Using High-Level Synthesis to Uncover PPA Tradeoffs of Various Hardware Accelerators

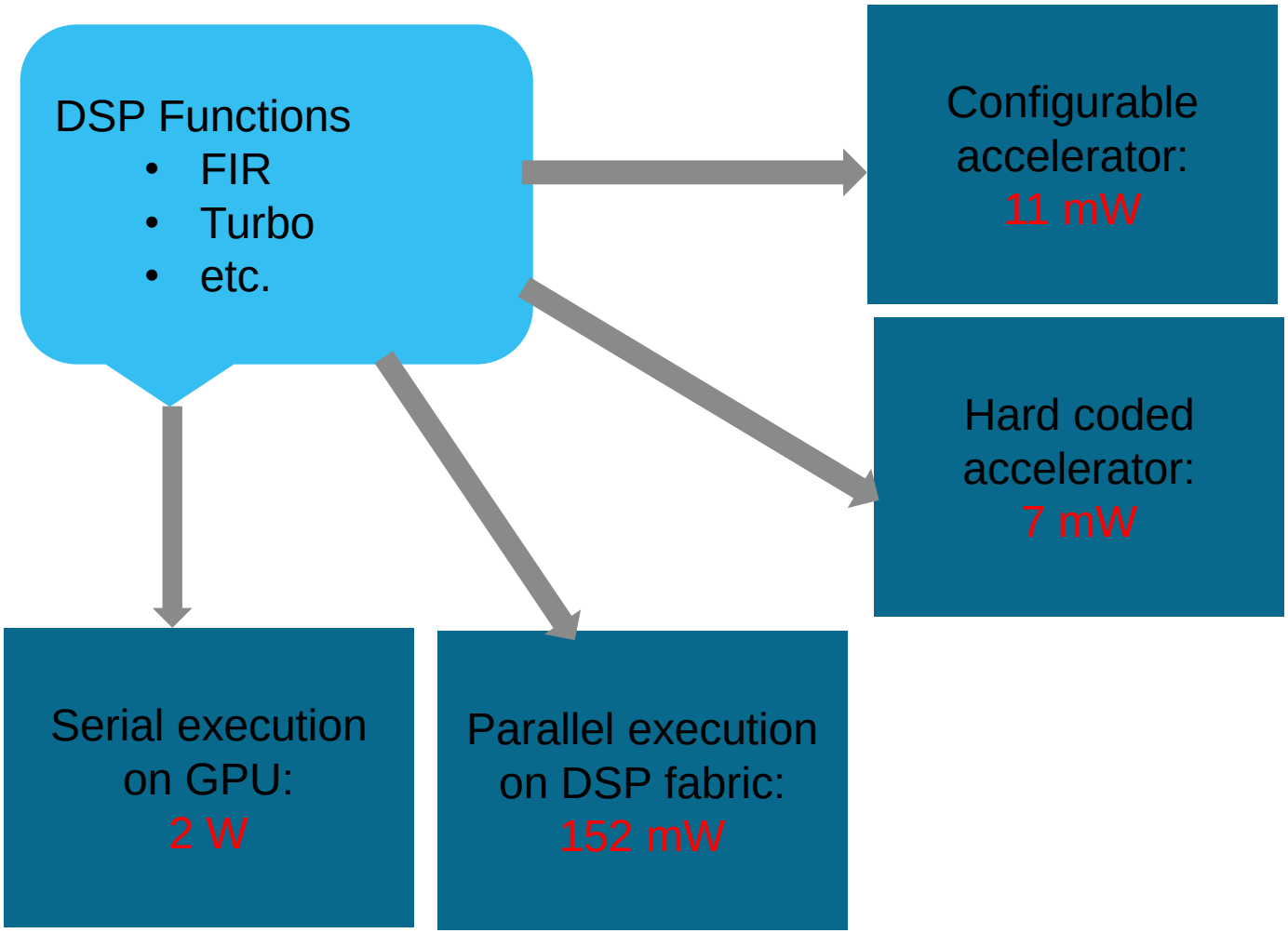
Andy Fox, Steve Anderson
CDNLive
Santa Clara, CA
March 11, 2015

