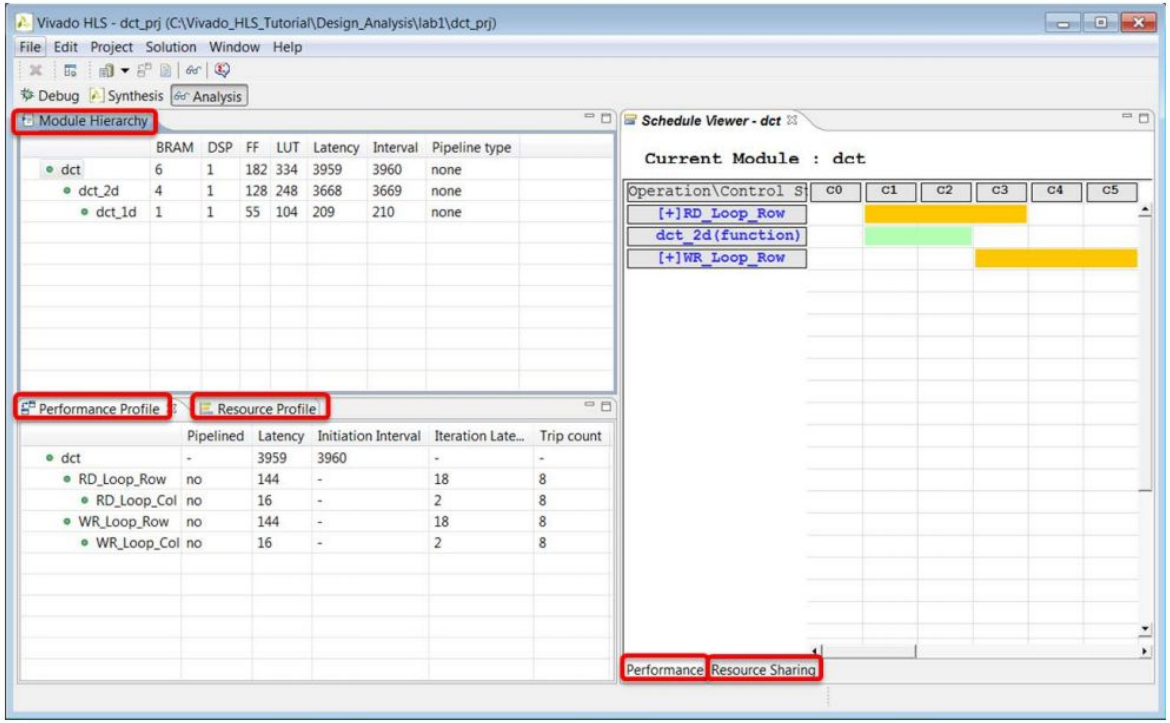
Synthesis of Digital Systems - Lab

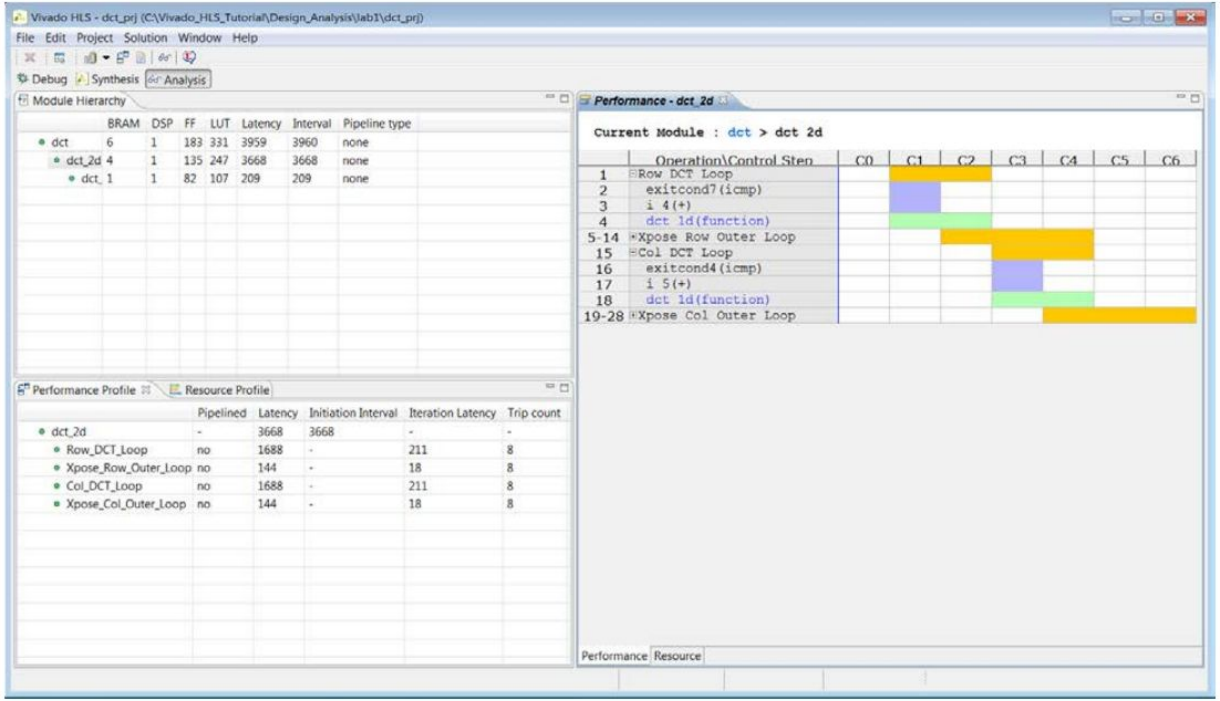
Module D: Video-Processing on ZedBoard

D0. Recap - HLS of DCT Algorithm

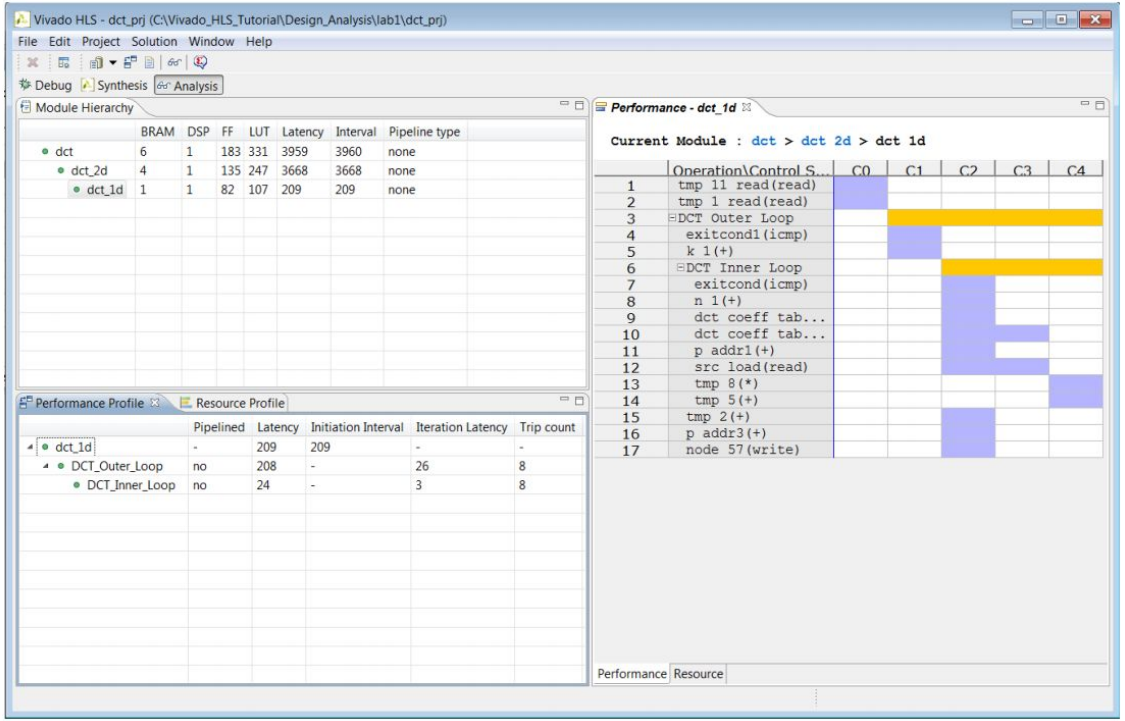
