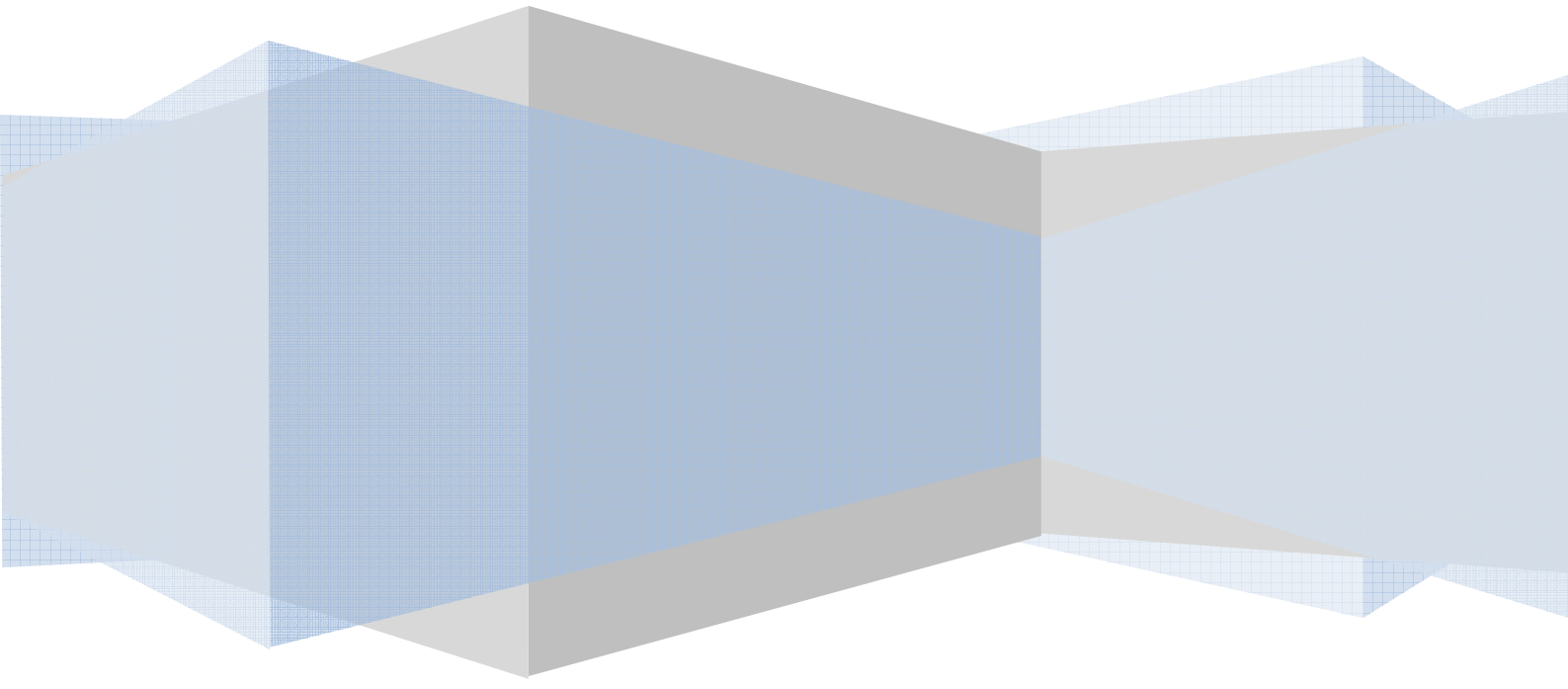
# JPEG DECODER

## V2.01 Data Sheet

Hidemi Ishihara



**改版履歴**

| Ver | Date | Description |
|---|---|---|
| 1.00 | 2006/10/01 | 初版 |
| 1.01 | 2006/10/04 | ラインセンスを LGPL にしました。<br>使っていないレジスタを削除しました。<br>JpegDecodeIdle の High になるタイミングを変更しました。 |
| 1.02 | 2007/02/22 | RAM のサイズを使用しているサイズに合わせました |
| 1.03 | 2007/04/13 | デコードの開始タイミングを変更(JpegDecodeStart の削除)<br>alywas(*)の使用禁止(function に変更)<br>実機検証済 |
| 1.04 | 2007/07/11 | Ver 1.03 との機能の違いはありません。<br>case 文に"// synthesis parallel_case"を追加しました。<br>iDCT の演算部分で Xilinx の XST で発生する不具合を記述で回避するようにしました。 |
| 1.05 | 2007/10/25 | ライセンスを CPL に変更しました。<br><br>iDCT<br>・演算部分で Xilinx の XST で発生する不具合を記述で回避するようにしました。<br><br>jpeg_regdata<br>・[31:0]=0xFF00FF00 の時、データのシフト量を間違え<br>・[15:0]=0xFF00 があった次に[39:24]=0xFF00 となった場合、シフトしてしまう<br>・元々、1 枚の JPEG データのデコード後は暴走する事が前提のカイロだったが最終データ(0xFFD9)を検出してから、Decode_FSM が ProcessIdle になった場合にデータを入力し内容にするように修正。<br>・RegWidth を 8bit から 7bit へ修正、及び、Width 計算の値をビット指定した<br><br>hm_decode<br>・ZERO が 15 個のカウント数を+1 足りなかった<br>・NextProcessCount は廃止<br>・ProcessCount を 7bit から 6bit に変更<br>・ステートから必要の無いステートを除去<br><br>ycbcr2rgb<br>・モジュール名の修正しました(ycbcbr2rgb→ycbcr2rgb)。<br>・組み合わせ回路にすべき部分でラッチを生成するようにしていたのを修正しました。<br><br>huffman |

| | | |
|---|---|---|
| | | ・全ての名称を haffuman ではなく huffman に修正しました。 |
| 2.00 | 2008/03/31 | 配布する梱包内容の見直し<br>特に問題ないので正式に Ver 2.00 へ昇格させました。 |
| 2.01 | 2009/10/05 | 配布する梱包内容の見直し<br>ドキュメントの整備、Xilinx ISE 11.3 でのインプリメント確認。 |

