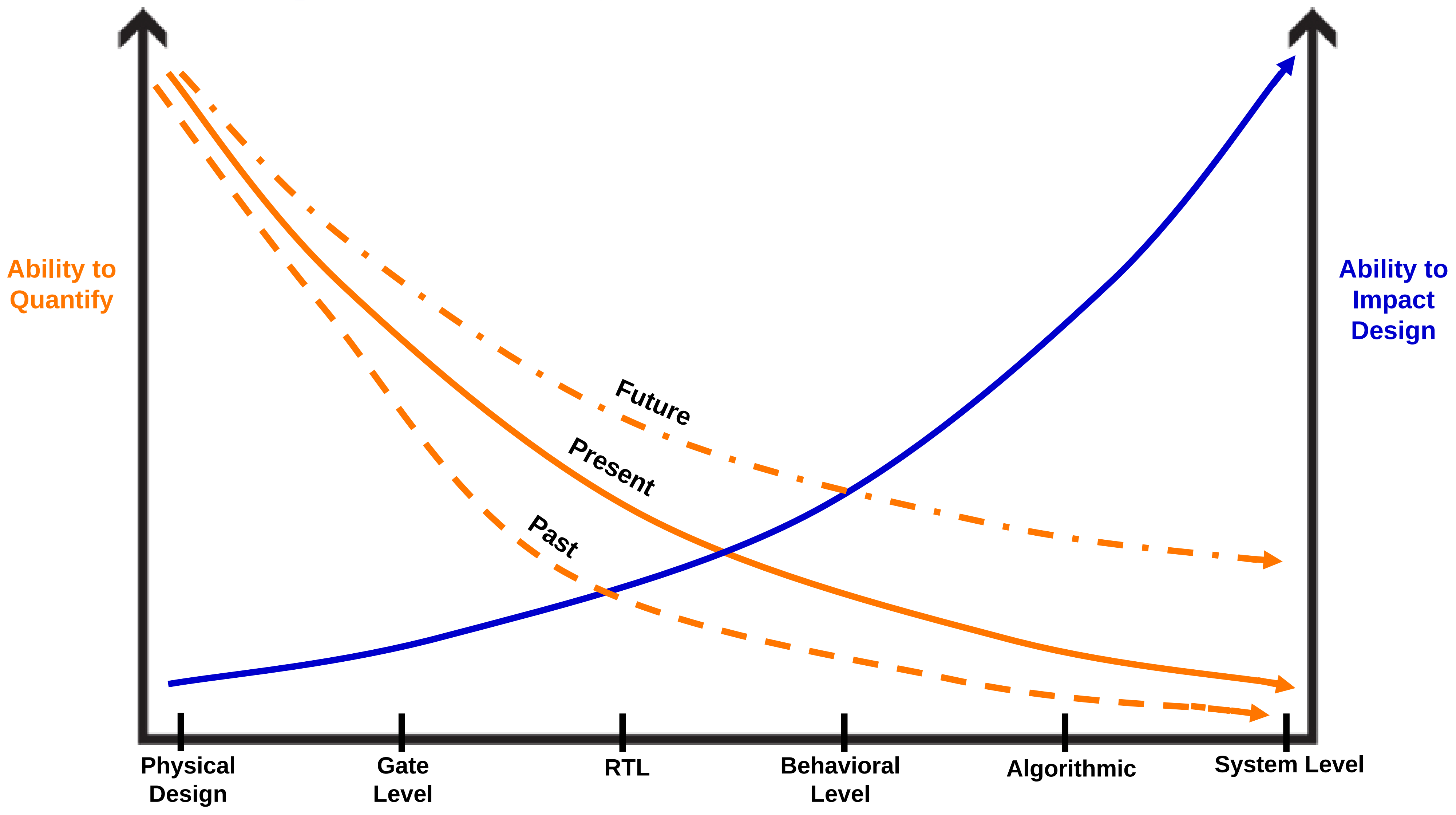
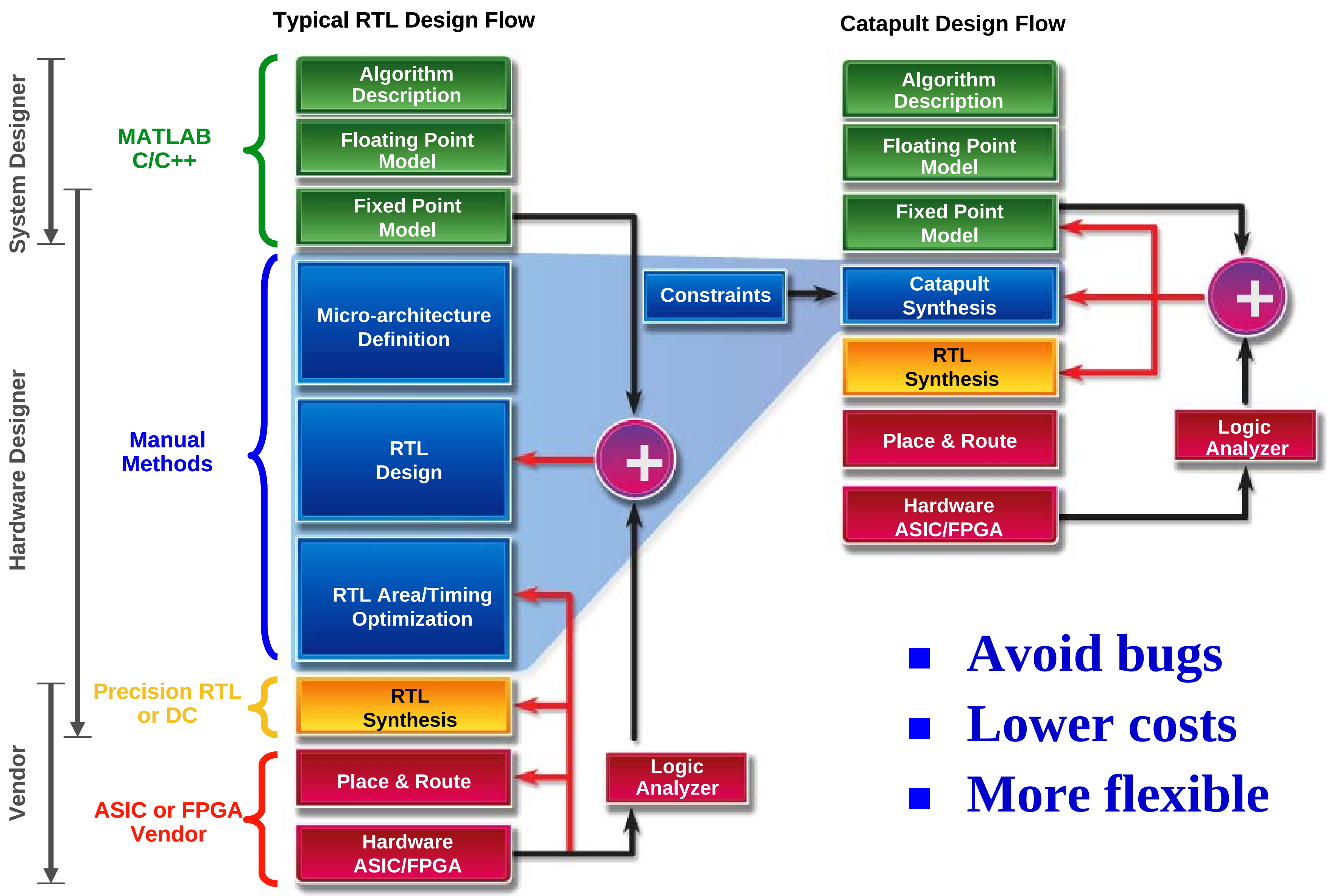
Low Power Algorithm Implementation And Verification Using C++