cadence®

Objectives

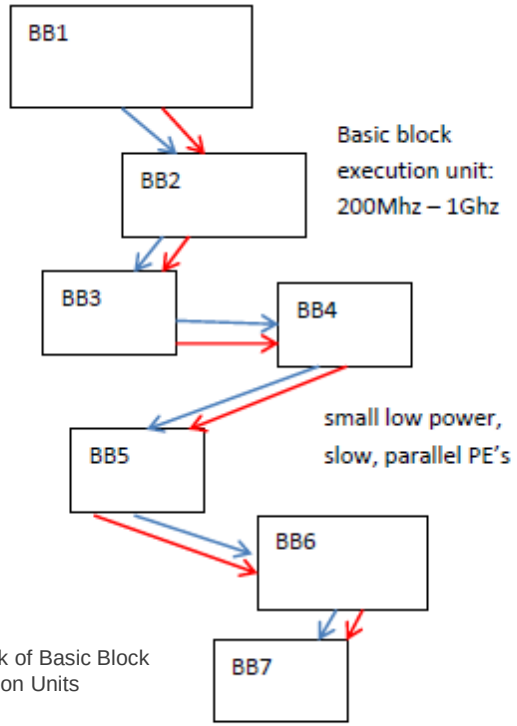
- Evaluate accelerator architecture for DSP applications expressed in “C”
  - Fully programmable DSP fabric expressed as a C++ / SystemC model
  - LLVM software tool chain maps algorithms to DSP fabric
  - Run the assembly code on our SystemC model of the DSP fabric
- Test: can we use this SystemC architecture model as basis for architectural tradeoffs?
  - Generate results (power, area) within 3 months and iterate on architecture.
- Also evaluate adding custom DSP functions to offload some portion of the algorithm
  - Partially programmable accelerators
  - Fixed function accelerators

Accelerators for Power Reduction



DSP fabric of BBPE (Basic Block Processing Elements)

```
void cic()
{
    //comb
    for(int i = 0; i < n; ++i){
        comb[i] = x[i] - delay_buffer[N/I - 1];
        for (int l = N/I - 2; l >= 0; --l) {
            delay_buffer[l + 1] =
delay_buffer[l];
        }
        delay_buffer[0] = x[i];
    }
    // seros inserting
    int j = 0;
    for(int i = 0; i < n + n; i = i + 2) {
        y[i] = comb[j];
        y[i + 1] = 0;
        ++j;
    }
    //integrator
    for (int j = 0; j < 2 * n - 1; ++j) {
        y[j + 1] = y[j + 1] + y[j];
    }
}
```