# 目次

# 1. 概要

本 JPEG DECORDER（以後、本回路）は入力された JPEG データをリアルタイム・デコードする Verilog HDL の RTL ソースコードです。

## 1.1. 特徴

- 仕様に定められている JPEG データをそのまま、デコードし、画像データを出力します。
- GIMP（GNU Image Manipulation Program）で JPEG 保存する場合のデフォルト設定の JPEG データをリアルタイム・デコードすることが可能です。
- 処理用の外付け RAM は必要ありません。ただし、入力用に FIFO、出力結果を収めるための何らかの領域は必要です。

## 1.2. 仕様

本回路では下記の JPEG データをデコードすることができます。

- ベースライン DCT
- ハフマン符号
- サンプリングは 4:1:1 のみ（4:2:2、4:4:4 についてはご要望があれば対応させていただきます）
- サムネイル不可
- DQT、DHT、SOF0、SOS、SOI。EOI のみデコードします
- DRI、RSTm、APP マーカは無視します

## 1.3. ライセンス

本回路は CPL(Common Public License、http://www.sopensource.org/licenses/cpl1.0.php)を適応しています。ライセンス条項に則っていれば商用・非商用問わず、自由に改変し、再配布および製品などに組み込んでいただいて問題ありません。

## 1.4. JPEG DECODER の処理能力

通常、JPEG デコードを VGA サイズで 30fps 処理できますと言い切るのはナンセンスと考えています。その理由は、デコードする JPEG データがどの程度圧縮されているか語られていないからです。もし、私であれば 30fps という数字を出されたら、それはすなわち非圧縮の JPEG データをデコードする状態、最悪の圧縮ケースでも 30fps を保障してくれるものと考えます。仮にランダム画像の JPEG データを処理する場合、JPEG データは圧縮効率が下がり極端にデコードにかかる処理時間が長くなります。JPEG デコードの処理速度は基本的に一般的な写真画像で語られることが多いと思いますが、どれが一般的な写真画像なのか？そこから議論しなければいけないことになるものと思います。本回路の処理能力はそれらをもとに述べません。本回路は 200～300KB 程度の JPGE データであれば、30fps を達成することを見込んでいます。しかし、1 データ辺りの処理時間は JPEG データ内のハフマンコード数に比例します。例えば、データが 300KB に収まっていたとしても、ハフマンコードの数が 300K 個であれば、おそらく、30fps は達成できないと思われます。GIMP で画像を作成するなら、画質は 1,920×1,080 サイズで圧縮率が 80～85%ぐらいに設

定すると出力される JPEG データは 200～300KB に収まります。これぐらいの JPEG データで本回路の動作周波数を 150MHz ぐらいにできれば、だいたい、30fps の処理能力を発揮できると思われます。今までに実施した Xilinx FPGA での合成結果例を表xに示ししておきます。

| Ver | Ver 1.01 | Ver 1.03 | Ver 1.05 | Ver 1.05 |
|---|---|---|---|---|
| ツール | Xilinx WebPack ISE 8.2i | Xilinx ISE 9.0.2i | Xilinx WebPack ISE 9.2i | Xilinx WebPack ISE 9.2i |
| 設定 | デバイス選択以外はデフォルト設定 | デバイス選択以外はデフォルト設定 | デバイス選択以外はデフォルト設定 | デバイス選択以外はデフォルト設定 |
| デバイス | XC4VLX25-10SF363C | XC4LX25-10SF363C | XC4LX25-10SF363C | XC5VLX30-3FF324C |
| スライス数 | 8,5758(81%) | 6,984(64%) | 9,014(83%) | 3,445(71%) |
| ブロック RAM | 16(22%) | 6(8%) | 6(8%) | 3(9%) |
| DSP | 21(43%) | 21(43%) | 21(43%) | 21(65%) |
| 最高動作周波数 | 84.189MHz | 81.070MHz<br>※100MHz で配置配線を行うと、85.778MHz まで向上する | 87.085MHz | 141.362MHz |

| Ver | 2.01 | 2.01 | 2.01 |
|---|---|---|---|
| ツール | Xilinx ISE 10.1.3 | Xilinx ISE 11.3 | Xilinx ISE 11.3 |
| 設定 | デバイス選択以外はデフォルト設定 | デバイス選択以外はデフォルト設定 | デバイス選択以外はデフォルト設定 |
| デバイス | XC5VLX30-3FF324 | XC5VLX30-3FF324 | XC6SLX16-3FTG256 |
| スライス数 | 4,421(23%) | 4,444(23%) | 4,492(24%) |
| 最高動作周波数 | 135.795MHz | 125.770MHz | 80.723MHz |

表 1:合成結果例

Ver 1.01 から Ver 1.03 ではブロック RAM が減っているのは使っていない RAM を減らしたので当然なのですがスライス数がずいぶんと減りました。しかし、デフォルトの設定では最高動作周波数が若干ながら落ちてしまいました。ツールが固定でなく、そこにも差が現れます。Ver 1.05 でターゲットデバイスを Virtex5 にしてみたところ、140MHz を超える結果が出ましたがどうも、これは正常な回路が生成されたわけではなかったようです。Xilinx ISE 11.3 及び 10.1.3 で確認したところ、ISE 11.3 で 125MHz、ISE 10.1.3 で 135MHz とでました。ISE 11.3 の方が 10MHz も違ってきました。さらに Virtex5 と Spartan6 と比べた場合、Spartan6 の方が良い結果がでると思いましたがそうではな

