1　序論
1.1　はじめに
1.2　目的
1.3　概要

2　CPLD
2.1　使用環境
2.2　CPLD とは？
2.3　内部の結線状態が書き換えられるしくみ
2.4　マイコンに対する CPLD のアドバンテージ
2.5　プログラミングする手段
2.6　CPLD のまとめ

3　CCD
3.1　使用環境
3.2　特長

4　プログラム
4.1　内容
4.2　入力対応
4.3　入出力ピン設定

5　CCD 制御波形の規格確認

6　CCD の動作測定

7　まとめ

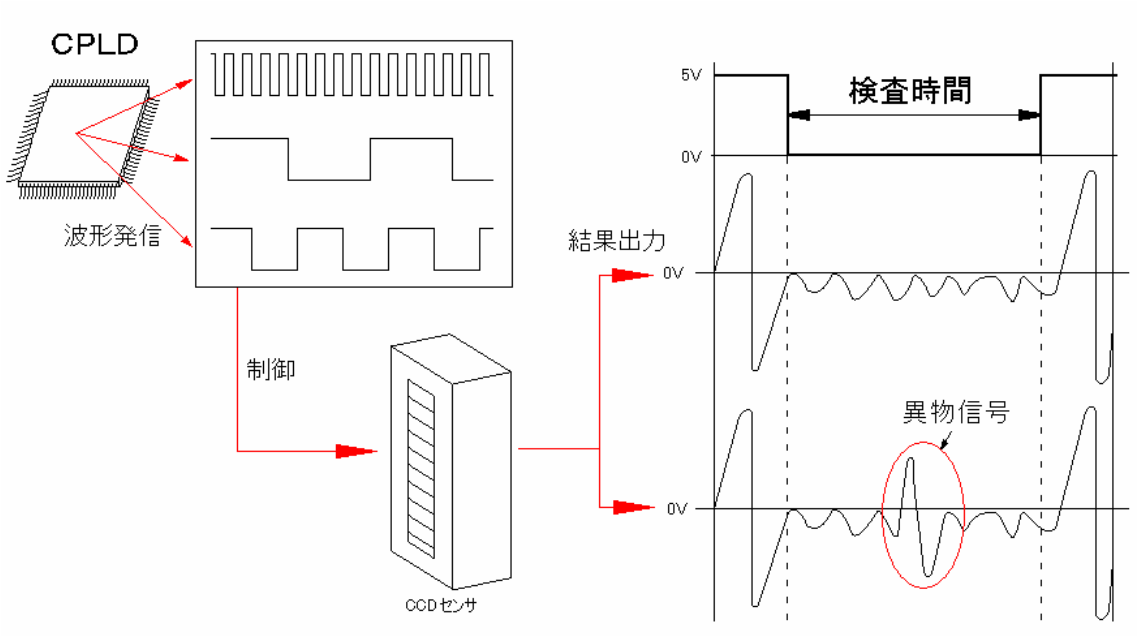
付録
参考文献
プログラム
プログラム解説

序論

目的

CCDイメージセンサは固体撮像素子のひとつで、ビデオカメラ、デジタルカメラなどに広く使用されている半導体素子である。既製のCCDユニットでは積算時間が自由に変えられず、光計測用途などでの使用が困難である。

本研究では線状の像を光電変換できる一次元イメージセンサを用い、積算時間を設定できるよう駆動信号をCPLD(Complex Programmable Logic Device)で作成し、CCDを駆動する信号を出力することを目的とする。




1. CPLD

## 2.1　使用環境

本実験ではCPLDの型番はアルテラ社製MAXIIシリーズ(EPM240T100C5)を使用した。これには供給電源として3.3V(3端子レギュレータを搭載すれば5V入力も可)を必要とし、100ピンのパッケージに、240ロジック・エレメント相当のロジック・セルが搭載されている。240ロジック・エレメントの規模はDIPタイプのロジックICに換算して数十個に相当する。

MAXIIには100本の端子があるが、使用した基板ではそのうちCN1、CN2、CN3の3つのピン・ソケットとして80本を入出力用として使うことができる。

クロックを発信するために5V、4ピン、正方形の10MHzの水晶発振器を使用した。

## 2.2　CPLDとは？

CPLDは、Complex Programmable Logic Deviceの頭文字を取ったもので、直訳すると「複雑な書き換えられるディジタル回路素子」となる。

ディジタル回路はNOT、OR、AND、NOR、NANDなどの理論ゲートと、'0'と'1'の2進数を記憶しておくことのできるレジスタからできている。

簡単に言うとCPLDは、これらの理論ゲートとレジスタがたくさん詰まったLSIである。そして、これらの接続をレゴブロックのように自由に組み替えることができる。

CPLDのCは複雑なという意味の修飾語だが、相対的な表現のため、今ではCを付けずに単に「PLD(Programmable Logic Device)」と呼ぶことも多くある。大規模なPLDの一部をFPGA(Field Programmable Gate Array)と呼ぶメーカーもある。

## 2.3　内部の結線状態が書き換えられるしくみ

本実験に使用したCPLD(MAX II)には100本もの端子があり、電源やグラウンド、入出力、内部フラッシュ・メモリ書き換え用の端子がある。

内臓のフラッシュ・メモリには、回路の結線情報などを書き込んで保存しておける。MAX IIに電源を入れると、フラッシュ・メモリから自動的に接続情報が読み出され、設計した機能をもつディジタルICに早変わりする。

CPLDの内部結線は、物理的に切り離されたり、つなぎ合わせられるわけでなく、内部スイッチがON/OFFして接続状態を変えているだけで、何度でも書き換えができる。ただし、フラッシュ・メモリの寿命によって書き換え回数には限りがある(100回まで保障されている)。

## 2.4　マイコンに対するCPLDのアドバンテージ

マイコンはプログラムを書き換えることで計算する内容を書き換えられることができるが、ディジタル回路としてとらえることができない。しかし、CPLDは回路の接続を簡単に書きかえられる。マイコンにプログラムをインストールし直すように新しい回路を書き込めば、その直後から別の計算ができるようになる。

今までマイコンで実現していたような処理も、CPLD を使うほうが手軽だ。特にディジタル回路には、複数の回路が独立して同時に動作するというマイコンにはないメリットがある。

回路が並列に動作することのメリットがあるのかというと、回路をたくさん並べて同時に複数の計算を行えば計算時間を短縮できる。しかし、この考え方はすでにマイコンにも取り込まれており、高性能なマイコンは計算を高速化するために複数の計算回路をもっている。

回路が並列に動作することのメリットは、単に処理速度の問題だけではない。

## 2.5　プログラミングする手段

本研究ではアルテラ社製統合開発環境である QuartusII により CPLD への書き込みを試みた。CPLD に書き込む手段として 2 通りの手段がある。その手段として 2.2 で述べたように CPLD 内で使える理論ゲートを結線し、それを書き込む方法。また、Verilog HDL 言語で処理内容を作成し、入出力を宣言し書き込む方法がある。

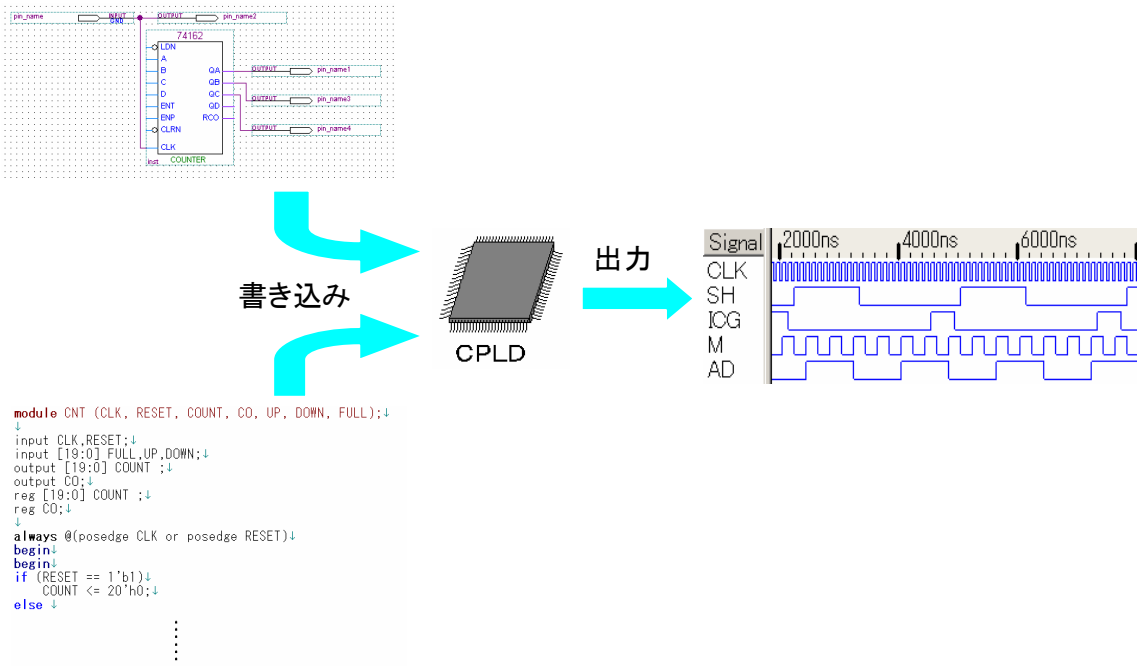



図 2-1　プログラミング手段

## 2.6　CPLD のまとめ

CPLD のアドバンテージをまとめると、マイコンのプログラムは、フローチャートの流れに添って一つ一つ進行していく。しかし、二つの処理をしたい場合にそれぞれの処理のフローチャートを書き、両方のフローチャートを同時に実行するということは基本的にはできない。二つの処理を完全に別々に実行するためには、マイコンが二つ必要になる。1 個のマイコンを使って、別々に動作する二つの処理を同時進行させたい場合は、二つの処理が一つのフローチャートの中に収まるように書き換えて、擬似的に二つの処理が同時に

動いているようにソフトウェアを改良しなければならない。

CPLD の場合は簡単で、それぞれの処理を行う回路を２セット用意して並べるだけでいい。CPLD(ディジタル回路)は並べた回路は並列に独立して動作する。一つの CPLD に何個の回路が入るかはそのときの回路の大きさによって決まるが、もし入るのであれば、10 個でも 20 個でも必要な数の回路を並べることができる。

このように、マイコンで複数の出力を扱う場合に比べて、CPLD は単純に回路を複数並べれば、これだけで並列に動作する回路を作れるというメリットがある。

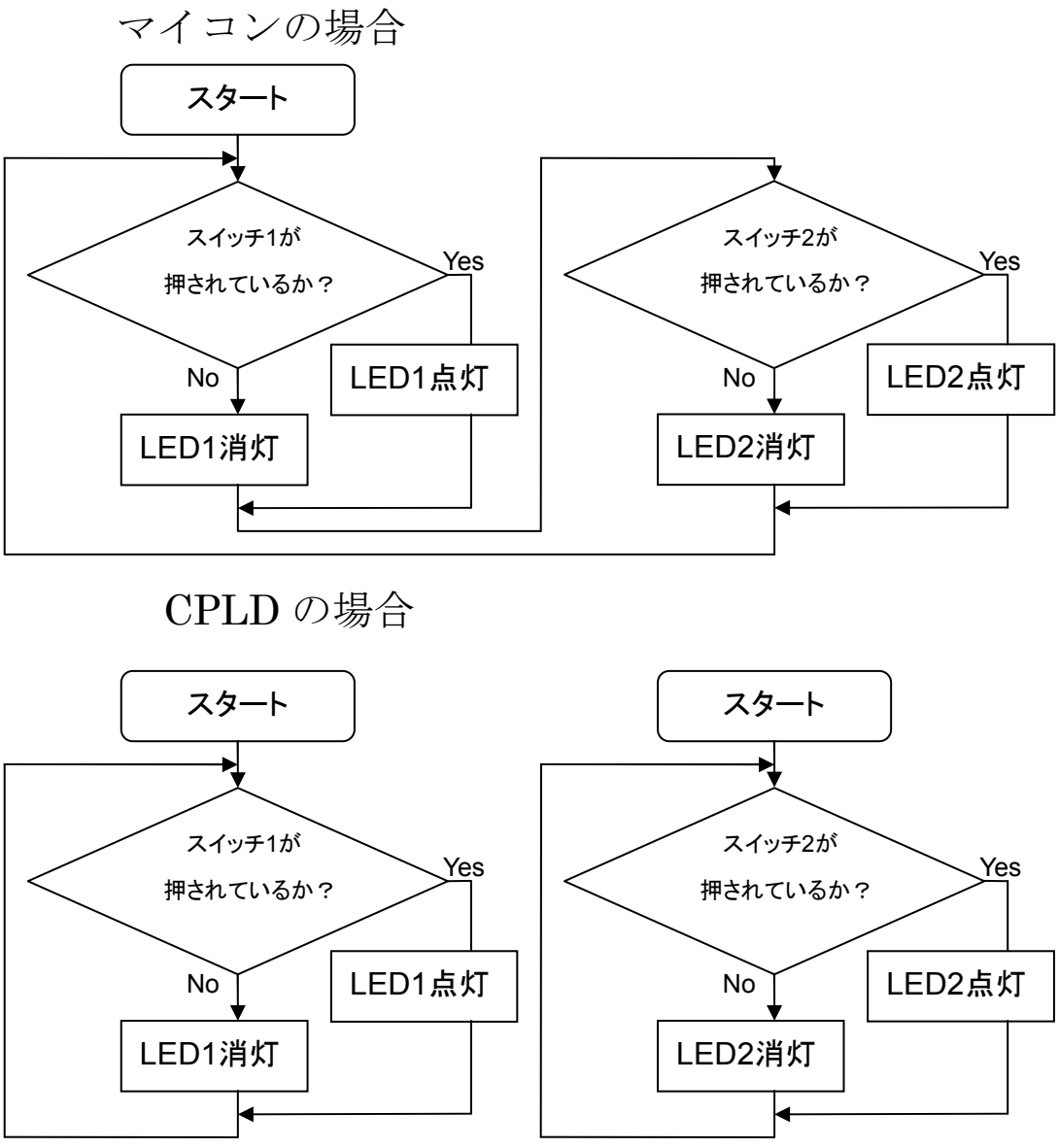



図 2－2　マイコンと CPLD の処理方法の違い

3　CCD(Change Coupled Device)

3.1　使用環境

本実験では CCD は東芝社製 TCD1304DG(素子数 3694)を使用した。

TCD130DG は、感光部に新構造の低暗時出力 pn フォトダイオードを採用した 3698 画素のバーコードリーダ用超高感度 CCD リニアイメージセンサである。

CCD 駆動回路はロジック回路とクロックドライバから構成され 3V 単一電源により動作可能であり、従来必要とした周辺回路を大幅に削減できる。

また、積分クリア機能を付加しているため、センサ内部で発生する光電荷量をコントロールでき、光の強度によるセンサ出力を常に一定に保つことができる。

3.2　特長

・有効画素数　　：3648
・画素サイズ　　：8μm×200μm(8μm ピッチ)
・感光部　　：高感度・低暗時出力 pn フォトダイオード
・駆動方式　　：CCD 駆動回路内臓
・電源電圧　　：3V 単一電源(MIN.)
・付加機能　　：電子シャッタ機能
　　サンプルホールド回路
・パッケージ　　：22Pin DIP
・質量　　：3.5ｇ

表 3－1　最大定格

| 項目 | 記号 | 定格 | 単位 |
|---|---|---|---|
| マスタクロックパルス電圧 | $V_{\phi M}$ | -0.3～7 | V |
| シフトパルス電圧 | $V_{SH}$ | | |
| ICG パルス電圧 | $V_{ICG}$ | | |
| デジタル電源電圧 | $V_{DD}$ | | |
| アナログ電源電圧 | $V_{AD}$ | | |
| 動作温度 | Topr | -25～60 | ℃ |
| 保存温度 | Tstg | -40～100 | ℃ |

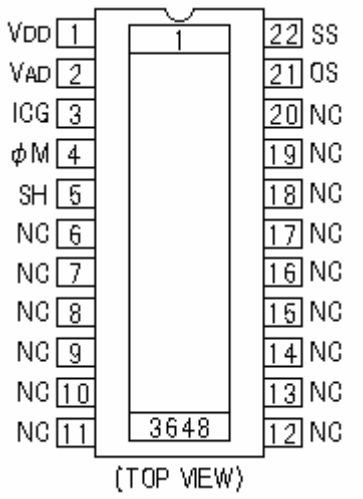
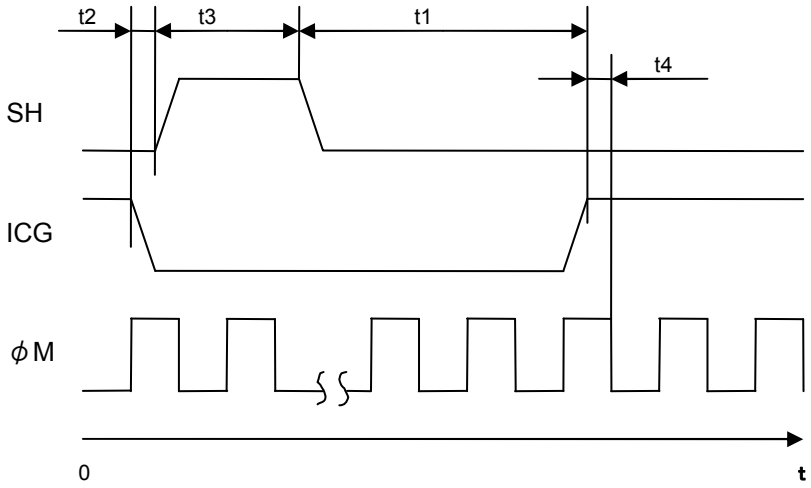



図 3－1　ピン接続図




| 項目 | 記号 | 最小 | 標準 | 最大 | 単位 |
|---|---|---|---|---|---|
| ICG パルス遅延期間 | t1 | 1000 | 5000 | － | ns |
| ICG・SH タイミング | t2 | 100 | 500 | 1000 | ns |
| SH 幅 | t3 | 1000 | － | － | ns |
| ICG・φM タイミング | t4 | 0 | 20 | ※ | ns |