Dan Gardner
Design Creation and Synthesis Division
Mentor Graphics

Improved Power and Cost Through Improved System Architecture

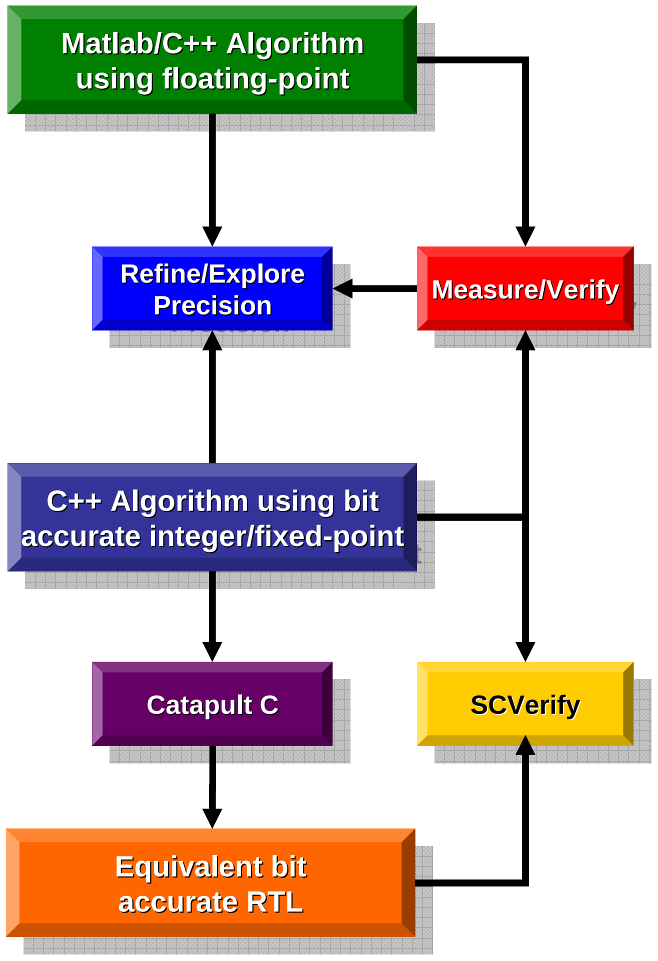


Traditional Flow vs. Catapult Flow



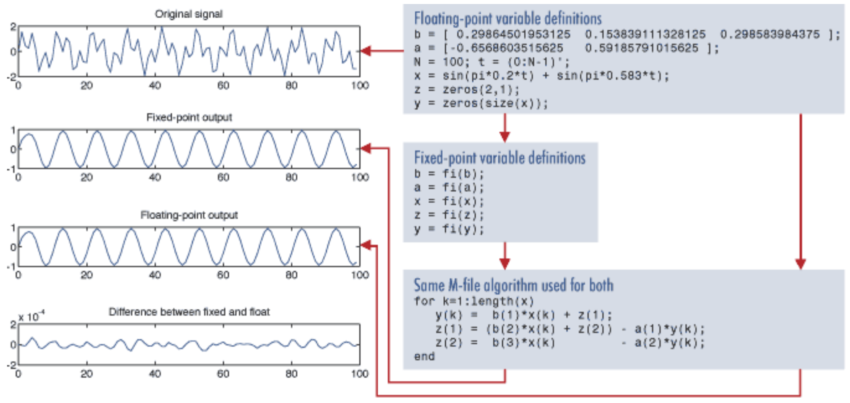
- Avoid bugs
- Lower costs
- More flexible

Numerical Refinement & Closed Loop Verification



Matlab/C++ Algorithm using floating-point
Refine/Explore Precision
Measure/Verify
C++ Algorithm using bit accurate integer/fixed-point
Catapult C
SCVerify
Equivalent bit accurate RTL

- Verification/Validation depends on application and granularity of algorithm
  - Bit Error Rate
  - Mean Square Error
  - No overflows requirement
- Floating-point may be optional step
  - Code fixed-point from the start
- Simulation speed essential for validation/verification
- Use exact bit-widths required to meet specification and save power/area



Floating-point variable definitions

```
b = [ 0.29864501953125  0.153839111328125  0.298583984375 ];
a = [-0.6568603515625   0.59185791015625 ];
N = 100; t = (0:N-1)';
x = sin(pi*0.2*t) + sin(pi*0.583*t);
z = zeros(2,1);
y = zeros(size(x));
```

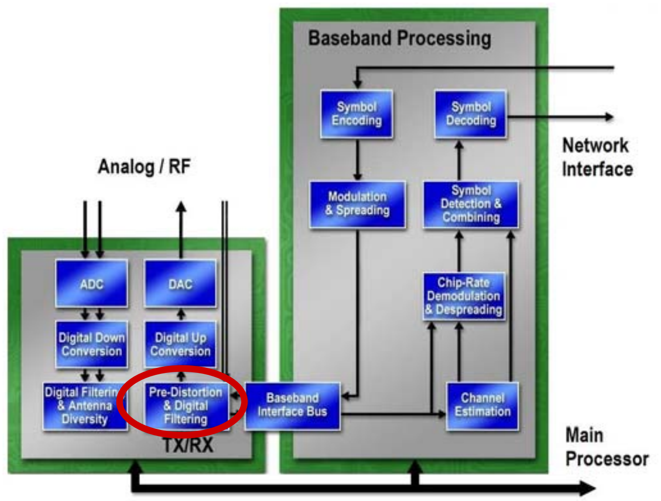
Fixed-point variable definitions

```
b = fi(b);
a = fi(a);
x = fi(x);
z = fi(z);
y = fi(y);
```

