卒 業 研 究 報 告

題　　目

アラーム付きストップウォッチの設計･製作

指 導 教 員

矢野　政顕　教授

報 告 者

学籍番号：1030174
氏名：　大川内　正樹

平成 15 年　2 月　24 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

現在、コンピュータや携帯電話などに代表される電子機器は、高性能化、多機能化、小型化が進み、私たちの生活には必需品となっている。こういった電子機器には集積回路（Integrated Circuit）が組み込まれており、現在も急激なスピードで集積化が進んでいる。この集積化技術の飛躍的な進歩により、私たちの生活も大変便利になってきた。こういった集積回路の設計技術の基礎となるのがディジタル回路である。そして、集積化技術の飛躍的な向上に伴い、設計するディジタル回路の規模は増大し、多種にわたる設計手法が考案されてきた。

本研究では、ディジタル回路に関する基礎知識の習得、ディジタル回路の設計手法の習得を目的として、以下の 2 つの設計手法を実践し、比較検討する。

・汎用ロジック IC を用いた回路設計

・ハードウェア記述言語である VHDL（Very high speed integrated circuit Hardware Description Language）による回路設計

これらの設計手法を比較するため、具体的な順序回路の例として、カウンタ動作、比較、切換、表示機能などを持たせたアラーム付きストップウォッチを設計・製作する。

本報告は 5 章から構成されている。第 1 章では卒業研究の背景について述べる。第 2 章ではアラーム付ストップウォッチを設計例として、汎用ロジック IC を用いた回路設計について説明する。第 3 章では、第 2 章で汎用ロジック IC によって構成していたディジタル回路部分をハードウェア記述言語である VHDL を用いて設計し、FPGA に実装した結果について報告する。第 4 章では、汎用ロジック IC を用いた回路と FPGA を用いた回路を比較・検討する。最後に第 5 章で全体をまとめる。

第 2 章　汎用ロジック IC を用いた回路設計

第 2 章では、汎用ロジック IC を用いた順序回路の設計例として、アラーム付ストップウォッチを設計・製作した結果を報告する。回路図エディタ、シミュレータには、Electronics Workbench 社製「Multisim 2001 PowerPro」を使用する。

2.1　仕様（機能）設計

今回設計するストップウォッチには、以下の機能を持たせる。

1. 表示機能
2. ストップウォッチ機能
3. アラーム機能
4. 表示切替機能

（1）表示機能

ストップウォッチの表示は“ 分 ”“ 秒 ”“ 1/10 秒 ”“ 1/100 秒 ” とし、“ 分 ”“ 秒 ” は 0～59、“ 1/10 秒 ”“ 1/100 秒 ” は 0～9 をカウント表示し、小数点も表示する（例：00 00.00）。

アラーム設定時の表示は“ 分 ”“ 秒 ” とし、それぞれ 00～59 をカウント表示する。アラーム設定時は小数点は表示せず、小数点以下の“ 1/10 秒 ”“ 1/100 秒 ” は 0 で表示する（例：00 00 00）。

（2）ストップウォッチ機能

「00 00.00～59 59.99」までカウントできるようにし、1/100 秒単位で最長 1 時間測定可能とする。また、SW1 を ON／OFF することでスタート／ストップの制御を行い、SW2 で内部のカウンタを Reset し、ストップウォッチの表示を「00 00.00」にする。

（3）アラーム機能

ストップウォッチ用とは別にアラーム用の“分”“秒”カウンタを用意し、アラーム用カウンタの BCD 出力と、ストップウォッチ用カウンタの BCD 出力を比較器に入れ、各 bit 入力が一致した場合のみアラームブザーを鳴らす。SW4 でアラームの“分”の設定、SW5 でアラームの“秒”の設定、SW6 でアラームの ON／OFF の制御を行う。SW7 で内部のカウンタを Reset し、アラームの時刻設定表示を「00 00 00」にする。

（4）表示切替機能

SW3 によって､ストップウォッチ表示､アラーム設定表示の切換を制御する。SW（SWITCH）1～7 の機能を表 2－1 に示す。また、LED 機能表を表 2－2 に示す。

表 2－1. SWITCH 機能表

SWITCH	ストップウォッチ	アラーム
SW1	スタート/ストップ	ー
SW2	リセット	ー
SW3	アラームへ表示切替	ストップウォッチへ表示切替
SW4	ー	「分」をカウントアップ
SW5	ー	「秒」をカウントアップ
SW6	アラーム ON/OFF	アラーム ON/OFF
SW7	ー	リセット

表 2－2．LED 機能表

LED	ストップウォッチ	アラーム
7 セグメント LED1	分の上位表示 (0～5)	分の上位表示 (0～5)
7 セグメント LED2	分の下位表示 (0～9)	分の下位表示 (0～9)
7 セグメント LED3	秒の上位表示 (0～5)	秒の上位表示 (0～5)
7 セグメント LED4	秒の下位表示 (0～9) 小数点表示	秒の下位表示 (0～9)
7 セグメント LED5	1/10 秒表示 (0～9)	“0“を表示
7 セグメント LED6	1/100 秒表示 (0～9)	“0“を表示
LED1 (ストップォッチ表示)	点灯	ー
LED2 (アラーム設定表示)	ー	点灯
LED3 (アラーム出力表示)	アラーム出力時、点灯	アラーム出力時、点灯

2.2　ブロック別の設計

今回設計したアラーム付ストップウォッチのブロック図を図 2－1 に示す。

今回設計したストップウォッチは、“ 分 ”“ 秒 ” を「00～59」、“ 1/100 秒 ” を「00～99」まで表示できるようにすることで、「00 00.00～59 59.99」までカウント表示し、1/100 秒単位で最長 1 時間まで測定可能とした。“ 分 ”“ 秒 ” の 00～59 を数えるために 60 進カウンタが、“ 1/100 秒 “ の 00～99 を数えるために 100 進カウンタが必要である。また、これとは別にアラーム用 “ 分 ”“ 秒 ” の 60 進カウンタを用意することで、アラームの時間設定を可能にし、比較回路により、ストップウォッチ回路の時間とアラーム回路の時間が一致した場合に、アラーム出力できるようにした。また、カウンタの値を表示する 7 セグメント LED はストップウォッチ、アラームで別々には用意せず、切替回路を表示回路の手前に用いることで、7 セグメント LED を共通で使用し、表示の切替機能を持たせた。以後、各ブロックごとの設計について説明する。

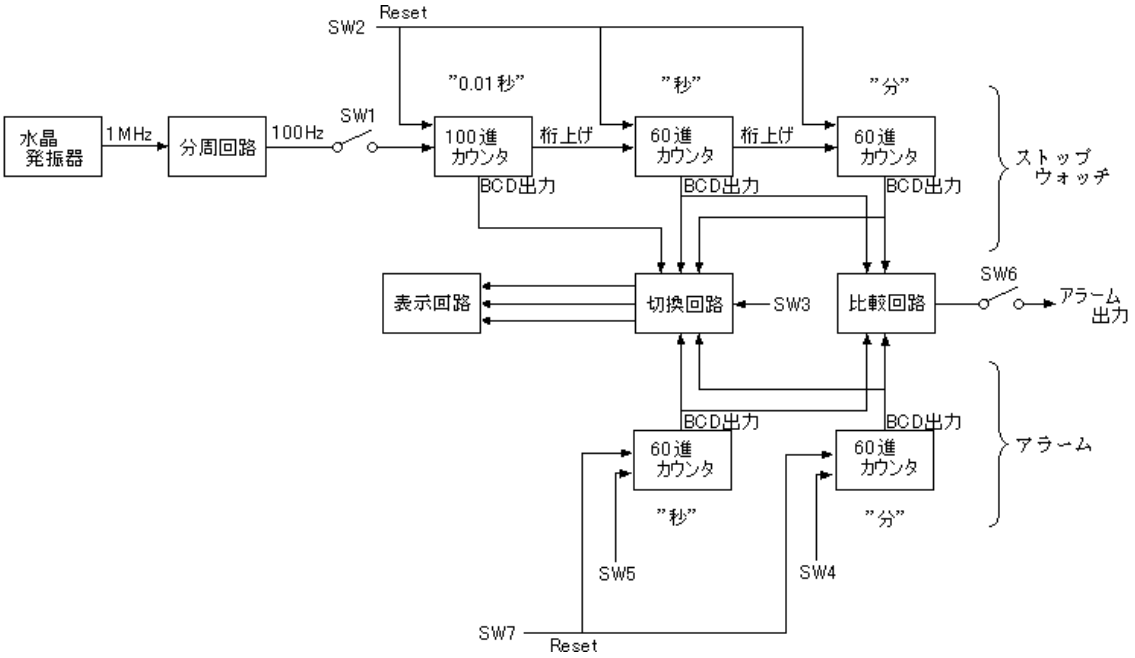


図 2－1. アラーム付きストップウォッチのブロック図

2.2.1　水晶発振器

水晶振動子とは、水晶から切り取った小さい水晶の結晶のことである。この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、水晶振動子の両面に電極をつけ、交流電圧を加えることで、圧力が水晶振動子の両面に交互に加わり水晶が伸び縮みし、共振する。このような水晶振動子を回路に組み込むことにより、正確で安定した発振回路を構成することが出来る。水晶発振器とは、水晶発振子を組み込んだ発振回路である。

今回は、“ 1/100 秒 ” 単位で計測できるストップウォッチを設計する、ある程度精度の良い発振パルスが必要であるため、水晶発振器を使用した。表 3 に今回使用した水晶発振器の主な仕様を示す。

表 3. 水晶発振器 HAT7400A の主な仕様

型　番	基本周波数	周波数安定性	供給電圧	最大消費電流	発振安定時間
HAT7400A	1.000000MHz	±25ppm	＋5.0V±10%	13mA	5ms

この水晶発振器は最大で±25ppm の誤差が発生するため、基本 CLOCK とした場合、最大で 40000 秒（約 11 時間）に 1 秒のずれが生じることになる。しかし、今回は最長 1 時間の計測を目的としているため、十分な精度といえる。

2.2.2　分周回路

分周回路では、水晶発振器の出力である 1MHz のパルスを 1/10000 に分周し、100Hz のパルスを生成する。図 2－2 に分周回路のブロック図を示す。

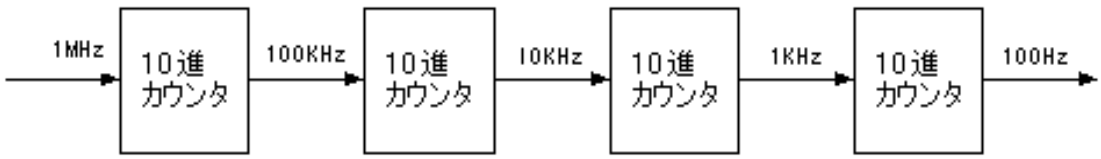
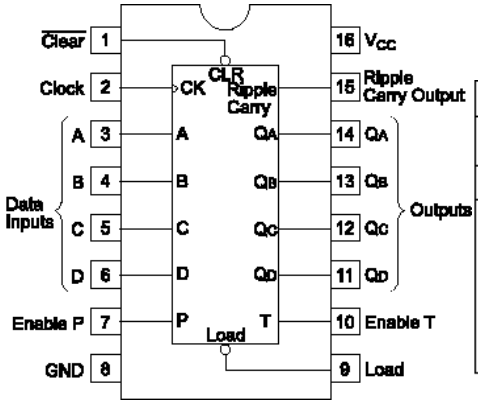


図 2－2. 分周回路のブロック図

1MHz の CLK を、10 進カウンタで 1/10 ずつ 4 回分周し、4 段目の桁上げ出力から 100Hz のパルスを得る。本設計では、10 進（BCD）カウンタとして 74HC160 を使用した。74HC160 のピン配置と真理値表を図 2－3 に、内部回路を図 2－4 に示す。



入力						出力			
$\overline{CLR}$	$\overline{LOAD}$	ENP	ENT	D～A	CLK	QD	QC	QB	QA
L	X	X	X	X	X	L	L	L	L
H	L	X	X	D～A	D～A	D～A を QD～QA へロード			
	H	X	L	X	↓	不変			
		L	X	X	↓	不変			
		H	H	X	↓	カウントアップ			
		X	X	X	X	↑	不変		

X：“H”“L”のいずれでもよい

図 2－3．74HC160 のピン配置と真理値表（参考文献 [6]）

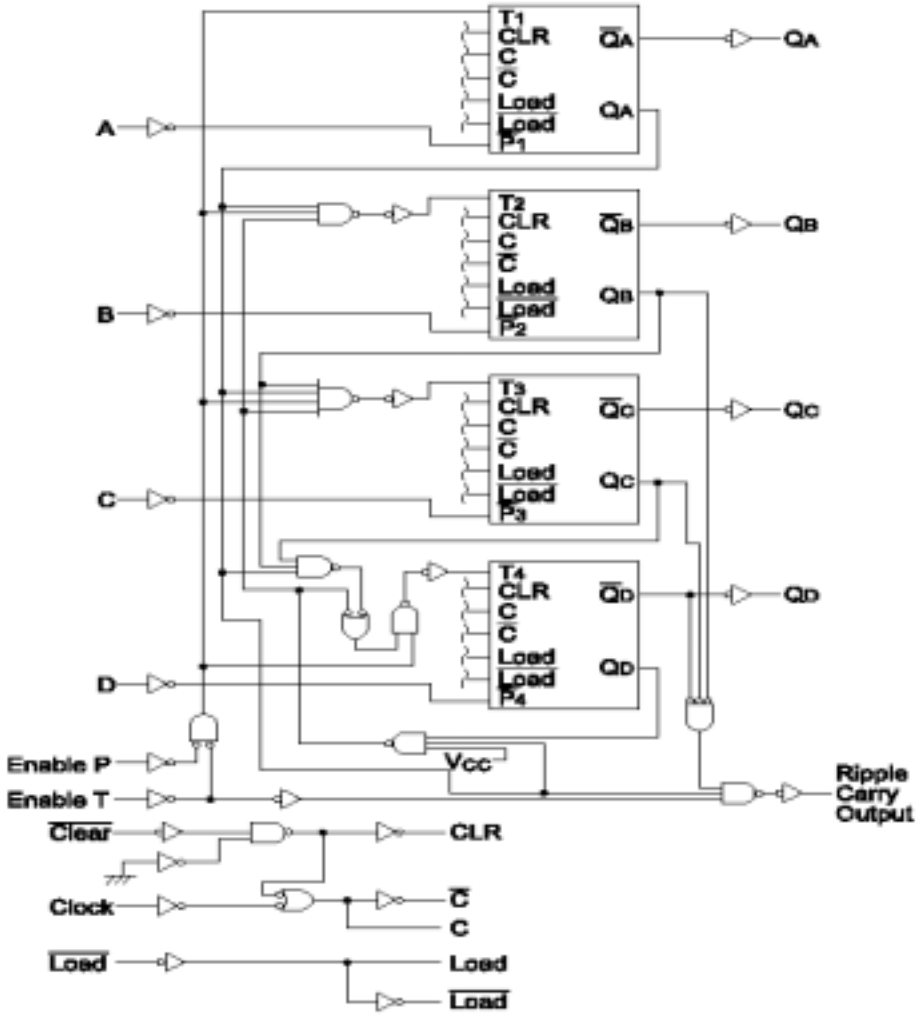


図 2－4．74HC160 の内部回路（参考文献 [6]）

74HC160 は Enable 入力（ENP、ENT）と桁上げ出力 RCO（Rippel Carry Out）を持っているため、外部で AND 等を用いて桁上げ信号を発生させる必要が無い。

図 2－2 の 10 進カウンタに 74HC160 を使用する。分周回路は同期式に動作

さるため CLK は共通の 1MHz を入れ、桁上げ出力 RCO を次段の Enable 入力に入れる。ここで注意すべき点は、74HC160 は単体で使用する場合は問題ないが、Enable 入力を使用して多段接続する際は、2 段目以降の RCO 出力が正常に動作しない場合があることである。図 2－4 に示した 74HC160 の内部回路では、RCO 信号を生成時に ENT 入力との AND をとっている。つまり、ENP を常に“1”とし、ENT に前段の RCO を接続すれば前段までの RCO と AND がとれ、正常に動作するはずであるが、Multi sim でシミュレーションを行った結果、内部の AND ゲートがうまく働いていないのか、前段までの RCO と AND が取れていない（図 2－5 参照）。

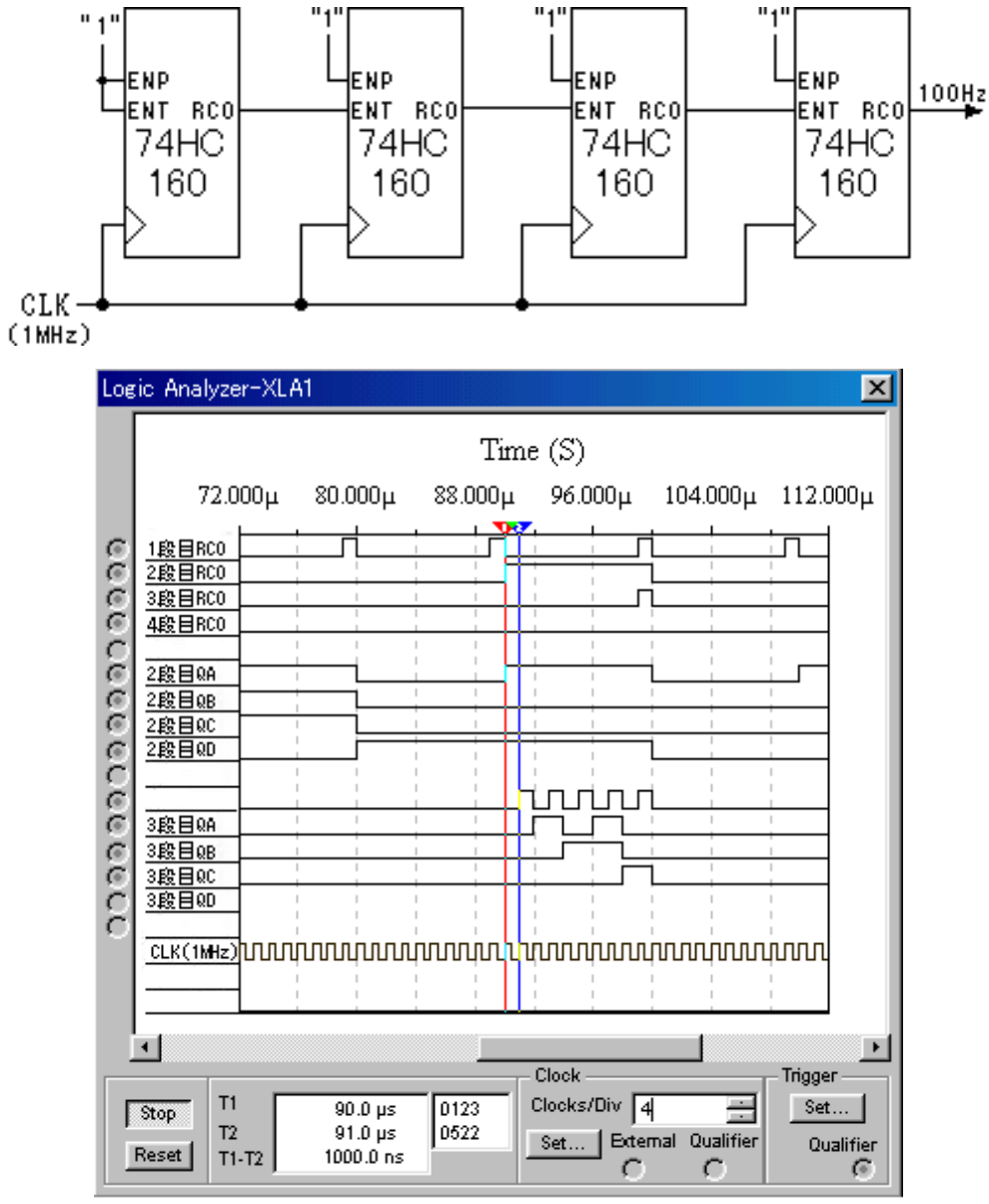


図 2－5．74HC160 の多段接続とそのシミュレーション結果

この桁上げ誤動作の対策のため、RCO を次の段の Enable 入力に入れる際、外部の AND ゲートで前段までの RCO との AND をとり、正確な桁上げ信号を作り直した後で ENT 入力に入れることで、誤動作を防ぐこととした。分周回路における 74HC160 の桁上げ修正接続とそのシミュレーション結果を図 2－5 に示す。今度のシミュレーション結果では、前段の桁上げとの AND が取れており、正常な動作をしていることが確認できる。