HLS DCT - Solution 1 (1)



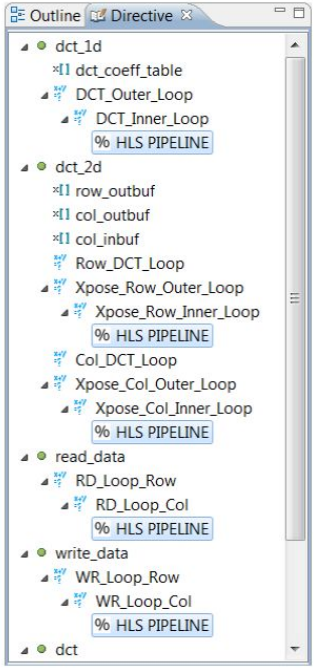
HLS DCT - Solution 1 (2)



HLS DCT - Solution 1 (3)

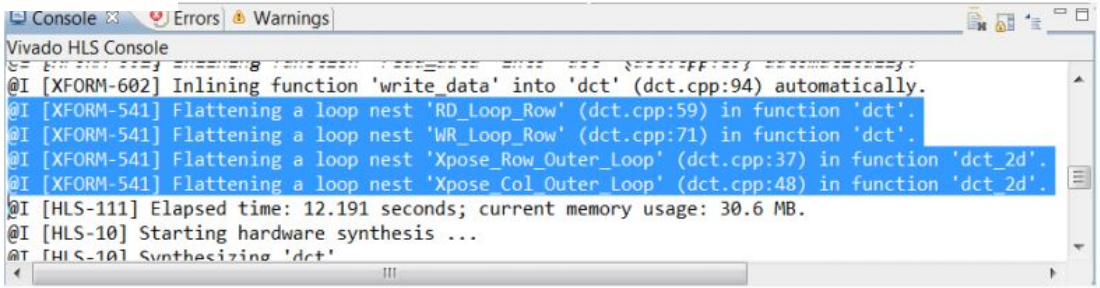


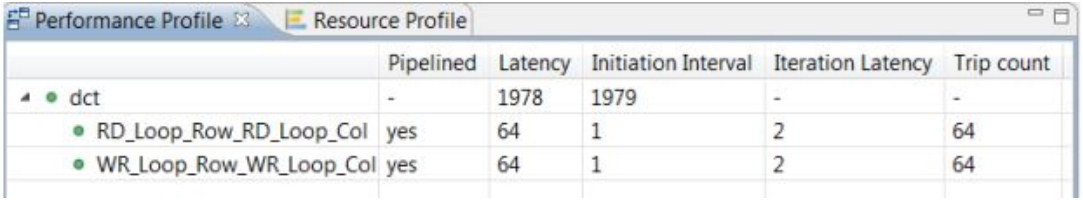
HLS DCT - Solution 2 (1)