Serial code on high speed processor

Massive Power Reduction

Massive increase in throughput

Network of Basic Block Execution Units

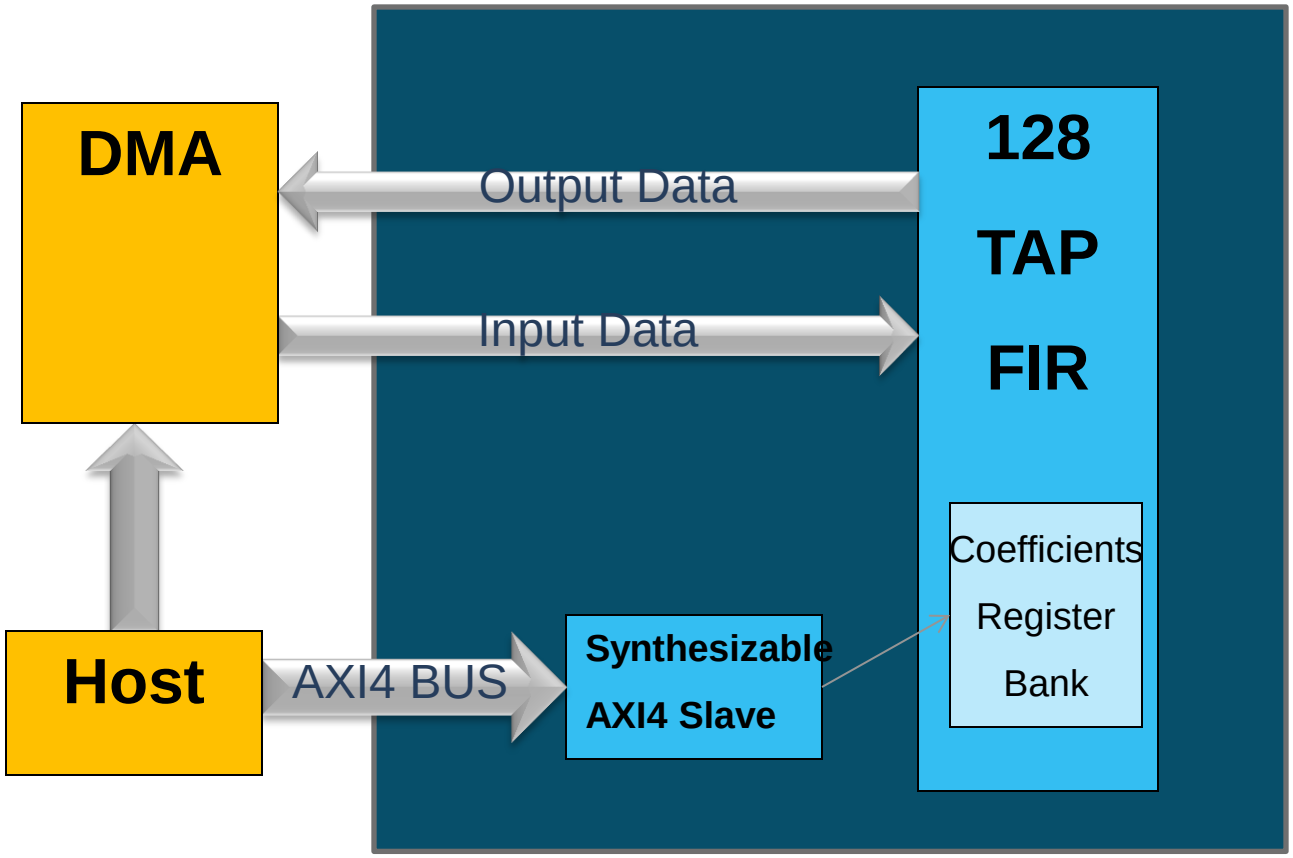
CPU: 1 GHz,

2.5W

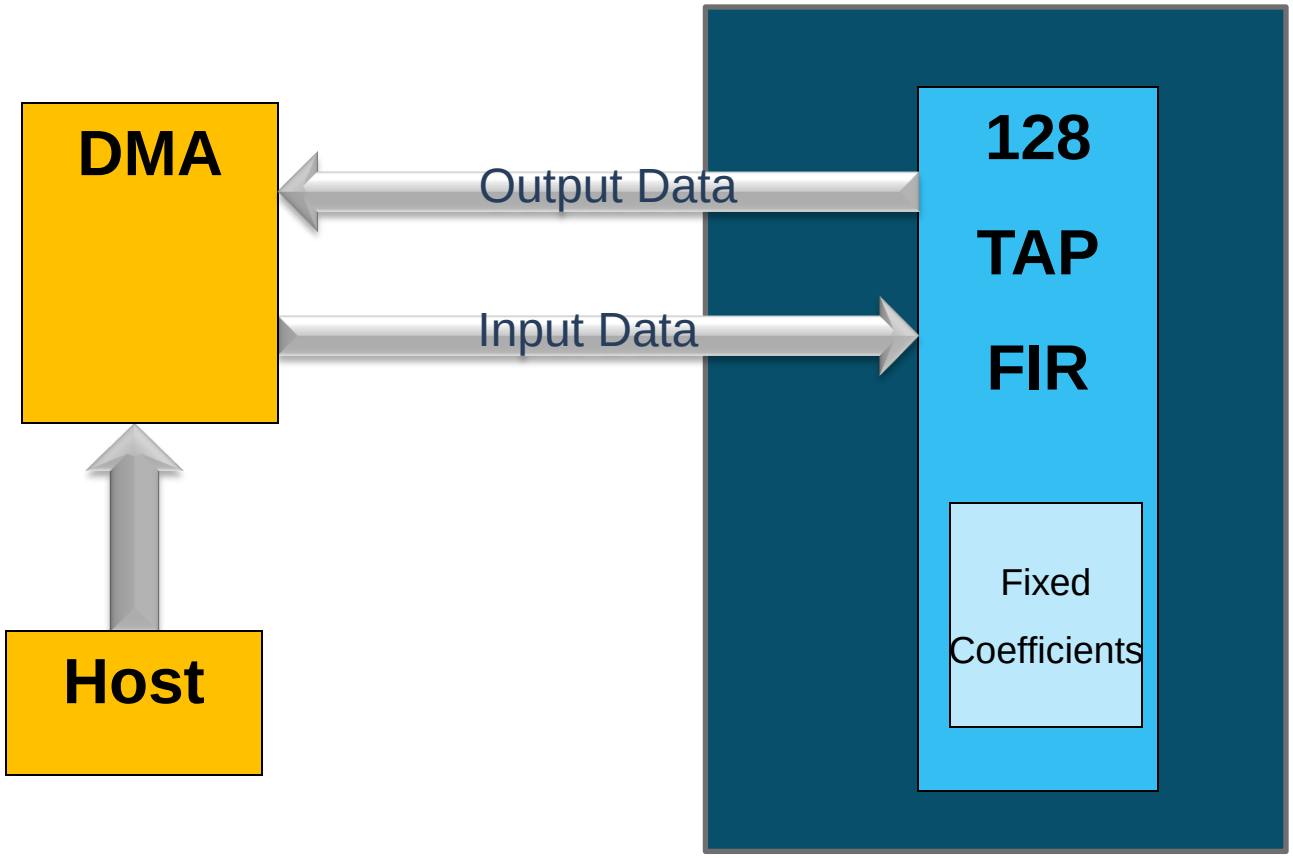
Basic Block processing elements:

80mw.

