Architectural Evaluation Of a Programmable Accelerator For Baseband, Phy and Video Applications Using High Level Synthesis

Andy Fox, Tigran Sargsyan, Steven Anderson

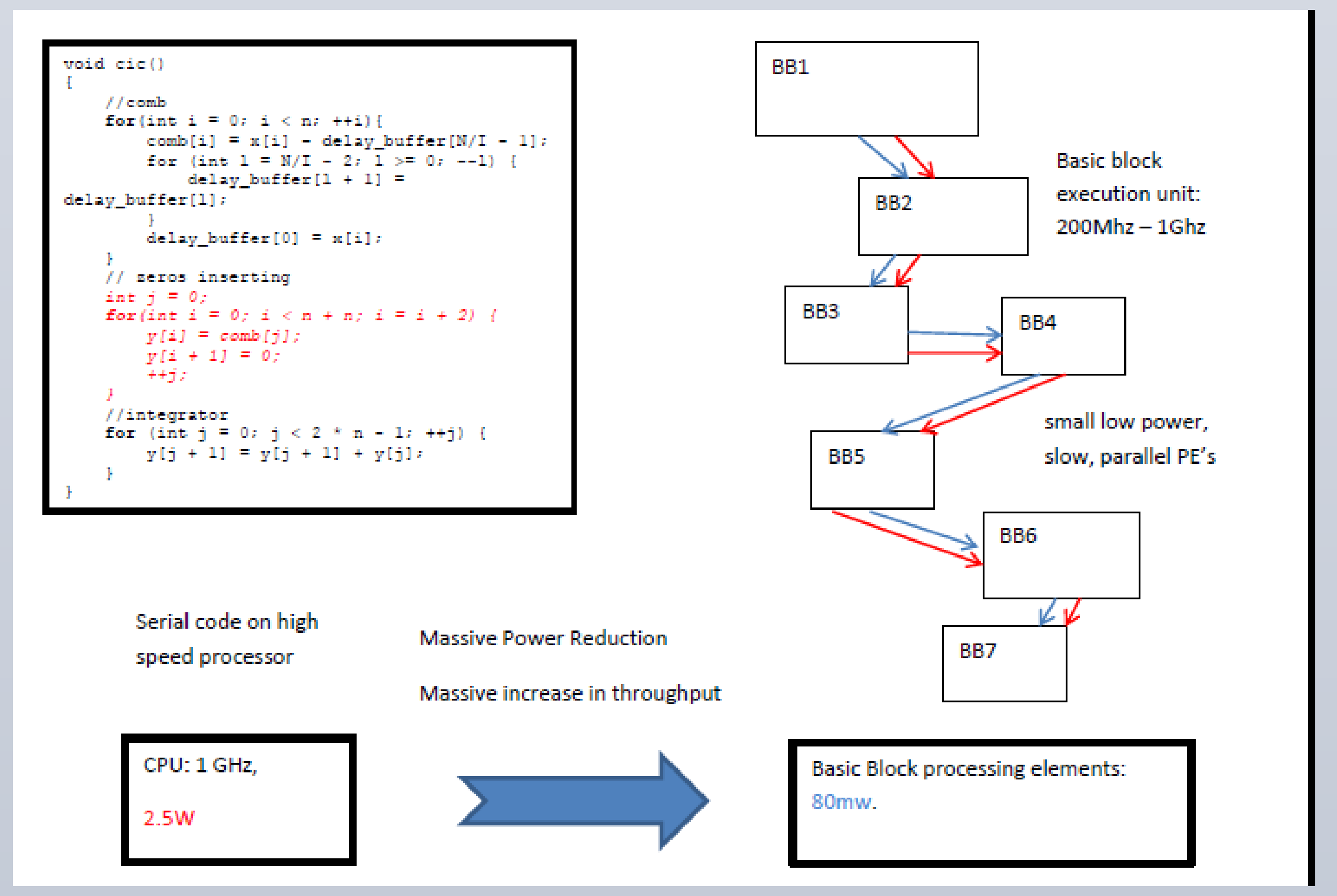
RUSHC, Forte Design Systems.

