The HercuLeS HLS environment

Nikolaos Kavvadias
nkavvadias@ajaxcompilers.com

CEO, Ajax Compilers,
Athens, Greece
www.ajaxcompilers.com
The need for high-level synthesis (HLS)

- Moore’s law anticipates an annual increase in chip complexity by 58%
- At the same time, human designer’s productivity increase is limited to 21% per annum
- This designer-productivity gap is a major problem in achieving time-to-market of hardware products

Solution  Adoption of a high-level design and synthesis methodology imposing user entry from a raised level of abstraction

• Hide low-level, time-consuming, error-prone details
• Drastically reduce human effort
• End-to-end automation from concept to production

HLS  An algorithmic description is automatically synthesized to a customized digital embedded system
Current status of HLS tools

- The HLS tree bares fruits of all sorts; academic and commercial
  - Source releases, binary releases, web tools, vendor-dependent tools, “unreleases” (unavailable for testing)
- Commercial (CatapultC, ImpulseC, Cadence C-to-Silicon, Synopsys Synphony, Xilinx Vivado HLS)
  - ASIC-oriented tools priced within the 6-digit range ($100,000+)
  - Xilinx Vivado HLS (formerly AutoESL) for FPGA prototyping is priced at $4,800
- Tools with free source/binaries (ROCCC, GAUT, SPARK, PandA, LegUp)
  - Unsupported in the long term; most of them abandoned after funding ends/Ph.Ds get completed
- Web access tools (C-to-Verilog, TransC)
  - Generating incomplete designs, no testbench, limited C support
Omissions, limitations and inefficiencies of current HLS tools

- The devise and use of non-standard, idiosyncratic languages
  - HercuLeS connects to external frontends via a simple interface. Apart from test frontends, work is underway for GCC/GIMPLE and clang/LLVM support

- Insufficient, opaque representations, recording only partial information
  - Uses a universal typed-assembly language, called NAC, as an intermediate representation

- Maintenance difficulties; code and API bloat as longevity threats
  - Optimizations added as self-contained external modules

- Mandating the use of code templates
  - HercuLeS does not rely on code templates since it uses a graph-based backend

- Succumbing to vendor and technology dependence
  - The generated HDL is human-readable and vendor- and technology-independent
What’s in a name

**HercuLeS**: An extensible, high-level synthesis (HLS) environment for whole-program hardware compilation with pluggable analyses and optimizations

named after the homonymous constellation and not the mighty but flawed demigod
The HercuLeS environment

- HercuLeS is a new high-level synthesis tool marketed by Ajax Compilers
- Easy to use, extensible, high-level synthesis (HLS) environment for whole-program hardware compilation
- In development since 2009
- HercuLeS targets both hardware and software engineers/developers
  - ASIC/SoC developers, FPGA-based/prototype/reference system engineers
  - Algorithm developers (custom HW algorithm implementations)
  - Application engineers (application acceleration)
The HercuLeS flow

- Optimized C code passed to GCC for GIMPLE generation
- `gimple2nac` translates to N-Address Code (NAC) IR
- HercuLeS = `nac2cdfg + cdfg2hdl`
  - `nac2cdfg`: SSA construction/CDFG extraction from NAC
  - `cdfg2hdl`: automatic FSMD hardware and self-checking testbench generation
- Modular and extensible flow; support for the basic GMP (multi-precision integer) API/DSL added in 24h (3 days)
  - `mpint.vhd`: MP integer and SLV operators: 500 LOC, 12h
  - `gimple2nac` extensions: 50 LOC, 4h
  - HercuLeS additions: 150 LOC, 8h
Features

- Automatic RTL VHDL code and testbench generation
- Automatic user-defined IP integration
- C subset frontend
- HercuLeS GUI
- Parallel operation scheduling with chaining optimizations
- Arithmetic optimizations, register optimization, C source code optimizer incl. array flattening optimizations
- VHDL-2008 floating-point and fixed-point arithmetic support
- GNU multi-precision integer extensions
- C verification backend
- GHDL/Modelsim support
- HercuLeS GUI
The user supplies NAC for a translation unit and reference test data

- **nac2cdfg**: translator from NAC to flat CDFGs; generates C backend files and VHDL packages for compound data types
- Each source procedure is represented by a CDFG
- **cdfg2hdl**: maps CDFGs (*.dot) to an extended FSMD MoC
- All required scripts are automatically generated (GHDL/Modelsim simulation, logic synthesis, backend C compilation)
- Diagnostic simulation output; tracing of VHDL signals/C variables

Nikolaos Kavvadias nkavvadias@ajaxcompilers.com
The HercuLeS HLS environment
NAC (N-Address Code)

- NAC is a procedural intermediate language
  - Extensible typed-assembly language similar in concept to GCC’s GIMPLE and LLVM, but more generic and simple
  - Arbitrary $m$-to-$n$ mappings, virtual address space per array
  - Statements: operations (atomic) and procedure calls (non-atomic)
  - Bit-accurate data types (integer, fixed-point, single/double/custom floating-point arithmetic)
  - Uses: RISC-like VM for static/dynamic analyses, CDFG extraction, graph-based data flow analyses, input to HLS kernels, software compilation

- GCC GIMPLE: three-address code IR; actual semantics not yet complete/stable; GCC code base huge and cluttered
- LLVM IR: low-level IR, not directly usable as a machine model; backward compatibility issues; provides access to a modern optimization infrastructure