※：φM が HIGH の状態の期間中に動作を行うこと。

図 3－2　パルス波形条件

4　プログラム

4.1　内容

CCD を制御するには ICG、φM、SH の 3 つの波形が必要となる。それぞれの波形は周期、HIGH 状態になる時間(タイミング)、LOW 状態になる時間が異なる。それぞれの波形を出力するために周期、HIGH 状態になる時間(タイミング)、LOW 状態になる時間をそれぞれの値を与えることによって図4－1のようにHIGHの状態になっている時間の長さを変更したり、その時間の長さを変更せずに周期の間であれば自由に遅らせたり進めたりでき、周期も自由に変更できるプログラムを図 4－2 のフローチャートをもとに作成した。この 3 つの値を与えることでそれぞれの波形の周期のずれを一定にすることができ、CCD を制御する波形の規格に合わせることができる。

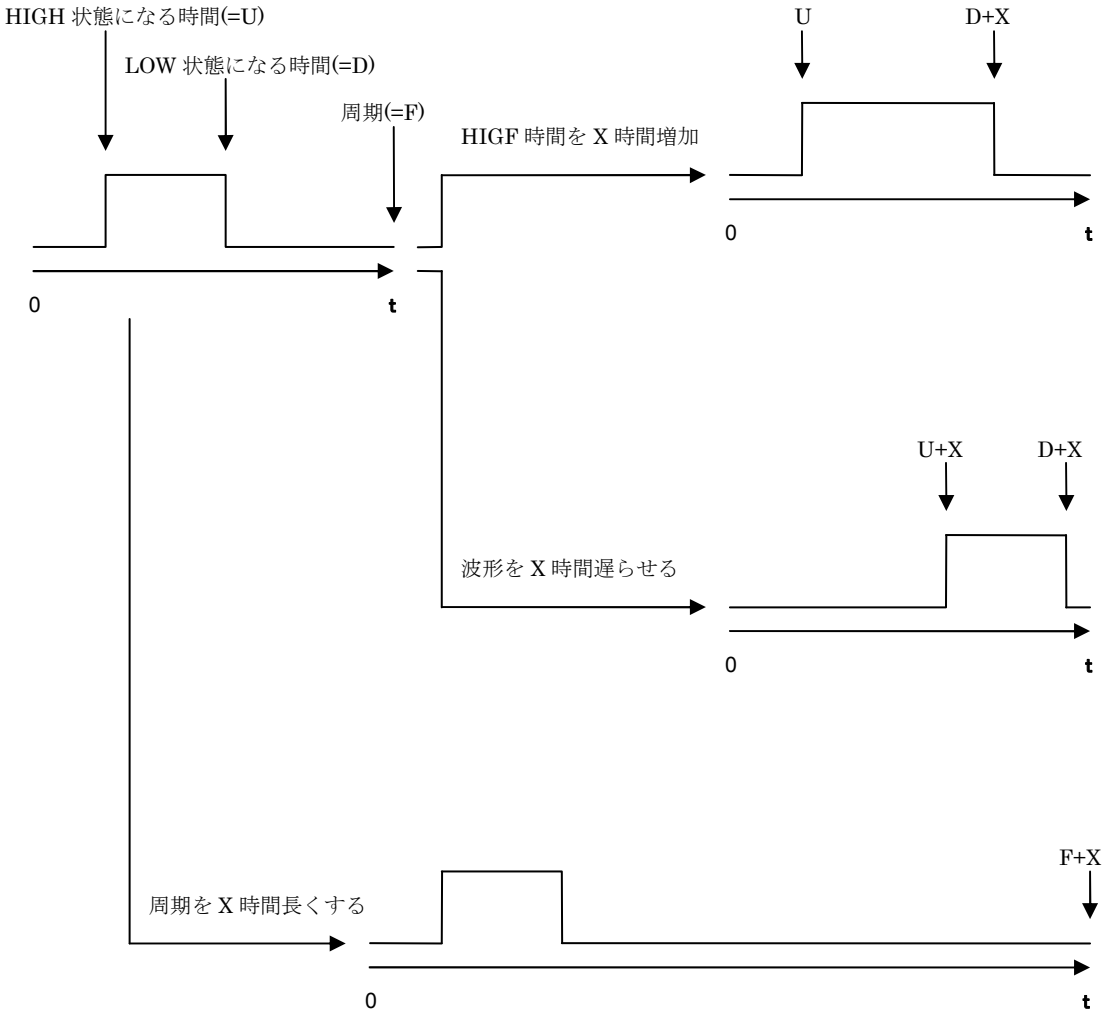



図 4－1

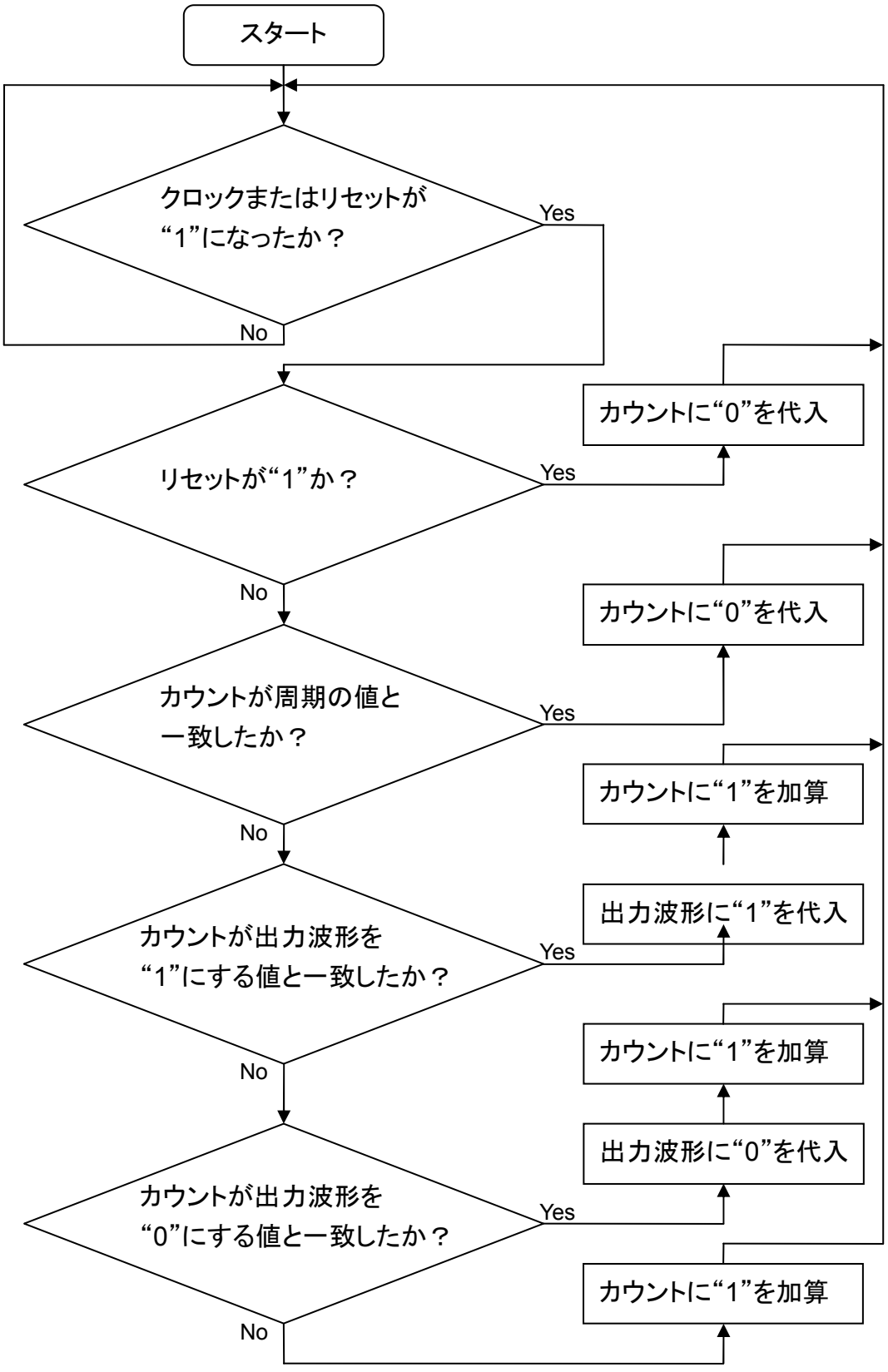

スタート
クロックまたはリセットが
“1”になったか？
Yes
No
リセットが“1”か？
Yes
No
カウントに“0”を代入
カウントが周期の値と
一致したか？
Yes
No
カウントに“0”を代入
カウントに“1”を加算
カウントが出力波形を
“1”にする値と一致したか？
Yes
No
出力波形に“1”を代入
カウントに“1”を加算
カウントが出力波形を
“0”にする値と一致したか？
Yes
No
出力波形に“0”を代入
カウントに“1”を加算

図 4−2　フローチャート

次に、CCD が受ける光の量によって SH の周期を変更しなければならない。ここで 1 から 10 までの値を 4bit スイッチで外部入力にし、1ms～1s の間で 10 段階に周期を変更できるようにした。ここで、図 4－3 のように値を変更した時に SH の周期にずれが発生し、CCD が正常に動作しない。図 4－4 のようにこの問題を改善するために ICG の周期の最後に 1 クロック分だけリセットを ON にする。この ICG から出力したリセットの波形を SH に入力する。これにより、ICG が 1 周期する度に SH が初期化され CCD を正常に動作する波形を出力することが可能となる。

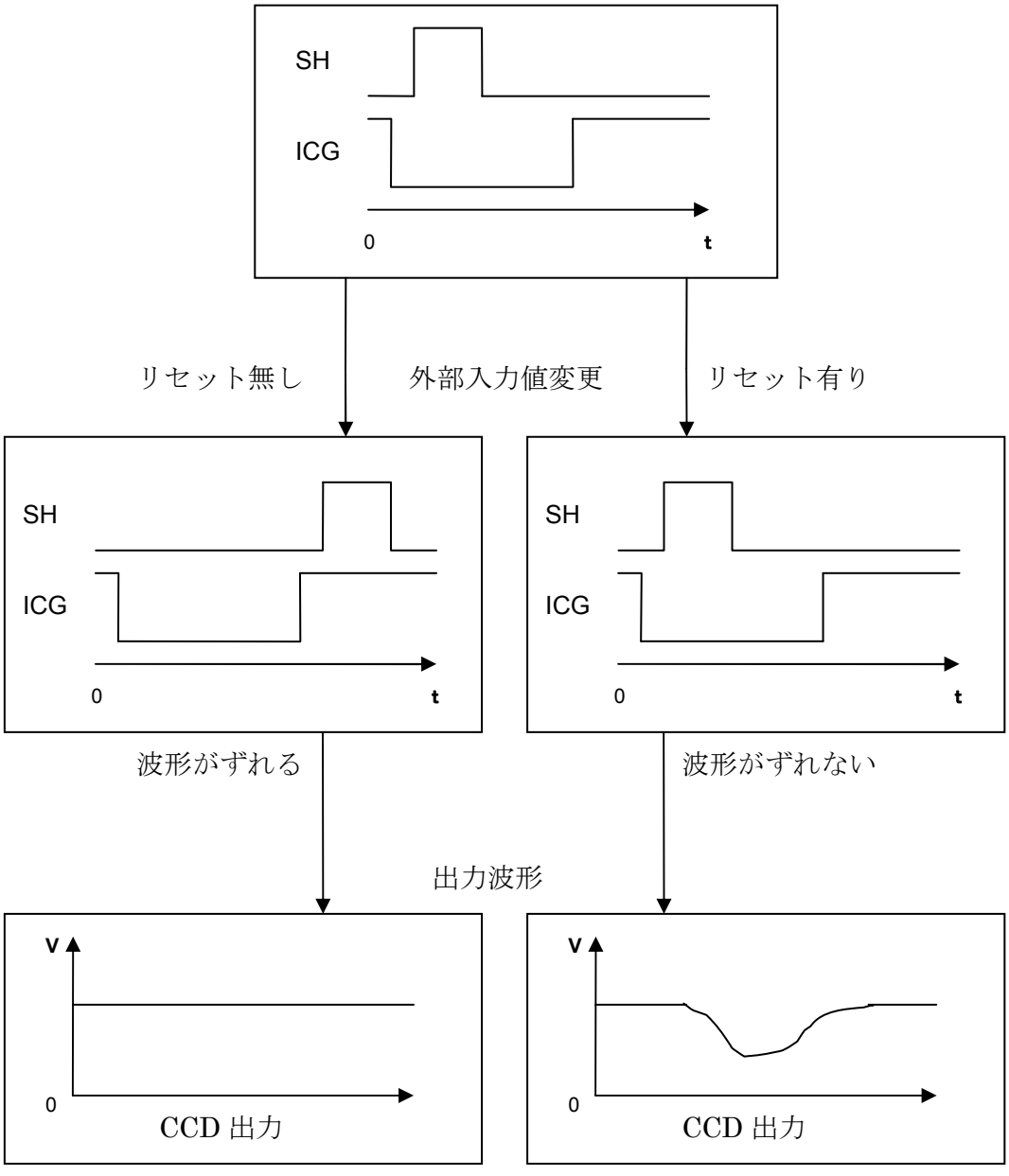



図 4－3　リセットがある場合と無い場合

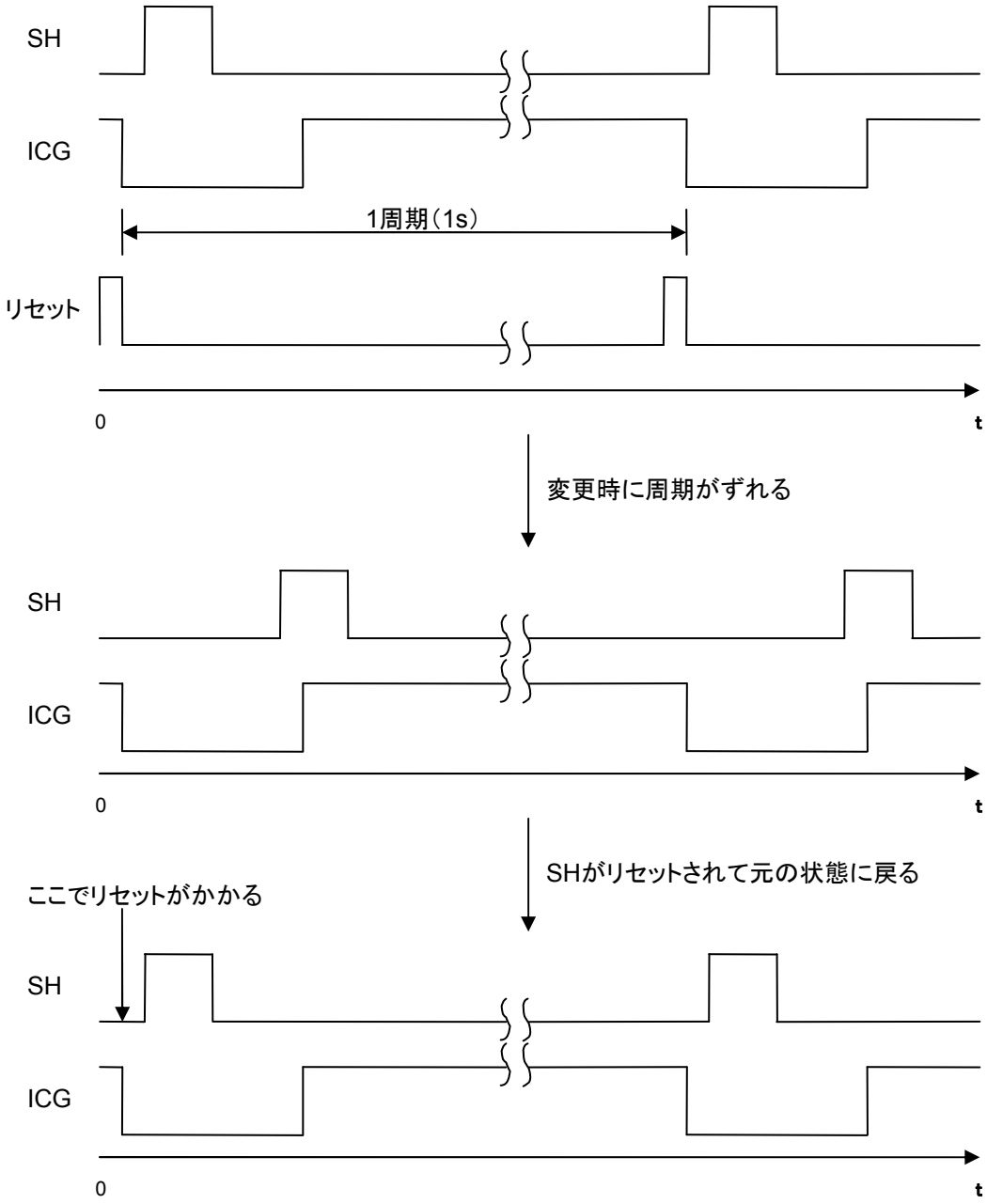

SH
ICG
1周期（1s）
リセット
0
t
変更時に周期がずれる
SH
ICG
0
t
SHがリセットされて元の状態に戻る
ここでリセットがかかる
SH
ICG
0
t

図 4－4　周期のずれ補正

4.2　入力対応

上記で記した外部入力と SH の周期の関係を表したものを表 4－1 に示す。

**ピン入力による出力周期対応表**

| ピン入力値 | 出力周期[ms] |
|---|---|
| 1 | 1000 |
| 2 | 500 |
| 3 | 200 |
| 4 | 100 |
| 5 | 50 |
| 6 | 20 |
| 7 | 10 |
| 8 | 5 |
| 9 | 2 |
| 10 | 1 |

表 4－1　入出力ピン設定

図 4－5 のように※QuartusII により CPLD へ書き込むために波形のプログラムをブロック(部品)としてイメージ化し、結線した。また、結線した結果、表 4－2 に示すように入力として SH の周期を決めるための 4bit のピン、リセット、クロックの 6 つのピン、出力として SH、ICG、φM の 3 つのピン、合計 9 個のピンを設定した。これをコンパイルし CPLD に書き込んだ。