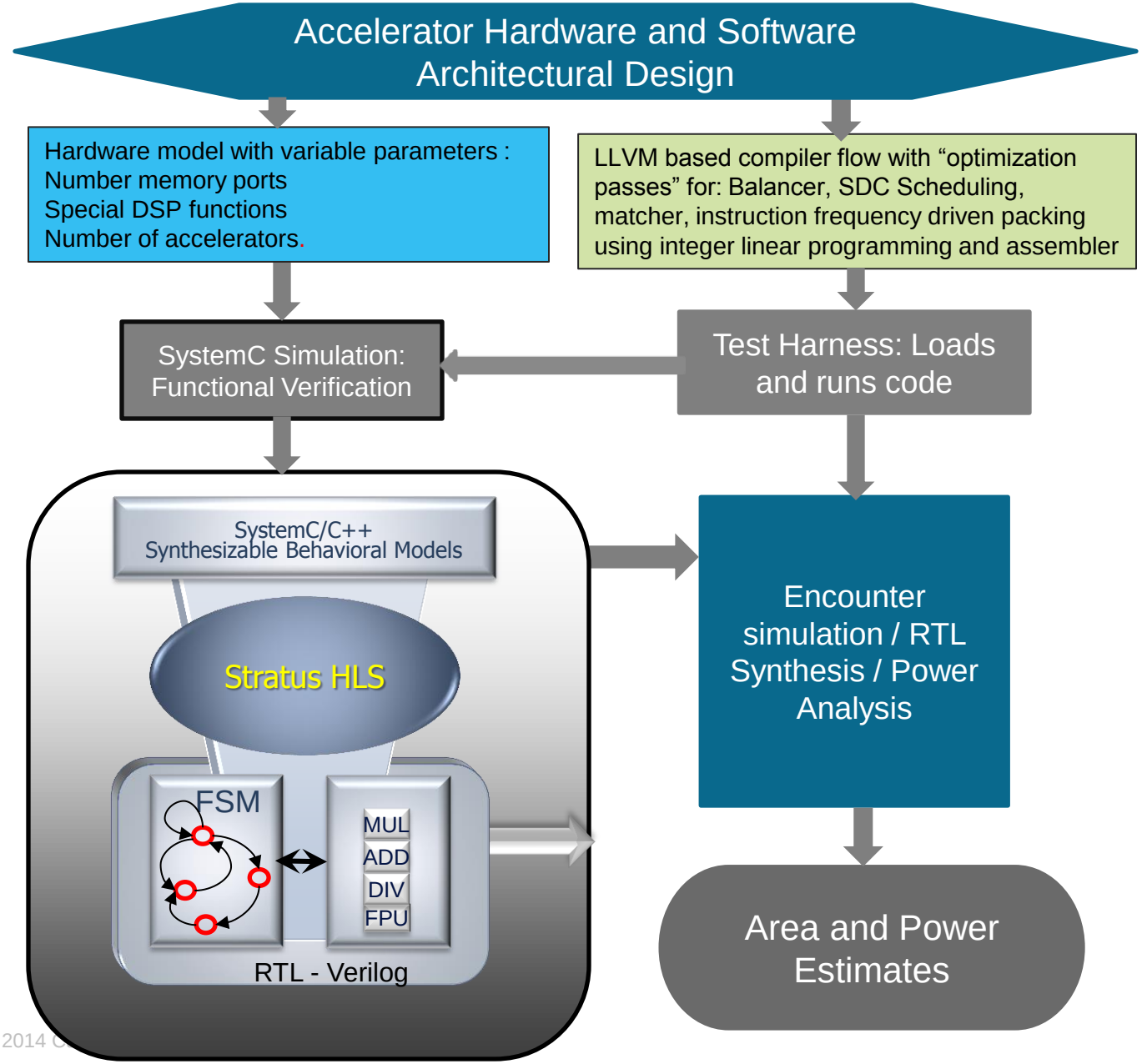
Configurable fixed function accelerator



Hard coded accelerator



Architectural Evaluation Flow



Realizing the ISA with System C: Dispatch model for execution unit

```
While(1){
  PIPELINE_MACRO;
  Instruction = instr.get();
  eop_code = Instruction.eop_code;

  switch(eop_code){
    case FIREVER: {
      HandleFirever(earg0);
      break;
    }
    case FIREVERIN: {
      HandleFireverin(earg0);
      break;
    }
    case SAD: {
      HandleSad();
      break;
    }
}
```

```
void alu::HandleFirever(sc_uint<DATA_WIDTH> arg0){

#ifdef ALU_TRACE
  std::cout << "FIREVER " << " " << arg0 <<
std::endl;
#endif

  sc_uint<DATA_WIDTH> immediate = arg0;
  acc_out.write(accum);
  accum = 0;
  if (counter == immediate) {
    counter = 0;
    inc_pc = 1;
  } else {
    inc_pc = 0;
    sc_uint<COUNTER_WIDTH> c = counter;
    sc_uint<ACCUM_WIDTH> a = accum;
    sc_biguint<DATA_WIDTH * 5> coeff = getCoeff(c);
    sc_int<DATA_WIDTH> data = getData(c);
    accum = a + coeff * data;
    acc_out.write(accum);
    ++c;
    counter = c;
  }
}
```