かったようです。

## 1.5. 同梱ファイル

本パッケージは下記のファイルが含まれています。

JPEG DECODER RTL ソースコード

JPEG DECODER シミュレーション用テストベンチ

JPEG DECODER C モデル

シミュレーション用テストベンチ支援アプリ ＆ ソースコード

### 1.5.1. RTL ソースコード

src ディレクトリには本回路、JPEG DECODER の RTL ソースコードが入っています。

| ファイル名 | 概要 |
|---|---|
| jpeg_decode.v | JPEG DECODER トップ階層 |
| jpeg_decode_fsm.v | JPEG マーカ・デコード |
| jpeg_dht.v | DHT テーブルメモリ |
| jpeg_dqt.v | DQT テーブルメモリ |
| jpeg_haffuman.v | ハフマン・デコードトップ階層 |
| jpeg_hm_decode.v | ハフマンデコード回路 |
| jpeg_idct.v | iDCT トップ階層 |
| jpeg_idctx.v | iDCT X 方向処理回路 |
| jpeg_idcty.v | iDCT Y 方向処理回路 |
| jpeg_regdata.v | JPEG データ読み込み回路 |
| jpeg_ycbcr.v | YCbCr-RGB 変換トップ階層 |
| jpeg_ycbcr2rgb.v | YCbCr-RGB 変換回路 |
| jpeg_ycbcr_mem.v | YCbCr メモリ |
| jpeg_ziguzagu.v | ジグザグ処理トップ階層 |
| jpeg_ziguzagu_reg.v | ジグザグ処理レジスタ |
| jpeg_test.v | JPEG DECODER テストベンチ |

表 2:RTL ソースコード

### 1.5.2. 実行形式のソフトウェア

c-model ディレクトリには実行形式のソフトウェアとそれら、ソフトウェアの C ソースが入っています。実行形式のソフトウェアは Linux 上で動作することを確認しています。もし、他の OS 環境で実行する場合やご使用になっている Linux ディストリビューションの環境で動作しない場合は、同梱されているソースコードをコンパイルしてお使いください。

| ファイル名 | 詳細 |
|---|---|
| djpeg | RTL ソースコードど同性能の JPEG DECODER の C モデルです。 |
| convbtoh | JPEG のバイナリファイルをシミュレーションで使用できるように 16 進数のファイルに変換します。 |

| | |
|---|---|
| convsim | シミュレーションで本回路が出力した RGB データを BITMAP に変換します。 |
| djpeg.c | djpeg のソースコードです。 |
| convbtoh.c | convbtoh のソースコードです。 |
| convsim.c | convsim のソースコードです。 |

表 3:実行形式のソフトウェア

### 1.5.3. テストベンチ

testbench ディレクトリにはシミュレーション用のファイルを収めています。

| ファイル名 | 詳細 |
|---|---|
| run.ms | シミュレーション用実行スクリプト |
| jpeg_test.v | テストベンチ |

表 4:テストベンチ

# 2. 機能

## 2.1. ブロック図

本回路は図 1 のように構成されています。

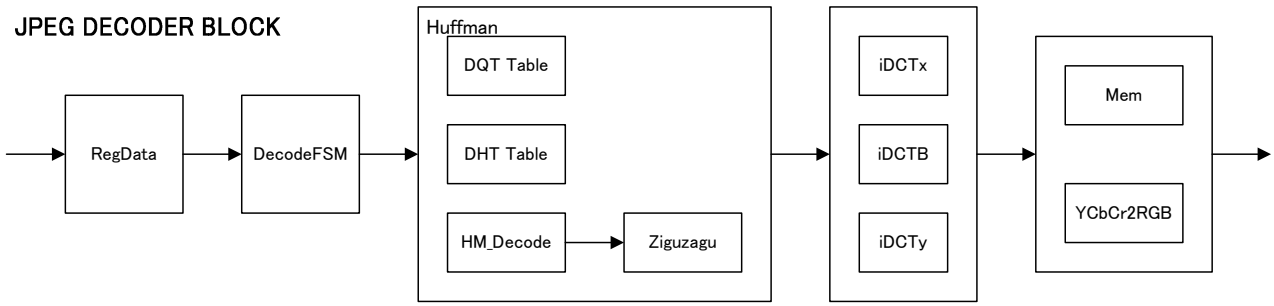



図 1:ブロック図

## 2.2. 入出力信号

| 信号名 | I/O | bit | 詳細 |
|---|---|---|---|
| rst | IN | 1 | 非同期リセット(Active Low) |
| clk | IN | 1 | クロック |
| DataIn | IN | 32 | JPEG データ入力<br>デコードする JPEG データを入力します。DataInEnable が High の時、リトル・エンディアンでデータが有効でなければいけません。通常は FIFO の ReadData に接続します。 |
| DatInEnable | IN | 1 | JPGE データ・イネーブル<br>High で DataIn が有効であることを示します。通常は FIFO の Empty 信号の反転信号に接続します。 |
| DataInRead | OUT | 1 | JPEG データ・リード<br>High で DataIn をリードしたことを示します。通常は FIFO の ReadEnable に接続します。 |
| JpegDecodeIdle | OUT | 1 | JPEG デコード・アイドル<br>High で JPEG DECODER がアイドル状態であることを示します。この信号が High でない場合に、処理中以外の新しい JPEG データを入力してはいけません。 |
| OutEnable | OUT | 1 | 出力画像データ・イネーブル<br>High で出力画像データ(OutR、OutG、OutB)を出力していることを示します。 |
| OutWidth | OUT | 16 | 出力画像データ・サイズ(X 方向)<br>処理中の出力画像データサイズの幅を示しています。 |

| | | | |
|---|---|---|---|
| OutHeight | OUT | 16 | 出力画像データ･サイズ（Y 方向）<br>処理中の出力画像データサイズの高さを示しています。 |
| OutPixelX | OUT | 16 | 出力画像データ･X 位置<br>出力中の出力画像データの横方向の位置を示しています。 |
| OutPixelY | OUT | 16 | 出力画像データ･Y 位置<br>出力中の出力画像データの縦方向の位置を示しています。 |
| OutR | OUT | 8 | 出力画像データ（赤） |
| OutG | OUT | 8 | 出力画像データ（緑） |
| OutB | OUT | 8 | 出力画像データ（青） |

