可能。同期式の回路記述は単純化されている

4.4　VHDLによるディジタル時計の設計

VHDLによるディジタル時計の設計として、以下の7セグメント表示部、時間、分、秒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。

4.4.1　7セグメント表示器

時計の出力を表示するために、7セグメント表示器（4-7デコーダ）を用いる。4ビットのデータから7ビットのデータへの変換を行う動作を記述している。表示する値は0～9までなので、残りの10～15はothersでまとめ、出力は不定を示す“XXXXXXX”を記述している。7セグメント表示器のVHDL記述とテストベンチを＜リスト4.1＞に、シミュレーション結果を図4.1に示す。

＜リスト4.1＞7セグメント表示器のVHDL記述

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

entity DECODER4TO7 is
    port (
        A,B,C,D: in std_logic;
        Y: out std_logic_vector(6 downto 0)
    );
end DECODER4TO7;

architecture RTL of DECODER4TO7 is

signal INDATA : std_logic_vector(3 downto 0);

begin

    INDATA <= D&C&B&A;

    process(INDATA)
    begin

    case INDATA is
```

```
        when "0000" => Y <= "0111111";   --0
        when "0001" => Y <= "0000110";   --1
        when "0010" => Y <= "1011011";   --2
        when "0011" => Y <= "1001111";   --3
        when "0100" => Y <= "1100110";   --4
        when "0101" => Y <= "1101101";   --5
        when "0110" => Y <= "1111101";   --6
        when "0111" => Y <= "0100111";   --7
        when "1000" => Y <= "1111111";   --8
        when "1001" => Y <= "1101111";   --9
        when others => Y <= "XXXXXXX";   --X

    end case;

    end process;
end RTL;
```

＜リスト 4.1＞7 セグメント表示器のテストベンチ記述

```
library ieee;
use ieee.std_logic_1164.all;
use std.textio.all;
use work.DECODER4TO7;

entity TESTBNCH is
end TESTBNCH;

architecture stimulus of TESTBNCH is
component DECODER4TO7 is
    port (
        A,B,C,D: in std_logic;
        Y: out std_logic
    );

end component;
```

```
signal A,B,C,D: std_logic;
signal Y: std_logic_vector(6 downto 0);

begin
   DUT: DECODER4TO7 port map (
          A,B,C,D,
          Y
        );

 STIMULUS1: process
   begin
    A <= '0'; wait for 20ns;
    A <= '1'; wait for 20ns;
 end process STIMULUS1;

 STIMULUS2: process
   begin
    B <= '0'; wait for 40ns;
    B <= '1'; wait for 40ns;
    B <= '0'; wait for 40ns;
    B <= '1'; wait for 40ns;
    B <= '0'; wait for 40ns;
 end process STIMULUS2;

 STIMULUS3: process
   begin
    C <= '0'; wait for 80ns;
    C <= '1'; wait for 80ns;
    C <= '0'; wait for 40ns;
 end process STIMULUS3;

 STIMULUS4: process
   begin
    D <= '0'; wait for 160ns;
    D <= '1'; wait for 40ns;
 end process STIMULUS4;
```

```
end stimulus;
```

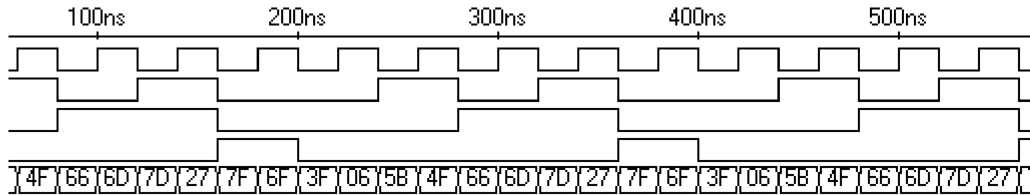


図 4.1　7 セグメント表示器シミュレーション結果

4.4.2　60 進カウンタ（分,秒）

分（MIN）と秒（SEC）は同じ動作をするので、プログラム内の名前のみを変更し記述は同じものとした。

このカウンタでは“ BCD１ ”が１の桁を、“ BCD10 ”が 10 の桁を表示する。“ BCD10 ”は 60 進カウンタの 10 の桁なので 0～5 までの値しか持たない。したがって 3 個のフリップフロップで足りる。また、この二つのカウンタは“ BCD１WR ”と“ BCD10WR ”によって値を書き込むことが可能である。この機能によって時刻合わせを行う。

①では“ BCD１WR ”が’１’の時、フリップフロップの強制リセットとセット端子に“ DATAIN ”の値が代入されるようにしている。図４.２のシミュレーション波形では、6ns の時刻に“ BCD１WR ”に’1’が入力され、“ DATAIN ”の“ BCD１ ”に代入されている。“ DATAIN ”は“ BCD１０ ”への書き込みと共有されている。②では“ BCD１0WR ”が’1’の時、“ DATAIN ”の値を“ BCD１０ ”に書き込んでいる。“ CIN ”は下位カウンタからの桁上がり信号で、この値が’１’でなければ“ BCD１ ”および“ BCD１０ ”のどちらも動作しない。“ CO ”は“ CIN ”とは逆に上位カウンタへの桁上がりである。③では、このカウンタの値が 59 で“ CIN ”信号が’1’の場合のみ、上位カウンタに’1’を出力するように記述する。

秒の VHDL 記述とテストベンチを＜リスト 4.2＞に分の VHDL 記述とテストベンチを＜リスト 4.3＞に示す。図４.２に秒のシミュレーション結果を示す。

＜リスト 4.2＞60 進カウンタ（SEC）の VHDL 記述

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

