High Level Synthesis techniques

Laurent Hili
ESA-ESTEC
19/09/2011
What are the challenges?

- Everything tends to become more complex (Moore’s Law)
- Miniaturisation thanks to CMOS technologies (65, 45, 28 and 22nm) offers possibilities to design chips exhibiting a complexity beyond a billion of transistors
- Systems on board(s) tend to move to systems on chip (SoC)
- Possibility to integrate various technologies on the same chip (SW, HW digital, HW analog, MEMs, sensors)
What are the challenges?

- Necessity to have higher abstraction languages to face the new challenges raised by tighter HW/SW integration and trade-offs.

- Necessity to gain in productivity in order to handle the ever growing complexity while using the same number of designers (or even less).

- Necessity to put in place advanced CAD techniques to enable the productivity gains.
  - Modeling: Transactions Level Modeling (TLM) & Transaction Based Verification (TBV).
  - CAD tools: High level synthesis / Virtual Platform.
Productivity through abstraction

**Productivity**

- **1970s**
  - **Granularity**
    - transistors
    - polygons
  - **Layout**
    - first CAD tools

- **1980s**
  - **Granularity**
    - gate level
  - **CAD**
    - netlist standardisation
    - EDIF, GDSII

- **1990s**
  - **Granularity**
    - cycle, bit, pin accurate
    - registers, logic, operators
  - **CAD**
    - VHDL, Verilog
    - floor plan
    - logic synthesis

- **2000s**
  - **ESL**
    - transaction accurate
    - pure behavioral description
    - C, C++, SystemC, System Verilog
  - **High level synthesis**
  - **Virtual platform**

**RTL**

```plaintext
process (CLK, RST)
if (RST = '1') then
  Q <= '0';
elsif rising_edge (CLK) then
  Q <= A and B and C and D;
end if;
```
Benefits of High Level Synthesis (HLS)

- Higher productivity
- New IP development ~ 2 to 3 times faster than RTL (VHDL / Verilog)
- Possibility to describe an algorithm in a very concise way compared to HDL languages (~ 10 times fewer lines of code to maintain)
- The designer can better focus on the functionality rather than implementation details
- HLS can generate many RTL derivatives from same C code in a time effective manner (architecture exploration)
- HLS can easily be merged in a HW/SW co-design flow
  - Integration with virtual platform (SystemC TLM)
  - Integration with HDL flow (simulators, logic synthesis)

(ST Microelectronics source, 9th annual ESL Symposium, June 2011)
Common features in HLS tools

- Architecture exploration → ability to generate RTL variants
  - Loops optimisation: parallelisation and/or pipelining
  - Arrays optimisation: registers, RAMs, ROMs
  - Interfaces optimisation: wire, enable, handshake, bus, NoC (Network on Chip interface)

- Gantt chart analysis → ability to display concurrency and/or dependencies between resources / tasks

- Verification flow analysis → ability to run regression tests, Transaction Based Verifications (TBV) in order to check the RTL code automatically generated against original C++ code
Common features in HLS tools

- Cross probing analysis → ability to identify resources in the Gantt chart or RTL and map them to the original C++

- Interfacing with conventional ASIC / FPGA back end flows (RTL simulators and logic synthesis)

- Architecture exploration → target technology aware
  - Resources aware: operators, memories, interfaces
  - Timing aware
  - Area aware
  - Power aware (feature available as an option sometimes)
HLS solutions on the market

✓ CatapultC originally developed by Mentor Graphics has been spun off to Calypto the 26/8/2011. This tool has been one of the first used for ASICs tape out. CatapultC is in full production in Thales where 3 ASICs out of 4 are now produced using this methodology.

✓ Synphony C compiler → Synopsys

✓ C to Silicon compiler → Cadence

✓ Cynthesizer → Forte Design Systems
ESA High Level Synthesis flow

Start point executable specification
- Executable IP specification
- C / C++ / Matlab
- Fixed point or Floating point untimed

C++ function
- Golden reference
- CatapultC (Mentor)
- Algorithm implementation
- bit accurate / untimed

C++ test bench
- HDL
- Regression tests
- HDL vs C++ test bench

fixed point IP
- Matlab / Simulink
- Algorithm performance validation
- bit accurate / untimed

CatapultC (Mentor)
- Algorithm implementation
- bit accurate / untimed

Modelsim / Questasim (Mentor)
- Algorithm Implementation
- bit & time accurate

Precision RTL plus (Mentor)
- Algorithm logic synthesis / FPGA
- Netlist
- pin / bit / time accurate

ISE design suite (Xilinx)
- Netlist Placed & routed

DC Ultra
- Design Compiler Ultra (Synopsys)
- Algorithm logic synthesis / ASIC
- Netlist

