


```

output                oCamSdaX,
//
// Camera System-Out
output                oCamXrstX,
output                oClkOutX,
//
// General Purpose-I/O
output[ 7:0]         oGpOutX,
input [ 7:0]         iGpInX,
//
//-----
// General CAM-PORT Destination Side for Internal Blockt
// Camera Standard-I/F
output[15:0]         oCamDtI,
output                oCamVsI,
output                oCamHsI,
output                oCamClkI,
//
// Camera I2C-I/F
output                oCamSclI,
input                 iCamSclI,
output                oCamSdaI,
input                 iCamSdaI,
//
// Camera System-Out
input                 iCamXrstI,
input                 iClkOutI,
//
// General Purpose-I/O
input [ 7:0]         iGpOutI,
output[ 7:0]         oGpInI,
//
//-----
// Block System Ports
// General Control/Status
input [31:0]         iBCTRL,

```

```

output[31:0]    oBSTTS,
//
// Common System Lines
input          iBRST, // Block System Reset(Positive Sync.)
input          iBCLK  // Block System Clock
//
);

//-----
wire           wPCLK = iCamClkX;           assign
            oCamClkI = wPCLK;
// 通常はカメラ・クロック入力をゲートすることなくスルーで使用。
// 将来的にゲートすることが必要になるかもしれないが、よい方法とは言えない。

wire          wPRST;           // wPCLK に同期したリセット信号

//-----
// 第一段階として、水平ライン毎に間引くことによって、垂直方向の削減をする。
// 単純に HSync をゲートすることによって間引く
//-----
wire[15:0]    wCamDtVH;
wire          wCamVsVH;
wire          wCamHsVH;
wire          wENA  = iBCTRL[0];
wire[ 3:0]    wRDCV = iBCTRL[11:8];
McamVertReducer
mCamVertReducer(
    // General CAM-PORT Source Side
    .iCamDtS(iCamDtX),    .iCamVsS(iCamVsX),
    .iCamHsS(iCamHsX),
    // General CAM-PORT Destination Side
    .oCamDtD(wCamDtVH), .oCamVsD(wCamVsVH),
    .oCamHsD(wCamHsVH),
    // Control/Status Port
    .iENA(wENA),    .iRDCV(wRDCV),
    // Common System Lines

```

```

        .iRST(wPRST), .iCLK(wPCLK)
    );

//-----
// 第二段階として、水平ライン中のピクセルを間引く。
// 最終的な出力として、有効な HSync アサート中、連続な有効画素データを出力する必要
// あるので、
// ラインバッファを持って、一ラインつめてから、出力するようにする。
//-----
wire[ 3:0]    wRDCH = iBCTRL[15:12];
wire          wBWnY = iBCTRL[4];
McamHorzReducer
    mCamHorzReducer(
        // General CAM-PORT Source Side
        .iCamDtS(wCamDtVH), .iCamVsS(wCamVsVH),
        .iCamHsS(wCamHsVH),
        // General CAM-PORT Destination Side
        .oCamDtD(oCamDtI),   .oCamVsD(oCamVsI),
        .oCamHsD(oCamHsI),
        // Control/Status Port
        .iENA(wENA), .iRDCH(wRDCH),      .iBWnY(wBWnY),
        // Common System Lines
        .iRST(wPRST), .iCLK(wPCLK)
    );

//-----
// 未使用の出力ポート処理
//-----
assignoBSTTS      = iBCTRL;
assignoCamScI_X   = iCamScI;
assignoCamScI     = iCamScI_X;
assignoCamSda_X   = iCamSdaI;
assignoCamSdaI    = iCamSda_X;
assignoCamXrst_X  = iCamXrstI;
assignoClkOut_X   = iClkOutI;
assignoGpOut_X    = iGpOutI;

```

```

    assign oGpInI          = iGpInX;

//-----
endmodule      // MSviPreprocessBlock
//-----
//-----
// 垂直水平方向に関する間引きモジュール
//-----
module MCamVertReducer
(
//-----
// General CAM-PORT Source Side
//
input [15:0]      iCamDtS,
input            iCamVsS,
input            iCamHsS,
//
//-----
// General CAM-PORT Destination Side
//
output[15:0]      oCamDtD,
output            oCamVsD,
output            oCamHsD,
//
//-----
// Control/Status Port
//
input            iENA,          // 前処理機能のイネーブル
input [ 3:0]      iRDCV,        // Reduce-V
//
//-----
// Common System Lines
// カメラのデータ・クロックと同じクロック。
input            iRST,  // System Reset(Positive Sync.)
input            iCLK   // System Clock
);