Nikolaos Kavvadias nkavvadias@ajaxcompilers.com

The HercuLeS HLS environment
nac_top = {gvar_def} {proc_def}.
gvar_def = "globalvar" anum decl_item_list ";".
proc_def = "procedure" [anum] "(" [arg_list] ")"
       "{" [{lvar_decl}] [{stmt}] "}".
stmt = nac | pcall | id ";".
nac = [id_list "="] anum [id_list] ";".
pcall = ["(" id_list ")" "="] anum ["(" id_list ")"] ";".
id_list = id {"," id}.
decl_item_list = decl_item "," decl_item}.
decl_item = (anum | uninitarr | initarr).
arg_list = arg_decl "," arg_decl}.
arg_decl = ("in" | "out") anum (anum | uninitarr).
lvar_decl = "localvar" anum decl_item_list ";".
initarr = anum ["" id "]" ")" ")" [{ numer "," numer} "]".
uninitarr = anum ["" [id] "]".
anum = (letter | "_") {letter | digit}.
id = anum | ([]"-" [integer | fxpnum]).
Example translation flow: 2D Euclidean distance approximation (overview)

- Approximating the euclidean distance of a point \((x, y)\) from the origin by:
  \[
  eda = \text{MAX}((0.875 \times x + 0.5 \times y), x) \quad \text{where} \quad x = \text{MAX}(|a|, |b|) \quad \text{and} \quad y = \text{MIN}(|a|, |b|)
  \]
- Average error against \((\text{dist} = \sqrt{a^2 + b^2})\) is 4.7\% when compared to the \([\text{dist}]\) (rounded-down) and 3.85\% to the \([\text{dist}]\) (rounded-up) value

```c
#define ABS(x) ((x)>0?(x):(-x))
#define MAX(x,y) ((x)>(y)?(x):(y))
#define MIN(x,y) ((x)<(y)?(x):(y))

int eda(int in1, int in2) {
    int t1, t2, t3, t4, t5;
    int t6, x, y;
    t1 = ABS(in1);
    t2 = ABS(in2);
    x = MAX(t1, t2);
    y = MIN(t1, t2);
    t3 = x >> 3;
    t4 = y >> 1;
    t5 = x - t3;
    t6 = t4 + t5;
    return MAX(t6, x);
}
```

ANSI C

```c
procedure eda (in s16 in1, in s16 in2, out u16 out1) {
    localvar u16 x, y, t1, t2, t3, t4, t5, t6, t7;
    S_1:
    t1 <= abs in1;
    t2 <= abs in2;
    x <= max t1, t2;
    y <= min t1, t2;
    t3 <= shr x, 3;
    t4 <= shr y, 1;
    t5 <= sub x, t3;
    t6 <= add t4, t5;
    t7 <= max t6, x;
    out1 <= mov t7;
}
```

NAC IR

Nikolaos Kavvadias nkavvadias@ajaxcompilers.com

The HercuLeS HLS environment
Example translation flow: 2D Euclidean distance approximation (VHDL code)

```vhdl
when S_ENTRY =>
  ready <= '1';
  if (start = '1') then
    next_state <= S_001_001;
  else
    next_state <= S_ENTRY;
  end if;
when S_001_001 =>
  if (in1(15) = '1') then
    t1_next <= slv(not(unsigned(in1)) + "1");
  else
    t1_next <= in1;
  end if;
  if (in2(15) = '1') then
    t2_next <= slv(not(unsigned(in2)) + "1");
  else
    t2_next <= in2;
  end if;
next_state <= S_001_002;
when S_001_002 =>
  if (t1_reg > t2_reg) then
    x_next <= t1_reg;
  else
    x_next <= t2_reg;
  end if;
  if (t1_reg < t2_reg) then
    y_next <= t1_reg;
  else
    y_next <= t2_reg;
  end if;
y_next <= t2_reg;
end if;
next_state <= S_001_003;
when S_001_003 =>
  t3_next <= "000" & x_reg;
  t4_next <= "0" & y_reg;
next_state <= S_001_004;
when S_001_004 =>
  t5_next <= slv(unsigned(x_reg) - unsigned(t3_reg));
next_state <= S_001_005;
when S_001_005 =>
  t6_next <= slv(unsigned(t4_reg) + unsigned(t5_reg));
next_state <= S_001_006;
when S_001_006 =>
  if (t6_reg > x_reg) then
    t7_next <= t6_reg;
  else
    t7_next <= x_reg;
  end if;
next_state <= S_001_007;
when S_001_007 =>
  out1_next <= t7_reg;
next_state <= S_EXIT;
when S_EXIT =>
  done <= '1';
next_state <= S_ENTRY;
```

Nikolaos Kavvadias nkavvadias@ajaxcompilers.com
The HercuLeS HLS environment
SSA (Single Static Assignment) form construction

- Pseudo-statements called $\phi$-functions join variable definitions from different control-flow paths
- Enforce a single definition site for each variable
- False dependencies are naturally removed
- Many analyses and optimizations are simplified
- HercuLeS supports minimal SSA (in the number of $\phi$s) and intrablock-only (pseudo) SSA
Introduction to the SSA (Static Single Assignment) form

- SSA enforces a single `def` site for each variable
- Pseudo-statements called $\phi$-functions join variable definitions from different control-flow paths
- Many analyses and optimizations are simplified
- HercuLeS supports minimal SSA (in the number of $\phi$s) and pseudo-SSA algorithms
  - Using a scan-based algorithm
    - $\phi$-functions for each variable in each basic block (BB)
    - Iterative removal of redundant $\phi$s
  - Using pseudo-SSA form (kind of intrablock SSA)
    - No $\phi$-insertion
    - Restore the unversioned variable names prior to the exit of each BB

