卒 業 研 究 報 告

題　　目

VHDL による順序回路の設計

指 導 教 員

矢野　政顕　教授

報 告 者

学籍番号: 1030220
氏名:　　松田　圭弘

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

現在の私たちの生活の発展は電子機器なしでは考えられない所まできている。少し周囲を見回しただけでもパソコン、携帯電話、PDA など様々なものに囲まれている。例えばパソコンを例に取ってみると、CPU、グラフィックスチップ、サウンドチップなどがあり、その中にも複数の回路がある。こうした機器の進化とともにその回路も多様化、複雑化し、設計はますます難しくなっており、膨大な時間と経費がかかる作業になってきている。そうした中で集積回路を比較的簡単になおかつ短期間で設計する手段としてハードウェア記述言語(Hardware Description Language : HDL)が生み出された。ここではその数あるハードウェア記述言語の中の一つである VHDL(VHSIC HDL)を使って小規模な回路を設計、シミュレーションすることにより回路設計に対する知識を深める。

本論文は 5 章から構成されている。第 1 章ではここまでの研究の経緯を述べる。第 2 章では VHDL の開発までの歴史や、その文法などを述べる。第 3 章では、VHDL で設計する回路の元となるものを汎用 IC で設計する。第 4 章では、第 3 章で設計した回路に他の複数の機能を付与して VHDL で再設計するとともに、シミュレーションを行いその結果について報告する。第 5 章では、本論文の集大成として第 4 章で行ったシミュレーション結果の考察などをまとめる。

第 2 章　VHDL について

2.1　ハードウェア記述言語

今までの回路図入力による設計手法に置き換わるものとしてハードウェア記述言語を使用した設計がさかんに行われるようになってきた。それには回路図入力に比べ HDL 入力に表 2.1 のような利点があるためである。[1]

表 2.1　回路図入力と HDL 入力による設計の違い

	回路図入力	HDL 入力
1	回路図入力に時間がかかる。	テキストで簡単に入力できる。
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	回路変更が難しい。	回路変更が容易。
4	設計者以外では、内容を理解しずらい。	誰にでも内容を理解しやすい。
5	特定の半導体メーカーのライブラリを使用して回路図入力をする。	半導体メーカーのライブラリを使用しない。どのメーカーでも作成可能。

主なハードウェア記述言語には、VHDL, Verilog-HDL, UDL/I, SFL があり(表 2.2 参照)、それぞれ一長一短があるが、その中でも業界標準として最も広く普及しているのが VHDL である。

表 2.2　各種 HDL の比較

言語名	開発元	特徴
VHDL	米国国防総省が中心	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	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。
UDL/I	日本電子工業振興協会	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。
SFL	PARTHENON システムの言語	RTL での記述のみ可能。完全同期式の回路に限定。単純で分かりやすい記述。

RTL : ロジック回路生成可能なレベル

2.2 VHDL の歴史

VHDL は、米国国防総省の VHSIC(Very High Speed Integrated Circuit)委員会で 1981 年に提唱された。VHDL が生まれた理由は、当時、大規模 IC の開発には、より上位のレベルでの検証が求められていたことと、国防省向けの ASIC(特定用途向けの集積回路)の開発が長いもので 3 年から 4 年もかかっていたからである。開発当初での最速の ASIC を使っていても開発が終了する頃に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入力するのではなく、ハードウェア記述言語で設計することによって、開発終了時点で最速の ASIC が選択できるようにする必要性があった。

こうして、1983 年から 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。

その後、1986 年には IEEE(米国電気電子技術者協会)での標準化作業が VASG(VHDL Analysis & Standardization Group)委員会で始まり、1987 年 5 月には言語仕様書が作成され、12 月には IEEE Std 1076-1987 として承認された。1989 年には VHDL シミュレータや論理合成ツールが販売されるようになった。さらに 1993 年には機能拡張が行われ、現在ではアナログ回路への拡張も行われている。

2.3 VHDL の仕組み

VHDL では、図 2.1 に示すように外部とのインターフェース部分(エンティティ)と内部の動作(アーキテクチャ)を別々に記述するようになっている。

エンティティ宣言では回路の基本的な構造を記述し、アーキテクチャ宣言ではその回路の中身を記述する。また、一つのエンティティ宣言の中に RTL、ゲートレベルといった複数のアーキテクチャ宣言を持つことが可能である。

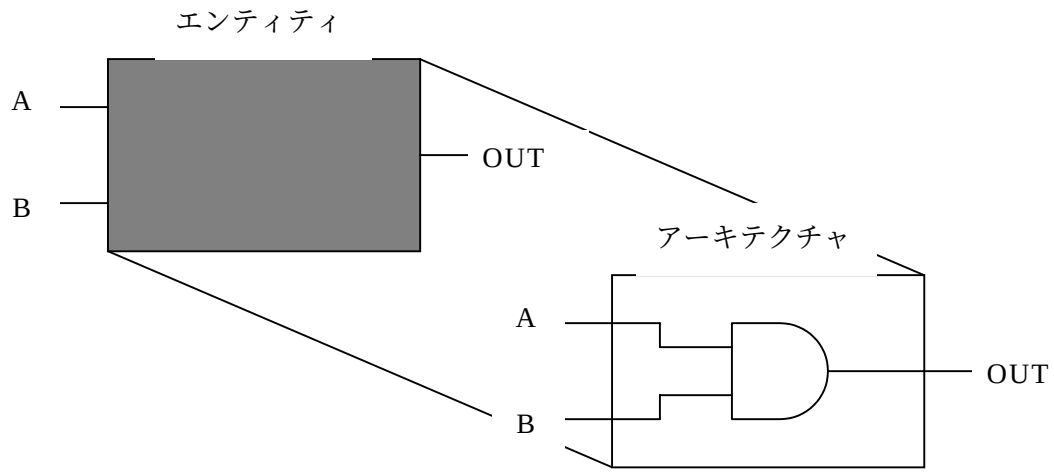


図 2.1　エンティティとアーキテクチャ

2.3.1　VHDL による設計方法

VHDL の基本的な記述例として、図 2.2 に示すハーフアダーについての記述例を示す。

・　ハーフアダーの記述例

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

entity HALF_ADDER is
        port (
                A : in std_logic;
                B : in std_logic;
                S : out std_logic;
                CO : out std_logic);
end HALF_ADDER;

architecture DATAFLOW of HALF_ADDER is
signal C, D : std_logic;
begin
        C <= A or B;
        D <= A nand B;
        CO <= not D;
        S <= C and D;
end DATAFLOW;
```

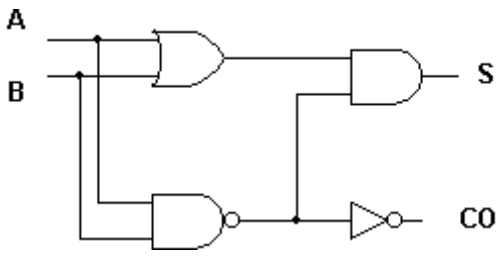


図 2.2　ハーフアダー

一番初めの 2 行ではライブラリの宣言とパッケージの呼び出しを行っている。また、IEEE のライブラリの使用を宣言している。

エンティティ部では使用するポートの宣言などを行う。

```
entity エンティティ名 is
        port(
                ポート名 : 方向 データタイプ);
end エンティティ名;
```

ポートには方向とタイプを宣言する必要がある。代表的なものを表 2.3 と表 2.4 に示す。

表 2.3　ポートの方向

方向	意味
in	入力
out	複数の信号を代入できる出力(内部で再利用できない)
inout	双方向
buffer	一つの信号しか代入できない出力(内部で再利用できる)
linkage	方向指定なし

表 2.4　主なデータタイプ

データタイプ	内容
bit	0, 1 のロジック値
integer	整数
real	浮動小数点
boolean	論理値(TRUE or FALSE)
std_logic	0, 1, 不定, ハイインピーダンス状態などおおよそ論理設計で必要なデータタイプがすべてある
bit_vector	bit の配列タイプ
std_logic_vector	std_logic の配列タイプ

VHDL ではデータタイプの区別に関しては非常に厳しい制約を設けている。基本的には異なるデータタイプ同士の論理演算や四則演算を行うことはできない。どうしても必要な場合は演算用のパッケージを使うか、同じデータタイプに変換する必要がある。

アーキテクチャ部は以下のような構成になっている。

```
architecture アーキテクチャ名 of エンティティ名 is
begin
        同時処理文
end アーキテクチャ名;
```

同時処理文とは字のごとく同時に処理されるもので begin，end 間の文の順番に関係なくすべて同時に処理される。回路の動作を記述する場所である。プロセス文を使うことで、条件文を用いたもっと複雑な記述をすることもできる。

```
architecture XXX of YYY is
begin
        process (A, B, C)
        begin
                条件文など
        end process;
end XXX;
```

process()の()内は、センシティビティリストと言ってこの中のどれかの値が変化するとプロセス文内の begin-end 間の再計算が行われる。

architecture と begin の間は信号、関数を定義する場所で、記述例では内部信号の定義を行っている。