```

```

//
//-----
// 水平ライン毎に間引くことによって、垂直方向の削減をする。
// 単純に HSync をゲートすることによって間引く
//-----
reg          rCamHsi;      // Intermediate-HS
reg          rCamVsi;      // Intermediate-VS
reg [15:0]   rCamDti;      // Intermediate-D
reg [ 3:0]   rHsCnt;       // HSync の立下りでデクリメントするダウン・カウン
タ
reg [ 2:1]   rCamHss;      // HSync のエッジ検出 9 レジスタ
wire        wCamHssTrg    = (rCamHss == 2'b10);
wire        wHsGate       = (rHsCnt  == 4'd0);
wire        wHsCntClr     = |{~iCamVsX,iRST};
wire        wHsCntSet     = &{wHsGate,wCamHssTrg};

always @( iCLK )
begin
  rCamVsi      <= iCamVsS;
  rCamDti      <= iCamDtS;
  if ( wPRST )          rCamHsi      <= 0;
  else                  rCamHsi      <= iCamHsS & (wHsGate
| ~iENA);

  if ( wHsCntClr )      rHsCnt <= 3'd0;
  else if ( wHsCntSet ) rHsCnt <= iRDCV;          // 水平ラインの数をカウン
トしながら、
  else if ( wCamHssTrg ) rHsCnt <= rHsCnt -1;    // 垂直方向を間引く
  else                  rHsCnt <= rHsCnt;
end

assignoCamDtD          = rCamDti;
assignoCamVsD          = rCamVsi;
assignoCamHsD          = rCamHsi;

//-----

```

```

endmodule // MCamVertReducer
//-----
//-----
// 水平方向に関する間引きモジュール
//-----
module MCamHorzReducer
(
//-----
// General CAM-PORT Source Side
//
input [15:0]    iCamDtS,
input          iCamVsS,
input          iCamHsS,
//
//-----
// General CAM-PORT Destination Side
//
output[15:0]    oCamDtD,
output          oCamVsD,
output          oCamHsD,
//
//-----
// Control/Status Port
//
input          iENA,          // 前処理機能のイネーブル
input [ 3:0]    iRDCH,        // Reduce-H
input          iBWnY,        // Bus-width 1:Word/0:Byte
//
//-----
// Common System Lines
// カメラのデータ・クロックと同じクロック。
input          iRST, // System Reset(Positive Sync.)
input          iCLK  // System Clock
);
//
//-----

```

// 水平ライン中のピクセルを間引く。最終的な出力として、有効な HSync アサート中、
// 連続な有効画素データを出力する必要があるので、
// ラインバッファを持って、一ラインつめてから、出力するようにする。

```
//-----
reg [15:0]    rCamDtd;    assign  oCamDtD    = rCamDtd;
reg          rCamVsd;    assign  oCamVsD    = rCamVsd;
reg          rCamHsd;    assign  oCamHsD    = rCamHsd;
reg [15:0]    rCamDts[1:2];
reg [ 2:1]    rCamVss;
reg [ 2:1]    rCamHss;

// Line Buffer, Width: 16+2(HS/VS), Depth: 4096
wire[17:0]    wFds    = {rCamVss[2],rCamHss[2],rCamDts[2][15:0]};
wire          wFes;
wire[17:0]    wFdd;
wire          wFed;
MFifoCamLineBuffer
  mFifoCamLineBuffer(
    .clk(iCLK),          .srst(iRST),
    .din(wFds),          .wr_en(wFes),  .full(),
    .dout(wFdd),        .rd_en(wFed),  .empty()
  );

//-----
reg          rLnAck;    // Line ACK, バッファに 1 ライン格納済み
wire         wCamHsd    = wFdd[16];
wire         wCamVsd    = wFdd[17];
assign       wFed        = rLnAck | ~wCamHsd;
wire         wLnAckClr   = (~wCamHsd & rCamHsd) | iRST;
wire         wLnAckSet   = ~rCamHss[1] & rCamHss[2];
always @( iCLK )
begin
  rCamDd          <= wFdd[15:0];
  rCamVsd         <= wCamVsd;
  rCamHsd         <= wCamHsd & rLnAck;
  rCamDs[1][15:0] <= iCamDtS;
end
```



```

rCamDs[2][15:0]    <= rCamDs[1][15:0];
rCamVss[2:1]      <= {rCamVss[1],iCamVsS};
rCamHss[2:1]      <= {rCamHss[1],iCamHsS};

if ( wLnAckClr )          rLnAck <= 0;
else if ( wLnAckSet ) rLnAck <= 1;
else                      rLnAck <= rLnAck;
end

//-----
reg [ 3:0]    rPiCnt;          // Pixel 単位でデクリメントするダウン・カウンタ
reg [ 1:0]    rDiCnt;          // Data 単位でデクリメントするダウン・カウンタ
assign        wFes            = (rPiCnt == 4'd0);
wire          wPiCntClr       = rCamHss[1] & ~rCamHss[2];
wire          wPiCntSet       = (rPiCnt == 4'd0) & (rDiCnt == 2'd0);
wire          wPiCntDec       = (rDiCnt == 2'd0);
wire          wDiCntSet       = wPiCntDec | wPiCntClr;
always @( iCLK )
begin
  if ( wPiCntClr )          rPiCnt <= 4'd0;
  else if ( wPiCntSet ) rPiCnt <= iRDCH;
  else if ( wPiCntDec ) rPiCnt <= rPiCnt -1;
  else                      rPiCnt <= rPiCnt;

  if ( wDiCntSet )          rDiCnt <= iBWnY ? 2'd3 : 2'd1;
  else if ( wDiCntDec ) rDiCnt <= rDiCnt -1;
end

//-----
endmodule // MCamHorzReducer
//-----

```