```
entity SEC is
   port (
       CLK,BCD1WR,BCD10WR,CIN: in std_logic;
       CO: out std_logic;
       DATAIN   : in  std_logic_vector(3 downto 0);
       sec1     : out std_logic_vector(3 downto 0);
       sec10    : out std_logic_vector(2 downto 0)
   );
end SEC;

architecture RTL of SEC is
signal sec1N  :  std_logic_vector(3 downto 0);
signal sec10N :  std_logic_vector(2 downto 0);
begin
sec1 <= sec1N; sec10 <= sec10N;
   process(CLK,BCD1WR)
   begin
       if(BCD1WR='1') then    --①
          sec1N <= DATAIN;
       elsif(CLK'event and CLK='1') then
          if(CIN='1') then
            if(sec1N=9) then
             sec1N <= "0000";
            else
             sec1N <= sec1N + 1;
            end if;
          end if;
       end if;
   end process;
   process(CLK,BCD10WR)
   begin
       if(BCD10WR='1') then    --②
          sec10N <= DATAIN(2 downto 0);
       elsif(CLK'event and CLK='1') then
            if(CIN='1' and sec1N=9 ) then
             if(sec10N=5) then
```

```
            sec10N <= "000";

          else
            sec10N <= sec10N + 1;
          end if;
        end if;
      end if;
    end process;

process(sec10N,sec1N,CIN)
begin
        if( CIN='1' and sec1N=9 and sec10N=5 ) then
            CO <= '1';
        else
            CO <= '0';
        end if;

end process;
end RTL;
```

＜リスト 4.2＞60 進カウンタ（SEC）のテストベンチ

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

entity TESTBNCH is
end TESTBNCH;

architecture stimulus of TESTBNCH is
component SEC is
    port (
        CLK,BCD1WR,BCD10WR,CIN   : in std_logic;
        CO      : out std_logic;
        DATAIN  : in  std_logic_vector(3 downto 0);
```

```
        sec1    : out std_logic_vector(3 downto 0);
        sec10   : out std_logic_vector(2 downto 0)
    );
end component;

signal CLK,BCD1WR,BCD10WR,CIN   : std_logic;
signal CO       : std_logic;
signal DATAIN  : std_logic_vector(3 downto 0);
signal sec1    : std_logic_vector(3 downto 0);
signal sec10   : std_logic_vector(2 downto 0);

begin
    DUT: SEC port map (
            CLK,BCD1WR,BCD10WR,CIN,
            CO,
            DATAIN,
            sec1,
            sec10
            );

    process
    begin
    CLK <= '1';
    wait for 5ns;
    CLK <= '0';
    wait for 5ns;
    end process;

    BCD1WR <= '0','1' after   6 ns,
                  '0'  after   7 ns;
    BCD10WR<= '0','1' after  18 ns,
                  '0'  after  19 ns;
    DATAIN <= "0110",
              "0101" after  13 ns,
              "XXXX" after  25 ns;
    CIN <= '0', '1'  after   30 ns;
```

```
end stimulus;
```

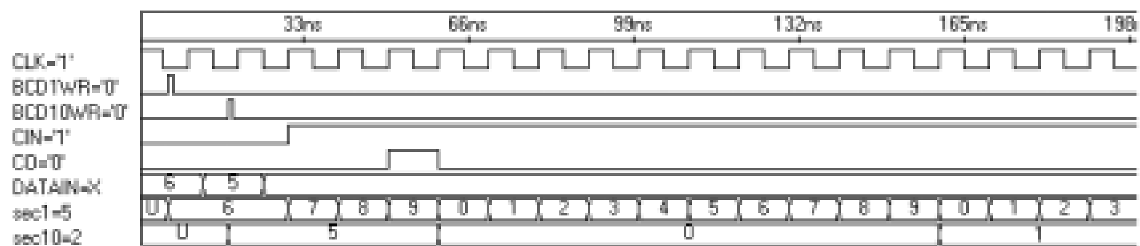


図4.2　６０進カウンタ（SEC）のシミュレーション結果

<リスト 4.3>60 進カウンタ（MIN）の VHDL 記述

```
Library IEEE;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity MIN is
   port (
        CLK,MIN1WR,MIN10WR,CIN  : in std_logic;
        CO      : out std_logic;
        DATAIN  : in  std_logic_vector(3 downto 0);
        MIN1    : out std_logic_vector(3 downto 0);
        MIN10   : out std_logic_vector(2 downto 0)
        );
end MIN;

architecture RTL of MIN is
signal MIN1N  :  std_logic_vector(3 downto 0);
signal MIN10N :  std_logic_vector(2 downto 0);
begin
MIN1 <= MIN1N; MIN10 <= MIN10N;
    process(CLK,MIN1WR)
    begin
        if(MIN1WR='1') then
           MIN1N <= DATAIN;
        elsif(CLK'event and CLK='0') then
           if(CIN='1') then
```

```
            if(MIN1N=9) then
               MIN1N <= "0000";
            else
               MIN1N <= MIN1N + 1;
            end if;
            end if;
            end if;
    end process;
    process(CLK,MIN10WR)
    begin
        if(MIN10WR='1') then
           MIN10N <= DATAIN(2 downto 0);
        elsif(CLK'event and CLK='0') then
           if(CIN='1' and MIN1N=9 ) then
            if(MIN10N=5) then
              MIN10N <= "000";
            else
              MIN10N <= MIN10N + 1;
            end if;
          end if;
        end if;
    end process;

process(MIN10N,MIN1N,CIN)
begin
        if( CIN='1' and MIN1N=9 and MIN10N=5 ) then
            CO <= '1';
        else
            CO <= '0';
        end if;
end process;
end RTL;
```