CN2(外側)

| コネクタ端子 | 1 | 3 | 5 | 7 | 9 | 11 | 13 | 15 | 17 | 19 | 21 | 23 | 25 | 27 | 29 | 31 | 33 | 35 | 37 | 39 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| MAXIIピン番号 | | 19 | 21 | 27 | 29 | 33 | 35 | 37 | 39 | 41 | | 43 | 47 | 49 | 51 | 53 | 55 | 57 | 61 | |
| 設定した機能 | | | | | | RE | DI[3] | DI[2] | DI[1] | DI[0] | GND | | | | | | | | | GND |

CN2(内側)

| コネクタ端子 | 2 | 4 | 6 | 8 | 10 | 12 | 14 | 16 | 18 | 20 | 22 | 24 | 26 | 28 | 30 | 32 | 34 | 36 | 38 | 40 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| MAXIIピン番号 | | 20 | 22 | 28 | 30 | 34 | 36 | 38 | 40 | 42 | | 44 | 48 | 50 | 52 | 54 | 56 | 58 | 62 | |
| 設定した機能 | | | | | | SH | ICG | | fM | CLK | GND | | | | | | | | CLK | GND |

入力　出力

表 4－2　ピン入出力設定とその役割

※：QuartusII においては後に付録として掲載する。

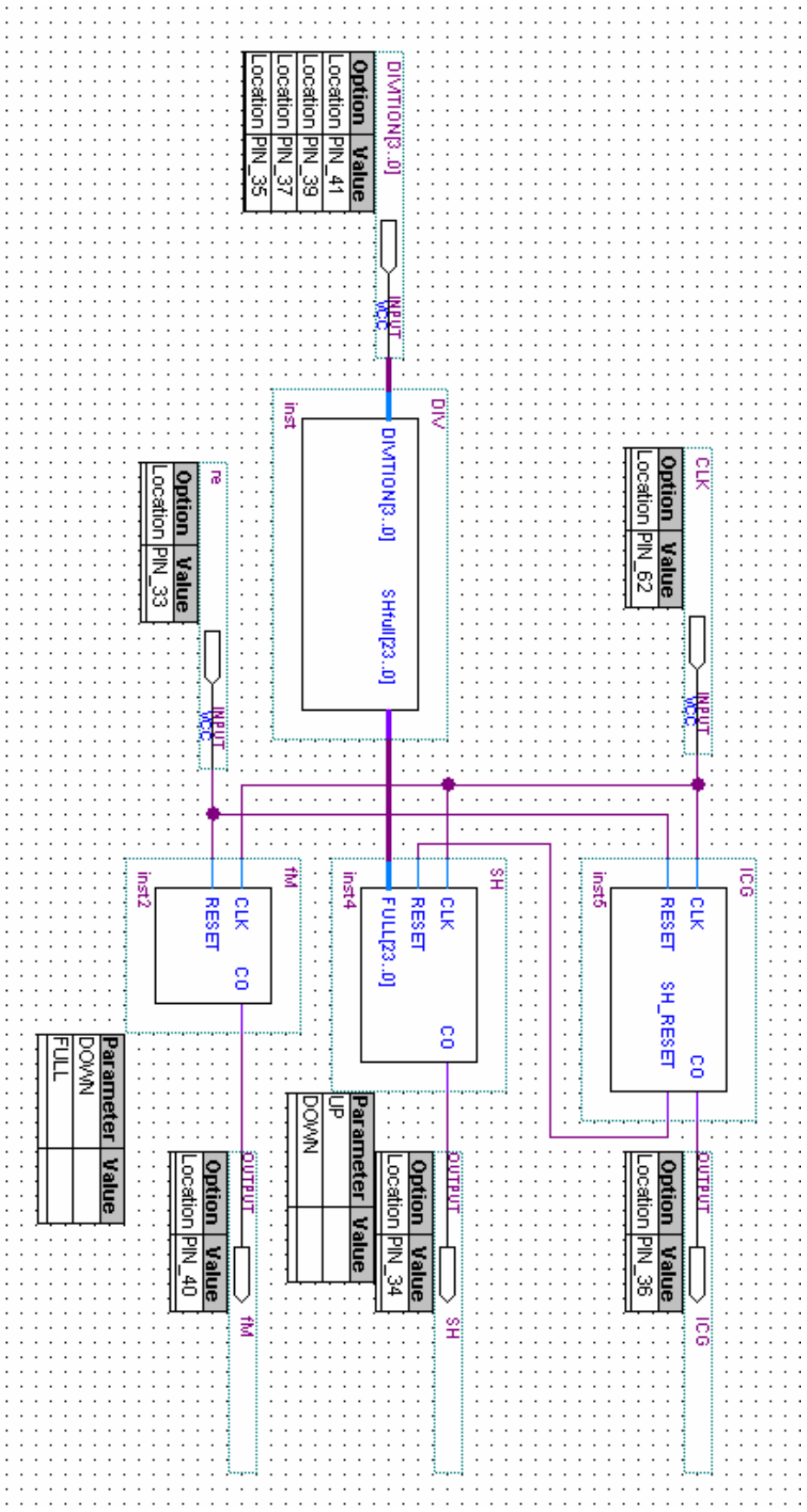
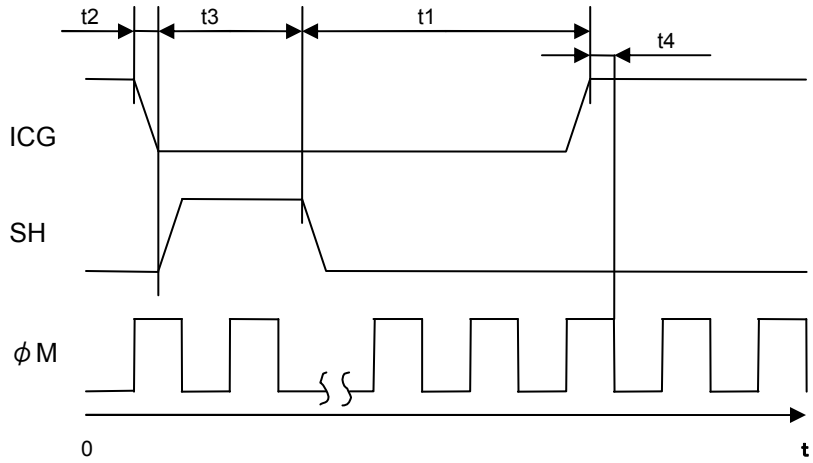

CLK
Option Value
Location PIN_62
INPUT
VCC
ICG
CLK
RESET
SH_RESET
CO
inst5
OUTPUT
Option Value
Location PIN_36
ICG
DIVTION[3..0]
Option Value
Location PIN_41
Location PIN_39
Location PIN_37
Location PIN_35
INPUT
VCC
DIV
inst
DIVTION[3..0]
SHfull[23..0]
SH
CLK
RESET
FULL[23..0]
CO
inst4
OUTPUT
Option Value
Location PIN_34
SH
Parameter Value
UP
DOWN
rst
Option Value
Location PIN_33
INPUT
VCC
fM
inst2
CLK
RESET
CO
OUTPUT
Option Value
Location PIN_40
fM
Parameter Value
DOWN
FULL

図4－5　ブロック図

5　CPLD により出力した CCD 制御波形の規格確認

3 で示したパルス波形条件を図に示す。




| 項目 | 記号 | 最小 | 標準 | 最大 | 単位 |
|---|---|---|---|---|---|
| ICG パルス遅延期間 | t1 | 1000 | 5000 | － | ns |
| ICG・SH タイミング | t2 | 100 | 500 | 1000 | ns |
| SH 幅 | t3 | 1000 | － | － | ns |
| ICG・φM タイミング | t4 | 0 | 20 | ※ | ns |

※：φM が HIGH の状態の期間中に動作を行うこと。

図 3－2　パルス波形条件

この条件をもとに、それぞれの波形の HIGH 状態になるタイミングと LOW 状態になるタイミングと周期を決めた。その値を表 5－1 に示す。

| | HIGH 状態になるタイミング[μs] | LOW 状態なるタイミング[μs] | 周期 |
|---|---|---|---|
| SH | 0.5 | 4.6 | 1ms |
| | | | 2ms |
| | | | 5ms |
| | | | 10ms |
| | | | 20ms |
| | | | 50ms |
| | | | 100ms |
| | | | 200ms |
| | | | 500ms |
| | | | 1s |
| ICG | 10 | 1000000（1s） | 1s |
| φM | 0 | 0.8 | 1.6μs |

表 5－1　パルス波形条件を満たすそれぞれの値

表で示した値をプログラムの中の parameter で定数として代入し、CPLD により波形を出力した。そのときの結果を図 5－1 に示す。

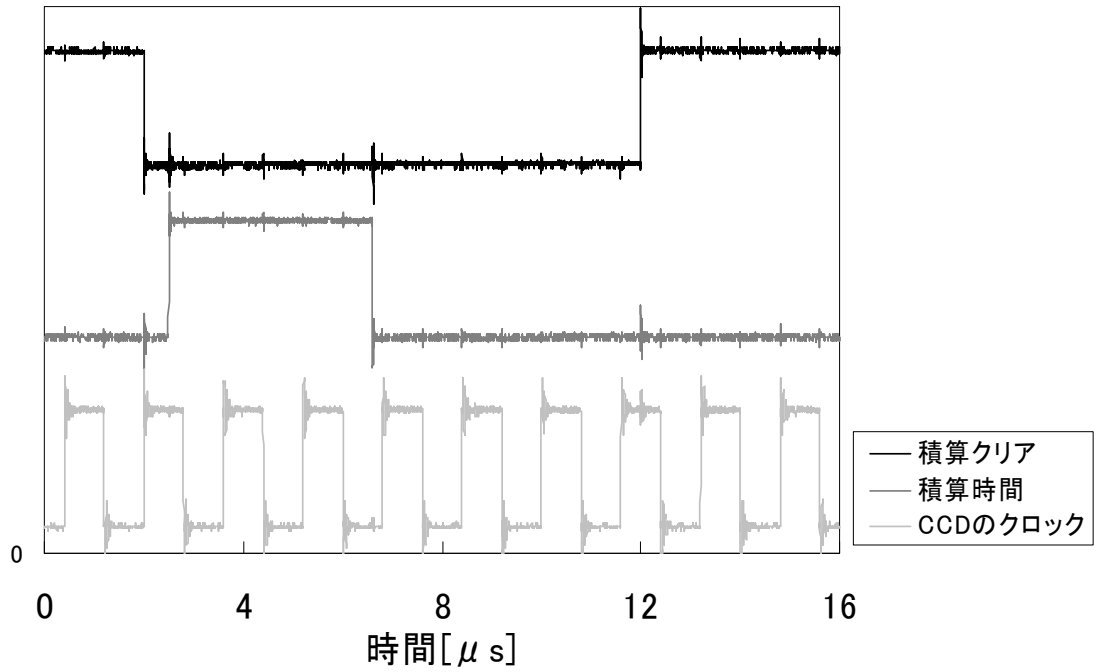



図 5－1　CPLD からの出力

積算時間の波形が立ち下がる瞬間と積算クリアの波形の立ち上がる瞬間までの時間(t1)を計算により表すと 10-4.1-0.5=5.4μs=5400μs になる。図 5－1 より 5μs 以上になっていることが分かり、t1 の条件を満たしている。

積算時間の波形が立ち上がる瞬間と積算クリアの波形の立ち下がる瞬間までの時間(t2)は 0.5μs になる。図 5－1 より CCD のクロックの立ち上がる瞬間と積算クリアの立ち下がる瞬間が同時になっている。また、CCD のクロックの HIGH 状態になっている時間が 0.8μs とした場合、積算時間が CCD のクロックの HIGH 状態の時間のおよそ 60～70%のあたりで立ち上がっている。これより、t2 の条件を満たしている。

積算時間の HIGH 状態の時間を計算により表すと 4.6-0.5=4.1μs となる。図 5－1 よりおよそ 4.0μs だということが分かり、t3 の 1μs 以上という条件を満たしている。

図 5－1 より、積算クリアの立ち上がる瞬間が CCD のクロックの HIGH 状態の期間中に動作していることが分かり、およそ 0.4μs の差があると見られる。これにより、t4 の条件を満たしている。

各々の波形に、ある一定の間隔でノイズが見られる。これは他の波形の立ち上がる瞬間と立ち下がる瞬間にノイズが発生していると考えられる。また、CCD を駆動するのにノイズの除去は必要無いと考えられ、次の手順に進んだ。

6　CCD の動作測定

CPLD により出力した波形により CCD を駆動し出力波形を観察した。また、そのときの実験手順を以下に示す。

1.　CCD の中央部あたりに図 6－1 のような光を遮る障害物を設置する。

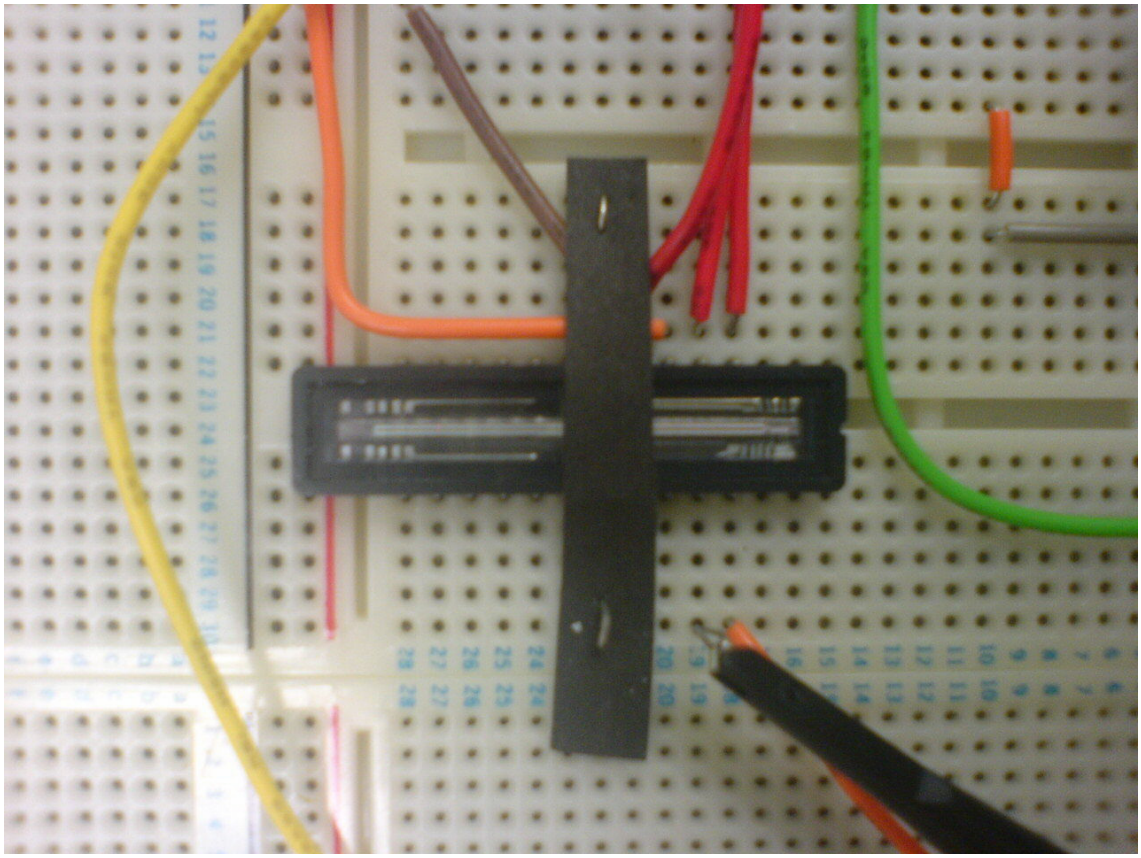


図 6－1　実験に用いた CCD と障害物

2.　積算時間の値を 9 にする。
3.　オシロスコープで CCD の出力が飽和しないよう明るさを調節する。
4.　オシロスコープで出力されている波形のデータを保存する。
5.　積算時間の値を 10 にする。
6.　オシロスコープで出力されている波形のデータを保存する。
7.　2～6 の手順で積算時間の値を 1 ずつ減らしていき、1 と 2 の場合の値のデータを取り終わるまで繰り返す。

この手順で 18 個のデータが取れる。データを取った順で 2 個ずつ比較したものをグタフに示す。これにより、同じ光量の環境で異なる 2 つの積算時間で CCD 出力した波形の比較ができる。