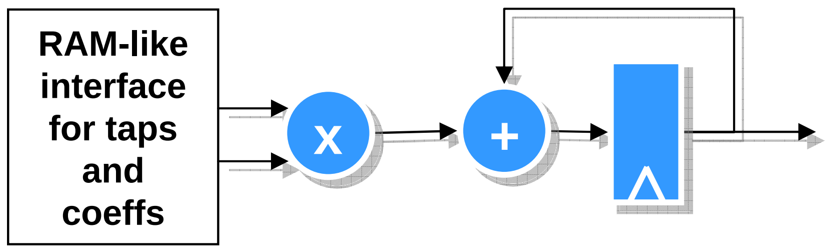
Same M-file algorithm used for both

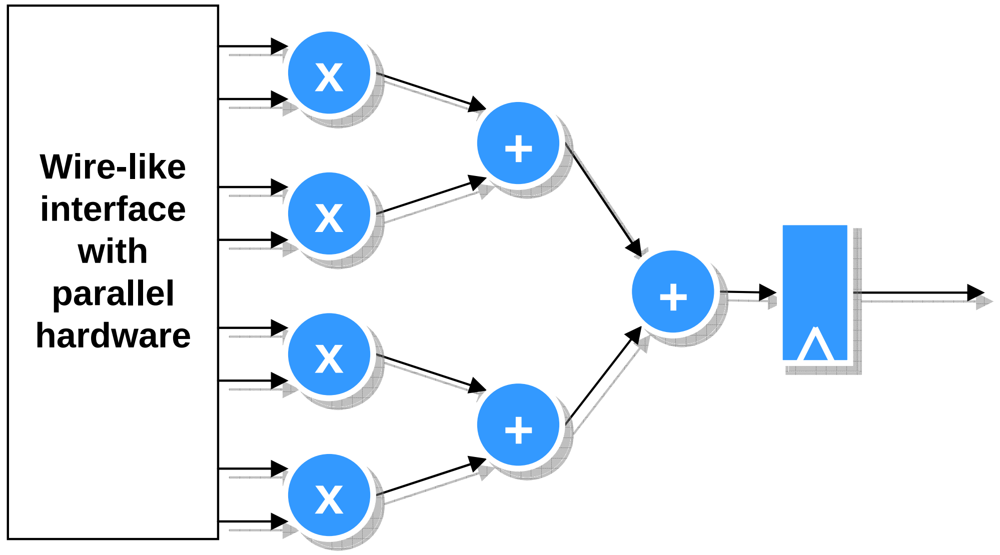
```
for k=1:length(x)
   y(k) =  b(1)*x(k) + z(1);
   z(1) = (b(2)*x(k) + z(2)) - a(1)*y(k);
   z(2) =  b(3)*x(k)         - a(2)*y(k);
end
```

Micro-Architecture Optimization



```
void MAC(int8 taps[4], int8 coefs[4], int10 *out)
{
  int18 accum = 0;
  for (int i=0; i<4; i++)
    accum += taps[i] * coefs[i];
  *out = accum >> 8;
}
```





Target Optimized RTL Code Generation

```
int multaddadd (short A[4], short B1[4])
{
  return (A[0]*B[0]) + (A[1]*B[1]) + \
         (A[2]*B[2]) + (A[3]*B[3]);
}
```

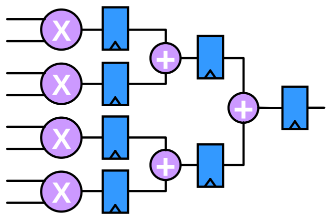
Architecture-neutral description

Architectural Constraints 1

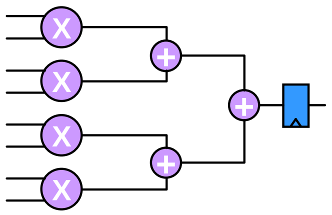
200MHz
Slow speed grade

Architectural Constraints 2

100MHz
Fast speed grade



RTL 1



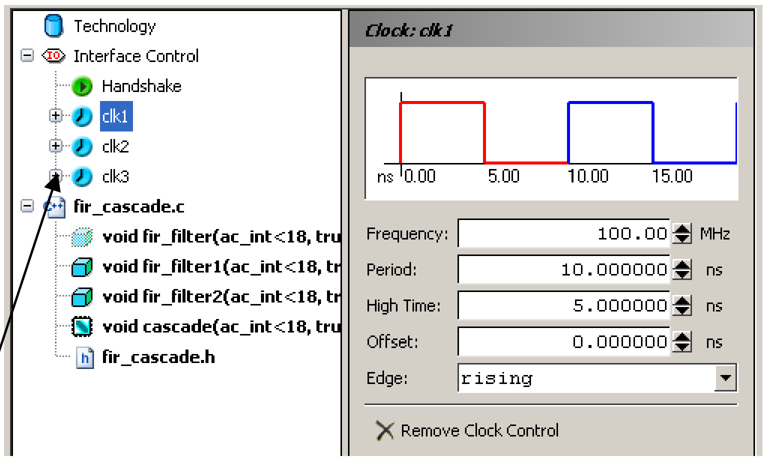
RTL 2

Frequency is a parameter

Multi-clock Design

- **Blocks with lower data rates run with slower clock**
  - Reduction in switching power
  - Reduction in static power by decreasing block area

Technology Constraints

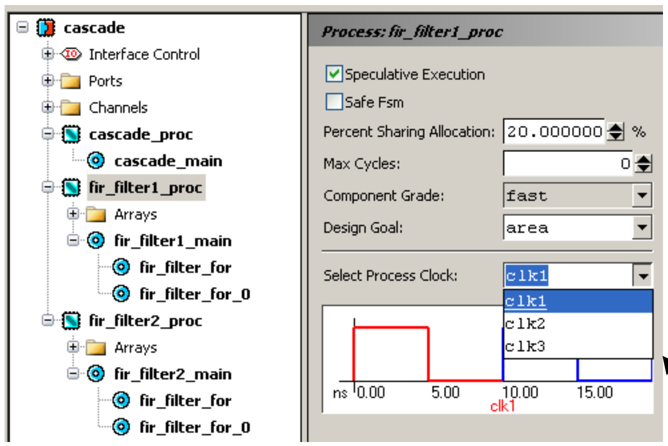


Technology
Interface Control
Handshake
clk1
clk2
clk3
fir_cascade.c
void fir_filter(ac_int<18, tru
void fir_filter1(ac_int<18, tr
void fir_filter2(ac_int<18, tr
void cascade(ac_int<18, tru
fir_cascade.h

Clock: clk1

Frequency:	100.00	MHz
Period:	10.000000	ns
High Time:	5.000000	ns
Offset:	0.000000	ns
Edge:	rising	

Architectural Constraints



cascade
Interface Control
Ports
Channels
cascade_proc
cascade_main
fir_filter1_proc
Arrays
fir_filter1_main
fir_filter_for
fir_filter_for_0
fir_filter2_proc
Arrays
fir_filter2_main
fir_filter_for
fir_filter_for_0

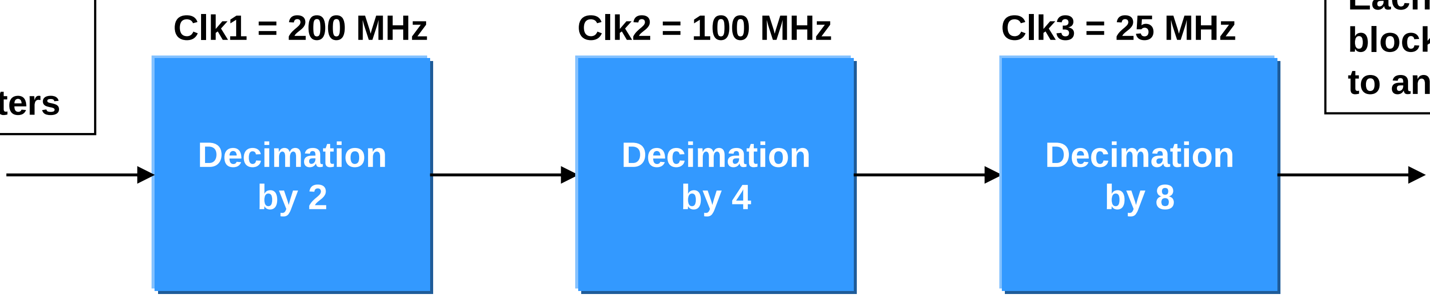
Process: fir_filter1_proc

[✓] Speculative Execution
[ ] Safe Fsm

Percent Sharing Allocation:	20.000000	%
Max Cycles:	0	
Component Grade:	fast	
Design Goal:	area	
Select Process Clock:	clk1	

