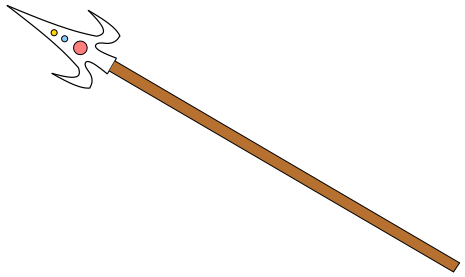
Trident Compiler - A compiler for Scientific Computing on FPGAs



Trident Compiler Team

Justin L. Tripp, Kris Peterson, Jeff Poznanovic, Christine Ahrens, Neil Steiner

Los Alamos National Laboratory

October 2005

Introduction

- Scientific Computing is typically floating point (singles and doubles)
- Scientific Computing requires support of legacy software
- Much of the software has high-level parallelism already extracted (e.g., MPI, etc.)
- Other Floating-Point Compilers for FPGAs
  - ⋆ Map compiler – only for the SRC Machine
  - ⋆ Celoxica DK – macro generation only, HandelC
- Need a compiler to work with high-level languages that provides acceleration for Scientific Computing.

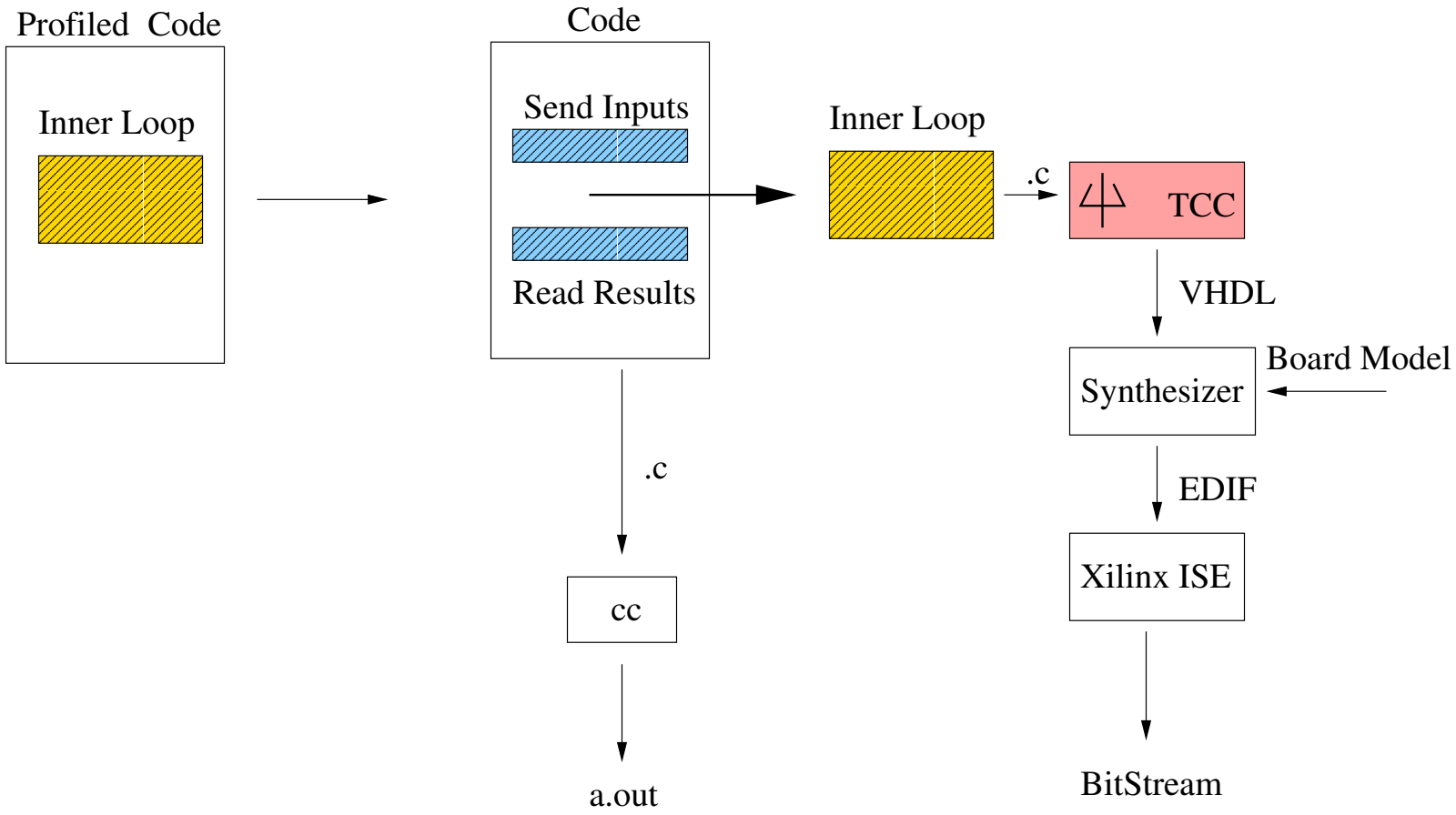
Trident Compiler

FPGAs can perform high-performance floating-point (FP) operations. However, traditional FPGA development is difficult and few tools exist to specifically aid FP hardware development.

Trident Goals:

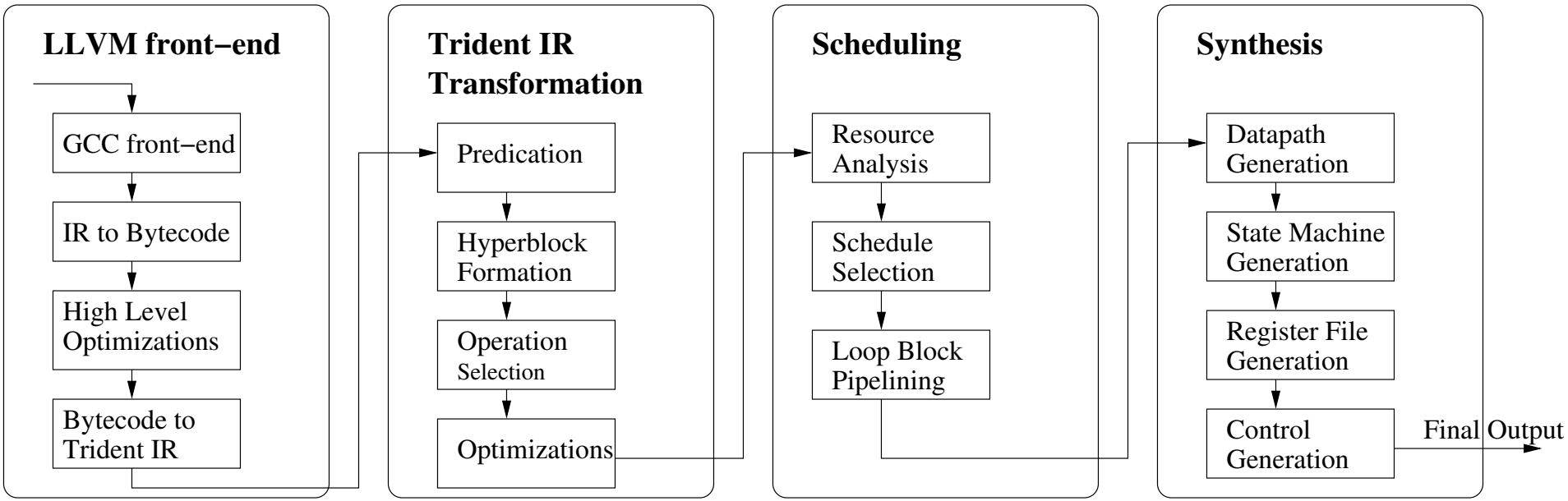
- Accept C input with double and float data types.
- Automatically extract available parallelism.
- Automatic pipelining of loops.
- Allow the selection from different FP libraries.
- Allow user developed FP libraries.