＜リスト 4.3＞60 進カウンタ（MIN）のテストベンチ

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

```vhdl
use ieee.std_logic_unsigned.all;

entity TESTBNCH is
end TESTBNCH;

architecture stimulus of TESTBNCH is
component MIN is
    port (
        CLK,MIN1WR,MIN10WR,CIN   : in std_logic;
        CO      : out std_logic;
        DATAIN  : in  std_logic_vector(3 downto 0);
        MIN1    : out std_logic_vector(3 downto 0);
        MIN10   : out std_logic_vector(2 downto 0)
    );
end component;

signal CLK,MIN1WR,MIN10WR,CIN   : std_logic;
signal CO      : std_logic;
signal DATAIN  : std_logic_vector(3 downto 0);
signal MIN1    : std_logic_vector(3 downto 0);
signal MIN10   : std_logic_vector(2 downto 0);

begin
    DUT: MIN port map (
            CLK,MIN1WR,MIN10WR,CIN,
            CO,
            DATAIN,
            MIN1,
            MIN10
            );

    process
    begin
    CLK <= '1';
    wait for 5ns;
    CLK <= '0';
```

```
    wait for 5ns;
    end process;

    MIN1WR <= '0','1' after   6 ns,
                '0'    after   7 ns;
    MIN10WR<= '0','1' after  18 ns,
                '0'    after  19 ns;
    DATAIN <= "0110",
              "0101"   after   13 ns,
              "XXXX"   after   25 ns;
    CIN  <= '0', '1'   after 30 ns;
end stimulus;
```

・ after によるシミュレーション記述

　＜リスト 4,2＞、＜リスト 4,3＞のテストベンチでは、“ after ” を使用して一つ一つの信号を別々に記述する方法をとっている。“ after ” は信号に値が代入されるまでの時間を設定する。まず、“ BCD1WR ” に最初'0'が代入されます。その後は “ after ” を使用して 6ns 後に'1'を代入している。after では最初の時刻からの絶対時間を記述していく。最初から 7ns 時間後、すなわち'1'になってから 1ns 後にまた'0'を代入する。

4.4.3　24 進カウンタ（時）

　時間は 24 時間表示で表すことにした。0～23 までをカウントし、また０に戻るという動作を記述した。後のまとめで合わせるために時間の始まりを 23 時からとしている。

　時の VHDL 記述とテストベンチを＜リスト 4,4＞にシミュレーション結果を図 4.3 に示す。

＜リスト 4,4＞24 進カウンタ（時）の VHDL 記述

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

```
entity HOUR2 is
    port (
        CLK,RESET   : in  std_logic;
        CO         : out std_logic;
        COUNT       : out std_logic_vector(4 downto 0)
    );
end HOUR2;

architecture RTL of HOUR2 is

signal COUNT_IN : std_logic_vector(4 downto 0);

begin

COUNT <= COUNT_IN;

    process(CLK,RESET)
    begin

        if(RESET = '1') then

COUNT_IN <= "10111";

        elsif(CLK'event and CLK='0') then

if(COUNT_IN = "10111") then
COUNT_IN <= "00000";

else
COUNT_IN <= COUNT_IN + '1';
end if;
if(COUNT_IN = "10110") then
CO <= '1';

else
CO <= '0';
```

```
end if;
end if;
end process;

end RTL;
```

＜リスト 4,4＞24 進カウンタ（時）のテストベンチ

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

entity TESTBNCH is
end TESTBNCH;

architecture stimulus of TESTBNCH is
component HOUR2 is
    port (
        CLK,RESET      : in  std_logic;
        CO            : out std_logic;
        COUNT          : out std_logic_vector(4 downto 0)
    );
end component;

constant CLOCK: time := 20 ns;

signal CLK,RESET    : std_logic;
signal CO          : std_logic;
signal COUNT        : std_logic_vector(4 downto 0);

begin
    DUT: HOUR2 port map (
            CLK=>CLK,
            RESET=>RESET,
            CO=>CO,
```

```
            COUNT=>COUNT
          );

    CLOCK1: process
    begin
CLK<='1';    wait for CLOCK/2;
CLK<='0';    wait for CLOCK/2;
    end process CLOCK1;

    STIMULUS1: process
    begin
RESET<='0';       wait for CLOCK/3;
RESET<='1';       wait for CLOCK;
RESET<='0';       wait;
end process STIMULUS1;