表 5:入出力信号

## 2.3. 動作概要

本回路は外部回路と図 2 のように入力側に FIFO、出力側にメモリコントローラなどを置くことと想定しています。

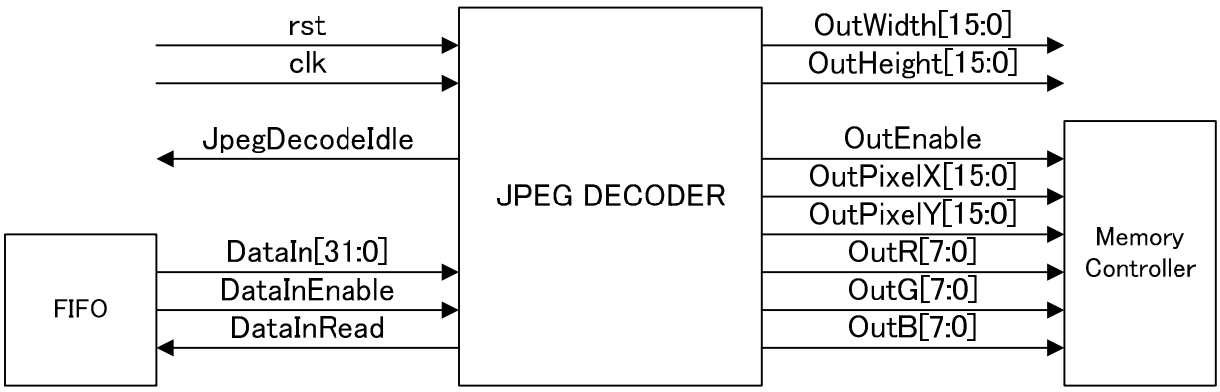



図 2:動作概要ブロック図

FIFO は単純な FIFO を接続していただければ問題ありません。ここでは特に FIFO の説明は省きます。出力側はメモリコントローラなどを置くのが望ましいですがデータ幅が 24bit のメモリとおくなら保存アドレスは OutWidth × OutPixelY＋OutPixrlX の位置になります。Ver 1.03 から本回路はリセットされると IDLE 状態になり FIFO のデータが有効になるとすぐにデコードの処理を開始します。本回路は JpegDocodeIdle が”1”のとき、新たに JPEG データを処理できる準備ができていることを示しています。1 つの JPEG データを処理中に JpegDecodeIdle が”0”の時に新たに JPEG データをデコードしようとしても、正常に処理されません。

## 2.4. タイミングチャート

図 3 にタイミングチャートを示します。

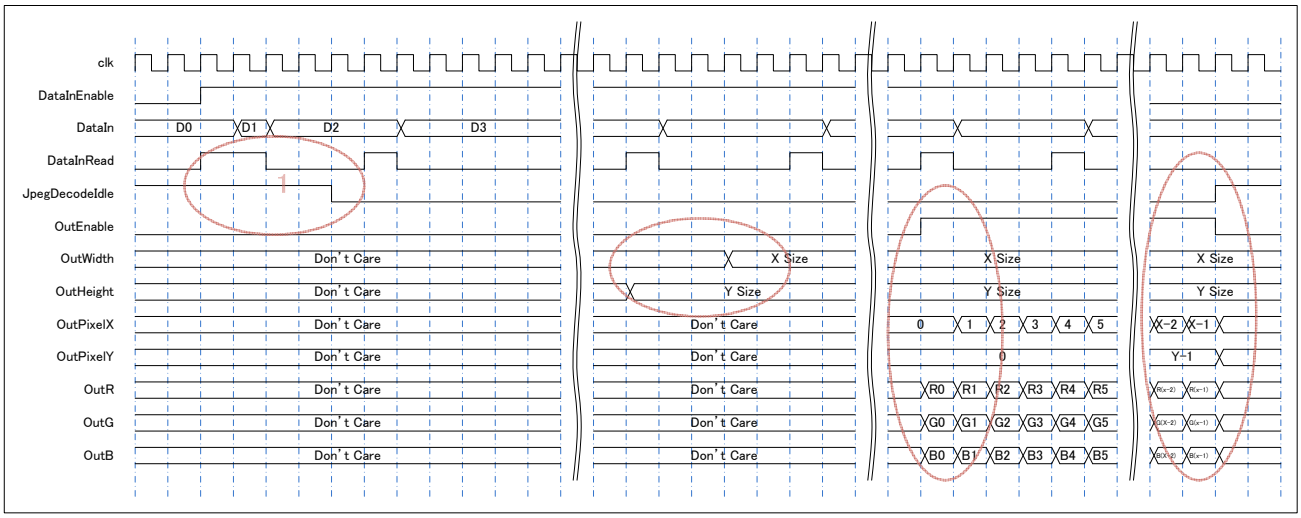



図 3:タイミングチャート

図 3 には A1～A4 からのパートを示していますがこれらのパートは下記のようになっています。

### A1:JPEG データの読み込み開始

JpegDecodeIdle が"0"でかつ、DataInEnable が"1"になると、JPEG DECODER は新しい JPEG データが準備されたと判断し、データを読み込み始めます。ただし、読み込み始めた時には JpegDecodeIdle は"0"にならないので注意してください。(データの読み込みと同時に JpegDecodeIdle が"0"になることを望む方は改造してください)

### A2:画像サイズの確定(出力)

JPEG データを解析中に画像のサイズが確定します。画像サイズが確定したことを示すフラグ信号は準備していません。もし、画像サイズを確定したことを示すフラグ信号が必要な場合は追加してください。