Abstract

- The goal of this project was to define an optimal architecture for a programmable hardware accelerator for processing common DSP functions.
- A software tool chain was developed in parallel with the hardware.
- The target applications such as trellis decoders, FFT's, FIR's, expressed in C were used as test cases to prove the value of the architectural choices.
- The hardware architecture was expressed as a SystemC model and a High Level Synthesis tool was used to generate RTL.
- The generated RTL was used to generate results (power, area) and then iterate with the architectural SystemC model.
- Questions to be answered:
  - Can we use a SystemC model as basis for developing designs for DSP fabric?
  - How effective is High Level Synthesis for this type of design activity?
  - What is the cost of programmability?
  - What are the optimal accelerator parameters?
  - Can we code the SystemC models in a natural C style and still get good results in the generated RTL?
  - Can we use the SystemC model as basis for applications simulator for software development?

Objectives

- Design a DSP fabric to achieve power reduction with increased throughput as shown below.



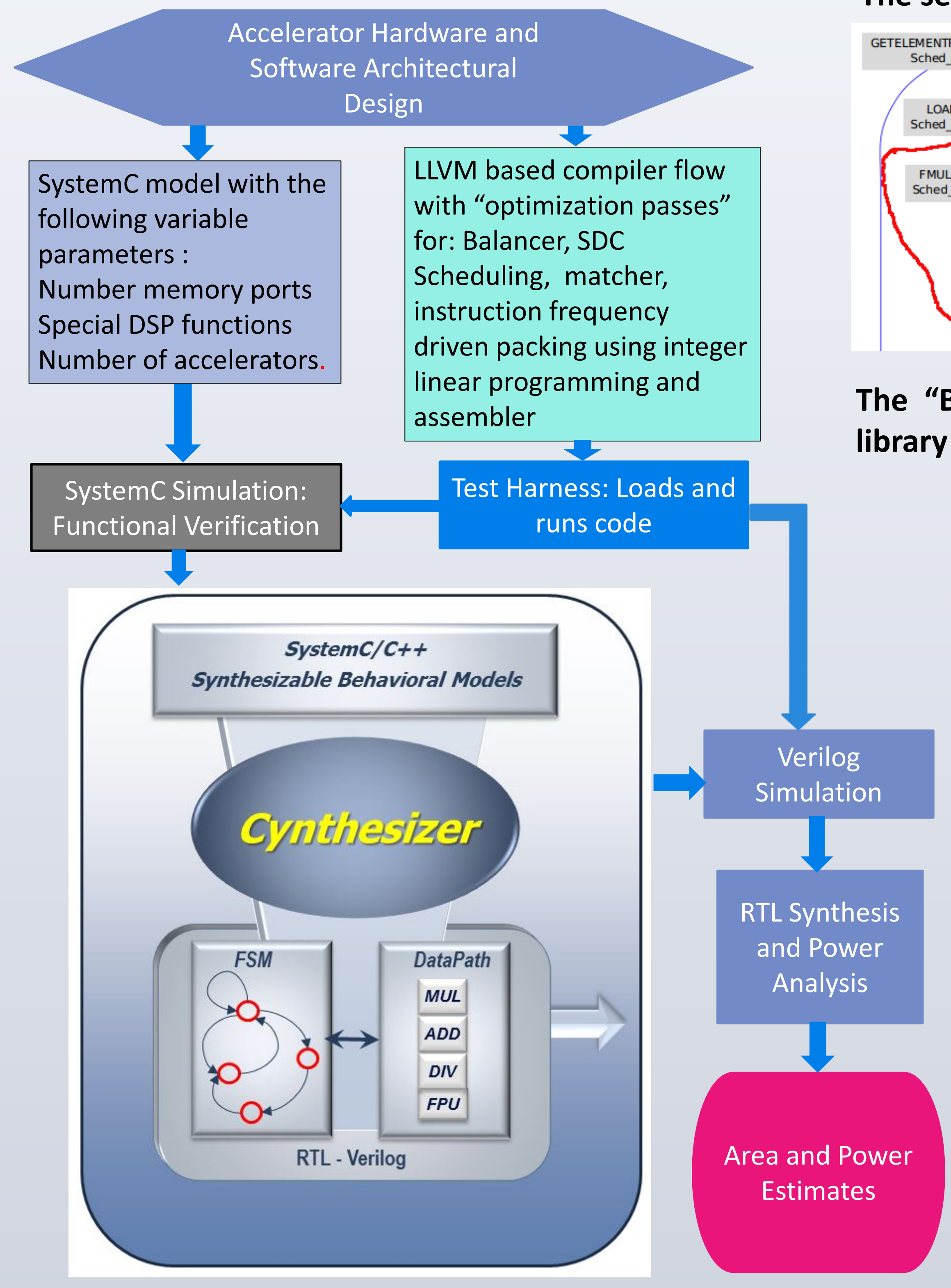
```
void cic()
{
    //comb
    for(int i = 0; i < n; ++i){
        comb[i] = x[i] - delay_buffer[N/I - 1];
        for (int l = N/I - 2; l >= 0; --l) {
            delay_buffer[l + 1] =
delay_buffer[l];
        }
        delay_buffer[0] = x[i];
    }
    // zeros inserting
    int j = 0;
    for(int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

BB1
BB2
BB3
BB4
BB5
BB6
BB7
Basic block execution unit: 200Mhz – 1Ghz
small low power, slow, parallel PE's
Serial code on high speed processor
Massive Power Reduction
Massive increase in throughput
CPU: 1 GHz, 2.5W
Basic Block processing elements: 80mw.

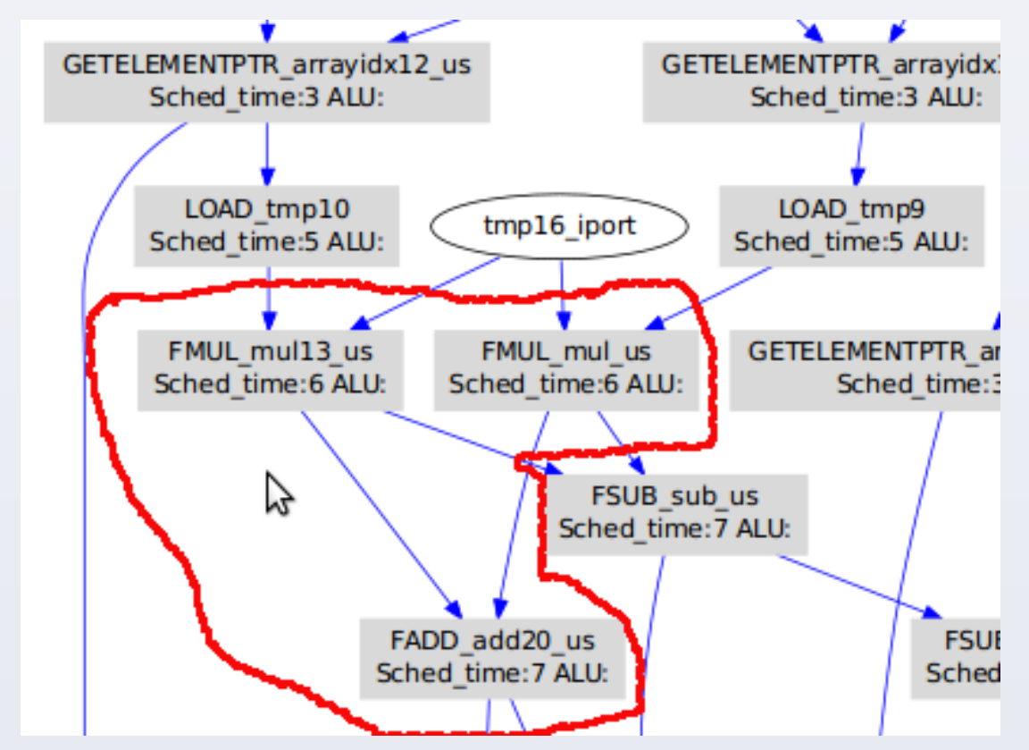
- Schedule: 3 months to complete the evaluation.