```
signal 内部信号名 : データタイプ;
```

第3章　順序回路の設計

順序回路の具体例として図 3.1 に示す時計回路を設計した。VHDL で設計する前に元となる順序回路(時計)の動作を回路シミュレータで確認した。この時計は時、分、秒をカウントするだけのシンプルなものである。時刻設定のためのスイッチも備えている。

3.1　時計の設計

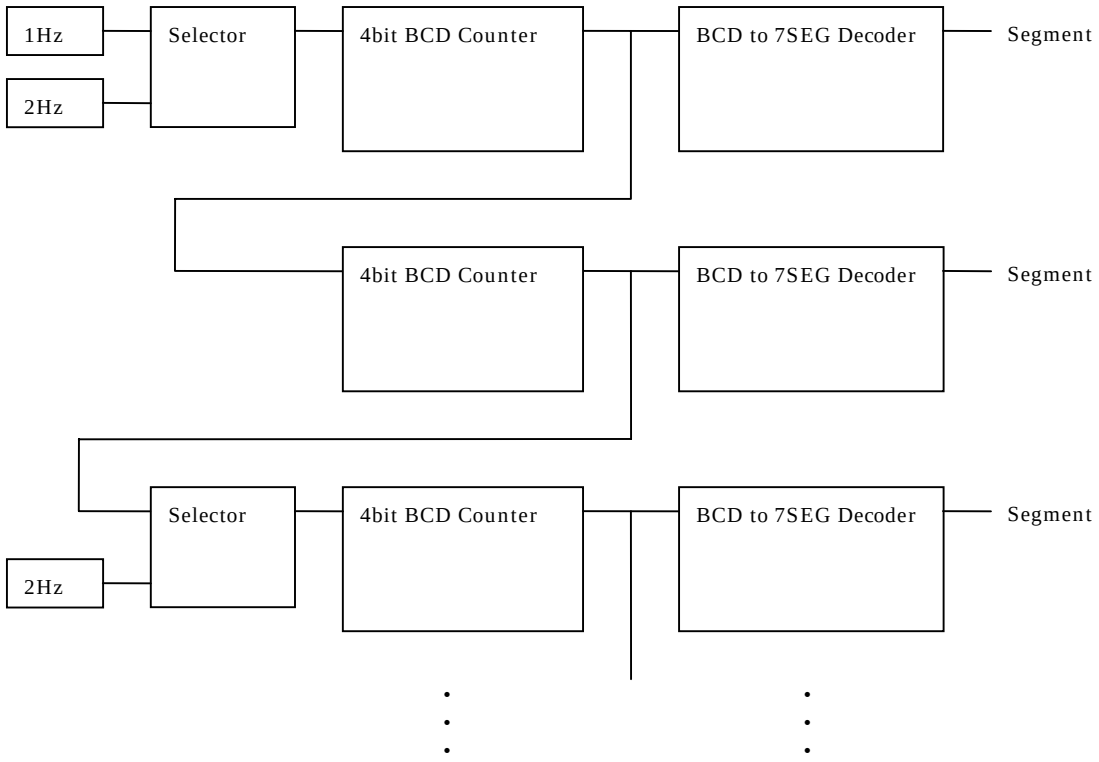
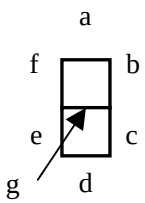


図 3.1　時計のブロック図

時間、分、秒、それぞれの桁ごとに 4bit の BCD カウンタを用意しておき、それを 10 進カウンタや 6 進カウンタとすることにより 60 進、24 進を表現する。BCD カウンタから出た信号はデコーダに入り 7 セグメント用の信号に変換される。桁上げは BCD カウンタからの出力をそれぞれ 10 進用 ”1010”, 6 進用 ”0110”を見て桁上げとカウンタのリセットを行う。時部分だけは 24 ”0010 0100”でもリセットをかける。時刻設定時はセレクタ(2 入力のマルチプレクサ)で、1Hz と 2Hz を切り替える。時、分、秒ごとにスイッチがありそれ

ぞれで早送りできる。

図 3.2 に BCD コードから 7 セグメントコードへの変換規則と、図 3.3 に 2 入力のマルチプレクサの回路図をのせる。 [2] [3]



BCD Code	7SEG Code	Display
0000	1111110	0
0001	0110000	1
0010	1101101	2
0011	1111001	3
0100	0110011	4
0101	1011011	5
0110	1011111	6
0111	1110010	7
1000	1111111	8
1001	1111011	9

図 3.2　BCD コードから 7 セグメント用コードへの変換規則

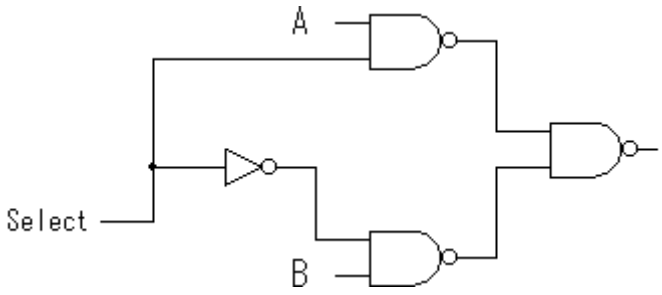


図 3.3　2 入力のマルチプレクサを使ったセレクタ

セレクト信号が 0 または 1 に変わることで、A の信号が入るか B の信号が入るかを決めることができる。

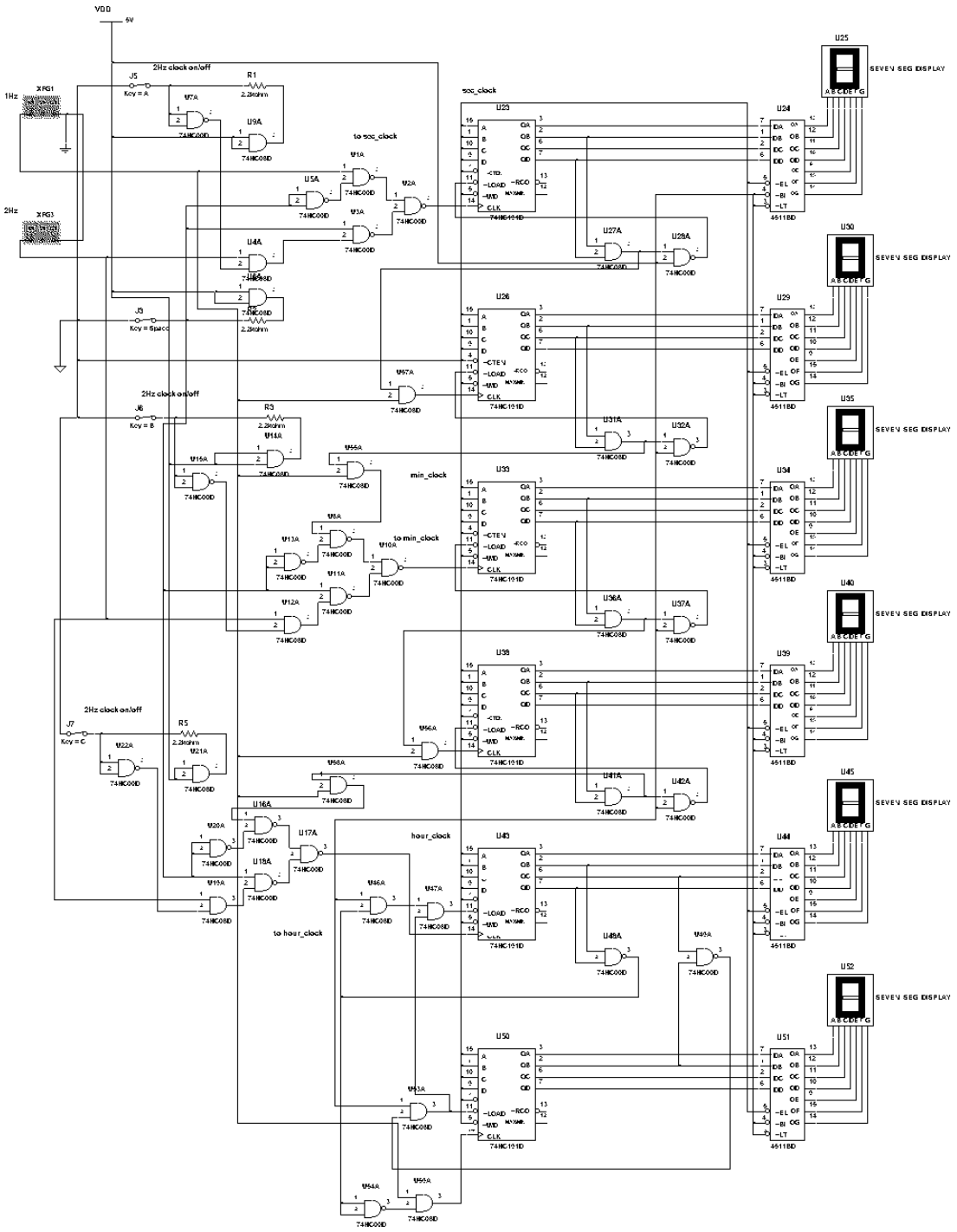

VDD
5V
2Hz clock on/off
J5
Key = A
R1
2.2kohm
1Hz
XTAL1
2Hz
XTAL2
U7A
U9A
74HC08D
to soc_clock
U1A
U5A
U2A
U3A
U4A
U6A
soc_clock
U23
74HC161D
U24
4511BD
U25
SEVEN SEG DISPLAY
U27A
U28A
74HC00D
U26
U29
U30
J3
Key = Space
R3
2.2kohm
U57A
2Hz clock on/off
J6
Key = B
R3
2.2kohm
U14A
U15A
U55A
min_clock
U8A
U13A
U10A
to min_clock
U11A
U12A
U31A
U32A
U33
U34
U35
U36A
U37A
2Hz clock on/off
J7
Key = C
R5
2.2kohm
U22A
U21A
U56A
U58A
U16A
U20A
U17A
U18A
U19A
hour_clock
U38
U39
U40
U41A
U42A
U43
U44
U45
U46A
U47A
to hour_clock
U48A
U49A
U50
U51
U52
U53A
U54A
U59A

図 3.4　時計の回路図

3.2　時計のシミュレーション結果と考察

シミュレーション速度が遅いため桁上げが起こるのを確認してから、時、分、秒の 2 つのカウンタごとにクロックを入れて調べている。

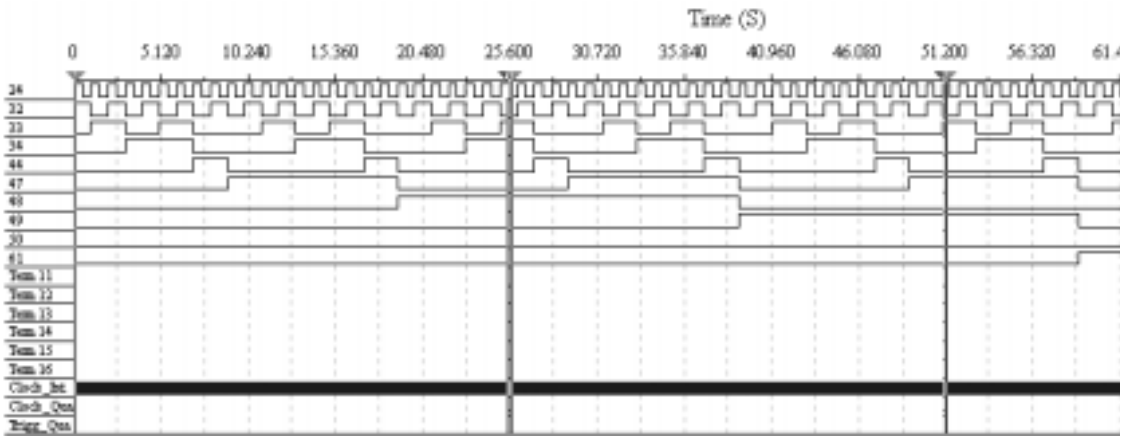


図 3.5　時計の秒のシミュレーション結果

図 3.5 は時計の秒のシミュレーション結果である。一番上から 1Hz のクロック、秒の下位桁のカウンタ出力 A、B、C、D、上位桁のカウンタ出力 A、B、C、D、分のカウンタ出力 A となっている。1 秒ごとに 0000 から順番にカウントしていって下位桁が 1010(10 秒)になったとき上位桁へ桁上げを行って、上位桁が 0110(60 秒)になったとき、分のカウンタへの桁上げが行われているので 60 進カウンタとしての正しい動作が確認できた。

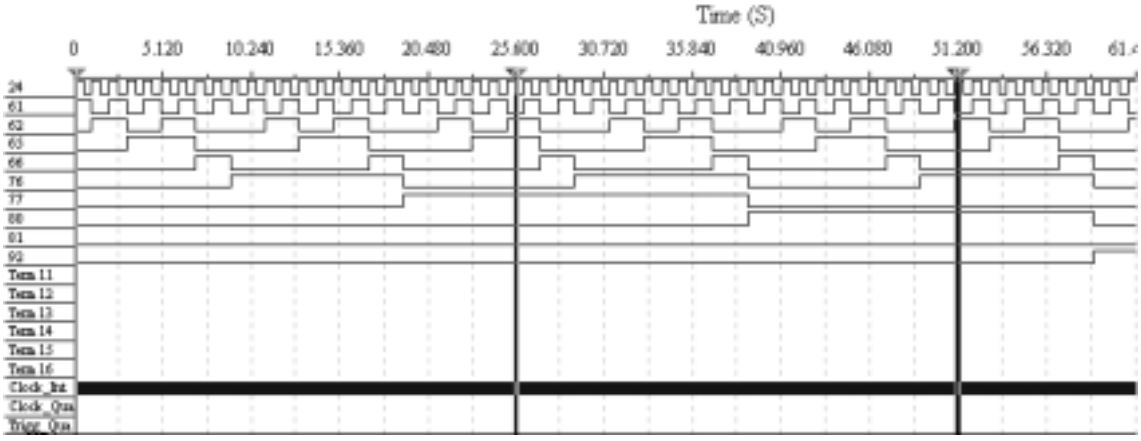


図 3.6　時計の分のシミュレーション結果

図 3.6 は時計の分のシミュレーション結果である。図 3.5 の場合と同様に 60 進カウンタである。一番上から 1Hz のクロック、分の下位桁のカウンタ出力 A、B、C、D、上位桁の

カウンタ出力 A、B、C、D、時のカウンタ出力 A となっている。図 3.5 と同じ検証方法から正しい動作であることが分かった。

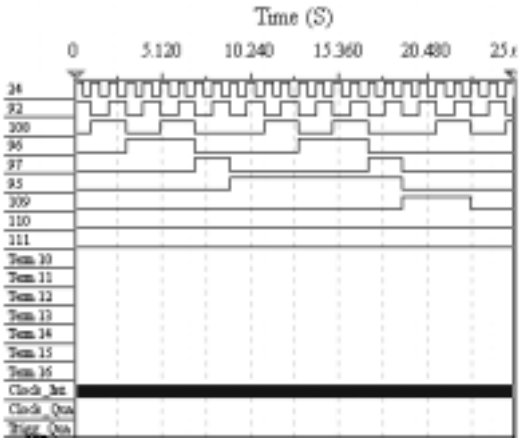


図 3.7　時計の時のシミュレーション結果

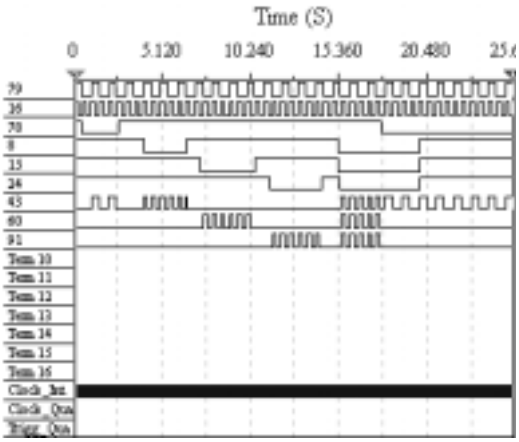


図 3.8　時計のスイッチのシミュレーション結果

図 3.7 は時計の時のシミュレーション結果である。一番上から 1Hz のクロック、時の下位桁のカウンタ出力 A、B、C、D、上位桁のカウンタ出力 A、B、C、D となっている。下位桁では 10 進カウンタとして 1010 になった場合と 24 時になった場合にもカウントが 0000 になっており、上位桁も 24 時で 0000 となっているので正しい動作であることが分かる。

図 3.8 の時計のスイッチのシミュレーション結果では上から順に 1Hz クロック、2Hz クロック、時刻設定用のスイッチ、秒、分、時のスイッチ、秒、分、時の下位桁のカウンタ出力 A となっている。最初に時刻設定スイッチをオフにしたとき普通の時計として 1Hz のクロック信号が秒のカウンタ出力 A に入っている。そして時刻設定スイッチがオンになってそれぞれの秒、分、時の早送りスイッチを押した時、それぞれのカウンタには 2Hz が入っている。そして時刻設定スイッチがオフになったとき、早送りスイッチを押しても出力には変化は起きていない。これは予定通りの動作である。

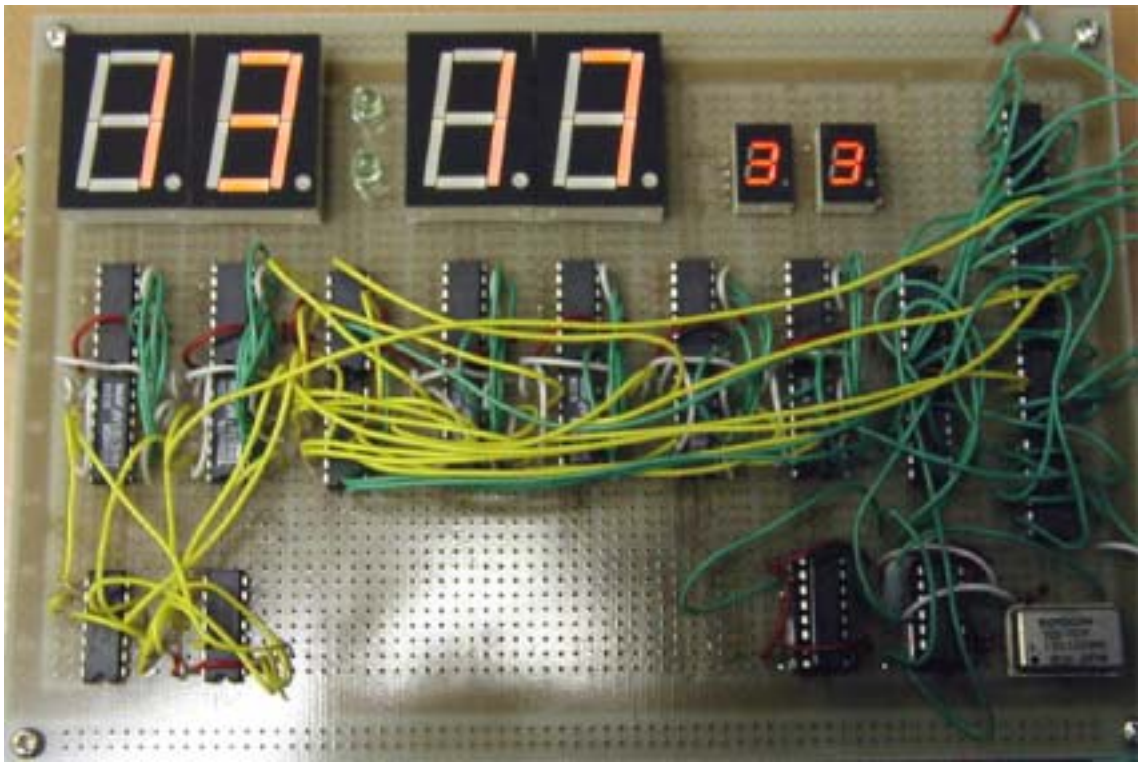


図 3.9 汎用ロジック IC で設計された時計

4.194304MHz(2^22)の水晶発信器から 20 個と 21 個の D フリップフロップを通して 1Hz と 2Hz を取り出している。中心部にはそれぞれの桁ごとに 4bit の BCD アップカウンタと 7 セグメントデコーダが用意されている。スイッチは画像には載ってないが別の小さな基盤上にある。マルチプレクサは NAND4 個を用いてある。

図 3.5～図 3.8 までのシミュレーション結果と図 3.9 の汎用ロジック IC で製作した回路で正しい動作が確認できたので第 4 章にて VHDL による設計に入る。

第4章　VHDL による順序回路の設計

第 3 章での回路の動作を元に VHDL で再設計する。この時計にはアラームタイマーとカレンダーの機能を新たに付与する。

4.1　VHDL による時計の設計

まず、普段の動作では CLK または前のカウンタからの桁上りを時刻設定時には CLK2 に切り替えるようなセレクタを 3 つ分作る。