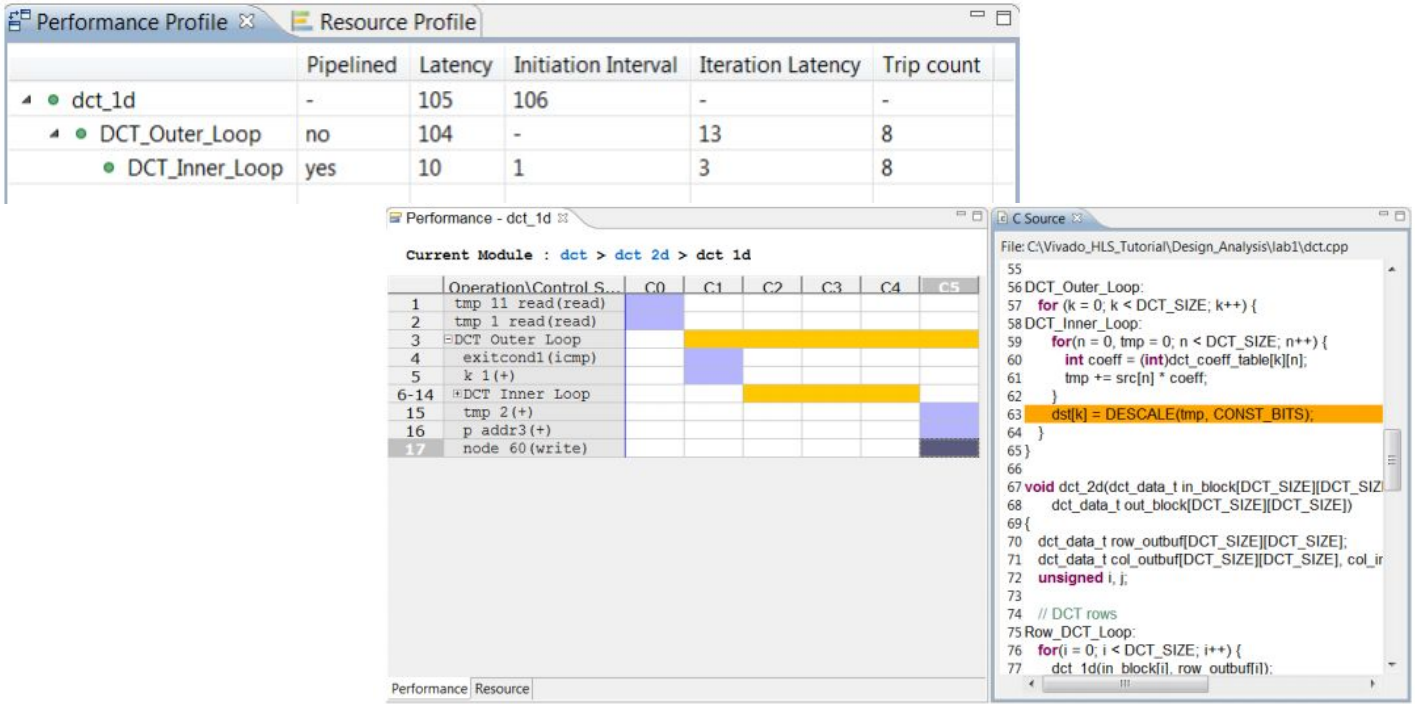
Latency (clock cycles)

		solution1	solution2
Latency	min	3959	1978
	max	3959	1978
Interval	min	3960	1979
	max	3960	1979

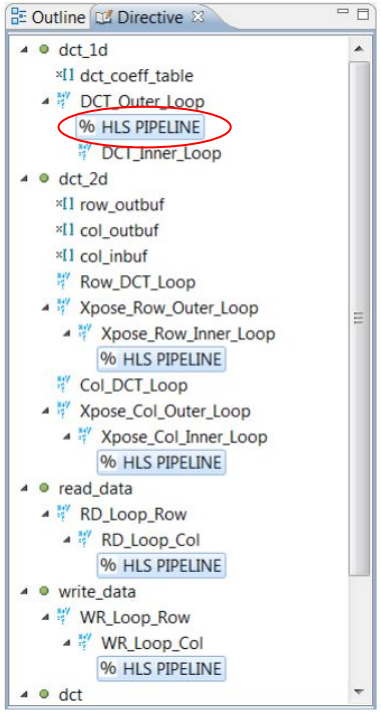




HLS DCT - Solution 2 (2)



HLS DCT - Solution 3 (1)



Latency (clock cycles)

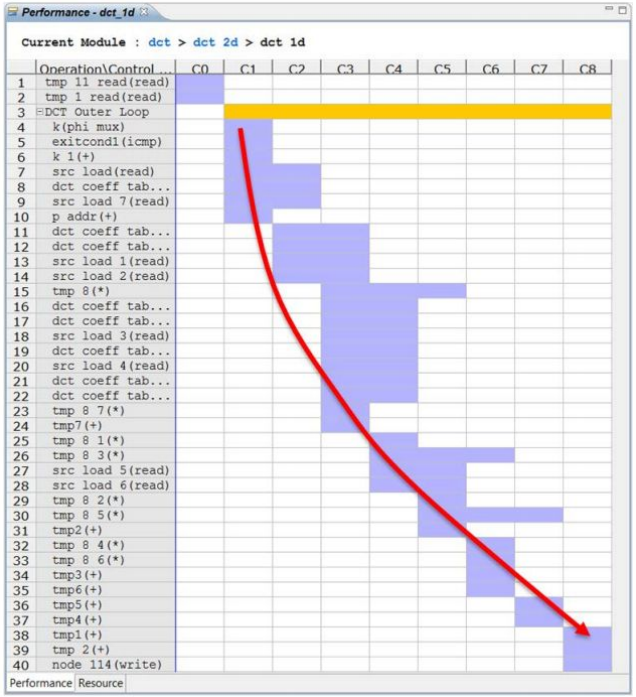
		solution2	solution3
Latency	min	1978	890
	max	1978	890
Interval	min	1979	891
	max	1979	891