end stimulus;
```

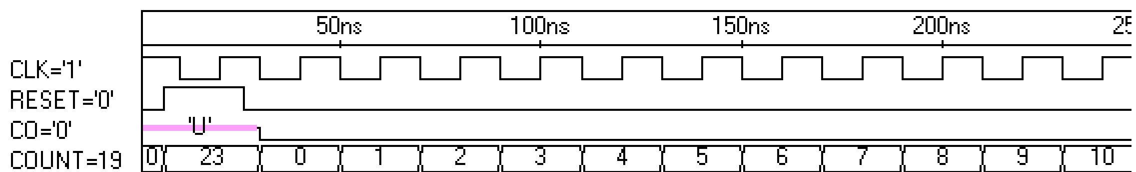


図 4.3　24 進カウンタ（時）のシミュレーション結果

4.4.4　時、分、秒の合成

上記で記した時,分,秒を階層設計によりまとめる。

まず、秒にはそのまま CLK の入力をいれ、秒から出た桁上げ信号 CO_1 を、分の入力信号（CLK）に代わりに入れ次に分から出た桁上げ信号 CO_2 を時間の入力信号とするような動作記述を考えた。秒では、CLK の立ち上がりで操作するようにしているが、秒、分共に 59 の時に桁上げ信号が出力されるようになっているので、分、時ともに桁上げ信号の立ち下りで動作するように記述した。4.4.3 節でも述べたように、正常な動作は 1.86us から始まるので、それに合わせるように時間は 23 時から始まっている。ここでは CLK＝1sec とすると値が大きすぎるので、CLK＝10ns としておく。

時,分,秒のVHDL記述とテストベンチを<リスト4,5>にシミュレーション結果を図4,4に示す。

<リスト4,5>時,分,秒のVHDL記述

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

entity WATCH is
   port (
      CLK,BCD1WR_1,BCD10WR_1,CIN  : in std_logic;
      RESET                      : in std_logic;
      MIN1WR_1,MIN10WR_1          : in std_logic;
      CO_1      : out std_logic;
      CO_2      : out std_logic;
      CO_3      : out std_logic;
      DATAIN    : in  std_logic_vector(3 downto 0);
      SEC1_1    : out std_logic_vector(3 downto 0);
      SEC10_1   : out std_logic_vector(2 downto 0);
      MIN1_1    : out std_logic_vector(3 downto 0);
      MIN10_1   : out std_logic_vector(2 downto 0);
      COUNT_1   : out std_logic_vector(4 downto 0)
         );
end WATCH;

architecture RTL of WATCH is

component SEC
    port (
       CLK,BCD1WR,BCD10WR,CIN : in std_logic;
       CO      :  out std_logic;
       DATAIN  :  std_logic_vector(3 downto 0);
       SEC1    :  std_logic_vector(3 downto 0);
       SEC10   :  std_logic_vector(2 downto 0)
);
end component;
component MIN
```

```
    port (
        CLK,MIN1WR,MIN10WR,CIN  :in  std_logic;
        CO      : out std_logic;
        DATAIN  :  in  std_logic_vector(3 downto 0);
        MIN1    :  out std_logic_vector(3 downto 0);
        MIN10   :  out std_logic_vector(2 downto 0)
);
end component;

component HOUR2
    port (
        CLK,RESET  :in std_logic;
        CO         :out std_logic;
        COUNT      :out std_logic_vector(4 downto 0)
);
end component;

begin
U0: SEC    port map
( CLK,BCD1WR_1,BCD10WR_1,CIN,CO_1,DATAIN,SEC1_1,SEC10_1);
U1: MIN    port map
( CO_1,MIN1WR_1,MIN10WR_1,CIN,CO_2,DATAIN,MIN1_1,MIN10_1);
U2: HOUR2  port map ( CO_2,RESET,CO_3,COUNT_1);


end RTL;
```

＜リスト 4,5＞時,分,秒のテストベンチ

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

entity TESTBNCH is
end TESTBNCH;

architecture stimulus of TESTBNCH is
component WATCH is
```

```
    port (
        CLK,BCD1WR_1,BCD10WR_1,CIN  : in std_logic;
        MIN1WR_1,MIN10WR_1          : in std_logic;
        CO_1      : out std_logic;
        CO_2      : out std_logic;
        CO_3      : out std_logic;
        DATAIN    : in  std_logic_vector(3 downto 0);
        SEC1_1    : out std_logic_vector(3 downto 0);
        SEC10_1   : out std_logic_vector(2 downto 0);
        MIN1_1    : out std_logic_vector(3 downto 0);
        MIN10_1   : out std_logic_vector(2 downto 0);
        COUNT_1   : out std_logic_vector(4 downto 0);
        RESET     : in  std_logic
     );

end component;

constant CLOCK : time := 10 ns;

signal CLK,BCD1WR_1,BCD10WR_1,CIN   :  std_logic;
signal MIN1WR_1,MIN10WR_1           :  std_logic;
signal CO_1                   :  std_logic;
signal CO_2                   :  std_logic;
signal CO_3                   :  std_logic;
signal DATAIN                 :  std_logic_vector(3 downto 0);
signal SEC1_1                 :  std_logic_vector(3 downto 0);
signal SEC10_1                :  std_logic_vector(2 downto 0);
signal MIN1_1                 :  std_logic_vector(3 downto 0);
signal MIN10_1                :  std_logic_vector(2 downto 0);
signal COUNT_1                :  std_logic_vector(4 downto 0);
signal RESET                  :  std_logic;

begin
    DUT: WATCH port map (
         CLK         => CLK,
         RESET       => RESET,
```

```
        BCD1WR_1     => BCD1WR_1,
        BCD10WR_1    => BCD10WR_1,
        CIN          => CIN,
        MIN1WR_1     => MIN1WR_1,
        MIN10WR_1    => MIN10WR_1,
        CO_1         => CO_1,
        CO_2         => CO_2,
        CO_3         => CO_3,
        DATAIN       => DATAIN,
        SEC1_1       => SEC1_1,
        SEC10_1      => SEC10_1,
        MIN1_1       => MIN1_1,
        MIN10_1      => MIN10_1,
        COUNT_1      => COUNT_1
       );