Big Picture


Profiled Code
Inner Loop
Code
Send Inputs
Read Results
.c
cc
a.out
Inner Loop
.c
TCC
VHDL
Synthesizer
Board Model
EDIF
Xilinx ISE
BitStream

Organization

Four principal phases of compilation:



LLVM - Low Level Virtual Machine (www.llvm.org)

IR - Intermediate Representation

LLVM: How Trident Uses it

- GCC-based C and C++ front-end provided by LLVM produces LLVM bytecode (optimizations and linking are disabled).
- Note: C programs should not contain print statements, recursion, malloc or free calls, function arguments or returned values, calls to functions with variable length argument lists or arrays without a declared size.
- LLVM Trident pass optimizes LLVM bytecode (constant propagation, small function inlining, loop invariant hoisting, tail call elimination, small loop unrolling, CSE and others) and generates modified form of LLVM language
- Trident parses LLVM language into Trident IR (Java)

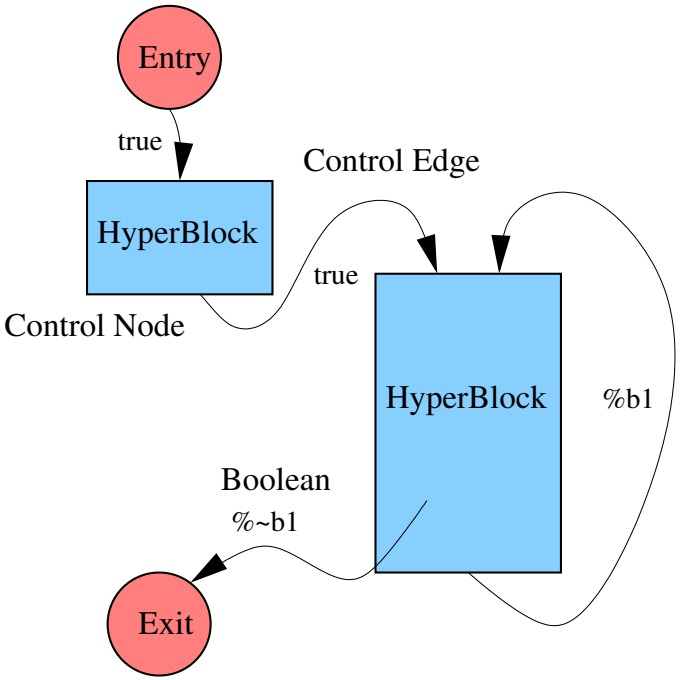
Trident Intermediate Representation (IR)

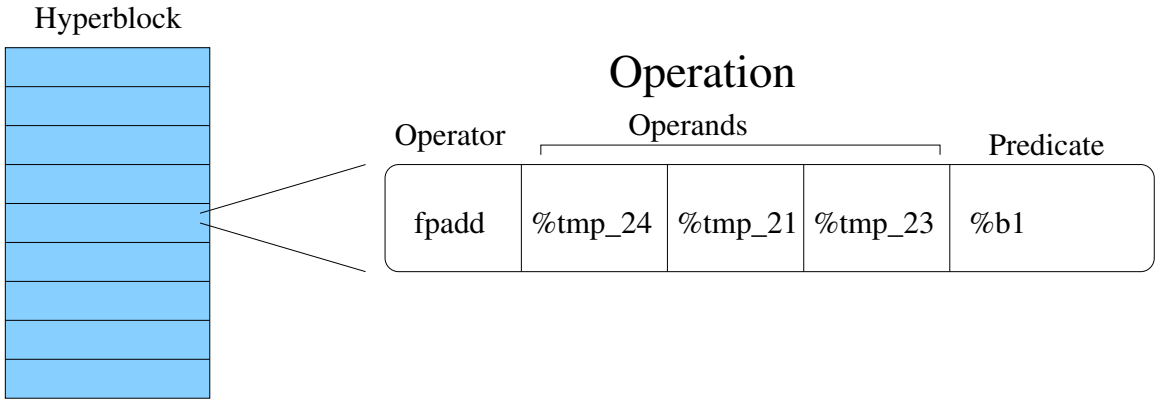
The Trident IR is used for the following:

- Convert operations into predicated operations and form hyperblocks.
- Optimize and modify the IR to eliminate unnecessary and expensive operations.
- Map generic operations to library specific operations.
- Resource allocation, scheduling, and synthesis.