PS
- Executable hardware IP
- on target technology

Optional flow
- Hardware in the loop co-simulation / co-emulation
- HAPS 60 (Synopsys)

Executable IP specification
- C / C++ / Matlab
CatapultC and Simulink integration

1. C++ algorithm coding
2. Simulink wrappers generation
3. Wrapper compilation with MEX
4. Simulink simulation (algo performances)
5. if step4 not OK then iterate in step1 otherwise goto step6
6. Generate RTL code
7. Run regression tests (SystemC verification flow / Transaction Based Verification TBV)

Step 1: C++ algorithm coding
Step 2: Simulink wrappers generation
Step 3: Wrapper compilation with MEX
Step 4: Simulink simulation (algo performances)
Step 5: if step4 not OK then iterate in step1 otherwise goto step6
Step 6: Generate RTL code
Step 7: Run regression tests (SystemC verification flow / Transaction Based Verification TBV)
FIR filter example (algorithm)

FIR filter direct implementation

\[ Y(n) = \sum_{i=0}^{N} H(i) \times X(n-i) \]

Let's assume a 8 TAPs FIR filter \( N=7 \)

\[ Y(n) = H(0).X(n) + H(1).X(n-1) + H(2).X(n-2) + H(3).X(n-3) + ... + H(7).X(n-7) \]

Let's assume \( n=7 \)

\[ Y(7) = H(0).X(7) + H(1).X(6) + H(2).X(5) + H(3).X(4) + ... + H(7).X(0) \]
**FIR filter example (source code)**

**fir.cpp**
```c
#include "fir.h"
#pragma hls_design top
void fir_filter (data_t &input, data_t coeffs[NUM_TAPS], data_t &output) {
  static data_t regs[NUM_TAPS];
  accu_t temp = 0;
  SHIFT:
  for (int i = NUM_TAPS-1; i >= 0; i--) {
    if (i == 0)
      regs[i] = input;
    else
      regs[i] = regs[i-1];
  }
  MAC:
  for (int i = NUM_TAPS-1; i >= 0; i--)
    temp += coeffs[i] * regs[i];
  output = temp;
}
```

**fir.h**
```c
#ifndef _FIR_FILTER_H
#define _FIR_FILTER_H
#include <ac_fixed.h>
#define NUM_TAPS 8
typedef ac_fixed<18, 2, 1, AC_TRN, AC_WRAP>  data_t;
typedef ac_fixed<24, 8, 1, AC_TRN, AC_WRAP>  accu_t;
void fir_filter (data_t &input, data_t coeffs[NUM_TAPS], data_t &output);
#define _FIR_FILTER_H
#endif
```

- **Pragmas** identifying the top or code to be synthesised
- **Loops** must be labelled
- **Fixed point types**
- **8 coefficients**
- **8 registers 18 bits**
FIR filter example (test bench)

```
#include <iostream>
#include "mc_scverify.h"
#include "fir.h"

// some functions for generating random test vectors
......

// Start of MAIN

CCS_MAIN(int argc, char *argv)
{
    // Place local testbench variables here
    ac_fixed<18, 2, true, AC_TRN, AC_WRAP> input;
    data_t coeffs[8];
    ac_fixed<18, 2, true, AC_TRN, AC_WRAP> output;

    // Initialize local variables to zero
    init_input(input);
    init_coeffs(coeffs);
    init_output(output);

    // Main test iterations start here
    for (int iteration = 0; iteration < 100; ++iteration) {
        // Set test values for this iteration
        throw_dice_for_input(input);
        throw_dice_for_coeffs(coeffs);

        // Call original function and capture data
        CCS_DESIGN(fir_filter)(input, coeffs, output);
    }

    // Return success
    CCS_RETURN(0);
}
```

Instructions in red are only ones differing from original C++ test bench
Those instructions are used by CatapultC synthesis

Unique test bench is used for regression tests RTL vs C++ (Transaction Based Verification)
FIR filter example (architecture constraints)
FIR filter example (architecture exploration)

<table>
<thead>
<tr>
<th>Solution</th>
<th>Latency Cycles</th>
<th>Latency Time</th>
<th>Throughput Cycles</th>
<th>Throughput Time</th>
<th>Slack</th>
<th>Total Area</th>
</tr>
</thead>
<tbody>
<tr>
<td>fir_filter.v3 (extract)</td>
<td>8</td>
<td>400.00</td>
<td>10</td>
<td>500.00</td>
<td>43.03</td>
<td>858.44</td>
</tr>
<tr>
<td>fir_filter.v5 (extract)</td>
<td>9</td>
<td>450.00</td>
<td>10</td>
<td>500.00</td>
<td>44.89</td>
<td>678.12</td>
</tr>
<tr>
<td>fir_filter.v11 (extract)</td>
<td>1</td>
<td>50.00</td>
<td></td>
<td>50.00</td>
<td>44.51</td>
<td>593.86</td>
</tr>
<tr>
<td>fir_filter.v12 (extract)</td>
<td>1</td>
<td>50.00</td>
<td></td>
<td>50.00</td>
<td>42.85</td>
<td>2966.85</td>
</tr>
</tbody>
</table>

