Appendix

Common HLS Directives in Vivado HLS:

1) Array partitioning:
   Partition an array into smaller elements (arrays or registers) to remove RAM bottlenecks
   Useful for applications which are:
   Having large RAM read write ports on a single big memory.
   Very Large Arrays like 2D/3D arrays like stored images etc.

2) Loop Unroll:
   The UNROLL pragma allows the loop to be fully or partially unrolled. Fully unrolling the loop creates a copy of the loop body in the RTL for each loop iteration so the entire loop can be run concurrently. Partially unrolling a loop lets you specify a factor N, to create N copies of the loop body and reduce the loop iterations accordingly. To unroll a loop completely, the loop bounds must be known at compile time. This is not required for partial unrolling.
   Example :

```
for (int i = 0; i < 6; i++) {
pragma HLS unroll factor=2 a[i] = b[i] + c[i];
 }
```

3) Dependence:

   Used to tell the HLS tool to take intelligent decision for pipelining. This is dependent on how a variable is written and read in a loop. It should be used with caution as any false dependency may cause functional issues!

4) Pipelining:

   Used to insert retiming flip-flops to schedule a particular section of code in a single clock cycle. This allows concurrent execution of operations.

```
for (int i = 0; i < 6; i++) {
pragma HLS PIPELINE II =1
a[i] =x +y;
b[i] = 7* a[i];
result = result + b[i];
 }
```

5) Loop merge:
   Merge consecutive loops into a single loop to reduce overall latency, increase sharing, and improve logic optimization.

Example:
Loop1:

```
for (i = 1; i <= 100 ; i++ )
 X = 7;
B = B+1;
C = 7;
```
Merge with LOOP 2:
```
for (j = 1; j <= 100 ; j++ )
T = 7;
Z = Z+1;

for (i = 1; i <= 100 ; i++ )
 X = 7;
B = B+1;
C = 7;
T=7;
Z=Z+1;
```

6) Streaming:

   Default array variables are implemented as RAM.

   If data is consumed sequentially, use FIFOs instead of RAM. It should not be used for streaming applications where input data length is unknown.

   #pragma HLS stream variable=<variable> depth=<int> dim=<int> off

7) Protocol:

   The PROTOCOL pragma specifies a region of the code to be a protocol region, in which no clock operations are inserted by Vivado HLS unless explicitly specified in the code

   Example:

   #pragma HLS protocol makes sure that there is no overlap of statements inside or outside the protocol region

8) Loop flattening:

   Allows nested loops to be flattened into a single loop hierarchy with improved latency.

   Example:

   #pragma HLS loop_flatten off

9) Array map:

   Combines multiple smaller arrays into a single large array to help reduce block RAM resources.

   Example:

   #pragma HLS array_map variable= instance= \ offset=

10) Data pack:

    Packs the data fields of a structure into a single scalar with a wider word width.

Example:

#pragma HLS data_pack variable= \ instance=

For other Vivado HLS directives, one can refer to Vivado HLS optimization guide [3.11]

Common HLS Directives in MATLAB HDL Coder :

1) Distributed pipelining: Distribute retiming flops at different places in the design in order to reduce critical paths.

2) RAM mapping: Mapping large delays, persistent variables in MATLAB code, and pipeline delays to RAM based on a threshold bit width.

3) Clock rate pipelining: Runs pipeline registers at a faster clock rate when you specify an oversampling factor greater than one, thereby improving retiming.

4) Resource Sharing: Identifies multiple functionally equivalent resources and replaces them with a single resource. Saves area with performance trade-offs.

5) Loop unrolling: Unrolls a loop by instantiating multiple instances of the loop body in the generated code. Also supports partial unrolling.

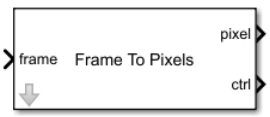
6) Streaming: Splits a vector data path into multiple smaller vector data paths based on the StreamingFactor that one specifies thereby reducing hardware resource consumption

For further MATLAB HDL coder directives, one can refer to HDL coder guide [4.11]

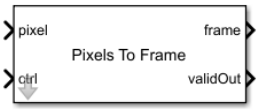
Simulink library blocks used as part of designs:

Vision HDL toolbox:

1) Conversion of frame to pixels for inputting to FPGA

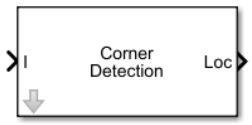


2) Conversion of pixels to frame to sample output from FPGA



3) Standard library block for Sobel filtering in images

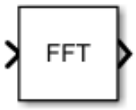
4) Standard library block for corner detection in images (Harris corner detector)



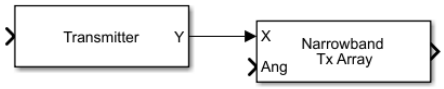
Signal Processing / DSP System toolbox/Phased array system toolbox:

1) FMCW waveform generator (Generates spectrum corresponding to a predefined target)

2) FFT computation for 2048 points:



3) RADAR transmitter with narrow band array



4) RADAR receiver with narrow band antenna array

5) Sources for waveform generation : RAMP/ pulse/ sinusoids and sweep

6) Sinks and scopes to plot time and frequency domain plots for different signals.