Hw Experiments

- Flow: System C -> Verilog -> RTL Compiler / Incisive

To Determine:

- Number of Accelerators
- Cost of Memory Interface
- Cost of advanced DSP instructions
- Cost of exposing accumulator

500 MHz constraint time.

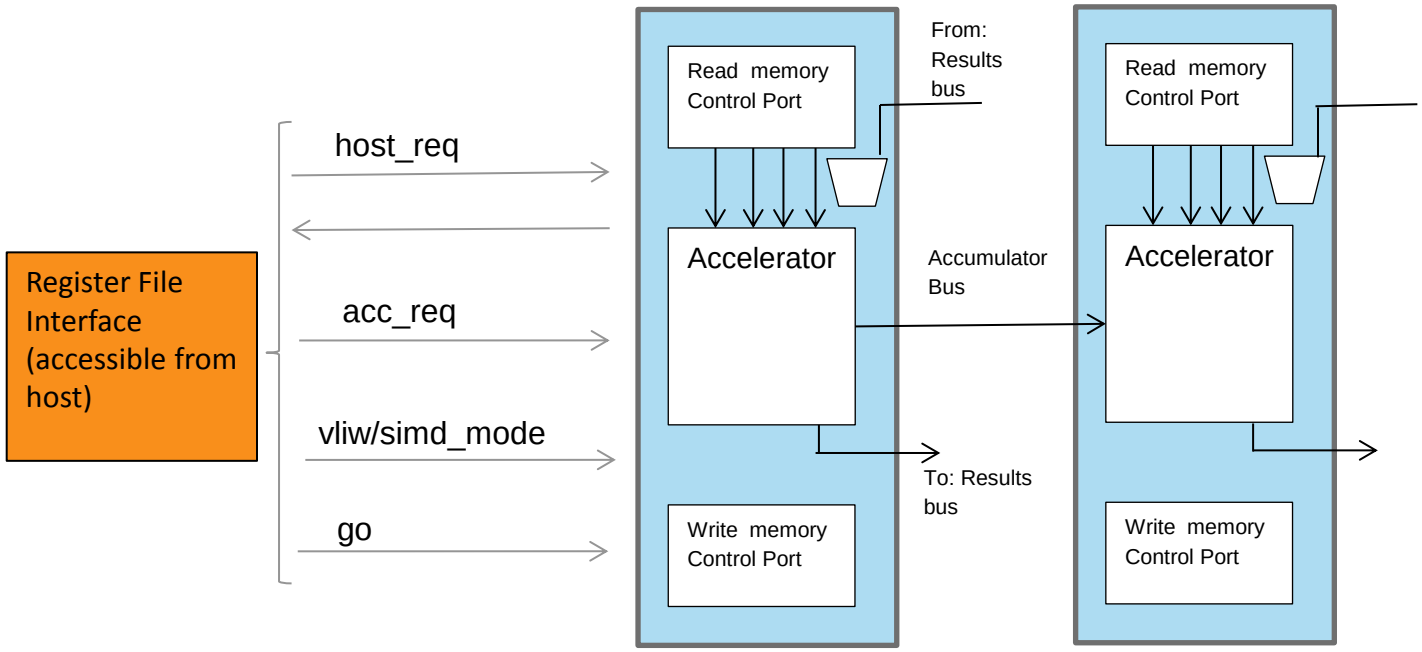
Estimates only (rc, and application of vectors to get power).