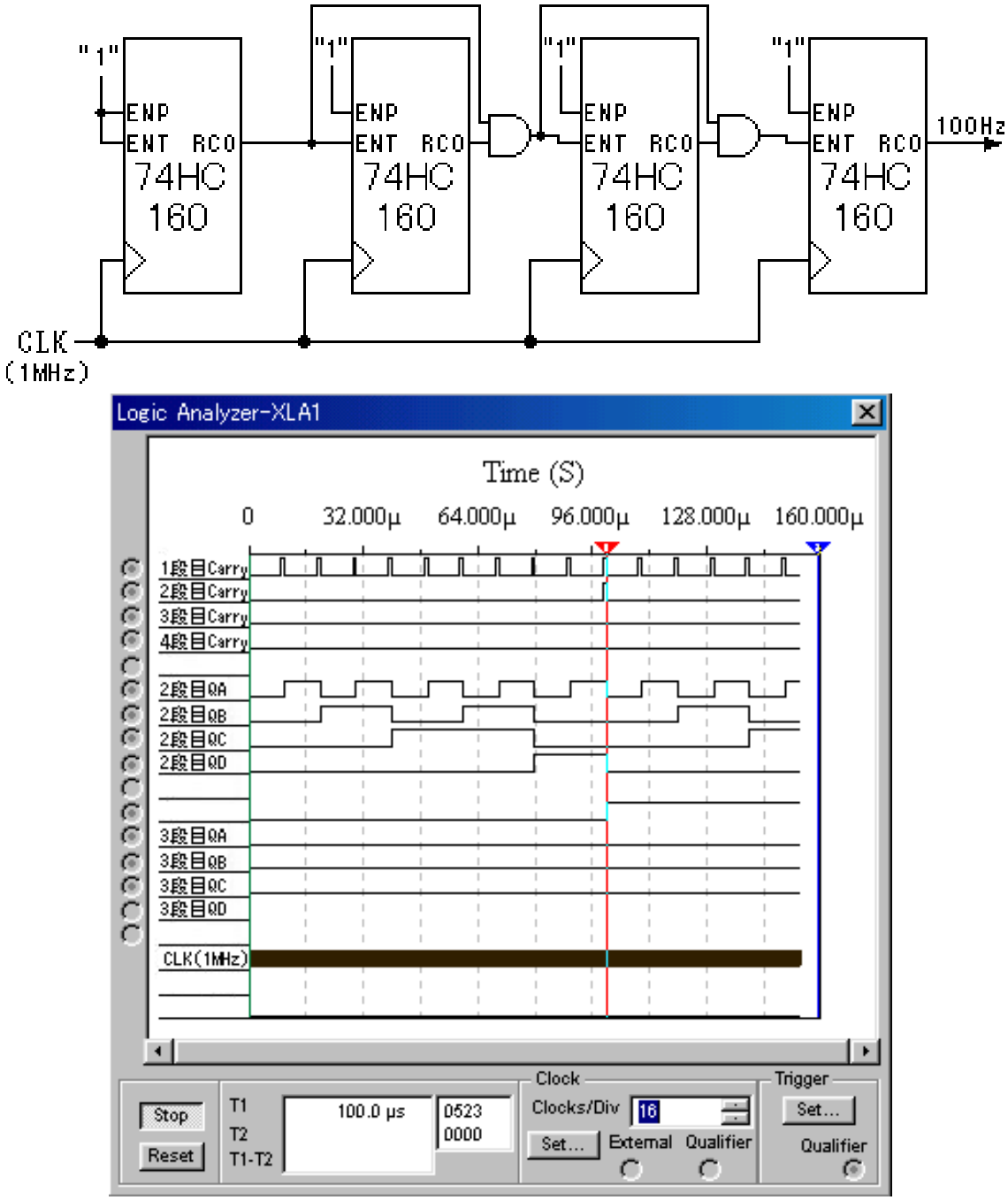
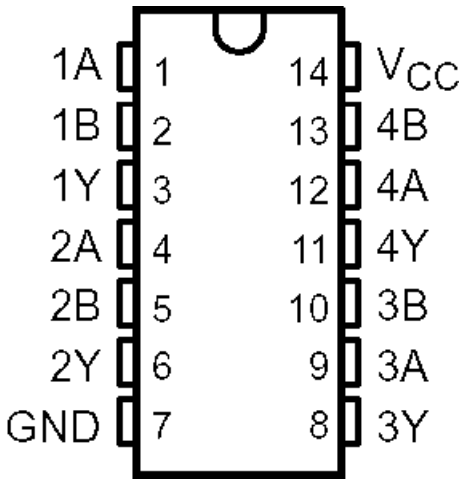


図 2－6. 74HC160 の多段接続（桁上げ修正後）とそのシミュレーション結果

また、図 2－6 における AND の IC は、内部に 2 入力 AND を４つ持つ 74HC08 を使用した。図 2－7 に 74HC08 のピン配置と真理値表、及び内部構成回路を示す。



FUNCTION TABLE
(each gate)

INPUTS		OUTPUT
A	B	Y
H	H	H
L	X	L
X	L	L

図 2－7．74HC08 のピン配置、真理値表、内部構成回路
（参考文献 [5]）

2.2.3　100 進カウンタ

“ 1/100 秒 ” のカウンタは、100 進カウンタである。分周回路から出力された 100Hz のパルスを、インバータを通して CLOCK 入力とし、10 進カウンタ（74HC160）を 2 段構成で使用し、100 進カウンタとする。

2.2.4　60 進カウンタ

ストップウォッチの “ 秒 ”“ 分 ” のカウンタは 60 進カウンタである。“ 1/100 秒 ” の 100 進カウンタの桁上げ出力 RCO を、桁上げ修正（前段の RCO との AND をとる）した後 Enable 入力に入れ、カウントさせる。図 2－8 に“ 秒 ”“ 分 ” の 60 進カウンタの構成を示す。

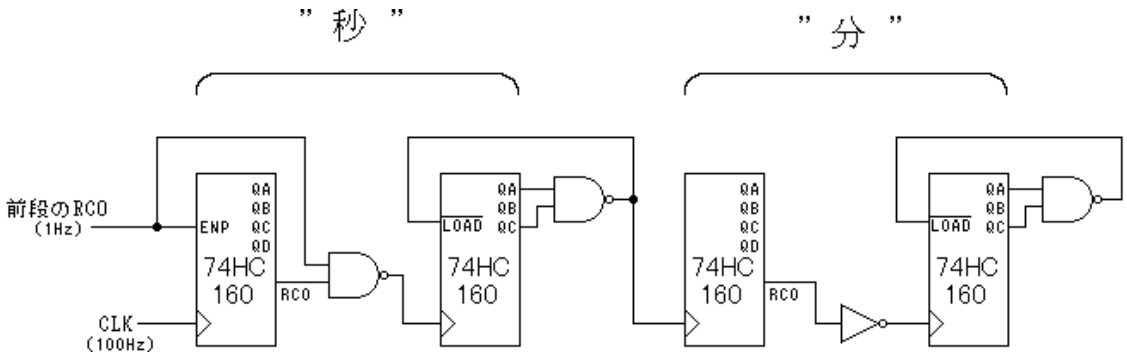
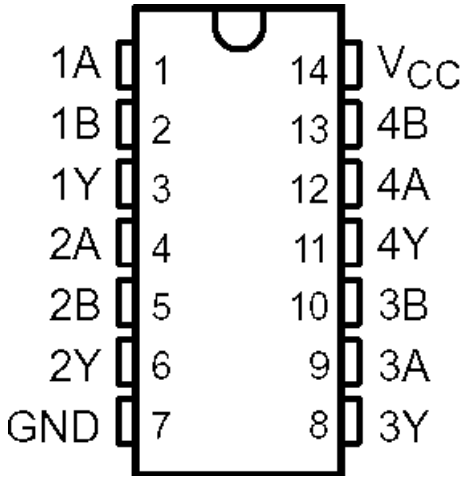


図 2－8. ストップォッチ用 60 進カウンタ（“ 秒 ”“ 分 ”）の構成

60 進カウンタは、10 進カウンタ（74HC160）を 2 段構成にし、2 段目は RCO を使わず、QA と QC の AND をとり、5（101）でデコードして桁上げするようにする。また、この信号を同時に LOAD 入力にいれることで、自身が 5 でリセットされるようにした。

ここで、IC の削減のため、“ 秒 ” の 2 段目以降は前述の桁上げ修正方法（前段の RCO との AND をとる）はとらず、Enable 入力（ENP、ENT）には常に Hi を入力し、5 でデコードした桁上げを次段の CLK 入力へ入れることで、非同期で動作させることにした。この場合、Enable 入力は変化せず、前段の RCO

が CLK となり、カウンタ動作するので、問題であったハザードは生じない。ただし、74HC160 の CLK 入力はポジティブエッジトリガ型であるため、そのままでは前段の桁上げの立上がりでカウントしてしまう。つまり前段のカウンタが 9 の時に RCO が発生し、CLK に入るので、「08→09→10→11」とはならず、「08→19→10→11」と動作してしまう。これを防ぐため、CLK 入力の前にインバータを入れることで、前段の RCO の立ち下がりでカウント動作するようにした。本来この方法はハザードの問題が発生する可能性があるが、今回はシミュレーションでも異常はなかったのでこの方法を使用した。また、LOAD 入力もアクティブローなので、5でデコードする際 NAND を用いることで直接接続できる。NAND 用 IC には内部に NAND を４つ持つ 74HC00 を用いた。74HC00 のピン配置と真理値表、及び内部構成回路を図 2－9 に示す。



FUNCTION TABLE
(each gate)

INPUTS		OUTPUT
A	B	Y
H	H	L
L	X	H
X	L	H

図 2－9．74HC00 のピン配置、真理値表、内部構成回路
（参考文献 [5]）

今回のストップウォッチには、00 分 00 秒～59 分 59 秒まで 1 分、1 秒刻みで時間設定可能なアラーム機能を付ける。ストップウォッチとは別にアラーム用のカウンタを用意し、それを比較することでアラーム出力を ON／OFF する。アラーム用“秒”“分”の 60 進カウンタも先ほど述べた 60 進カウンタと基本構成は同様だが、“秒”から“分”への桁上げはない。そして、“秒”の 60 進カウンタは SW5 を、“分”の 60 進カウンタは SW4 を CLOCK 入力とする。そうすることで、SW4、SW5 を押すたびにカウントアップし、アラームの時間設定ができるようにしている。（図 2－10 参照）

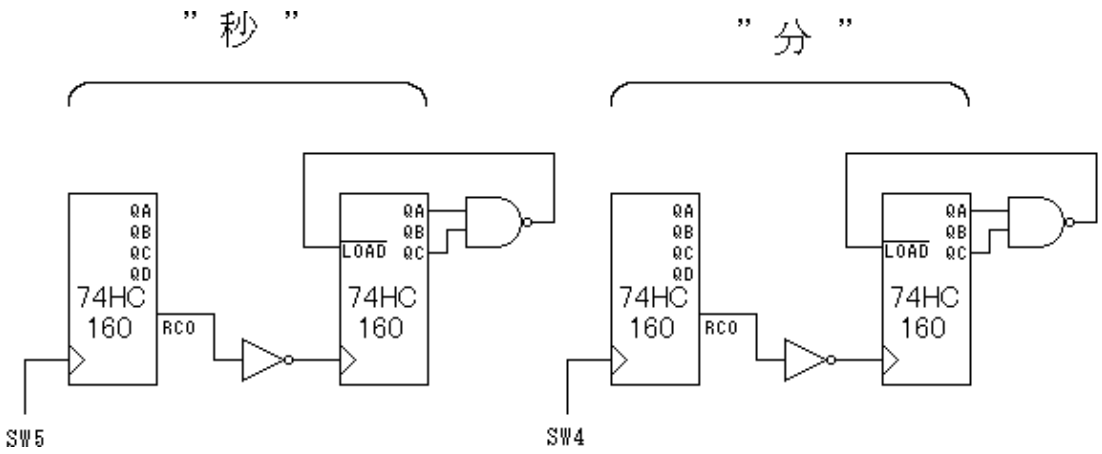


図 2－10．アラーム用 60 進カウンタ（“秒”“分”）の構成

2.2.5　比較回路

比較回路には 4bit comparator（74HC85）を 4 個使用する。74HC85 のピン配置を図 2－11 に、真理値表を図 2－12 に示す。

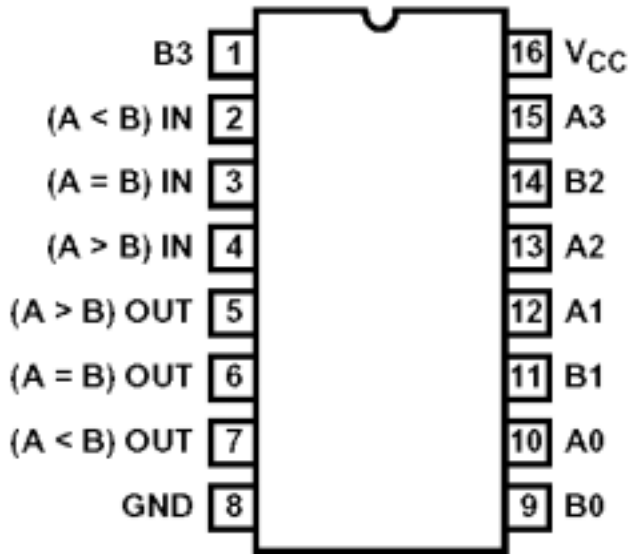
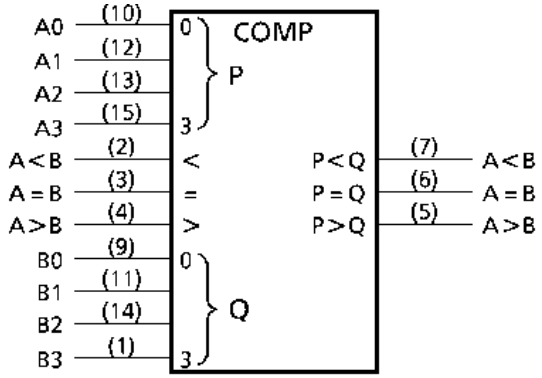


図 2－11．74HC85 のピン配置



COMPARING INPUT				CASCADING INPUT			OUTPUT		
				A>B	A<B	A=B	A>B	A<B	A=B
A3>B3	X	X	X	X	X	X	H	L	L
A3=B3	A2>B2	X	X	X	X	X	H	L	L
A3=B3	A2=B2	A1>B1	X	X	X	X	H	L	L
A3=B3	A2=B2	A1=B1	A0>B0	X	X	X	H	L	L
A3=B3 , A2=B2 , A1=B1 , A0=B0				L	L	L	H	H	L
				X	X	H	L	L	H
				L	H	L	L	H	L
				H	L	L	H	L	L
				H	H	L	L	L	L
A3=B3	A2=B2	A1=B1	A0<B0	X	X	X	L	H	L
A3=B3	A2=B2	A1<B1	X	X	X	X	L	H	L
A3=B3	A2<B2	X	X	X	X	X	L	H	L
A3<B3	X	X	X	X	X	X	L	H	L

X : Don't Care

図 2－12. 74HC85 の真理値表（参考文献 [5]）

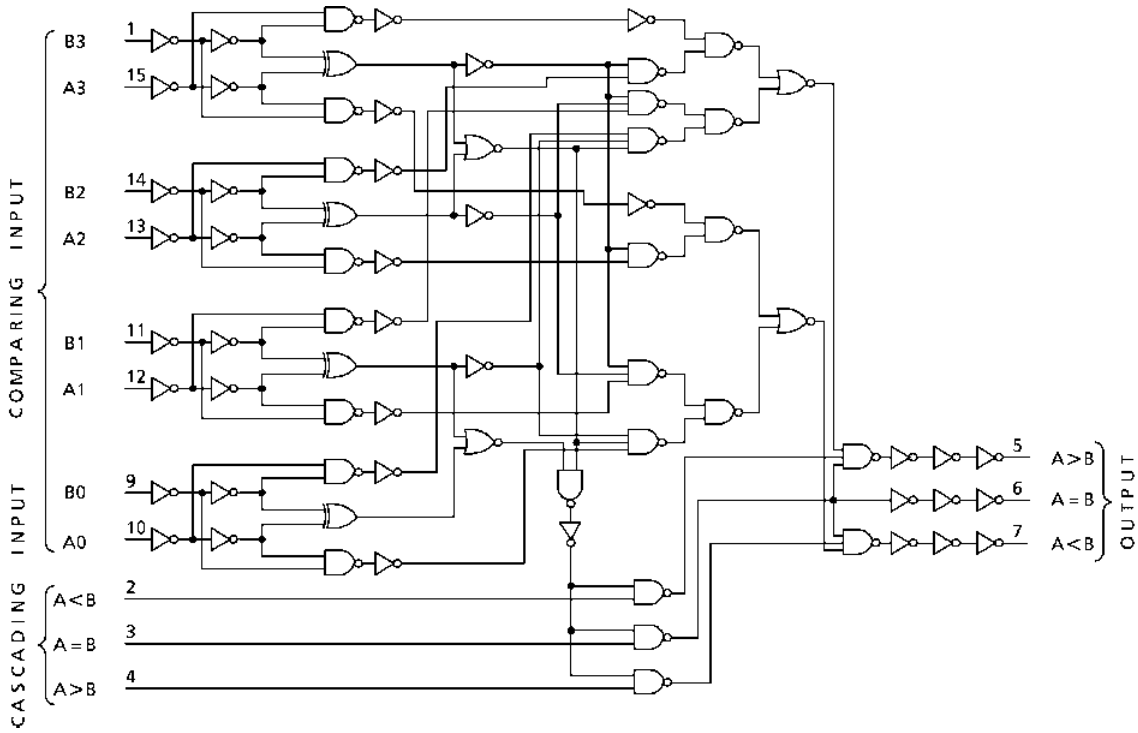


図 2－13. 74HC85 の内部構成回路（参考文献 [7]）

ストップウォッチ“分”“秒”の BCD 出力（7bit＋7bit）を A 入力、アラーム“分”“秒”の BCD 出力（7bit＋7bit）を B 入力とし、A=B 出力（一致出力）を得る。74HC85 の A=B IN、A=B OUT 端子を用いて 4 段構成とし、4 段目の A=B OUT をアラーム出力とする（図 2－14 参照）。

