Low vs High Level Programming for FPGA.

Jan Marjanovic (DESY) 2018-09-13

IBIC2018, Shanghai

MicroTCA TECHNOLOGY LAB
A HELMHOLTZ INNOVATION LAB

DESY

Outline

- High Level Synthesis introduction
- Theory of operation
- Examples
- Conclusion

Any fool can write code that a computer can understand.
Good programmers write code that humans can understand.

Martin Fowler, 2008

Introduction

Motivation

FPGA devices are large $\Rightarrow$ main challenges are:

- implementation of the algorithms
- connecting IP cores (modules) together

Algorithms - Levels of abstraction

Mathematical model

requirements for the application/project

$$pos = K\frac{a-b}{a+b}$$

High Level

close to the mathematical model, technical limitations starts to become visible

```
float pos = K * (a - b) / (a + b);
```

Register Transfer Level

implementation becomes disconnected from model

```
every clock cycle:
  state  := decide_next_state(state, input)
  output := gen_output(state)
```

Gate Level

physical details (e.g. propagation delay) are considered

Algorithms - Levels of abstraction

Mathematical model

requirements for the application/project

$$pos = K\frac{a-b}{a+b}$$

High Level

close to the mathematical model, technical limitations starts to become visible

```
float pos = K * (a - b) / (a + b);
```

high level synthesis

Register Transfer Level

implementation becomes disconnected from model

```
every clock cycle:
  state  := decide_next_state(state, input)
  output := gen_output(state)
```

synthesis

Gate Level

physical details (e.g. propagation delay) are considered

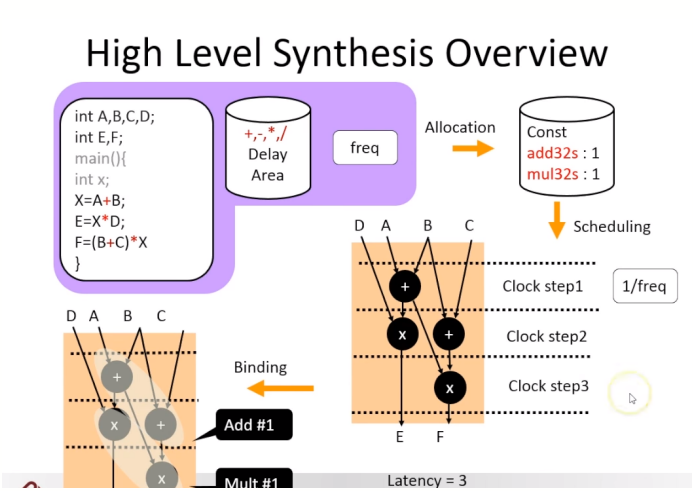
State-of-the-art compilers

- Xilinx Vivado HLS
- Xilinx System Generator for DSP
- Intel HLS Compiler
- LabVIEW FPGA
- Mathworks HDL Coder
- Cadence Stratus
- Mentor Graphics Catapult
- Synopsys Synphony C Compiler
- Panda Bamboo
- LegUp

Reasons to base the presentation on Vivado HLS

- Experience with Xilinx FPGAs in general and with related tools (Vivado, Vivado HLS)
- Xilinx FPGAs are heavily used at DESY and on MicroTCA AMC boards
- The created IP integrates nicely with the rest of the IPs in Xilinx ecosystem
- Vivado HLS is significantly cheaper than other HLS software suites; therefore it is very likely that industrial partners will have access to it