CLOCK1: process
begin
clk <= '1';  wait for CLOCK/2;
clk <= '0';  wait for CLOCK/2;
end process CLOCK1;

STIMULUS1 : process
begin
RESET<='0'; wait for CLOCK/3;
RESET<='1'; wait for CLOCK;
RESET<='0'; wait;
end process STIMULUS1;


    BCD1WR_1        <=   '0','1' after 6ns,
                        '0'     after 7ns;
    BCD10WR_1       <=   '0','1' after 18ns,
                        '0'     after 19ns;
    MIN1WR_1        <=   '0','1' after 6ns,
                        '0'     after 7ns;
```

```
    MIN10WR_1          <=   '0','1' after 18ns,
                           '0'     after 19ns;
    DATAIN            <=   "0110",
                           "0101"  after 13ns,
                           "XXXX"  after 25ns;
    CIN              <=   '0','1' after 30ns;

end stimulus;
```

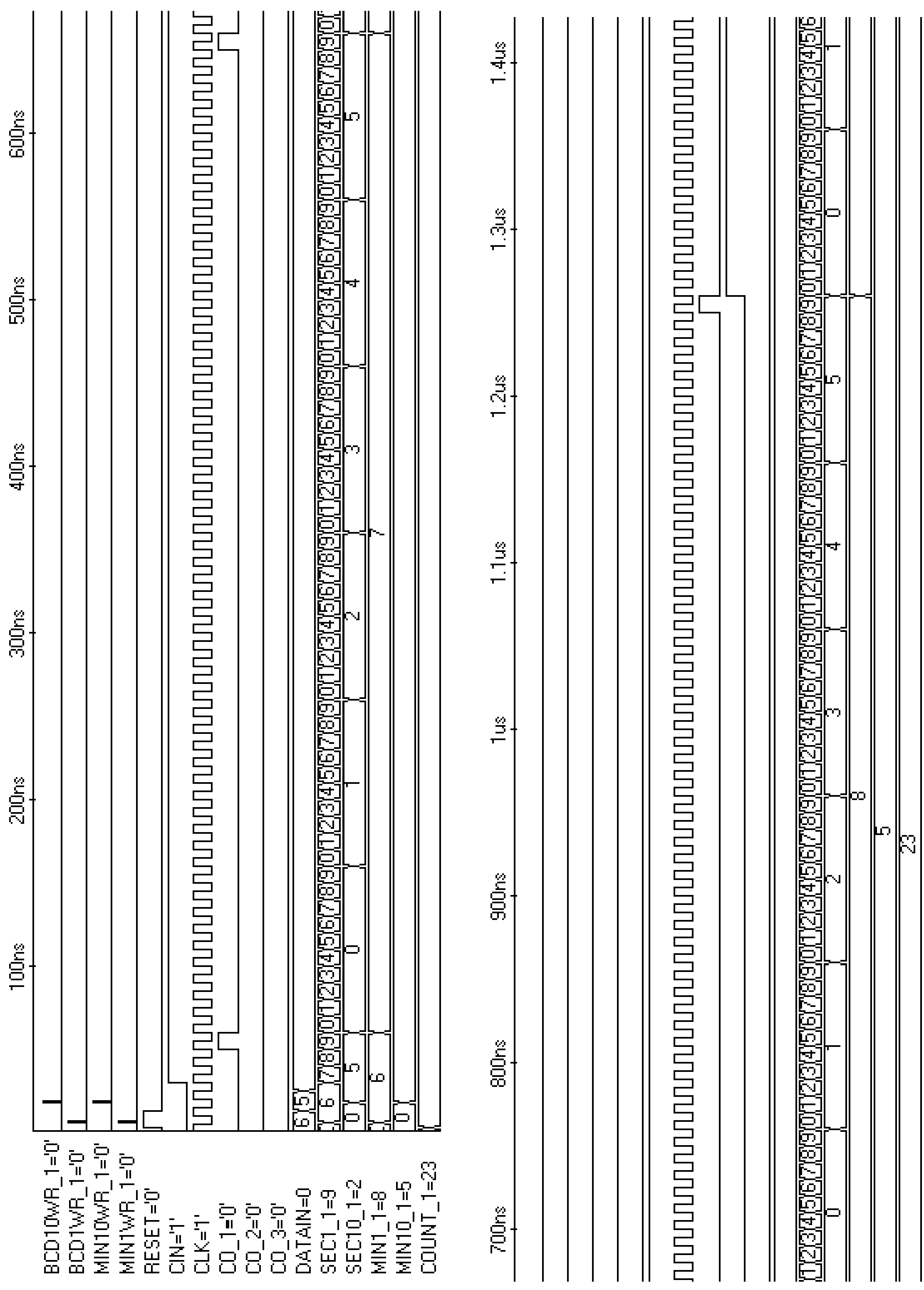


図 4,4　時,分,秒のシミュレーション結果①

1.4us 1.5us 1.6us 1.7us 1.8us 1.9us 2us

2.1us 2.2us 2.3us 2.4us 2.5us 2.6us 2.7us

図 4,4　時,分,秒のシミュレーション結果②

第 5 章 まとめ

本研究において、ディジタル回路の設計方法および VHDL の基礎を学び、ディジタル時計を設計することで理解した。第 3 章では基板上に実装するために PSpice 等を用いて回路設計を試みた。Pspice では求める波形が得られたが、実際基板上に実装してみるとシミュレーション通りにはいかなかった。第 4 章では VHDL によってディジタル時計の設計を試みた。VHDL 記述では、なかなか思い通りにいかず苦しんだが、エラー等を直しているうちに徐々に理解を深めることができたと思う。双方ともまだまだ改良できる点があったのだが良い方法が見つからなかったのが残念だった。

本研究において、ディジタル回路設計および VHDL の理解を深めることができた。今回の研究を行ったことによって、今後色々な面で役立っていくと思っている。

謝辞

本研究を進めるにあたり、貴重なご指導、ご助力を頂いた高知工科大学工学部 電子・光システム工学科 矢野政顕　教授に深く感謝いたします。

また、日頃から多くのご助言を頂き大変お世話になりました原央教授、橘昌良助教授、他先生方に厚くお礼申し上げます。

最後に直接的なご指導、ご助言を賜りました大学院生の村松暢也氏、石川純平氏他、同研究室の皆様方に心から感謝の意を表します。

参考文献

[1]ディジタル電子回路　藤井信生著　講談社

[2]ディジタル IC 回路の基礎　　松田勲／井原充博著　技術評論社

[3]論理回路　　高木直史著　株式会社　昭晃堂

[4]VHDL によるハードウェア設計入門　　長谷川裕恭著　CQ 出版社

[5]コンピューターアーキテクチャの基礎　　柴山潔著　　近代科学社

付録１

case 文の記述

case 文は、

when　値　=＞　文　　　　　注：　“ =＞ ”は関係演算子ではない

という形式で記述していく。

If 文の場合は、最初の条件が処理されたあと、次の条件が処理されるが、case 文の場合は値の順番がなく、すべてが並列に処理される。したがって、1 度“ when ”項に記述した値を、そのあとで再び使用すると文法エラーになってしまう。値が重複しないように注意する。また、すべての場合を記述しないと文法エラーになる。使わない値は“ others ”と記述し、すべての場合を記述するのはたいへんなので、これで対応する。“ others ”という記述は残りすべての場合という意味になる。＜表 2＞に case 文の書式を示す。

＜表 2＞　case 文の書式

