Vitis - High Level Synthesis
Review
Xilinx tools

RajPat

2020
October

thanks to
Vitis HLS, www.xilinx.com

Contents

Listings

1 High Level Synthesis

1.1 Introduction

1. HLS translates behavioral C/C++/OpenCL to RTL implementation.
2. Analogous to DSPs: C $\rightarrow$ machine code.
3. Analogous to GPUs: C (OpenCL) $\rightarrow$ Optimized C.
4. Vitis HLS has following libraries:
   (a) Arbitrary precision
   (b) Vision
   (c) Math
   (d) Stream
   (e) LogiCore IP
5. Provides a method to simulate the C/C++ kernels with GCC/G++, compare with a golden model.
6. Compiler directives can be used to generate low latency, high throughput design.
7. Coding style directly affects hardware realization.
8. Everything in code needs to specified at compile time.
9. Some stuff that's not supported.
   (a) System calls - printf(), fprintf(), fscanf(), getc(), time(), sleep(), etc. Vitis HLS defines "__SYNTHESIS__" macro which allows excluding non-synthesizable code form design.
   (b) Dynamic memory allocation is not supported, i.e. mallow(), alloc() and free() are not supported.
   (c) Recursive functions are not supported.
   (d) File I/Os are not supported.
   (e) Dynamic virtual funciton calls are not supported, polymorphism stuff is not supported.
   (f) General pointer casting is not supported, but native type pointer casting is supported. Funciton pointers are not supported. Pointer to pointers are not supported.
10. Many implementations of same source code are possible based on different coding styles.
11. Entire block always runs on a single clock. Multiple clock domains in a single design is not possible.

1.2 HLS Compiler

The following are phases/steps of HLS compiler

1. Scheduling (operation sequencing)
2. Binding (bind to a specific hardware to be realized)
3. Control logic optimization (control flow - FSMs)

Terminology and Metrics to describe a design:

1. Latency
2. Initiation Internal - Number of clock cycles before function can accept new input data
3. Throughput
4. Loop iteration latency
5. Loop initiation interval
6. Loop latency - Time after which new loop run can start. In absence of pragmas, hardware is reused.
7. Trip count - Number of iterations in loop
8. Data-rate

2 Vitis HLS tool flow

2.1 Phases

1. Converts function to hardware.
2. Converts each argument of top-level function to physical connection (I/O interface port of RTL).
3. Type of argument, i.e. data type/width, etc. - influences area and performance.
4. Comprehensive C/C++ support (Vitis HLS : C++11/C++14)
5. Ultrafast methodology optimization steps:
   (a) Simulate design
   (b) Synthesize design
   (c) Initial optimization (define interface, datapacking; loop trip counts)
   (d) Pipeline and dataflow
   (e) Partition memories and ports
   (f) Remove false dependencies
   (g) Optionally specify latency requirements to reduce it
   (h) Improve area by resource sharing

2.2 CLI commands

1. **vitis_hls -i** : Interactive mode, type tcl commands one at a time
2. **vitis_hls -f run.tcl** : tcl batch file
3. **vitis_hls -p my.prj** : GUI mode analysis of a project
4. **vitis_hls** : HLS GUI
5. **help** and **help ¡command¿** are super useful

2.3 Steps in Vitis HLS tool flow

1. Create project
2. Add sources
3. Add testbench sources
4. Set solution properties (clock period, uncertainty(margin for PnR - default is 12.5 % of tCK), part, directives, etc.)
5. csim_design
6. csynth_design
7. cosim_design (Re-uses same C testbench)
8. Export IP to catalog for use in Vivado

2.4 Initialization vs Reset

2.4.1 Initialization

1. Variables defined with static qualifier and those in global scope are initialized to 0 by default.
2. Can assign a specific initial value at compile time. Same initial value is implemented in RTL.

2.4.2 Reset

1. config_rtl config can be used to instantiate a reset port in RTL implementation.
2. Polarity and sync/async nature can be configured.
3. Which registers are reset when reset is applied (none, control, state(static and global vars) and all).