Theory of operation



B. Carrion Schafer, ”High-Level Synthesis (HLS) theory”, from:

https://www.youtube.com/watch?v=FzS-8hlnb2g

Other examples of HLS in experimental physics

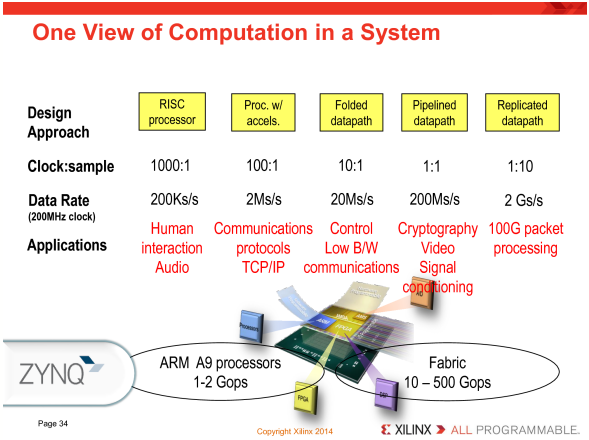
- P. Predki, M. Heuer, Ł. Butkowski, K. Przygoda, H. Schlarb, and A. Napieralski, **"Rapid-X - an FPGA development toolset using a custom Simulink library for MTCA.4 modules"**
- E. Schubert and U. Langenbach, **"FPGA-based hardware accelerators for 10/40 GigE TCP/IP and other protocols,"**
- T. Marc-Andre, **"Two FPGA case studies comparing High Level Synthesis and manual HDL for HEP applications,"**
- J. Duarte et al., **"Fast inference of deep neural networks in FPGAs for particle physics,"**

Examples

Examples introduction

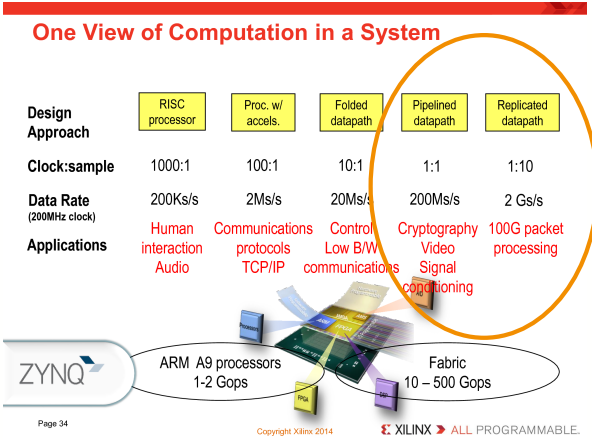
- Example 1: linearization module
  polynomial evaluation,
  uni-directional data flow
- Example 2: two-dimensional mean and stdev
  multiply-and-accumulate,
  dependency between accumulator and the incoming samples
- Example 3: IIR filter
  well-studied DSP block
  required rate: 1 sample per clock cycle

Examples introduction



From: https://ieeexplore.ieee.org/document/7086413/

Examples introduction



Presented examples require high performance implementation

From: https://ieeexplore.ieee.org/document/7086413/

Example 1: linearization

Introduction:

- DFMC-DS800 (800 MHz, 12-bit digitizer) is installed on Xilinx KCU105 evaluation kit
- ADC is producing samples at 800 MSPS
- Data from ADC is split into 4 lanes $\Rightarrow$ helps with timing closure
- Coefficients are static (defined at the compile time)

On each sample we need to evaluate a polynomial:

$$y[n] = c_0 + c_1 * x[n] + c_2 * x^2[n]$$

$$y[n] = c_0 + x[n] * [c_1 + [x[n] * c_2]]$$

Values are not important for this talk, but it is important to mention:

$$c_1 \approx 1, c_2 \lll 1$$

Example 1: VHDL implementation

```
GEN_STAGES: for i in 0 to C_NUM_COEFS-1 generate
  proc_stage: process (clk)
  begin
    if rising_edge(clk) then
      stage_out(2*i+1) <= resize(
          arg => to_sfixed(in_data_prev(2*i),
                           in_data'left, in_data'right)
                 * stage_out(2*i),
          size_res => stage_out(0)
      );

      stage_out(2*i+2) <= resize(
          arg => C_COEFS(i) + stage_out(2*i+1),
          size_res => stage_out(0)
      );
    end if;
  end process;
end generate;
```

Example 1: VHDL implementation

```
GEN_STAGES: for i in 0 to C_NUM_COEFS-1 generate
  proc_stage: process (clk)
  begin
    if rising_edge(clk) then
      stage_out(2*i+1) <= resize(
          arg => to_sfixed(in_data_prev(2*i),
                           in_data'left, in_data'right)
                 * stage_out(2*i),
          size_res => stage_out(0)
      );

      stage_out(2*i+2) <= resize(
          arg => C_COEFS(i) + stage_out(2*i+1),
          size_res => stage_out(0)
      );
    end if;
  end process;
end generate;
```

