第3章

JPEGエンコーダのSystemCモデルを作成してビヘイビア合成

―― ツールにまかせる部分と人手で最適化する部分を見極める

赤星博輝

ここでは，ディジタル・カメラのJPEGエンコーダの開発をイメージしながら，SystemCとビヘイビア合成の利用方法を解説する．JPEGエンコーダの処理のうち，ハードウェア化したときの効果が高いと思われるRGB-YUV変換とDCT（離散コサイン変換）の部分についてSystemCモデルを作成し，ビヘイビア合成を実施した．ビヘイビア合成ツールにすべてをまかせることは現実的ではなく，人手による最適化を加えて性能を逐次改善する必要がある．（編集部）

最近のLSI設計は規模に対して設計期間が短いことが多いのですが，今回の記事も設計期間（と執筆期間）がたいへん短く，その制約の中でビヘイビア合成を用いてどのようにハードウェア化を行ったかについて紹介します．

設計期間が短いことから，設計の手戻りを極力避ける必要があり，最初に設計フローを検討しました．検討した設計フローでは，まず全体の分析を行うためにSystemCでモデリングし，次にそのモデルを分析してビヘイビア合成を行う場所を決定し，続いてビヘイビア合成用のSystemCモデルを作成します（**図1**）．ビヘイビア合成用モデルの記述を，ただ合成可能なだけのモデルから，よりサイクル数の短い回路を生成するモデルへ書き換えました．その後，ビヘイビア合成を行った結果をフィードバックした分析モデルを作成し，少し詳細な評価を行いました．

ここではビヘイビア合成ツールとして，礎デザインオートメーションの「DesignPrototyper」を利用していますが，最適化の考えかたなどは，ほかのツールでも共通に適用できると思います．

● JPEGエンコーダ部を対象に設計

今回，ハードウェア化を検討したのはDCT（離散コサイン変換）を基本としたJPEG（Joint Photographic Experts Group）のエンコーダ部です（p.52のコラム「DCTとは」を参照）．JPEGは画像圧縮/伸張方式の一つで，非可逆変換と可逆変換，およびモノクロとカラーに対応しています．

処理概要を**図2**に示します．入力された画像がRGBで与えられます．これをYUV（YC_bC_r）に変換し，その変換されたYUVに対してDCT処理を行い，周波数空間に変換します（p.54のコラム「RGBとYUV」を参照）．その後，量子化処理で信号レベルを制限し，ハフマン符号化を行います．量子化処理では，低い周波数の信号は詳細に量子化し，高い周波数の信号は粗く量子化することで圧縮率を上げています．また，ハフマン符号ではよく出現する符号に短いコードを，あまり出現しない符号に長いコードを割り当てる

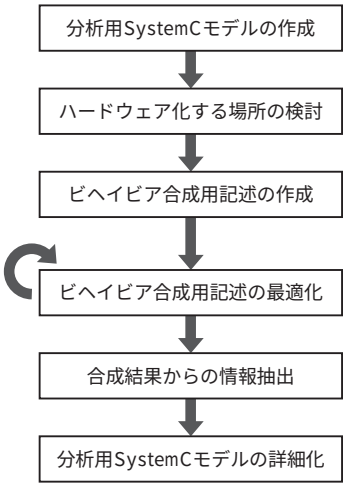

図1　今回の設計フロー

複数のことを同時に行うと失敗する可能性が高く，失敗したときの解析に時間がかかる．「急がば回れ」の発想で設計フローを考えた．

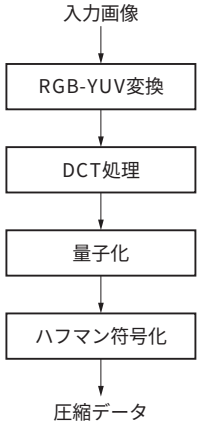

図2　JPEGエンコーダの概要

入力画像のRGBデータをYUV（YC_bC_r）データに変換する．その変換されたYUVデータに対してDCT処理を行い，周波数空間に変換する．その後，量子化処理で信号レベルを制限し，ハフマン符号化を行う．