```
process ( TIMER_SET, SEC_SET, CLK, CLK2 ) begin
        if ( TIMER_SET = "01" ) then
            if ( SEC_SET = '0' ) then
                SEL1_OUT <= '0';
            elsif ( SEC_SET <= '1' ) then
                SEL1_OUT <= CLK2;
            end if;
        elsif ( TIMER_SET = "00" ) then
            SEL1_OUT <= CLK;
        end if;
end process;
```

次に BCD カウンタを設計する。リセット機能を持たせてからそれぞれのカウンタごとに桁上げを行う値を決め、桁上げ信号も作る。そしてカウンタ自体の動作を記述する。

```
process ( RST, SEL1_OUT, COUNTER1_OUT, COUNTER2_CLK ) begin
        if ( RST = '1' ) then
            COUNTER1_OUT <= "0000";
            COUNTER2_CLK <= '0';
        elsif ( COUNTER1_OUT = "1010" ) then
            COUNTER1_OUT <= "0000";
            COUNTER2_CLK <= '1';
        else
            COUNTER2_CLK <= '0';
        end if;
        if ( SEL1_OUT'event and SEL1_OUT = '1' ) then
            COUNTER1_OUT <= COUNTER1_OUT + '1';
        end if;
end process;
```

時の一桁目だけは 24 時になった瞬間でもカウンタの値のリセットを行う。

```
elsif ( COUNTER5_OUT = "1010" or (COUNTER5_OUT="0100" and COUNTER6_OUT="0010") ) then
```

```
COUNTER5_OUT <= "0000";
COUNTER6_CLK <= '1';
```

最後に BCD コードを 7 セグメント用のコードに変えるために case 文を使って BCD カウンタからの出力信号をすべての場合において記述する。時刻カウント用の BCD カウンタは 6 個あるので 7 セグメント用のデコーダも 6 個分記述する。

```
process () begin
        case COUNTER1_OUT is
             when "0000" => 7SEG_DEC1 <= "1111110";
             when "0001" => 7SEG_DEC1 <= "0110000";
             when "0010" => 7SEG_DEC1 <= "1101101";
             when "0011" => 7SEG_DEC1 <= "1111001";
             when "0100" => 7SEG_DEC1 <= "0110011";
             when "0101" => 7SEG_DEC1 <= "1011011";
             when "0110" => 7SEG_DEC1 <= "1011111";
             when "0111" => 7SEG_DEC1 <= "1110010";
             when "1000" => 7SEG_DEC1 <= "1111111";
             when "1001" => 7SEG_DEC1 <= "1111011";
             when "1010" => 7SEG_DEC1 <= "1111110";
             when others => 7SEG_DEC1 <= "1111110";
        end case;
end process;
```

4.2 アラームタイマーの設計

このアラームタイマーは設定されたある時刻に合わせてアラーム用の信号を出力するというものである。

アラーム用の時刻設定にはモードを切り替えることにより本体の時計部分のスイッチを兼用する。本体の時計の時間、分の所を比較しながら設定された時刻になるとアラームがオンになる。強制停止用のスイッチを押すとアラームはオフとなる。

4.2.1 VHDL によるアラームタイマーの設計

まず信号宣言でアラームに必要な信号を定義する。

```
signal ALARM_MIN : std_logic_vector(0 to 3);
signal ALARM_MIN2 : std_logic_vector(0 to 3);
signal ALARM_HOUR : std_logic_vector(0 to 3);
signal ALARM_HOUR2 : std_logic_vector(0 to 3);
signal ALARM_MIN_AGARI : std_logic;
signal ALARM_HOUR_AGARI : std_logic;
```

```
signal ALARM_SEL : std_logic;
signal ALM_OUT : std_logic;
```

次にアラーム用のタイマーの部分を記述する。TIMER_SET(時計の状態を切り替えるための信号)が 10 のときにアラームがセットできるようして、その他の部分は時計用のカウンタと同じような動作にする。

```
process ( RST, CLK2, TIMER_SET, MIN_SET, ALARM_MIN) begin
        if ( RST = '1' ) then
            ALARM_MIN <= "0000";
        elsif ( TIMER_SET = "10" and MIN_SET = '1' and … ) then
            ALARM_MIN <= ALARM_MIN + '1';
        end if;
        if ( ALARM_MIN = "1010" ) then
            ALARM_MIN <= "0000";
            ALARM_MIN_AGARI <= '1';
        else
            ALARM_MIN_AGARI <= '0';
        end if;
end process;
```

内部の信号の状態が分からないとアラームタイマーをセットできないので視認できるように TIMER_SET の状態に合わせてセグメントの表示を切り替えるための記述を追加する。

```
            if ( TIMER_SET="10" ) then
                SEG_SEL3 <= ALARM_MIN;
            else
                SEG_SEL3 <= COUNTER3_OUT;
            end if;
```

4.3 カレンダーの設計

このカレンダーは 2000 年から 3999 年までの日付、曜日を表示するもので、うるう年にも対応している。曜日は年数、日付が確定すれば自動で計算する。4000 年になった場合は 2000 年に戻る。

4.3.1 カレンダーのアルゴリズム

まず年にはうるう年というものが設定されている。平年かうるう年かで 2 月の日数が変わってくるので区別することが必要である。現在の暦でのその定義は「年数が 4 の倍数の

年をうるう年とする。ただし、100 の倍数の年で 400 の倍数でない年は、平年とする。」となっている。そのため、400 年や 800 年はうるう年だが、500 年、900 年は平年となる。

次に曜日について考える。ツェラー(Zeller)の公式によれば、 [4]

> 西暦 y 年 m 月 d 日の曜日は、
> (y + [y/4] - [y/100] + [y/400] + [2.6m+1.6] + d) mod 7
> この値が 0 なら日曜日、1 なら月曜日、・・・6 なら土曜日である。
> ただし、1、2 月は前年の 13、14 月とする。
> (有効範囲は 1583 年から 3999 年までとする。)
>
> [ ]はガウス記号 ([ ]の中の値を超えない最大の整数)

となる。これらを踏まえた上でカレンダーを設計する。

4.3.2 VHDL によるカレンダーの設計

始めにカレンダー用の信号を定義する。YEAR などの信号で integer を使っているのは曜日の計算に乗算や除算を使用するためである。(std_logic_vector では除算などを直接、行うことができないため)

```
signal YEAR : integer range 0 to 11;
signal MONTH : integer range 0 to 3;
signal MAX_DAYS : integer range 0 to 4;
signal DAY : integer range 0 to 4;
signal MONTH_KETA_AGARI : std_logic;
signal DAY_KETA_AGARI : std_logic;
signal TMP_YEAR : integer range 0 to 11 := 0;
signal TMP_MONTH : integer range 0 to 3 := 0;
signal TMP_YOUBI : integer range 0 to 2 := 0;
```

次にその月の日数を決める。平年かうるう年かで 2 月の日数を変え、その他の月はそれぞれ 30 日か 31 日かで分ける。うるう年の計算に必要な YEAR が 4、100、400 の倍数かの判断は除算の余りで決定する。

```
    if ( MONTH = 2 and ( ( YEAR mod 4 = 0 and YEAR mod 100 /= 0 ) or ( YEAR mod 400
= 0 ) ) ) then
        MAX_DAYS <= 29;
    elsif ( MONTH = 2 and not ( ( YEAR mod 4 = 0 and YEAR mod 100 /= 0 ) or ( YEAR
mod 400 = 0 ) ) ) then
        MAX_DAYS <= 28;
```

```
    elsif ( MONTH = 1 or MONTH = 3 or MONTH = 5 or MONTH = 7 or MONTH = 8 or MONTH
= 10 or MONTH = 12 ) then
        MAX_DAYS <= 31;
    elsif ( MONTH = 4 or MONTH = 6 or MONTH = 9 or MONTH = 11 ) then
        MAX_DAYS <= 30;
    end if;
```

最後に重要な曜日の計算であるが、ツェラーの公式から 1 月と 2 月は前年の 13 月、14 月で計算するため、その場合分けを行っている。曜日の計算には時間がかかるため曜日用に別のクロックを用意している。計算した後、TMP_YOUBI の値によって日曜日から土曜日を 000 から 110 で出力している。