Very simple example, but we are already modifying the original equation

Example 1: C++ implementation

Implementation:

```
output_t linearize(input_t in) {
  internal_t tmp = 0;
  for (int i = COEFFS_LEN-1; i >= 0; i--) {
    tmp = COEFFS[i] + in * tmp;
  }
  return tmp;
}
```

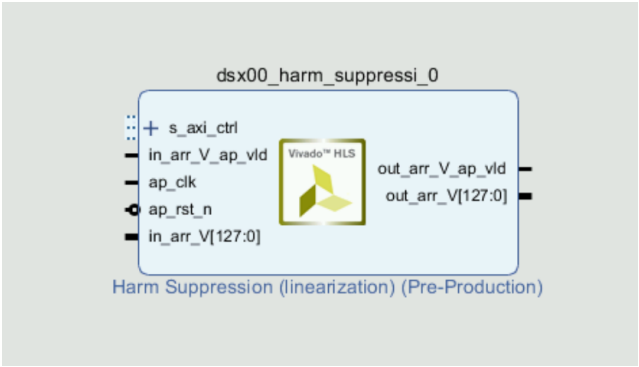
Example 1: C++ implementation

Implementation:

```
output_t linearize(input_t in) {
  internal_t tmp = 0;
  for (int i = COEFFS_LEN-1; i >= 0; i--) {
    tmp = COEFFS[i] + in * tmp;
  }
  return tmp;
}
```

Better, original equation still preserved

Example 1: generated module



Example 1: results

Compiled for XCKU040 on KCU105 board

resource	HLS (in C++)	RTL (in VHDL)
CLB	221	314
LUT	461	1081
FF	884	938
DSP	16	24
latency	6	6
interval	1	1
clk period	3.903 ns	4.954 ns
lines of code	52	170

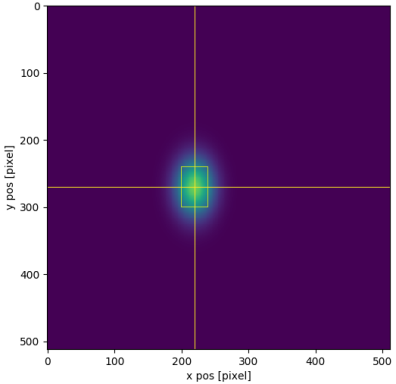
Table: Comparison between implementations of 2nd order polynomial in HLS vs RTL

HLS uses GCC front-end to analyze the code and to perform analysis on the constants → bits which are not needed are trimmed away

Example 2: two-dimensional standard deviation

Intro:

- GigE Vision implementation on FPGA
- 2D image transmitted as AXI-Stream
- data could be transmitted over 10 Gigabit Ethernet (32-bit, 312.5 MHz)



Example 2: two-dimensional standard deviation

Algorithms to consider:

- naïve (two-pass method)

$$\mu = \frac{1}{N}\sum x_i, \quad \sigma^2 = \frac{1}{N}\sum (x_i - \mu)^2$$

- Knuth's

$$M_k = M_{k-1} + (x_k - M_{k+1})/k$$

- naïve, rewritten

$$\sigma^2 = \frac{1}{N}\sum x_i^2 - \mu^2$$

Example 2: code

Top-level function:

```
void two_dim_stdev(
  hls::stream<input_t> &in, int &meanx, int &stdx, int &meany, int &stdy
) {

#pragma HLS INTERFACE ap_ctrl_none port=return
#pragma HLS INTERFACE axis register both port=in
#pragma HLS DATA_PACK variable=in field_level
#pragma HLS INTERFACE s_axilite port=meanx bundle=ctrl
#pragma HLS INTERFACE s_axilite port=stdx  bundle=ctrl
#pragma HLS INTERFACE s_axilite port=meany bundle=ctrl
#pragma HLS INTERFACE s_axilite port=stdy  bundle=ctrl
```

...

Example 2: code

...
initialization of the variables

```
const int N = 512; // frame size
const int PAR = 4; // number of parallel HW instances for accum

ap_uint<30> accum[PAR*INPUT_W];
ap_uint<38> accum_x[PAR*INPUT_W], accum_y[PAR*INPUT_W];
ap_uint<45> accum_x2[PAR*INPUT_W], accum_y2[PAR*INPUT_W];

loop_init: for (int i = 0; i < PAR*INPUT_W; i++) {
  accum[i] = 0; accum_x[i] = 0; accum_y[i] = 0;
  accum_x2[i] = 0; accum_y2[i] = 0;
}
```

...

