Catapult C® Synthesis

High Level Synthesis Webinar

Stuart Clubb
Technical Marketing Engineer
April 2009

Mentor Graphics®

Agenda

- How can we improve productivity?
- C++ Bit-accurate datatypes and modeling
- Using C++ for hardware design
  - A reusable, programmable, variable decimator
- Synthesizing, optimizing and verifying our C++
  - Live demo

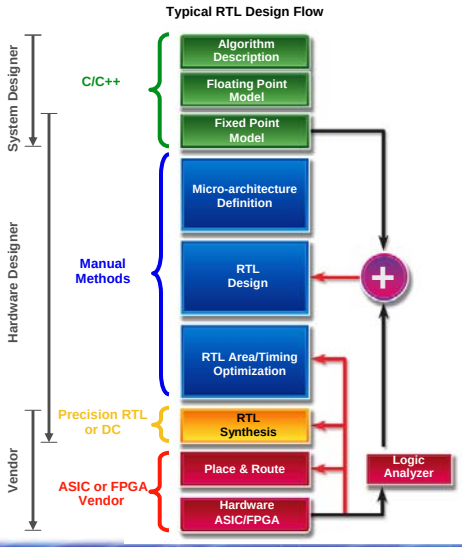
How can we improve productivity

- Designs bring ever increasing complexity
- More complex designs require more
  - Time
  - People
  - Resources
- Increase of "Gates Per Day" for RTL has stalled
  - Time to validate algorithm
  - Time to code RTL
  - Time to Verify RTL

Productivity Bottlenecks

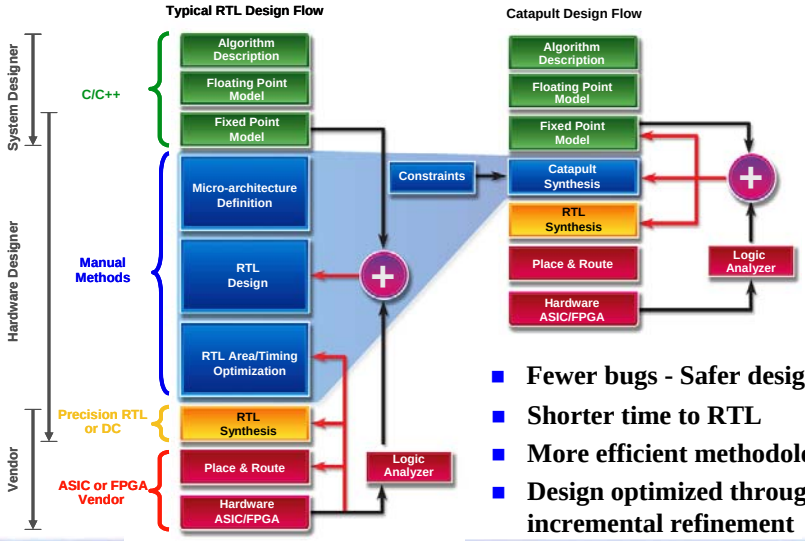
- Finding an algorithm's optimal hardware architecture and implementing it in a timely manner
- Reducing the number of bugs introduced by the RTL design process
- Verification of the RTL implementation to show that it matches the original algorithm

The RTL Flow: Past History



- **Manual Steps**
  1. Define micro-architecture
  2. Write RTL
  3. Optimize area/speed through RTL synthesis
- **Drawbacks**
  1. Disconnect causes design errors
  2. RTL hard-codes technology making re-use impractical
  3. Manual RTL coding too time-consuming leading to fewer iterations and sub-optimal designs
  4. Designs typically overbuilt

Traditional Flow vs. Catapult Flow



- Fewer bugs - Safer design flow
- Shorter time to RTL
- More efficient methodology
- Design optimized through incremental refinement

C++ Bit Accurate Data Types

- SystemC data types or Mentor Graphics Algorithmic C data types
- Hardware Designers need exact bit widths
  - Extra bits costs gates ($$) and performance ($$)
- Rounding and Saturation are important
- Simulating what you will synthesize is key
  - Simulation speed affects validation efforts

SystemC DataTypes

- Limited Length Integer and Fixed-point
  - sc_int/sc_uint – maximum 64-bit integer result
  - sc_fixed_fast/sc_ufixed_fast actually based on a double with maximum 53-bit fixed-point result
  - Problems mixing signed and unsigned
    - (sc_int<2>) -1 > (sc_uint<2>) 1 returns true!
- Arbitrary Length Integer and Fixed Point
  - Resolves most, but not all, issues of ambiguity/compatibility
  - Slow simulation with fixed-point
  - Fixed point conditionally compiled due to speed
    - SC_INCLUDE_FX