Nikolaos Kavvadias nkavvadias@ajaxcompilers.com

The HercuLeS HLS environment
SSA construction example

Prior SSA

\[
i = 123 \\
j = i \times j \\
PRINT(j) \\
t = j > 5 \\
i = i + 1 \\
t = i \leq 234
\]

Minimal SSA

\[
i_0 = \\
\quad j_0 = i_0 \times j \\
i_1 = i_0 \\
\quad j_1 = i_1 \times j_0 \\
\quad \text{PRINT}(j_1) \\
i_4 = i_2 + 1 \\
t_0 = j_1 > 5 \\
i_2 = \phi(i_1, i_4) \\
\quad \text{PRINT}(j_1) \\
\quad t_1 = i_4 \leq 234 \\
i_7 = \phi(i_4, i_2)
\]

Pseudo SSA

\[
i_{\$1} = 123 \\
\quad j_{\$1} = i_{\$1} \times j \\
i = i_{\$1} + 1 \\
\quad \text{PRINT}(j) \\
i_{\$1} = i + 1 \\
t_{0\$1} = j > 5 \\
\quad i = i_{\$1} + 1 \\
i_{\$1} = i_{\$1} + 1 \\
t_{1\$1} = i \leq 234 \\
i_{\$1} = i + 1 \\
t_{1\$1} = i \leq 234
\]

Nikolaos Kavvadias nkavvadias@ajaxcompilers.com

The HercuLeS HLS environment
SSA destruction example

BB1: $i1 = 123; j1 = i1 \times j0$

prevbb = 1; goto BB2;

BB2: switch (prevbb) {
    case 1: $i2 = i1; break$
    case 5: $i2 = i4; break$
    default: break;
}

printf("j1 = %08x\n", j1);
if (j1 > 5) t0$3 = 1;
else t0$3 = 0;
prevbb = 2;
if (t0$3 == 1) {goto BB3;}
else {goto BB4;}

BB3: $i4 = i2 + 1$

prevbb = 3; goto BB5;

BB4: prevbb = 4; goto BB6;

BB5: if (i$4 <= 234) t1$6 = 1;
else t1$6 = 0;
prevbb = 5;
if (t1$6 == 1) {goto BB2;}
else {goto BB6;}

BB6: switch (prevbb) {
    case 4: $i7 = i2; break$
    case 5: $i7 = i4; break$
    default: break;
}

C code for keeping SSA

Nikolaos Kavvadias nkavvadias@ajaxcompilers.com

The HercuLeS HLS environment
Representing hardware as FSMDs

FSMD (Finite-State Machine with Datapath) as a MoC is universal, well-defined and suitable for either data- or control-dominated applications

FSMDs = FSMs with embedded datapath actions

HercuLeS supports extended FSMDs (hierarchical calls, communication with on-chip memories, IP integration)

<table>
<thead>
<tr>
<th>Port</th>
<th>Dir.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>clk</td>
<td>I</td>
<td>external clocking source</td>
</tr>
<tr>
<td>reset</td>
<td>I</td>
<td>asynchronous (or synchronous) reset</td>
</tr>
<tr>
<td>start</td>
<td>I</td>
<td>enable computation</td>
</tr>
<tr>
<td>din</td>
<td>I</td>
<td>data inputs</td>
</tr>
<tr>
<td>dout</td>
<td>O</td>
<td>data outputs</td>
</tr>
<tr>
<td>ready</td>
<td>O</td>
<td>the block is ready to accept new input</td>
</tr>
<tr>
<td>valid</td>
<td>O</td>
<td>a data output port is streamed out</td>
</tr>
<tr>
<td>done</td>
<td>O</td>
<td>end of computation for the block</td>
</tr>
</tbody>
</table>
Representing hardware as FSMDs (Finite-State Machine with Datapaths)

- FSMD as a MoC is universal, well-defined and suitable for either data- or control-dominated applications
- HercuLeS generates extended FSMD architectures
- FSMDs are FSMs with embedded datapath actions within the next state generation logic
- HercuLeS FSMDs use fully-synchronous conventions and register all their outputs
- Array data ports are supported; multi-dimensional data ports are feasible based on their equivalent single-dimensional flattened array type definition
- Use of `din: in std_logic_vector(M*N-1 downto 0);` for `M` related ports of width `N`
- A selection of the form `din((i+1)*N-1 downto i*N)` is typical for a `for-generate` loop in order to synthesize iterative structures
Basic FSMD I/O interface

- **clk**: signal from external clocking source
- **reset**: asynchronous (or synchronous) reset
- **start**: enable computation
- **din**: data inputs
- **dout**: data outputs
- **ready**: the block is ready to accept new input
- **valid**: asserted when a certain data output port is streamed-out from the block (generally it is a vector)
- **done**: end of computation for the block
Hierarchical calls between FSMDs

- Supported to arbitrary depth and complexity (apart from recursion)
- Example of a caller FSMD, handing over computation to callee superstate (a square root computation)
- Variable-related quantities are represented by three signals: 
  * _next (value to-be-written), 
  * _reg (value currently read from register), 
  * _eval (callee output)
**load** Requires a wait-state register for devising a dual-cycle substate (address + data cycles)

**store** Raises block RAM write. Stored data are made available in the subsequent machine cycle.
Communication with embedded memories