Example 2: code

...

loops

```
    loop_y: for (int y = 0; y < N; y++) {
      loop_x: for (int x = 0; x < N/PAR/INPUT_W; x++) {
        input_t z[PAR];
        loop_pipe: for (int i=0; i < PAR; i++) {
#pragma HLS PIPELINE II=1
          in >> z[i];
          loop_arr: for (int j = 0; j < INPUT_W; j++) {
            accum[i]    += z[i].arr[j];
            ap_uint<10> pos_x = (PAR*INPUT_W*x+INPUT_W*i+j);
            ap_uint<10> pos_y = y;
            accum_x[i]  += pos_x*z[i].arr[j];
            accum_y[i]  += pos_y*z[i].arr[j];
            accum_x2[i] += pos_x*pos_x*z[i].arr[j];
            accum_y2[i] += pos_y*pos_y*z[i].arr[j];
          }
        }
      }
    }
```

...

Example 2: code

...
gathering all together

```
    ap_uint<30> accum_tot = 0;
    ap_uint<38> accum_x_tot = 0, accum_y_tot = 0;
    ap_uint<45> accum_x2_tot = 0, accum_y2_tot = 0;

    loop_gather: for (int  i=0; i < PAR; i++) {
      accum_tot    += accum[i];
      accum_x_tot  += accum_x[i];
      accum_y_tot  += accum_y[i];
      accum_x2_tot += accum_x2[i];
      accum_y2_tot += accum_y2[i];
    }

    meanx = accum_x_tot / accum_tot;
    meany = accum_y_tot / accum_tot;
    ap_int<16> varx = accum_x2_tot / accum_tot - meanx*meanx;
    ap_int<16> vary = accum_y2_tot / accum_tot - meany*meany;
    stdx  = hls::sqrt(varx);
    stdy  = hls::sqrt(vary);
}
```

Example 2: results

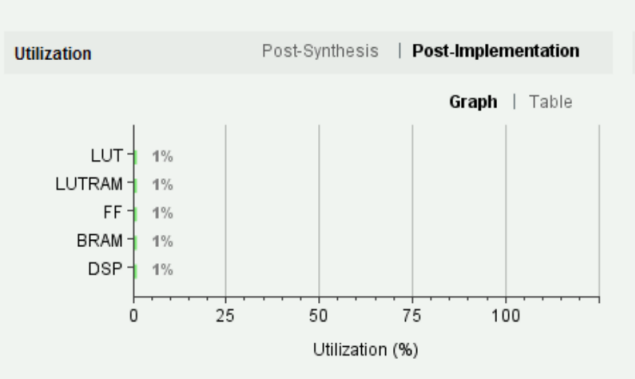
Compiled for XC7420T on DAMC-TCK7 board

resource	HLS (in C++)	RTL (in VHDL)
slice	764	-
LUT	1714	-
FF	2507	-
DSP	18	-
BRAM	14	-
SRL	38	-
latency	65663	-
interval	65663	-
clk period	3.095 ns	-

Table: Report of the utilization of resources from implementation of two-dimensional standard deviation and mean in Vivado HLS

Example 2: results

resource utilization in percent as reported by Vivado



Example 3: IIR filter

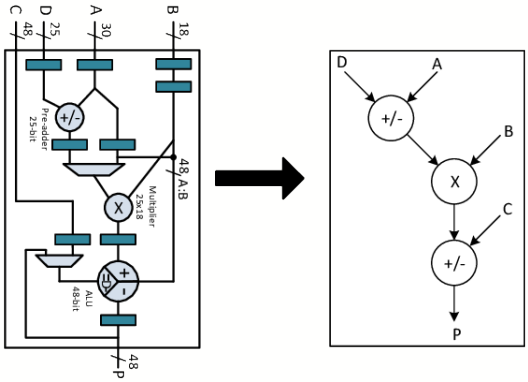
- Second-order IIR filter
- compared with the VHDL implementation used in our MTCA Firmware framework