- Architectural parameters:
  - Number of parallel execution units
  - Propagation of unsaturated results between accelerators
  - Number of memory ports
  - Cost of advanced DSP instructions for FIR, Sum of absolute differences (SAD), Butterfly combinations.
- The decision criteria: Area and Power as measured by Cadence RTL design tools.
- Start from C code version of applications then measure:
  - Number Accelerators needed and their frequency
  - Instruction parallelism per basic block
  - Memory bandwidth (number of memory ports)
- Gauge cost of programmability for one of our applications (run a case through to ASIC implementation).

Materials and Methods

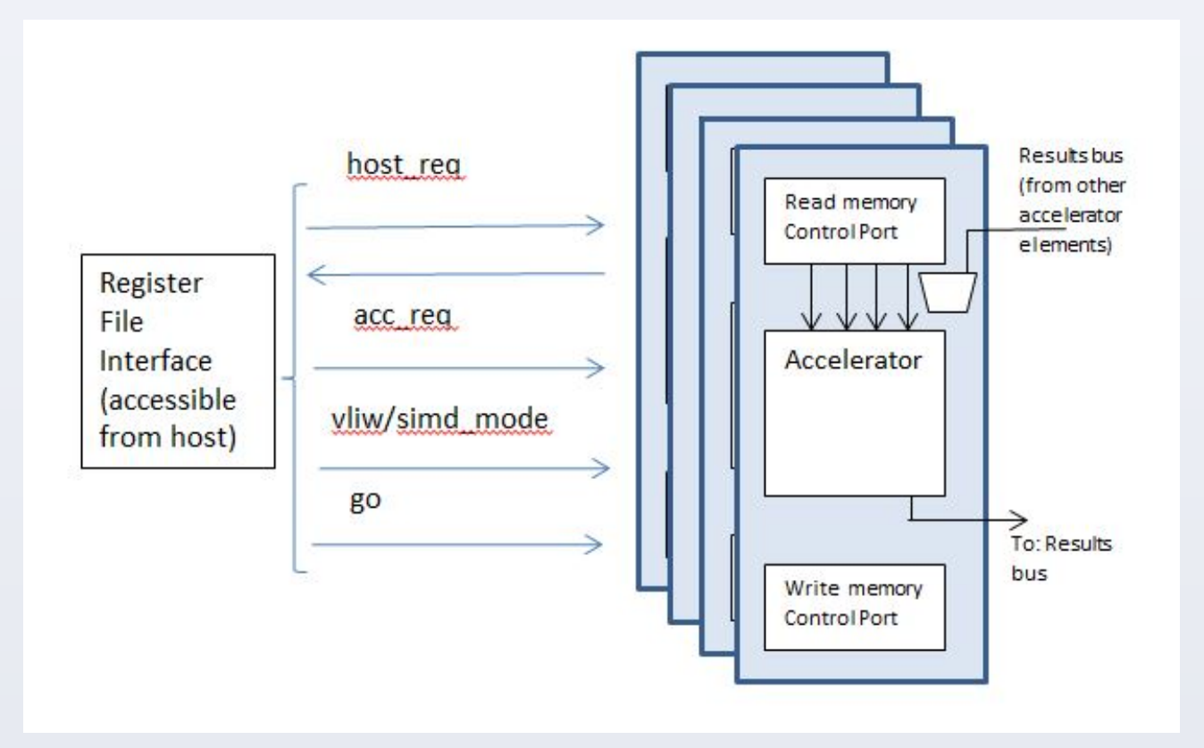
Evaluation Flow


Accelerator Hardware and Software Architectural Design
SystemC model with the following variable parameters : Number memory ports Special DSP functions Number of accelerators.