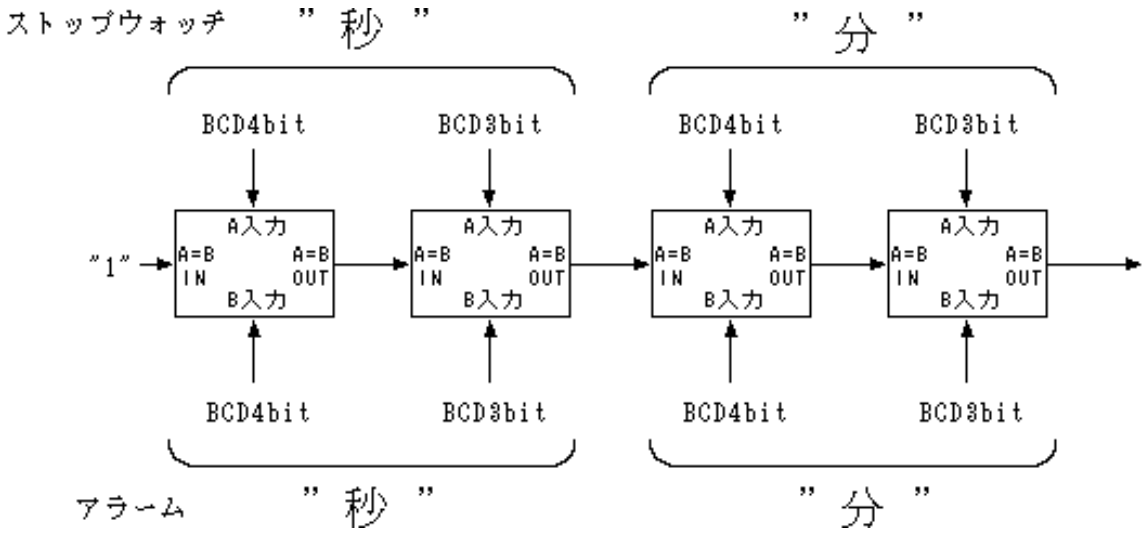
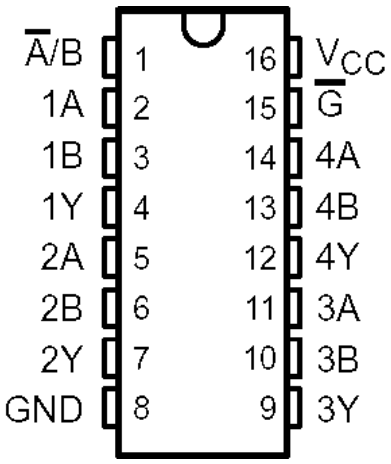


図 2－14. 比較回路 74HC85 の 4 段接続

2.2.6　切換回路

切換回路には 4bit Selector／multiplexor（74HC157）を 6 個使用する。ストップウォッチ“ 分 ”“ 秒 ”“ 1/100 秒 ” の BCD 出力（7bit＋7bit＋8bit）を A 入力、アラーム「分」「秒」の BCD 出力（7bit＋7bit）を B 入力とし、Select 端子に SW3 を繋ぎ、表示切替機能を制御する。74HC157 のピン配置と真理値表を図 2－15 に、内部構成回路を図 2－16 に示す。



FUNCTION TABLE

INPUTS				OUTPUT
$\overline{G}$	SELECT $\overline{A}$/B	DATA A	DATA B	Y
H	X	X	X	L
L	L	L	X	L
L	L	H	X	H
L	H	X	L	L
L	H	X	H	H

図 2－15．74HC157 のピン配置と真理値表（参考文献 [5]）

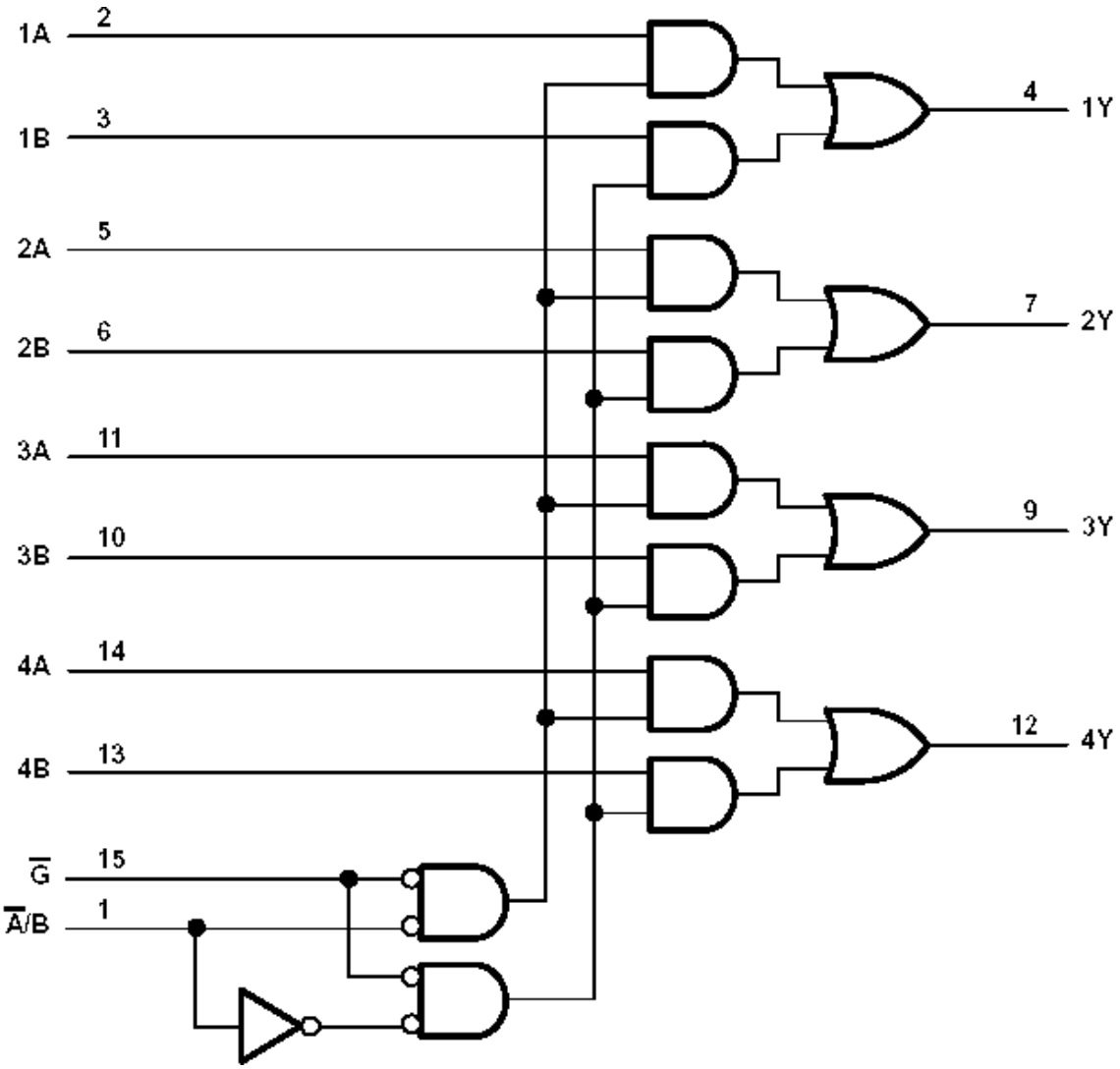
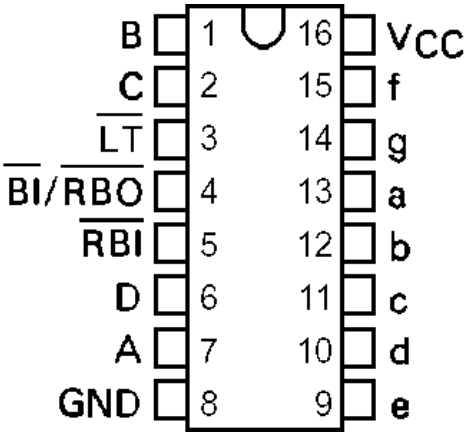
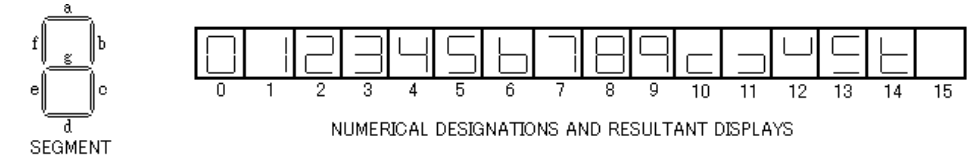


図 2－16. 74HC157 の内部構成回路（参考文献 [5]）

2.2.7　表示回路

Selector からの BCD 信号を 7 セグメント LED 用の信号に変換しなくてはならない。BCD to 7 Segment Decoder に、74LS47 を 6 個使用する。使用する 7 セグメント LED はアノードコモン(Hi で消灯、Low で点灯)であるから、active low 出力の 74LS47 を使用することにした。74LS47 のピン配置と真理値表を図 2－17 に、内部構成回路を図 2－18 に示す。





NUMERICAL DESIGNATIONS AND RESULTANT DISPLAYS

SEGMENT IDENTIFICATION

74LS47 FUNCTION TABLE

DECIMAL OR FUNCTION	INPUTS						BI/RBO	OUTPUTS						
	LT	RBI	D	C	B	A		a	b	c	d	e	f	g
0	H	H	L	L	L	L	H	ON	ON	ON	ON	ON	ON	OFF
1	H	X	L	L	L	H	H	OFF	ON	ON	OFF	OFF	OFF	OFF
2	H	X	L	L	H	L	H	ON	ON	OFF	ON	ON	OFF	ON
3	H	X	L	L	H	H	H	ON	ON	ON	ON	OFF	OFF	ON
4	H	X	L	H	L	L	H	OFF	ON	ON	OFF	OFF	ON	ON
5	H	X	L	H	L	H	H	ON	OFF	ON	ON	OFF	ON	ON
6	H	X	L	H	H	L	H	OFF	OFF	ON	ON	ON	ON	ON
7	H	X	L	H	H	H	H	ON	ON	ON	OFF	OFF	OFF	OFF
8	H	X	H	L	L	L	H	ON	ON	ON	ON	ON	ON	ON
9	H	X	H	L	L	H	H	ON	ON	ON	OFF	OFF	ON	ON
10	H	X	H	L	H	L	H	OFF	OFF	OFF	ON	ON	OFF	ON
11	H	X	H	L	H	H	H	OFF	OFF	ON	ON	OFF	OFF	ON
12	H	X	H	H	L	L	H	OFF	ON	OFF	OFF	OFF	ON	ON
13	H	X	H	H	L	H	H	ON	OFF	OFF	ON	OFF	ON	ON
14	H	X	H	H	H	L	H	OFF	OFF	OFF	ON	ON	ON	ON
15	H	X	H	H	H	H	H	OFF	OFF	OFF	OFF	OFF	OFF	OFF
BI	X	X	X	X	X	X	L	OFF	OFF	OFF	OFF	OFF	OFF	OFF
RBI	H	L	L	L	L	L	L	OFF	OFF	OFF	OFF	OFF	OFF	OFF
LT	L	X	X	X	X	X	H	ON	ON	ON	ON	ON	ON	ON

H = high level, L = low level, X = irerevant

図 2－17．74LS47 のピン配置、真理値表
（参考文献 [5]）

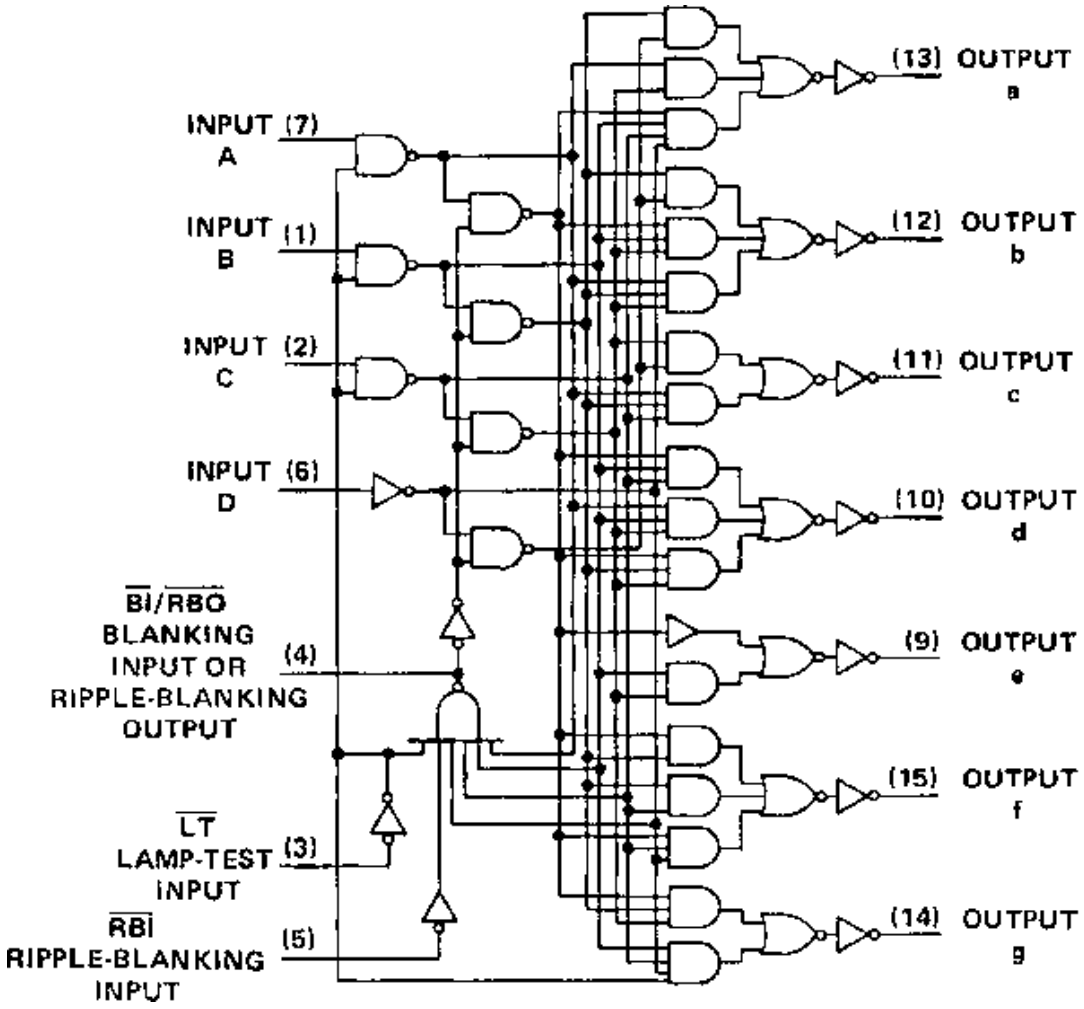


図 2－18．74LS47 の内部構成回路（参考文献 [5]）

2.2.8　チャタリング防止回路

設計するアラーム付ストップウォッチには 7 個のスイッチを使用している。各スイッチの機能は以下のとおりである。

SW1：ストップウォッチ start／stop
SW2：ストップウォッチ Reset
SW3：表示切替（ストップウォッチ／アラーム）
SW4：アラーム“秒”設定
SW5：アラーム“分”設定
SW6：アラーム出力 ON／OFF
SW7：アラーム Reset

ここでは 2 種類の SWITCH が必要である。1 つは 3 端子切換スイッチであり、もう 1 つはプッシュボタンスイッチである。この 2 つのスイッチには以下の特徴がある。

・3 端子切換スイッチ

レバーを ON／OFF することで、中央の端子と、その左右のどちらの端子を短絡させるか制御するスイッチ。

・プッシュボタンスイッチ

ピンが 4 本あるが、通常はお互い孤立している。ボタンをプッシュすることで、それぞれの対角線上のピンを短絡させるスイッチ。

以上の特徴を踏まえ、SW1、SW3、SW6 には 3 端子の切換スイッチを、SW2、SW4、SW5、SW7 にはプッシュボタンスイッチを用いる。

ここで、一つ注意すべき点がある。機械式接点を使用したスイッチ（プッシュスイッチなど）では、接触面の凹凸が原因で、スイッチを押した瞬間に接触面が幾度も接触・非接触を繰り返した後に、安定する（図 2－19 参照）。この現象をチャタリングという。チャタリングの継続時間は、スイッチの種類によって異なるが、長いもので数 10ms 発生する場合もある。

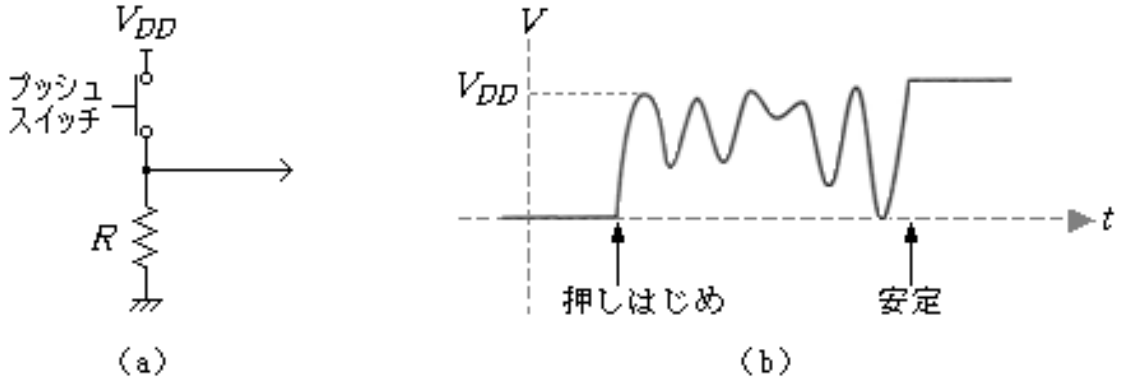


図 2－19.（a）スイッチ回路、（b）スイッチを押した瞬間の波形

切換スイッチである SW1、SW3、SW6 ではレバーによる切換が用途であるから、チャタリングを考慮する必要は無い。しかし、プッシュボタンスイッチである SW4、SW5 は、手動でカウントアップさせるための CLOCK 入力であるので、1 回押したら正確に 1CLOCK を与えるようにしなくてはならない。

図 2－20 にチャタリング防止回路の例を示す。

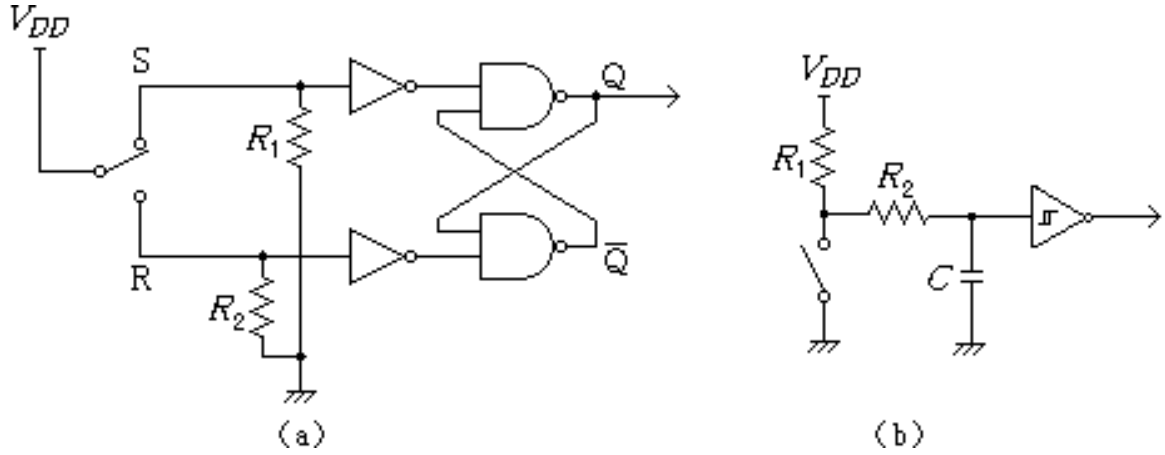


図 2－20．チャタリング防止回路

（a）RS－FF を用いた回路、（b）シュミットトリガインバータを用いた回路

図 2－20(a)は RS－FF を利用したチャタリング防止法である。RS－FF はセット端子 S に一度「1」が入力されれば、以降は S 端子に他の信号が入力されても出力 Q は「1」の状態を保持する。その後、リセット端子 R に「1」を入力すると、出力 Q は「0」になる（表 2－4 参照)。よって、この方法だと確実にチャタリングの影響を受けない。