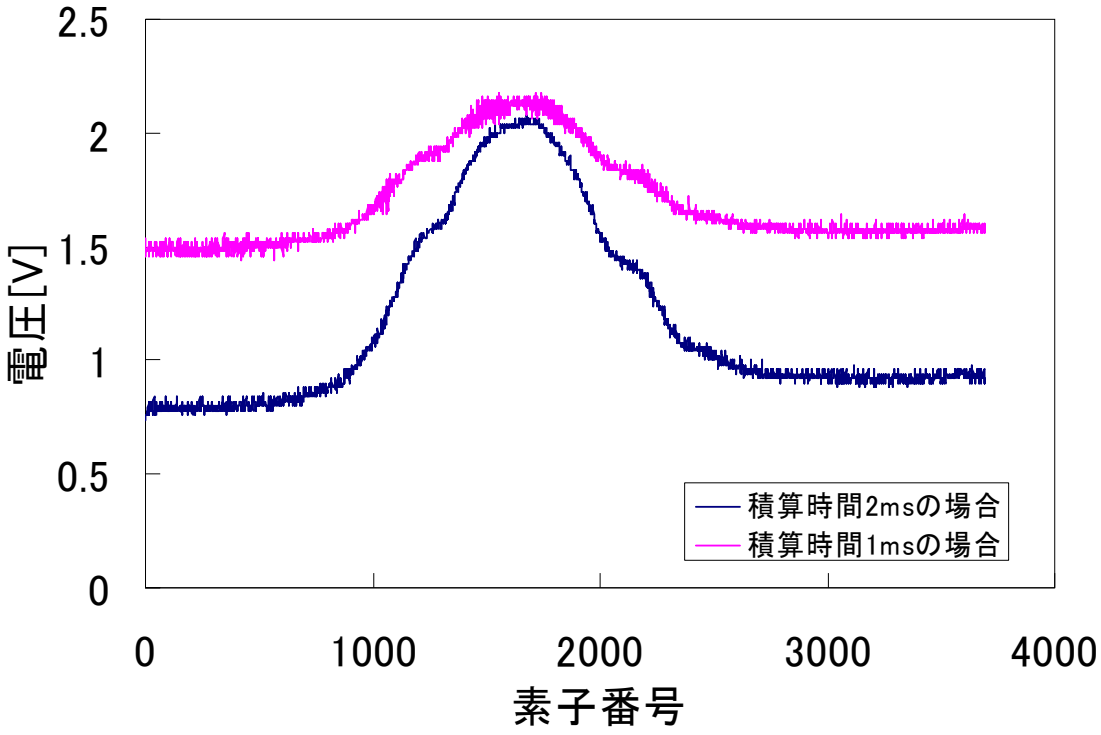



図 6－2　積算時間 1ms と 2ms の場合の CCD 出力比較

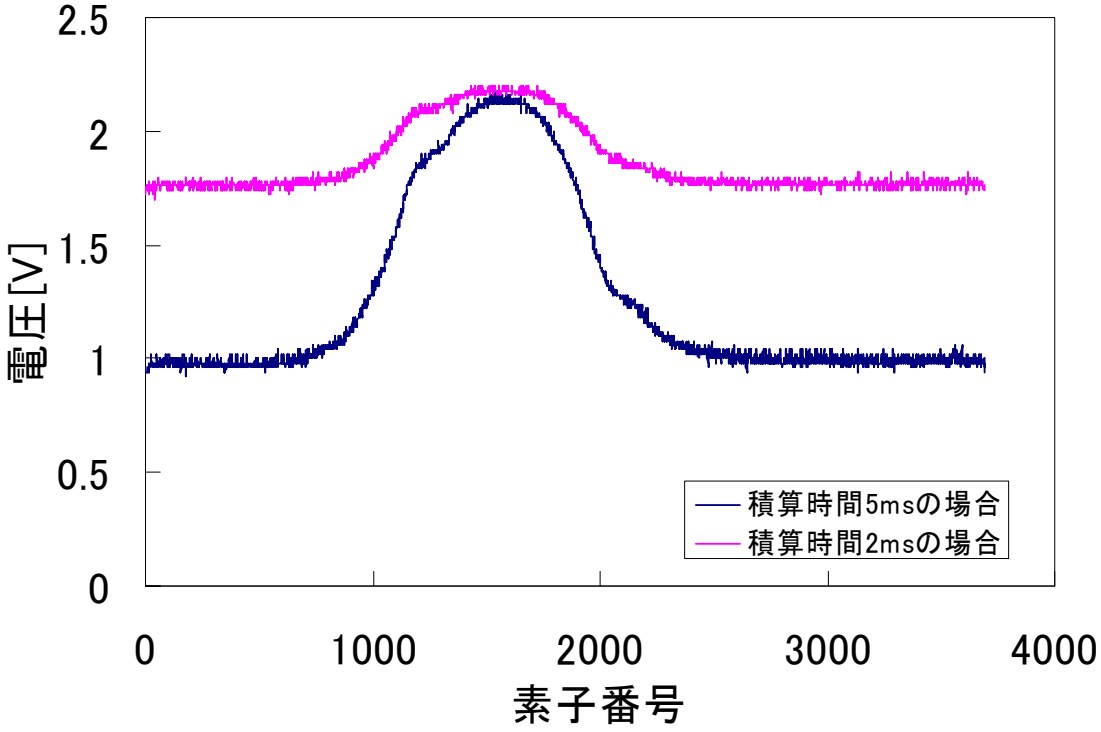



図 6－3　積算時間 2ms と 5ms の場合の CCD 出力比較

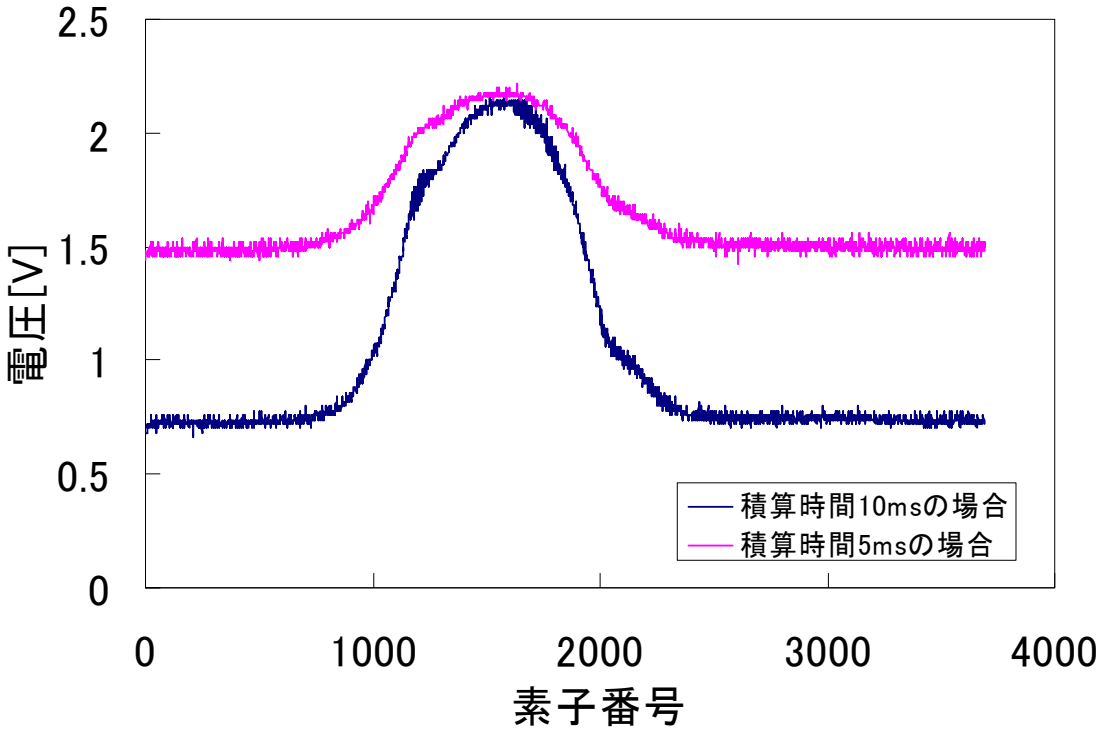



図 6－4　積算時間 5ms と 10ms の場合の CCD 出力比較

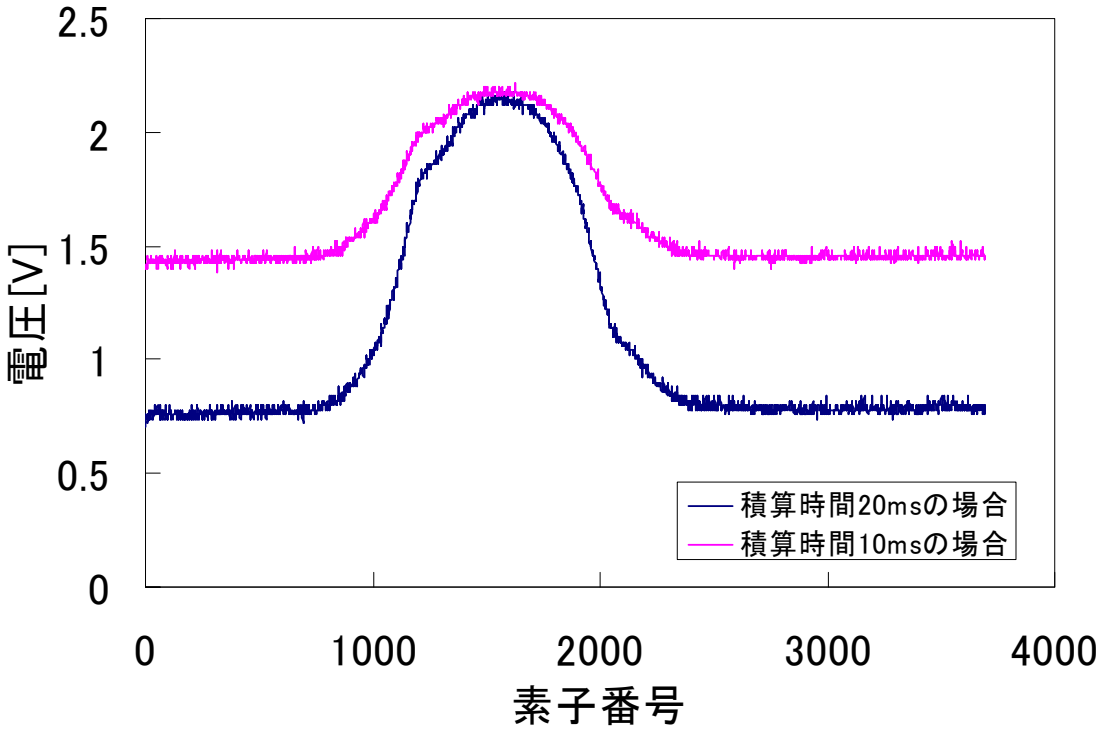



図 6－5　積算時間 10ms と 20ms の場合の CCD 出力比較

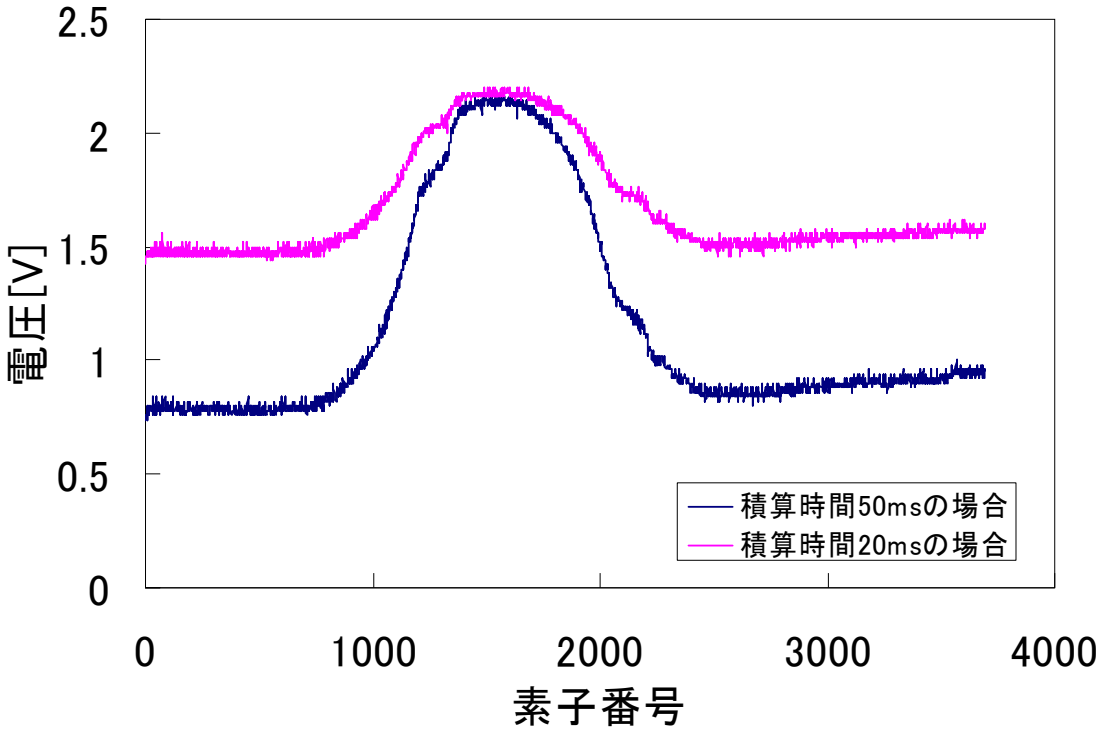



図 6－6　積算時間 20ms と 50ms の場合の CCD 出力比較

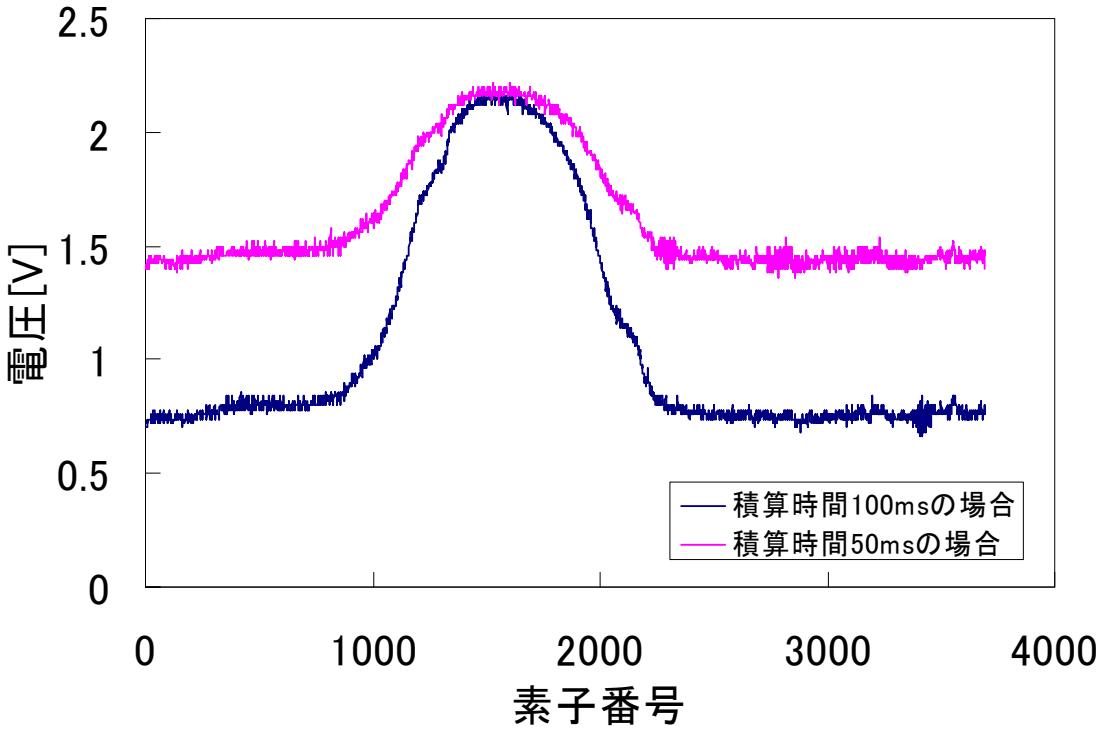



図 6－7　積算時間 50ms と 100ms の場合の CCD 出力比較

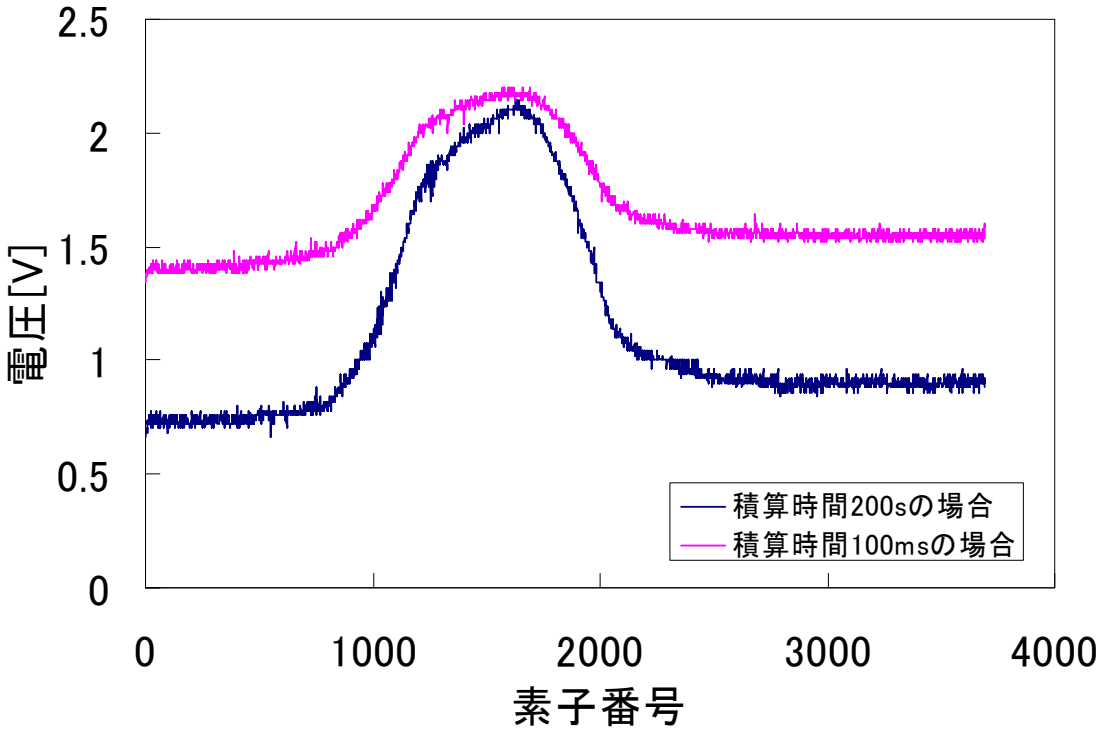



図 6－8　積算時間 100ms と 200ms の場合の CCD 出力比較

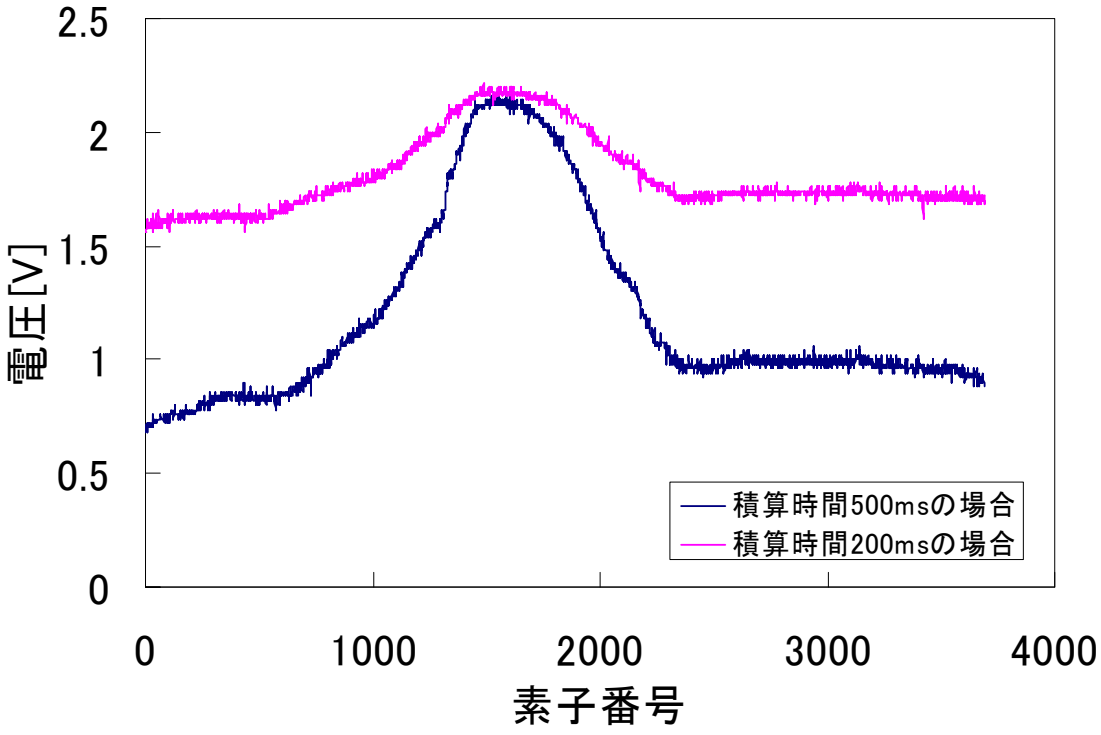



図 6－9　積算時間 200ms と 500ms の場合の CCD 出力比較

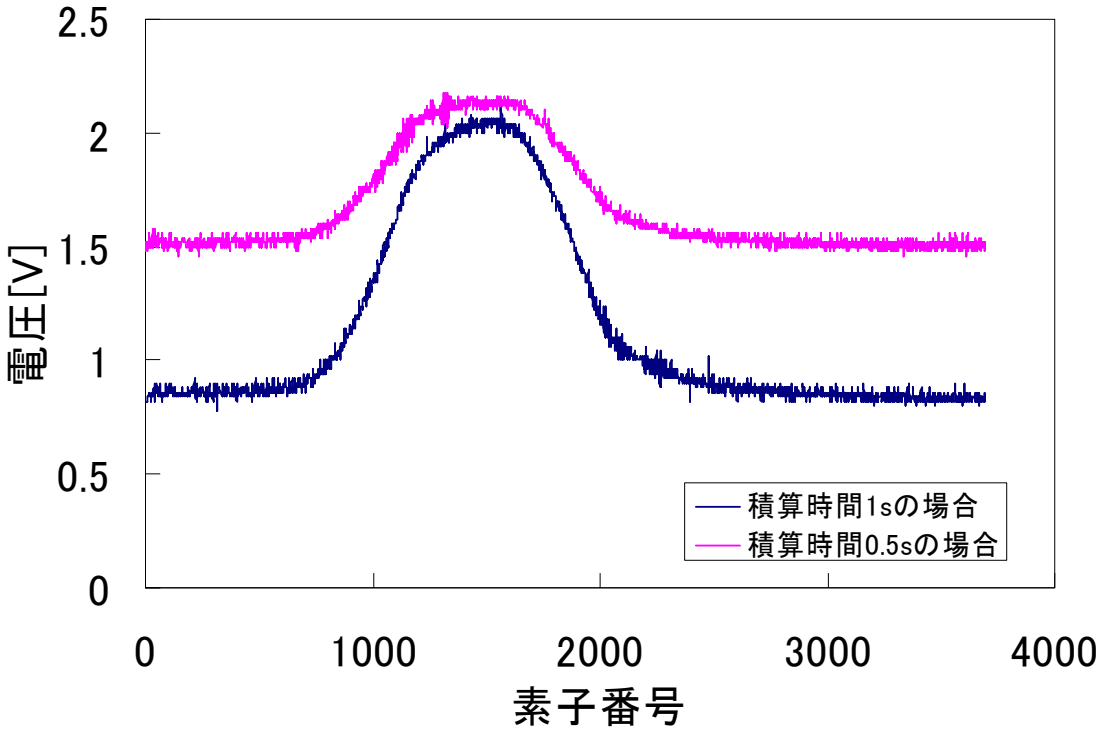



図 6－10　積算時間 0.5s と 1s の場合の CCD 出力比較

それぞれの波形の素子番号 1500 の周辺に光を遮る障害物の営業として電圧が大きくなっているのが分かる。

図 6－2～図 6－10 より、積算時間が小さい方の電圧が大きくなっている。これは CCD のフォトダイオードに逆起電圧がかかっているからである。つまり、受光する量が大きければ多きいほど電圧が低くなると言える。

全く光の無い状態では電圧の大きさが 2.2V となる。この値を基準としそれぞれの出力の比をグラフで見ると積算時間が 1：2 のグラフではおよそ 2 倍になっており、2：5 のグラフの場合でもおよそ 2.5 倍になっていることが分かる。これにより、受光する量が一定で CCD の出力が飽和しなければ積算時間と出力電圧は比例することが考えられた。

## 6　まとめ

CCD を駆動するのに必要な波形を Verilog HDL 言語を用いて CPLD により出力し、受光する量に応じて積算時間を変更できるプログラムを作成した。また、作成した信号により、任意の積算時間で CCD を動作させることができた。

謝辞

本研究を行うにあたり、懇切親身且つ具体的なご指導と御支援さらに本論文作成について多くの貴重な御意見を賜りました本校電子制御工学科由井四海教員に心から感謝いたします。

また研究実験および論文作成に当って貴重な御意見を多数下さいました制御情報システム工学専攻科 2 年中谷健一氏、制御情報システム工学専攻科 1 年鹿熊航太氏にも深く感謝の意を表したいと思います。

ここに心より感謝の意を表明することで、皆様方への謝辞とさせて頂きます。

付録

10.1　参考文献

1) 小林　優 「入門Verilog HDL記述」, CQ出版社

2) 猪飼　國夫、本多　中二 「ディジタル・システムの設計」、CQ出版社

3) 猪飼　國夫 「HDL設計練習帳」、CQ出版社

4) 島海　佳考、田原迫　仁治、横溝　憲治 「HDLサンプル記述集」、CQ出版社

5)「トランジスタ技術　4月号」、CQ出版社

プログラム

実際に作成したプログラムを示す。

```
module DIV (DIVITION, SHfull);

input [3:0]DIVITION;
output [23:0]SHfull;
reg [23:0]SHfull ;

always @(DIVITION)
begin
if (DIVITION == 4'h1)
        begin
    SHfull <= 24'h98967F ;
        end
else if (DIVITION == 4'h2)
        begin
        SHfull <= 23'h4C4B3F;
        end
else if (DIVITION == 4'h3)
        begin
        SHfull <= 21'h1E849F;
        end
else if (DIVITION == 4'h4)
        begin
        SHfull <= 20'hF423F;
        end
else if (DIVITION == 4'h5)
        begin
        SHfull <= 19'h7A11F;
        end
else if (DIVITION == 4'h6)
        begin
        SHfull <= 18'h30D3F;
        end
else if (DIVITION == 4'h7)
        begin
        SHfull <= 17'h1869F;
        end
else if (DIVITION == 4'h8)
        begin
        SHfull <= 16'hC34F;
        end
else if (DIVITION == 4'h9)
        begin
        SHfull <= 15'h4E1F;
        end
else if (DIVITION == 4'hA)
        begin
        SHfull <= 14'h270F;
        end
else
        begin
        SHfull <= 0;
        end
end
endmodule
```

・ $\phi$ M

```
module fM (CLK, RESET, CO);

input CLK,RESET;
output CO;
reg [3:0] COUNT ;
reg CO;

always @(posedge CLK or posedge RESET)

if (RESET == 1'b1)
      COUNT <= 0;
else
    begin
     if (COUNT == 4'hf)
                    begin
          COUNT <= 0;
                    CO <= 1'b1;
                    end
          else if(COUNT == 3'h7)
                    begin
                    CO <= 1'b0;
                    COUNT <= COUNT + 4'h1;
                    end
     else
          COUNT <= COUNT + 4'h1;
    end
endmodule
```

・ICG

```
module ICG (CLK, RESET, CO, SH_RESET);

input CLK, RESET;
output CO, SH_RESET;
reg [23:0] COUNT;
reg CO, SH_RESET;

always @(posedge CLK or posedge RESET)

if (RESET == 1'b1)
      COUNT <= 0;
else
    begin
     if (COUNT == 24'h98967F)
                    begin
          SH_RESET <= 1'b0 ;
                    CO <= 1'b0;
               COUNT <= 24'h0;
                    end
          else if (COUNT == 24'h98967E)
                    begin