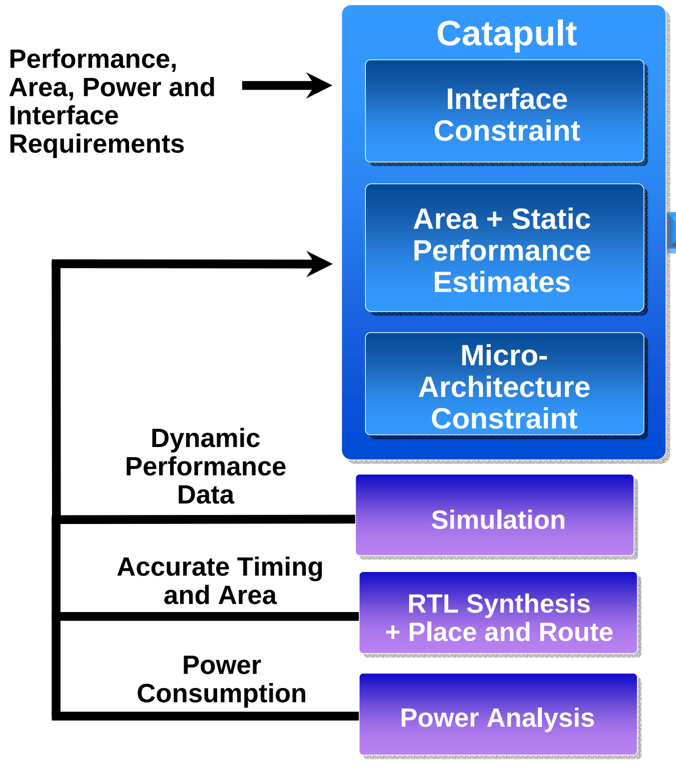
Multiple clocks specified with unique parameters

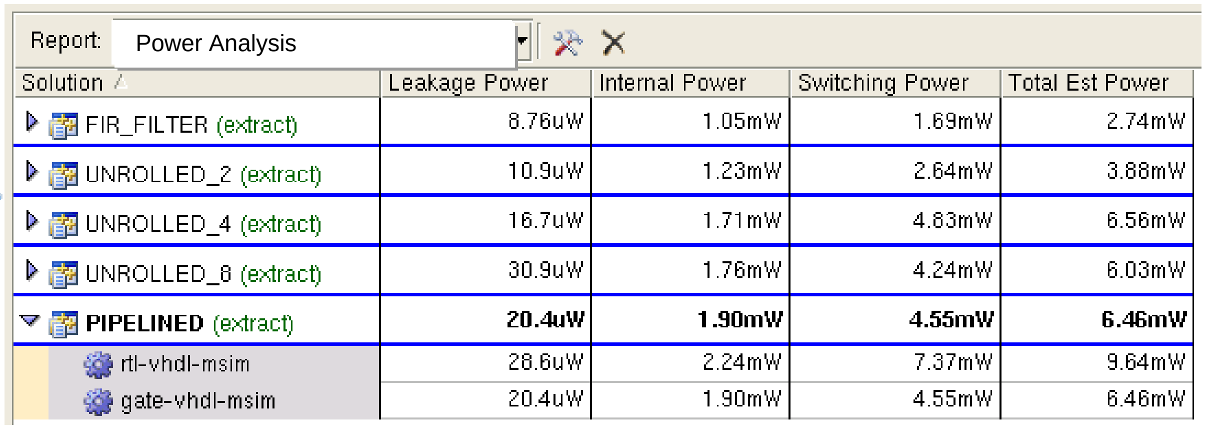
Each hierarchical block can be assigned to any clock domain



Closed-loop Power Analysis and Optimization

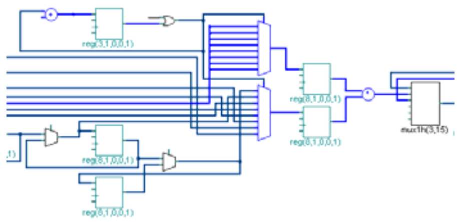
- Power consumption data annotated into Catapult using leading power analysis tools
- Micro-architecture optimizations used to balance power/area/performance
- Average 30% power savings using this flow


Performance, Area, Power and Interface Requirements
Catapult
Interface Constraint
Area + Static Performance Estimates
Micro-Architecture Constraint
Dynamic Performance Data
Simulation
Accurate Timing and Area
RTL Synthesis + Place and Route
Power Consumption
Power Analysis

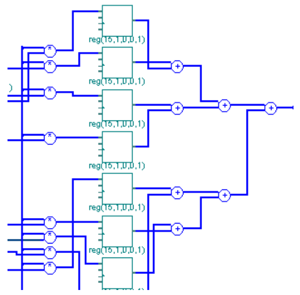

Report: Power Analysis

Solution	Leakage Power	Internal Power	Switching Power	Total Est Power
FIR_FILTER (extract)	8.76uW	1.05mW	1.69mW	2.74mW
UNROLLED_2 (extract)	10.9uW	1.23mW	2.64mW	3.88mW
UNROLLED_4 (extract)	16.7uW	1.71mW	4.83mW	6.56mW
UNROLLED_8 (extract)	30.9uW	1.76mW	4.24mW	6.03mW
PIPELINED (extract)	**20.4uW**	**1.90mW**	**4.55mW**	**6.46mW**
rtl-vhdl-msim	28.6uW	2.24mW	7.37mW	9.64mW
gate-vhdl-msim	20.4uW	1.90mW	4.55mW	6.46mW