表 2－4．RS－FF の真理値表

S	R	Q	$\overline{Q}$	動作
0	0	Q	$\overline{Q}$	保持
0	1	0	1	リセット
1	0	1	0	セット
1	1	不定		禁止

図 2－20(b)は、シュミットトリガゲートを利用した、チャタリング防止法である。スイッチを押し、チャタリングが発生した信号を、CR フィルタを通して平滑し、その後シュミットトリガゲートで波形整形を行なう。シュミットトリガゲートでは、通常のゲートと違って、入力電圧を上げていくときと下げていくときでスレッショルド電圧が異なる。つまり 2 つのスレッショルド電圧を持っているので、その中間値のノイズが乗ったとしても除去できる。このような特性をヒステリシスと呼ぶ。図 2－21 にヒステリシス特性による雑音除去の例を示す。

スレッショルド①は入力電圧を上げていくときのスレッショルド電圧、スレッショルド②は入力電圧を下げていくときのスレッショルド電圧である。雑音が混入した信号を一般のバッファに入力すると、出力波形は（b）のように、雑音まで Hi として出力してしまう。しかし、雑音が混入した信号をシュミットトリガへ入力すれば、（d）のように、シュミットトリガのヒステリシス特性が雑音の影響を吸収して、出力信号には入力と同じ２つのパルスが現れる。

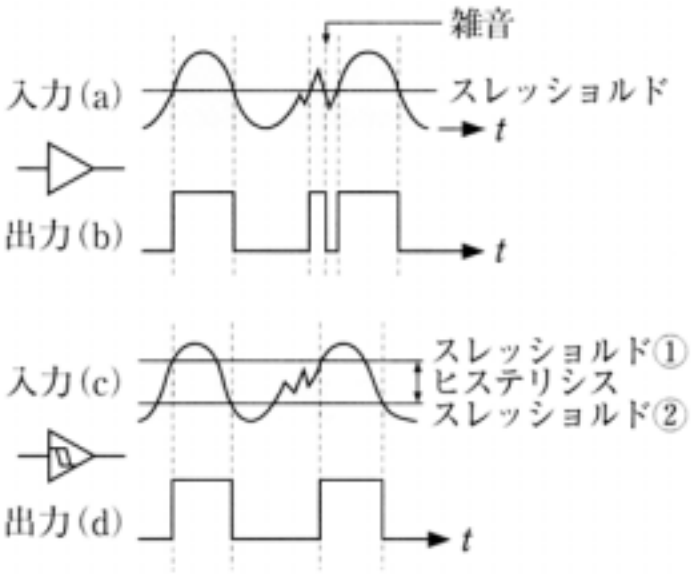


図 2－21. ヒステリシス特性による雑音除去

チャタリング防止法としては RS－FF を利用した方法が確実だが、今回は、SW4、SW5 がプッシュボタンスイッチであること、また IC の削減にもなることから、シュミットトリガ型インバータを使用してチャタリング防止回路を構成した（図 2－22（a））。この回路では、チャタリングの継続時間を 10ms 程度としてコンデンサ、抵抗の値を設定している。また、SW2、SW7 もプッシュボタンスイッチであるが、これは RESET ボタンとして使用するので、チャタリング防止は特に必要ない（図 2－22（b））。

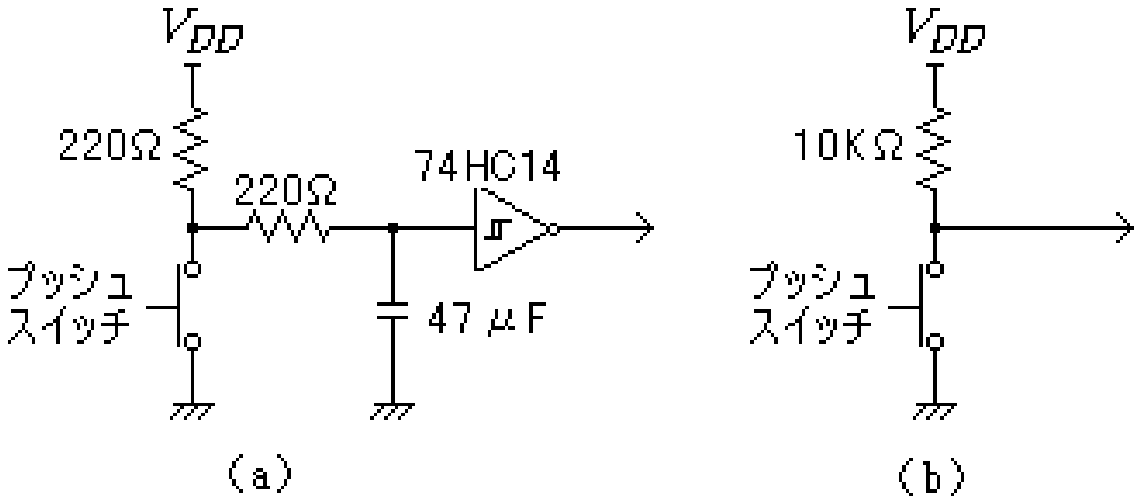
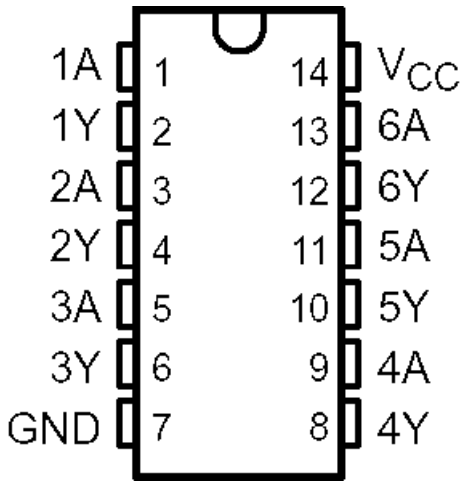


図 2－22. スイッチ回路 （a）チャタリング防止あり、（b）チャタリング防止なし

シュミットトリガ型インバータ用 IC には、内部にシュミットトリガ型インバータを 6 つ持つ 74HC14 を使用した。図 2－23 に 74HC14 のピン配置、真理値表、内部回路を示す。



FUNCTION TABLE
(each inverter)

INPUT A	OUTPUT Y
H	L
L	H

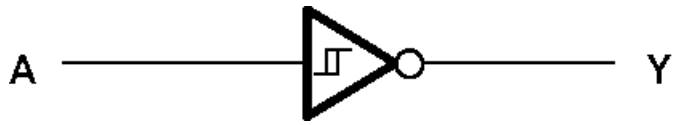


図 2－23. 74HC14 のピン配置、真理値表、内部回路（参考文献 [5]）

2.3　全体の設計

2.3.1　全体回路図

アラーム付きストップォッチの全体回路図を図 2－24 に示す。

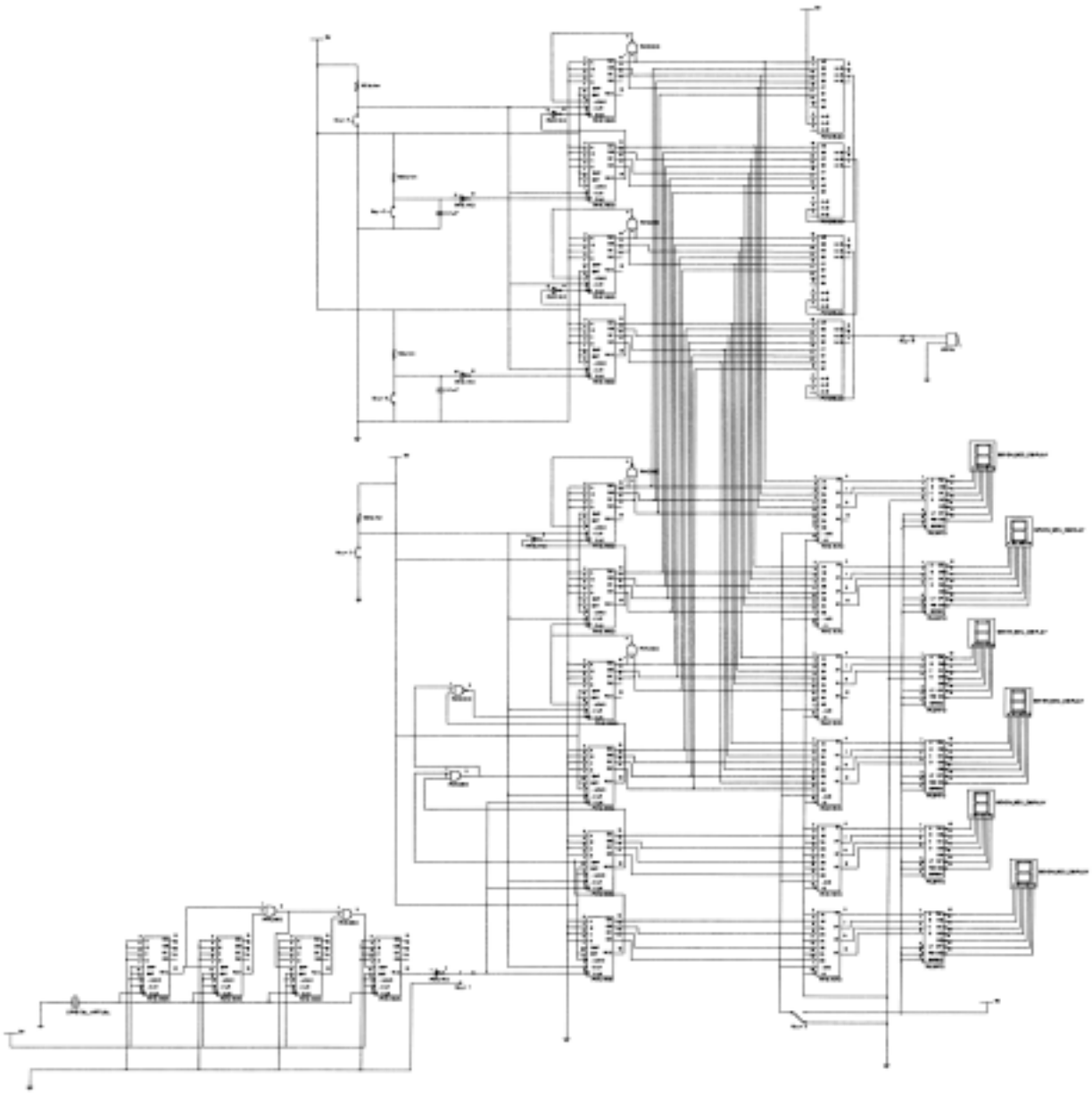
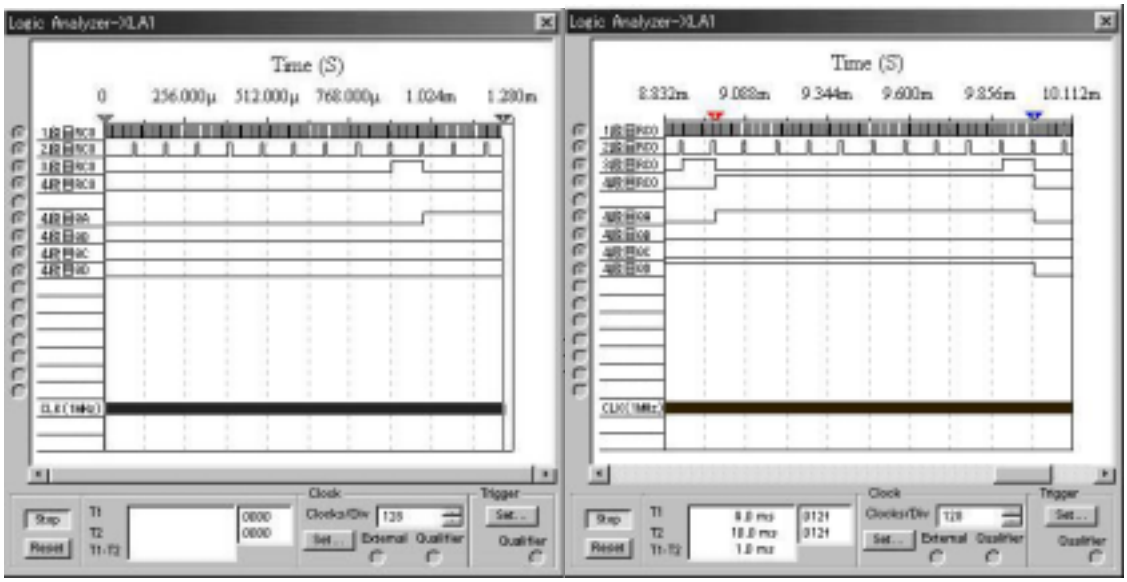


図 2－24.　全体回路図

2.3.2　シミュレーション

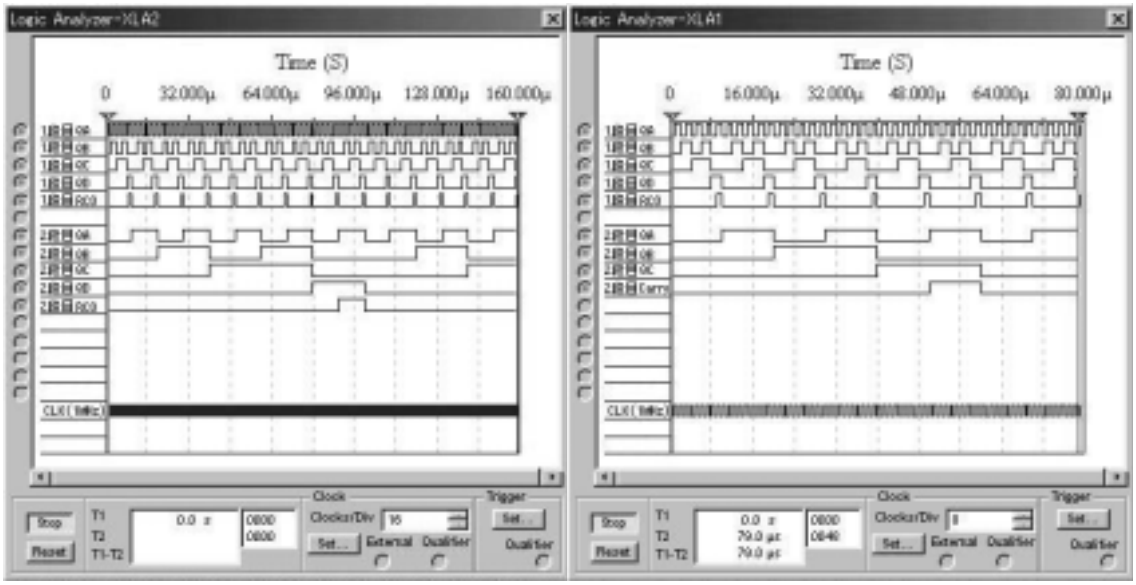
回路図全体のシミュレーションは、回路規模が大きいためシミュレーションに時間がかかるので、部分回路ごとのシミュレーションを行った。図 2－25 に分周回路のシミュレーション結果を示す。



(a)　　(b)

図 2－25. 分周回路部のシミュレーション結果

10 進カウンタ直列 4 段接続で構成する分周回路は、1MHｚの CLK 入力を 1/10000 分周し、4 段目の RCO から 100Hz で出力しなければならない。動作を確認するため最小表示しているので CLK はパルス 1 つ 1 つは目視できないが、図 2－25(a)では 2 段目、3 段目の RCO がそれぞれ前段の 10 倍周期で出力されていることが分かる。そして 3 段目 RCO によって 4 段目 QA が動作を始めている。最小化表示でも 1 画面では表示しきれないため、同シミュレーションの 9ms～10ms あたりを表示しているのが図 2－25(b)である。図 2－25(b)では 9ms に 4 段目 RCO が立ち上がり、10ms に立ち下がっていることが確認できる。つまり 10msec＝1/100Hz であるから、100Hz のパルスが出力されている。よって、分周回路は正常に動作していることが分かる。



（a）100 進カウンタ　　（b）60 進カウンタ

図 2－26．100 進、60 進カウンタのシミュレーション結果

図 2－26 に 100 進カウンタと 60 進カウンタのシミュレーション結果を示す。

シミュレーションの高速化、高精度化のために CLK は 1MHz でシミュレーションを行った。図 2－26（a）は、“ 1/100 秒 ” に使用している 100 進カウンタのシミュレーション結果である。1 段目の QA~QD が 0～9（0000～1001）までカウントし、9（1001）になると RCO をセットし、その立下りで 2 段目の QA が動作を始めている。2 段目も 0～9 までカウント、9 で RCO をセットし、正常に動作することが確認できる。

図 2－26（b）は “ 秒 ”“ 分 ” に使用している 60 進カウンタのシミュレーション結果である。2 段目の QD 出力は関係ないので省略している。1 段目の動作は（a）と同じだが、2 段目は 0～5（000～101）までカウントし、5（101）になると Carry をセットしている（この Carry は QA と QC の AND を取った桁上げ出力である）。60 進カウンタも正常に動作することが確認できた。

2.4　基板への実装

（1）部品表

設計結果をまとめると、使用する部品は表 2－5 のようになった。

表 2－5. 部品表

各部名称	部品名	仕様･型番	数量
発振器	水晶発振器 (1MHz)	HAT7400A　1MHz	1
分周回路、カウンタ (1/100 秒)	10 進カウンタ	HD74HC160D	6
	2 入力 AND	SN74HC08	1
カウンタ (“秒”、“分”)	10 進カウンタ	HD74HC160D	4
	6 進カウンタ (10 進を 5 でデコード)	HD74HC160D	4
	2 入力 NAND	SN74HC00D	1
比較回路	4bit comparator	TC74HC85AD	4
切換回路	4bit selector/Multiplexor	SN74HC157D	6
表示回路	BCD→7 セグメントデコーダ/ドライバ	SN74LS47	6
	7 セグメント LED (アノードコモン)	S GL9R10 8	6
	抵抗	330Ω	42
	抵抗(小数点用)	470Ω	1
スイッチ	切換スイッチ (SW1、3、6)		3
	プッシュスイッチ (SW2、4、5、7)		4
チャタリング防止回路	シュミットトリガ NOT	SN74HC14N	1
	抵抗	220Ω	4
	コンデンサ	47μF	2
アラーム、その他	アラームブザー		1
	LED (赤、緑、黄)		3
	抵抗	330Ω	3
	抵抗	10KΩ	2