Mentor Graphics "Algorithmic C" types

- Fixed-point and Integer types
- Faster execution on same platform
  - >200x faster than SystemC types
- Easy to use, consistent, with no ambiguity
- Parameterized
  - Facilitate reusable algorithmic development
- Built in Rounding and Saturation modes
- Freely available for anyone to download

http://www.mentor.com/esl

Templatized AC Fixed Data Types

```
ac_fixed<W,I,S,Q,O> my_variable
```

- `W` = Overall Width
- `I` = Number of integer bits
- `S` = signed or unsigned (boolean)
- `Q` = Quantization mode
- `O` = Overflow mode

```
ac_fixed<8,1,true,AC_RND,AC_SAT> my_variable ;
    "0.0000000" 8-bit signed, round & saturate
ac_fixed<8,8,true,AC_TRN,AC_WRAP> my_variable ;
    "00000000" 8-bit signed, no fractional bits.
```

Using C++ for hardware design

- Function call with all I/O on the interface
  - Represents the I/O of the algorithm
- C++ object-oriented reusable hardware
  - Technology, implementation, and Fmax independent
  - Multiple instantiations of functions (objects) with state
    - RTL component instantiation
  - Instantiations with differing implementations
    - RTL VHDL architectures

A programmable variable decimator

- Programmable ratio (phases)
- Tap Length based on decimation factor and 'N'
  - x1 decimation = 1 * N taps;
  - x4 decimation = 4 * N taps
  - x8 decimation = 8 * N taps
- Seamless transitions between output rates
  - Two sets of externally programmable coefficients
  - Centered delay line access

Top Level Filter function

```
void my_filter (
    ac_channel<d_type>    &data_in,
    ratio_type            ratio,
    bool                  sel_a,
    c_type                coeffs_a[N_TAPS_1*N_PHASES_1],
    c_type                coeffs_b[N_TAPS_1*N_PHASES_1],
    ac_channel<d_type>    &data_out
) {

static decimator<ratio_type,d_type,c_type,a_type,N_TAPS_1,N_PHASES_1> filter_1 ;

filter_1.decimator_shift(data_in,ratio,sel_a,coeffs_a,coeffs_b,data_out) ;

}
```

typedef's for data types passed to class object

- Simple instantiation of templatized class
- Call member function "decimator_shift"
- Write the member function once
  - Implement a filter with any tap length, and any data types

Data types used in this example

```
#define N_TAPS_1 8
#define N_PHASES_1 8
#define LOG_PHASES_1 3

#define DATA_WIDTH 8
#define COEFF_WIDTH 10

typedef ac_fixed<DATA_WIDTH,DATA_WIDTH,true,AC_RND,AC_SAT> d_type ;
typedef ac_fixed<COEFF_WIDTH,1,true,AC_RND,AC_SAT> c_type ;
typedef ac_fixed<DATA_WIDTH+COEFF_WIDTH+7,DATA_WIDTH+7+1,true> a_type ;

// 0 to 7 rate
typedef ac_int<LOG_PHASES_1,false> ratio_type ;
```

Data type will round and saturate when written

Full Precision Accumulator
- Saturation is order dependent

3-bit unsigned for decimation ratio

- Use of AC data types for bit-accurate modeling and Synthesis ensures 100% match between RTL and C++

Class Object for FIR filter

```
template <class rType, class dType, class cType, class aType, int N_TAPS, int N_PHASES>
class decimator {
     // data members
     dType taps[N_TAPS*N_PHASES];
     aType acc;
     // member functions
public:
     decimator() { // default constructor
          for (int i=0;i<N_TAPS*N_PHASES;i++) {
               taps[i] = 0 ;
          }
     };
     void decimator_shift(
          ac_channel<dType>   &data_input,
          rType               ratio,
          bool                sel_a,
          cType               coeffs_a[N_TAPS_1*N_PHASES_1],
          cType               coeffs_b[N_TAPS_1*N_PHASES_1],
          ac_channel<dType>   &data_out
     ) ;
} ;
```

taps and accumulator are private objects

Default constructor Initializes tap registers to zero (reset)

Member function prototype

Decimator code

```
if(data_input.available(ratio+1)) {
   acc = 0 ;
   PHASE:for(int phase=0; phase<N_PHASES; phase++) {

      SHIFT:for(int z=(N_TAPS*N_PHASES-1);z>=0;z--) {
         taps[z] = (z==0) ? data_input.read() : taps[z-1] ;
      }

      MAC:for(int i=0;i<N_TAPS;i++) {
         int tap_offset = (N_PHASES * N_TAPS)/2 - ((ratio.to_int()+1)*N_TAPS/2) ;
         int tap_index = (i*(ratio.to_int()+1)) ;
         int coeff_index = tap_index + (ratio-phase) ;
         tap_index = tap_index + tap_offset ;
         cType coeff_read = (sel_a) ? coeffs_a[coeff_index] : coeffs_b[coeff_index] ;
         acc += coeff_read * taps[tap_index] ;
      }

      if (phase==ratio) {
         data_out.write(acc) ;
         break ;
      }
   }
}
```

Phase for decimation reads

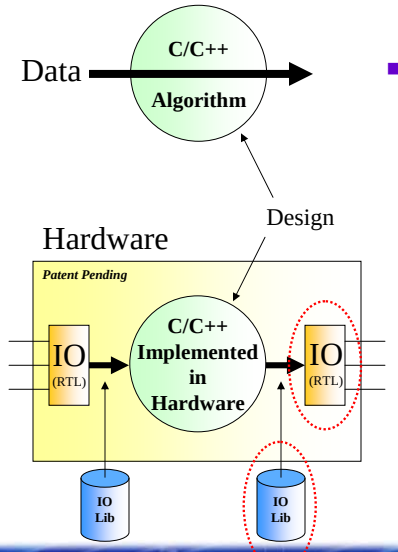
Implied shift register architecture captures data streaming in