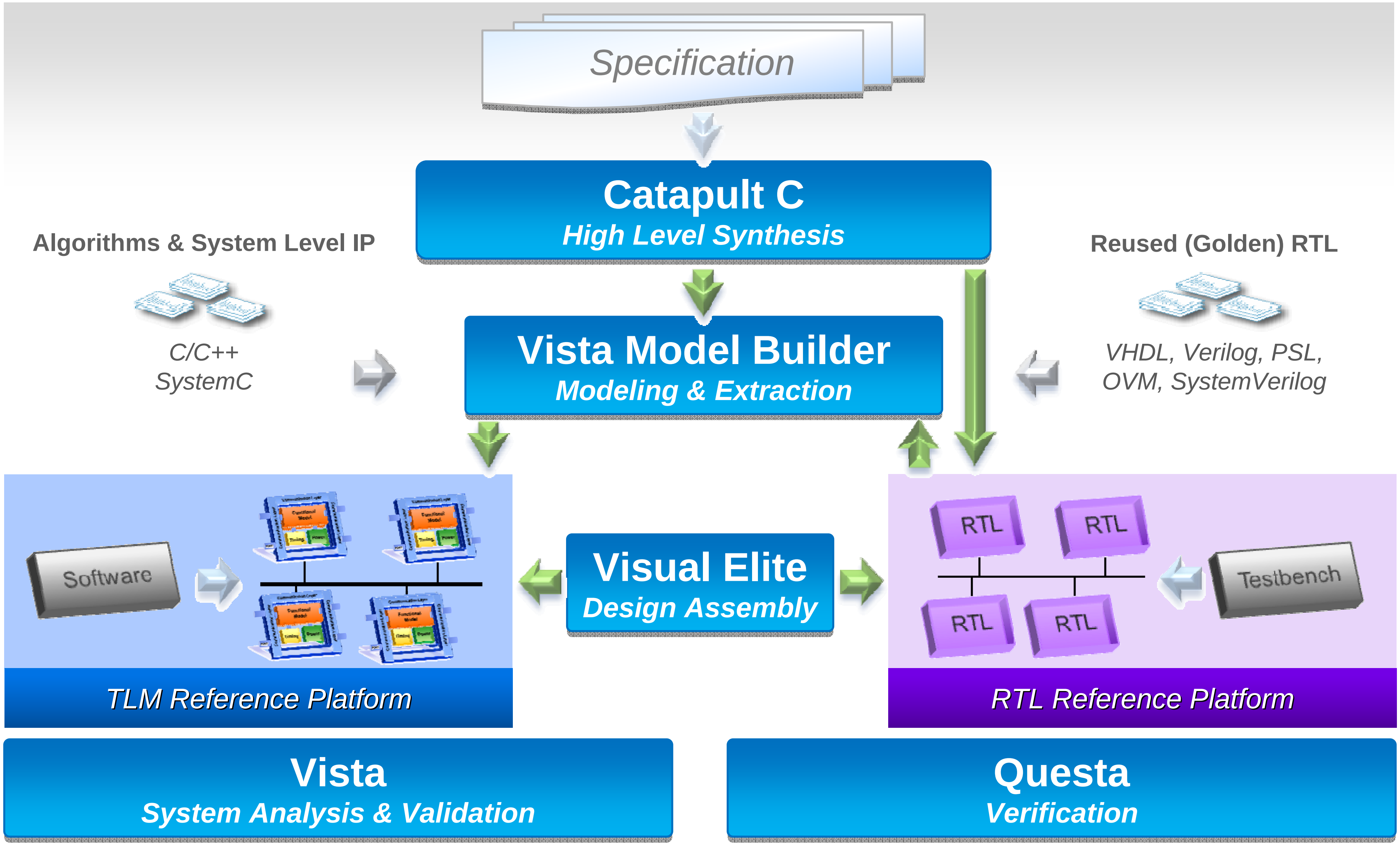
Low power / Small area



Higher power / Larger area



ESL Flow

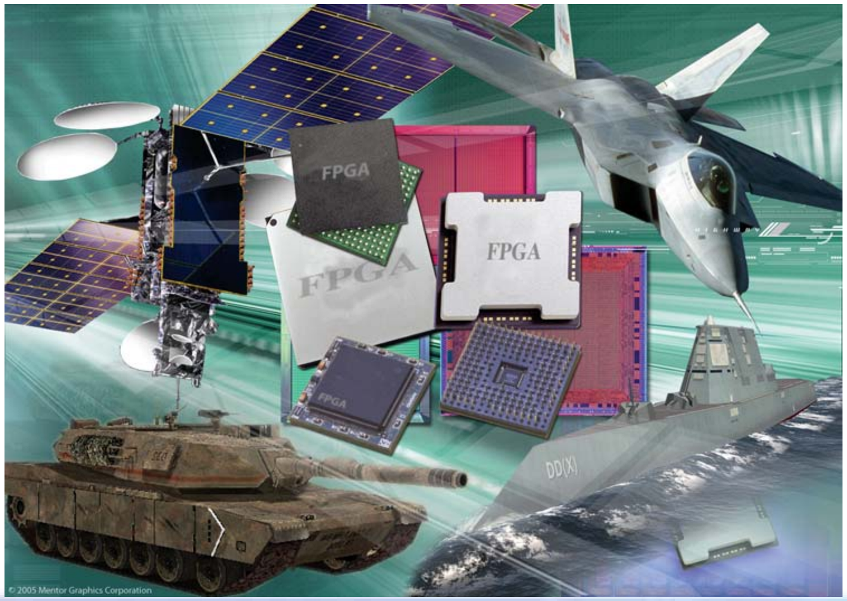

Specification
Catapult C
High Level Synthesis
Algorithms & System Level IP
C/C++
SystemC
Vista Model Builder
Modeling & Extraction
Reused (Golden) RTL
VHDL, Verilog, PSL,
OVM, SystemVerilog
Software
TLM Reference Platform
Visual Elite
Design Assembly
RTL
RTL
RTL
RTL
Testbench
RTL Reference Platform
Vista
System Analysis & Validation
Questa
Verification

Additional Information

Special Challenges with Mil/Aero DSP

"High Cost of Failure"

- **Design reuse**
  - **Very long product life cycles**
  - **Legacy design difficult to retarget**
  - **Switching between FPGA vendors is very expensive**
- **Design quality**
  - **Achieving optimal numerical precision is difficult**
  - **Finding optimal hardware architecture is time consuming**
  - **Designs are typically overbuilt to guardband design goals**
- **Functional correctness**
  - **Mandatory for mission critical hardware**
  - **Up to 60% of design errors come from disconnect between functional spec and RTL implementation**
  - **RTL is too slow for system verification**
- **Time to Market**
  - **Tight milestones in government projects**
  - **Late changing requirements**