### A3:最初の画像データの出力

ここでは画像データの最初のデータの出力を示しています。

### A4:最後の画像データの出力

ここでは画像データの最後のデータの出力を示しています。最後のデータが出力されたあとは OutEnable が"0"になるのと同時に、JpegDecodeIdle も"0"になります。それ以前に、DataInEnable は"0"になっているものと思います。

# 3. モジュール

本章では各モジュールについて簡単に解説します。

## 3.1. jpeg_regdata

jpeg_regdata は Deocde FSM、Huffman ブロックで使用されるデータを出力します。各ブロックで使用されたビット数分を差し引き、次のデータを用意します。

| 信号名 | I/O | bit | 詳細 |
|---|---|---|---|
| DataIn | IN | 32 | データ入力<br>通常は FIFO の ReadData に接続します。 |
| DataInEnable | IN | 1 | データ・イネーブル<br>1 のとき、DataIn にデータを準備できていることを示します。 |
| DataInRead | OUT | 1 | データ・リード<br>1 のとき、DataIn からデータを読み込んだことを示します。<br>通常は FIFO の ReadEnable 信号に接続します。 |
| DataOut | OUT | 32 | データ出力 |
| DataOutEnable | OUT | 1 | データ出力イネーブル<br>1 のとき、DataOut にデータが準備できたことを示します。 |
| ImageEnable | IN | 1 | イメージデータ・イネーブル<br>1 のとき、JPEG デコード処理はイメージデコード処理に入っていることを示します。 |
| ProcessIdle | IN | 1 | 処理アイドル<br>1 のとき、処理がアイドル状態であることを示します。 |
| UseBit | IN | 1 | 使用ビット<br>1 のとき、DataOut が UseWidth 分のビットが使用されたことを示します。 |
| UseWidth | IN | 1 | 使用ビット幅<br>使用されたビット幅を示します。 |
| UseByte | IN | 1 | 使用バイト<br>1 のとき、DataOut が 1 バイト使用されたことを示します。 |
| UseWord | IN | 1 | 使用ワード<br>1 のとき、DataOut が 2 バイト使用されたことを示します。 |

表 6:信号詳細

### 3.1.1. DecodeFSM

DecodeFSM は JPEG データのヘッダの解析を行い、DHT や DHQ のデータをそれぞれのテーブルに保存します。

| 信号名 | I/O | bit | 詳細 |
|---|---|---|---|
| DataInEnable | IN | 1 | データ入力イネーブル |
| DataIn | IN | 32 | データ入力 |
| JpegDecodeIdle | OUT | 1 | データ・デコード・アイドル |
| OutWidth | OUT | 16 | 画像サイズ（X 方向） |
| OutHeight | OUT | 16 | 画像サイズ（Y 方向） |
| OutBlockWidth | OUT | 12 | ブロックサイズ |
| OutEnable | IN | 1 | 出力イネーブル |
| OutPixelX | IN | 16 | 出力位置（X 方向） |
| OutPixelY | IN | 16 | 出力位置（Y 方向） |
| DqtEnable | OUT | 1 | DQT イネーブル |
| DqtTable | OUT | 1 | DQT テーブル |
| DqtCount | OUT | 6 | DQT カウント |
| DqtData | OUT | 8 | DQT データ |
| DhtEnable | OUT | 1 | DHT イネーブル |
| DhtTable | OUT | 2 | DHT テーブル |
| DhtCount | OUT | 8 | DHT カウント |
| DhtData | OUT | 8 | DHT データ |
| HuffmanEnable | OUT | 1 | ハフマン・イネーブル |
| HuffmanTable | OUT | 2 | ハフマン・テーブル |
| HuffmanCount | OUT | 4 | ハフマン・カウント |
| HuffmanData | OUT | 16 | ハフマン・データ |
| HuffmanStart | OUT | 8 | ハフマン・スタート |
| ImageEnable | OUT | 1 | イメージ・イネーブル |

| | | | |
|---|---|---|---|
| UseByte | OUT | 1 | 使用バイト |
| UseWord | OUT | 1 | 使用ワード |

表 7:信号詳細

### 3.1.2. jpeg_hufman

jpeg_huffman ではハフマン符号の複合を行い、ジグザグ変換を行います。

| 信号名 | I/O | bit | 詳細 |
|---|---|---|---|
| DqtInEnable | IN | 1 | DQT 入力イネーブル |
| DqtInColor | IN | 1 | DQT 入力カラー |
| DqtInCount | IN | 6 | DQT 入力カウント |
| DqtData | IN | 8 | DQT データ |
| DhtInEnable | IN | 1 | DHT 入力イネーブル |
| DhtInColor | IN | 2 | DHT 入力カラー |
| DhtInCount | IN | 8 | DHT 入力カウント |
| DhtInData | IN | 8 | DHT 入力データ |
| HuffmanTableEnable | IN | 1 | ハフマン・イネーブル |
| HuffmanTableColor | IN | 2 | ハフマン・カラー |
| HuffmanTableCount | IN | 4 | ハフマン・カウント |
| HuffmanTableCode | IN | 16 | ハフマン・コード |
| HuffmanTableStart | IN | 8 | ハフマン・スタート |
| DataInRun | IN | 1 | データ入力動作 |
| DataInEnable | IN | 1 | データ入力イネーブル |
| DataIn | IN | 32 | データ入力 |
| DecodeUseBit | OUT | 1 | デコード使用ビット |
| DecodeUseWidth | OUT | 7 | デコード使用ビット幅 |
| DataOutIdle | IN | 1 | データ出力アイドル |
| DataOutEnable | OUT | 1 | データ出力イネーブル |
| DataOutColor | OUT | 3 | データ出力カラー |
| DataOutSel | IN | 1 | データ出力セレクト |
| Data[00:63]Reg | OUT | 16 | データレジスタ |
| DataOutRelease | IN | 1 | データ出力リリース |