Seamless, variable iterations using "break"

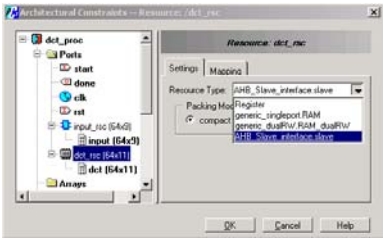
- Simple, bit-accurate, C++
- Technology independent
- Yes, that's it – design done
  - We need a testbench main()

Defining The Hardware Interface

Patented Interface synthesis makes it possible



- Pure C++ has no concept of interfaces

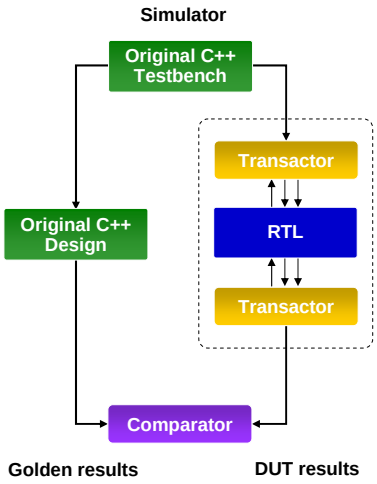


- How does this help?
  - ANY interface is possible
  - Design is built to the interface
  - C++ source remains independent of the interface

Optimizing C++ Algorithms

- Catapult maps physical resources for each variable in the C++ code
  - Wires, handshakes, registers, RAM's, custom interfaces, custom components
- Catapult builds efficient hardware optimized to the constraints of resource bandwidth
- Catapult enables you to quickly find architectural bottlenecks in an algorithm
- Datapath pipelines are created to meet desired frequency target

Verification of Catapult RTL using C++



- Catpult automates verification of the synthesized design
- The original C++ testbench can be reused to verify the design
  - RTL or Cycle Accurate
  - VHDL or Verilog
- RTL can be replaced with gate netlist for VCD driven power analysis of solutions

More productive than RTL

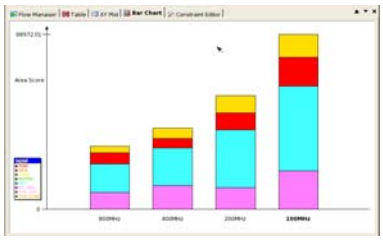
- Higher level of abstraction with considerably faster verification
- High Level Synthesis drives implementation details
  - Interfaces
  - Frequency, latency, throughput
  - All based on target technology
- Design reuse and configurability is enhanced
- Hand coded RTL designed for one technology is not always optimal for another
  - Excessive pipelining increases power and area
  - Faster technologies allow for more resource sharing at same F_{max}

Synthesizing the Decimator

- 90nm example library
- N=8 (filter is effectively 8 taps to 64 taps)
- 100M Samples maximum data rate in
- 4 micro-architectures to solve the design
  - 1, 2, 4, 8 multipliers
  - 800MHz down to 100 MHz
- Which is “right” solution?

Which is the right solution?

- Area => 800MHz



- Power => 100Mhz

Flow Manager | Table | XY Plot | Bar Chart | Constraint Editor

Report: Atrenta SpyGlassPower

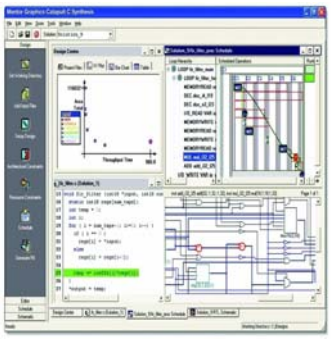
Solution	Leakage Power	Internal Power	Switching P...	Total Est ...
800MHz (extract)	81.5uW	11.6mW	39.1mW	50.8mW
400MHz (extract)	99.9uW	7.67mW	36.3mW	44.1mW
200MHz (extract)	143uW	6.49mW	45.8mW	52.4mW
100MHz (extract)	**208uW**	**4.23mW**	**24.7mW**	**29.1mW**

  - Interesting “saddle” at 400MHz

Catapult C Synthesis

The Five Key Technologies which Make Catapult C Different

- **Key: Synthesize standard ANSI C++**
  - **Not a 'hardware C' but pure ANSI C++**
  - **No proprietary extensions, universal standard, easiest to write & debug**



- **Optimization for ASIC or FPGA**
  - **Generation of technology optimized RTL**
- **Incremental design methodology**
  - **Maximum visibility, maximum control**
- **Interface synthesis**
  - **Interface exploration and optimization**
- **Integrated SystemC verification**
  - **Provides automatic verification environment**
  - **Pure ANSI C++ in, Verified RTL out**

Mentor Graphics®

www.mentor.com