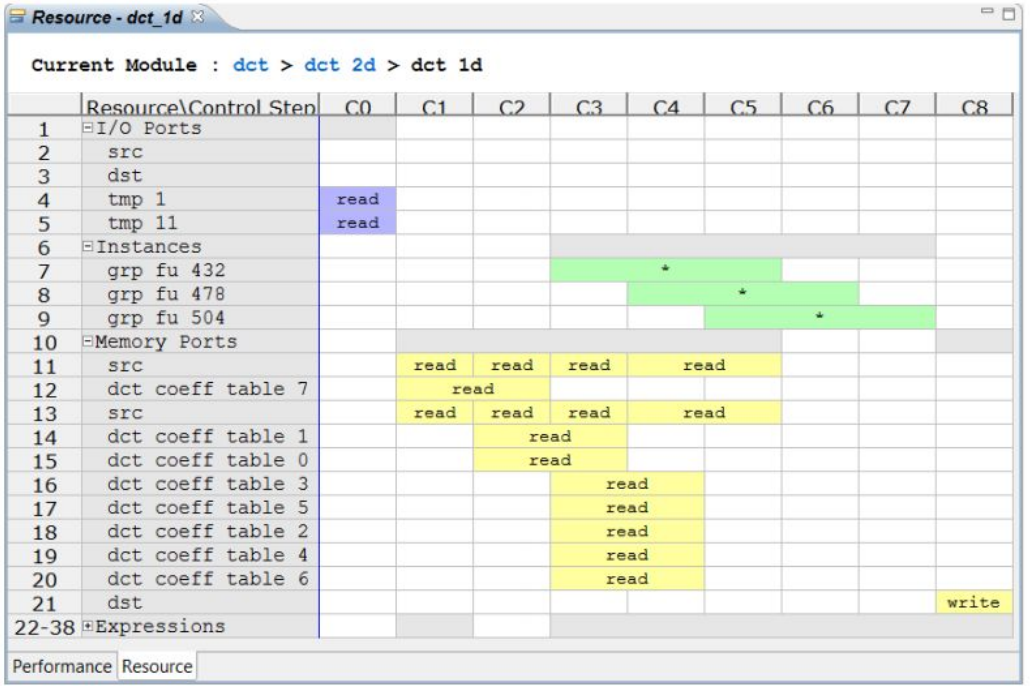
Utilization Estimates

	solution2	solution3
BRAM_18K	6	13
DSP48E	1	8
FF	241	543
LUT	451	469

Export the report(.html) using the Export Wizard

HLS DCT - Solution 3 (2)





HLS DCT - Solution 4

Outline | Directive

- dct_1d
  - dct_coeff_table
  - DCT_Outer_Loop
    - % HLS PIPELINE
    - DCT_Inner_Loop
- dct_2d
  - row_outbuf
  - col_outbuf
  - col_inbuf
  - % HLS ARRAY_PARTITION variable=col_inbuf complete dim=2
  - Row_DCT_Loop
  - Xpose_Row_Outer_Loop
    - Xpose_Row_Inner_Loop
      - % HLS PIPELINE
  - Col_DCT_Loop
  - Xpose_Col_Outer_Loop
    - Xpose_Col_Inner_Loop
      - % HLS PIPELINE
- read_data
  - RD_Loop_Row
    - RD_Loop_Col
      - % HLS PIPELINE
- write_data
  - WR_Loop_Row
    - WR_Loop_Col
      - % HLS PIPELINE
- dct
  - buf_2d_in
  - % HLS ARRAY_PARTITION variable=buf_2d_in complete dim=2
  - buf_2d_out
  - input
  - output

compare reports

Performance Estimates

Timing (ns)

Clock		solution3	solution4
default	Target	8.00	8.00
	Estimated	5.90	5.83

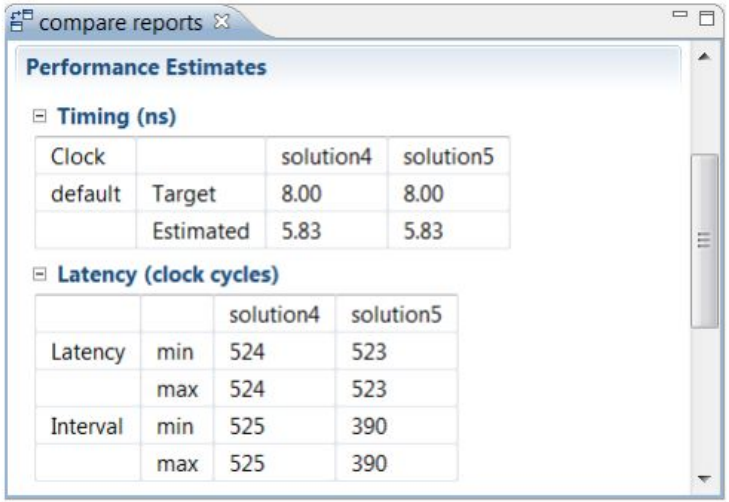
Latency (clock cycles)

		solution3	solution4
Latency	min	890	524
	max	890	524
Interval	min	891	525
	max	891	525

Module Hierarchy

	BRAM	DSP	FF	LUT	Latency	Interval	Pipeline type
dct	11	8	1037	914	524	525	none
dct_2d	10	8	721	584	389	389	none
read_data	0	0	27	58	66	66	none

HLS DCT - Solution 5 (1)



HLS DCT - Solution 5 (2)

Module Hierarchy

	BRAM	DSP	FF	LUT	Latency	Interval	Pipeline type
dct	12	8	1304	1064	523	390	dataflow
read_data	0	0	28	59	66	66	none
dct_2d	10	8	722	585	389	389	none
dct_1d	8	8	394	86	14	14	none
write_data	0	0	31	67	66	66	none

Performance Profile | Resource Profile

	Pipelined	Latency	Initiation Interval	Iteration Latency	Trip count
dct_2d	-	389	389	-	-
Row_DCT_Loop	no	128	-	16	8
Xpose_Row_Outer_Loop_Xpose_Row_Inner_Loop	yes	64	1	2	64
Col_DCT_Loop	no	128	-	16	8
Xpose_Col_Outer_Loop_Xpose_Col_Inner_Loop	yes	64	1	2	64

HLS DCT - Solution 6 (1)

Outline | Directive

- dct_1d
  - dct_coeff_table
  - DCT_Outer_Loop
    - % HLS PIPELINE
    - DCT_Inner_Loop
- dct_2d
  - % HLS INLINE
  - row_outbuf
  - col_outbuf
  - col_inbuf
  - % HLS ARRAY_PARTITION partition variable=col_inbuf complete dim=2
  - Row_DCT_Loop
  - Xpose_Row_Outer_Loop
    - Xpose_Row_Inner_Loop
      - % HLS PIPELINE
  - Col_DCT_Loop
  - Xpose_Col_Outer_Loop
    - Xpose_Col_Inner_Loop
      - % HLS PIPELINE
- read_data
  - RD_Loop_Row
    - RD_Loop_Col
      - % HLS PIPELINE
- write_data
  - WR_Loop_Row
    - WR_Loop_Col
      - % HLS PIPELINE
- dct
  - % HLS DATAFLOW
  - buf_2d_in
  - % HLS ARRAY_PARTITION partition variable=buf_2d_in complete dim=2
  - buf_2d_out
  - input
  - output

compare reports

Performance Estimates

Timing (ns)