LLVM based compiler flow with "optimization passes" for: Balancer, SDC Scheduling, matcher, instruction frequency driven packing using integer linear programming and assembler
SystemC Simulation: Functional Verification
Test Harness: Loads and runs code
SystemC/C++ Synthesizable Behavioral Models
Cynthesizer
FSM
DataPath
MUL
ADD
DIV
FPU
RTL - Verilog
Verilog Simulation
RTL Synthesis and Power Analysis
Area and Power Estimates

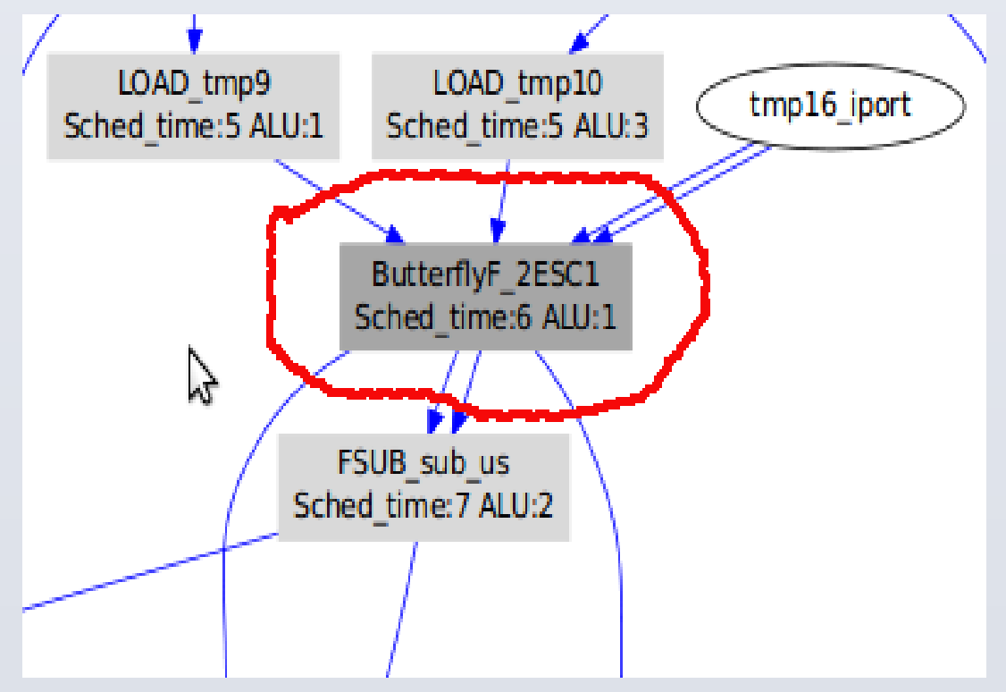
The selected part is "Butterfly"


GETELEMENTPTR_arrayidx12_us Sched_time:3 ALU:
GETELEMENTPTR_arrayidx Sched_time:3 ALU:
LOAD_tmp10 Sched_time:5 ALU:
tmp16_iport
LOAD_tmp9 Sched_time:5 ALU:
FMUL_mul13_us Sched_time:6 ALU:
FMUL_mul_us Sched_time:6 ALU:
GETELEMENTPTR_ar Sched_time:
FSUB_sub_us Sched_time:7 ALU:
FADD_add20_us Sched_time:7 ALU:
FSUB Sched

Initial Architecture


Register File Interface (accessible from host)
host_req
acc_req
vliw/simd_mode
go
Read memory Control Port
Accelerator
Write memory Control Port