```
case 式 is
条件式
end　case；
条件式：when 値＝＞順次処理文
　　　　when 値｜値｜値＝＞順次処理文
　　　　when 値 to 値＝＞順次処理文
　　　　when other＝＞順次処理文
```

コンポーネント宣言

VHDL では、コンポーネント・インスタンス文で下位のモジュールを呼び出す前に、“ architecture ”と“ begin ”の間にそのモジュールがどのような形で構成されているかを表すコンポーネント宣言をしなければならない。各記述をそれぞれ単独で処理できるようにするため、このような記述が必要となる。エンティティ宣言との違いは、“ entity ”を“ component ”に変えて、“ is ”を消去し、

最後を “ end component ” にするだけである。

　component　コンポーネント名

　　　　　[ジェネリック文]

　　　　　[ポート文]

　end　component；

　コンポーネント宣言は、“ architecture ” と “ begin ” の間以外に、パッケージ文で宣言することもできる。

コンポーネント・インスタンス文

　コンポーネント・インスタンス文は

　　ラベル名　：　コンポーネント名　port　map　（信号，…）；

と記述する。ラベル名は、このコンポーネントに付けられる名前で、そのアーキテクチャ宣言の中で唯一のものでなければならない。

　下位階層のポートと信号は、“ port　map ” で結合する。＜リスト１＞（FULL_ADDER）の①は位置による関連付けになり、ポート文で書かれた順番に結合される。最初に書かれている信号 “ U0_S ” は、“ HALF_ADDER ” の最初のポート ‘ A ’ に、“ CIN ” は ‘ B ’ に、‘ S ’ は ‘ S ’ に、“ U1_CO ” は “ CO ” に結合される。結合のさせかたは、「位置による関連付け」と「名前による関連付け」の 2 種類がある。

　名前による関連付けは、

　　ポート名　＝＞　信号名

と記述する。FULL_ADDER の VHDL 記述例を＜リスト１＞に記す。

＜リスト１＞FULL_ADDER の VHDL 記述例

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

entity FA is
    port (
        A,B,CIN: in std_logic;
```

```
      S,CO: out std_logic
   );
end FA;

architecture STRUCTURE of FA is

component HA
port ( A,B : in std_logic;
      S,CO : out std_logic);
end component;

signal U0_CO,U0_S,U1_CO : std_logic;

begin

U0 : HA port map ( A,B,U0_S,U0_CO );
U1 : HA port map ( U0_S,CIN,S,U1_CO ); --①

CO <= U0_CO or U1_CO;


end STRUCTURE;
```

コメント文

VHDL では“ －－ ”から行末までコメント文になる。コメント文はリストの中にその作成者の名前を入れたり、日付を入れたり、その信号の意味や機能などを付加します。コメント文は VHDL ではいっさい処理されないが

wait 文

単純に“ wait ; ”と記述すると無限実行停止という意味になり、このプロセス文はそこから永遠に動作しなくなる。

Wait 文には、この他に“ wait until ”や“ wait on ”がある。“ wait on ”は、信号が変化するまで実行を停止するという意味である。“ wait on ”とプロセス文のセンシティビティ・リストの違いは“ wait　on ”がプロセス文中に何度も使用