Clock		solution5	solution6
default	Target	8.00	8.00
	Estimated	5.83	5.80

Latency (clock cycles)

		solution5	solution6
Latency	min	523	409
	max	523	409
Interval	min	390	71
	max	390	71

Module Hierarchy

	BRAM	DSP	FF	LUT	Latency	Interval	Pipeline type
dct	22	16	2096	1218	409	71	dataflow
read_data	0	0	28	59	66	66	none
dct_Loop_Row_DCT_Loop_proc	8	8	467	128	70	70	none
dct_Loop_Xpose_Row_Outer_Loop_proc	0	0	28	61	66	66	none
dct_Loop_Col_DCT_Loop_proc	8	8	467	128	70	70	none
dct_Loop_Xpose_Col_Outer_Loop_proc	0	0	29	69	66	66	none
write_data	0	0	31	67	66	66	none

HLS DCT

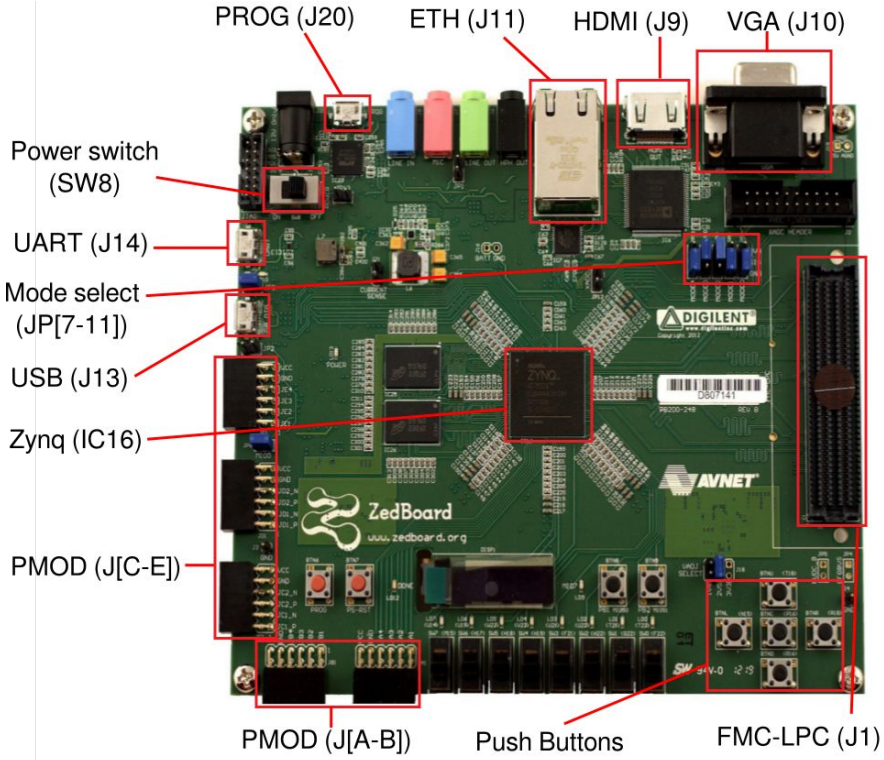
Solution	Directive	Latency	Init. Interval	BRAM	DSP	FF	LUT
Sol1: Base Design	-	3959	3959	5	1	278	982
Sol2: pipelining inner-loops	PIPELINE (inner-loops)	1851	1851	5	1	256	1286
Sol3: pipelining outer-loop to remove loop hierarchy	PIPELINE (outer-loop)	875	875	5	8	678	1484
Sol4: Partition the 2D array into 1D arrays	ARRAY-PARTITION	509	509	3	8	1294	2014
Sol5: Top-level parallelism	DATAFLOW	508	375	3	8	1300	1921
Sol6: Flatten inner function to top-level and make use of top-level parallelism	INLINE	495	114	3	16	1950	1932

D1. Video-Processing System Design

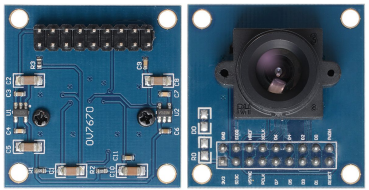
Video Processing

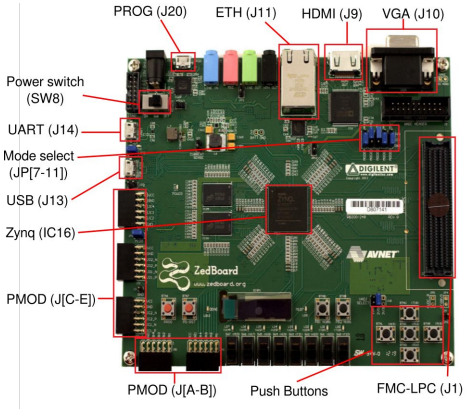
- Essentially a data-processing task-pipeline
  - Information captured from vision sensor
    - Pixel data
  - Signal processing algorithms
    - Filtering, FFT etc
  - Outputs
    - Enhanced visualization of captured scene
    - Extract useful representation for other sub-systems
- Generally data and compute intensive
  - High optimization potential for HLS
    - regular data-flow oriented nature
    - pixel-level parallelism etc.