Trident IR - Data Structures

Control Flow Graph





Operations also include:

- Start and Stop times
- Operand Type
- HW Reuse set
- Operator Class Information

Trident IR - Passes

Optimization	Hardware	Fix Up	Util and Debug
AddPredicates	AllocateArraysPass	AddTypeToPrimals	CreateDependenceFlow
ALAPSchedule	AnalyzeHWContraints	CallReplace	GenSchedulerStats
ASAPchedule	AnalyzeLogicSpace	CheckBlockNames	SetNodeOrder
CalcII	GenerateCircuit	ControlRemoval	PrintDataFlow
CSERemoval	LoadHWInfo	ConvertGep	PrintGraph
FDSchedule	OperationSelection	CvrtBlkNonPrimToPrim	PrintLoopGraph
FixFAbsInst	SwitchIfConvert	EnsureSingleDefs	PrintVariables
FixFSubInst		FixLoadPtrInsts	VerifyBlockGraph
GlobalDeadCode		PhiLowering	VerifyOperands
MergeParallelBlocks		RemoveGlobalPred	
MergeSerialBlocks			
ModuloSchedule			

Hardware Analysis and Instruction Scheduling

There is always a limited possible communication bandwidth with memory and space on any given FPGA chip. Ensuring successful implementation of the circuit on the target chip and achieving maximum execution speed requires analysis of the hardware.

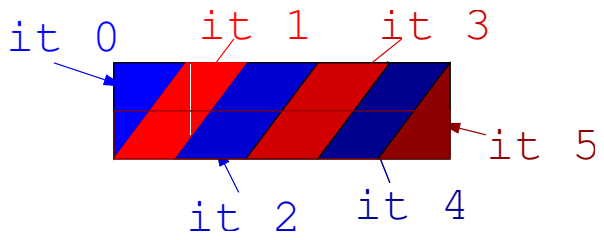
- Hardware Analysis
  - ⋆ Preliminary schedule to determine times of memory reads and writes
  - ⋆ Array to memory allocation
  - ⋆ Logic space requirements analysis
- Instruction Scheduling (schedule type chosen by user)
  - ⋆ non-loop code - ASAP, ALAP, Force-Directed
  - ⋆ loop code - Modulo scheduling

Force-Directed Scheduling: Results

	FD	asap	alap
simple pi			
Average Ops/Cycle	0.163	0.163	0.162
Max Ops/Cycle	2.0	4.0	2.0
Cycle Count	122	123	122
photon			
Average Ops/Cycle	0.378	0.321	0.378
Max Ops/Cycle	4.0	13.0	4.0
Cycle Count	111	111	111
euclid			
Average Ops/Cycle	0.229	0.229	0.229
Max Ops/Cycle	4.0	6.0	4.0
Cycle Count	70	70	70

Modulo Scheduling

Modulo Scheduling is an algorithm that schedules a loop as just shown (with a body having a depth of II cycles), while simultaneously finding a working II and scheduling the instructions within the body of the loop such that hardware conflicts and interloop dependency problems can be avoided.