（2）基板配置レイアウト

図 2－27 に基板配置レイアウト図を示す。今回は 155mm×230mm サイズのユニバーサル基板を 2 枚使用し、製作する。

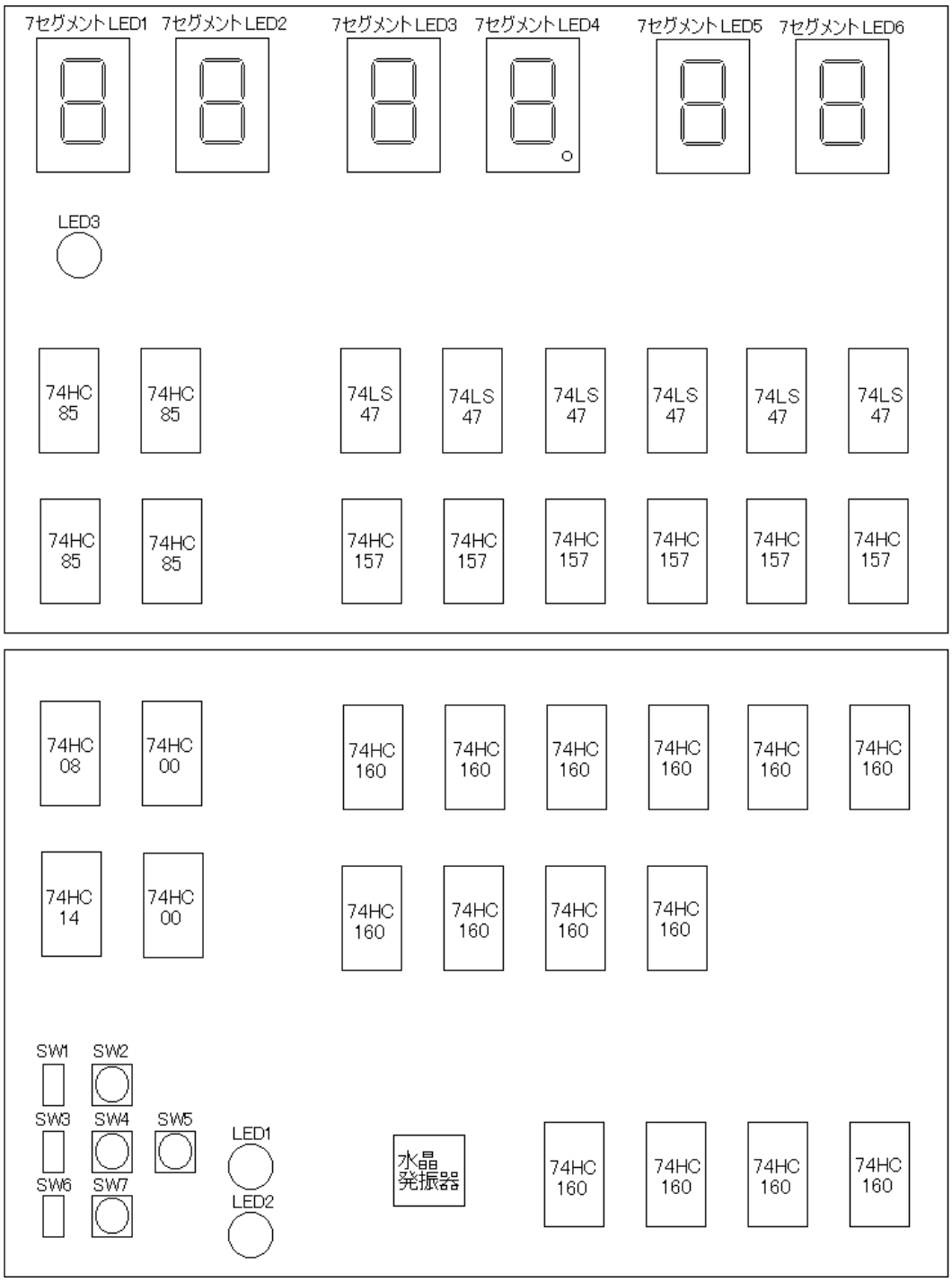


図 2－27．基板配置レイアウト

（3）基板実装写真

図 2－28 に製作したアラーム付きストップウォッチの写真を示す。

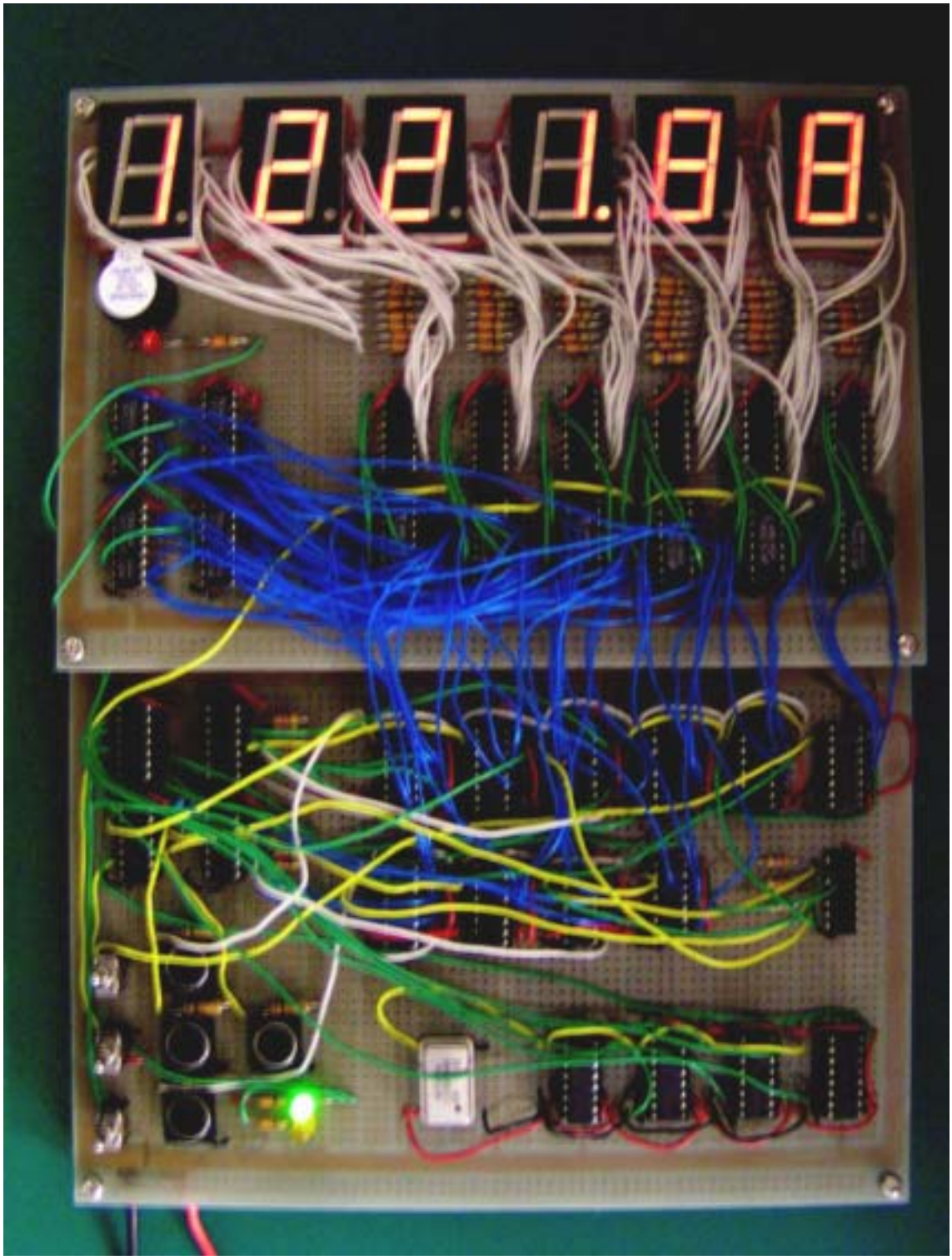


図 2－28. 汎用ロジック IC を用いたアラーム付きストップウォッチの写真

第 3 章　VHDL による回路設計

従来ディジタル回路の設計作業は人間の手作業によるものであった。しかし、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回路の規模は増大し、大規模なディジタルシステムの設計では、人間の手作業による回路図作成では限界がでてきた。そのため、現在こういった現場ではハードウェア記述言語（Hardware Discription Language）を用いたディジタル回路設計が行われるようになっている。

HDL は、従来のプログラム言語とは異なり、ハードウェアを設計する上で欠かすことの出来ない並列処理や時間、タイミングの概念などを記述できる特徴を持つ。HDL による設計手法は、すでに ASIC（特定用途向け集積回路）などの大規模集積回路の設計でさかんに利用されている。また、ASIC に限らず、FPGA や PLD などを使用した比較的小規模な設計にも様々なメリットをもたらしている。HDL によって、より抽象度の高いレベルで設計することが可能となり、難しい論理式を考える必要がなくなり、設計期間を短縮することができるようになった。また、抽象度の高い記述は、設計の変更を容易にし、より完成度の高いシステム構築が可能になった。

ハードウェア記述言語には複数の種類があるが、その中で業界標準としてもっとも広く普及しているのが VHDL である。

3.1　VHDL とは

VHDL（Very high speed integrated circuit Hardware Description Language）は米国国防省の VHSIC（Very High Speed Integrated Circuit）委員会で 1981 年に提唱され、開発が開始された。そして、1986 に IEEE（米国電気電子技術者協会）での標準化作業が始まり、1987 年 5 月には LPM（Language Reference Manual）が作成され、同年 12 月 IEEE Std 1076 として規格が認証され、世界的に標準化された。さらに 1993 年には規格の見直しが行われ、IEE Std 1164 となった。

VHDL はシステムからスイッチに至るハードウェア全般の記述を対象としており、ハードウェアの構造と動作の両方を記述できる。VHDL では回路を階層的にとらえ、ある回路は下位のより小さい回路が接続されたものとして記述する。最下層の回路を動作的に記述し、それより上位の回路を構造的に記述する。

VHDL は、ユーザ定義のタイプ、抽象度の高いデータタイプ、回路構成をコントロールするためのコンフィギュレーションなどに特徴がある。

3.2　ブロック図

図 3－1 にアラーム付ストップウォッチを VHDL によって設計を行う場合のブロック図を示す。

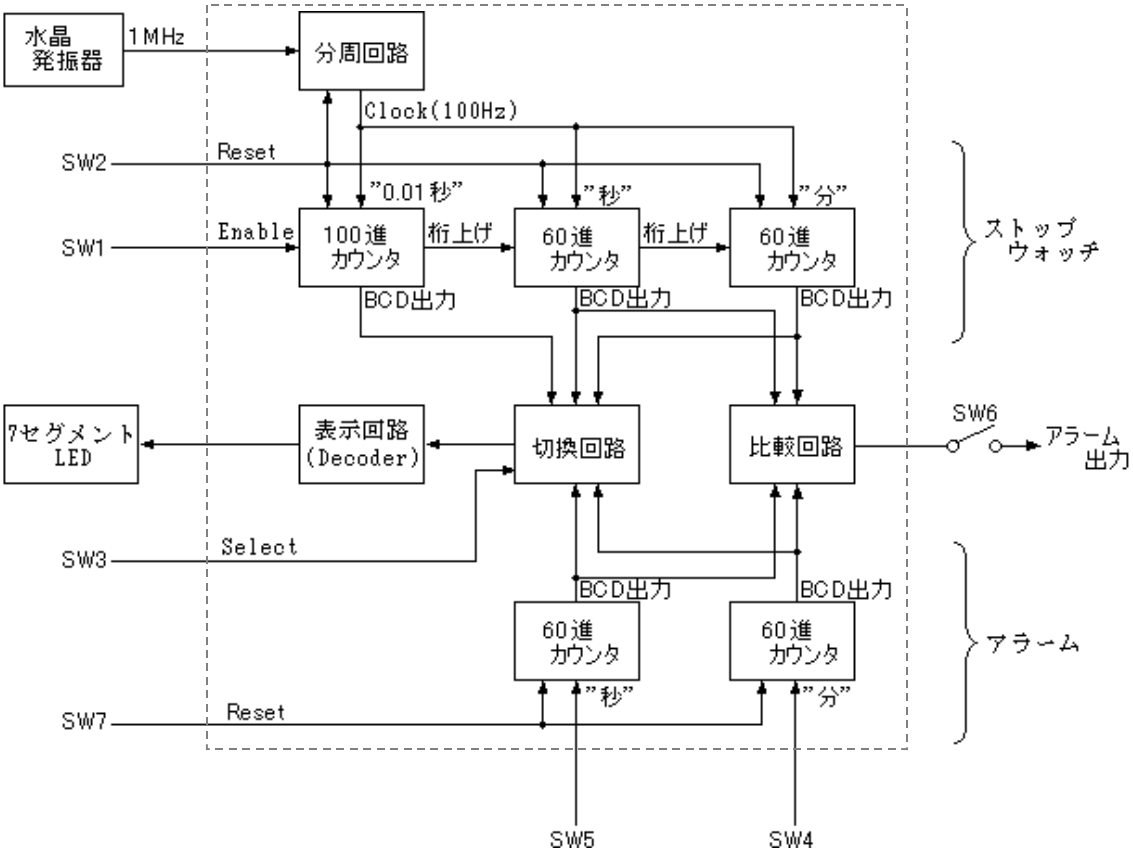


図 3－1. VHDL を用いたアラーム付きストップウォッチのブロック図

第 2 章と同様に、1MHz の水晶発振器をクロックとして使用する。必要な回路は第 2 章で行った IC を用いた設計と基本的に同じであるが、図 3－1 の点線内のディジタル回路を VHDL で設計・記述し、FPGA に実装してアラーム付ストップウォッチの回路を構成する。SWITCH（SW1～7）の機能は第 2 章で行った IC を用いた設計と同様である。

3.3　VHDL によるアラーム付ストップウォッチの設計

本節では、Electronics Workbench 社製「Multisim VHDL Power Professional」を使用して、VHDL による設計、シミュレーションを行う。本設計では、分周回路、表示回路を個別で設計、動作確認し、全体回路を記述する際に分周回路、表示回路を component 宣言にて呼び出し、使用する。

3.3.1　分周回路の設計

VHDL で設計する分周回路の仕様を表 3－1 に示す。

表 3－1．分周回路の仕様

信号名	入出力	Bit	説明
CLK0	IN	1	クロック信号
RESET	IN	1	内部カウンタの Reset
CLK1	OUT	1	出力信号（分周後のクロック信号）

分周回路では、1MHｚのクロック入力を 1/10000 に分周して 100Hz の信号を出力する。第 2 章と同様に 10 進カウンタを 4 段構成した分周回路を、VHDL で設計する。分周回路のソースコードをリスト 3－1 に、テストベンチをリスト 3－2 に、シミュレーション結果を図 3－2 に示す。なお、シミュレーションの高速化・高精度化のため、テストベンチではクロック周期を 1ns としている。

リスト 3－1．分周回路のソースコード 1/3

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

entity BUNSYU is
    port(   CLK0  : in std_logic;
            RESET: in std_logic;
            CLK1  : out std_logic
        );
end BUNSYU;
```

リスト 3−1. 分周回路のソースコード 2/3

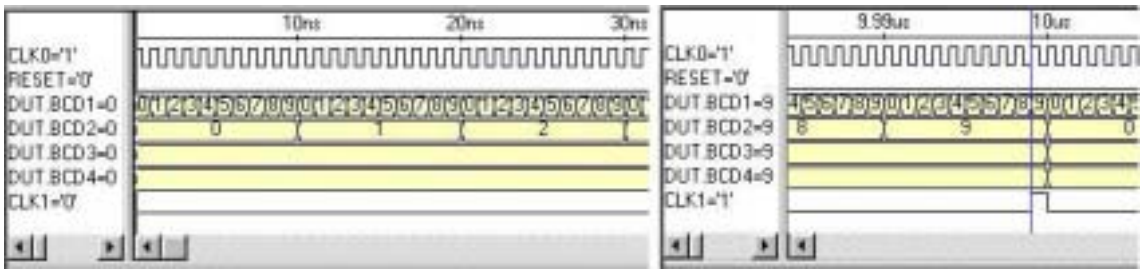
```
architecture RTL of BUNSYU is

signal BCD1 : std_logic_vector(3 downto 0);
signal BCD2 : std_logic_vector(3 downto 0);
signal BCD3 : std_logic_vector(3 downto 0);
signal BCD4 : std_logic_vector(3 downto 0);

begin
    P1:process(CLK0,RESET) begin
        if(RESET='1') then
            BCD1 <= "0000";
        elsif(CLK0' event and CLK0='1') then
            if(BCD1 = "1001") then
                BCD1 <= "0000";
            else
                BCD1 <= BCD1 + 1;
            end if;
        end if;
    end process P1;

    P2:process(CLK0,RESET) begin
        if(RESET='1') then
                BCD2 <= "0000";
        elsif(CLK0' event and CLK0='1') then
            if(BCD1="1001") then
                if(BCD2 = "1001") then
                    BCD2 <= "0000";
                else
                    BCD2 <= BCD2 + 1;
                end if;
            end if;
        end if;
    end process P2;

    P3:process(CLK0,RESET) begin
```

リスト 3－1. 分周回路のソースコード 3/3

```
        if(RESET='1') then
                BCD3 <= "0000";
        elsif(CLK0' event and CLK0='1') then
            if (BCD1="1001" and BCD2="1001") then
                if(BCD3 = "1001") then
                    BCD3 <= "0000";
                else
                    BCD3 <= BCD3 + 1;
                end if;
            end if;
        end if;
    end process P3;

    P4:process(CLK0,RESET) begin
        if(RESET='1') then
            BCD4 <= "0000";
        elsif(CLK0' event and CLK0='1') then
            if(BCD1="1001" and BCD2="1001" and BCD3="1001") then
                if(BCD4 = "1001") then
                    BCD4 <= "0000";
                else
                    BCD4 <= BCD4 + 1;
                end if;
            end if;
        end if;
    end process P4;

    P5:process(BCD1,BCD2,BCD3,BCD4) begin
        if(BCD1="1001" and BCD2="1001" and BCD3="1001" and
BCD4="1001") then
            CLK1 <= '1';
        else
            CLK1 <= '0';
        end if;
    end process P5;
end RTL;
```

リスト 3−2. 分周回路のテストベンチ

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

entity TEST_BUNSYU is
end TEST_BUNSYU;

architecture SIM of TEST_BUNSYU is

component BUNSYU
    port(   CLK0  : in std_logic;
            RESET: in std_logic;
            CLK1  : out std_logic
        );
end component;

signal CLK0   : std_logic;
signal RESET  : std_logic;
signal CLK1   : std_logic;

begin
    DUT: BUNSYU port map (    CLK0 => CLK0,
                              RESET => RESET,
                              CLK1 => CLK1
                          );

    SIM1:process begin
        CLK0 <= '1';        wait for 0.5 ns;
        CLK0 <= '0';        wait for 0.5 ns;
    end process SIM1;

    SIM2:process begin
        RESET <= '1';       wait for 0 ns;
        RESET <= '0';       wait;
    end process SIM2;
end SIM;
```



(a) (b)

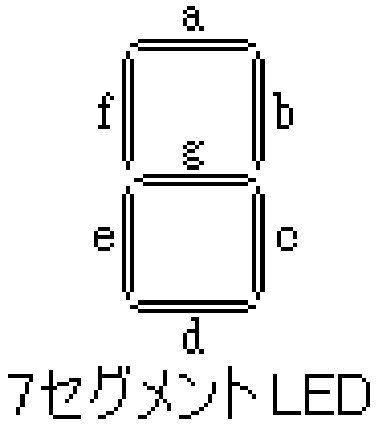
図 3－2. 分周回路のシミュレーション結果

図 3－2(a)はシミュレーション開始直後で、図 3－2(b)は CLK=‘ 1 ’になる時点のシミュレーション結果である。クロック周期 1ns のパルス（CLK0）を 10000 回分周し、10us 周期のパルス（CLK1）が出力されていることが確認できる。