- Only two memory communication primitives are needed in NAC: load and store

load Requires a wait-state register to devise a dual-cycle substate (address + data cycles)

store Raises BRAM write. Stored data are made available in the subsequent machine cycle

```
when STATE_1 =>
    mem_addr <= index;
    wstate_next <= not (wstate_reg);
    if (wstate_reg = '1') then
       mysignal_next <= mem_dout;
        next_state <= STATE_2;
    else
        next_state <= STATE_1;
    end if;
when STATE_2 =>
    ...
```

```
when STATE_1 =>
    mem_we <= '1';
    mem_addr <= index;
    mem_din <= mysignal_reg;
    next_state <= STATE_2;
when STATE_2 =>
    ...
```

load example

store example
Operation chaining

- Assign multiple data-dependent operations to a single control step
- Simple means for selective operation chaining involve merging ASAP states to compound states
- Intermediate registers are eliminated
- Basic block partitioning heuristic for critical path reduction

```plaintext
when S_1_3 =>
  t3_next <= "000"&x_reg(15 downto 3);
  t4_next <= "0"&y_reg(15 downto 1);
  next_state <= S_1_4;
when S_1_4 =>
  t5_next <= x_reg - t3_reg;
  next_state <= S_1_5;
when S_1_5 =>
  t6_next <= t4_reg + t5_reg;
  next_state <= S_1_6;
when S_1_7 =>
  out1_next <= t7_reg;
  next_state <= S_EXIT;
when S_1_1 =>
  ...
  t3_next <= "000"&x_next(15 downto 3);
  t4_next <= "0"&y_next(15 downto 1);
  t5_next <= x_next - t3_next;
  t6_next <= t4_next + t5_next;
  ...
  out1_next <= t7_next;
  next_state <= S_EXIT;
```
Automatic IP integration

IPs  Third-party components used in hardware systems (e.g. dividers, floating-point operators)

- How to import and use your own IP
  1. Implement IP and place in proper subdirectory
  2. Add entry in text database
  3. Replace operator uses by black-box function calls
  4. HercuLeS creates a hierarchical FSMD with the requested callee(s)
The optimization space of HercuLeS

- **source language**
  - N-Address Code
  - Graphviz CDFGs
  - VHDL

- **level**

- **Restructuring**
  - SSA construction
  - if-conversion
  - peephole transformations
  - basic block partitioning
  - black box function insertion
  - GNU MP extensions

- **Loop-oriented**
  - partial loop unrolling
  - full loop unrolling
  - loop coalescing
  - strip mining

- **Arithmetic**
  - constant multiplication optimization
  - constant division optimization
  - optimization of polynomial expressions
  - low-level superoptimizations

- **Array flattening**
  - sequential scheduling
  - ASAP scheduling
  - dead node elimination

- **Hardware interpretation of SSA form**
  - synchronous embedded memories (block RAMs)
  - operation chaining

Nikolaos Kavvadias nkavvadias@ajaxcompilers.com

The HercuLeS HLS environment
Optimizations

- Optimizers are implemented as external modules
- ANSI/ISO C optimizations
  - Basic loop optimizations: strip mining, loop coalescing, partial and full loop unrolling
  - Syntactical transformations among iteration schemes
  - Arithmetic optimizations
    - Constant multiplication/division/modulo
    - Optimization of linear systems
    - Univariate polynomial evaluation: Horner scheme, Estrin scheme
    - Multivariate polynomial evaluation: parallelized, optimized, brute-force variants
- NAC optimizations
  - Single constant multiplication/division; peephole optimizations; use of superoptimized operation sequences
- Graphviz/CDFG optimizations, e.g. redundancy removal
- VHDL optimizations, e.g. operation chaining
Optimizations at the C level

- Source-level optimizer targets speed improvement via generic code restructuring and loop-specific optimizations
- Implemented as TXL transformations

<table>
<thead>
<tr>
<th>Transformation</th>
<th>Description</th>
<th>Params</th>
</tr>
</thead>
<tbody>
<tr>
<td>bump</td>
<td>Alter loop boundaries by an offset</td>
<td>offset, step</td>
</tr>
<tr>
<td>extension</td>
<td>Extend loop boundaries</td>
<td>lo, hi</td>
</tr>
<tr>
<td>reduction</td>
<td>Reverse the effect of extension</td>
<td>–</td>
</tr>
<tr>
<td>reversal</td>
<td>Reverse iteration direction</td>
<td>–</td>
</tr>
<tr>
<td>normalization</td>
<td>Convert arbitrary to well-behaved loops</td>
<td>–</td>
</tr>
<tr>
<td>fusion</td>
<td>Merges bodies of successive loops</td>
<td>–</td>
</tr>
<tr>
<td>coalescing</td>
<td>Nested loops into single loop</td>
<td>–</td>
</tr>
<tr>
<td>unswitching</td>
<td>Move invariant control code</td>
<td>–</td>
</tr>
<tr>
<td>strip mining</td>
<td>Single loop tiling</td>
<td>tilesize</td>
</tr>
</tbody>
</table>
Optimizations at the NAC level

- Examples include if-conversion and arithmetic optimizations (constant multiplication and division)
- Integer constant division: replace a variable divider by custom circuit
- Use multiplicative inverse followed by a number of compensation steps
- Requires 2-9 states (amortized maximum)
- Can be further optimized by operation chaining and replacing the 64-bit (long long) constant multiply
Example: Prime factorization (1/4)

- A streaming-output implementation of prime factorization

```
void pfactor(unsigned int x, unsigned int *outp)
{
    unsigned int i=2, n=x;
    while (i <= n) {
        while ((n % i) == 0) {
            n = n / i;
            *outp = i;
        }
        i = i + 1;
    }
}
```