          SH_RESET <= 1'b1 ;
                    COUNT <= COUNT + 24'h1;
                    end
          else if (COUNT == 7'h63)
                    begin
                    CO <= 1'b1;
                    COUNT <= COUNT + 24'h1;
                    end
      else
          COUNT <= COUNT + 24'h1;
    end
endmodule
```

・ＳＨ

```
module SH (CLK, RESET, CO, FULL);

parameter UP = 3'h4 ;
parameter DOWN = 6'h2d ;

input CLK,RESET;
input [23:0]FULL;
output CO;
reg [23:0] COUNT ;
reg CO;

always @(posedge CLK or posedge RESET)

if (RESET == 1'b1)
      COUNT <= 0;
else
    begin
          if (COUNT == FULL)
                    begin
                    COUNT <= 24'h0;
                    end
          else if (COUNT == UP)
                    begin
                    CO <= 1'b1;
                    COUNT <= COUNT + 24'h1;
                    end
          else if (COUNT == DOWN)
                    begin
                    CO <= 1'b0;
                    COUNT <= COUNT + 24'h1;
                    end
          else
          COUNT <= COUNT + 24'h1;
    end
endmodule
```

10.2.1　プログラム解説

Verilog HDL 言語の記号の意味、単語の意味を説明する。

**・module A (X, Y, Z);**

モジュールを宣言している。Aにはモジュール名を宣言し、X、Y、Z はそのモジュールで入出力する変数を宣言している。

例　module SH (CLK, RESET, CO, FULL);

モジュール名は SH で、CLK、RESET、CO、FULL の 4 つの変数を入出力として扱う。

**・parameter A = B'hC ;**

定数を宣言している。A には定数名、B にはビット数、C には値が入る。また、h は 16 進数という意味であり、他に b(2 進数)や d(10 進数)が使える。

これで宣言される定数はプログラム内でAと使用することによって宣言した値として使うことができる。

例　parameter GO= 1' b 0 ;

LI <= GO;

LI という変数に 1 ビットの 1(2)が代入される。

**・input　A　;**

入力を宣言している。モジュールを宣言したときの変数を入力としてモジュール内で扱うことができる。

例　module A (X, Y, Z);

input　X　;

モジュールで宣言された X をモジュール内で入力の変数として扱うことができる。

**・output　A　;**

出力を宣言している。モジュールを宣言したときの変数を出力としてモジュール内で扱うことができる。

例　module A (X, Y, Z);

input　X　;

モジュールで宣言された X をモジュール内で出力の変数として扱うことができる。

**・reg　A　；**

レジスタ（変数)。A はレジスタ名を宣言している。宣言したプログラム内で値の入る箱(1bit)として扱うことができる。

例　reg GO；

GO を 1bit のレジスタとして扱うことができる。

**・reg　[X:Y]A　；**

レジスタ。上記のレジスタの bit 数を増やしたもの。X、Y にはそれぞれ値が入り、bit 数は(X－Y+1)で表すことができる。

例　reg　[4:1]K　；

K という名の 4bit のレジスタとして扱うことができる。

**・always @(　A　)**

always 文。A には変数が入る。A を観察し、A に変化があると always 文のプログラムの内容を実行する。また、かっこ内で posedge　A と宣言することで A が立ち上がるとプログラムを実行し､negedge　A と宣言することで A が立ち下がるとプグラムを実行するということもできる。また、posedge　A　or　negedge　B と宣言することによって A が立ち上がるか B が立ち下がるかという条件でプログラムを実行させることができる。

2 つ以上の always 文を使うことのよって always 文が同時に条件を満たしたとき、同時にプログラムが実行され、同時処理を可能とする。しかし、このとき一つのモジュール内で 2 つ以上の always 文を使用する場合には注意が必要である。同じ変数が 2 つ以上の always 文で使用された場合、コンパイル時にエラーが出る。例え理論上ではその一つの変数が使用されている always 文が同時に実行されないとしても可能性としてあると考えられ、always 文が同時に処理された場合、一つの変数に異なった処理が行われることになるからである。

例　always @(posedge A)

　　LI <= GO;

A が立ち上がった瞬間に変数 LI に GO が代入される。

**・if (A = = B)**

if 文。A には変数が入り、B には変数や値が入る。この場合 A の値と B の値が等しい場合、次の行のプログラムが実行される。

**・else if (A = = B)**

else if 文。if 文の後に記述することによって使用できる。if 文と同様にして使用できる。if 文の条件を満たさなかった場合にかっこ内の条件を満たしていれば次の行のプログラムが実行される。

・**else**

else 文。if 文、else if 文の最後に記述することによって使用できる。if 文や else if 文の条件を満たさなかった場合に次の行のプログラムが実行される。

・**endmodule**

モジュールの終わりを宣言する。

・**begin**

**end**

if 文や always 文において二つ以上の処理を行いたい場合に使用する。処理の始めには begin を、処理の終わりには end と記述する必要がある。

QuartusII

1　プロジェクト作成

QuartusII 上では、作る回路が一つのプロジェクトとして扱われる。まずは、新しいプロジェクトを作成する。

1）QuartusII を起動する。図 10－1 のようなウィンドウが開かれる。

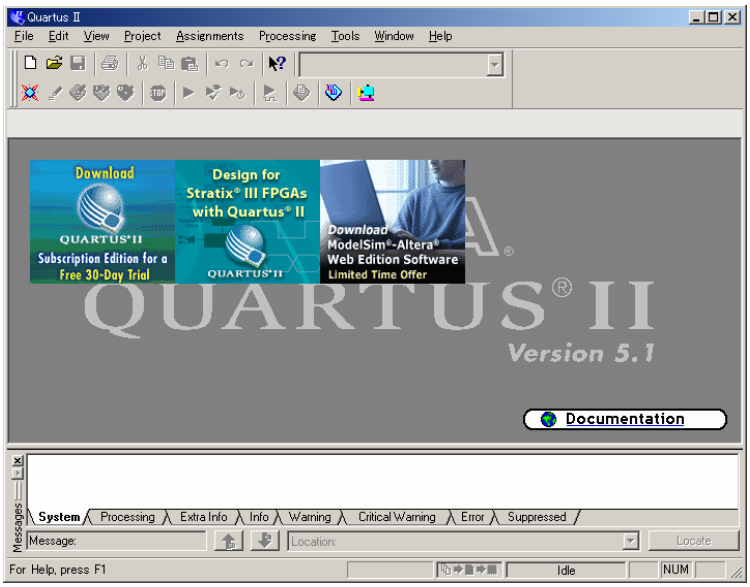



図 10－1

2）新規にプロジェクトを作成するために図 10－2 で示すように[File]メニューからを[New Project Wizard]を選択し、プロジェクト作成ウィザードを起動する。このウィザードを使えば、ダイアログに表示された質問に答えるだけでプロジェクトを作成できる。

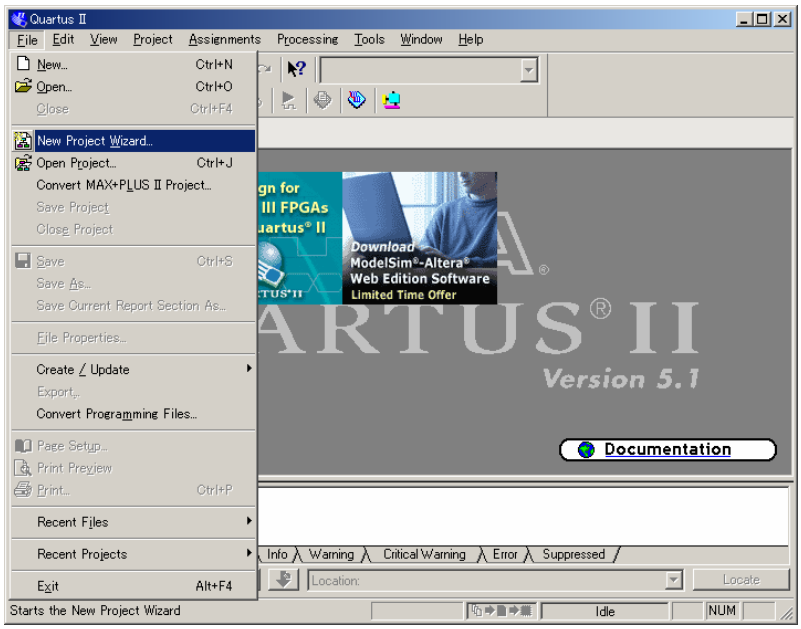



図 10－2

3）プロジェクト名を決める。

最初に、図 10－3 のような New Project Wizard に関する説明が表示される。そのまま[Next]を押す。すると図 10－4 にようなプロジェクトの名前や保存場所を入力するダイアログが開かれる。ここでは main と入力しておく。このとき半角のアルファベットや数字などを組み合わせただけの単純な名前にすることに注意する。