Value of Algorithmic Synthesis

Functional

Function and Operate Well

⬇ Cost

Avoid Bugs

Optimized

Optimized Resources Lower Power

⬆ Margin

Reduce Cost

Flexible

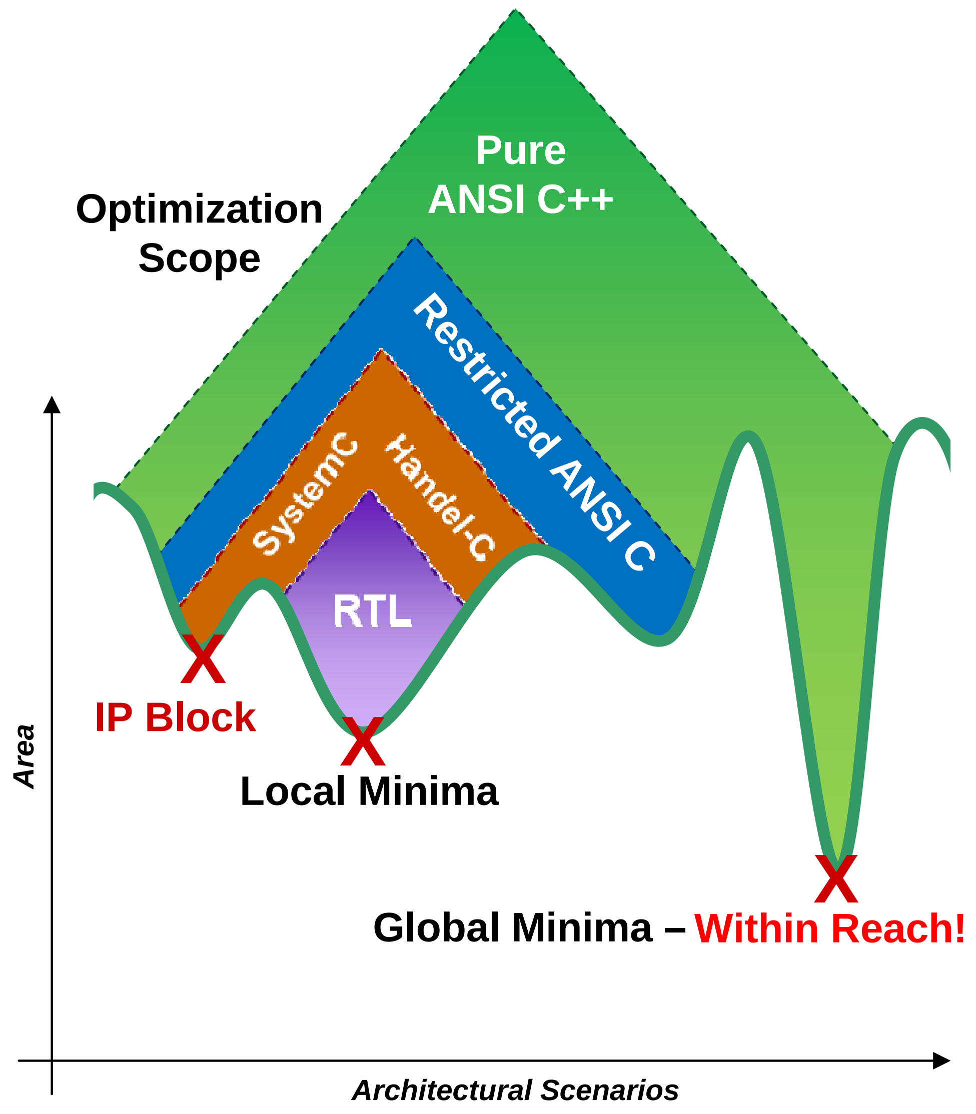
Quickly Deliver Derivative Designs

Market

First to Market

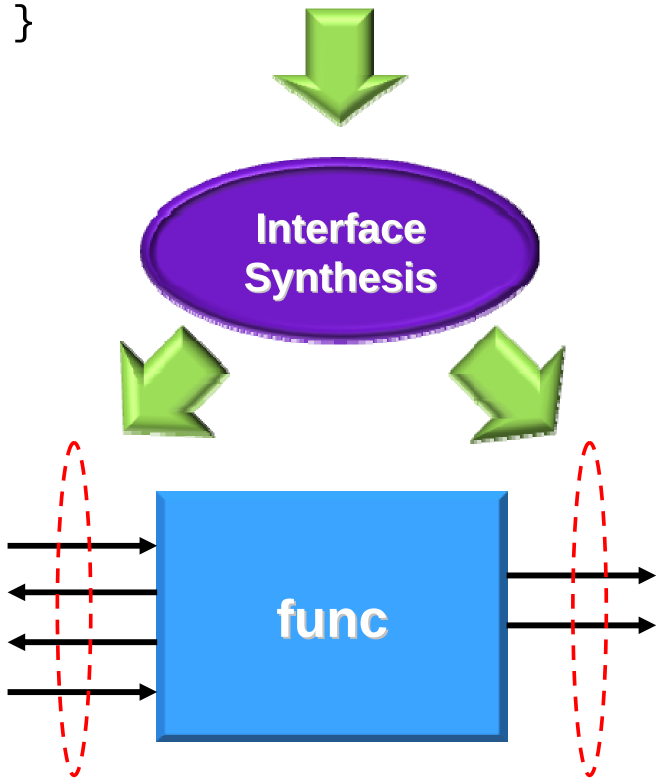
Optimized Design Architecture

- RTL confines your implementation to few solutions in close proximity
- Structural languages offer limited trade-off's
  - Architectural details embedded in the source
- Restricted ANSI C
  - Limits reuse
  - Complicates coding style
  - Prevents bit-accurate modeling & numerical refinement
- Pure ANSI C++ allows exhaustive exploration of design space
  - Extremely compact
  - Object oriented hardware reuse
  - Optimization through interactive constraints
  - Optimize serial vs. parallel
  - Optimize sequential vs. pipelined



Interface Optimization With Interface Synthesis

```
void func ( int  A[5][16],
            char B[16],
            bool mode   ) {
  ...
}
```