表 8:信号詳細

Huffman 符号の復号後に ziguzagu スキャンを戻しますが、ziguzagu スキャンを戻すとは Huffman でデコードできる順番を 8×8 のブロックで下図のようにデータを置き換えます。

| | X0 | X1 | X2 | X3 | X4 | X5 | X6 | X7 |
|---|---|---|---|---|---|---|---|---|
| **Y0** | 0 | 1 | 5 | 6 | 14 | 15 | 27 | 28 |
| **Y1** | 2 | 4 | 7 | 13 | 16 | 26 | 29 | 42 |
| **Y2** | 3 | 8 | 12 | 17 | 25 | 30 | 41 | 43 |
| **Y3** | 9 | 11 | 18 | 24 | 31 | 40 | 44 | 53 |
| **Y4** | 10 | 19 | 23 | 32 | 39 | 45 | 52 | 54 |
| **Y5** | 20 | 22 | 33 | 38 | 46 | 51 | 55 | 60 |
| **Y6** | 21 | 34 | 37 | 47 | 50 | 56 | 59 | 61 |
| **Y7** | 35 | 36 | 48 | 49 | 57 | 58 | 62 | 63 |

表 9:ziguzagu スキャンを戻す対応表

※huffman の ziguzagu 以降、データは 2Way のバスで処理されています。これは iDCT の処理時間が最低限、X 方向に 64 クロック、Y 方向に 64 クロックのウェイトを入れなければいけないので 2Way 機構にしています。

### 3.1.3. jpeg_idct

jpeg_idct は iDCT 処理を行います。処理の関係上、2Way 構成にしています。

| 信号名 | I/O | bit | 詳細 |
|---|---|---|---|
| DataInEnable | IN | 1 | データ入力イネーブル |
| DataInSel | OUT | 1 | データ入力セレクト |
| Data[00:63]In | IN | 16 | データ入力 |
| DataInIdle | OUT | 1 | データ入力アイドル |
| DataInRelease | OUT | 1 | データ入力リリース |
| DataOutEnable | OUT | 1 | データ出力イネーブル |
| DataOutPage | OUT | 3 | データ出力ページ |
| DataOutCount | OUT | 2 | データ出力カウント |
| Data0Out | OUT | 9 | データ出力0 |
| Data1Out | OUT | 9 | データ出力1 |

表 10:信号詳細

iDCTx は以下のようにデータを使用します。 下図の(X-Y)とは、ziguzagu スキャンを戻した際の X-Y 位置で(Y 位置×8+X 位置)の場所を指しています。Huffman とは Huffman デコードできた順番で上図の中に書いてある番号で ziguzagu スキャンをする前の番号です。A/B 側となっているのは、iDCTx/y の処理は(A×数値+B×数値)で計算されていく部分があるのでわかれています。

| ワード数 | (X-Y)A 側 | (X-Y)B 側 | Huffman(A 側) | Huffman(B 側) |
|---|---|---|---|---|
| **0** | 0 | 4 | 0 | 14 |
| **1** | 2 | 6 | 5 | 27 |
| **2** | 1 | 7 | 1 | 28 |
| **3** | 5 | 3 | 15 | 6 |
| **4** | 8 | 12 | 2 | 16 |
| **5** | 10 | 14 | 7 | 29 |
| **6** | 9 | 15 | 4 | 42 |
| **7** | 13 | 11 | 26 | 13 |
| **8** | 16 | 20 | 3 | 25 |
| **9** | 18 | 22 | 12 | 41 |
| **10** | 17 | 23 | 8 | 43 |
| **11** | 21 | 19 | 30 | 17 |
| **12** | 24 | 28 | 9 | 31 |

| | | | | | |
|---|---|---|---|---|---|
| **13** | 26 | 30 | 18 | 44 |
| **14** | 25 | 31 | 11 | 53 |
| **15** | 29 | 27 | 40 | 24 |
| **16** | 32 | 36 | 10 | 39 |
| **17** | 34 | 38 | 23 | 52 |
| **18** | 33 | 39 | 19 | 54 |
| **19** | 37 | 35 | 45 | 32 |
| **20** | 40 | 44 | 20 | 46 |
| **21** | 42 | 46 | 33 | 55 |
| **22** | 41 | 47 | 22 | 60 |
| **23** | 45 | 43 | 51 | 38 |
| **24** | 48 | 52 | 21 | 50 |
| **25** | 50 | 54 | 37 | 59 |
| **26** | 49 | 55 | 34 | 61 |
| **27** | 53 | 51 | 56 | 47 |
| **28** | 56 | 60 | 35 | 57 |
| **29** | 58 | 62 | 48 | 62 |
| **30** | 57 | 63 | 36 | 63 |
| **31** | 61 | 59 | 58 | 49 |

表 11:iDCT 処理対応表

RAM は 4 バンク分あり、0 から順番に収められていきます。 RAM 以外に、バンク毎に 32bit×2(RAM の A/B を示す)の Enable Bit があり、各バンクの 0 が書き込まれたときに 0 以外の Enable Bit を 0 にします。 Enable Bit は入力されるアドレスの Bit を 1 にします。 iDCTx に出力する際は Enable Bit が 1 の場合、出力するバンクの RAM データを出力し、0 の場合は 0 を出力します。iDCTx ではこのような処理を行い、iDCTy でも同様な処理が行われます。