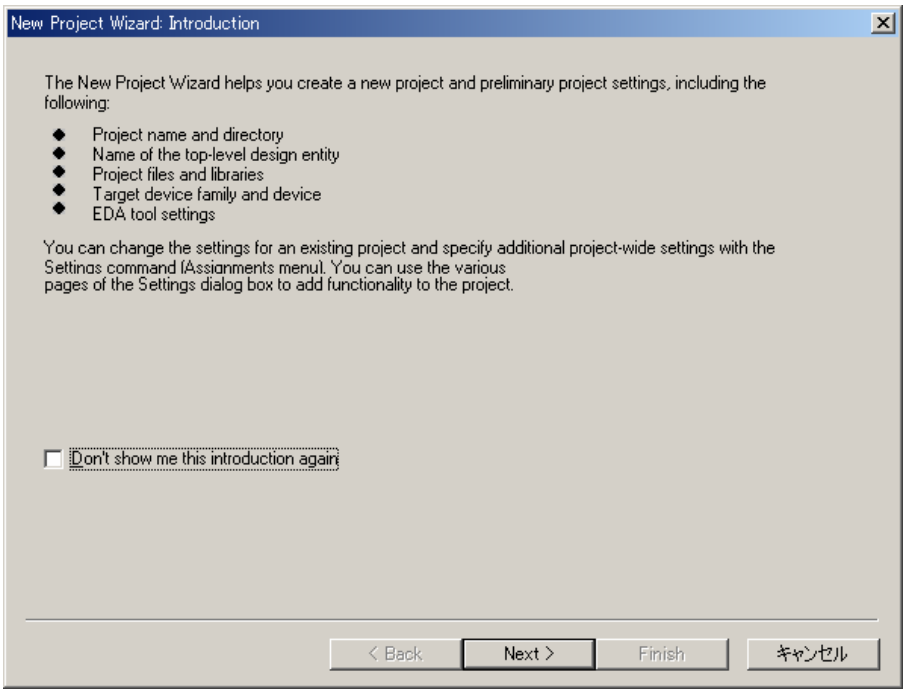



図 10－3

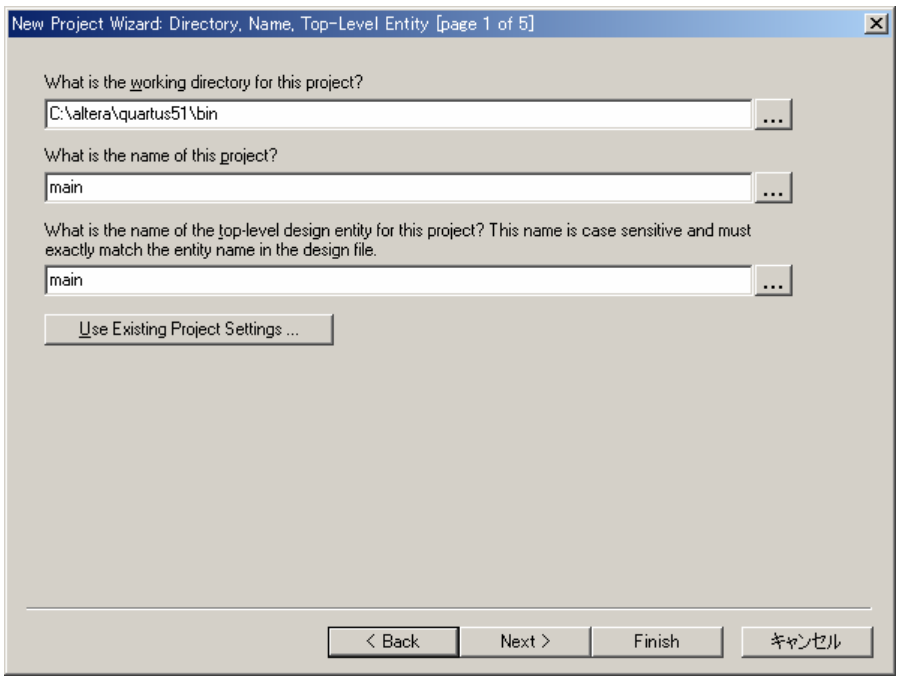



図 10－4

4）使用する CPLD の種類を選択する。

[Next]を押すとプロジェクトに追加したいファイルがあるときに利用するダイアログが表示される。この場合、追加したいファイルが無いので[Next]を押す。すると図 10-3 のような使う CPLD デバイスの選択ダイアログが開く。本実験で使用した CPLD は、MAXII ファミリの EPM240T100C5 を使用したので、[Family]というプルダウン・メニューで図 10－5 の丸部分をクリックし、[MAXII]を選択し、Available devices の中から[EPM240T100C5]を選び、[Next]を選択する。

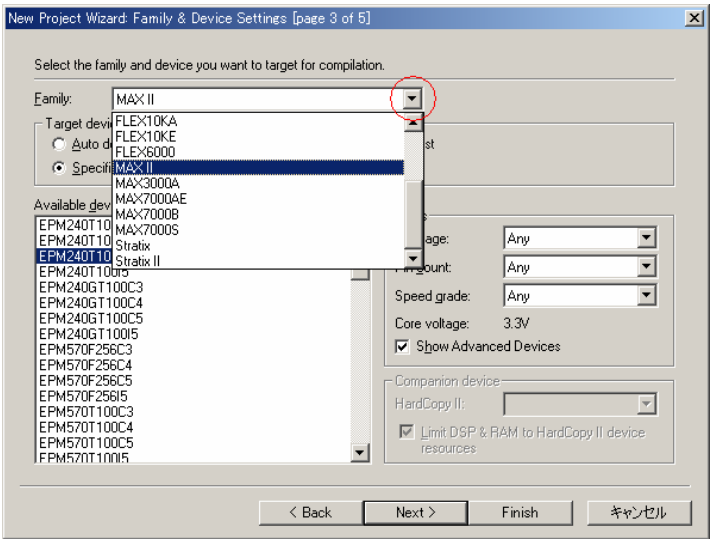



図 10－5

5）プロジェクト作成を完了する。

次のダイアログでは、このプロジェクトで使用したい QuartusII 以外のツールを選択できる。本実験では QuartusII だけで設計を行うので、何も選択せずに[Next]を押す。次に図 10－6 のようなダイアログが開くので内容を確認して[Finish]を押す。

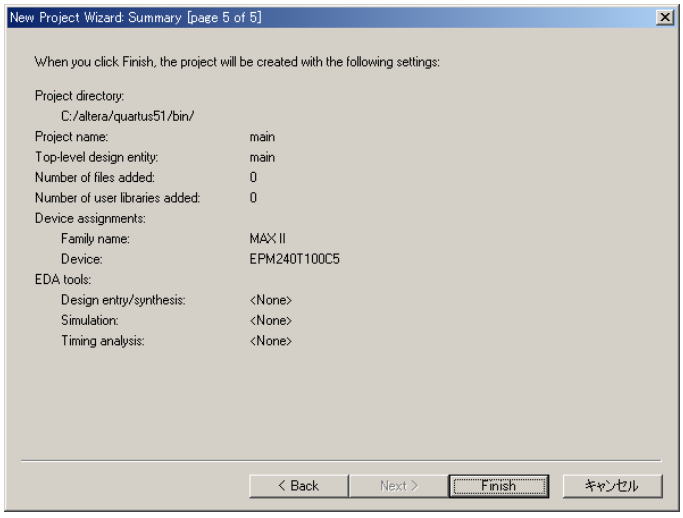



図 10－6

2　プログラム作成

1）ファイル作成

図 10－7 のように[File]メニューから[New…]を選択する。すると図 10－8 のようなダイアログが開かれる。Design File の欄で[Verilog HDL File]を選択し[OK]を押す。すると図 10－9 のように新しい Verilog HDL ファイルを作成される。

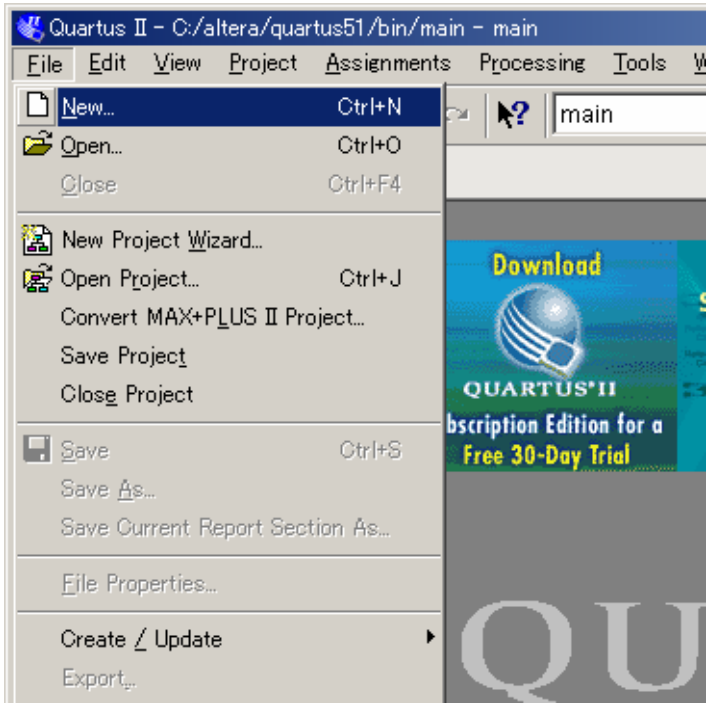



図 10－7

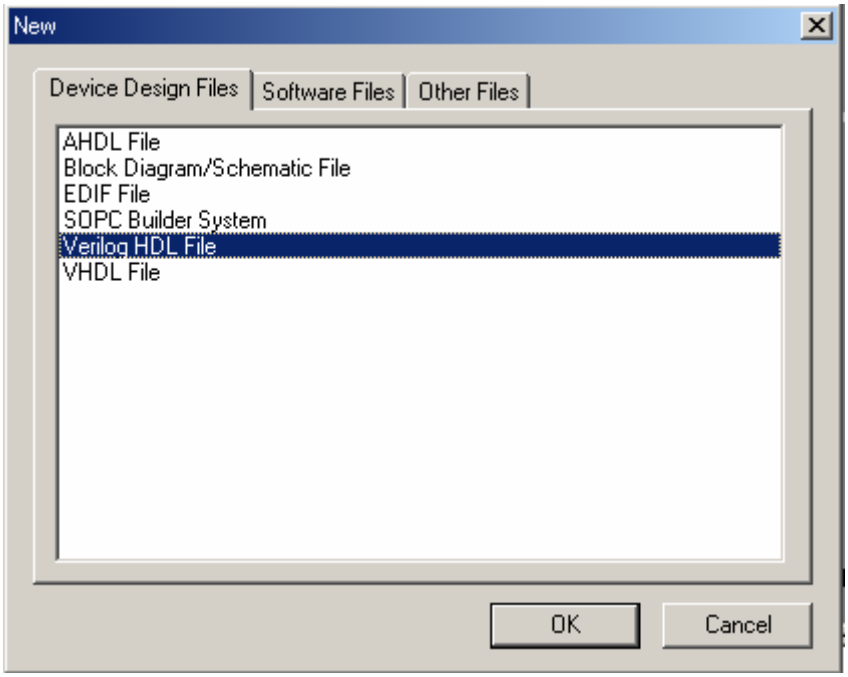



図 10－8

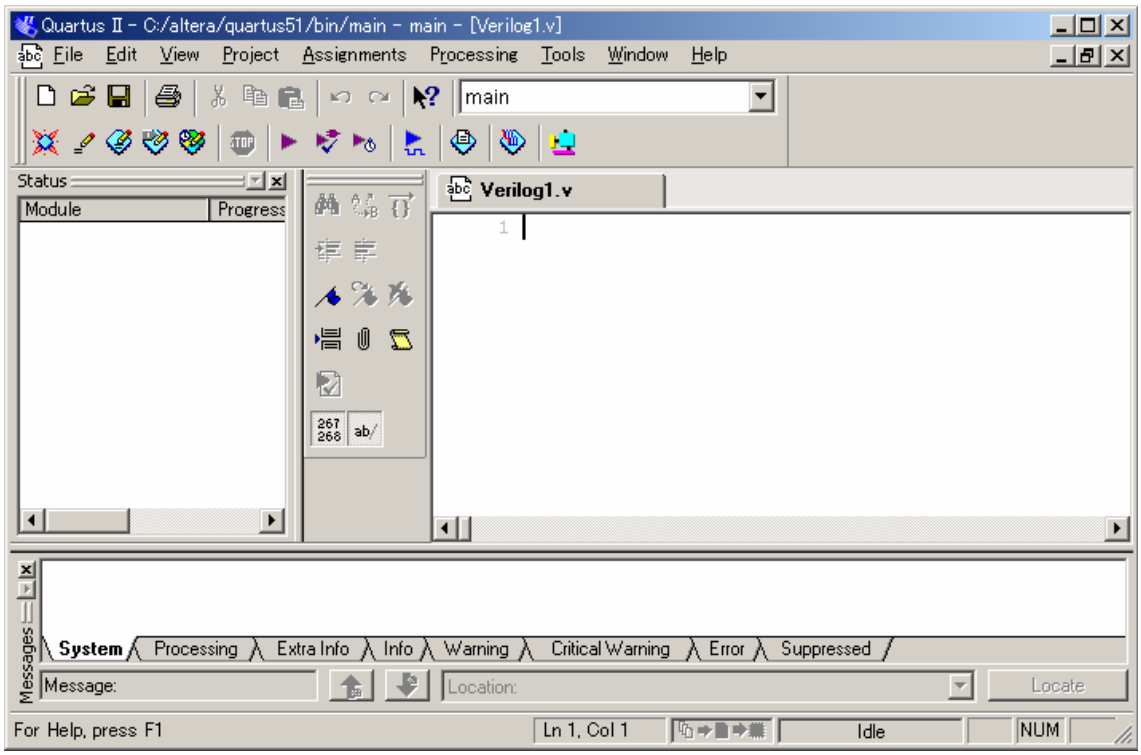



図 10－9

2）プログラムを保存する

処理したい内容を作成する。[File]メニューから[Save as...]を選択し、図 10－10 のようなファイル名と保存先を決めるダイアログが開かれるので、ファイル名と保存先を設定し保存する。

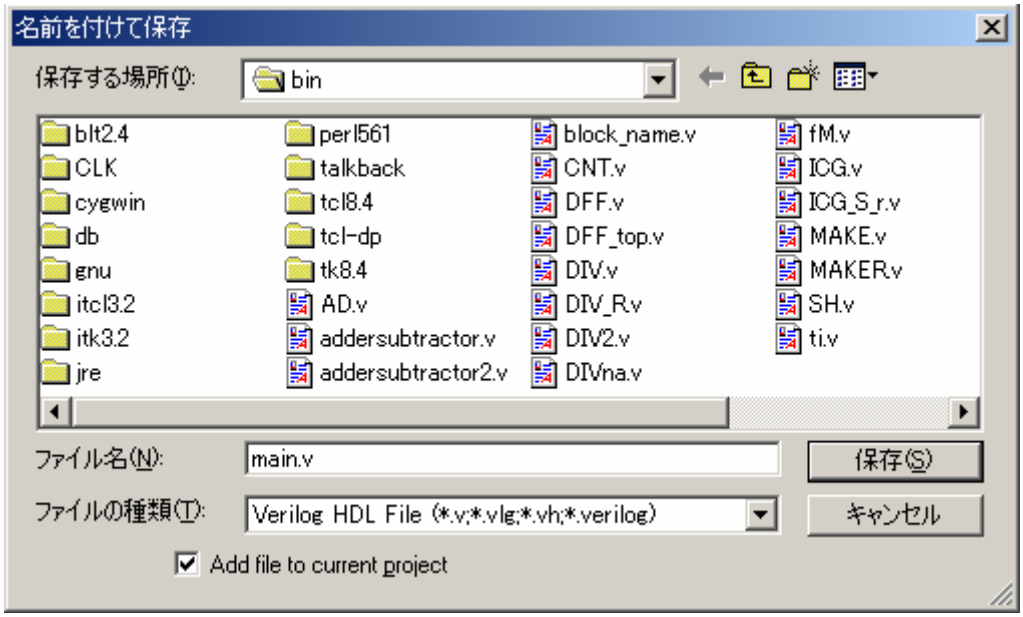



図 10－10

2　　プログラムをコンパイルする

[Processing]メニューから[Start Compilation]を選択するか、図 10－11 において丸で囲んである  のボタンを押すとコンパイルが始まる。このとき、作成したプログラムに間違いが出るとエラーとして表示される。このとき、プロジェクト名とモジュール名が同一でなければコンパイル時にエラーが出る。

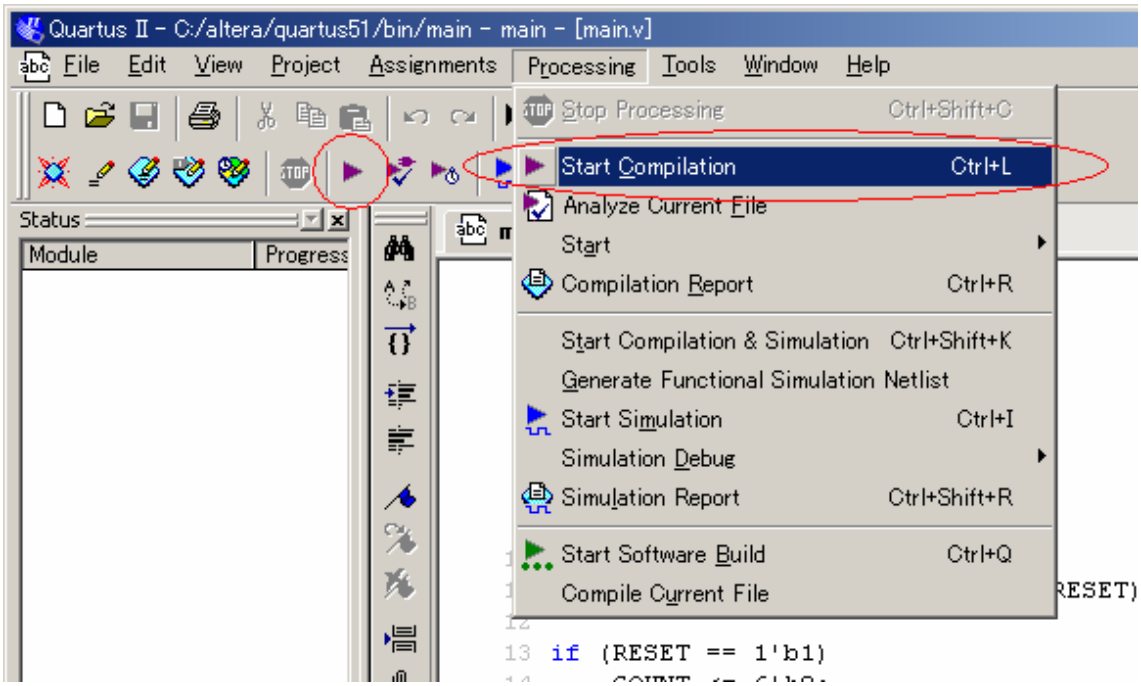



図 10-11

3　プログラムをシンボル化（パーツ）する

図 10－12 のように[File]メニューから[Create/Update]を選択し、[Create Symbol Files for Current File]選択する。すると、処理が行われシンボル化する。