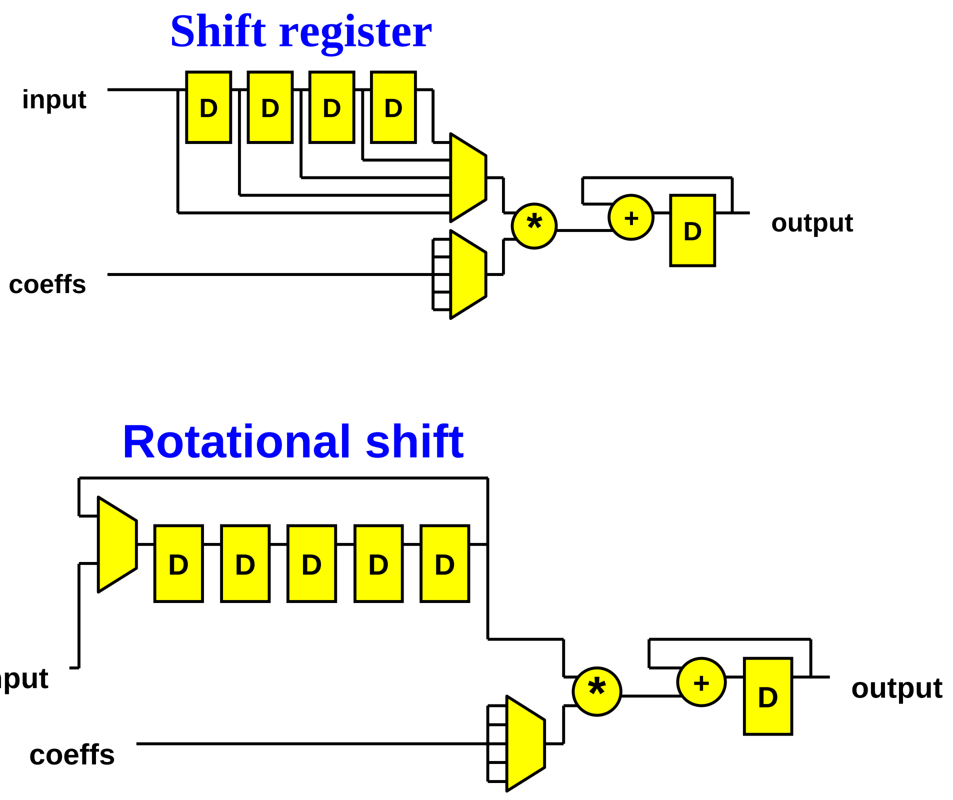


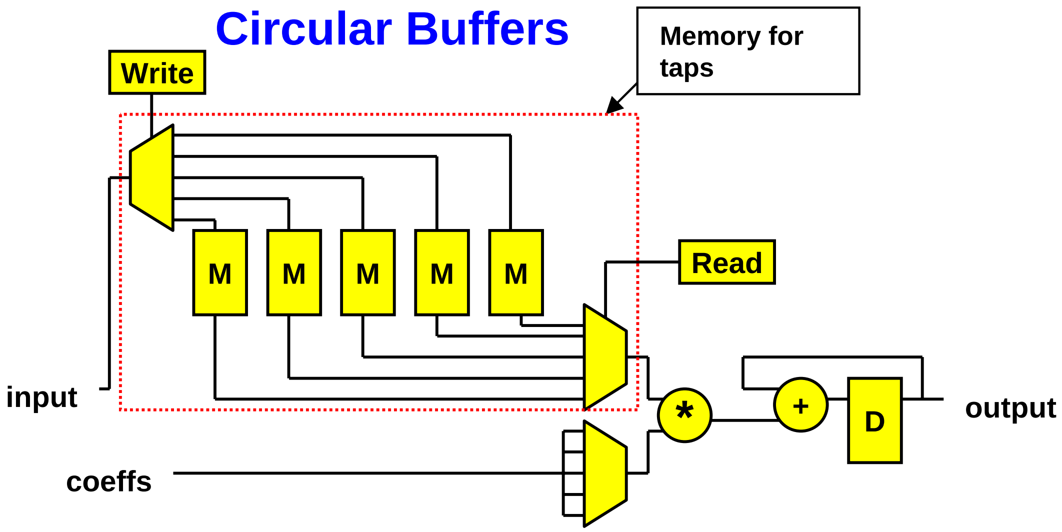
- C++ source and testbench independent of HW interface
- Designers focus on architecture and function
- Micro-architecture tuned to the interface
  - Memories
  - Busses
  - Streaming data
- Adjust bit-widths to balance performance and power

Patent-pending

Memory Architecture in C++

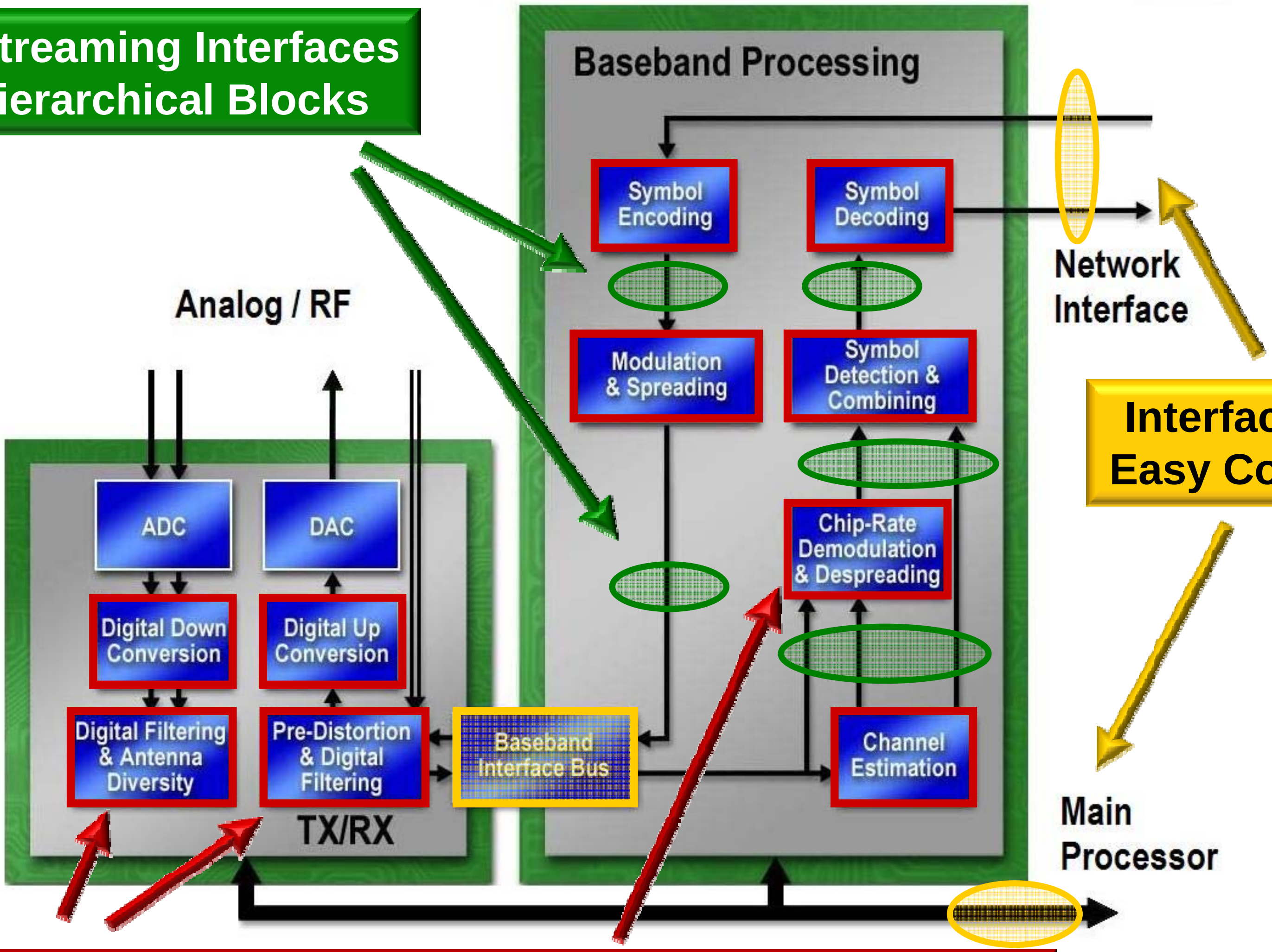
- Power, performance and area for many algorithms are highly dependent on memory architecture
- C++ makes various memory architectures easy to explore
  - For example, something as simple as a FIR filter can take numerous "forms"