```
       TMP_YOUBI <= ( TMP_YEAR + TMP_YEAR / 4 - TMP_YEAR / 100 + TMP_YEAR / 400 +
( 13 * TMP_MONTH + 8 ) / 5 + DAY ) mod 7;
            if ( MONTH = 1 or MONTH = 2 ) then
                TMP_YEAR <= YEAR - 1;
                TMP_MONTH <= MONTH + 12;
            else
                TMP_YEAR <= YEAR;
                TMP_MONTH <= MONTH;
            end if;
        end process;
        process ( TMP_YOUBI ) begin
            if ( TMP_YOUBI = 0 ) then
                YOUBI <= "000";
            elsif ( TMP_YOUBI = 1 ) then
                YOUBI <= "001";
            elsif ( TMP_YOUBI = 2 ) then
                YOUBI <= "010";
            elsif ( TMP_YOUBI = 3 ) then
                YOUBI <= "011";
            elsif ( TMP_YOUBI = 4 ) then
                YOUBI <= "100";
            elsif ( TMP_YOUBI = 5 ) then
                YOUBI <= "101";
            elsif ( TMP_YOUBI = 6 ) then
                YOUBI <= "110";
            end if;
```

第 5 章でシミュレーション結果の考察を行う。

第５章　シミュレーション結果と考察

5.1　シミュレーション結果

次のページ以降にまとめてシミュレーション結果の図を示す。時計とカレンダー本体に関してはシミュレーション結果が膨大なため、桁上げなどの重要な部位だけをのせている。

図 5.1～図 5.3 は時計本体の動作のシミュレーション結果である。TIMER_SET は時刻設定用の信号、HOUR_SET、MIN_SET、SEC_SET は時、分、秒のそれぞれの早送り信号になっている。SEG_DEC は 7 セグメント用の信号、COUNTER_OUT は上から順に秒、分、時の BCD カウンタの出力である。BCD カウンタの出力は 0，１，・・・ 9 または 0，1，・・・ 5 で次段への桁上げと正常な動作になっているのが確認できる。時に関しても 24 時になった瞬間 00 に戻っており時計の動作となっている。次に 7 セグメント用の信号の出力であるが、図 5.1～図 5.3 の BCD カウンタの出力と表 5.1 の結果を照らし合わせると正しいことが確認できる。

表 5.1　7 セグメント用信号の出力結果の確認

シミュレーション結果の表記(16 進)	2 進数での表記	Display
7E	(0)1111110	0
30	(0)0110000	1
6D	(0)1101101	2
79	(0)1111001	3
33	(0)0110011	4
5B	(0)1011011	5
5F	(0)1011111	6
72	(0)1110010	7
7F	(0)1111111	8
7B	(0)1111011	9

図 5.4､図 5.5 は時刻設定のシミュレーション結果である｡時刻設定用の信号の TIMER_SET はあらかじめ 1(01)(時刻設定用)に設定されている。TIMER_SET が 1(01)のときの時計用のカウンタは現在の値を保ったまま停止する。HOUR_SET、MIN_SET、SEC_SET を 1 にすることで時、分、秒ごとのカウンタだけを早送りしている。このとき、60 または 24(時の場合)になったとしても上位の桁への桁上がりは起こらないようになっている。

図 5.6、図 5.7 はアラームタイマーのシミュレーション結果である。TIMER_SET を 10 にすることによりアラームの設定ができるようになっている。その間、時計は通常通りの動作をしている。アラーム設定時のみ HOUR_SET、MIN_SET でアラームタイマーの時、分を設定できる。TIMER_SET を 11 にした場合は時計の時、分とアラームタイマーの時、分が同じ値のとき ALARM_OUT が 1 になりアラームがオンとなる。ALARM_STOP が 1 になることにより ALARM_OUT は 0 になりアラームはオフとなる。

図 5.8、図 5.9 はカレンダーのシミュレーション結果である。2000 年は 100 の倍数であるが、400 の倍数でもあるため、うるう年となるから 2 月は 29 日まである。その他の月は 1 月、3 月～12 月まではそれぞれ 31 日(1 月)、31 日(3 月)、30 日(4 月)、31 日(5 月)、30 日(6 月)、31 日(7 月)、31 日(8 月)、30 日(9 月)、31 日(10 月)、30 日(11 月)、31 日(12 月)となっていて実際の日数と一致する。曜日についても 2000 年 1 月 1 日は 6(011)で土曜日、2004 年 2 月 1 日は 0(000)で日曜日となっており、これも実際のものと一致する。

図 5.10 はカレンダー設定のシミュレーション結果である。YMD_MODE(カレンダーの設定用)の信号が 0(00)、1(01)、2(10)、3(11)で普段のカレンダー動作、日設定、月設定、年設定用の状態に対応している。YMD_MODE が設定用の信号になっているとき YMD_SWITCH を 1 にするとそれぞれの信号に応じた日、月、年を早送りできる。

図 5.11 は時計とカレンダーの全体動作のシミュレーション結果である。日付が変わるときの曜日の変化、その他のカウント値のリセット動作も全体として正常にできている。

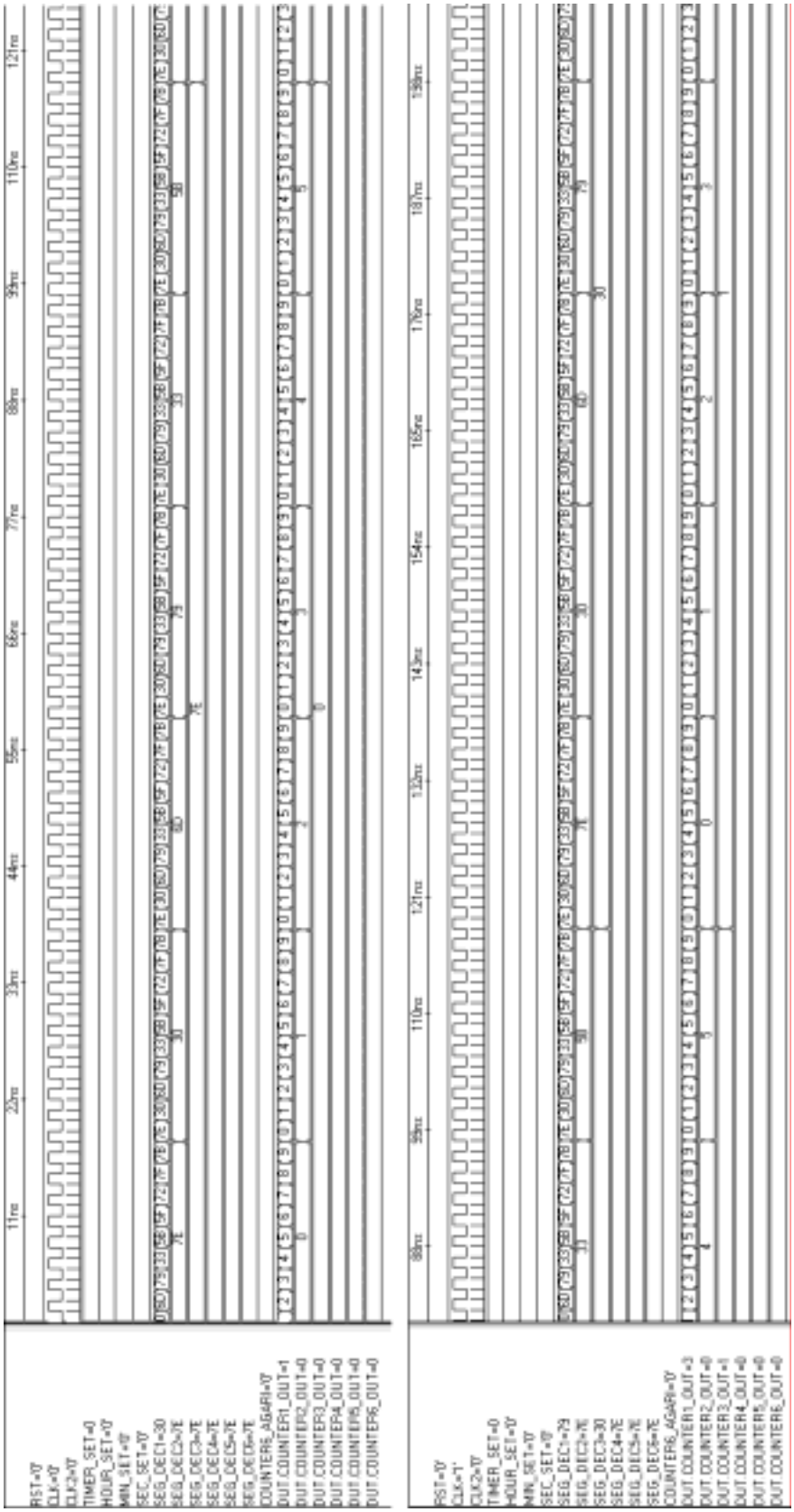


図 5.1　時計のシミュレーション結果 1

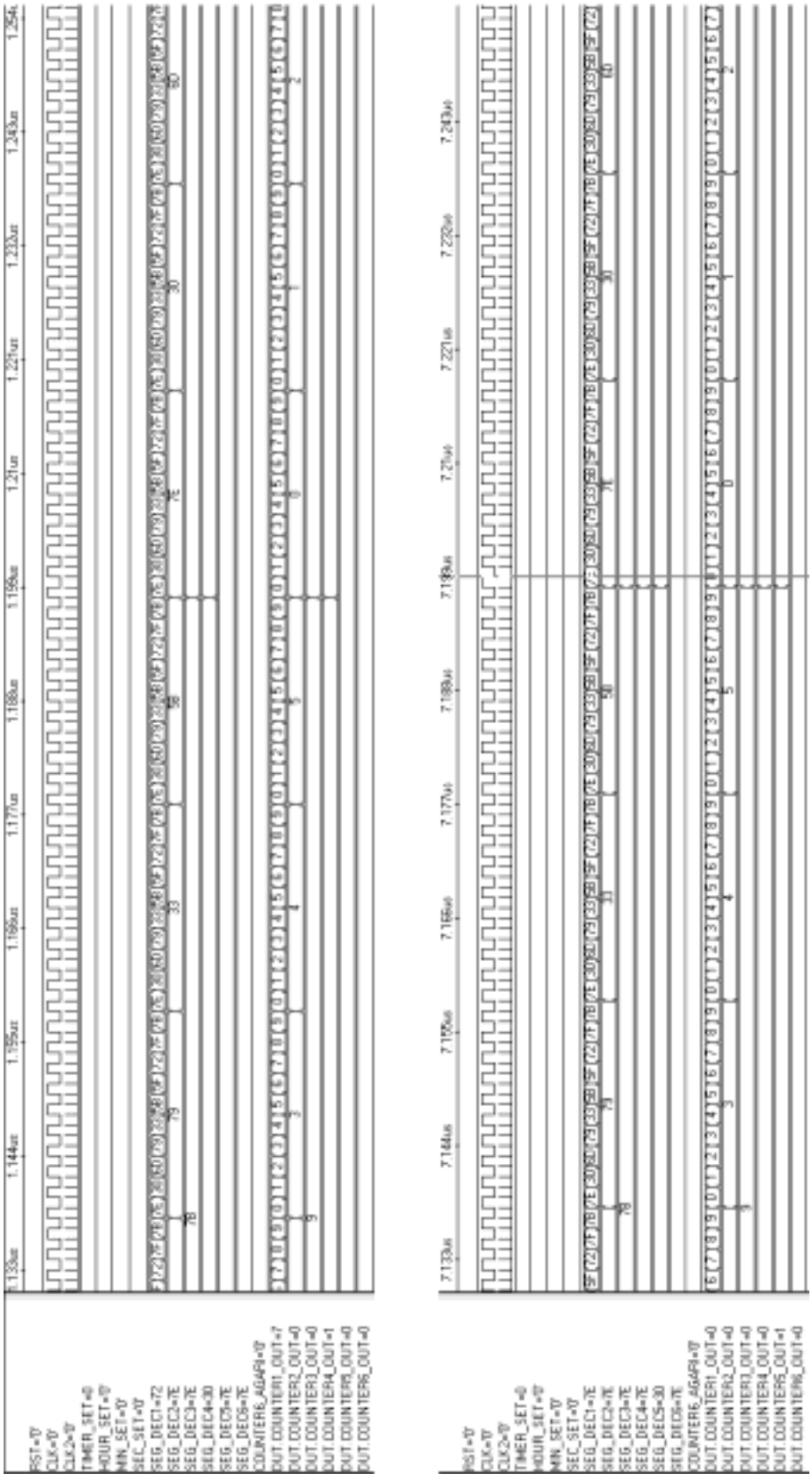


図 5.2　時計のシミュレーション結果 2

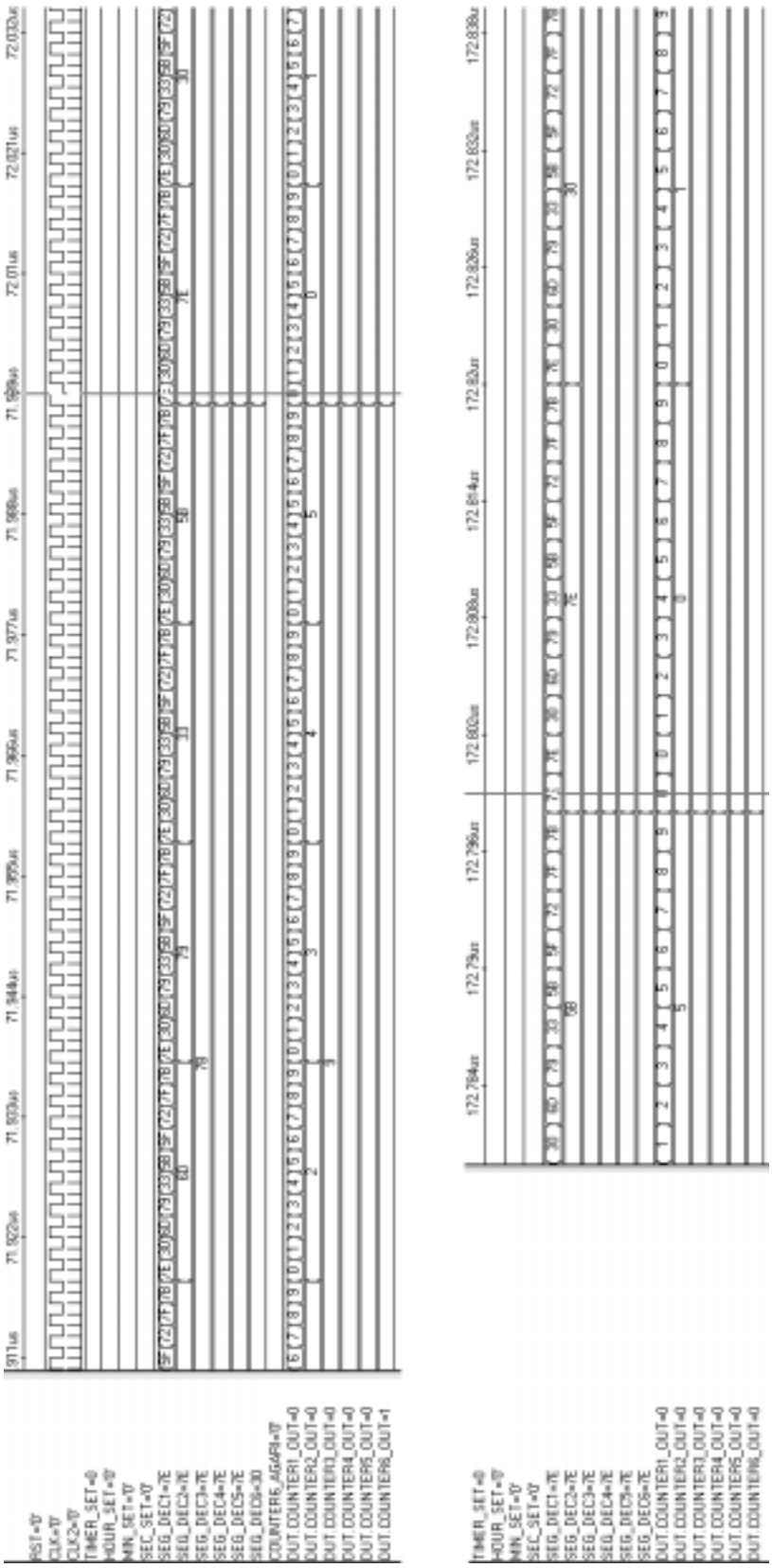


図 5.3　時計のシミュレーション結果 3

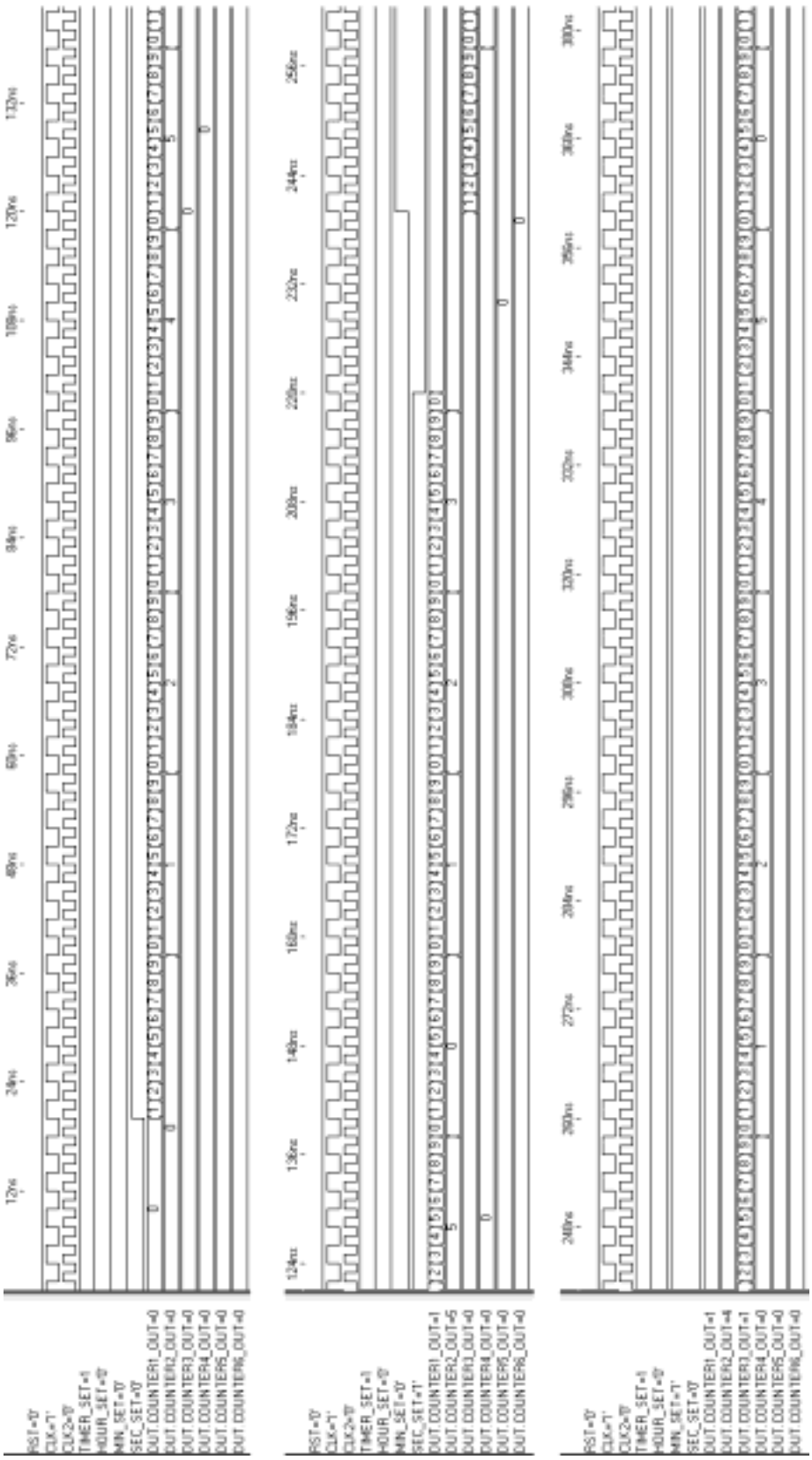


図 5.4　時刻設定のシミュレーション結果 1

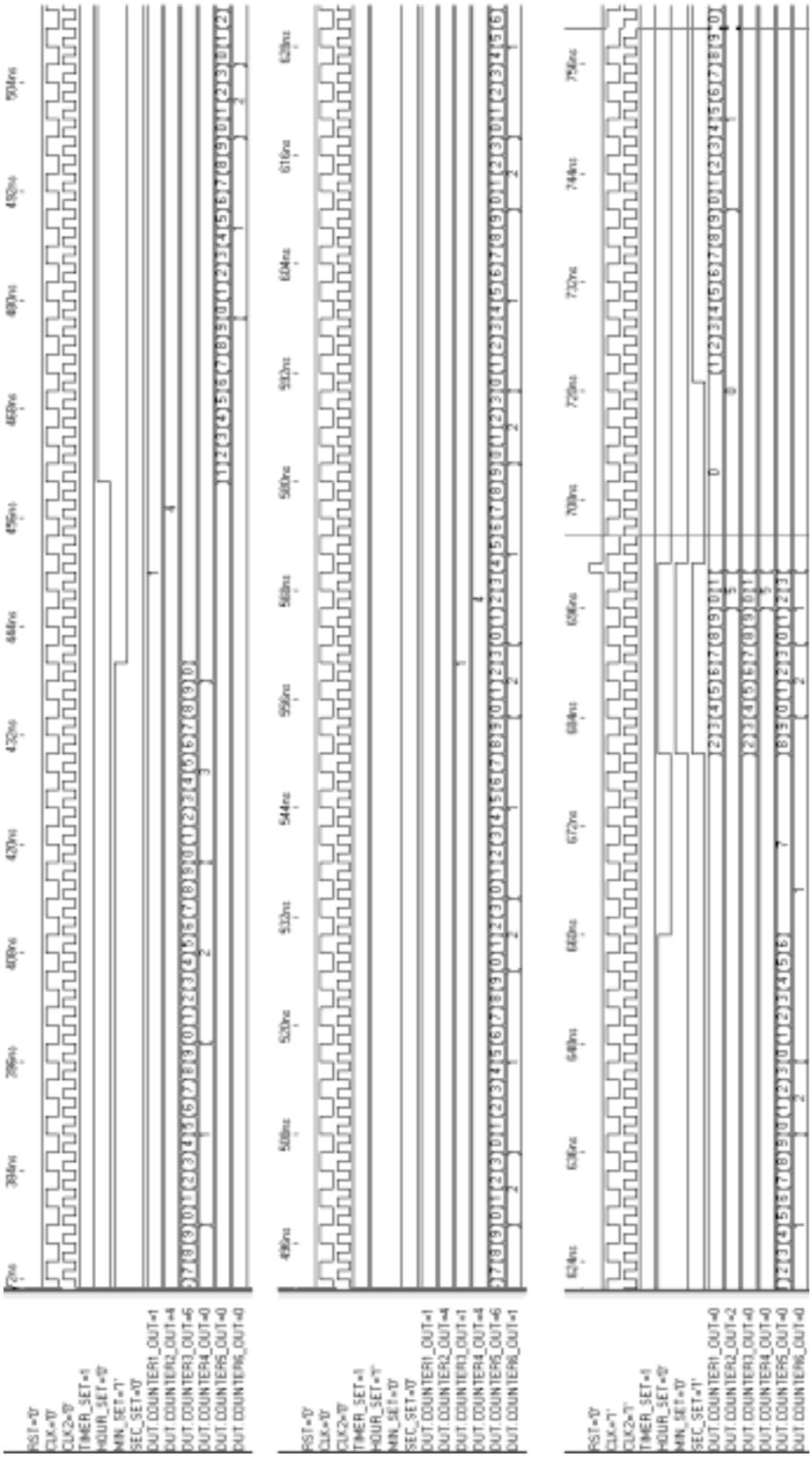


図 5.5　時刻設定のシミュレーション結果 2

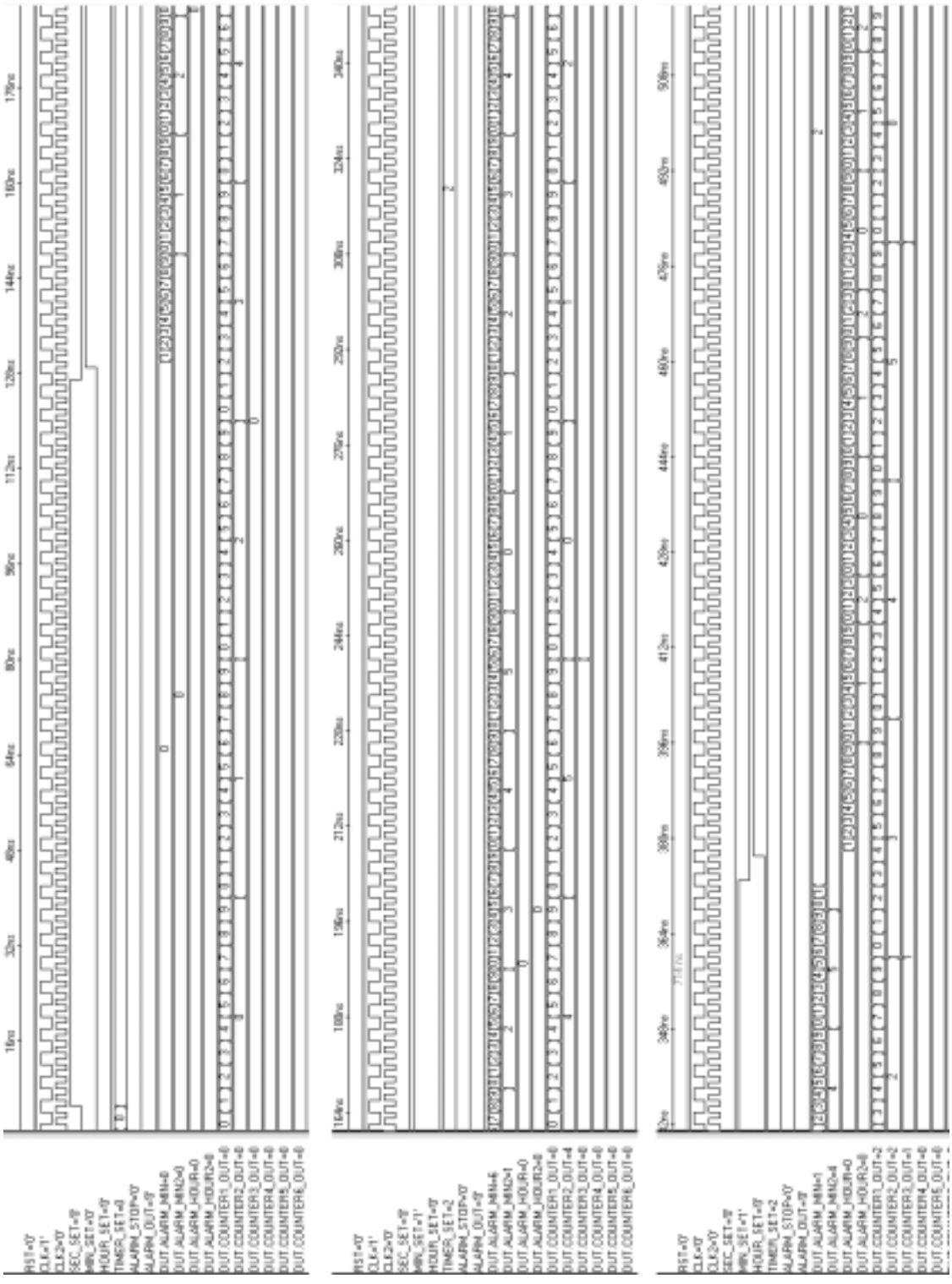


図 5.6　アラームタイマーのシミュレーション結果 1

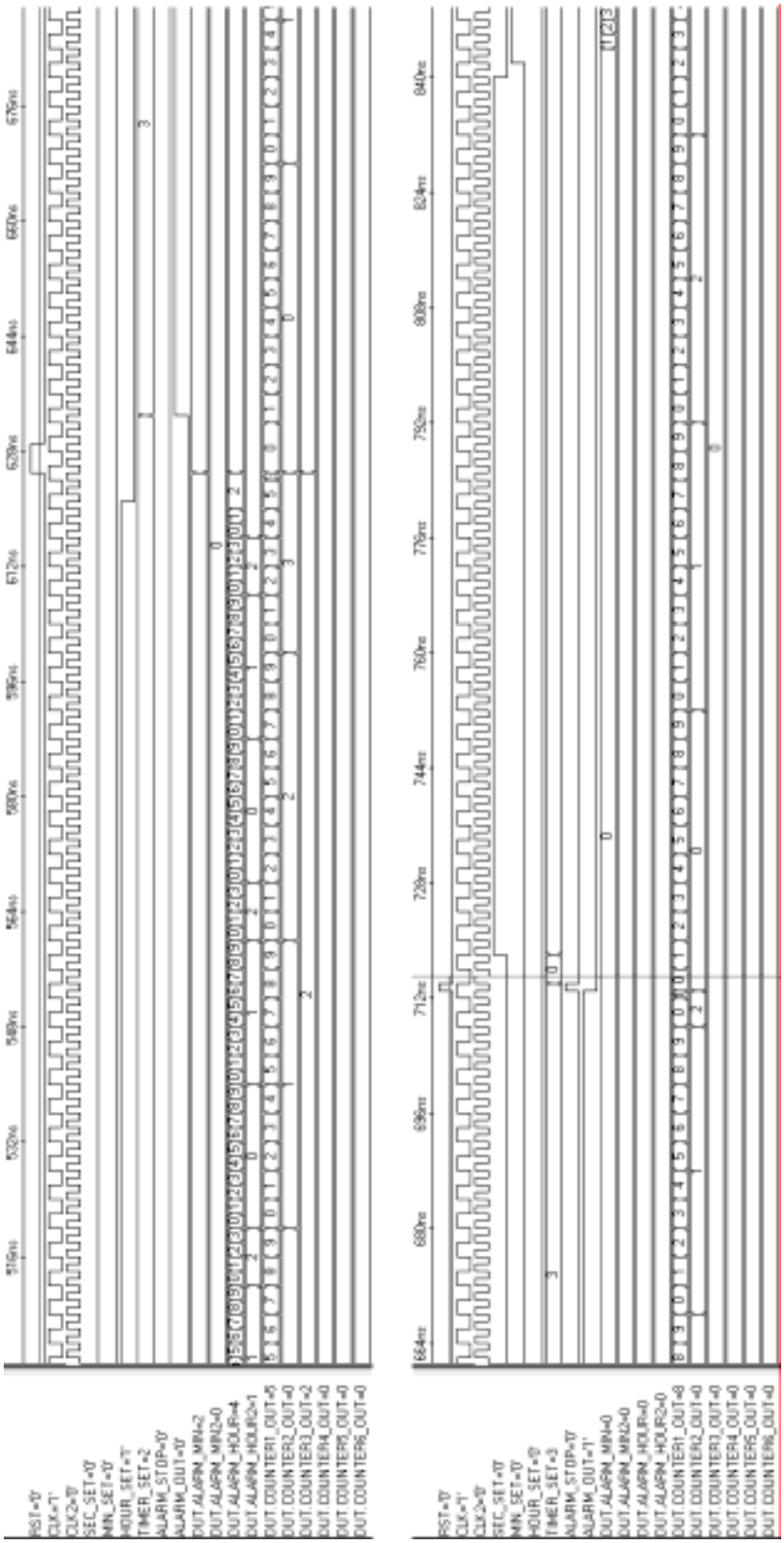


図 5.7　アラームタイマーのシミュレーション結果 2

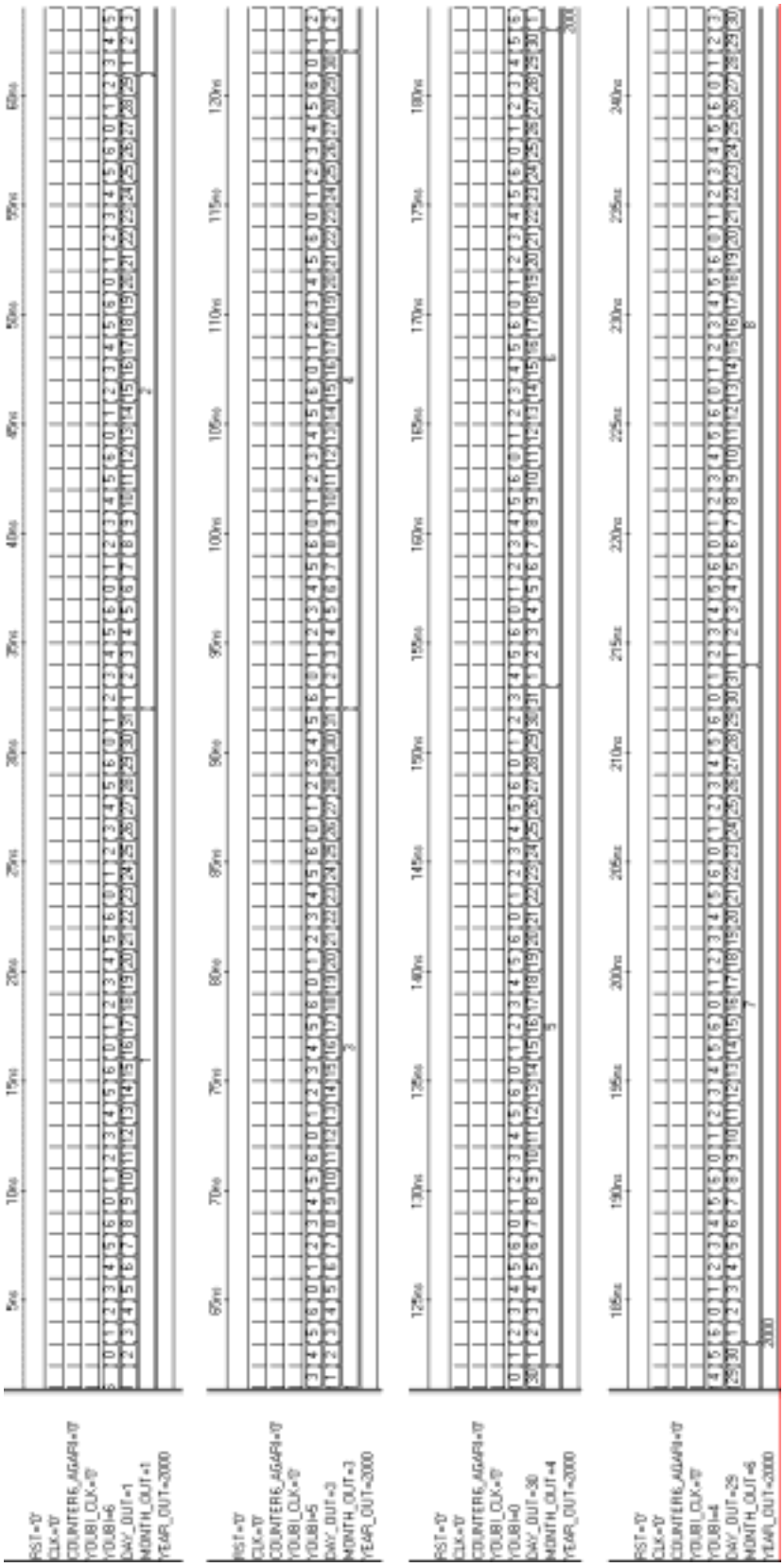


図 5.8　カレンダーのシミュレーション結果の一部 1

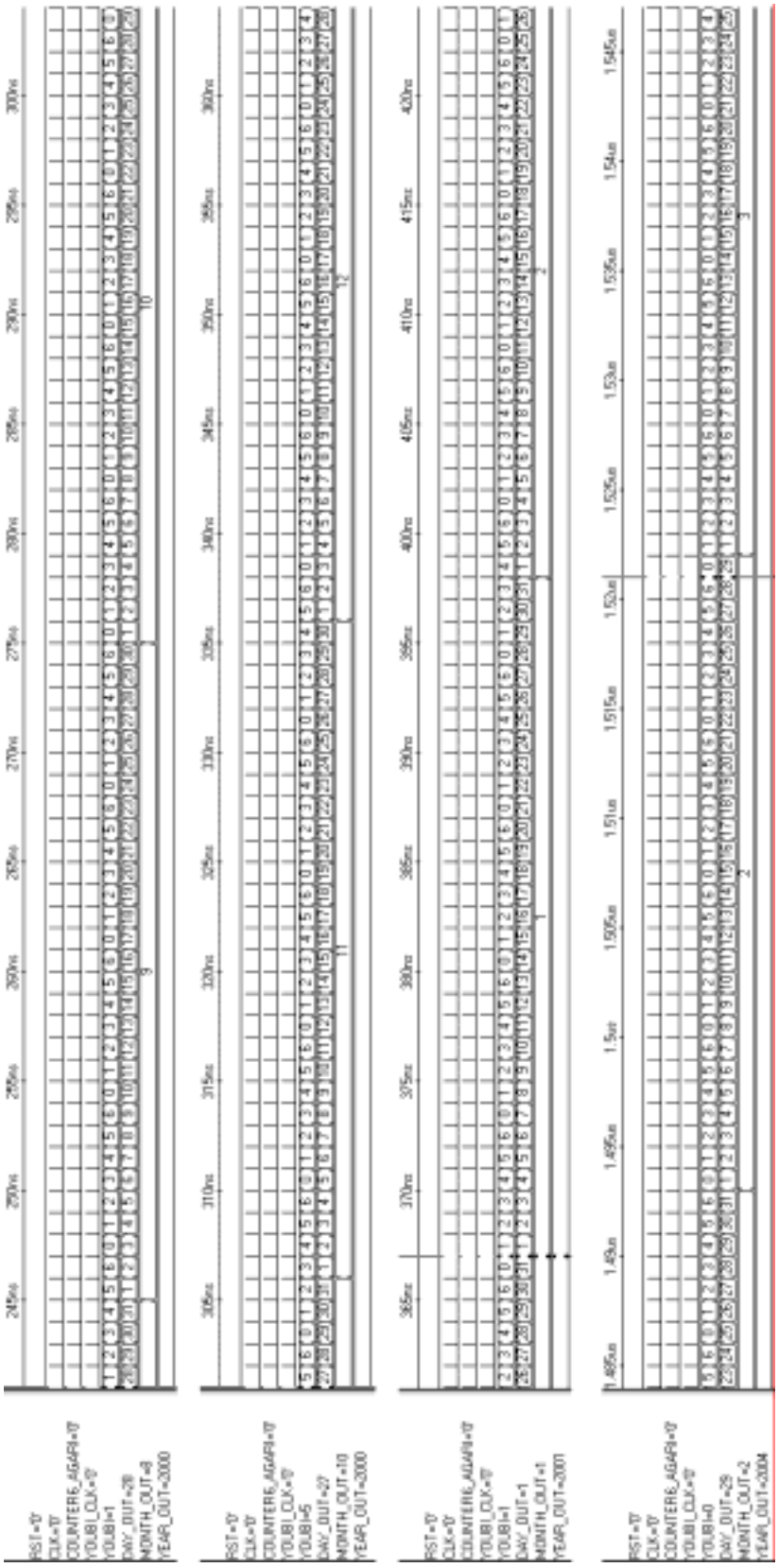


図 5.9　カレンダーのシミュレーション結果の一部 2

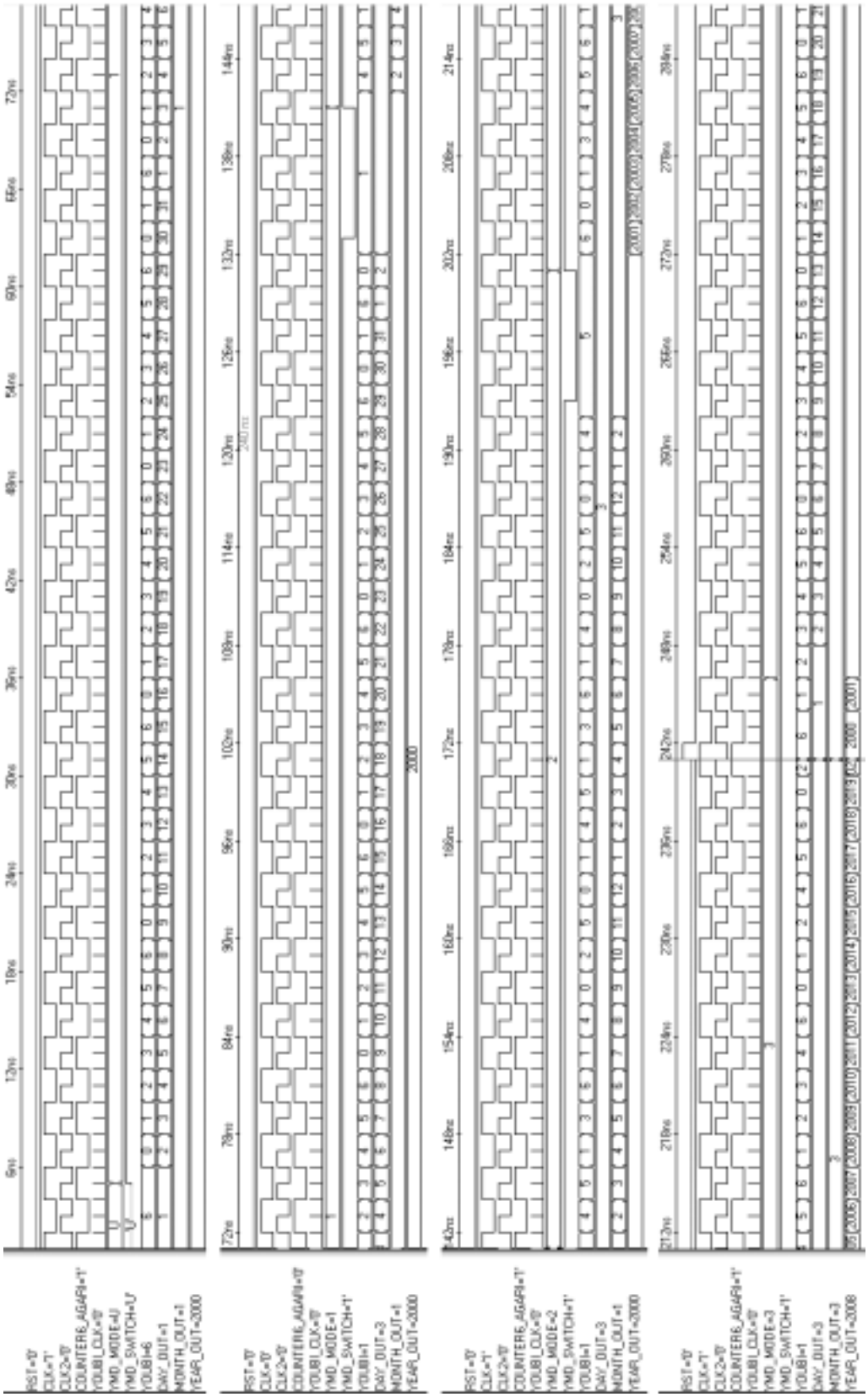


図 5.10　カレンダー設定のシミュレーション結果

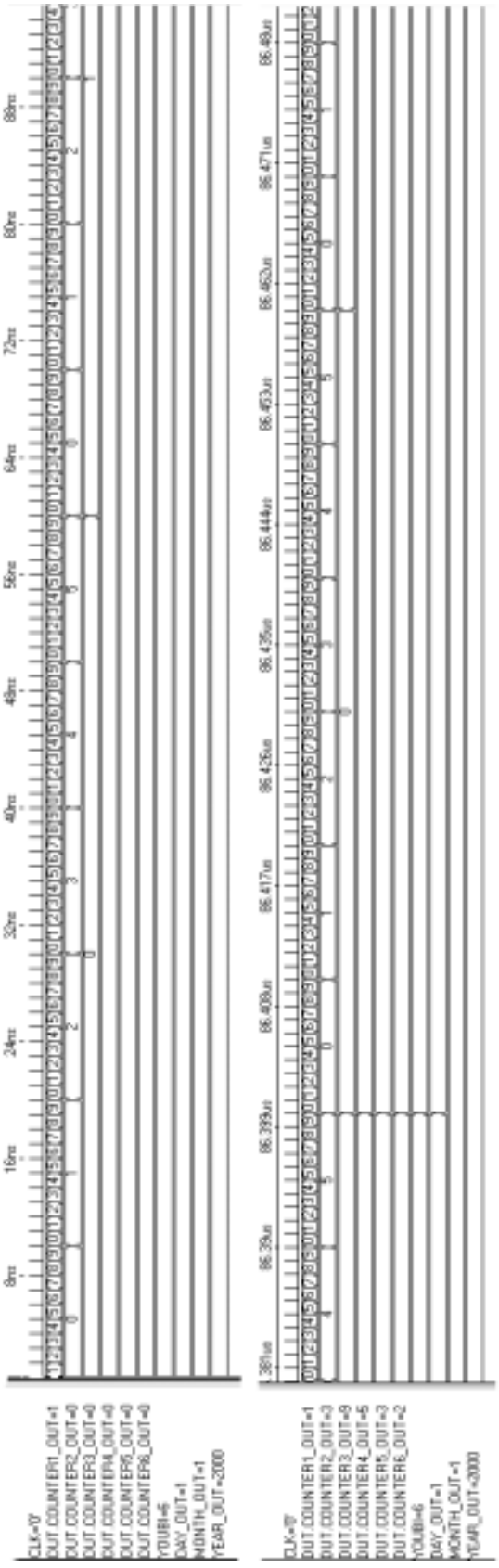


図 5.11　時計とカレンダーの全体動作のシミュレーション結果

5.2　考察と感想

時計本体の動作やその早送りの動作は時計として問題のないシミュレーション結果が得られた。アラームやカレンダーについても正常な動作が確認できた。しかし、問題もある。それはシミュレーション上では遅延が 0 であるため、この回路の場合、非同期式特有の遅延が上位に蓄積する現象が確認しづらいことである。もし、遅延があまりにも大きいようなら同期式に改良する必要性がある。

今回、取り組むことができなかったが、電波時計を設計するというアイデアもあった。電波で時刻を設定すれば手動で設定する必要もないし、ズレも最小限で抑えることができる。また機会があれば、この問題点の修正と新しい機能の追加に挑戦してみたい。

謝辞

本研究の全過程を通じて懇切丁寧な指導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、原　央教授、橘　昌良助教授に感謝します。御助言を頂きました電子・光システム工学コース修士課程の木村　知史氏、松村 暢也氏、石川 純平氏および原研究室、橘研究室の皆様に厚く御礼申し上げます。

参考文献

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

[2] 「2002 最新 汎用ロジック・デバイス規格表」　CQ 出版社

[3] 「論理回路入門」　浜辺 隆二 著　森北出版株式会社

[4] 「日付に関するアルゴリズム」
http://www.ht-net21.ne.jp/~tomokazu/pclab/days/days.html

VHDL のソース

回路全体の VHDL のソースを以下に示す。

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


entity PTIMER is
    port (
        CLK : in std_logic;
        CLK2 : in std_logic;
        RST : in std_logic;
        TIMER_SET : in std_logic_vector(0 to 1);
        HOUR_SET : in std_logic;
        MIN_SET : in std_logic;
        SEC_SET : in std_logic;

        SEG_DEC1 : out std_logic_vector(0 to 6);