できるのに対し、センシティビティ･リストは文頭に一度しか使用できない点である。

プロセス文にセンシティビティ･リストがある場合は、プロセス文内で wait 文を使用することはできない。

Wait 文の種類を＜表 1＞に示す。

＜表 1＞　wait 文の種類

wait on 信号､信号･･･
wait until 条件
wait for 時間
wait

付録 2

カレンダー記述

時計の桁上げからカレンダーを動かそうとしたが間に合わなかったので別に作ったものを示す。今回はうるう年を考えないで VHDL 記述した。年は 4 桁を表すことにし、1980 年の場合、上位 19 と下位 80 とに分けて記述することにした。動作はリセットが 1 から 0 に変化した後の初めのクロックの立ち上がりから動作するようになっている。

それぞれの VHDL 記述とテストベンチを下記に記す。

日,月の VHDL 記述

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

entity DAYMONTH is
   port (
        CLK,RESET : in  std_logic;
        DAY       : out std_logic_vector(4 downto 0);
        MONTH     : out std_logic_vector(3 downto 0);
        CO        : out std_logic
        );

end DAYMONTH;

architecture RTL of DAYMONTH is

signal NICHI : std_logic_vector(4 downto 0);
signal TUKI  : std_logic_vector(3 downto 0);

begin
DAY   <= NICHI;
MONTH <= TUKI;

process(RESET,CLK)
```

```
begin
 if (RESET='1') then
           TUKI  <= "0001";
           NICHI <= "00000";
 elsif(CLK'event and CLK='1') then

    if(TUKI= "0001" or TUKI= "0011" or TUKI= "0101" or
      TUKI= "0111" or TUKI= "1000" or TUKI= "1010" ) then
           if(NICHI="11111")then
              NICHI <= "00001";
              TUKI  <= TUKI + '1';
              CO    <= '0';
           else
              NICHI<=NICHI+'1';
              CO    <= '0';
           end if;
    elsif(TUKI="0010") then
         if(NICHI="11100") then
          NICHI <= "00001";
          TUKI  <= TUKI + '1';
          CO    <= '0';
         else
          NICHI <= NICHI + '1';
          CO    <= '0';
         end if;
    elsif(TUKI="1100") then
          if(NICHI="11111") then
          NICHI <= "00001";
          TUKI  <= "0001";
          CO    <= '1';
          else
          NICHI <= NICHI+'1';
          CO    <= '0';
          end if;
    else
           if(NICHI="11110")then
```

```
                NICHI <= "00001";
                TUKI  <= TUKI + '1';
                CO    <= '0';
              else
                NICHI <= NICHI + '1';
                CO    <= '0';
              end if;
      end if;
  end if;
end process;

end RTL;
```

日、月のテストベンチ

```
library ieee;
use ieee.std_logic_1164.all;
use std.textio.all;
use work.CALE;

entity TESTBNCH is
end TESTBNCH;

architecture STIMULUS of TESTBNCH is

component DAYMONTH is
   port (
   CLK   : in  std_logic;
   RESET : in  std_logic;
   DAY   : out std_logic_vector(4 downto 0);
   MONTH : out std_logic_vector(3 downto 0);
   CO    : out std_logic
   );
end component;

constant CLOCK : time := 10 ns;
```

```
signal CLK   : std_logic;
signal RESET : std_logic;
signal DAY   : std_logic_vector(4 downto 0);
signal MONTH : std_logic_vector(3 downto 0);
signal CO    : std_logic;

begin
    DUT: DAYMONTH port map (
             CLK   => CLK,
             RESET => RESET,
             DAY   => DAY,
             MONTH => MONTH,
             CO    => CO
           );

STIMULUS2:process
begin
CLK<='0'; wait for CLOCK;
CLK<='1'; wait for CLOCK;
end process STIMULUS2;

STIMULUS3:process
begin
RESET<='0';wait for 3ns;
RESET<='1';wait for 6ns;
RESET<='0';wait;
end process STIMULUS3;

end STIMULUS;
```

年（下位二桁）の VHDL 記述

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

entity YEAR_DOWN is
   port (
       CLK,RESET  : in  std_logic;
       YEAR_D     : out std_logic_vector(6 downto 0);
       YEAR_D_OUT : out std_logic
   );
end YEAR_DOWN;

architecture RTL of YEAR_DOWN is
signal YEAR_D_IN : std_logic_vector(6 downto 0);
begin

YEAR_D <= YEAR_D_IN;

   process(CLK,RESET)  begin

       if(RESET='1') then

YEAR_D_IN <= "0000000";
YEAR_D_OUT<= '0';

       elsif(CLK'event and CLK='1') then

if(YEAR_D_IN = "1100011") then
YEAR_D_IN <= "0000000";
YEAR_D_OUT<= '1';
else
YEAR_D_IN <= YEAR_D_IN + 1;
YEAR_D_OUT<= '0';
end if;
end if;
```

```
end process;
end RTL;
```

年（下位二桁）のテストベンチ

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

entity TESTBNCH is
end TESTBNCH;

architecture stimulus of TESTBNCH is
component YEAR_DOWN is
    port (
        CLK,RESET: in std_logic;
        YEAR_D: out std_logic
    );

end component;

constant PERIOD: time := 20 ns;

signal CLK,RESET: std_logic;
signal YEAR_D: std_logic_vector(6 downto 0);


begin
    DUT: YEAR_DOWN port map (
            CLK=>CLK,
            RESET=>RESET,
            YEAR_D=>YEAR_D
          );