System Level Capabilities

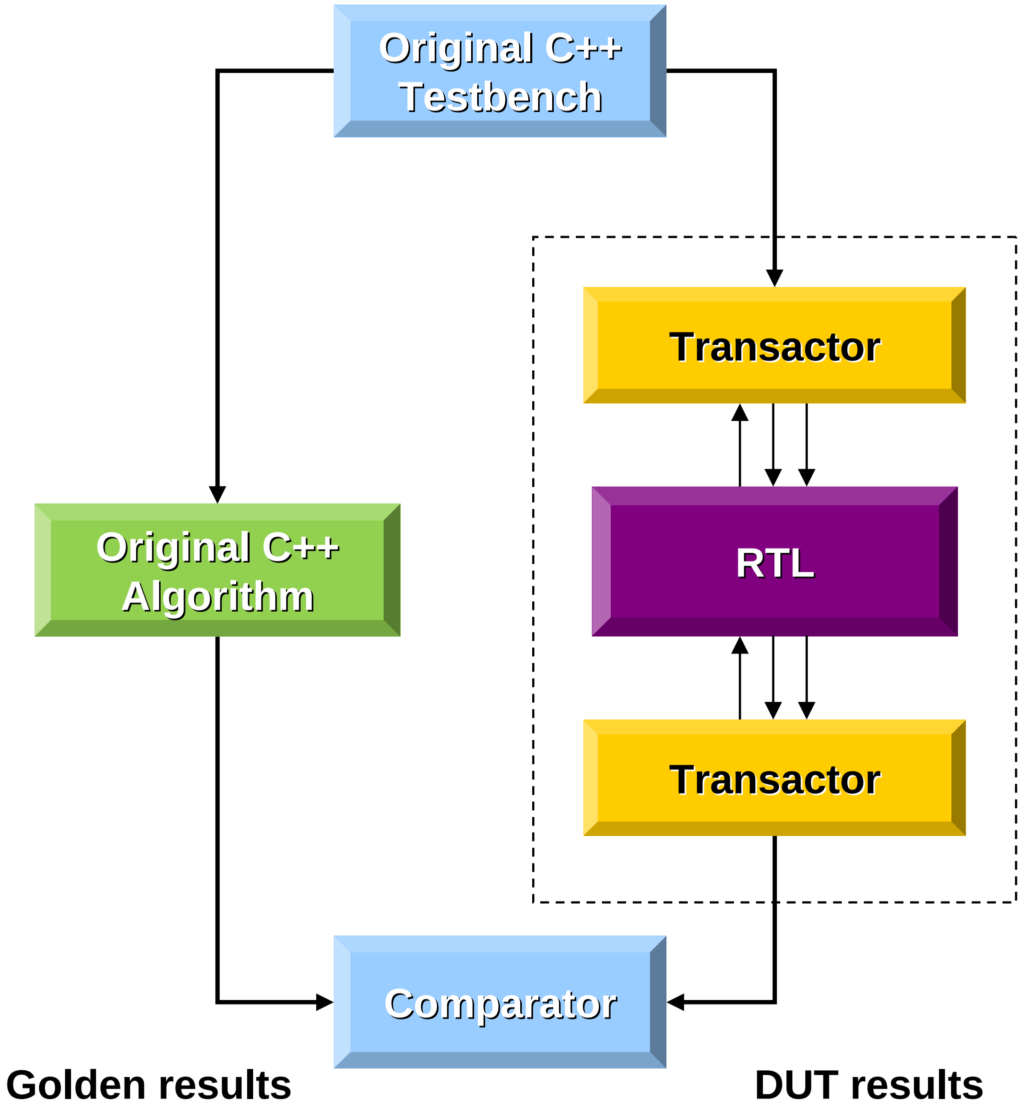
Automatic Streaming Interfaces Between Hierarchical Blocks



Interface Synthesis for Easy Connection to Bus

Optimized hardware creation using Catapult Synthesis Starting from High Speed pure ANSI C++ Algorithmic Model

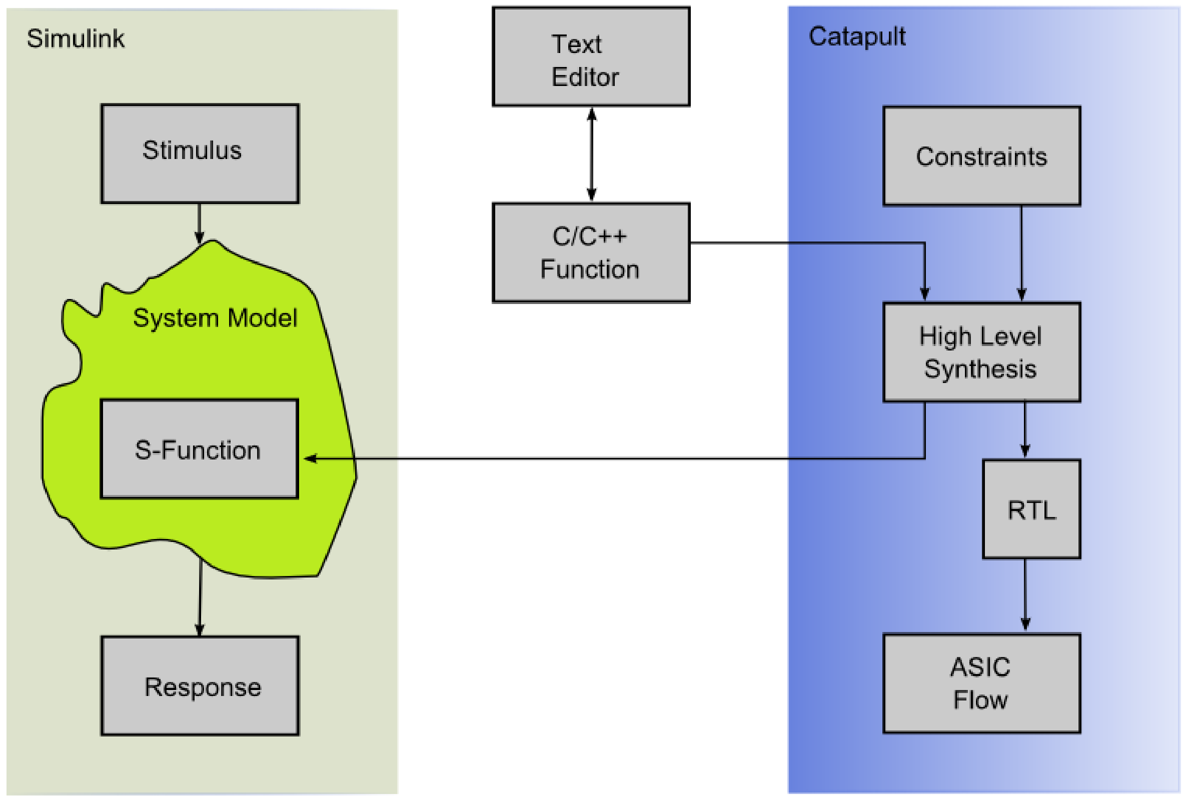
Catapult Verification Extension SCVerify



- SystemC Transactors
- Original C++ testbench reused to verify the RTL
- Transactors convert function calls to pin-level signal activity
- Push button solution creates Makefiles and Simulation Scripts

Catapult & Mathworks Partnership

- **Provides link between Catapult and MATLAB/ Simulink**
  - System Simulation
  - Numerical refinement
  - HW verification
- **Closes the gap between algorithm design and implementation**
- **Focus on high-end FPGA and ASIC**


Simulink
Stimulus
System Model
S-Function
Response
Text Editor
C/C++ Function
Catapult
Constraints
High Level Synthesis
RTL
ASIC Flow