ことで圧縮率を高めています．圧縮データから画像を生成するには，圧縮と逆の手順で実行することになります．

設計フローの第1段階として，ここではJPEGエンコーダ（以下，JPEG）のSystemCモデルを作成します．といってもJPEGのプログラムを一から作るとたいへんです．SystemCを使う利点の一つは既存のコードを再利用しやすいことなので，この部分はサボることを考えました．つまり，すでにC言語で作成されたJPEGのプログラムをリファレンスとして利用します．検索サイトで「JPEG　ソース」などと入力して検索すると，Independent JPEG GroupのWebサイト（http://www.ijg.org/）にたどり着きました．その使用条件を見たところとくに問題がないと思われるので，ftp://ftp.uu.net/graphics/jpeg/jpegsrc.v6b.tar.gzのソース・コードを利用しました（利用などに際しての条件は，同梱のREADMEファイルの記述に従うものとする）．

● リファレンスCコードからSystemCモデルを作成

このようなリファレンスのC/C++プログラムがあれば，第1段階のSystemCモデルの作成は簡単になります．このリファレンスからSystemCモデルを作成するため，以下の作業を行う必要がありました．

リスト1
JPEGのSC_MODULEの定義
起動のイベントとなるcmdを定義し，リファレンスのmain関数を利用するためにコンストラクタでSC_THREADのプロセスmainを宣言した．

```
SC_MODULE(jpeg6)
{
  sc_fifo_in<int> cmd;

  void main();

  SC_CTOR(jpeg6) {
    SC_THREAD(main);
    }
};
```

1）SC_MODULE（jpeg6）の作成

まずは，SC_MODULEとしてjpeg6を作成します（**リスト1**）．この段階ではクロックやリセットを意識しないので，プロセスとしてSC_THREADを選択します．コンストラクタにSC_THREADとしてmainを宣言しました．JPEGの処理を起動する信号が必要になるため，入力信号cmdを作成しました．**リスト1**を見てわかるように，簡単にSC_MODULEを定義できます．

2）main関数の変更

さらにプロセスmainを完成させる必要があります．ここではリファレンスのmain関数を変更し，SystemCのプロセスにしました．これはリファレンスのC言語記述がSystemCの流儀に従っていないためで，以下の2種類の変更が必要でした．

- **int main（int argc，char **argv）の返り値，引き数の変更**
- **プロセスにおけるwhileループの作成**

リスト2に示すように，モジュールのmainプロセスにするため，返り値をvoidとし，引き数をローカル変数にしました．また，引き数をローカル変数にすると必要な情報が渡せなくなるので，グローバル変数を利用してデータを渡すことにしました．あまり美しいとは言えない方法なのですが，時間をかけずにSystemCにする一つの方法ではあると思います．

リファレンスでは1度処理すると終了するプログラムとなっていましたが，今回の用途では要求があったときにJPEGの処理を行うことになるので，mainの処理部分をwhile(1)で囲みます．

COLUMN　DCTとは

画像処理では，離散フーリエ変換（DFT：discrete Fourier transform）の一種である離散コサイン変換（DCT：discrete cosine transform）がよく利用されます．とくに8画素×8画素のブロックを対象としたDCTがよく用いられ，これは2次元の画像信号を周波数空間に変換することに相当します．また逆に，周波数空間から画像信号に戻す変換はIDCT（inverse DCT；逆離散コサイン変換）と呼ばれています．

8画素×8画素のDCTは以下のように定義されます．

$$DCT(i,j)=\frac{1}{4}C_iC_j\sum_{x=0}^{7}\sum_{y=0}^{7}pixel(x,y)\cos\frac{(2x+1)i\pi}{16}\cos\frac{(2y+1)j\pi}{16} \quad \cdots\cdots\cdots(a\text{-}1)$$

ここで，$i=0$のときは$C_i=1/\sqrt{2}$，そのほかのときは$C_i=1$になります．

DCTが画像処理によく使われる理由は，以下のとおりです．

- 高速な解法（chen，wangなど）があり，画像のようなサイズの大きいデータにも適用できる
- 整数演算で実現できるため，（浮動小数点演算を使う場合より）容易に利用できる
- 高域と低域を別々に圧縮することにより，視覚的に劣化の少ない画像圧縮を実現できる（自然画像では低域の成分が大きくなり，高域の成分が小さくなる傾向にあり，かつ人間の感覚は高域より低域の変化に敏感であるため）