        SEG_DEC2 : out std_logic_vector(0 to 6);
        SEG_DEC3 : out std_logic_vector(0 to 6);
        SEG_DEC4 : out std_logic_vector(0 to 6);
        SEG_DEC5 : out std_logic_vector(0 to 6);
        SEG_DEC6 : out std_logic_vector(0 to 6);

        ALARM_STOP : in std_logic;
        ALARM_OUT : out std_logic;

        YMD_MODE : in std_logic_vector(0 to 1);
-- normal=00 day=01 month=10 year=11
        YMD_SWITCH : in std_logic;
        YEAR_OUT : out integer range 0 to 11;
        MONTH_OUT : out integer range 0 to 3;
        DAY_OUT : out integer range 0 to 4;
        YOUBI : out std_logic_vector(0 to 2)
         );

end PTIMER;

architecture RTL of PTIMER is

        signal SEL1_OUT : std_logic;
        signal SEL2_OUT : std_logic;
        signal SEL3_OUT : std_logic;
```

```
        signal COUNTER2_CLK : std_logic;
        signal COUNTER3_CLK : std_logic;
        signal COUNTER4_CLK : std_logic;
        signal COUNTER5_CLK : std_logic;
        signal COUNTER6_CLK : std_logic;

        signal COUNTER1_OUT : std_logic_vector(0 to 3);
        signal COUNTER2_OUT : std_logic_vector(0 to 3);
        signal COUNTER3_OUT : std_logic_vector(0 to 3);
        signal COUNTER4_OUT : std_logic_vector(0 to 3);
        signal COUNTER5_OUT : std_logic_vector(0 to 3);
        signal COUNTER6_OUT : std_logic_vector(0 to 3);

        signal COUNTER6_AGARI : std_logic;

        signal SEG_SEL1 : std_logic_vector(0 to 3);
        signal SEG_SEL2 : std_logic_vector(0 to 3);
        signal SEG_SEL3 : std_logic_vector(0 to 3);
        signal SEG_SEL4 : std_logic_vector(0 to 3);
        signal SEG_SEL5 : std_logic_vector(0 to 3);
        signal SEG_SEL6 : std_logic_vector(0 to 3);

        signal ALARM_MIN : std_logic_vector(0 to 3);
        signal ALARM_MIN2 : std_logic_vector(0 to 3);
        signal ALARM_HOUR : std_logic_vector(0 to 3);
        signal ALARM_HOUR2 : std_logic_vector(0 to 3);
        signal ALARM_MIN_AGARI : std_logic;
        signal ALARM_HOUR_AGARI : std_logic;
        signal ALARM_SEL : std_logic;
        signal ALM_OUT : std_logic;

        signal YEAR : integer range 0 to 11;
        signal MONTH : integer range 0 to 3;
        signal MAX_DAYS : integer range 0 to 4;
        signal DAY : integer range 0 to 4;
        signal MONTH_KETA_AGARI : std_logic;
        signal DAY_KETA_AGARI : std_logic;
        signal TMP_YEAR : integer range 0 to 11 := 0;
        signal TMP_MONTH : integer range 0 to 3 := 0;
        signal TMP_YOUBI : integer range 0 to 2 := 0;
        signal YOUBI_CLK : std_logic;

begin

-- SELECTER1(SEC)
-- NORMAL=00 TIMERSET=01 ALARMSET=10 NONE=11
process ( TIMER_SET, SEC_SET, CLK, CLK2 ) begin
        if ( TIMER_SET = "01" ) then
```

```
            if ( SEC_SET = '0' ) then
                SEL1_OUT <= '0';
            elsif ( SEC_SET <= '1' ) then
                SEL1_OUT <= CLK2;
            end if;
        elsif ( TIMER_SET = "00" ) then
            SEL1_OUT <= CLK;
        end if;
end process;

-- SELECTER2(MIN)
process ( TIMER_SET, MIN_SET, COUNTER3_CLK, CLK2 ) begin
        if ( TIMER_SET = "01" ) then
            if ( MIN_SET = '0' ) then
                SEL2_OUT <= '0';
            elsif ( MIN_SET = '1' ) then
                SEL2_OUT <= CLK2;
            end if;
        elsif ( TIMER_SET = "00" ) then
            SEL2_OUT <= COUNTER3_CLK;
        end if;
end process;

-- SELECTER3(HOUR)
process ( TIMER_SET, HOUR_SET, COUNTER5_CLK, CLK2 ) begin
        if ( TIMER_SET = "01" ) then
            if ( HOUR_SET = '0') then
                SEL3_OUT <= '0';
            elsif ( HOUR_SET = '1' ) then
                SEL3_OUT <= CLK2;
            end if;
        elsif ( TIMER_SET = "00" ) then
            SEL3_OUT <= COUNTER5_CLK;
        end if;
end process;

-- BCD COUNTER1
process ( RST, SEL1_OUT, COUNTER1_OUT ) begin
        if ( RST = '1' ) then
            COUNTER1_OUT <= "0000";
            COUNTER2_CLK <= '0';
        elsif ( COUNTER1_OUT = "1010" ) then
            COUNTER1_OUT <= "0000";
            COUNTER2_CLK <= '1';
        else
            COUNTER2_CLK <= '0';
        end if;
        if ( SEL1_OUT'event and SEL1_OUT = '1' ) then
```

```
            COUNTER1_OUT <= COUNTER1_OUT + '1';
        end if;
end process;

-- BCD COUNTER2
process ( RST, COUNTER2_CLK, COUNTER2_OUT ) begin
        if ( RST = '1' ) then
            COUNTER2_OUT <= "0000";
            COUNTER3_CLK <= '0';
        elsif ( COUNTER2_OUT = "0110") then
            COUNTER2_OUT <= "0000";
            COUNTER3_CLK <= '1';
        else
            COUNTER3_CLK <= '0';
        end if;
        if ( COUNTER2_CLK'event and COUNTER2_CLK = '1' ) then