3.3.2 表示回路（Decoder）の設計

0～9 の BCD 信号（4bit）を、７セグメント用の信号（7bit）に変換する回路が必要である。表 3－2 に BCDto7SegmentDecoder の真理値表を示す。

表 3－2．BCD to 7 Segment Decoder の真理値表



decimal	Binary	a	b	c	d	e	f	g
0	0000	1	1	1	1	1	1	0
1	0001	0	1	1	0	0	0	0
2	0010	1	1	0	1	1	0	1
3	0011	1	1	1	1	0	0	1
4	0100	0	1	1	0	0	1	1
5	0101	1	0	1	1	0	1	1
6	0110	1	0	1	1	1	1	1
7	0111	1	1	1	0	0	1	0
8	1000	1	1	1	1	1	1	1
9	1001	1	1	1	1	0	1	1
A ・ ・ ・ F	1010 ・ ・ ・ 1111	0	0	0	0	0	0	0

本来 BCD 信号は 0～9 のみだが、Decoder 部の記述に使用した case 文では、全ての場合を記述する必要があるため、0～9 以外の信号が入った場合は、全て消灯させることとする。

“ 1/100 秒 ”“ 秒 ”“ 分 ”の BCD 出力には、それぞれ十の位(4bit)、一の位(4bit)があるので、2 桁分を扱う Decoder を設計する。表 3－3 に 2 桁分の BCD to 7 Segment Decoder の仕様を示す。

表 3－3．BCD to 7 Segment Decoder の仕様

信号名	入出力	Bit	説明
BCD	IN	8	上位 4bit：十の位の BCD 入力 下位 4bit：一の位の BCD 入力
SEG1	OUT	7	7 セグメント出力（一の位）
SEG10	OUT	7	7 セグメント出力（十の位）

VHDL によって設計した BCD to 7 Segment Decoder のソースコードをリスト 3－3 に、テストベンチをリスト 3－4 に、図 3－3 にシミュレーション結果を示す。

リスト 3−3. BCD to 7 Segment Decoder のソースコード 1/2

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

entity BCDto7segDecoder is
    port(   BCD    : in std_logic_vector(7 downto 0);
            SEG1   : out std_logic_vector(6 downto 0);
            SEG10 : out std_logic_vector(6 downto 0)
            );
end BCDto7segDecoder;

architecture RTL of BCDto7segDecoder is
signal INDATA1   : std_logic_vector(3 downto 0);
signal INDATA10 : std_logic_vector(3 downto 0);

begin
    INDATA1  <=  BCD(3 downto 0);
    INDATA10<=  BCD(7 downto 4);

    Decoder1:process(INDATA1) begin
            case INDATA1 is
                when "0000" => SEG1 <= "1111110";  --0
                when "0001" => SEG1 <= "0110000";  --1
                when "0010" => SEG1 <= "1101101";  --2
                when "0011" => SEG1 <= "1111001";  --3
                when "0100" => SEG1 <= "0110011";  --4
                when "0101" => SEG1 <= "1011011";  --5
                when "0110" => SEG1 <= "1011111";  --6
                when "0111" => SEG1 <= "1110010";  --7
                when "1000" => SEG1 <= "1111111";  --8
                when "1001" => SEG1 <= "1111011";  --9
                when others => SEG1 <= "0000000";  --X
            end case;
    end process Decoder1;
```

リスト 3−3. BCD to 7 Segment Decoder のソースコード 2/2

```
    Decoder10:process(INDATA10) begin
            case INDATA10 is
                when "0000" => SEG10 <= "1111110";     --0
                when "0001" => SEG10 <= "0110000";     --1
                when "0010" => SEG10 <= "1101101";     --2
                when "0011" => SEG10 <= "1111001";     --3
                when "0100" => SEG10 <= "0110011";     --4
                when "0101" => SEG10 <= "1011011";     --5
                when "0110" => SEG10 <= "1011111";     --6
                when "0111" => SEG10 <= "1110010";     --7
                when "1000" => SEG10 <= "1111111";     --8
                when "1001" => SEG10 <= "1111011";     --9
                when others => SEG10 <= "0000000";--X
            end case;
    end process Decoder10;
end RTL;
```

リスト 3−4. BCD to 7 Segment Decoder のテストベンチ 1/3

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

entity TEST_BCDto7segDecoder is
end TEST_BCDto7segDecoder;

architecture SIM of TEST_BCDto7segDecoder is
component BCDto7segDecoder
    port(       BCD    : in std_logic_vector(7 downto 0);
                SEG1   : out std_logic_vector(6 downto 0);
                SEG10 : out std_logic_vector(6 downto 0)
            );
end component;

signal BCD      : std_logic_vector(7 downto 0);
```

リスト 3−4. BCD to 7 Segment Decoder のテストベンチ 2/3

```
signal SEG1   : std_logic_vector(6 downto 0);
signal SEG10  : std_logic_vector(6 downto 0);

begin
        DUT: BCDto7segDecoder port map(   BCD => BCD,
                                          SEG1 => SEG1,
                                          SEG10 => SEG10
                                      );
    S1:process begin
                BCD <= "00000000";   wait for 10ns;  --00
                BCD <= "00000001";   wait for 10ns;  --01
                BCD <= "00000010";   wait for 10ns;  --02
                BCD <= "00000011";   wait for 10ns;  --03
                BCD <= "00000100";   wait for 10ns;  --04
                BCD <= "00000101";   wait for 10ns;  --05
                BCD <= "00000110";   wait for 10ns;  --06
                BCD <= "00000111";   wait for 10ns;  --07
                BCD <= "00001000";   wait for 10ns;  --08
                BCD <= "00001001";   wait for 10ns;  --09
                BCD <= "00001010";   wait for 10ns;  --0A
                BCD <= "00001011";   wait for 10ns;  --0B
                BCD <= "00001100";   wait for 10ns;  --0C
                BCD <= "00001101";   wait for 10ns;  --0D
                BCD <= "00001110";   wait for 10ns;  --0E
                BCD <= "00001111";   wait for 10ns;  --0F
                BCD <= "00010000";   wait for 10ns;  --10
                BCD <= "00100000";   wait for 10ns;  --20
                BCD <= "00110000";   wait for 10ns;  --30
                BCD <= "01000000";   wait for 10ns;  --40
                BCD <= "01010000";   wait for 10ns;  --50
                BCD <= "01100000";   wait for 10ns;  --60
                BCD <= "01110000";   wait for 10ns;  --70
                BCD <= "10000000";   wait for 10ns;  --80
                BCD <= "10010000";   wait for 10ns;  --90
                BCD <= "10100000";   wait for 10ns;  --A0
                BCD <= "10110000";   wait for 10ns;  --B0
```

リスト 3－4. BCD to 7 Segment Decoder のテストベンチ 3/3

```
                BCD <= "11000000";   wait for 10ns;  --C0
                BCD <= "11010000";   wait for 10ns;  --D0
                BCD <= "11100000";   wait for 10ns;  --E0
                BCD <= "11110000";   wait for 10ns;  --F0
                wait;
    end process S1;
end SIM;
```

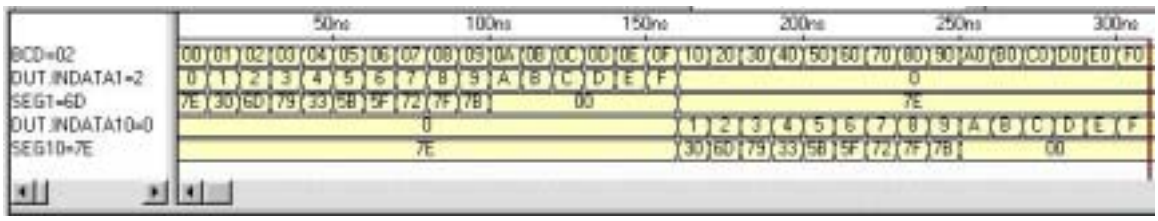


図 3－3. BCD to 7 Segment Decoder のシミュレーション結果

入力信号の BCD（8bit）の一の位を内部信号 INDATA1（4bit）に,十の位を INDATA10（4bit）に入れ、それに伴って SEG1、SEG10 が変化していることが確認できる。

3.3.3　全体回路の設計

アラーム付ストップウォッチの全体回路の仕様を表 3－4 に示す。

表 3－4. 全体回路の仕様

信号名	入出力	Bit	説明
CLK	IN	1	クロック信号（1MHz）
RESET_SW	IN	1	ストップウォッチ用 Reset 信号
SW1	IN	1	スタート／ストップ制御信号
A_CLK1	IN	1	アラーム“秒”用クロック
A_CLK2	IN	1	アラーム“分”用クロック
RESET_ALARM	IN	1	アラーム用 Reset 信号
SEL	IN	1	切換回路の Select 信号
ALARMOUT	OUT	1	アラーム出力
SEG_OUT1	OUT	7	7 セグメント出力（“1/100 秒”一の位）
SEG_OUT2	OUT	7	7 セグメント出力（“1/100 秒”十の位）
SEG_OUT3	OUT	7	7 セグメント出力（“秒”一の位）
SEG_OUT4	OUT	7	7 セグメント出力（“秒”十の位）
SEG_OUT5	OUT	7	7 セグメント出力（“分”一の位）
SEG_OUT6	OUT	7	7 セグメント出力（“分”十の位）

全体回路では、ストップウォッチ用のカウンタ、アラーム用のカウンタ、Selector、Comparator をメインで記述し、前節で設計した分周回路、表示回路を component 宣言にて呼び出し、使用している。全体回路のソースコードをリスト 3－5 に、テストベンチをリスト 3－6 に示す。

リスト 3−5. 全体回路のソースコード 1/9

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

entity Total is
    port(   CLK,SW1,RESET_SW      : in std_logic;
            A_CLK1,A_CLK2,RESET_ALARM    : in std_logic;
            SEL             : in std_logic;
            ALARMOUT : out std_logic;
            SEG_OUT1 : out std_logic_vector(6 downto 0);
            SEG_OUT2 : out std_logic_vector(6 downto 0);
            SEG_OUT3 : out std_logic_vector(6 downto 0);
            SEG_OUT4 : out std_logic_vector(6 downto 0);
            SEG_OUT5 : out std_logic_vector(6 downto 0);
            SEG_OUT6 : out std_logic_vector(6 downto 0)
            );
end Total;

architecture RTL of Total is

component BUNSYU
    port(   CLK0   : in std_logic;
            RESET: in std_logic;
            CLK1   : out std_logic
            );
end component;

component BCDto7segDecoder
    port(   BCD    : in std_logic_vector(7 downto 0);
            SEG1   : out std_logic_vector(6 downto 0);
            SEG10 : out std_logic_vector(6 downto 0)
            );
end component;

signal CLK_SW    : std_logic;
signal BCD_a1     : std_logic_vector(3 downto 0);
```

リスト 3−5. 全体回路のソースコード 2/9

```
signal BCD_a10    : std_logic_vector(3 downto 0);
signal BCD_b1     : std_logic_vector(3 downto 0);
signal BCD_b10    : std_logic_vector(2 downto 0);
signal BCD_c1     : std_logic_vector(3 downto 0);
signal BCD_c10    : std_logic_vector(2 downto 0);
signal CIN_b      : std_logic;
signal CIN_c      : std_logic;
signal BCD_d1     : std_logic_vector(3 downto 0);
signal BCD_d10    : std_logic_vector(2 downto 0);
signal BCD_e1     : std_logic_vector(3 downto 0);
signal BCD_e10    : std_logic_vector(2 downto 0);
signal BCD_IN     : std_logic_vector(23 downto 0);

begin
-------------------------------------------------------------------------------------------------
--bunsyu
-------------------------------------------------------------------------------------------------
    P0: BUNSYU port map(
                        CLK0 => CLK,
                        RESET => RESET_SW,
                        CLK1 => CLK_SW
                        );

-------------------------------------------------------------------------------------------------
--0.01second 7SegDecoder
-------------------------------------------------------------------------------------------------
    P1: BCDto7segDecoder port map(
                        BCD(7 downto 0) => BCD_IN(7 downto 0),
                        SEG1(6 downto 0) => SEG_OUT1(6 downto 0),
                        SEG10(6 downto 0) => SEG_OUT2(6 downto 0)
                        );

-------------------------------------------------------------------------------------------------
--second 7SegDecoder
-------------------------------------------------------------------------------------------------
    P2: BCDto7segDecoder port map(
```

```
                        BCD(7 downto 0) => BCD_IN(15 downto 8),
                        SEG1(6 downto 0) => SEG_OUT3(6 downto 0),
                        SEG10(6 downto 0) => SEG_OUT4(6 downto 0)
                        );


----------------------------------------------------------------------------------------------------
---minit 7SegDecoder
----------------------------------------------------------------------------------------------------
    P3: BCDto7segDecoder port map(
                        BCD(7 downto 0) => BCD_IN(23 downto 16),
                        SEG1(6 downto 0) => SEG_OUT5(6 downto 0),
                        SEG10(6 downto 0) => SEG_OUT6(6 downto 0)
                        );


----------------------------------------------------------------------------------------------------
--Stopwatch_0.01second
----------------------------------------------------------------------------------------------------
    a1:process(CLK_SW,RESET_SW) begin
        if(RESET_SW='1') then
                BCD_a1 <= "0000";
        elsif(CLK_SW' event and CLK_SW='1') then
                if(SW1='1')then
                    if(BCD_a1 = "1001") then
                        BCD_a1 <= "0000";
                    else
                        BCD_a1 <= BCD_a1 + 1;
                    end if;
                end if;
        end if;
    end process a1;

    a2:process(CLK_SW,RESET_SW) begin
        if(RESET_SW='1') then
                BCD_a10 <= "0000";
        elsif(CLK_SW' event and CLK_SW='1') then
                if(SW1='1' and BCD_a1="1001") then
```

```
                    if(BCD_a10="1001") then
                        BCD_a10 <= "0000";
                    else
                        BCD_a10 <= BCD_a10 + 1;
                    end if;
                end if;
            end if;
        end process a2;

        a3:process(SW1,BCD_a1,BCD_a10) begin
            if(SW1='1' and BCD_a1="1001" and BCD_a10="1001") then
                    CIN_b <= '1';
            else
                    CIN_b <= '0';
            end if;
        end process a3;


-------------------------------------------------------------------------------------------------
--Stopwatch_second
-------------------------------------------------------------------------------------------------
        b1:process(CLK_SW,RESET_SW) begin
            if(RESET_SW='1') then
                    BCD_b1 <= "0000";
            elsif(CLK_SW' event and CLK_SW='1') then
                    if (CIN_b = '1') then
                        if(BCD_b1 = "1001") then
                            BCD_b1 <= "0000";
                        else
                            BCD_b1 <= BCD_b1 + 1;
                        end if;
                    end if;
            end if;
        end process b1;

        b2:process(CLK_SW,RESET_SW) begin
            if(RESET_SW='1') then
```

```
                BCD_b10 <= "000";
        elsif(CLK_SW' event and CLK_SW='1') then
                if (CIN_b='1' and BCD_b1="1001") then
                    if(BCD_b10 = "101") then
                        BCD_b10 <= "000";
                    else
                        BCD_b10 <= BCD_b10 + 1;
                    end if;
                end if;
        end if;
    end process b2;

    b3:process(CIN_b,BCD_b1,BCD_b10) begin
        if(CIN_b='1' and BCD_b1="1001" and BCD_b10="101") then
                    CIN_c <= '1';
        else
                    CIN_c <= '0';
        end if;
    end process b3;


----------------------------------------------------------------------------------------------------
--Stopwatch_minutes
----------------------------------------------------------------------------------------------------
    c1:process(CLK_SW,RESET_SW) begin
        if(RESET_SW='1') then
                BCD_c1 <= "0000";
        elsif(CLK_SW' event and CLK_SW='1') then
                if (CIN_c = '1') then
                    if(BCD_c1 = "1001") then
                        BCD_c1 <= "0000";
                    else
                        BCD_c1 <= BCD_c1 + 1;
                    end if;
                end if;
        end if;
    end process c1;
```

```
    c2:process(CLK_SW,RESET_SW) begin
        if(RESET_SW='1') then
                BCD_c10 <= "000";
        elsif(CLK_SW' event and CLK_SW='1') then
                if (CIN_c='1' and BCD_c1="1001") then
                    if(BCD_c10 = "101") then
                        BCD_c10 <= "000";
                    else
                        BCD_c10 <= BCD_c10 + 1;
                    end if;
                end if;
        end if;
    end process c2;

-------------------------------------------------------------------------------------------------
--ALARM_Second
-------------------------------------------------------------------------------------------------
     d1:process(RESET_ALARM,A_CLK1) begin
        if(RESET_ALARM='1') then
                BCD_d1 <= "0000";
        elsif(A_CLK1' event and A_CLK1='1') then
                    if(BCD_d1 = "1001") then
                        BCD_d1 <= "0000";
                    else
                        BCD_d1 <= BCD_d1 + 1;
                    end if;
        end if;
    end process d1;

    d2:process(RESET_ALARM,A_CLK1) begin
        if(RESET_ALARM='1') then
                BCD_d10 <= "000";
        elsif(A_CLK1'event and A_CLK1='1') then
                if (BCD_d1="1001") then
                    if(BCD_d10 = "101") then
```

リスト 3−5. 全体回路のソースコード 7/9

```
                    BCD_d10 <= "000";
                else
                    BCD_d10 <= BCD_d10 + 1;
                end if;
            end if;
        end if;
    end process d2;

----------------------------------------------------------------------------------------------------
--ALARM_minutes
----------------------------------------------------------------------------------------------------
    e1:process(RESET_ALARM,A_CLK2) begin
        if(RESET_ALARM='1') then
                BCD_e1 <= "0000" ;
        elsif(A_CLK2'event and A_CLK2='1') then
                if(BCD_e1 = "1001") then
                    BCD_e1 <= "0000";
                else
                    BCD_e1 <= BCD_e1 + 1;
                end if;
        end if;
    end process e1;

    e2:process(RESET_ALARM,A_CLK2) begin
        if(RESET_ALARM='1') then
                BCD_e10 <= "000";
        elsif(A_CLK2'event and A_CLK2='1') then
                if (BCD_e1="1001") then
                    if(BCD_e10 = "101") then
                        BCD_e10 <= "000";
                    else
                        BCD_e10 <= BCD_e10 + 1;
                    end if;
                end if;
        end if;
    end process e2;
```

リスト 3−5. 全体回路のソースコード 8/9

```
----------------------------------------------------------------------------------------------------
--COMPARATOR
----------------------------------------------------------------------------------------------------
    COMPARATOR:process(BCD_b1,BCD_b10,BCD_c1,BCD_c10,BCD_d1,BCD_d10,BCD_e1,BCD_e10) begin
            if(BCD_b1=BCD_d1 and BCD_b10=BCD_d10 and BCD_c1=BCD_e1 and BCD_c10=BCD_e10) then
                ALARMOUT <= '1';
            else
                ALARMOUT <= '0';
            end if;
    end process COMPARATOR;