Equation for the second order IIR filter:

$$y[n] = b_0 * x[n] + b_1 * x[n-1] + b_2 * x[n-2] - a_1 * y[n-1] - a_2 * y[n-2]$$

Example 3: IIR filter

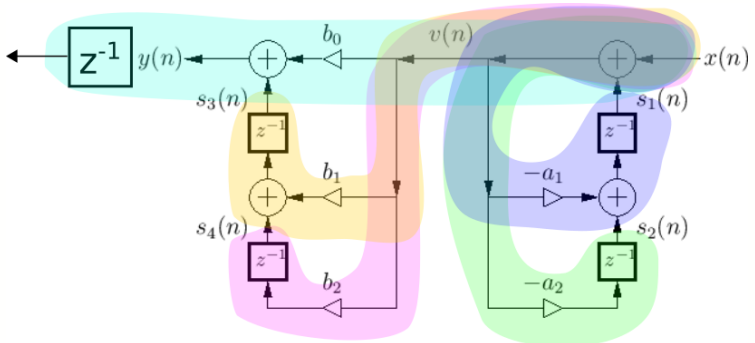
Xilinx DSP48E1 hard-IP block



from R. Bajaj, S. Fahmy: Mapping for maximum performance on FPGA DSP blocks

Example 3: IIR filter

VHDL implementation instantiatetes 5 DSP48E1 blocks



Example 3: IIR filter

C++ implementation is much more clearer

```
template<int FIX_W, int FIX_I>
class BiquadFilter {
public:
  BiquadFilter (
      ap_fixed<FIX_W, FIX_I> b0,
      ap_fixed<FIX_W, FIX_I> b1,
      ap_fixed<FIX_W, FIX_I> b2,
      ap_fixed<FIX_W, FIX_I> a1,
      ap_fixed<FIX_W, FIX_I> a2) :
    b0(b0), b1(b1), b2(b2), a1(a1), a2(a2) {
  }

  ap_fixed<FIX_W, FIX_I> operator()(ap_fixed<FIX_W, FIX_I> x);

private:
  ap_fixed<FIX_W, FIX_I> b0, b1, b2, a1, a2;
  ap_fixed<25, 9> s1; ap_fixed<48, 9> s2, s3, s4;
};
```

Example 3: IIR filter

C++ implementation is much more clearer

```
ap_fixed<FIX_W, FIX_I> BiquadFilter::operator()(ap_fixed<FIX_W, FIX_I> x){
  ap_fixed<25, 9> tmp_s1    = (x+s1) * -a1 + s2;
  ap_fixed<48, 9> tmp_s2    = (x+s1) * -a2     ;
  ap_fixed<48, 9> tmp_s3    = (x+s1) *  b1 + s4;
  ap_fixed<48, 9> tmp_s4    = (x+s1) *  b2     ;
  ap_fixed<FIX_W, FIX_I> y = (x+s1) *  b0 + s3;

  s1 = tmp_s1; s2 = tmp_s2; s3 = tmp_s3; s4 = tmp_s4;
  return y;
}
```

Example 3: results

Compiled for XCKU040 on KCU105 board

resource	HLS (in C++)	RTL (in VHDL)
CLB	32	16
LUT	156	0
FF	147	149
DSP	5	5
BRAM	0	0
SRL	0	0
latency	4	4
interval	1	1
clk period	4.704 ns	4.732 ns
lines of code	60	246

Table: Comparison between implementations of 2nd order IIR filter in HLS vs RTL

Other examples

I have used Vivado HLS to develop:

- Linear calibration (on Virtex 5 on DAMC-FMC25)
- PID controller (on Zynq)
- IIR low-pass filter (on Zynq, Spartan-6 on DAMC-FMC25)
- UDP/IPv4 packet generator (on Virtex 5 on DAMC-FMC25 with DFMC-4SFP+)
- Piece-wise linear calibration (on Virtex 5 on DAMC-FMC25)

Conclusion

- Vivado HLS provides an easier way to implement DSP algorithms
- Less code = less bugs
- Code is more readable $\Rightarrow$ wider audience
- Quality of Result is comparable with hand-coded logic
- Provides easy migration between different FPGA families
- Requires (some) knowledge of FPGA architecture

Thank you for your attention!