PPA results for DSP fabric

Design	Power (mw)	#Accelerators	Area (mm*2)
16GSPS 128 tap	4000	208	126
Turbo	152	18	5.2
Viterbi K=8	24	6	1.48
DCT 8x8	28.12	8	1.94
Smith Waterman	24	6	1.58
Non programmable Turbo (same code as Turbo above)	*7.61mw.*	N/A	0.21

Notes on results:
Scaling factors used to normalize to 28nm from 45nm
Power Scaling: 0.66, freq scaling: 1.34
Area: 0.62.
Pre-placement and routing
Used rc compiler from Cadence

Programmable Accelerator: BBPE Final Architecture



PPA tradeoffs for 3 different architectures

Implementation	Power (mw)	Flexibility	Area (mm*2)
Fully Programmable	4,000	Can implement any DSP function	126
Partially Programmable	1,542	Implements a specific function for multiple applications	5.9
Hard coded	929	Application specific	4.4

Some things just best said in “C” eg least mean squares filter.

```
void lmsq16_8(uchar u0l, uchar u0h,…
  static short u[8] = {0};
    static short w[8] = {0};
    int i;
    int ys = 0;
   //set up input data
    u[0] = u0h << 8 | u0l; // instant assignment
 for (i=0;i<8;i++){
        ys += (u[i]*w[i]); //value immediately ready
}
```

Designer thinks in terms of application.

Verilog

```
always @(posedge CLK)
     begin
        if (~RST_n)
          begin
             mem_ul0_q<= #1    8'h0;
              mem_uh0_q <=#1 8'h0;
        end
else
  begin
            mem_ul0_q <= #1 u0l;
            mem_ul0_q<= #1 u0h;

  ..
assign ys0 = signed_mult2(u0h,u0l, mem_wh0_q,mem_wl0_q);
```

Designer thinks in terms of “implementation state”

Conclusions

- Generated power/area numbers in timely manner (< 3 mos!). System “C” enabled successful evaluation.
- System C model not fast enough for debugging applications (we chose p-threaded model instead). This is based on execution of instructions (not pipe-stall accurate).


cadence®

Back up slides (Data tables)

A note on simulation times

Case	#instructions	Internal simulator	SystemC simulator	P-Threads with compiled code*
Single Accelerator test (all others asleep)	0.26M	4 sec	0.9s	<0.001
100 Accelerator test (all same program)	26M	84s(312K instr/second) (~ 3 us per instruction)	22s (1192K instr/second) (~ 1us per instruction)	0.024s(13M instr/second) (compiled code)

- Serious problem for us: Wanted to use same model for software development. Why is SystemC so slow ?
- Tried: Packing of signals into a struct to avoid signal updates.
- Fundamental issue: SC_THREAD's don't run in parallel threads !

Hw Evaluation Results

Design	Power (mw)	#Accelerators	Area (mm*2)
16GSPS 128 tap	4000	208	126
Turbo	152	18	5.2
Viterbi K=8	24	6	1.48
DCT 8x8	28.12	8	1.94
Smith Waterman	24	6	1.58
Non programmable Turbo (same code as Turbo above)	*7.61mw.*	N/A	0.21

Notes on results:
Scaling factors used to normalize to 28nm from 45nm
Power Scaling: 0.66, freq scaling: 1.34
Area: 0.62.

Pre-placement and routing
Used rc compiler from Cadence

cadence®

Results Highlights: Max/Min Concurrency

Basic Blocks	Accelerator Frequency	Instruction Count	Invocation count	Latency (2,4 ports)	Parallelism
CIC					
BB11	100	9	20	4,4	2.2
BB1	200	4	21	3,3	1.3
FFT					
BB5	500	7	10	4,4	1.75
BB1	1000	12	496	9,9	1.3
Turbo					
BB4	500	36	64	14,14	2.6
BB12	1000	11	384	8,8	1.4
DCT					
BB6	100	111	64	33,31	3.4
BB1	300	4	8	3,3	1.3