Psuedo-code for “Application specific bit width for intermediate data nodes” algorithm:

//Simulation Run:

```
Csim_design (“probe -create all -depth all” , “$file”);
// Run the simulation with probe of all nets, saving values in $file (*.vcd)
my $vcd = parse_vcd ($file);
// Parsing the name of Value change dump, VCD file and storing into a variable.
my @signals = list_sigs ($vcd);
```

// All signal names and values returned from VCD file and stored into a variable
int val_nodes = [clock_cycle -1 : 0] [length (signals)]
// length(signals) Returns the number of nodes in design, stored in a variable
// val_nodes creates an empty array with length of number of clock cycles during the simulation runtime
int nodes[] = { signals };
// Initializing the nodes array with all signals read from VCD file
foreach (int *v, nodes) {
minv = minimum (val_nodes[v])
//Minimum value of a node as a floating point number
maxv = maximum (val_nodes[v])
//Maximum value of a node as a floating point number
diffv = maxv – minv
// difference of maximum and minimum in floating point
diffvfix = fixdt (maxv – minv)
// Difference of maximum and minimum values in simulation run which is a floating point
// number is type casted to a fixed point data type
fixedv[1] = type_cast (integer(diffv))
// Type cast returns with bit width required to store the floating point value of //exponent (integer)
fixedv[0] = type_cast (frac(diffv))
// Type cast returns with bit width required to store the floating point value of //mantissa(fraction)
optdimv[v] = { fixedv[1], fixedv[0]}
// Integer and fractional parts are concatenated to a single array
}

//Synthesis Run:

Csynth_design -constraints constraint_file.tcl

// constraint_file.tcl is a high level synthesis constraint file which contains:
foreach (int *y, nodes) {

width_y = fixdt (optdimv[y])

// Constraining the synthesis tool to treat fixed point number with specific bits for integer and fraction

List of publications

International Journals

1. Sikka Prateek, Abhijit R Asati, Chandra Shekhar, "Real-time FPGA Implementation of a High-Speed and Area-Optimized Harris Corner Detection Algorithm." **Elsevier** Microprocessors and Microsystems. (2020) – SCI Indexed, Impact Factor 1.161

2. Sikka Prateek, Abhijit R Asati, Chandra Shekhar, "Power and Area Optimized High-Level Synthesis Implementation of a Digital Down Converter for Software-Defined Radio Applications." **Springer** Circuits, Systems and Signal Processing (CSSP). 2020 SCI Indexed, Impact Factor 1.681

3. Sikka Prateek, Abhijit R Asati, Chandra Shekhar, "Speed optimal FPGA implementation of Encryption Algorithms for Telecom Applications." **Elsevier** Microprocessors and Microsystems. (2020) - SCI Indexed, Impact Factor 1.161

4. Sikka, Prateek, Abhijit R Asati, Chandra Shekhar, "High-Level Synthesis Assisted Design and Verification Framework for Automotive Radar Processors." **Elsevier** Microprocessors and Microsystems. (2020) SCI Indexed, Impact Factor 1.161

5. Sikka Prateek, Abhijit R Asati, Chandra Shekhar, "High-throughput field-programmable gate array implementation of the advanced encryption standard algorithm for automotive security applications." **Springer** Journal of Ambient Intelligence and Humanized Computing, 29 July, 2020. SCI Indexed, Impact Factor 4.594

6. Sikka Prateek, Abhijit R. Asati, Chandra Shekhar, "High-speed and area-efficient Sobel edge detector on field-programmable gate array for artificial intelligence and machine learning applications." *Computational Intelligence* ***Wiley*** *Online Library* (2020). SCI Indexed, Impact Factor 1.196

7. Sikka, Prateek, Abhijit R Asati, Chandra Shekhar, Area, speed and power optimized implementation of a band-pass FIR filter using high-level synthesis" **Springer** Wireless Personal Communications. (2021) – SCI Indexed, Impact Factor 1.061

Patent Application

1. Sikka Prateek, Abhijit Asati, Chandra Shekhar, "Method of High-Level Synthesis in Integrated Circuit Design using application specific bit widths." India Patent Application No. 201911028124.

International Conference (Presented Papers)

1. Sikka Prateek, Abhijit Asati, Chandra Shekhar, "Area-optimal FPGA implementation of the YOLO v2 algorithm using High-Level Synthesis." 2020 IEEE UPCON 26 Nov., MNNIT Allahabad.

2. Sikka Prateek, Abhijit Asati, Chandra Shekhar, "Low Area, High Throughput Field Programmable Gate Array Implementation of Microprocessor without Interlocked Pipeline Stages." 2020 Springer 3rd International Conference on VLSI, Communication and Signal Processing. 9-11 Oct, MNNIT Allahabad.

3. Sikka Prateek, Abhijit Asati, Chandra Shekhar," High Speed and Area Efficient Sobel Edge Detector on FPGA using application specific bit widths for intermediate nodes." iCASIC, 27-28 Feb, VIT, Vellore.

International Conference (Invited Tutorials)

1. Invited Tutorial: Sikka Prateek. "Design and Verification challenges for complex SoCs". 2019 Springer 2nd International Conference on VLSI, Communication and Signal Processing, 2019. MNNIT Allahabad.

2. Invited Tutorial: Sikka Prateek. "FPGA prototyping for VLSI designs using High level Synthesis." 2020 Springer 3rd International Conference on VLSI, Communication and Signal Processing., 2020. MNNIT Allahabad.

3. Invited Tutorial: Sikka Prateek, Abhijit Asati, Chandra Shekhar. "Novel methods for Area, Speed and Power optimization using HLS for FPGA prototyping." IEEE UPCON, 2020 MNNIT Allahabad