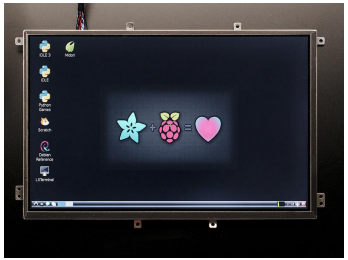
ZedBoard Capabilities



Video Processing on ZedBoard

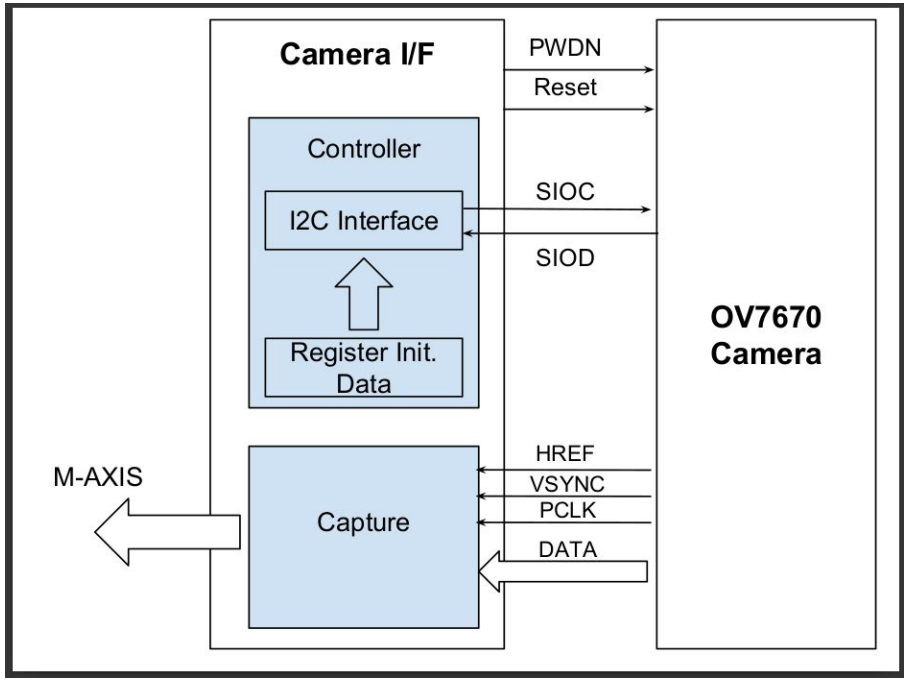




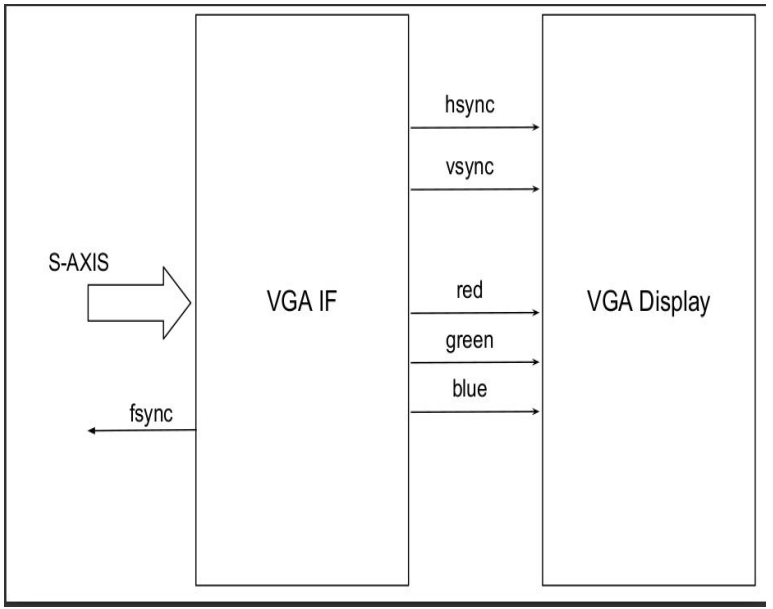


Interfaces

Camera i/f



VGA i/f

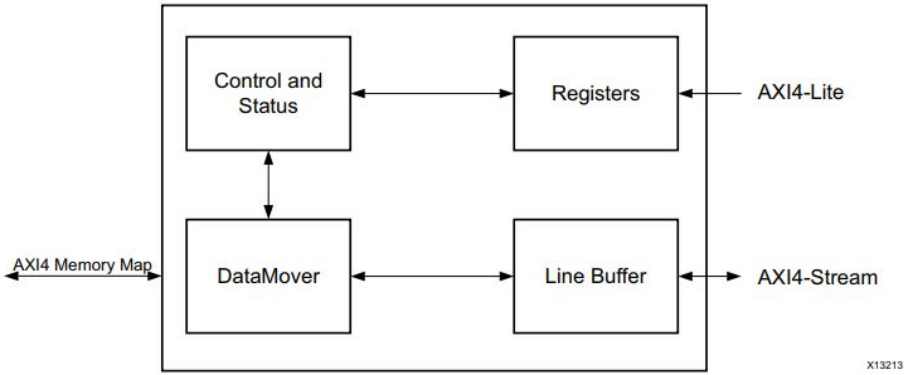


AXI Interfaces in Zynq

- Part of ARM AMBA family of buses
- Zynq uses:
  - **AXI4:** for high performance memory-mapped
  - **AXI4-Lite:** for simple, low-throughput memory-mapped
    - e.g. programming peripheral registers
  - **AXI4-Stream:** for high-speed streaming data b/w IPs
    - No addr info before starting transaction

VDMA IP

- High BW direct memory access
  - Offloads CPU of memory data flow between sub-systems
- b/w memory and AXI4-Stream target peripherals
- Especially optimized for Video data (with its various formats)



Grayscale Filtering

- Single pixel-operation
  - Repeated for each pixel with no ordering constraint
- Luminosity method:

$$Y = 0.299R + 0.587G + 0.114B$$

- Applications:
  - Fast extraction of useful info
  - Resource-constrained Video-processing
  - Pre-processing for other processing algorithms