            COUNTER2_OUT <= COUNTER2_OUT + '1';
        end if;
end process;

-- BCD COUNTER3
process ( RST, SEL2_OUT, COUNTER3_OUT ) begin
        if ( RST = '1' ) then
            COUNTER3_OUT <= "0000";
            COUNTER4_CLK <= '0';
        elsif ( COUNTER3_OUT = "1010" ) then
            COUNTER3_OUT <= "0000";
            COUNTER4_CLK <= '1';
        else
            COUNTER4_CLK <= '0';
        end if;
        if ( SEL2_OUT'event and SEL2_OUT = '1' ) then
            COUNTER3_OUT <= COUNTER3_OUT + '1';
        end if;
end process;

-- BCD COUNTER4
process ( RST, COUNTER4_CLK, COUNTER4_OUT) begin
        if ( RST = '1' ) then
            COUNTER4_OUT <= "0000";
            COUNTER5_CLK <= '0';
        elsif ( COUNTER4_OUT = "0110" ) then
            COUNTER4_OUT <= "0000";
            COUNTER5_CLK <= '1';
        else
            COUNTER5_CLK <= '0';
        end if;
        if ( COUNTER4_CLK'event and COUNTER4_CLK = '1' ) then
```

```
            COUNTER4_OUT <= COUNTER4_OUT + '1';
        end if;
end process;

-- BCD COUNTER5
process ( RST, SEL3_OUT, COUNTER5_OUT, COUNTER6_OUT ) begin
        if ( RST = '1' ) then
            COUNTER5_OUT <= "0000";
            COUNTER6_CLK <= '0';
        elsif ( COUNTER5_OUT = "1010" or ( COUNTER5_OUT = "0100" and COUNTER6_OUT
= "0010" ) ) then
            COUNTER5_OUT <= "0000";
            COUNTER6_CLK <= '1';
        else
            COUNTER6_CLK <= '0';
        end if;
        if ( SEL3_OUT'event and SEL3_OUT = '1' ) then
            COUNTER5_OUT <= COUNTER5_OUT + '1';
        end if;
end process;

-- BCD COUNTER6
process ( RST, COUNTER6_OUT, COUNTER6_CLK ) begin
        if ( RST = '1' ) then
            COUNTER6_OUT <= "0000";
            COUNTER6_AGARI <= '0';
        elsif ( COUNTER6_OUT = "0011" ) then
            COUNTER6_OUT <= "0000";
            COUNTER6_AGARI <= '1';
        else
            COUNTER6_AGARI <= '0';
        end if;
        if ( COUNTER6_CLK'event and COUNTER6_CLK = '1' ) then
            COUNTER6_OUT <= COUNTER6_OUT + '1';
        end if;
end process;

-- ALARM
process ( RST, CLK2, TIMER_SET, MIN_SET, ALARM_MIN) begin
        if ( RST = '1' ) then
            ALARM_MIN <= "0000";
        elsif ( TIMER_SET = "10" and MIN_SET = '1' and CLK2'event and CLK2 = '1' )
then
            ALARM_MIN <= ALARM_MIN + '1';
        end if;
        if ( ALARM_MIN = "1010" ) then
            ALARM_MIN <= "0000";
            ALARM_MIN_AGARI <= '1';
```

```
        else
            ALARM_MIN_AGARI <= '0';
        end if;
end process;

process ( RST, ALARM_MIN_AGARI, TIMER_SET, ALARM_MIN2 ) begin
        if ( RST = '1' ) then
            ALARM_MIN2 <= "0000";
        elsif ( ALARM_MIN_AGARI = '1' and ALARM_MIN_AGARI'event ) then
            ALARM_MIN2 <= ALARM_MIN2 + '1';
        end if;
        if ( ALARM_MIN2 = "0110") then
            ALARM_MIN2 <= "0000";
        end if;
end process;

process ( RST, CLK2, TIMER_SET, HOUR_SET, ALARM_HOUR, ALARM_HOUR2 ) begin
        if ( RST = '1' ) then
            ALARM_HOUR <= "0000";
        elsif ( TIMER_SET = "10" and HOUR_SET = '1' and CLK2'event and CLK2 = '1' )
then
            ALARM_HOUR <= ALARM_HOUR + '1';
        end if;
        if ( ALARM_HOUR = "1010" ) then
            ALARM_HOUR <= "0000";
            ALARM_HOUR_AGARI <= '1';
        elsif ( ALARM_HOUR = "0100" and ALARM_HOUR2 = "0010" ) then
            ALARM_HOUR <= "0000";
            ALARM_HOUR_AGARI <= '0';
        else
            ALARM_HOUR_AGARI <= '0';
        end if;
end process;

process ( RST, ALARM_HOUR_AGARI, TIMER_SET, ALARM_HOUR, ALARM_HOUR2 ) begin
        if ( RST = '1' ) then
            ALARM_HOUR2 <= "0000";
        elsif ( ALARM_HOUR_AGARI = '1' and ALARM_HOUR_AGARI'event ) then
            ALARM_HOUR2 <= ALARM_HOUR2 + '1';
        end if;
        if ( ALARM_HOUR = "0100" and ALARM_HOUR2 = "0010" ) then
            ALARM_HOUR2 <= "0000";
        end if;
end process;
ALARM_OUT <= ALM_OUT;
process ( TIMER_SET, ALARM_STOP, ALARM_MIN2, ALARM_MIN, ALARM_HOUR2, ALARM_HOUR,
COUNTER6_OUT, COUNTER5_OUT, COUNTER4_OUT, COUNTER3_OUT, ALM_OUT ) begin
ALARM_SEL <= not( (ALARM_HOUR2(0) xor COUNTER6_OUT(0)) and (ALARM_HOUR2(1) xor
```

```
COUNTER6_OUT(1)) and (ALARM_HOUR2(2) xor COUNTER6_OUT(2)) and (ALARM_HOUR2(3) xor
COUNTER6_OUT(3)) and (ALARM_HOUR(0) xor COUNTER5_OUT(0)) and (ALARM_HOUR(1) xor
COUNTER5_OUT(1)) and (ALARM_HOUR(2) xor COUNTER5_OUT(2)) and (ALARM_HOUR(3) xor
COUNTER5_OUT(3)) and (ALARM_MIN2(0) xor COUNTER4_OUT(0)) and (ALARM_MIN2(1) xor
COUNTER4_OUT(1)) and (ALARM_MIN2(2) xor COUNTER4_OUT(2)) and (ALARM_MIN2(3) xor
COUNTER4_OUT(3)) and (ALARM_MIN(0) xor COUNTER3_OUT(0)) and (ALARM_MIN(1) xor