### 3.1.4. ycbcr2rgb

ycbcr2rgb は iDCT されたデータを YCbCr から RGB に変換します。

| 信号名 | I/O | bit | 詳細 |
|---|---|---|---|
| DataInEnable | IN | 1 | データ入力イネーブル |
| DataInPage | IN | 3 | データ入力ページ |
| DataInCount | IN | 2 | データ入力カウント |
| DataInIdle | OUT | 1 | データ入力アイドル |
| Data0In | IN | 9 | データ入力0 |
| Data1In | IN | 9 | データ入力1 |
| DataInBlockWidth | IN | 12 | データ入力ブロック幅 |
| OutaEnable | OUT | 1 | データ出力イネーブル<br>1のとき、OutPixelX/Y に位置情報、OutR/G/B に RGB データが出力されます。 |
| OutPixelX | OUT | 16 | データ出力位置(X 方向) |
| OutPixelY | OUT | 16 | データ出力位置(Y 方向) |
| OutR | OUT | 8 | データ出力(R) |
| OutG | OUT | 8 | データ出力(G) |
| OutB | OUT | 8 | データ出力(B) |

表 12:信号詳細

# 4. 検証

本パッケージには 3 つの検証環境を同梱しています。論理検証（論理シミュレーション）は本回路単体でのシミュレーションを行います。ゲート・シミュレーションでは、Xilinx 環境を使用したタイミング・シミュレーションです。最後に、システム検証があります。システム検証では本回路が PCI-BUS に接続されたことを想定して、限定したサイズではありますが実機さながらのシミュレーションを行います。

## 4.1. 論理検証（論理シミュレーション）

本パッケージには、論理シミュレーション環境が入っています。論理シュミレーション環境は ModelSim が使用できる環境であれば、コマンドを実行するだけで自動的にシミュレーションを行うことができます。シミュレーションを実施する際には、デコード元になる JPEG ファイルをご用意してください。本回路で処理できるマーカしかない JPEG ファイルは、GIMP で JPEG ファイルを出力することで簡単に作成することができます。image ディレクトリにシミュレーションで使用したい JPEG ファイルを置き、下記のようにコマンドを実行するとシミュレーションを行うことができます。

% run.ms JPEG ファイル名(拡張子は要らない)

このパッケージには、あらかじめ、test.jpg という JPEG ファイルを用意していますので下記のように実行することが可能です。

% run.ms test

シミュレーションでは test.jpg をデコードし、sim.dat というファイルを結果として生成します。その結果(sim.dat)から out?test.bmp というビットマップファイルを生成します。出来上がった out_test.bmp と test.jpg を見比べて、問題なければデコードは成功しています。デコード元の JPEG ファイルとシミュレーション結果の画像をどう見比べればいいかというと、GIMP や PhotoShop などで両方の画像を置いて色成分の差分をとることで見比べることができます。ここで注意しなければいけないのは JPEG ファイルをデコードする際には、計算劣化が発生していますので、出来上がったビットマップファイルの差分を比較すると色成分が 256 段階であれば、1 や 2 ぐらいの誤差が出る部分があります。また、シミュレーションを実行するとモジュールのロード後、下記のようなメッセージが表示されます。

```
# Start Clock: 3
# End Clock: 2526
```

このメッセージは JPEG デコードの処理を開始したクロックと、JPEG デコードが完了したクロックを示しています。上記の場合、1 つの JPEG データをデコードするのに、2,523 クロック必要であることを示しています。

## 4.2. ゲートシミュレーション

ゲートシミュレーションは ModelSim および Xilinx 社 ISE(確認している環境は Foundation WebPack ISE9.2i)がインストールされている環境であれば、コマンドのみでシミュレーションを実行することができます。まず、インプリメント（xxを参照のこと）を行います。インプリメントが完了すれば、論理シミュレーションと同様に下記のコマンドでゲートシミュレーションを行うことができます。

```
% run.fpga test
```

結果は論理シミュレーションと同じように表示され、出力結果として out_test.bmp が生成されます。なお、ゲートシミュレーションを行う前に、下記のコマンドを使用して ModelSim 用に Xilinx のライブラリを生成しておかなければいけません。（ISE に付属している ModelSim XIlinx 版を使用する場合はすでにライブラリが用意されていのでこの限りではありません）

```
% comxlib –s mti_se –f virtex5 –l verilog
```

※注意事項

ゲートシミュレーション実行中に、RAM 関係のエラーが発生します。このエラーは同一クロックまたは次のクロックで RAM の 2 つのポートで同じアドレスに片側のポートで書き込み、もう一方をリードしている場合にデータが用意できていないというエラーです。本回路上、この状態になる場合がありますがエラー発生時の読み込んだデータは使用していないので問題ありません。

ゲートシミュレーションは遅延計算も含まれていますので非常に時間がかかります。大きな JPEG データを検証するとシミュレーション完了まで数時間から数十時間かかることがあるので注意してください。

## 4.3. システム検証

このパッケージには実機検証が簡単に行えるように PCI-BUS も接続した簡単なシステムを同梱しています。この実機検証のシステムをシミュレーションと合わせて使用できます。

### 4.3.1. 仕様

- PCI-BUS Rev 2.1 32bit × 33MHz のターゲット
- JPEG データのデコード後の大きさは 256 ドット×256 ドットのみ展開されます

### 4.3.2. ブロック図

システム検証のブロック図を図xに示します。システム検証の構成はx章「機能」の「動作概要」の項に示したように入力側に FIFO を、出力側に 24bit × 65,536Word のメモリを置いています。