Application Profiling

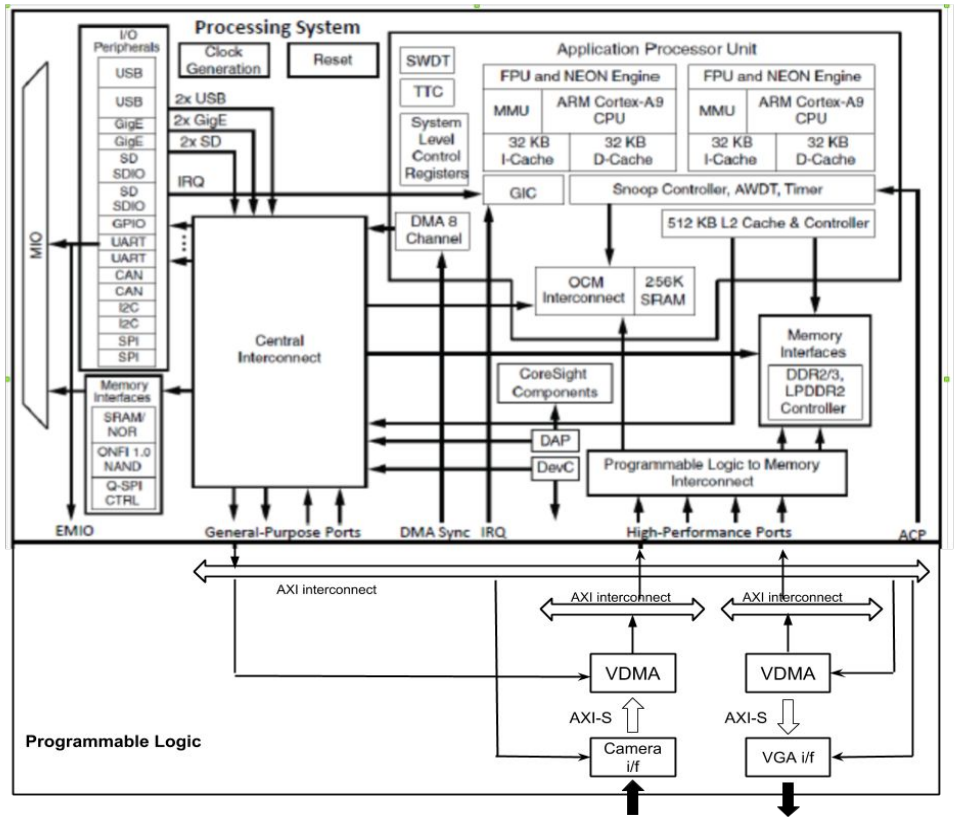
- BSP functions
  - *EnablePerfCounters()*
  - *init_perfcounters()*
  - *get_cyclecount()*

- Profile the Grayscale routine
  - Processor Cycles
  - Conversion to time

System Design

SW Algorithm:

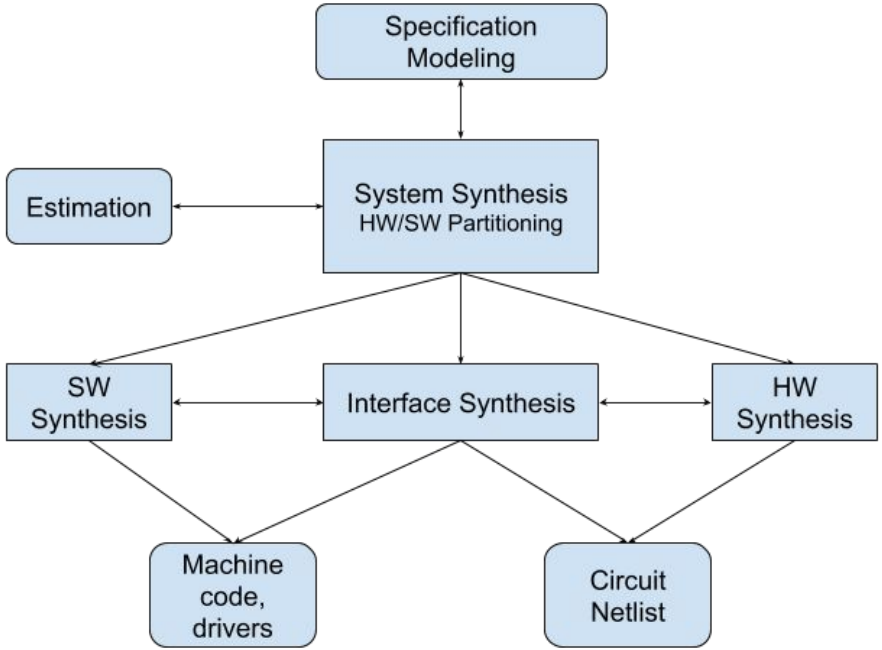
- Initialize the VDMAs and program them to transfer data to/from Camera/VGA FIFOs to system-memory
- Capture the raw-video data and store it as RGB888 (32b pixel-words)
- Convert to Grayscale and store it in some other section of system-memory
- Display the processed video stream



D2. HW Acceleration

HW Acceleration within HW/SW Codesign

- Dedicated HW Accelerators to offload compute-intensive tasks from CPU (SW)
- Can gain huge performance gains using HW-accel. for data-oriented SW routines
- Grayscale Filtering is one such algorithm
  - Maps well to HW structures because of its inherent parallelism



OpenCV - Computer Vision Library

- Open-source SW library for real-time computer-vision applications
  - Pretty stable and mature (development since 1999)
  - Widely leveraged within Academia and Industry
- Supported Platforms:
  - C++ with interfaces to Java, Python, MATLAB etc.
  - Linux, WinOS, Android, OS-X etc.
- Xilinx HLS Video library
  - Comes packaged with Vivado-HLS tool
  - Provides HL-Synthesized IPs of useful OpenCV functions
  - OpenCV compatible API + OpenCV interface functions

HW Accelerated Grayscale Filtering (1)

```
44 void gray_scale(AXI_STREAM& input, AXI_STREAM& output, int rows, int cols) {
45     //Create AXI streaming interfaces for the core
46 #pragma HLS RESOURCE variable=input core=AXIS metadata="-bus_bundle INPUT_STREAM"
47 #pragma HLS RESOURCE variable=output core=AXIS metadata="-bus_bundle OUTPUT_STREAM"
48
49 #pragma HLS RESOURCE core=AXI_SLAVE variable=rows metadata="-bus_bundle CONTROL_BUS"
50 #pragma HLS RESOURCE core=AXI_SLAVE variable=cols metadata="-bus_bundle CONTROL_BUS"
51 #pragma HLS RESOURCE core=AXI_SLAVE variable=return metadata="-bus_bundle CONTROL_BUS"
52
53 #pragma HLS INTERFACE ap_stable port=rows
54 #pragma HLS INTERFACE ap_stable port=cols
55
56     RGB_IMAGE img_0(rows, cols);
57     RGB_IMAGE img_1(rows, cols);
58 #pragma HLS dataflow
59     hls::AXIvideo2Mat(input, img_0);
60     hls::CvtColor<HLS_RGB2GRAY>(img_0,img_1);
61     hls::Mat2AXIvideo(img_1, output);
62 }
63
```