ANSI C

```
procedure pfactor(in u32 x, out u32 outp) {
    localvar u32 D_1369, i, n;
    L0005:
        n <= mov x;
        i <= ldc 2;
        D_1366 <= jmpun;
    D_1365:
        D_1363 <= jmpun;
    D_1362:
        (n) <= divu(n, i);
        outp <= mov i;
        D_1363 <= jmpun;
    D_1363:
        (D_1369) <= modu(n, i);
        D_1362, D_1364 <= jmpeq D_1369, 0;
    D_1364:
        i <= add i, 1;
        D_1366 <= jmpun;
    D_1366:
        D_1365, D_1367 <= jmple i, n;
    D_1367:
        nop;
}
```

NAC

Nikolaos Kavvadias nkavvadias@ajaxcompilers.com

The HercuLeS HLS environment
Example: Prime factorization (2/4)
Example: Prime factorization (3/4)

```vhdl
when S_002_001 =>
  modu_10_start <= '1';
  next_state <= S_004_001;
when S_003_001 =>
  if (divu_6_ready='1' and
      divu_6_start='0') then
    n_1_next <= n_1_eval;
    next_state <= S_003_002;
  else
    next_state <= S_003_001;
  end if;

when S_004_001 =>
  if (modu_10_ready='1' and
      modu_10_start='0') then
    D_1369_1_next <= D_1369_1_eval;
    next_state <= S_004_002;
  else
    next_state <= S_004_001;
  end if;
when S_004_002 =>
  if (D_1369_1_reg = CNST_0) then
    divu_6_start <= '1';
    next_state <= S_003_001;
  else
    next_state <= S_005_001;
  end if;
```

### VHDL (cont.)

```vhdl
divu_6 : entity WORK.divu(fsm)
  generic map (W => 32)
  port map (clk, reset, divu_6_start, n_reg, i_reg, n_1_eval, divu_6_done, divu_6_ready);

modu_10 : entity WORK.modu(fsm)
  generic map (W => 32)
  port map (clk, reset, modu_10_start, n_reg, i_reg, D_1369_1_eval, modu_10_done, modu_10_ready);
```

Nikolaos Kavvadias nkavvadias@ajaxcompilers.com

The HercuLeS HLS environment
Example: Prime factorization (4/4)

00000004 00000002 00000002
00000005 00000005
00000006 00000002 00000003
00000007 00000007
00000008 00000002 00000002 00000002
00000009 00000003 00000003
0000000A 00000002 00000005

Input vector data

x=00000004 outp=00000002 outp_ref=00000002
x=00000004 outp=00000002 outp_ref=00000002

PFACTOR OK: Number of cycles=212

x=00000005 outp=00000005 outp_ref=00000005

PFACTOR OK: Number of cycles=265

x=00000006 outp=00000002 outp_ref=00000002
x=00000006 outp=00000003 outp_ref=00000003

PFACTOR OK: Number of cycles=256

x=00000007 outp=00000007 outp_ref=00000007

PFACTOR OK: Number of cycles=353

x=00000008 outp=00000002 outp_ref=00000002
x=00000008 outp=00000002 outp_ref=00000002
x=00000008 outp=00000002 outp_ref=00000002

PFACTOR OK: Number of cycles=291

x=00000009 outp=00000003 outp_ref=00000003
x=00000009 outp=00000003 outp_ref=00000003

PFACTOR OK: Number of cycles=256

x=0000000A outp=00000002 outp_ref=00000002
x=0000000A outp=00000005 outp_ref=00000005

Diagnostic output
Example: Multi-function CORDIC

- Universal multi-function CORDIC IP core automatically generated from NAC
- Supports all directions (ROTATION, VECTORING) and modes (CIRCULAR, LINEAR, HYPERBOLIC); uses Q2.14 fixed-point arithmetic
- I/O interface similar to Xilinx CORDIC (xin, yin, zin; xout, yout, zout; dir, mode)
- Computes \( \cos(x_{in}) \), \( \sin(y_{in}) \), \( \arctan(y_{in}/x_{in}) \), \( y_{in}/x_{in} \), \( \sqrt{w} \), \( 1/\sqrt{w} \) with \( x_{in} = w + 1/4, y_{in} = w - 1/4 \) (in two stages: a. \( y = 1/w \), b. \( z = \sqrt{y} \))
- Monolithic design, low area, requires external scaling (e.g. for the square root)
- Self-checking testbench autogenerated
- Scheduler: ASAP + chaining w/o BB partitioning
- Lines-of-code: C = 29, NAC = 56, Graphviz = 178, VHDL = 436

<table>
<thead>
<tr>
<th>Design</th>
<th>Description</th>
<th>Max. freq.</th>
<th>Area (LUTs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>cordic1cyc</td>
<td>1-cycle/iteration; uses asynchronous read LUT RAM</td>
<td>204.5</td>
<td>741</td>
</tr>
<tr>
<td>cordic5cyc</td>
<td>5-cycles/iteration; uses synchronous read (Block) RAM</td>
<td>271.5</td>
<td>571, 1 BRAM</td>
</tr>
</tbody>
</table>
void cordicopt(dir, mode, xin, yin, zin, *xout, *yout, *zout) {
    ...
    x = xin; y = yin; z = zin;
    offset = ((mode == HYPER) ? 0 : ((mode == LIN) ? 14 : 28));
    kfinal = ((mode != HYPER) ? CNTAB : CNTAB+1);
    for (k = 0; k < kfinal; k++) {
        d = ((dir == ROTN) ? ((z>=0) ? 0 : 1) : ((y<0) ? 0 : 1));
        kk = ((mode != HYPER) ? k :
            cordic_hyp_steps[k]);
        xbyk = (x>>kk);
        ybyk = ((mode == HYPER) ? -(y>>kk) : ((mode == LIN) ? 0 :
            (y>>kk)));
        tabval = cordic_tab[kk+offset];
        x1 = x - ybyk; x2 = x + ybyk;
        y1 = y + xbyk; y2 = y - xbyk;
        z1 = z - tabval; z2 = z + tabval;
        x = ((d == 0) ? x1 : x2);
        y = ((d == 0) ? y1 : y2);
        z = ((d == 0) ? z1 : z2);
    }
    *xout = x; *yout = y; *zout = z;}