Results bus (from other accelerator elements)
To: Results bus

The "Butterfly" is matched and covered by library element "Butterfly_2ESC"


LOAD_tmp9 Sched_time:5 ALU:1
LOAD_tmp10 Sched_time:5 ALU:3
tmp16_iport
ButterflyF_2ESC1 Sched_time:6 ALU:1
FSUB_sub_us Sched_time:7 ALU:2

Declarations

```
sc_int<DATA_WIDTH>
                    idata[IDATA_SIZE];
sc_biguint<DATA_WIDTH*5>
                     imem[IMEM_SIZE];
sc_uint<DATA_WIDTH>
                     immediate;
sc_uint<COUNTER_WIDTH>    address;
sc_uint<ACCUM_WIDTH>      a;
sc_biguint<DATA_WIDTH*5>  coeff;
sc_int<DATA_WIDTH>        data;
```

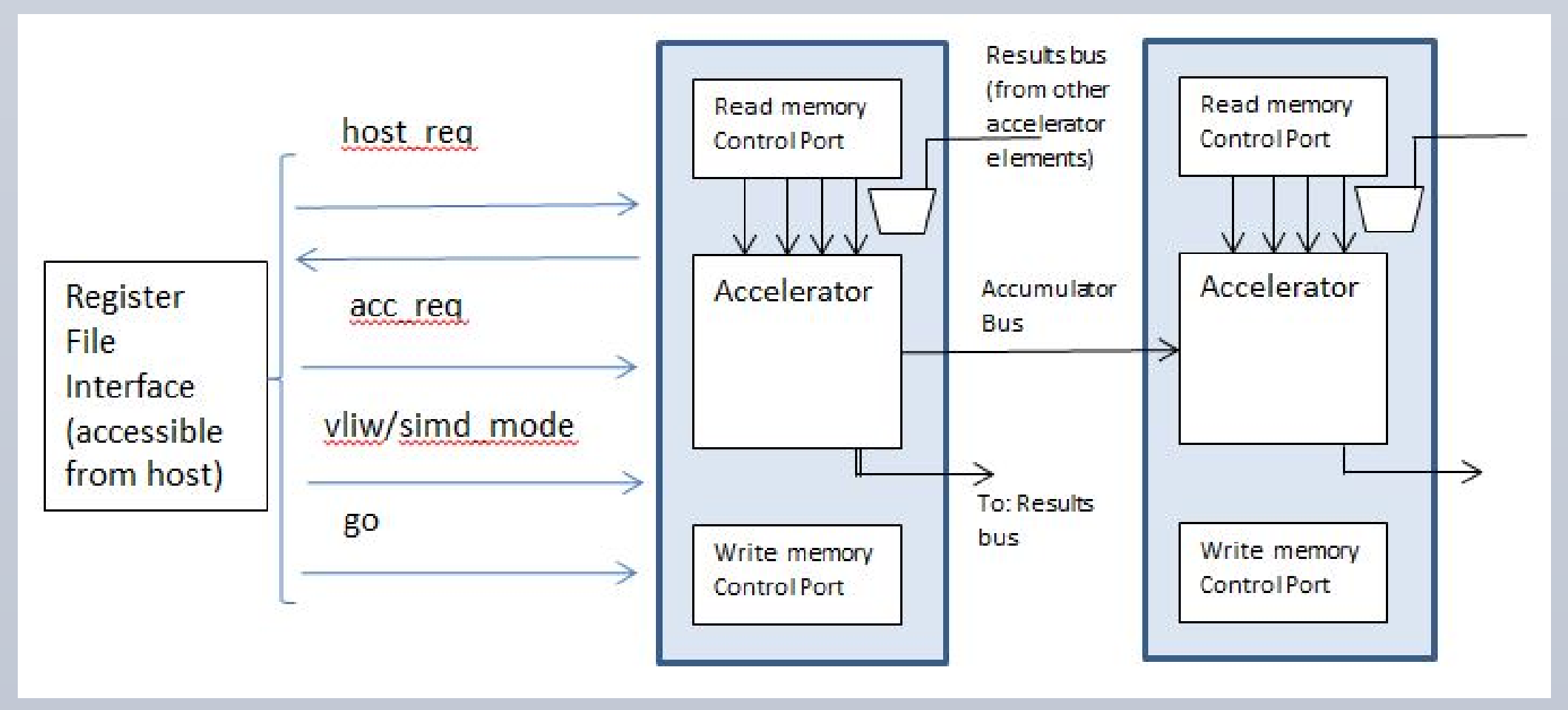
Dispatch thread

```
CYN_ARRAY_MEMORY(idata);
CYN_ARRAY_MEMORY(imem);
while(1){
 CYN_INITIATE();
 CYN_LATENCY();
 switch(eop_code){
   case FIREVER: {
     HandleFirever(earg0);
     break;
   }
   case FIREVERIN: {
     HandleFireverin(earg0);
     break;
   }
   case SAD: {
     HandleSad(earg0);
     break;
    }
    case....
