FLIR Systems AB | Stefan Olsson

Development of High Performance Video Processing Using HDL Coder

Challenge
Reduce developing cost and time to market for new video processing functionality

Solution
Introducing MATLAB HDL Coder as a way bridge the gap between system designer and hardware engineer

Results
- We have successfully integrating HDL Coder in our workflow
- A number of new features has been added to our new HD infrared camera using only auto generated VHDL
- We intend to keep on working with HDL Coder and with gradual improvements we expect to meet our long term efficiency goals
Six Business Segments

**Instruments**
- Automation/Process Monitoring
- Building Diagnostics
- Electrical/Mechanical Inspection
- Optical Gas Imaging
- Range/Science/R&D
  - Extech brand

**OEM & Emerging**
- Automotive Night Vision
- Personal Vision Systems
- Intelligent Traffic Systems
- Mobile Accessories
- Camera Cores and Detectors

**Surveillance**
- Border/Force Protection Solutions
  - Airborne
  - Maritime
- Land Surveillance
  - Vehicle Systems
  - Man Portable

**Maritime**
- Commercial Maritime
- Navigation/Night Vision
  - Raymarine brand

**Detection**
- Chemical
- Biological
- Radiological
- Nuclear
- Explosives

**Security**
- Facility Security
  - Lorex brand
Key figures 2013
• Revenue 1 496 MUSD
• Employees 2 962
• R&D 148 MUSD (9.9% of revenue)
• Net earnings 177 MUSD
FLIR ONE™ (human detection ~40 m)

Ranger HDC (human detection ~25 km)
Why Video Processing?

Short introduction

Using simplest algorithm
Only 1% of the available information reach the operator!!
The Challenge

Algorithms are cheap

Implementation is expensive

Silicon is cheap

Productivity

Graphical System Editor
Qsys
Simulink

OpenCL
HLS

MATLAB

Behavioral HDL

Structural HDL
Workflow – Algorithm development

**Capture**
- Capture HD-SDI 16-bit raw video

**Analyze**

**Implement**

**Fine tune**

- Algorithm development
Workflow - Implementation

Algorithm work
- Test of new ideas in MATLAB
- Write and document reference algorithm C-compliant MATLAB

Implementation work
- Auto generate C/C++
- Hand code C/C++
- Hand code HDL
- Auto generate HDL
- Hand code C/CUDA

Embedded target
- CPU/DSP
- FPGA
- GPU
Workflow - Implementation

Algorithm work

Test of new ideas in MATLAB

Write and document reference algorithm C-compliant MATLAB

Auto generate C/C++
Hand code C/C++
Hand code HDL
Auto generate HDL
Hand code C/CUDA

CPU/DSP
FPGA
GPU
Workflow - MATLAB HDL Coder

Task from stakeholder
Test of ideas in MATLAB
Reference algorithm C-compliant
Streaming loop float version
Streaming loop fixed point version
HDL Generation
Integration into HDL top
Analyze results (timing, precision, verify, size)
Validation in camera

Write documentation
Hand-code HDL
Verify results

Size/Timing still not OK for production
Coding example (pipelining)

%% Sum of weighted pixels
pixel_sum = sum( k_lp .* wt );

%% Sum of weights
weight_sum = sum( wt );

11 multiplications and 22 additions on one clock-cycle
Far from meeting timing (148MHz, 6ns)
Coding example (hierarchy)