Nikolaos Kavvadias nkavvadias@ajaxcompilers.com

The HercuLeS HLS environment
architecture fsmd of cordicopt is
  type state_type is (S_ENTRY, S_EXIT, S_001, S_002, S_003, S_004);
  signal current_state, next_state: state_type;
  -- Scalar, vector signals and constants
...
begin
  process (*)
  begin
    ...
      case current_state is ...
      when S_003 =>
        t1_next <= cordic_hyp_steps(to_integer(unsigned(k_reg(3 downto 0))));
        if (lmode_reg /= CNST_2(15 downto 0)) then
          kk_next <= k_reg(15 downto 0);
        else
          kk_next <= t1_next(15 downto 0);
        end if;
        t2_next <= shrv4(y_reg, kk_next, '1');
      ...
      x1_next <= slv(signed(x_reg) - signed(ybyk_next(15 downto 0)));
      y1_next <= slv(signed(y_reg) + signed(xbyk_next(15 downto 0)));
      z1_next <= slv(signed(z_reg) - signed(tabval_next(15 downto 0)));
      ...
    end process;
    zout <= zout_reg;
    yout <= yout_reg;
    xout <= xout_reg;
  end fsmd;

Nikolaos Kavvadias nkavvadias@ajaxcompilers.com The HercuLeS HLS environment
Floating-point example: \textit{lerp} – linear interpolation (1/2)

- \textit{lerp}(t, a, b) = a \times (1. - t) + b \times t = a + t(b - a) calculates a number between two numbers \(a, b\) at a specific relative increment \(0.0 \leq t \leq 1.0\)
- Used in computer graphics for drawing dotting lines and in Perlin noise functions (for terrain generation)

\begin{verbatim}
#define LERP(t,a,b) (a+t*(b-a))
double lerp(double t, double x, double y) {
    double temp;
    temp = LERP(t, x, y);
    return (temp);
}
\end{verbatim}

\textbf{ANSI C}

\begin{verbatim}
procedure lerp(in f1.11.52 t,
in f1.11.52 x,
in f1.11.52 y,
out f1.11.52 D_1365) {
    localvar f1.11.52 D_1363;
    localvar f1.11.52 D_1364;
    localvar f1.11.52 temp;
    L0005:
    D_1363 <= sub y,x;
    D_1364 <= mul D_1363,t;
    temp <= add D_1364,x;
    D_1365 <= mov temp;
}
\end{verbatim}

\textbf{NAC IR}

Nikolaos Kavvadias nkavvadias@ajaxcompilers.com  The HercuLeS HLS environment
Floating-point example: *lerp* – linear interpolation (2/2)

```vhdl
entity lerp is
    port (
        clk : in std_logic;
        reset : in std_logic;
        start : in std_logic;
        t : in float64;
        x : in float64;
        y : in float64;
        D_1365 : out float64;
        done : out std_logic;
        ready : out std_logic
    );
end lerp;
```

**VHDL I/O interface**

```vhdl
when S_001_001 =>
    D_1363_next <= subtract(y, x);
    next_state <= S_001_002;
when S_001_002 =>
    D_1364_next <= multiply(D_1363_reg, t);
    next_state <= S_001_003;
when S_001_003 =>
    temp_next <= add(D_1364_reg, x);
    next_state <= S_001_004;
when S_001_004 =>
    D_1365_next <= temp_reg(11 downto -52);
    next_state <= S_EXIT;
```

**VHDL FSMD excerpt**
Explore different choices both in frontend translation (e.g. SSA construction) and hardware optimization

Numerous configurations are possible

The following configuration sets will be used

- **O1**: sequential scheduling
- **O2**: ASAP scheduling using SSA
- **O3**: O2 with operation chaining (collapsing dependent operations to a single state)
- **O4**: O3 with pseudo-SSA construction
- **O5**: O3 with preserving $\phi$ functions
- **O6**: O3 with block RAM inference
Fibonacci series example: Introduction

- Fibonacci series computation is defined as

\[
F(n) = \begin{cases} 
0 & n = 0 \\ 
1 & n = 1 \\ 
F(n - 1) + F(n - 2) & n > 1 
\end{cases}
\]

- Three iterative variants
  - A Addition and subtraction in the main loop
  - B Addition with one more temporary
  - C Addition with an in-situ register swap

```c
uint32 fibo(uint32 x) {
    uint32 f0=0, f1=1, k=2;
    #ifdef B
        uint32 f;
        #endif
    do {
        k = k + 1;
        #if A
            f1 = f1 + f0;
            f0 = f1 - f0;
        #elif B
            f = f1 + f0;
            f0 = f1;
            f1 = f;
        #elif C
            f0 = f1 + f0;
            SWAP(f0, f1);
        #endif
    } while (k <= x);
    #if A || B
        return (f1);
    #else
        return (f0);
    #endif
}
```