----------------------------------------------------------------------------------------------------
--SELECTOR
----------------------------------------------------------------------------------------------------
    SELECTOR:process(SEL,BCD_a1,BCD_a10,BCD_b1,BCD_b10,BCD_c1,BCD_c10,BCD_d1,BCD_d10,BCD_e1,BCD_e10) begin
            if(SEL='1') then
                    BCD_IN(3 downto 0) <= BCD_a1(3 downto 0);
                    BCD_IN(7 downto 4) <= BCD_a10(3 downto 0);
                    BCD_IN(11 downto 8) <= BCD_b1(3 downto 0);
                    BCD_IN(14 downto 12) <= BCD_b10(2 downto 0);
                    BCD_IN(15) <= '0';
                    BCD_IN(19 downto 16) <= BCD_c1(3 downto 0);
                    BCD_IN(22 downto 20) <= BCD_c10(2 downto 0);
                    BCD_IN(23) <= '0';
            else
                    BCD_IN(3 downto 0) <= "0000";
                    BCD_IN(7 downto 4) <= "0000";
                    BCD_IN(11 downto 8) <= BCD_d1(3 downto 0);
                    BCD_IN(14 downto 12) <= BCD_d10(2 downto 0);
                    BCD_IN(15) <= '0';
                    BCD_IN(19 downto 16) <= BCD_e1(3 downto 0);
                    BCD_IN(22 downto 20) <= BCD_e10(2 downto 0);
```

リスト 3－5. 全体回路のソースコード 9/9

```
                    BCD_IN(23) <= '0';
            end if;
    end process SELECTOR;
end RTL;
```

リスト 3－6. 全体回路のテストベンチ 1/3

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

entity TEST_TOTAL is
end TEST_TOTAL;

architecture SIM of TEST_TOTAL is
component TOTAL
    port(   CLK,SW1,RESET_SW     : in std_logic;
            A_CLK1,A_CLK2,RESET_ALARM    : in std_logic;
            SEL             : in std_logic;
            ALARMOUT    : out std_logic;
            SEG_OUT1    : out std_logic_vector(6 downto 0);
            SEG_OUT2    : out std_logic_vector(6 downto 0);
            SEG_OUT3    : out std_logic_vector(6 downto 0);
            SEG_OUT4    : out std_logic_vector(6 downto 0);
            SEG_OUT5    : out std_logic_vector(6 downto 0);
            SEG_OUT6    : out std_logic_vector(6 downto 0)
            );
end component;

signal CLK,SW1,RESET_SW : std_logic;
signal A_CLK1,A_CLK2,RESET_ALARM         : std_logic;
signal SEL              : std_logic;
signal ALARMOUT     : std_logic;
signal SEG_OUT1     : std_logic_vector(6 downto 0);
signal SEG_OUT2     : std_logic_vector(6 downto 0);
```

リスト 3－6. 全体回路のテストベンチ 2/3

```
signal SEG_OUT3     : std_logic_vector(6 downto 0);
signal SEG_OUT4     : std_logic_vector(6 downto 0);
signal SEG_OUT5     : std_logic_vector(6 downto 0);
signal SEG_OUT6     : std_logic_vector(6 downto 0);


begin
    DUT: TOTAL port map (
                        CLK => CLK,
                        SW1 => SW1,
                        RESET_SW => RESET_SW,
                        A_CLK1 => A_CLK1,
                        A_CLK2 => A_CLK2,
                        RESET_ALARM => RESET_ALARM,
                        SEL => SEL,
                        ALARMOUT => ALARMOUT,
                        SEG_OUT1 => SEG_OUT1,
                        SEG_OUT2 => SEG_OUT2,
                        SEG_OUT3 => SEG_OUT3,
                        SEG_OUT4 => SEG_OUT4,
                        SEG_OUT5 => SEG_OUT5,
                        SEG_OUT6 => SEG_OUT6
                       );

    P0: process begin                   --CLK
        CLK <= '0';     wait for 5 ns;
        CLK <= '1';     wait for 5 ns;
    end process P0;

    P1: process begin                   --START/STOP SWICH
        SW1 <= '1';     wait for 50 ns;
        SW1 <= '0';     wait for 50 ns;
        SW1 <= '1';     wait;
    end process P1;

    P2: process begin                   --RESET SWICH
```

リスト 3−6. 全体回路のテストベンチ 3/3

```
        RESET_SW <= '1'; wait for 0 ns;
        RESET_SW <= '0'; wait for 92 ns;
        RESET_SW <= '1'; wait for 1 ns;
        RESET_SW <= '0'; wait;
    end process P2;

    P3: process begin                    --ALARM CLK1
        A_CLK1 <= '0';     wait for 10 ns;
        A_CLK1 <= '1';     wait for 10 ns;
    end process P3;

    P4: process begin                    --ALARM CLK2
        A_CLK2 <= '0';     wait for 30 ns;
        A_CLK2 <= '1';     wait for 30 ns;
    end process P4;

    P5: process begin                    --RESET ALARM
        RESET_ALARM <= '1';    wait for 0 ns;
        RESET_ALARM <= '0';    wait for 92 ns;
        RESET_ALARM <= '1';    wait for 1 ns;
        RESET_ALARM <= '0';    wait;
    end process P5;

    P6: process begin                    --SELECT SW
        SEL <= '1';      wait for 50 ns;
        SEL <= '0';      wait for 50 ns;
    end process P6;

end SIM;
```

全体回路は回路規模が大きく、そのままではシミュレーションに時間がかかるため、全体回路のシミュレーションは、CLK を分周回路に通さず、CLK_SW に直接接続して行った（分周回路のシミュレーション結果は　　参照)。

具体的には、リスト 5−5. 全体回路のソースコード 5／9 内にある分周回路の接続記述を、コメント文にして VHDL として処理されないよう無効にし、さらに、CLK を内部信号 CLK_SW に直接接続するため

```
CLK_SW  <= CLK
```

という記述を加えたソースを用い、シミュレーションを行った。シミュレーション結果を図 3−4 に示す。

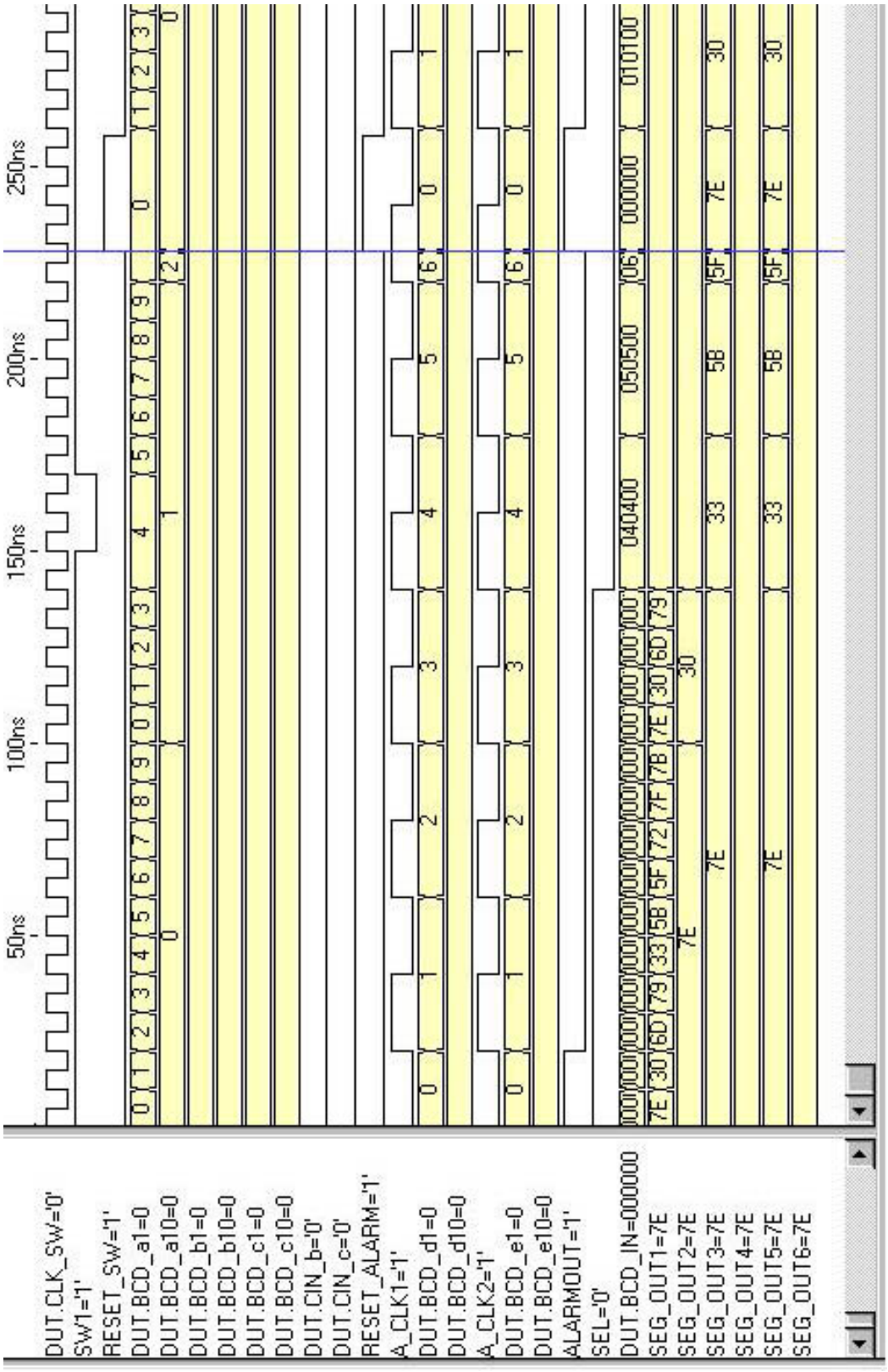


図 3－4. 全体回路シミュレーション結果（a）

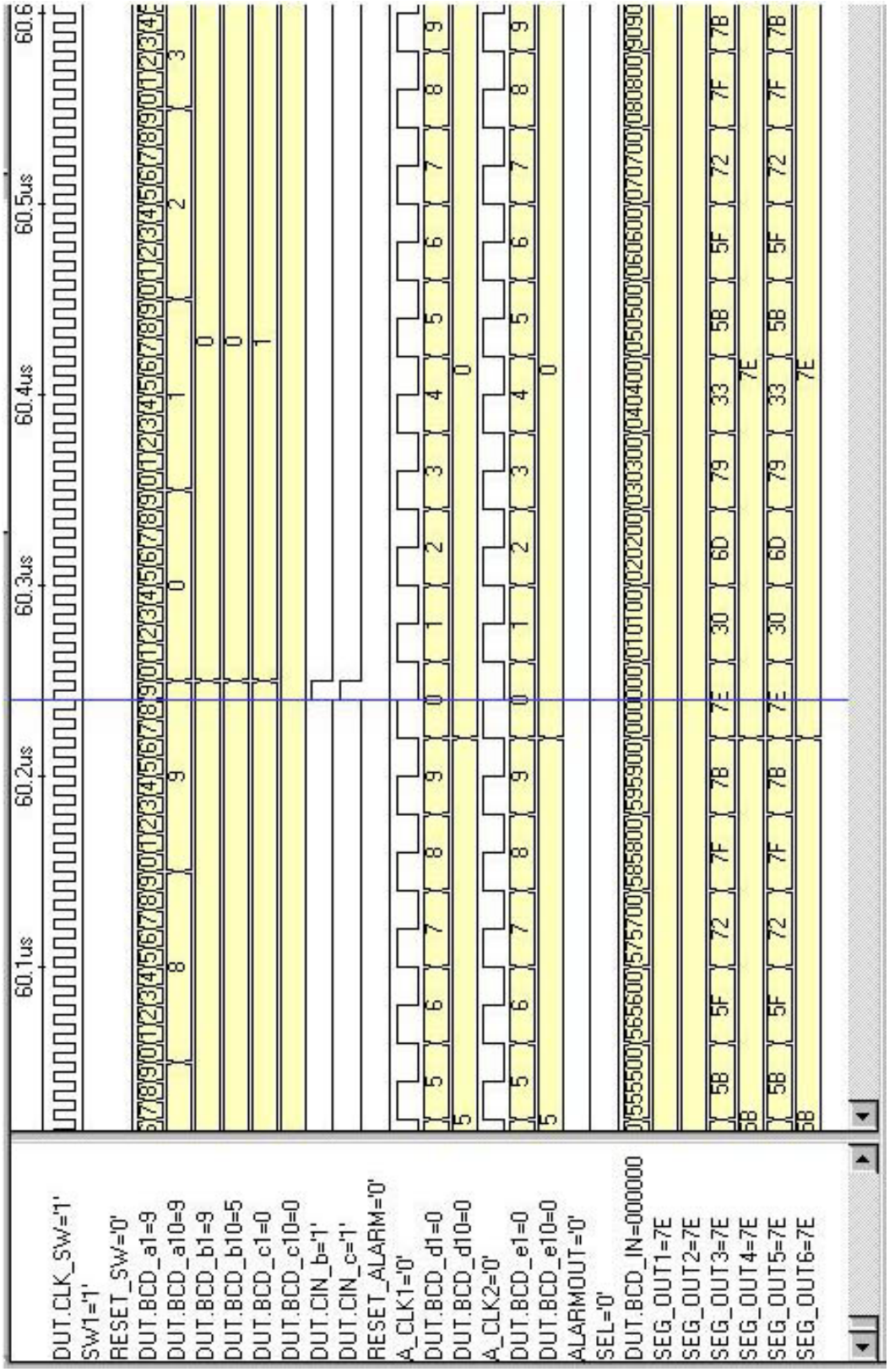


図 3－4. 全体回路シミュレーション結果（b）

図 3－4（a）では、ストップウォッチとアラームの Reset 動作、Selector、Comparator が正常に動作していることが確認できる。RESET_SW、RESET_ALARM はクロックに非同期で動作し、それぞれのカウンタを Reset している。Comparator は、ストップウォッチ用カウンタの“秒”“分”とアラーム用カウンタの“秒”“分”の値が一致している間のみ、ALARMOUT が立ち上がっている。Selector は、SEL＝‘1’ならストップウォッチ用カウンタの値が、SEL=‘0’ならアラーム用カウンタの値が内部信号 BCD_IN に入り、セグメント出力が切り替えられている。

図 3－4（b）では、ストップウォッチ用カウンタが正常に動作していることが確認できる。“1/100 秒”から“秒”への桁上げ信号 CIN_b、“秒”から“分”への桁上げ信号 CIN_c が正常に動作し、00 分 59 秒 59 から 01 分 00 秒 00 へ正常にカウントアップしている。

3.4　FPGA への実装

3.4.1　FPGA の説明

VHDL で設計した回路を、ALTERA 社製「MAX＋plusⅡ」と、aritec 社製「ICP プログラマ＆チェッカ」を使用して FPGA へ書き込む。FPGA チップとして ALTERA 社製「EPM9320LC84-15」と、回路情報プログラミング用専用アダプターとして「ISP-01-9320-84」を使用する。

表 3－5 に「EPM9320LC84-15」の特長を、表 3－6 に電気特性を示す。

表 3－5. EPM9320LC84-15 の特長

特　長	数量
ユーザブルゲート数	6000
フリップフロップ数	484
マクロセル数	320
ロジックアレイブロック数	20
全ピン数	84
ユーザ I/O ピン数（入力専用ピン含む）	60

表 3－6. EPM9320LC84-15 の電気特性

絶対最大定格

シンボル	パラメータ	最小	最大	単位
V_{CC}	供給電圧	–2.0	7.0	V
V_I	DC入力電圧	–2.0	7.0	V
V_{CCISP}	イン・システム・プログラミング時の供給電圧	–2.0	7.0	V
I_{OUT}	ピン当たりのDC出力電流	–25	25	mA
T_{STG}	保存温度	–65	150	°C
T_{AMB}	動作周囲温度	–65	135	°C

推奨動作条件

シンボル	パラメータ	最小	最大	単位
V_{CCINT}	内部ロジックと入力バッファ用供給電圧	4.75 (4.50)	5.25 (5.50)	V
V_{CCIO}	5.0-V動作時の出力バッファ用供給電圧	4.75 (4.50)	5.25 (5.50)	V
	3.3-V動作時の出力バッファ用供給電圧	3.00 (3.00)	3.60 (3.60)	V
V_{CCISP}	イン・システム・プログラミング時の供給電圧	4.75	5.25	V
V_I	入力電圧	0	V_{CCINT}	V
V_O	出力電圧	0	V_{CCIO}	V
T_A	動作周囲温度	0	70	°C

EPM9320LC84-15 は 84 ピンの FPGA であるが、その全てのピンをユーザーが自由に使えるわけではない。電源や GND 供給用のピンや、回路情報書き込み時に使用するピンなど決まっており、ユーザーが入出力ピンとして使用できるのは、I/O ピンと入力専用ピンである。表 3－7 にピンの種類を、表 3－8 にピン番号と信号名の対応表を示す。

表 3－7. EPM9320LC84－15 のピンの種類

ピンの種類	説明	ピン数
入力専用(DIN1～4)	内部マクロセルへ CLK、CLR 信号などを供給する入力専用ピン	4
JTAG	回路情報書き込み時の通信用ピン	4
GND	電源の 0V 電位	8
VCCINT	内部ロジックと入力バッファ用電源ピン	6
VCCIO	I/O 用電源ピン	4
Vpp	回路情報プログラミング時の電源供給ピン	1
NC	未使用ピン	1
I/O	入力にも出力にも自由に使えるピン	56

表 3－8. EPM9320LC84－15 のピン番号、信号名対応表

ピン番号	信号名	ピン番号	信号名	ピン番号	信号名
1	DIN1(GCLK1)	29	NC	57	VccINT
2	I/O	30	TDO	58	I/O
3	I/O	31	I/O	59	I/O
4	I/O	32	I/O	60	VccIO
5	I/O	33	I/O	61	GND
6	GND	34	I/O	62	I/O
7	I/O	35	I/O	63	I/O
8	I/O	36	I/O	64	VccINT
9	I/O	37	VccIO	65	I/O
10	I/O	38	I/O	66	I/O
11	I/O	39	I/O	67	GND
12	I/O	40	I/O	68	I/O
13	DIN3(GCLR)	41	I/O	69	I/O
14	VccINT	42	TDI	70	GND
15	VccIO	43	TCK	71	VccINT
16	I/O	44	I/O	72	DIN4(GOE)
17	I/O	45	I/O	73	I/O
18	GND	46	I/O	74	I/O
19	I/O	47	I/O	75	I/O
20	I/O	48	GND	76	I/O
21	VccINT	49	I/O	77	I/O
22	I/O	50	I/O	78	I/O
23	I/O	51	I/O	79	VccIO
24	GND	52	I/O	80	I/O
25	GND	53	I/O	81	I/O
26	I/O	54	I/O	82	I/O
27	I/O	55	TMS	83	I/O
28	VccINT	56	Vpp	84	DIN2(GCLK2)

3.4.2　FPGA へのプログラミング

FPGA へのプログラミングを行う前に、VHDL で設計した回路の入出力信号を、FPGA のピンに設定しなければならない。今回は、ユーザが自由に使える I/O ピンと、入力専用ピン（DIN1（GCLK1）、DIN2（GCLK2）、DIN3（GCLR））を使用することとし、入出力信号とピンの対応を設定する。表 3－9 に入出力信号のピン設定表を示す。