Clock period constrained to 50ns (20 MHz)

<table>
<thead>
<tr>
<th>solution</th>
<th>MAIN loop</th>
<th>SHIFT loop</th>
<th>MAC loop</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fir_filter.v3</td>
<td>Rolled</td>
<td>Rolled</td>
<td>Rolled</td>
</tr>
<tr>
<td>Fir_filter.v5</td>
<td>Rolled</td>
<td>Unrolled</td>
<td>Rolled</td>
</tr>
<tr>
<td>Fir_filter.v11</td>
<td>Pipelined</td>
<td>Unrolled</td>
<td>Pipelined</td>
</tr>
<tr>
<td></td>
<td>II=1</td>
<td></td>
<td>II=1</td>
</tr>
<tr>
<td>Fir_filter.v12</td>
<td>Pipelined</td>
<td>Unrolled</td>
<td>Unrolled</td>
</tr>
<tr>
<td></td>
<td>II=1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Default implementation loops are rolled and if timings constraints allow loops are merged

II=1
Initial Interval = 1
1 data fed in the pipeline
Every clock cycle
FIR filter example (architecture exploration)
FIR filter example (architecture exploration)

FIR_filter.V12 is the fastest solution but also consumes more resources (direct implementation)
FIR filter example (Gantt chart analysis)

High level synthesis is technology aware (Virtex 6 in the present case). HLS tool can provide info on:
- Speed
- Area
- Power consumption (optional)

Control step 1 of the main loop. For each iteration:
- input is read
- coeff is read
- register is shifted
- MAC is performed
- output is written

MAC and SHIFT loops in this example are merged in the same control step.

10% slack to account for place & route

8 TAPs shift register each TAP = 18 bits
FIR filter example (RTL code & target technology netlist generation)

- Coeffs 8 * 18 bits
- Input data 18 bits
  - 1 data_in & 1 data_out every 8 clock cycles
- Mux 2→1 to hold input stream during accumulation (8 cycles)

**Fir_filter.V11**
- Main loop: pipelined
- Shift loop: unrolled
- MAC loop: pipelined

**Best area solution**
FIR filter example (RTL code & target technology netlist generation)

Fir_filter.V12

- Main loop: pipelined
- Shift loop: unrolled
- MAC loop: unrolled

Best timing solution
FIR filter example (implementation after synthesis & place-route)

Complete layout

MAC mapped on DSP 48 block (example based on Virtex 6)
FIR filter example (simulink validation)

8 TAPs FIR filter symbol

FIR Filter Functional Verification using Catapult’s Simulink Flow

Test bench
Integration of HLS flow with virtual platform

**Legend:**
- TLM 2.0: SystemC Transaction Level Modeling (IEEE 1666)
- TBV: Transaction Based Verification
- LT: loosely timed
- AT: approximately timed
- CA: cycle accurate

**Spec capture / analysis**

**HW/SW partitioning**

**IPs repository**
- Software IPs
- Hardware IPs
- TLM IPs
- RTL IPs

**Virtual Platform**

**HW refinement flow**
- HW architecture exploration
  TLM 2.0 AT, CA
- High Level Synthesis
- HW RTL

**RTL platform**
- RTL / ESL verification
- TBV
  Optional: HW in the loop

**SW refinement flow**
- SW development
  TLM 2.0 LT
- Compile & optimise
- SW executable

**Legend:**
- TLM 2.0: SystemC Transaction Level Modeling (IEEE 1666)
- TBV: Transaction Based Verification
- LT: loosely timed
- AT: approximately timed
- CA: cycle accurate

**Spec modifications**

**HW / SW partitioning modifications**

**HW modifications**

**SW modifications**
Virtual Platform deployment view

- **System Engineer**
  - Design optimisation,
  - Performance,
  - Power analysis

- **Hardware Engineer**
  - Generation of RTL
  - from TLM $\rightarrow$ C/C++/ SystemC

- **Software Engineer**
  - Software / Firmware optimisation

- **Verification Engineer**
  - TLM / RTL verification
  - and sign-off of HW/SW platform
Thanks for your attention

Special thanks to my colleague Jelle Poupaert and Stephane Labert (Mentor Graphics France) for their support

Any question?