C code

Nikolaos Kavvadias nkavvadias@ajaxcompilers.com

The HercuLeS HLS environment
Fibonacci series example: Machine cycles

<table>
<thead>
<tr>
<th>Design</th>
<th>Cycles</th>
<th>Design</th>
<th>Cycles</th>
<th>Design</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>A0</td>
<td>$n$</td>
<td>B0</td>
<td>$n$</td>
<td>C0</td>
<td>$2n - 1$</td>
</tr>
<tr>
<td>A1</td>
<td>$4n + 3$</td>
<td>B1</td>
<td>$5n + 2$</td>
<td>C1</td>
<td>$7n + 1$</td>
</tr>
<tr>
<td>A2</td>
<td>$4n + 2$</td>
<td>B2</td>
<td>$4n + 2$</td>
<td>C2</td>
<td>$7n$</td>
</tr>
<tr>
<td>A3-A5</td>
<td>$n + 2$</td>
<td>B3-B5</td>
<td>$n + 2$</td>
<td>C3-C5</td>
<td>$2(n + 1)$</td>
</tr>
</tbody>
</table>

- Hand-optimized designs are A0, B0, C0
- The benefit of ASAP is not significant due to the data dependencies in the algorithm
- Cycle reduction is achieved through operation chaining
- HercuLeS can closely match the result of a human expert for optimization schemes O3-O5
- The slight differences in cycle performance are due to specific design choices of the human expert
  - initializing $f_0$, $f_1$ and $k$ in the FSMD entry state
  - passing the output data argument without use of an intermediate register

Nikolaos Kavvadias nkavvadias@ajaxcompilers.com

The HercuLeS HLS environment
Fibonacci series example: Execution time vs LUTs

- Better results are placed near the bottom-left corner
- Pareto-optimal designs by human expert: B0 and C0
- Pareto-optimal designs by HercuLeS: B2, B3, B4
Fibonacci series example: Execution time vs Registers

- Better results are placed near the bottom-left corner
- Pareto-optimal designs by human expert: A0
- Pareto-optimal designs by HercuLeS: A1 and B3
Benchmark results: Speed

- Average computation time is reduced by 44.3% when comparing O1 to O3
- This gain is limited to 37.3% for O6 due to block RAM timing
- float2half and half2float achieve up to 4× execution time reduction
- Maximum operating frequencies in the range of 119-450MHz
Benchmark results: Area

- O1 generates (slower and) smaller hardware in terms of LUTs and registers
- O3 introduces the highest LUT requirements; O2 the highest register demand
- Registers are reduced by 17.5% among O2 and O6; LUTs by 14% among O3 and O6
- Block RAM inference leads to significant LUT/register area reduction (*smwat*)

Nikolaos Kavvadias nkavvadias@ajaxcompilers.com
The HercuLeS HLS environment
### Example algorithmic benchmarks

<table>
<thead>
<tr>
<th>Bench.</th>
<th>Type</th>
<th>Description</th>
<th>C LOC</th>
<th>NAC</th>
<th>dot</th>
<th>VHDL</th>
</tr>
</thead>
<tbody>
<tr>
<td>edgedet</td>
<td>M</td>
<td>Edge detection</td>
<td>35</td>
<td>145</td>
<td>873</td>
<td>1921</td>
</tr>
<tr>
<td>float2half</td>
<td>C</td>
<td>Convert float-to-half</td>
<td>25</td>
<td>71</td>
<td>157</td>
<td>370</td>
</tr>
<tr>
<td>fsme</td>
<td>M</td>
<td>Motion estimation</td>
<td>65</td>
<td>159</td>
<td>1483</td>
<td>2730</td>
</tr>
<tr>
<td>half2float</td>
<td>C</td>
<td>Convert half-to-float</td>
<td>12</td>
<td>32</td>
<td>55</td>
<td>174</td>
</tr>
<tr>
<td>icbrt</td>
<td>C</td>
<td>Integer cubic root</td>
<td>18</td>
<td>36</td>
<td>83</td>
<td>213</td>
</tr>
<tr>
<td>isqrt</td>
<td>C</td>
<td>Integer square root</td>
<td>20</td>
<td>28</td>
<td>84</td>
<td>199</td>
</tr>
<tr>
<td>mandel</td>
<td>C/M</td>
<td>Mandelbrot fractal</td>
<td>60</td>
<td>108</td>
<td>259</td>
<td>639</td>
</tr>
<tr>
<td>matmult</td>
<td>M</td>
<td>Matrix mult.</td>
<td>40</td>
<td>94</td>
<td>763</td>
<td>1511</td>
</tr>
<tr>
<td>sierpinski</td>
<td>C/M</td>
<td>Sierpinski triangle</td>
<td>51</td>
<td>70</td>
<td>300</td>
<td>630</td>
</tr>
<tr>
<td>smwat</td>
<td>M</td>
<td>Smith-Waterman kernel</td>
<td>68</td>
<td>159</td>
<td>753</td>
<td>1615</td>
</tr>
<tr>
<td>walsh</td>
<td>M</td>
<td>2D Walsh transform</td>
<td>32</td>
<td>71</td>
<td>326</td>
<td>704</td>
</tr>
<tr>
<td>yuv2rgba</td>
<td>C</td>
<td>Color space conv.</td>
<td>27</td>
<td>98</td>
<td>240</td>
<td>679</td>
</tr>
</tbody>
</table>
Against competition

- New frontends, analyses and optimizations are easy to add
- Maps character I/O and \texttt{malloc/free} to efficient hardware
- Uses open IRs and formats (GIMPLE, NAC, Graphviz)
- Vendor and technology-independent HDL code generation
- Preliminary results against Vivado HLS 2013.1