表 3－9. 入出力信号のピン設定表

入力信号	ピン
ACLK1	84
ACLK2	17
CLK	1
RESET_ALARM	20
RESET_SW	13
SEL	22
SW1	23

出力信号	ピン
ALARMOUT	27
SEG_OUT10	33
SEG_OUT11	34
SEG_OUT12	35
SEG_OUT13	36
SEG_OUT14	38
SEG_OUT15	39
SEG_OUT16	40
SEG_OUT20	41
SEG_OUT21	44
SEG_OUT22	45
SEG_OUT23	46
SEG_OUT24	47
SEG_OUT25	49
SEG_OUT26	50
SEG_OUT30	51
SEG_OUT31	52
SEG_OUT32	54
SEG_OUT33	58
SEG_OUT34	59
SEG_OUT35	62
SEG_OUT36	63

出力信号	ピン
SEG_OUT40	65
SEG_OUT41	66
SEG_OUT42	68
SEG_OUT43	69
SEG_OUT44	73
SEG_OUT45	75
SEG_OUT46	76
SEG_OUT50	77
SEG_OUT51	78
SEG_OUT52	80
SEG_OUT53	81
SEG_OUT54	82
SEG_OUT55	83
SEG_OUT56	2
SEG_OUT60	3
SEG_OUT61	4
SEG_OUT62	5
SEG_OUT63	7
SEG_OUT64	8
SEG_OUT65	9
SEG_OUT66	10

以上の設定を行い、「MAX＋plusⅡ」で再コンパイル後、ISP プログラマを用いてプログラミングを行う。

3.5　基板への実装

（1）部品表

設計結果をまとめると使用部品は表 3－10 のようになった。

表 3－10. 部品表

各部名称	部品名	仕様･型番	数量
発振器	水晶発振器 (1MHz)	HAT7400A　1MHz	1
FPGA		EPM9320LC84-15	1
スイッチ	切換スイッチ (SW1、3、6)		3
	プッシュスイッチ (SW2、4、5、7)		4
表示部	7 セグメント LED (カソードコモン)	S GL8R10 8	1
	抵抗	220Ω	42
	抵抗(小数点用)	470Ω	1
チャタリング防止	シュミットトリガ NOT×2	SN74HC14N	1
	抵抗	220Ω	4
	コンデンサ	47μF	2
アラーム、その他	アラームブザー		1
	LED (赤、緑、黄)		3
	抵抗	330Ω	3

（2）基板配置レイアウト

図 3－5 に基板配置レイアウトを示す。今回は 9.5×14.5mm サイズのユニバーサル基板を 2 枚使用する。

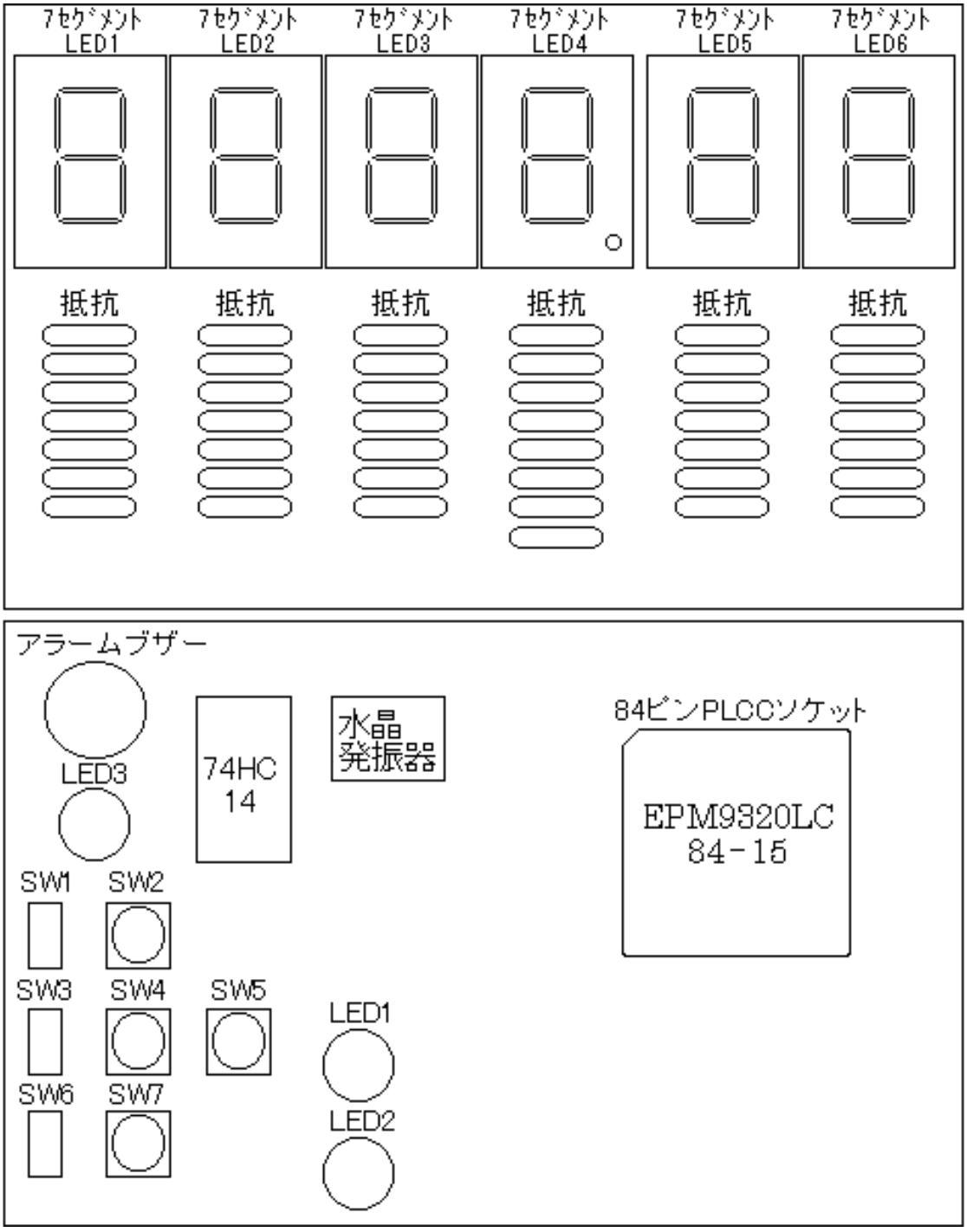
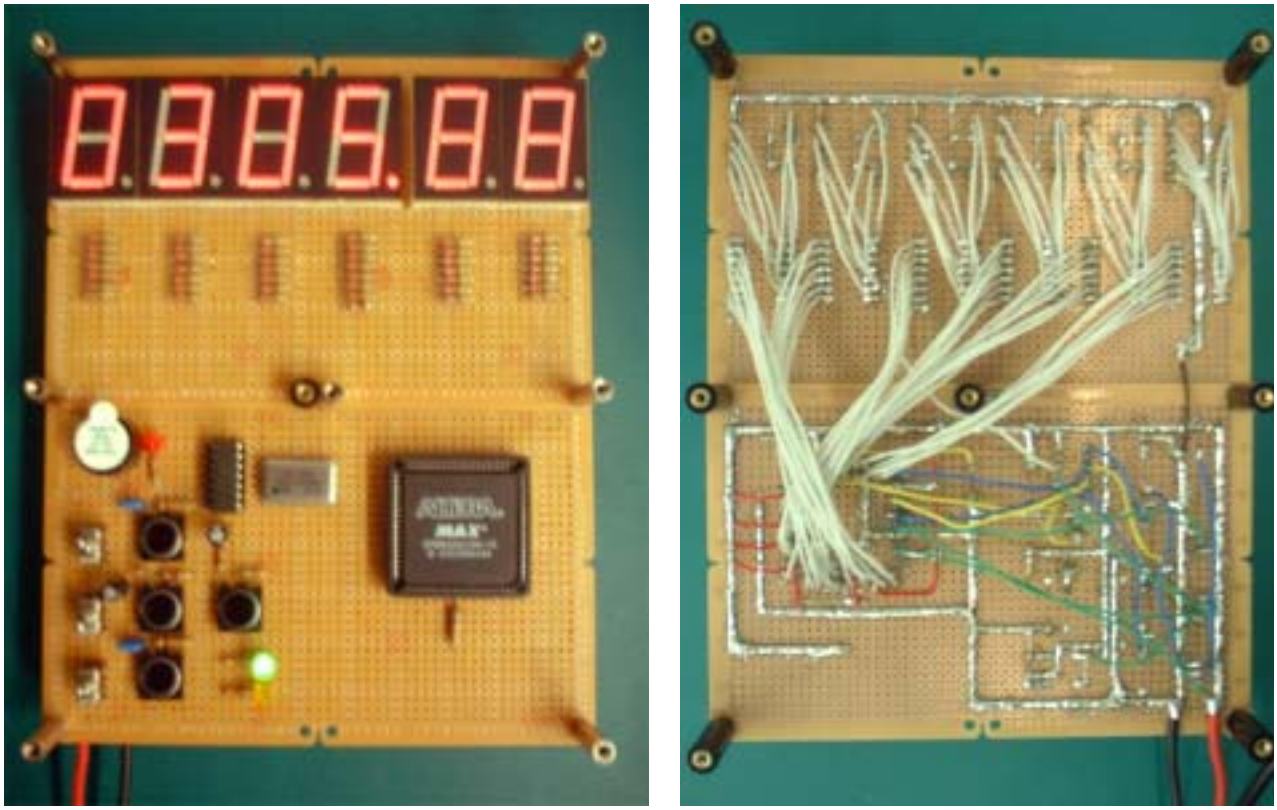


図 3－5. 基板配置レイアウト

（3）基板実装写真

図 3－6 に製作したアラーム付ストップウォッチの写真を示す。今回は部品実装面では配線を行わず、全て半田面で配線を行なった。



（部品面）　　（半田面）

図 3－6．FPGA を用いたアラーム付ストップウォッチの写真

第 4 章　比較・検討

本章では、第 2 章で製作した汎用ロジック IC を用いた回路と、第 3 章で製作した FPGA を用いた回路を比較・検討する。図 4－1 に汎用ロジック IC を用いた回路と FPGA を用いた回路を比較した写真を示す。

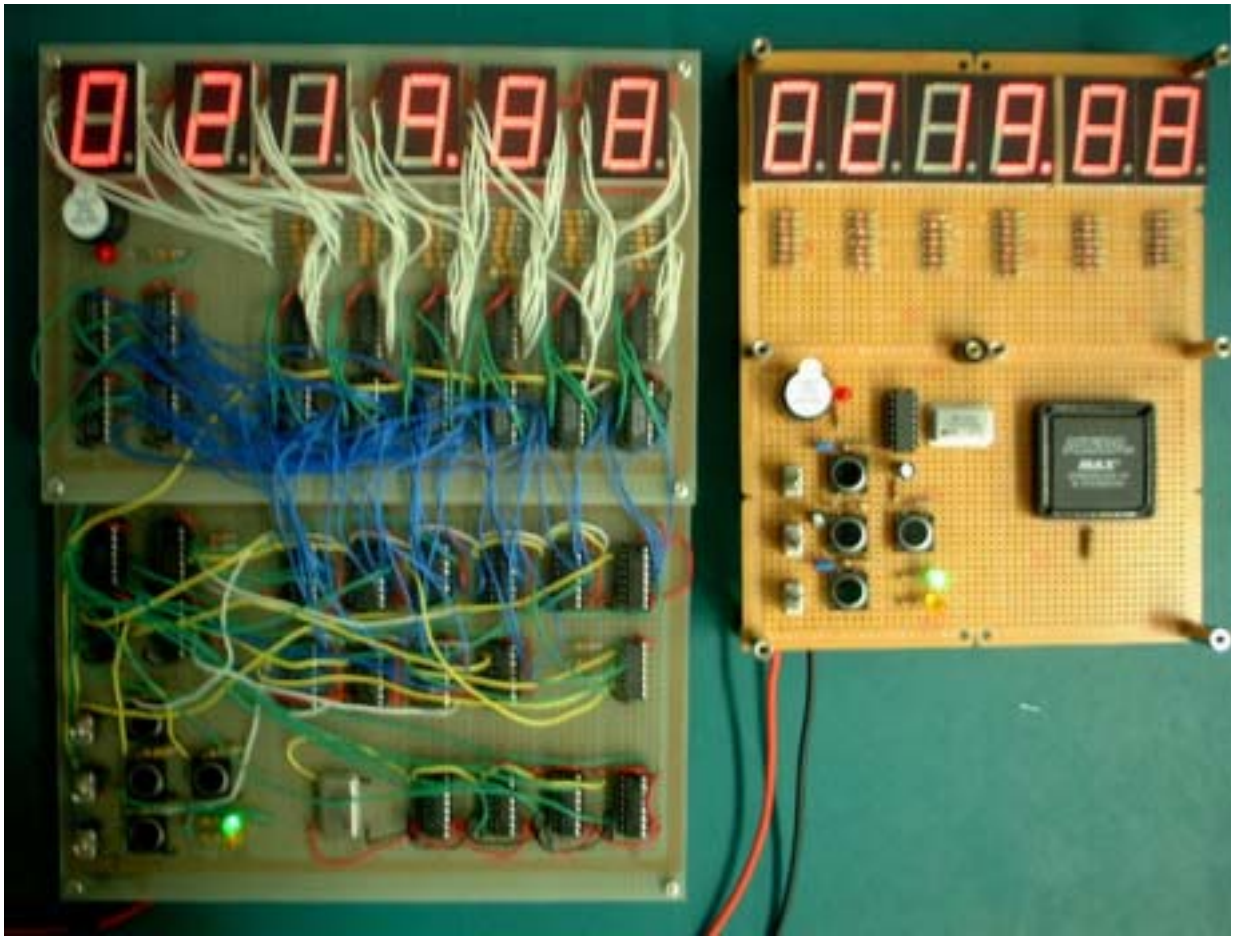


図 4－1. 汎用ロジック IC を用いた回路(左)と FPGA を用いた回路(右)の写真

両回路の違いは、ディジタル回路の部分を汎用ロジック IC で構成しているか、FPGA で構成しているかである。両回路の 5.00V 動作時の消費電流を測定した結果を表 4－1 に示す。

表 4－1. 測定結果

	電源電圧	消費電流	消費電力
汎用ロジック IC を用いた回路	5.00V	258.3mA	1.292W
FPGA を用いた回路	5.00V	160.9mA	0.805W

両回路の 5.00V 動作時の消費電流は、汎用ロジック IC を用いた回路は 258.3mA、FPGA を用いた回路では 160.9mA であった。今回、両回路とも表示の大きい 7 セグメント LED を使用しているので、全体的に消費電流は大きいが、両回路の消費電流の差は、汎用ロジック IC と FPGA の違いと解釈して良いだろう。

汎用ロジック IC を用いた回路では、設計した機能を構成するため、IC を 34 個使用している。主に低消費電力の CMOS IC を使用していたが、数が多いため消費電流は多くなっている。それに比べ、FPGA を用いた回路では、チャタリング防止用のシュミットインバータ（74HC14）以外のディジタル回路を、すべて FPGA で実現している。つまり、今回の設計では、汎用ロジック IC33 個分を FPGA で構成しているため、汎用ロジック IC を用いた回路に比べ消費電流も少ない。また、基板面積も小さくすみ、配線個所も FPGA ソケットの PIN に集中しているので、半田面のみで配線を行なうことが可能だった。

両回路を比較した結果をまとめると、汎用ロジック IC には、たくさんの種類があり、様々な機能を持つ IC が存在しているので、設計する回路に必要なものを探し、組み合わせることで比較的簡単に回路を作ることが出来る。しかし、ある程度規模の大きい回路を作ろうとすると、使わない部分や、使用方法によっては外部で修正を行なう必要もでてくる。今回の場合はカウンタを大量に使用しているため、IC の数が多く、基板面積、消費電流も多く大きくなった。

FPGA を用いた回路では、パッケージ化されている汎用ロジック IC とは違い、自分で自由に FPGA 内部の回路を設計できるので、使用しない回路や無駄な部分を含まず、効率よく機能を実現できている。そのため、基板面積も小さく収まり、汎用 IC を用いた回路と比べ消費電力も少なくなっている。

また、回路を修正する場合、一度基板に実装してしまうと、汎用 IC では IC の付け替え、配線の変更などの必要があり大変である。しかし、FPGA の場合は、VHDL 記述を修正することで回路を変更し、修正後も入出力信号とピン番号の組み合わせを設定して、修正した回路を再び FPGA に書き込める。今回製作した回路は、2.54mm ピッチのユニバーサル基板に実装するため、ソケットを使用している。そのため、FPGA の取り外しがすぐできるので、基板の修正が必要なく、容易に回路変更が可能である。

第 5 章　終わりに

本研究では、複数の機能を持つ順序回路を、部分回路ごとに設計を行なった。まず、汎用ロジック IC を用いて設計・製作を行なったことで、ディジタル回路の設計に関する基礎的な知識を学ぶことができた。

次に、複数の機能を持った順序回路を、VHDL によって設計を行なったことで、VHDL の基本的な記述法を学ぶとともに、記述の修正による回路変更の容易さ、過去の資材を有効に流用できる点など、ハードウェア記述言語を用いた設計手法の有効性を確認できた。このハードウェア記述言語の存在によって、集積回路を容易に、効率よく設計できるということを実感した。さらに、VHDL によって設計した回路情報を、実際に FPGA へプログラミングし、基板に実装し動作させたことで、集積回路の設計・製作を体験することができた。そして、同じ機能を持つものを、汎用ロジック IC を用いた回路と、FPGA を用いた回路で設計・製作し、比較したことで、回路の集積化の有用性をあらためて認識することが出来た。

今回の研究では、汎用ロジック IC を用いた設計法、ハードウェア記述言語である VHDL を用いた設計法を習得すること、そして設計を行なう上で必要な知識を得ることを目的としていた。この研究を行なう前に比べ、回路に関する理解を深めることができたと思うが、あくまで基礎的なことであり、実用的な製品の設計を行なうにはまだまだ知識・技術の向上が必要である。今後は、アセンブラによるマイコンを用いた設計手法等も経験し、回路設計における全体的な知識・技術の向上を目標としたい。

謝　辞

本研究を行うに際し、懇切丁寧な御指導、御鞭撻を賜りました、高知工科大学工学部電子・光システム工学科 矢野政顕 教授に心から感謝致します。

研究中、懇切丁寧な御指導を賜りました高知工科大学電子・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 橘 昌良 助教授に厚く御礼申し上げます。

また終始、適切なご助言、ご助力を頂きました高知工科大学工学研究科基盤工学専攻電子･光エレクトロニクスコース修士課程、矢野研究室の木村知史氏、中村基継氏、石川純平氏、松村暢也氏、及び原研究室、橘研究室の諸氏に心から感謝し、御礼申し上げます。

参考文献

[1] 中村　次男　著
「図解　ディジタル回路入門」　日本理工出版会

[2] 中村　次男　著
「ディジタル回路設計法－ワンチップ化の実例集－」　日本理工出版会

[3] 岩本洋　監修　　堀　啓太郎　著
「絵とき　ディジタル回路入門　早わかり」　オーム社

[4] 2000 年版　最新半導体規格表シリーズ
「汎用ロジック・デバイス規格表」　CQ 出版社

[5] TEXAS INSTRUMENTS
http://www.ti.com/、http://www.tij.co.jp/

[6] HITACHI Semiconductor Products
http://www.hitachisemiconductor.com/sic/servlet/request

[7] 東芝セミコンダクター社
http://www.semicon.toshiba.co.jp/prd/logic/ft_logic.html

[8] 長谷川　裕恭　著
「VHDL によるハードウェア設計入門」　CQ 出版社