第2章　無料のVivado HL WebPACK Editionで高位合成にチャレンジしよう(AXI4マスタ編)

ライン・バッファ/バースト転送/最適化指示子を駆使した高速化テクニック

小野 雅晃 Masaaki Ono

ここでは前章で作成したフィルタ回路を基本形に，ライン・バッファを追加したり，memcpy()関数を使ってバースト転送を発生させたり，ブロックRAMの割り当て方を修正してボトルネックを解消したり，PIPELINE指示子やARRAY_PARTITION指示子を入れて最適化したりすることで，ステップアップしながら高速化テクニックを解説します．最終的に当初のコードと比較して，約25倍の高速化を実現できました．

1. 高速化第1段階 ～ライン・バッファで高速化～

特設第1章で解説した第0段階のラプラシアン・フィルタでは，直接外部メモリから画像データを読み込んで，ラプラシアン・フィルタ処理を行い，外部メモリに書き込んでいました．しかしハードウェアではキャッシュがないため，この方法では性能が出ません．

● ライン・バッファを採用

そこで，あらかじめ1行分の画像を1ピクセルずつ

リスト1　第1段階のラプラシアン・フィルタCソースコード(laplacian_filter1.c)

```
// laplacian_filter1.c

～中略～

int lap_filter_axim(volatile int *cam_fb,
volatile int *lap_fb)
{
  #pragma HLS INTERFACE s_axilite port=return

#pragma HLS INTERFACE m_axi depth=3072
           port=cam_fb offset=slave bundle=cam_fb
#pragma HLS INTERFACE m_axi depth=3072 _
           port=lapfb offset=slave bundle=lap_fb

  int line_buf[3][HORIZONTAL_PIXEL_WIDTH];
  int x, y;
  int lap_fil_val;
  int a, b;
  int fl, sl, tl;

  // RGB値をY(輝度成分)のみに変換し、ラプラシアン・フィルタを掛けた
  Loop0: for (y=0; y<VERTICAL_PIXEL_WIDTH; y++){
    Loop1: for (x=0; x<HORIZONTAL_PIXEL_WIDTH; x++){
      if (y==0 || y==VERTICAL_PIXEL_WIDTH-1){
                          // 縦の境界の時の値は0とする
        lap_fil_val = 0;
      }else if (x==0 || x==HORIZONTAL_PIXEL_
                WIDTH-1){ // 横の境界の時も値は0とする
        lap_fil_val = 0;
      }else{
        if (y == 1 && x == 1){ // 最初のラインの最初のピクセルでは2ライン分の画素を読み出す
          Loop3: for (a=0; a<2; a++){ // 2ライン分
            Loop4: for (b=0; b<HORIZONTAL_
                        PIXEL_WIDTH; b++){ // ライン
              line_buf[a][b] =
            cam_fb[(a*HORIZONTAL_PIXEL_WIDTH)+b];
              line_buf[a][b] =
                      conv_rgb2y(line_buf[a][b]);
            }
          }
        }
        if (x == 1) {     // ラインの最初なので、2つのピクセルを読み込む
          Loop5: for (b=0; b<2; b++){ // ライン
            line_buf[(y+1)%3][b] = cam_
            fb[((y+1)*HORIZONTAL_PIXEL_WIDTH)+b];

            // (y+1)%3 は、使用済みのラインがに読み込む
            //  y=2 の時 line[0], y=3の時 line[1],
                                    y=4の時 line[2]
            line_buf[(y+1)%3][b] =
                conv_rgb2y(line_buf[(y+1)%3][b]);
          }
        }
        // 1つのピクセルを読み込みながらラプラシアン・フィルタを実行する
        line_buf[(y+1)%3][x+1] = cam_
        fb[((y+1)*HORIZONTAL_PIXEL_WIDTH)+(x+1)];

        // (y+1)%3 は、使用済みのラインがに読み込む
        //  y=2 の時 line[0], y=3の時 line[1],
                                    y=4の時 line[2]
        line_buf[(y+1)%3][x+1] = conv_rgb2y(
                          line_buf[(y+1)%3][x+1]);

        fl = (y-1)%3;     // 最初のライン, y=1 012,
                         y=2 120, y=3 201, y=4 012
        sl = y%3;          // 2番めのライン
        tl = (y+1)%3;     // 3番目のライン
        lap_fil_val = laplacian_fil(
              line_buf[fl][x-1], line_buf[fl][x],
             line_buf[fl][x+1],line_buf[sl][x-1],
              line_buf[sl][x], line_buf[sl][x+1],
               line_buf[tl][x-1],line_buf[tl][x],
                              line_buf[tl][x+1] );
      }
      // ラプラシアン・フィルタ・データの書き込み
      lap_fb[(y*HORIZONTAL_PIXEL_WIDTH)+x] =
               (lap_fil_val<<16)+(lap_fil_val<<8)
                                    +lap_fil_val ;
               // printf("x = %d  y = %d", x, y);
    }
  }
  return(0);
}

～中略～
```