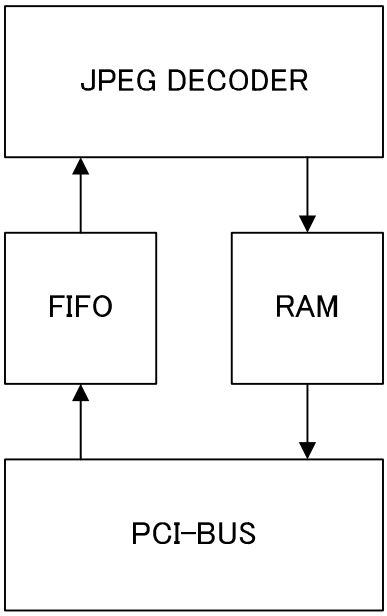



図 4:シミュレーション環境ブロック図

### 4.3.3. PCI-BUS メモリマップ

表xに PCI-BUS 上に見えるメモリマップを示しています。これらのメモリ空間は PCI-BUS の BASE0 をベースアドレスにして見ることができます。

| アドレス | |
|---|---|
| 0x00000～0x3FFFF | |
| 0x40000 | コマンドレジスタ(未使用) |
| 0x40004 | 割り込みレジスタ(未使用) |
| 0x40008 | ステータスレジスタ<br>[3] JPEG DECODER IDLE<br>[2] FIFO ALMOST FULL |

| | |
|---|---|
| | [1] FIFO FULL<br>[0] COMMAND BUSY(未使用) |
| 0x4000C | FIFO 書き込みレジスタ<br>ここにデータを書き込むと FIFO にデータが溜まっていきます。 |
| 0x40010 | 汎用レジスタ(未使用) |
| 0x40014 | ステータスレジスタ(未使用) |
| 0x4001C | JPEG DECODER CONTROL<br>[0] リセット |
| 0x40020～0x7FFFF | 未使用 |

表 13:PCI-BUS アドレスマップ

### 4.3.4. シミュレーションの実行手順

fpga/testbench ディレクトリへ移動して、単体の論理シミュレーションと同じように下記のコマンドを実行します。

% run.ms JPEG ファイル名(拡張子は必要ない)

全て PCI-BUS を通じてアクセスするシミュレーションになっていますのでシミュレーションが終わるまでに若干の時間がかかります。

# 5. インプリメント

このパッケージはインプリメント環境として、Altera 社 StratixII と Xilinx 社 Virtex5 でインプリメントできる環境を同梱しています。

## 5.1. Altera 社 StratixII インプリメント

Altera 社 StratixII をインプリメントする際の構成はシステム検証で用いた回路を使用します。したがって、PCI-BUS を用いた回路をインプリメントします。（たしか、QuartusII 6 でインプリメントが可能です）

fpga/testbench にある、テストベンチのソース以外の RTL ソースと本回路の RTL ソースを使用してインプリメントします。内部クロックは全て PCI-BUS のクロックを使用しますが本回路だけ、別のクロックを使用して検証したい場合はトップ階層に別のクロックを準備し、トップ階層にある sys_clk と入れ替えることで本回路だけを別のクロックで稼働させることができます。

fpga ディレクトリで下記のようにコマンドを実行するとインプリメントが行えます。

% quartus_sh –flow compile djpeg_fpga –c djpeg_fpga

## 5.2. Xilinx 社 Virtex5 インプリメント

Xilinx 社 Virtex5 をインプリメントする際の構成は本回路のみをインプリメントします。fpga ディレクトリで下記のようにコマンドを事項するとインプリメントを行うことができます。Xilinx ISE バージョンは 11.3 に合わせこんでいます。ほかのバージョンの場合、同梱のファイルを修正していただくとコマンドラインでインプリメントが可能です。

% jpeg_decode.cmd

上記のコマンドを実行すると Xilinx ディレクトリを生成し、その中でインプリメントを行います。インプリメントを完了すると、ゲートシミュレーション用に下記のファイルが生成され、その後、testbench ディレクトリにコピーされます。

jpeg_decode.v
jpeg_decode.sdf

## 5.3. 実機確認用アプリケーション

fpga/etst_program に Windows 用のアプリケーションが入っています。これは Microsoft Visual C++ 2005 Express でコンパイルが可能なソースになっています。システムをインプリメントして、FPGA にロードできたらアプリケーションを実行して、ld コマンドを使用すると PC 上の JPEG ファイルを FPGA にアップロードして、FPGA 上の JPEG DECODER で JPEG データをデコードします。mb コマンドを使用するとデコードした RGB のデータを BITMAP で保存することができます。

※注意

アプリケーションをご使用の際にはガジマルの森（http:/www.otto.to/~kasiwano/）の pcidebug.lzh から pcidebug.dll と pcidebus.sys などのライブラリをダウンロードしてください。

# 6. 最後に

## 6.1. サポート

本回路について、機能拡張やご質問などのサポートをメールにて行っております。ただし、ご回答レスポンスの速さは期待しないでください。ご質問などに関しては hidemi@sweetcafe.jp まで送っていただければ対応させていただきます。なお、多くのメールを受信しておりますのでメールを送信していただく際にはサブジェクトをわかりやすく目立つようなサブジェクトにしていただけると幸いです。また、デバイスを指定したサポートの場合、実機環境をお貸しいただければ、サポートさせていただくこともございますのでご気楽にご連絡ください。

# 7. 参考資料

## 7.1. JPEG 前提条件

本回路を改造する場合に参考にしてください。

1. フォーマット上、16nit のデータが 1 個の場合、1 ブロックは
   (ハフマンコード(16bit) + データ(16bit)) × 縦 8Pixel × 横 8Pixel = 2,048bit
   これが理論値としての最大値となる

2. ハフマンコードの復元はハード上、最大処理時間 $\tau$ は他の全てが 0 個で、16bit 目が 162 個存在する場合で 15 $\tau$ +162 $\tau$ =177 $\tau$ かかる

3. フォーマット上は 32bit の圧縮コードの中に最低、データが 1 個ある。

4. 1 ブロックは 1 つの DC 成分と 63 個の AC 成分から構成される

5. EOB が現れるとそのブロックの残りは全てゼロである

6. ZRL はゼロランが 15 を超えた場合に、ゼロラン 15 個示すハフマンコードである
   ZRL の場合、数値データが続くのではなくハフマンコードが続く

7. 1 ブロックは下記を達成すると終了になる
   EOB が現れた場合
   要素が 64 個を超えた場合

8. データは最大 16bit×64Word 以内にある
   EOB＝0x00、ZRL＝0xF0

9. Vm のフォーマット

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| ゼロランの個数 | | | | データのビット数 | | | |