COUNTER3_OUT(1)) and (ALARM_MIN(2) xor COUNTER3_OUT(2)) and (ALARM_MIN(3) xor
COUNTER3_OUT(3)) );
        if ( ALARM_SEL = '1' and TIMER_SET = "11" ) then
            ALM_OUT <= '1';
        else
            ALM_OUT <= '0';
        end if;
        if ( ALARM_SEL = '0' and TIMER_SET = "11" and ALM_OUT = '1' ) then
            ALM_OUT <= '0';
        end if;
        if ( ALARM_STOP = '1' ) then
            ALM_OUT <= '0';
        end if;
end process;

-- FOR SEC
-- BCD to 7SEG DECORDER1
process ( TIMER_SET, SEG_SEL1, COUNTER1_OUT ) begin
        if ( TIMER_SET="10" ) then
            SEG_SEL1 <= "0000";
        else
            SEG_SEL1 <= COUNTER1_OUT;
        end if;
        case SEG_SEL1 is
            when "0000" => SEG_DEC1 <= "1111110";
            when "0001" => SEG_DEC1 <= "0110000";
            when "0010" => SEG_DEC1 <= "1101101";
            when "0011" => SEG_DEC1 <= "1111001";
            when "0100" => SEG_DEC1 <= "0110011";
            when "0101" => SEG_DEC1 <= "1011011";
            when "0110" => SEG_DEC1 <= "1011111";
            when "0111" => SEG_DEC1 <= "1110010";
            when "1000" => SEG_DEC1 <= "1111111";
            when "1001" => SEG_DEC1 <= "1111011";
            when "1010" => SEG_DEC1 <= "1111110";
            when others => SEG_DEC1 <= "1111110";
        end case;
end process;

-- BCD to 7SEG DECORDER2
process (TIMER_SET, SEG_SEL2, COUNTER2_OUT, SEG_DEC2) begin
        if ( TIMER_SET="10" ) then
```

```
            SEG_SEL2 <= "0000";
        else
            SEG_SEL2 <= COUNTER2_OUT;
        end if;
        case SEG_SEL2 is
            when "0000" => SEG_DEC2 <= "1111110";
            when "0001" => SEG_DEC2 <= "0110000";
            when "0010" => SEG_DEC2 <= "1101101";
            when "0011" => SEG_DEC2 <= "1111001";
            when "0100" => SEG_DEC2 <= "0110011";
            when "0101" => SEG_DEC2 <= "1011011";
            when "0110" => SEG_DEC2 <= "1011111";
            when "0111" => SEG_DEC2 <= "1110010";
            when "1000" => SEG_DEC2 <= "1111111";
            when "1001" => SEG_DEC2 <= "1111011";
            when "1010" => SEG_DEC2 <= "1111110";
            when others => SEG_DEC2 <= "1111110";
        end case;
end process;


-- FOR MIN
-- BCD to 7SEG DECORDER3
process (TIMER_SET, SEG_SEL3, COUNTER3_OUT, SEG_DEC3) begin
        if ( TIMER_SET="10" ) then
            SEG_SEL3 <= ALARM_MIN;
        else
            SEG_SEL3 <= COUNTER3_OUT;
        end if;
        case SEG_SEL3 is
            when "0000" => SEG_DEC3 <= "1111110";
            when "0001" => SEG_DEC3 <= "0110000";
            when "0010" => SEG_DEC3 <= "1101101";
            when "0011" => SEG_DEC3 <= "1111001";
            when "0100" => SEG_DEC3 <= "0110011";
            when "0101" => SEG_DEC3 <= "1011011";
            when "0110" => SEG_DEC3 <= "1011111";
            when "0111" => SEG_DEC3 <= "1110010";
            when "1000" => SEG_DEC3 <= "1111111";
            when "1001" => SEG_DEC3 <= "1111011";
            when "1010" => SEG_DEC3 <= "1111110";
            when others => SEG_DEC3 <= "1111110";
        end case;
end process;

-- BCD to 7SEG DECORDER4
process (TIMER_SET, SEG_SEL4, COUNTER4_OUT, SEG_DEC4) begin
        if ( TIMER_SET="10" ) then
```

```
            SEG_SEL4 <= ALARM_MIN2;
        else
            SEG_SEL4 <= COUNTER4_OUT;
        end if;
        case SEG_SEL4 is
            when "0000" => SEG_DEC4 <= "1111110";
            when "0001" => SEG_DEC4 <= "0110000";
            when "0010" => SEG_DEC4 <= "1101101";
            when "0011" => SEG_DEC4 <= "1111001";
            when "0100" => SEG_DEC4 <= "0110011";
            when "0101" => SEG_DEC4 <= "1011011";
            when "0110" => SEG_DEC4 <= "1011111";
            when "0111" => SEG_DEC4 <= "1110010";
            when "1000" => SEG_DEC4 <= "1111111";
            when "1001" => SEG_DEC4 <= "1111011";
            when "1010" => SEG_DEC4 <= "1111110";
            when others => SEG_DEC4 <= "1111110";
        end case;
end process;

-- FOR HOUR
-- BCD to 7SEG DECORDER5
process (TIMER_SET, SEG_SEL5, COUNTER5_OUT, SEG_DEC5) begin
        if ( TIMER_SET="10" ) then
            SEG_SEL5 <= ALARM_HOUR;
        else
            SEG_SEL5 <= COUNTER5_OUT;
        end if;
        case SEG_SEL5 is
            when "0000" => SEG_DEC5 <= "1111110";
            when "0001" => SEG_DEC5 <= "0110000";
            when "0010" => SEG_DEC5 <= "1101101";
            when "0011" => SEG_DEC5 <= "1111001";
            when "0100" => SEG_DEC5 <= "0110011";
            when "0101" => SEG_DEC5 <= "1011011";
            when "0110" => SEG_DEC5 <= "1011111";