2.5 Library support

1. There are library header files in "include" directory of tool installation.
2. Arbitrary precision datatype library:
   (a) C based native datatypes are on 8 bit boundaries.
   (b) Arbitrary precision bit width are supported using integer and fixed point arbitrary precision datatypes.
3. Math library - hls_math.h
   (a) Provides coverage of C++ cmath libraries.
   (b) Can be used in both C simulation and synthesis.
   (c) Floating point for all functions and fixed-point support for majority of functions.
   (d) Functions are grouped in "hls" namespace. Can be used as "in-place replacement" of std namespace from standard C++ math library (cmath).
4. Streaming data library
   (a) C++ template class for modeling streaming data structures: **hls::stream¡¿**
   (b) Behaves like FIFO of infinite depth in C code (no need to define size).
   (c) Implemented with **ap_fifo** interface on the top-level interface.
   (d) **hls::stream<Type>** → Specify type of stream.
   (e) **hls::stream<Type, Depth>** → Specify type and depth of stream.
5. Video library
   (a) hls_video.h replaced by Vitis Vision library.
   (b) Computer Vision functions accelerated on FPGA.
   (c) **xf::cv::Mat** for templates and arguments that are images.
   (d) All funcitons in xf::cv namespace.
   (e) Major template arguments - Max. size of image, data type of each pixel, number of pixels to be processed per clock cycle, etc.
6. HLS IP Library
   (a) C++ libraries to implement IP Catalog IPs.
   (b) hls_fft.h - infer FFT from IP catalog.
   (c) hls_fir.h - inter FIR filter from IP catalog.
   (d) hls_dds.h - implements direct digital synthesizer (DDS)
   (e) ap_shift_reg.h - C++ class to implement shift register using SRL primitive.

2.6 Pragmas

Design exploration can be done using some "pragmas" and "directives". This can be used to optimize area/optimization. Some directive:

1. Unroll directive - Loop unrolling - realizes different hardware to run for different operations in loop count.
2. Pipeline directive
3. Dataflow directive

2.7 Arrays as arguments

1. When arrays are used as arguments to top-level function, Vitis HLS assumes memory is off hip, BRAM and HLS synthesizes interface pots to access memory.
2. Resource directive can be used to specify single-port/dual-port RAMs. By default uses single-port RAM.
3. Array partitioning: Splitting array into multiple independent arrays since each RAM only has max 2 ports. Block, cyclic and complete partitioning are 3 types of array partitioning.
4. Array reshaping: Combines data into wider containers, i.e. increase RAM data width. **ARRAY_RESHAPE** directive is used.

2.8 Ultrafast Design Methodology

1. onverts function to hardware.
2. Converts each argument of top-level function to physical connection (I/O interface port of RTL).
3. Type of argument, i.e. data type/width, etc. - influences area and performance.
4. Comprehensive C/C++ support (Vitis HLS : C++11/C++14)

3 Opt 1 - I/O Interfaces

1. All ports added by Vitis HLS have prefix of "ap_*".
2. If the design is combinatorial, no handshake signals are inferred.
3. On sequential designs, HLS adds handshake signals: "ap_start, ap_done, ap_idle".

4. User can use interface synthesis to define standard protocols or user defined protocol.
5. Vitis HLS has 2 types of interface synthesis protocols
   (a) Block level protocol:
   (b) Interface level protocol
6. Block-Level protocol is specified using INTERFACE directive with port = return.

$$example : \#pragma HLS\_INTERFACE ap\_ctrl\_h\_port = return$$

7. Port-Level protocol is specified using INTERFACE directive with port = ¡port name¿.

$$example : \#pragma HLS\_INTERFACE ap\_fifo\_depth = 16 port = a$$

3.1 Block Level Protocol

1. These signals control RTL block, independent of any port level I/O protocol.
2. By default a block level protocol is added to design (ap_start, ap_done, ap_idle, ap_ready if function is pipelined, etc.).
3. ap_ctrl_none, ap_ctrl_hs, ap_ctrl_chain are only specified on function return.
4. ap_ctrl_none : No handshake signals
5. ap_ctrl_hs : Default handshake signals, i.e. ap_start, ap_done, ap_idle, ap_ready
6. ap_ctrl_chain : In case the funciton is a pipelined model, i.e. ap_continue, ap_ready
7. s_axilite : Used when PS is used to configure HLS block. Many ports can be tied to axilite.
8. One or more or following conditions must be true for C/RTL cosim to work correctly with block level IO protocol.
   (a) Top level func must be synthesized using ap_ctrl_hs or ap_ctrl_chain.
   (b) Design must be purely combo.
   (c) Interface must be streaming and implemented with ap_fifo, ap_hs or axis.
   (d) Top level func. must have II of 1.

3.2 Port Level Protocol

1. After block level protocol has been used to control operation of block, port level I/O protocols are used to sequence data into and out of block.
2. Created for each argument of top level function.
3. All possible port level I/O protocols are
   (a) AXI Interface protocol (axis, m_axi, s_axilite)
   (b) No I/O protocol (ap_none, ap_stable)
   (c) Wire handshake protocol (ap_ack, ap_vld, ap_ovld, ap_hs)
   (d) Memory Interface protocol (ap_memory, bram, ap_fifo)
4.