Testbench

```matlab
for pix_count = 1:length(i_data)
    [lb_data, lb_hload, lb_vload, lb_hsync, lb_vsync] = ...;
    GenLineBuffer_3(i_data(pix_count), i_hload(pix_count), ...
    i_vload(pix_count), i_hsync(pix_count), ...
    i_vsync(pix_count));

    [lb_aa_data, lb_aa_hload, lb_aa_vload, lb_aa_hsync, lb_aa_vsync] = ...;
    GenLineBuffer_3(i_active_area(pix_count), i_hload(pix_count), ...
    i_vload(pix_count), i_hsync(pix_count), ...
    i_vsync(pix_count));

    [lb_fb_data, lb_fb_hload, lb_fb_vload, lb_fb_hsync, lb_fb_vsync] = ...;
    GenLineBuffer_3(o_fb_data(pix_count), i_hload(pix_count), ...
    i_vload(pix_count), i_hsync(pix_count), ...
    i_vsync(pix_count));

    [atf_data, atf_filtered_data, atf_active_area, atf_weights_data, ...
    atf_imdiff, atf_filtered, atf_ffc_weights, atf_hload, atf_vload, atf_hsync, atf_vsync, o_version] = ...;
    ATF(lb_data', lb_fb_data', lb_aa_data', ATF_NF, ATF_NF_GLOBAL, ...
    lb_hload, lb_vload, lb_hsync, lb_fb_vsync);

    o_data(pix_count) = atf_data;
    o_fb_data(FB_OFFSET+pix_count-9-(SIM_BUF_COLS+SIM_COLS)) = atf_filtered_data;
    o_active_area(pix_count) = atf_active_area;
    o_hload(pix_count) = atf_hload;
    o_vload(pix_count) = atf_vload;
    o_hsync(pix_count) = atf_hsync;
    o_vsync(pix_count) = atf_vsync;
    o_processed_data(pix_count) = atf_filtered_data;
end
```

Does not work!
Coding example (hierarchy)

Testbench

```matlab
for pix_count = 1:length(i_data)
    [lb_data, lb_hload, lb_vload, lb_hsync, lb_vsync] = ...
        GenLineBuffer_3(i_data(pix_count), i_hload(pix_count), ... 
        i_vload(pix_count), i_hsync(pix_count), ... 
        i_vsync(pix_count));

    [lb_aa_data, lb_aa_hload, lb_aa_vload, lb_aa_hsync, lb_aa_vsync] = ...
        GenLineBuffer_3AA(i_active_area(pix_count), i_hload(pix_count), ... 
        i_vload(pix_count), i_hsync(pix_count), ... 
        i_vsync(pix_count));

    [lb_fb_data, lb_fb_hload, lb_fb_vload, lb_fb_hsync, lb_fb_vsync] = ...
        GenLineBuffer_3FB(o_fb_data(pix_count), i_hload(pix_count), ... 
        i_vload(pix_count), i_hsync(pix_count), ... 
        i_vsync(pix_count));

    [atf_data, atf_filtered_data, atf_weights_data, ...
    atf_imdiff, atf_imdiffc, atf_ffc_weights, atf_hload, atf_vload, atf_hsync, atf_vsync, o_version] = ...
        ATF(lb_data', lb_fb_data', lb_aa_data', ATF_NF, ATF_NF_GLOBAL, ...
        lb_hload, lb_vload, lb_hsync, lb_fb_vsync);

    o_data(pix_count) = atf_data;
    o_fb_data(FB_OFFSET+pix_count-9-(SIM_BUF_COLS+SIM_COLS)) = atf_filtered_data;
    o_active_area(pix_count) = atf_active_area;
    o_hload(pix_count) = atf_hload;
    o_vload(pix_count) = atf_vload;
    o_hsync(pix_count) = atf_hsync;
    o_vsync(pix_count) = atf_vsync;
    o_processed_data(pix_count) = atf_filtered_data;
end
```
Coding example (mixed hand-coded and autogen.)