            when "0111" => SEG_DEC5 <= "1110010";
            when "1000" => SEG_DEC5 <= "1111111";
            when "1001" => SEG_DEC5 <= "1111011";
            when "1010" => SEG_DEC5 <= "1111110";
            when others => SEG_DEC5 <= "1111110";
        end case;
end process;

-- BCD to 7SEG DECORDER6
process (TIMER_SET, SEG_SEL6, COUNTER6_OUT, SEG_DEC6) begin
        if ( TIMER_SET="10" ) then
            SEG_SEL6 <= ALARM_MIN2;
```

```
        else
            SEG_SEL6 <= COUNTER6_OUT;
        end if;
        case SEG_SEL6 is
            when "0000" => SEG_DEC6 <= "1111110";
            when "0001" => SEG_DEC6 <= "0110000";
            when "0010" => SEG_DEC6 <= "1101101";
            when "0011" => SEG_DEC6 <= "1111001";
            when "0100" => SEG_DEC6 <= "0110011";
            when "0101" => SEG_DEC6 <= "1011011";
            when "0110" => SEG_DEC6 <= "1011111";
            when "0111" => SEG_DEC6 <= "1110010";
            when "1000" => SEG_DEC6 <= "1111111";
            when "1001" => SEG_DEC6 <= "1111011";
            when "1010" => SEG_DEC6 <= "1111110";
            when others => SEG_DEC6 <= "1111110";
        end case;
end process;

-- CALENDAR
process ( CLK, CLK2, YMD_MODE ) begin
        if ( YMD_MODE = "01" or YMD_MODE = "10" or YMD_MODE = "11" ) then
            YOUBI_CLK <= CLK2;
        else
            YOUBI_CLK <= CLK;
        end if;
end process;

-- days of MONTH
process ( MONTH, YEAR  ) begin
        if ( MONTH = 2 and ( ( YEAR mod 4 = 0 and YEAR mod 100 /= 0 ) or ( YEAR mod
400 = 0 ) ) ) then
            MAX_DAYS <= 29;
        elsif ( MONTH = 2 and not ( ( YEAR mod 4 = 0 and YEAR mod 100 /= 0 ) or ( YEAR
mod 400 = 0 ) ) ) then
            MAX_DAYS <= 28;
        elsif ( MONTH = 1 or MONTH = 3 or MONTH = 5 or MONTH = 7 or MONTH = 8 or MONTH
= 10 or MONTH = 12 ) then
            MAX_DAYS <= 31;
        elsif ( MONTH = 4 or MONTH = 6 or MONTH = 9 or MONTH = 11 ) then
            MAX_DAYS <= 30;
        end if;
end process;

-- DAY
process ( CLK2, DAY, RST, YMD_MODE, MAX_DAYS, DAY_KETA_AGARI, COUNTER6_AGARI,
YMD_SWITCH ) begin
        if ( RST = '1' or ( YMD_MODE = "01" and DAY = MAX_DAYS + 1 ) ) then
```

```
            DAY <= 1;
        elsif ( YMD_MODE = "00" and DAY = MAX_DAYS + 1 ) then
            DAY <= 1;
            DAY_KETA_AGARI <= '1';
        else
            DAY_KETA_AGARI <= '0';
        end if;
        if ( ( YMD_MODE = "00" and COUNTER6_AGARI = '1' and COUNTER6_AGARI'event )
or ( YMD_MODE = "01" and YMD_SWITCH = '1' and CLK2'event and CLK2 = '1' ) ) then
            DAY <= DAY + 1;
        end if;
        DAY_OUT <= DAY;
end process;

-- MONTH
process ( CLK2, RST, YMD_MODE, MONTH, DAY_KETA_AGARI, YMD_SWITCH ) begin
        if ( RST = '1' or ( YMD_MODE = "10" and MONTH = 13 ) ) then
            MONTH <= 1;
        elsif ( YMD_MODE = "00" and MONTH = 13 ) then
            MONTH <= 1;
            MONTH_KETA_AGARI <= '1';
        else
            MONTH_KETA_AGARI <= '0';
        end if;
        if ( ( YMD_MODE = "00" and DAY_KETA_AGARI = '1' and DAY_KETA_AGARI'event )
or ( YMD_MODE="10" and YMD_SWITCH='1' and CLK2'event and CLK2='1') ) then
            MONTH <= MONTH + 1;
        end if;
        MONTH_OUT <= MONTH;
end process;

-- YEAR
process ( CLK2, RST, YMD_MODE, YEAR, MONTH_KETA_AGARI, YMD_SWITCH ) begin
        if ( RST = '1' or YEAR = 2100 ) then
            YEAR <= 2000;
        elsif ( ( YMD_MODE = "00" and MONTH_KETA_AGARI = '1' and
MONTH_KETA_AGARI'event ) or ( YMD_MODE = "11" and YMD_SWITCH = '1' and CLK2'event
and CLK2 = '1' ) ) then
            YEAR <= YEAR + 1;
        end if;
        YEAR_OUT <= YEAR;
end process;

-- YOUBI
process ( DAY, MONTH, TMP_MONTH, TMP_YEAR ) begin
        TMP_YOUBI <= ( TMP_YEAR + TMP_YEAR / 4 - TMP_YEAR / 100 + TMP_YEAR / 400 +
( 13 * TMP_MONTH + 8 ) / 5 + DAY ) mod 7;
        if ( MONTH = 1 or MONTH = 2 ) then
```

```
            TMP_YEAR <= YEAR - 1;
            TMP_MONTH <= MONTH + 12;
        else
            TMP_YEAR <= YEAR;
            TMP_MONTH <= MONTH;
        end if;
end process;
-- Sun=000 Mon=001 Tue=010 Wed=011 Thu=100 Fri=101 Sat=110
process ( TMP_YOUBI ) begin
        if ( TMP_YOUBI = 0 ) then
            YOUBI <= "000";
        elsif ( TMP_YOUBI = 1 ) then
            YOUBI <= "001";
        elsif ( TMP_YOUBI = 2 ) then
            YOUBI <= "010";
        elsif ( TMP_YOUBI = 3 ) then
            YOUBI <= "011";
        elsif ( TMP_YOUBI = 4 ) then
            YOUBI <= "100";
        elsif ( TMP_YOUBI = 5 ) then
            YOUBI <= "101";
        elsif ( TMP_YOUBI = 6 ) then
            YOUBI <= "110";
        end if;
end process;
end RTL;
```