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Vivado HLS</th>
<th></th>
<th>HercuLeS</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LUTs</td>
<td>Regs</td>
<td>Time (ns)</td>
<td>LUTs</td>
</tr>
<tr>
<td>Array sum</td>
<td>102</td>
<td>132</td>
<td>26.5</td>
<td>103</td>
</tr>
<tr>
<td>Bit reversal</td>
<td>67</td>
<td>39</td>
<td>72.0</td>
<td>42</td>
</tr>
<tr>
<td>Edge detection*</td>
<td>246</td>
<td>130</td>
<td>1636.3</td>
<td>680</td>
</tr>
<tr>
<td>Fibonacci series</td>
<td>138</td>
<td>131</td>
<td>60.2</td>
<td>137</td>
</tr>
<tr>
<td>FIR filter</td>
<td>102</td>
<td>52</td>
<td>833.4</td>
<td>217</td>
</tr>
<tr>
<td>Greatest common divisor</td>
<td>210</td>
<td>98</td>
<td>35.2</td>
<td>128</td>
</tr>
<tr>
<td>Cubic root approx.</td>
<td>239</td>
<td>207</td>
<td>260.6</td>
<td>365</td>
</tr>
<tr>
<td>Population count</td>
<td>45</td>
<td>65</td>
<td>19.4</td>
<td>53</td>
</tr>
<tr>
<td>Prime sieve*</td>
<td>525</td>
<td>595</td>
<td>6108.4</td>
<td>565</td>
</tr>
<tr>
<td>Sierpinski triangle</td>
<td>88</td>
<td>163</td>
<td>11326.5</td>
<td>230</td>
</tr>
</tbody>
</table>
The HercuLeS GUI

- Bundled with HercuLeS v1.0.0 (2013a) released on June 30
- Specify code generation, simulation and synthesis options
- Includes embedded results viewer

Nikolaos Kavvadias nkavvadias@ajaxcompilers.com

The HercuLeS HLS environment
HercuLeS is an extensible HLS environment for hardware and software engineers

- Straightforward to use from ANSI C, generic assembly (NAC) or custom DSLs (Domain Specific Languages) to producing compact VHDL designs with competitive QoR

Product information
- HercuLeS GUI for specifying code generation, simulation and synthesis options
- Commercial distribution: http://www.ajaxcompilers.com
- Technical details: http://www.nkavvadias.com/hercules
- FREE, BASIC, and FULL software licensing schemes (2013.a version)
fibo.c: Iterative computation of the Fibonacci series

- Setup the environment: source env-hercules-lin.sh
- Inspect input ANSI C code: fibo function, main() used for data vector generation
- Compile C code; inspect fibo_test_data.txt
- Use the run-fibo.sh script
- After the simulation ends
  - End-upon-assertion
  - Inspect diagnostic output fibo_alg_test_results.txt
  - Inspect generated NAC and Graphviz CDFG
  - Visualize CFG and CDFG with gwenview
  - Inspect generated VHDL code: code size, naming conventions, readability of code
  - Inspect waveform data (fibo_fsmd.ghw)
- Open discussion (e.g. generation settings, more optimizations, comparison to manual design, etc)
easter.c: Easter Sunday calculations

- Easter Sunday calculations based on Gauss’ algorithm
- No need to have any knowledge of the algorithm!
- Compile C code; inspect easter_test_data.txt
- Use the run-easter.sh script
- After the simulation ends
  - Inspect generated NAC
  - Inspect generated VHDL code
  - Automatic IP integration for modulo
  - Automatic optimization of constant multiplications
  - Inspect results; did we get last year’s Easter date right?
- Open discussion
**fir.c: Finite Impulse Response filter**

- Small FIR example
- Reads data from single-dimensional arrays
- Compile C code; inspect `fir_test_data.txt`
- Use the `run-fir.sh` script
- After the simulation ends
  - Inspect generated VHDL code: array access not to block RAM
  - Alter HercuLeS generation script (`run-fir.sh`) to enable block RAM inference (`-blockmem -read-first`)
  - Reiterate
- Open discussion
FPGA demo 1: PLOTLINE IP

- Demo IP that will be offered for free from Ajax Compilers web store
- Synthesizable RTL model of Bresenham’s line drawing algorithm
- Integer version of the algorithm using addition, subtraction, bit shifting, comparison, absolute value and conditional moves
- Consists of a single loop that keeps track of the propagated error over the x- and y-axis, which is then used for calculating the subsequent pixel on the line
- The loop terminates when the end point is reached
- Invoke IMPACT and upload plotline_system.bit to the Spartan-3AN Starter Kit board
- Dimensions: 320x240, upscaled by x2 to VGA
FPGA demo 2: Pixel generation function

- Maps the masked value of $f(x,y)$ to the corresponding pixel without using any memory
- Generates either grey (same function) or colored (different function per chromatic component) images

$$f(x,y) = \begin{cases} 
R & x, & G & y, & B & x \oplus y, & \text{mode} == 0 \\
R & x, & G & x + y, & B & x + y, & \text{mode} == 1 \\
R & x, & G & x \times y, & B & x \times y, & \text{mode} == 2 \\
R & x, & G & y, & B & 0, & \text{mode} == 3 \\
R & x^2 + y^2, & G & x^2 + y^2, & B & x^2 + y^2, & \text{mode} == 4 \\
R & x + y, & G & x - y, & B & x \oplus y, & \text{mode} == 5 \\
R & x^2 - y^2, & G & x^2 - y^2, & B & x^2 - y^2, & \text{mode} == 6 \\
R & \max(x,y), & G & \max(x,y), & B & \max(x,y), & \text{mode} == 7 
\end{cases}$$
Thank you

for your interest