    CLOCK1: process

begin
```

```
clk<='1';        wait for PERIOD/2;
clk<='0';        wait for PERIOD/2;
end process CLOCK1;

STIMULUS1: process
begin
RESET<='0'; wait for PERIOD/3;
RESET<='1'; wait for PERIOD;
RESET<='0'; wait;
end process STIMULUS1;

end stimulus;
```

年（上位二桁）の VHDL 記述

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

entity YEAR_UP is
    port (
        CLK,RESET  : in  std_logic;
        YEAR_U     : out std_logic_vector(6 downto 0)
    );
end YEAR_UP;

architecture RTL of YEAR_UP is
signal YEAR_U_IN : std_logic_vector(6 downto 0);
begin

YEAR_U <= YEAR_U_IN;

    process(CLK,RESET)  begin

        if(RESET='1') then

YEAR_U_IN <= "0000000";
```

```
      elsif(CLK'event and CLK='1') then

if(YEAR_U_IN = "1100011") then
YEAR_U_IN <= "0000000";
else
YEAR_U_IN <= YEAR_U_IN + 1;
end if;
end if;

end process;
end RTL;
```

年（上位二桁）のテストベンチ

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

entity TESTBNCH is
end TESTBNCH;

architecture stimulus of TESTBNCH is
component YEAR_UP is
   port (
      CLK,RESET: in std_logic;
      YEAR_U: out std_logic
   );

end component;

constant PERIOD: time := 20 ns;

signal CLK,RESET: std_logic;
signal YEAR_U: std_logic_vector(6 downto 0);
```

```
begin
   DUT: YEAR_UP port map (
            CLK=>CLK,
            RESET=>RESET,
            YEAR_U=>YEAR_U
         );

   CLOCK1: process

begin
clk<='1';       wait for PERIOD/2;
clk<='0';       wait for PERIOD/2;
end process CLOCK1;

STIMULUS1: process
begin
RESET<='0'; wait for PERIOD/3;
RESET<='1'; wait for PERIOD;
RESET<='0'; wait;
end process STIMULUS1;

end stimulus;
```

日、月、年の階層設計の VHDL 記述

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

entity TOTALcale is
    port (
        CLK,RESET    : in  std_logic;
        DAY_1        : out std_logic_vector(4 downto 0);
        MONTH_1      : out std_logic_vector(3 downto 0);
        CO_1         : out std_logic;
        YEAR_D1      : out std_logic_vector(6 downto 0);
        YEAR_D_OUT1  : out std_logic;
        YEAR_UP1     : out std_logic_vector(6 downto 0)
    );
end TOTALcale;

architecture RTL of TOTALcale is

component DAYMONTH
    port (
        CLK,RESET    : in  std_logic;
        DAY          : out std_logic_vector(4 downto 0);
        MONTH        : out std_logic_vector(3 downto 0);
        CO           : out std_logic
);
end component;

component YEAR_DOWN
    port (
        CLK,RESET    : in    std_logic;
        YEAR_D       : out   std_logic_vector(6 downto 0);
        YEAR_D_OUT   : out   std_logic
);
end component;
```

```
component YEAR_UP
    port (
        CLK,RESET     : in    std_logic;
        YEAR_U        : out   std_logic_vector(6 downto 0)
);
end component;

begin
U0:  DAYMONTH  port map ( CLK, RESET, DAY_1, MONTH_1, CO_1);
U1:  YEAR_DOWN port map ( CO_1, RESET, YEAR_D1, YEAR_D_OUT1);
U2:  YEAR_UP   port map ( YEAR_D_OUT1, RESET, YEAR_UP1);

end RTL;
```

日,月,年の階層設計のテストベンチ

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

entity TESTBNCH is
end TESTBNCH;

architecture stimulus of TESTBNCH is
component TOTALcale is
    port (
        CLK,RESET    : in     std_logic;
        DAY_1        : out    std_logic_vector(4 downto 0);
        MONTH_1      : out    std_logic_vector(3 downto 0);
        CO_1         : out    std_logic;
        YEAR_D1      : out    std_logic_vector(6 downto 0);
        YEAR_D_OUT1  : out    std_logic;
        YEAR_UP1     : out    std_logic_vector(6 downto 0)
    );

end component;
constant PERIOD: time := 20 ns;
```

```
signal CLK,RESET   : std_logic;
signal DAY_1       : std_logic_vector(4 downto 0);
signal MONTH_1     : std_logic_vector(3 downto 0);
signal CO_1        : std_logic;
signal YEAR_D1     : std_logic_vector(6 downto 0);
signal YEAR_D_OUT1 : std_logic;
signal YEAR_UP1    : std_logic_vector(6 downto 0);

begin
   DUT: TOTALcale port map (
          CLK          =>  CLK,
          RESET        =>  RESET,
          DAY_1        =>  DAY_1,
          MONTH_1      =>  MONTH_1,
          CO_1         =>  CO_1,
          YEAR_D1      =>  YEAR_D1,
          YEAR_D_OUT1  =>  YEAR_D_OUT1,
          YEAR_UP1     =>  YEAR_UP1
        );

CLOCK1: process
begin
clk<='1';    wait for PERIOD/2;
clk<='0';    wait for PERIOD/2;
end process CLOCK1;

STIMULUS1: process
   begin
RESET<='0'; wait for 3  ns;
RESET<='1'; wait for 17 ns;
RESET<='0'; wait;
end process STIMULUS1;

end stimulus;
```