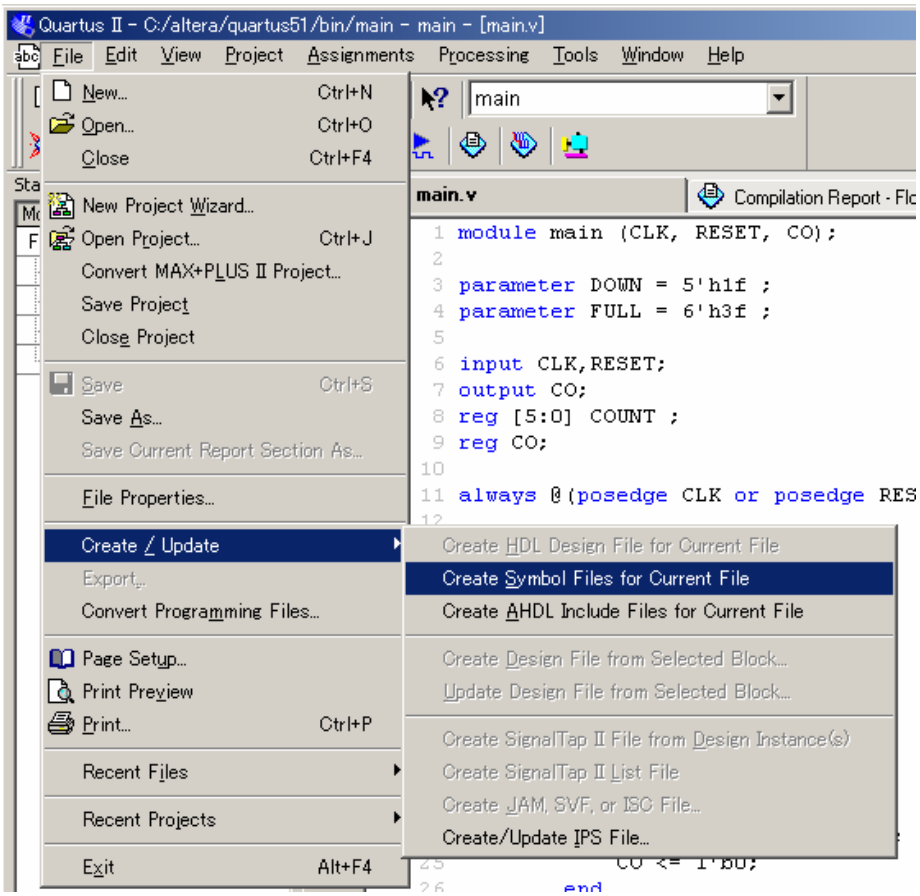



図 10-12

4　作成したシンボルを結線する。

1）新しいプロジェクトを作成する。

2）ファイル作成

図 10－7 のように[File]メニューから[New…]を選択する。すると図 10－13 のようなダイアログが開かれる。Design File の欄で[Block Diagram/Schematic File]を選択し[OK]を押す。すると図 10－14 のように新しいファイルを作成される。

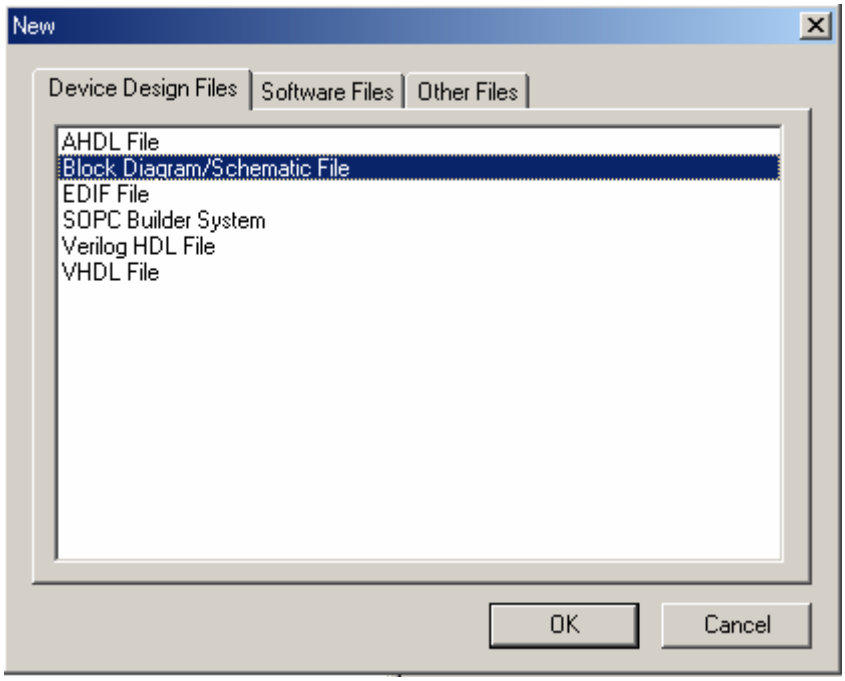

New
Device Design Files
Software Files
Other Files
AHDL File
Block Diagram/Schematic File
EDIF File
SOPC Builder System
Verilog HDL File
VHDL File
OK
Cancel

図 10－13

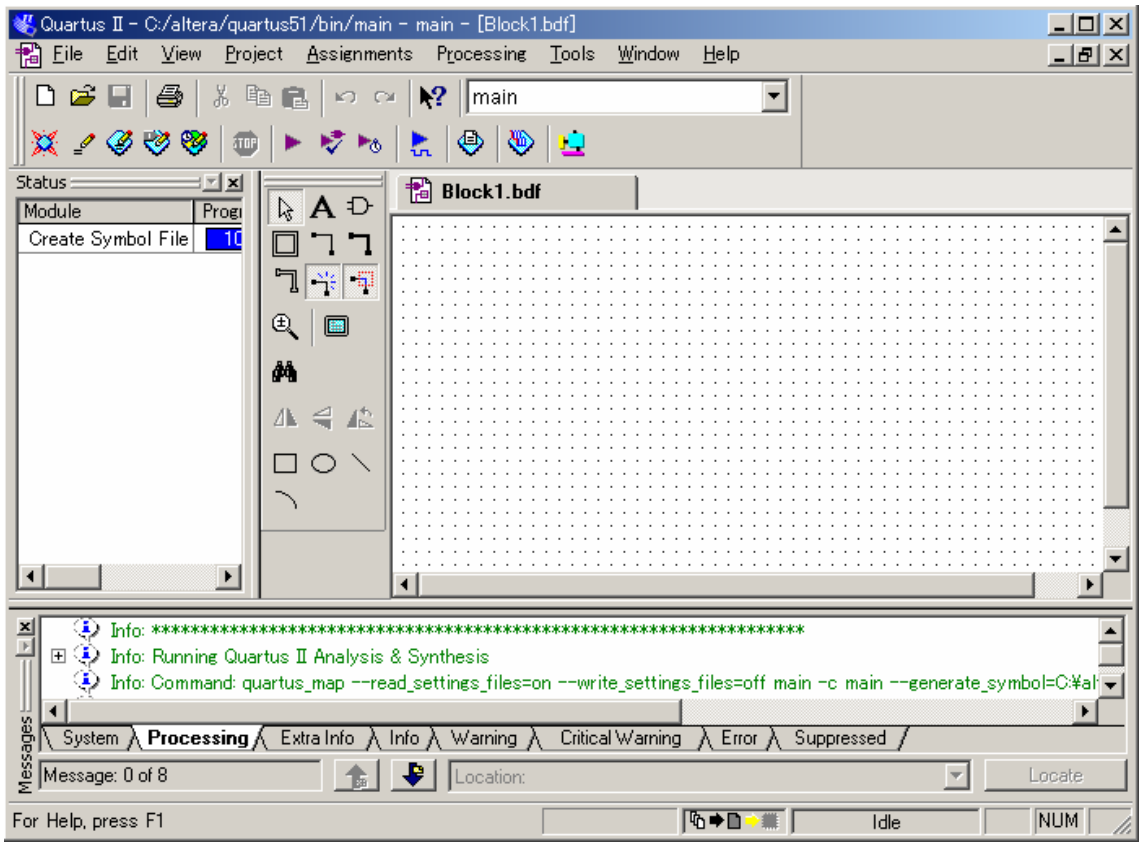

Quartus II - C:/altera/quartus51/bin/main - main - [Block1.bdf]
File Edit View Project Assignments Processing Tools Window Help
main
Status
Module
Create Symbol File
Block1.bdf
Info: Running Quartus II Analysis & Synthesis
Info: Command: quartus_map --read_settings_files=on --write_settings_files=off main -c main --generate_symbol=C:¥al
System Processing Extra Info Info Warning Critical Warning Error Suppressed
Message: 0 of 8
Location:
Locate
Messages
For Help, press F1
Idle
NUM

図 10－14

3)　作成したシンボルを設置する。

図 10－15 に記した丸で囲んである[Symbol Tool]のボタンを押すと図 10－16 のようなダイアログが開かれる。Libraries 欄に、project、C:/altera/Quartus51/libraries/という項目がある。Project の左にある[+]記号をクリックすると、中身が表示される。その中に作成したファイルと同じモジュール名がある。これを選択し[OK]を押す。次に、設置したい場所にマウスのカーソルを移動し、クリックすることによってシンボルが設置させる。

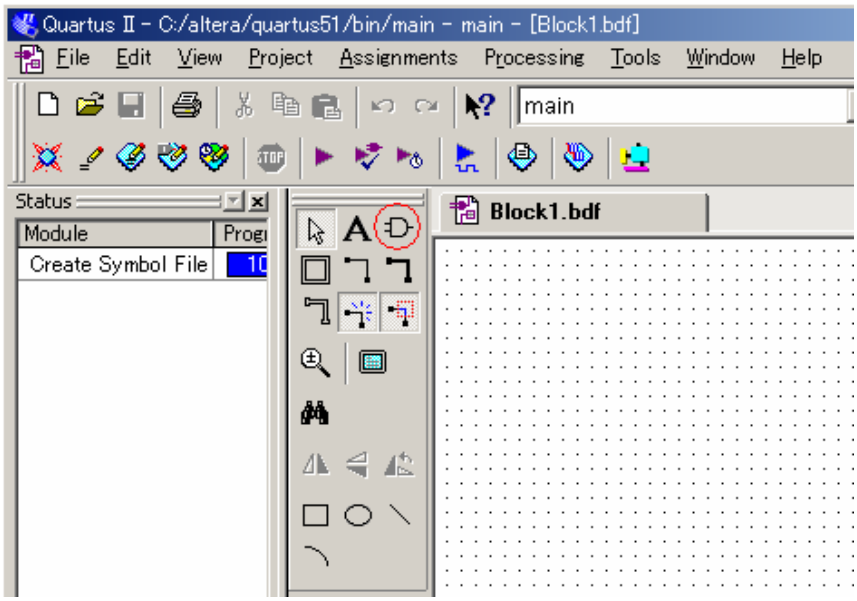



図 10－15

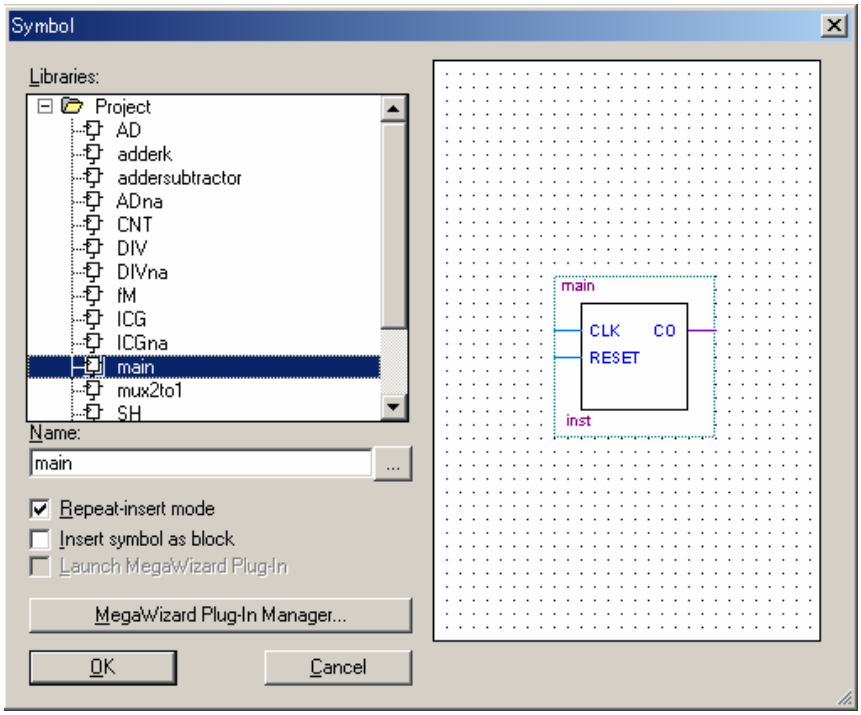



図 10－16

4)　input（入力）シンボル、output（出力）シンボルを設置する。

3)と同様に C:/altera/Quartus51/libraries/の左にある[+]記号をクリックし、primitives、pin の順で[+]記号をクリックすると output、input のシンボルがある。これを設置する。図では入力が二つになっているので input シンボルを二つ設置する。

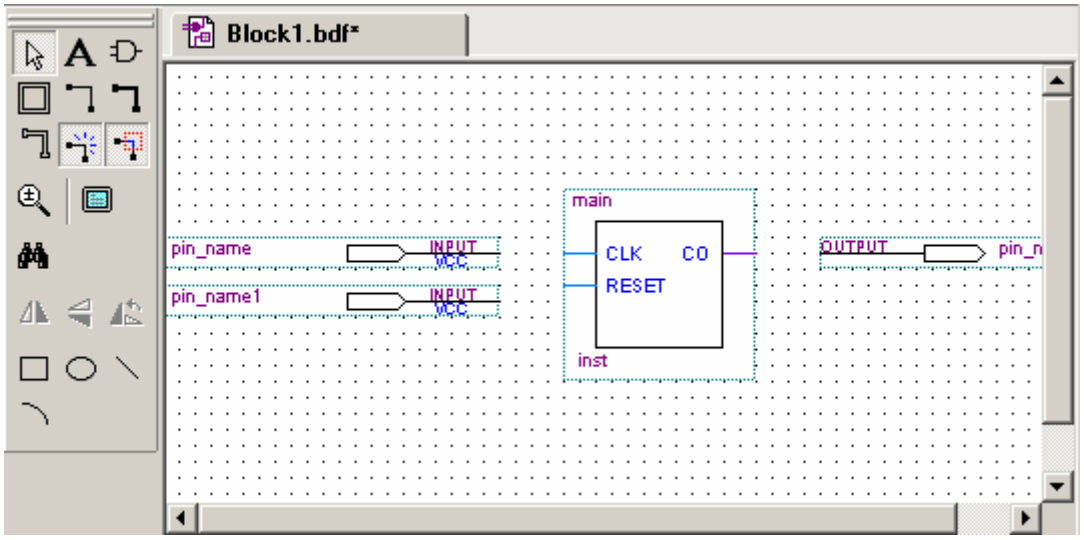



図 10－17

5)　結線する。

丸部分で囲った[Orthogonal Node Tool]のボタンを押す。次にシンボルのピンからつなげたいピンまでをドラッグ＆ドロップするようにマウスのボタンを押しっぱなしにして結線する。右の[Orthogonal Bus Tool]のボタンは一つの線で二つ以上のデータを扱い場合に用いる。

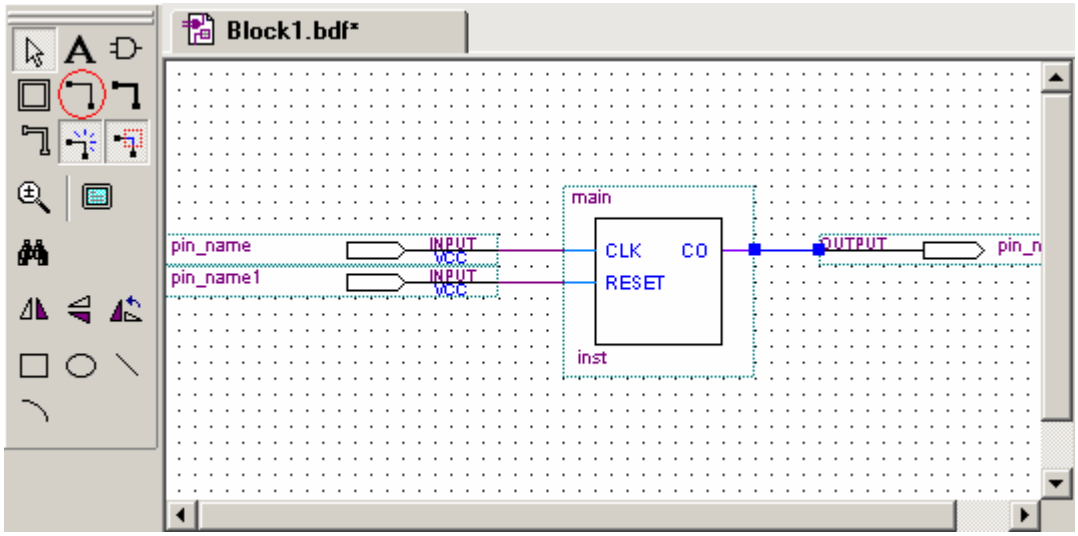



図 10－18

6)　ピンの名前を決める。

図 10－18 の pin_name の部分をダブルクリックし、変更できる。または、シンボルをダブルクリックするか、シンボルを左クリックして表示される Properties を選択すると図 10－19 のようなダイアログが開かれ、Pin name(s)欄でピンの名前を決め[OK]を押す。