CIC

Basic Block	Accelerator Frequency	Number of Instructions	Invocation Count	Latency 2,4 memory ports	Average number of instructions in parallel per cycle
BB1	200	4	21	3,3	1.3
BB2	300	4	21	3,3	1.3
BB3	200	4	40	3,3	1.3
BB4	200	4	20	3,3	1.3
BB5	400	2	20	1,1	2
BB6	100	7	20	5	1.4
BB7	200	4	100	3,3	1.3
BB8	400	2	20	1,1	2
BB9	100	8	80	5,3	1.6
BB10	300	3	80	1,1	3
BB11	100	9	20	4,4	2.2
BB12	300	2	20	1,1	2
BB13	100	8	39	5,5	1.6
BB14	300	2	39	1,1	2

CIC

Basic Block	Accelerator Frequency	Number of Instructions	Invocation Count	Latency 2,4 memory ports	Average number of instructions in parallel per cycle
BB1	200	4	21	3,3	1.3
BB2	300	4	21	3,3	1.3
BB3	200	4	40	3,3	1.3
BB4	200	4	20	3,3	1.3
BB5	400	2	20	1,1	2
BB6	100	7	20	5	1.4
BB7	200	4	100	3,3	1.3
BB8	400	2	20	1,1	2
BB9	100	8	80	5,3	1.6
BB10	300	3	80	1,1	3
BB11	100	9	20	4,4	2.2
BB12	300	2	20	1,1	2
BB13	100	8	39	5,5	1.6
BB14	300	2	39	1,1	2

CIC

Basic Block	Accelerator Frequency	Number of Instructions	Invocation Count	Latency 2,4 memory ports	Average number of instructions in parallel per cycle
BB1	200	4	21	3,3	1.3
BB2	300	4	21	3,3	1.3
BB3	200	4	40	3,3	1.3
BB4	200	4	20	3,3	1.3
BB5	400	2	20	1,1	2
BB6	100	7	20	5	1.4
BB7	200	4	100	3,3	1.3
BB8	400	2	20	1,1	2
BB9	100	8	80	5,3	1.6
BB10	300	3	80	1,1	3
BB11	100	9	20	4,4	2.2
BB12	300	2	20	1,1	2
BB13	100	8	39	5,5	1.6
BB14	300	2	39	1,1	2

DCT

Basic Block	Acc freq	Instr Count	Invoc count	Latency: 2,4	Parallelism
BB1	300	4	8	3,3	1.3
BB2	200	12	8	8,8	3
BB3	100	111	64	32,30	3.4
BB4	300	10	8	4,4	2.5
BB5	200	42	8	22,22	1.9
BB6	100	111	64	33,31	3.4

Sw Results Findings

- 1 memory port per operand
- 2 instructions in parallel per Basic Block

Memory Interface Costs

- Two ports per operand

Instance	Cells	Cell Area	Net Area	Wireload
accelerator	349435	1413514	3795369	5K_hvratio_1_1 (D)

Significant increase in area.

Software results for increasing memory port count per operand show only modest reduction in latency:

Basic Block	Accelerator Frequency	Instruction Count	Invocation Count	Latency 2,4 memory ports	Number of Instructions in Parallel
DCT:					
BB3	100	111	64	32,30	3.4
BB5	200	42	8	22,22	1.9

Decision: keep at one memory port per operand

Cost of Advanced DSP operations

- Base line: (no dsp, no accumulator bus): experiment fabricnodsp

- Instance Cells Cell Area Net Area Total Area Wireload
- ---
- accelerator * * 2391895 5K_hvratio_1_1 (D)

- Including FIRever, SOD, SAD (dsp).
  - Experiment fabricdsp

- Instance Cells Cell Area Net Area Total Area Wireload
- ---
- accelerator * * 2429029 5K_hvratio_1_1 (D)

Accumulator Bus Cost

- With 40 bit accumulator bus (allows propagation of unsaturated results – needed for dsp): Experiment fabricdsp_accbus

Instance	Cells	Cell Area	Net Area	Total Area	Wireload
accelerator	174572	707663	1689725	2397388	5K_hvratio_1_1 (D)

  - Without:

Neglibile difference (330 instances)
Decision: keep accumulator bus. (treated as base in following)