Testbench

```matlab
for pix_count = 1:length(i_data)
    [alph_data, alph_psum, alph_wsum, alph_diffsum, alph_entval, ...]
    alpha_wsum, alpha_idxg, alpha_hload, alpha_vload, alpha_hsync, ...
    alph_vsync, o_version] = ...
    ALPH(i_data(pix_count), i_data(pix_count), 0, SPAT_GAIN, ...
    RAD_GAIN, i_hload(pix_count), i_vload(pix_count), ...]
    i_hsync(pix_count), i_vsync(pix_count));
    [ alpdiv_lpdata, alpdiv_alpha, alpdiv_entval ] = ...
    ALPDIV( alph_data, alph_psum, alph_wsum, alph_diffsum, ...]
    alph_entval, DNR_GAIN, ENT_GAIN, R );
    [genhist_data, genhist_ip_data, genhist_hist_data, genhist_hload, ...]
    genhist_vload, genhist_hiStatsync, genhist_hsync, genhist_vsync, ...
    genhist_loPercL, genhist_hiPercL, genhist_histSum] = ...
    GenHist(alph_data, alpdiv_lpdata, alpdiv_entval, ...]
    alpha_hload, alpha_vload, ...
    alph_hsync, alpha_vsync, heq_hiStatsync, ...
    NOPIXLOW, NOPIXHIGH, GENHIST_MAX);
    [div_sync, div_res] = div_gen_v4_0(heq_div_sync, heq_div_num, heq_div_denum);
    [ heq_data, heq_ip_data, heq_he_data, o_hload(pix_count), ...]
    o_vload(pix_count), o_hsync(pix_count), o_vsync(pix_count), ...]
    heq_hiStatsync, heq_div_sync, heq_div_denum, heq_div_num, ...]
    heq_max_tfn_slope, o_version, o_debug1, o_debug2, o_debug3 ] = ...
    HEQ( genhist_data, genhist_ip_data, genhist_data, genhist_hist_data, ...
    genhist_hload, genhist_vload, genhist_hiStatsync, ...]
    genhist_vsync, genhist_vsync, genhist_loPercL, ...
    genhist_hiPercL, genhist_histSum, HE_FRAC OFFSET, ...]
    HE_FRAC_GAIN, HE_FOOT_OFFSET, HE_FOOT_GAIN, HE_HEAD_OFFSET, ...
    HE_HEAD_GAIN, DET_GAIN, div_sync, div_res);
    o_data(pix_count) = heq_data;
    o_processed_data(pix_count) = heq_he_data;
end
```

Mix of auto-generated and manual coding
Reasons:
1. The block already exists
2. Difficult to meet timing using auto-generated
3. Unable to generate RAM
4. Auto-generated take too much logic
5. Multi-clock domain requirements
Evaluation of MATLAB HDL Coder

Strengths

+ MATLAB is the language of algorithm designers
  ⇒ *No need of translation*
+ Early test of ideas in real hardware
  ⇒ *Rapid algorithm iteration*
+ Test-bench stimulus generation and analysis is very easy in MATLAB
  ⇒ *Speed up development*
+ Forces the algorithm designer to think FPGA’s
  ⇒ *Streamlines the algorithm to “fit” in FPGA’s*
+ The MATLAB code can easily be shared across different development sites
  ⇒ *Assures best practice and that the wheel is not reinvented*
+ Facilitate the collaboration between algorithms designer and HW engineer
  ⇒ *Minimize misunderstandings and improve the working climate*
Evaluation of MATLAB HDL Coder

Weaknesses

- Less efficient HDL code
  ⇒ Waste of hardware resources (in our case 0-50% more than hand-coded)
- Difficult for a traditional MATLAB user to write successful code for HDL generation
  ⇒ Slow down development
- Cumbersome way of adding pipeline registers
  ⇒ Makes m-code difficult to read
- Difficult to map variables to RAM memory
  ⇒ The algorithm unnecessarily waste logic
- Single clock domain only
  ⇒ Multi clock domain units has to be hand-coded (e.g. CPU-interface)
- No way of reuse m-functions in the same algorithm due to persistent variables
  ⇒ Makes it difficult to build a library of functions needed for implement complex algorithms.
    (this could be improved by using System Objects)
- Many updates and new features at each new release
  ⇒ The way you can write efficient m-code changes and requires rework
Conclusions

- Work done so far in HDL Coder
  - Automatic Detail Enhancement
  - Motion Compensated Temporal Filter
  - Electronic Stabilization
  - Pixelwise Video Blending
  - Lens Distortion Correction
  - Scene-based Image Uniformity Correction

- All done using auto generated code only!

- FLIR Systems is going to keep on working with HDL Coder – Requires a tight collaboration between FLIR and Mathworks

- The most significant result so far is bridging the gap between algorithm/system designer and hardware engineer

- Next steps (biggest issues):
  1. Easy pipelining without messing around with M-code
  2. More efficient use of logic
  3. Solve the modularization/hierarchy problem
  4. Get rid of dependencies on hand-coded VHDL
  5. More readable MATLAB code
Thank you!