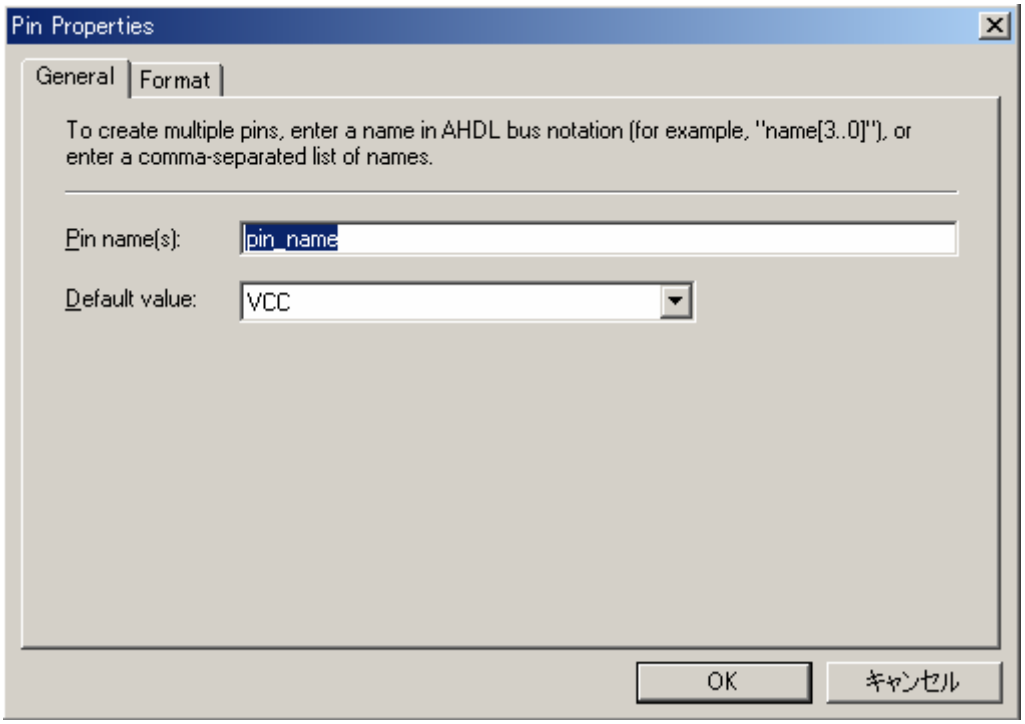



図 10－19

5　回路上の部品の入出力と MAXII の端子を関連づける。

1)　回路をコンパイルする。

プログラムをコンパイルしたときと同じ手順で[Processing]メニューから[Start Compilation]を選択するか、図 10－11 において丸で囲んである  のボタンを押すとコンパイルが始まる。このとき、作成したプログラムに間違いが出るとエラーとして表示される。

2)　ピン・アサインを設定する。

図 10－20 に示してあるように[Assignments]メニューの[Pins]を選択する。すると図 10－21 のような Assignment Editor というウィンドウが開く。

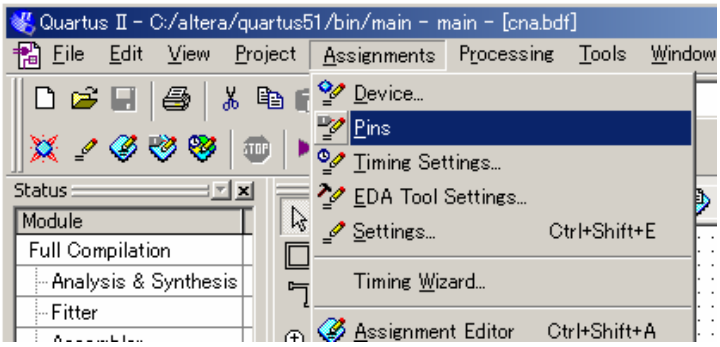



図 10－20

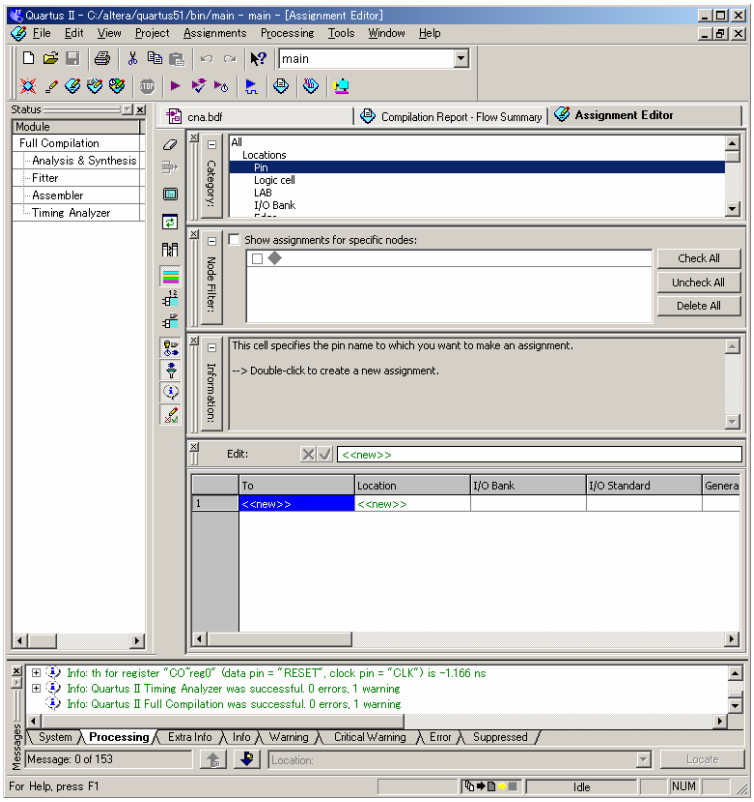


図 10－21

まず、入力ピンを割り当てる。図 10－21 に示すように、Category 欄で Pin が選ばれているのを確認し、[Edit]メニューから 4 で設定した名前のピンを選択するかその下の To の欄の<<new>>をダブルクリックして選択できる。すると、下のほうにある表に選択したピンに関するさまざまな条件を設定できるようになる。

指定したいのはピンの場所、つまり Location で、選択したピンの行の Location をダブルクリックし、図 10－22 のように示される。ここで設定したいピンを選ぶ。

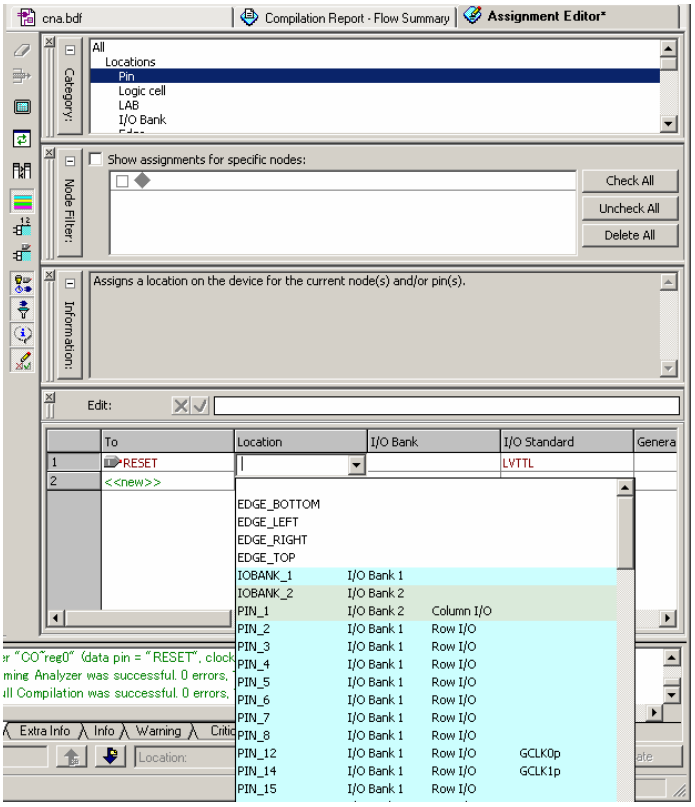



図 10－22

3） もう一度コンパイルする。

すべてのピン設定が終わったら[File]から[Save Project]を選択してプロジェクト全体をセーブし、再度コンパイルを実行する。

6　MAXII に書き込む

1）プログラムツールを開く

図 10－23 に示してあるように[Tool]メニューから[Programmer]を選択すると、MAXII に回路を書き込むソフトウェア[Programmer ツール]が起動する。

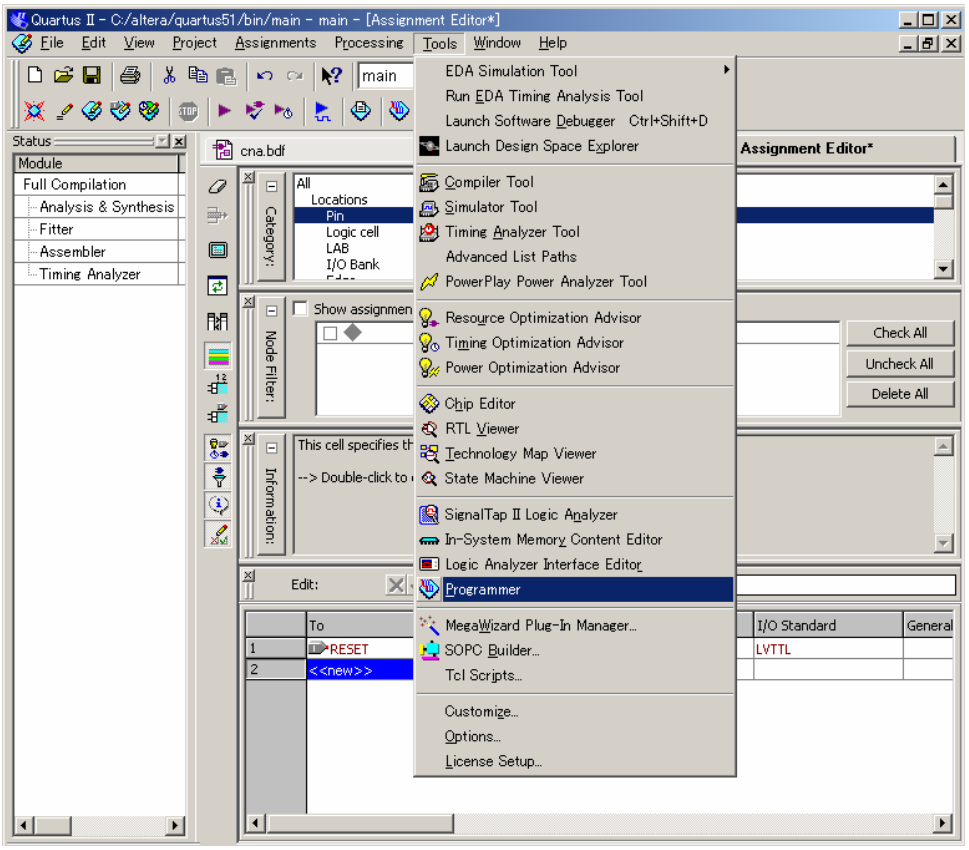



図 10－23

2）書き込みケーブルのセットアップ

次に、[Hardware Setup]というボタンを押す。すると、図 10－14 のようなダイアログが開かれるので、[Add Hardware]ボタンを押し、[ByteBlaster MV or ByteBlaster II]を選択する。Port 欄には、接続しているプリンタ・ポートの番号を選択する。[OK]ボタンを押すと ByteBlaster MV というハードウェアが追加される。また、このときに CPLD 基板とパソコンを接続しておくこと。

Hardware ダイアログを閉じたら[Auto Detect]というボタンを押す。これは、ケーブルの先にある CPLD が認識されると、EPM240 という名前の CPLD が発見される。されない場合はケーブルが正しく接続されてないか、CPLD 基板に電源が入っていないか、はんだ付けに不良がないか確認しなければならない。

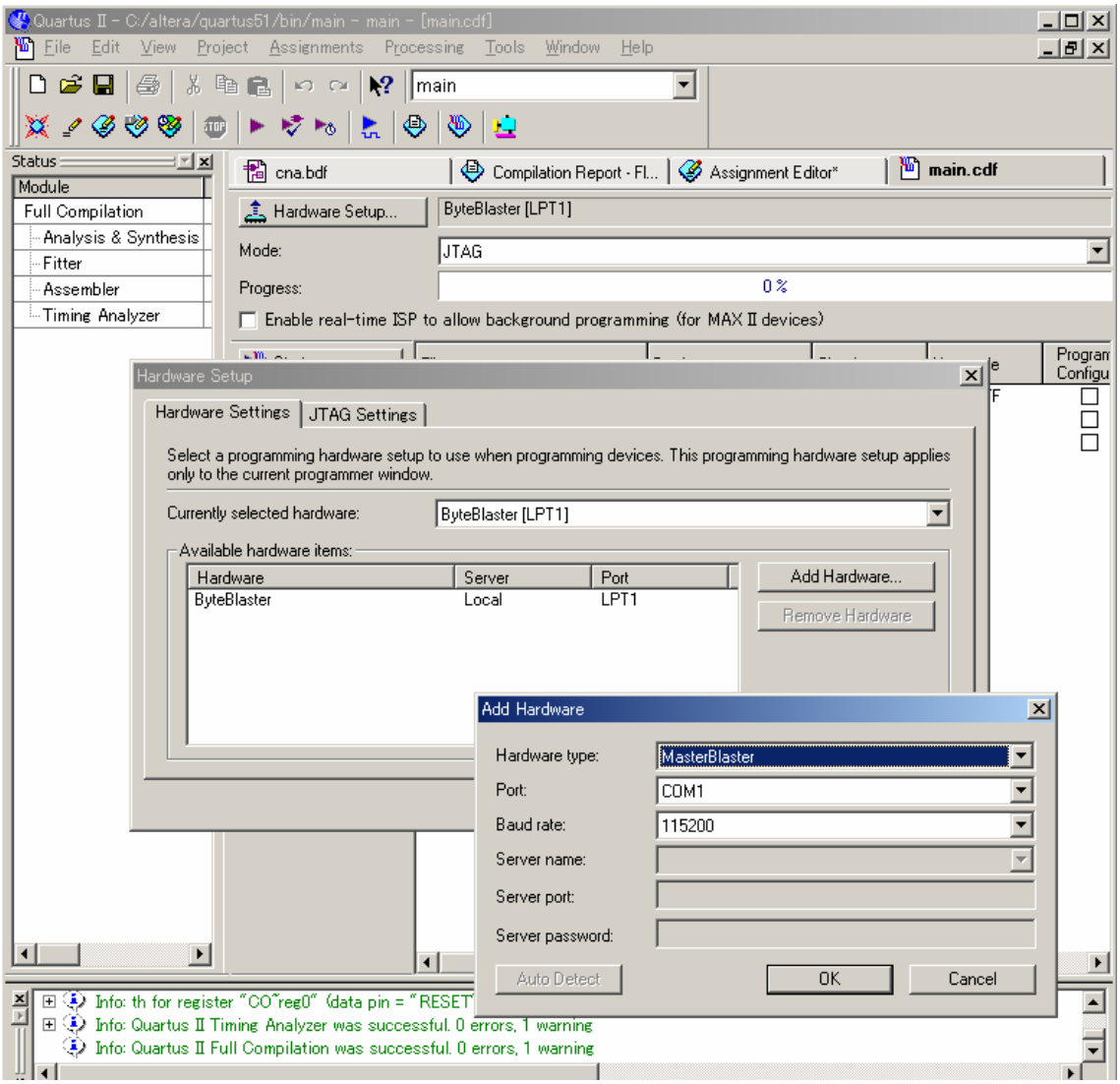



図 10－24

3) 書き込み

書きこむ前に図 10－25 に示すように Program/Configure と Verify の欄にチェックを入れる。次に CPLD 基板に電源が入っており、プリンタケーブルが接続されているかを確認し、[Start]を押すと書き込みが始まる。Progress が 100%になると書き込みが完了する。

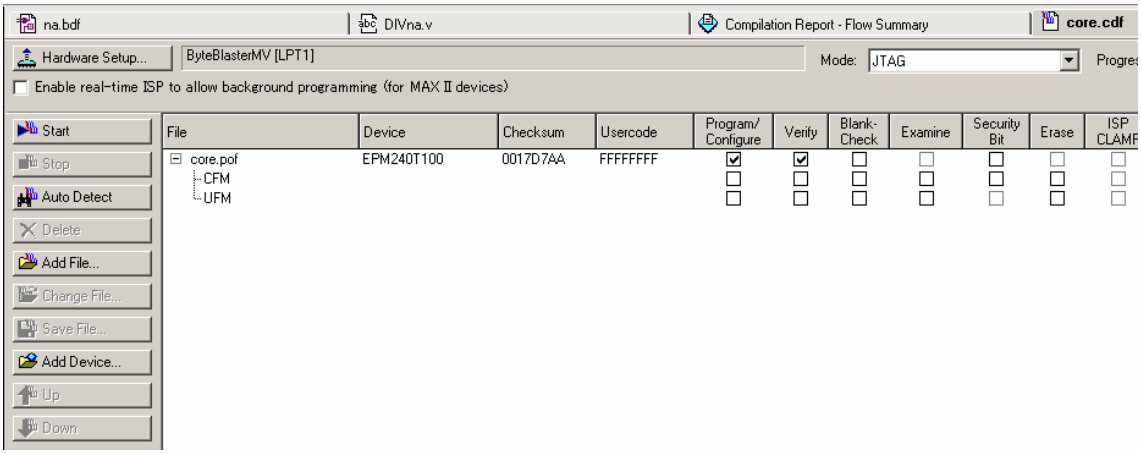

na.bdf
DIVna.v
Compilation Report - Flow Summary
core.cdf
Hardware Setup...
ByteBlasterMV [LPT1]
Mode: JTAG
Enable real-time ISP to allow background programming (for MAX II devices)
Start
Stop
Auto Detect
Delete
Add File...
Change File...
Save File...
Add Device...
Up
Down
File
Device
Checksum
Usercode
Program/ Configure
Verify
Blank-Check
Examine
Security Bit
Erase
ISP CLAMP
core.pof
EPM240T100
0017D7AA
FFFFFFFF
CFM
UFM

図 10－25