 }
}
```

Autonomous FIR instruction

```
void alu::HandleFirever
    (sc_uint<DATA_WIDTH> arg0){
  immediate = arg0;
  acc_out.write(accum);
  accum = 0;
  if (counter == immediate) {
    counter = 0;
    inc_pc = 1;
  } else {
    inc_pc = 0;
    address = counter;
    a = accum;
    coeff = imem[address];
    data = idata[address];
    accum = a + coeff * data;
    acc_out.write(accum);
    ++address;
    counter = address;
  }
}
```

Final Architecture


Register File Interface (accessible from host)
host_req
acc_req
vliw/simd_mode
go
Read memory Control Port
Accelerator
Write memory Control Port
Results bus (from other accelerator elements)
Accumulator Bus
To: Results bus
Read memory Control Port
Accelerator
Write memory Control Port

Simulation performance

Case	#instructions	Internal simulator	SystemC simulator	Pthread's 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)

* Instructions executed directly, not interpreted nor decoded (pipeline not modeled). This is "application" simulation (instructions translated directly to host machine)

Results

Software Experiments Min / Max results

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

Power and Area Results for final architecture

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
ASIC Turbo *	7.61	N/A	0.21

* Same C source code

Conclusions

- SystemC with generated Verilog allowed us to get accurate area and power estimates very quickly (we met our 3 month deadline).
- The combination of HLS technology with SystemC source code meant that we could use a very natural C coding style for all of the hardware models and still get optimized RTL.
- Unfortunately the SystemC models were not fast enough to be an effective virtual prototype for application development.
  - A Pthread model allowed much faster hardware simulation – fast enough to be used for application development.
- The overhead for using programmable accelerators was determined to be about 5x vs. a hard coded RTL implementation for one application (trellis decoder).
- The software tools were unable to effectively use more than 2 accelerators. Our automated flow could only find on the average 2 parallel instructions per basic block.
- 1 memory port per operand was sufficient for all of the applications we tested.
- The accumulator bus (40 bits) came with very low overhead and enabled ganging of accelerators.
- Advanced autonomous DSP instructions were relatively easy to define and evaluate using this design flow.
  - For example the autonomous fir instruction, which enables instruction to stream data from memory without incrementing pc and gets coefficients from Instruction memory, was easily expressed in SystemC and evaluated.
  - Several of these were included in the final implementation since they effectively reduced latency and power for our target application.

References

1. G. Venkatesh, J. Sampson, N. Goulding, S Garcia, V Brysin, J Lugo Martinez, S Swanson, M Bedford Taylor, *Conservation Cores: Reducing the Energy of Mature Computations*, Proceedings of the fifteenth edition of ASPLOS on Architectural Support for Programming Languages and Operating Systems.
2. H. Esmaeilzadeh, E. Blem, R Amant, K Sankaralingam, D Burger, *Dark Silicon and The End of Multicore Scaling*, Proc of the 38th International Symposium on Computer Architecture, 2011.
3. J. Cong, Z. Zhang *An Efficient and Versatile Scheduling Algorithm based on SDC formulation.* Proceedings of the 43rd Annual Design Automation Conference.
4. V Ganesh, D Dill, *A Decision Procedure for Bit-Vectors and Arrays.* Proc Computer Aided Verification 2007.
5. LLVM: An Infrastructure for Multi-Stage Optimization, C Lattner, MS Thesis, CS Dept, University of Illinois at Urbana Champaign, Dec 2002.
6. Nadav Rotem, Haifa University, Presentation titled "*High Level Synthesis Using LLVM*", http://llvm.org/devmtg/2010-11/Rotem-CToVerilog.pdf
7. Scott Mahkle et al, *Effective Compiler Support For Predicated Execution Using the HyperBlock.* Micro 25, Proceedings of the 25th annual international symposium on Microarchitecture, pages 45-54, 1992.
8. Aline Mello et al, "Parallel Simulation of SystemC TLM 2.0 Compliant MPSoC on SMP Workstations", DATE Conference, 2010.

Acknowledgements

We thank the anonymous referees for their constructive feedback and acknowledge the helpful discussions with Edvard Ghazaryan.

Contacts

Andy Fox, RUSHC, andy@rushc.com
Steven Anderson, Forte Design Systems, sanderson@forteds.com
Tigran Sargsyan, RUSHC, tigran@rushc.com