HW Accelerated Grayscale Filtering (2)

```
1933 template<typename CONVERSION,int SRC_T, int DST_T,
1934     int ROWS,int COLS>
1935 void CvtColor(
1936         Mat<ROWS, COLS, SRC_T>  &_src,
1937         Mat<ROWS, COLS, DST_T>  &_dst
1938         )
1939 {
1940     kernel_CvtColor<CONVERSION,HLS_TNAME(SRC_T),HLS_TNAME(DST_T)> kernel_opr;
1941     int     cols=_src.cols;
1942     int     rows=_src.rows;
1943     assert(rows <= ROWS);
1944     assert(cols <= COLS);
1945     Scalar<HLS_MAT_CN(SRC_T),HLS_TNAME(SRC_T)> _s;
1946     Scalar<HLS_MAT_CN(DST_T),HLS_TNAME(DST_T)> _d;
1947  loop_height: for(int i= 0; i < rows; i++) {
1948     loop_width: for (int j= 0; j < cols; j++) {
1949 #pragma HLS LOOP_FLATTEN OFF
1950 #pragma HLS PIPELINE
1951             _src >> _s;
1952             kernel_opr.apply(_s,_d);
1953             _dst << _d;
1954         }
1955     }
1956 }
```

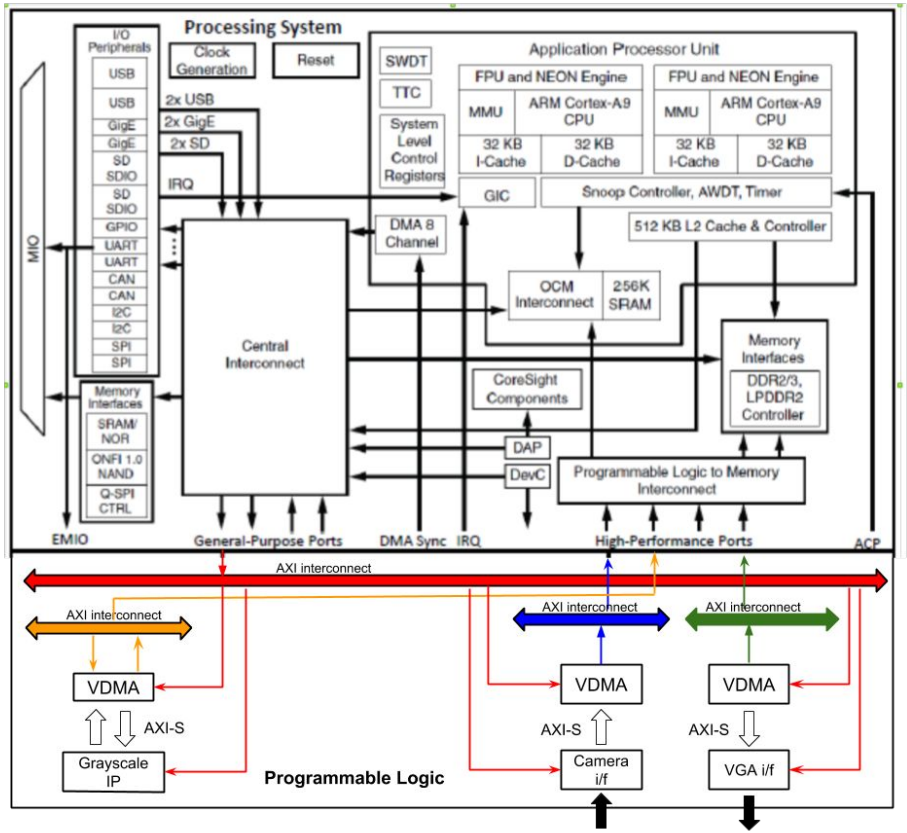
HW Accelerated Grayscale Filtering (3)

```
1518 template<typename SRC_T, typename DST_T>
1519 class kernel_CvtColor<HLS_RGB2GRAY, SRC_T, DST_T> {
1520     Scalar<3, typename CvtColor_traits<SRC_T>::COEFF_T > par;
1521 public:
1522     kernel_CvtColor() : par(0.299,0.587,0.114) {};
1523     template<int CN1,int CN2>
1524     void apply(Scalar<CN1,SRC_T>& _src, Scalar<CN2,DST_T>& _dst) {
1525         ColorConverter<SRC_T,DST_T>::convert(_src, _dst.val[0], par);
1526     }
1527 };
```

```
1466 template <typename SRC_T, typename DST_T>
1467 class ColorConverter {
1468 public:
1469     template <int CN1>
1470     static void convert(Scalar<CN1,SRC_T>& _src, DST_T& result, Scalar<3, typename CvtColor_traits<SRC_T>::COEFF_T > par) {
1471 #pragma HLS inline
1472         typename CvtColor_traits<SRC_T>::MULT_T b,g,r;
1473         r=par.val[0]*_src.val[0];
1474         b=par.val[1]*_src.val[1];
1475         g=par.val[2]*_src.val[2];
1476         typename CvtColor_traits<SRC_T>::ACCUM_T c;
1477         c=r+g+b;
1478         result=sr_cast<DST_T> (c);
1479     }
1480 };
```

Zynq Design

- Another AXI-based subsystem for Grayscale-IP
- The corresponding VDMA is enabled in both S2MM and MM2S
- Grayscale IP:
  - Read in the captured-data via VDMA-MM2S
  - Process the data
  - Write out the processed data via VDMA-S2MM



SW Application

- Initialize and setup Grayscale IP and its VDMA
- Profile Grayscale IP operation
  - How much speedup over pure-SW design?

- [BONUS] Interactive Application:
  - ‘r’ key from user -> display raw-video
  - ‘p’ key from user -> display SW processed-video
  - ‘g’ key from user -> display HW processed-video

Questions?

DEADLINE: See Moodle!