II - Initiation Interval

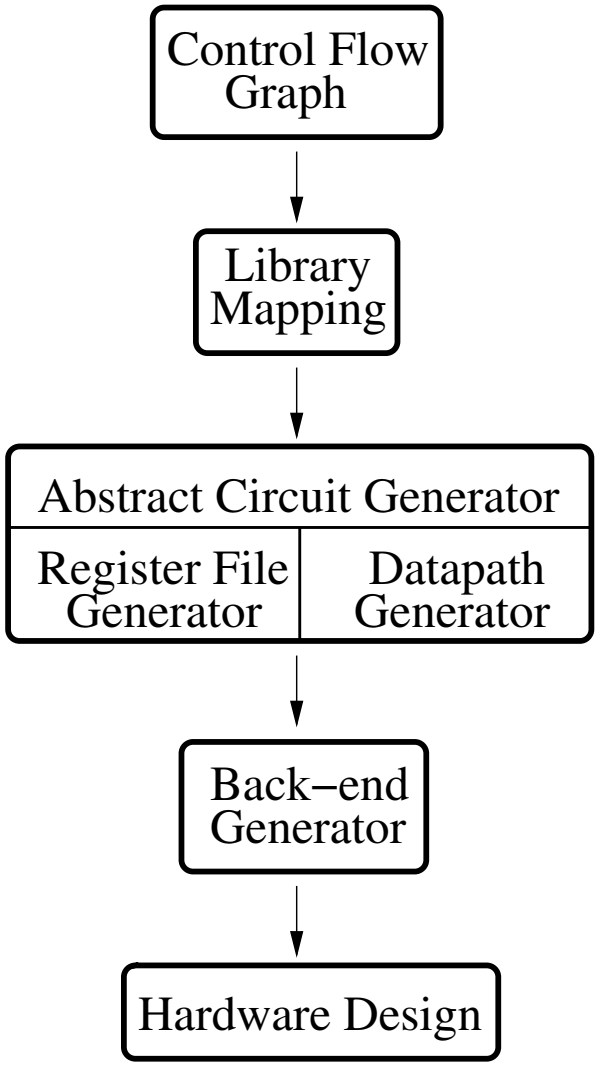
Modulo Scheduling: Results

These results are for the main loop of the code:

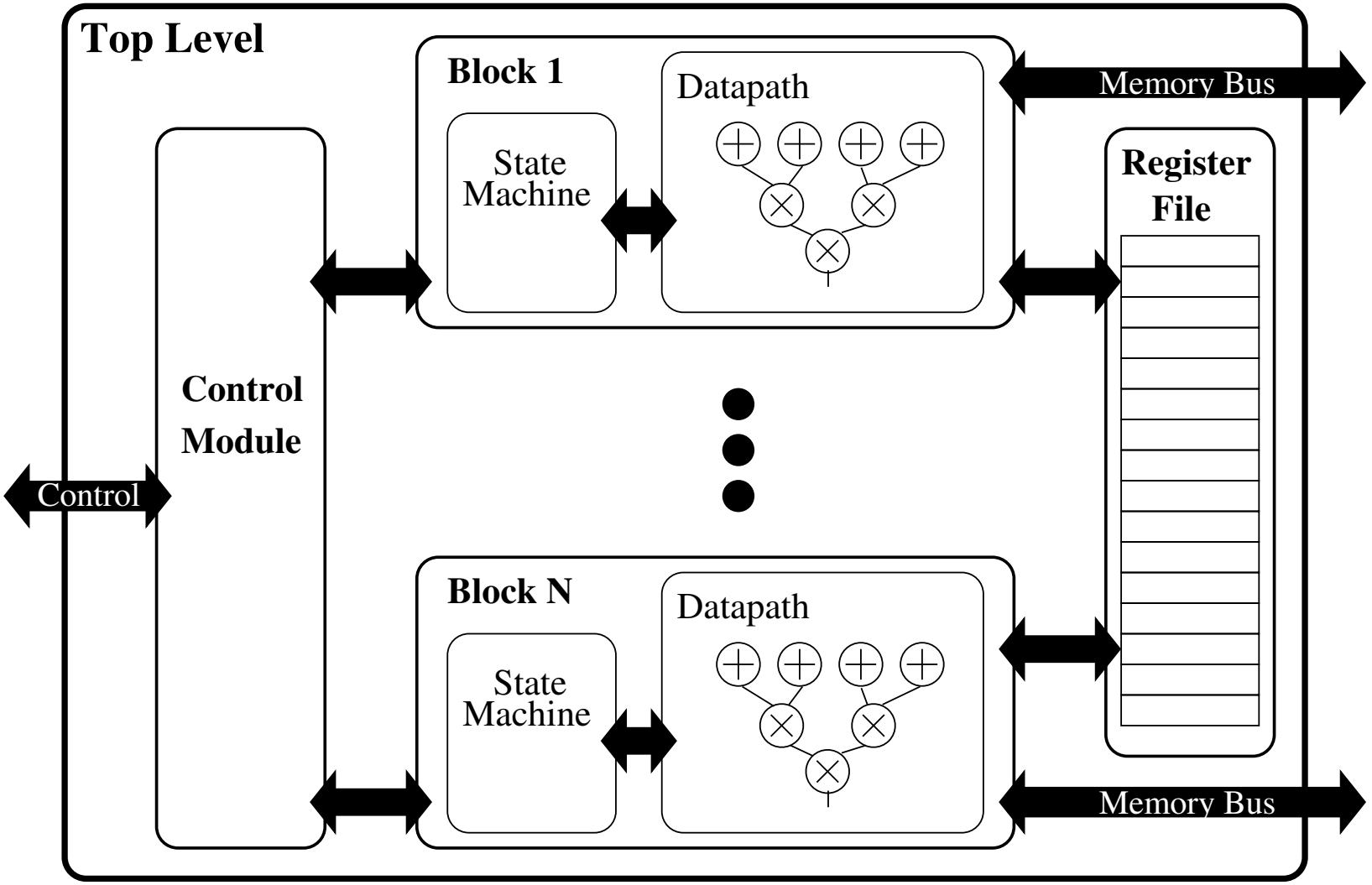
	No Modulo			Modulo		
	FD	ASAP	ALAP	FD	ASAP	ALAP
simple pi						
Average Ops/Cycle	0.168	0.168	0.170	1.76	1.90	1.82
Max Ops/Cycle	2.0	4.0	2.0	25.0	27.0	26.0
Cycle Count	89	89	88	29	29	29
simple float loop						
Average Ops/Cycle	2.12	2.12	2.83	29.25	29.25	29.25
Max Ops/Cycle	4.0	4.0	4.0	50.0	50.0	50.0
Cycle Count	17	17	17	4	4	4

Synthesis

- Datapath generation
- FSM and Multiple-block Control
- Arrays
- FP Library Integration
  - ⋆ Quixilica
  - ⋆ Arénaire
  - ⋆ Trident (local library)
- HDL Generation
  - ⋆ Abstract Circuit
  - ⋆ VHDL Backend
  - ⋆ Board Interface



Synthesis – Structure



Synthesis Results

Here are the results for a few benchmarks. These were obtained for the Cray XD1 using the ISE 6.3p3 tools with the Quixilica floating-point library. The overhead for interfacing to the Cray FPGA Board is about 10-15%.

Benchmark	Clk (MHz)	Slice Count	%Area	Blocks	States
Photon	187	12,109	51	1	112
Photon-hand	*98*	*8,819*	*20*	*1*	*98*
Euclid	200	7,039	19	1	71

Photon-hand is an engineer generated design for the Radiative Heat transfer application (Photon). The numbers are just for the design pipeline and do not include any overhead required to interface with a particular board.

Current Status

Trident provides an open framework for FP computation exploration.

- Support for Single and Double floating point operations.
- Support for fixed size arrays.
- Limited applications can be synthesized, simulated and executed on the Cray XD1.
- C code for the inner loop of the Radiant Heat Transfer Application executes on the XD1.
- Two different FP Libraries can be selected (Quixilica, Arénaire).
- Loop Pipelining via Modulo Scheduling is supported.

Future Directions

- Complete Trident 1.0 Open Source Release
- Add support for streaming data
- More aggressive memory bandwidth allocation
- Support for other HW platforms
- Dynamic Memory Allocation (variable sized arrays)