Ardbeg Vector Processor

Mladen Wilder
ARM Ltd

30 October 2008

Agenda

- About ARM
  - Very brief history of ARM
- Ardbeg vector processor
  - Wireless trends
  - Ardbeg vector processor
  - Ardbeg system programming tools

About ARM

ARM designs the technology that lies at the heart of advanced digital products, from wireless, networking and consumer entertainment solutions to imaging, automotive, security and storage devices.

- ARM's comprehensive product offering includes:
  - 32-bit RISC microprocessors
  - graphics processors
  - enabling software
  - ASIC cell libraries and embedded memories
  - high-speed connectivity products
  - peripherals
  - development tools.

History

- **1985** – Acorn Computer Group developed the world's first commercial RISC processor
- **1987** – Acorn's ARM processor debuts as the first RISC processor for low-cost PCs
- **1990** – Advanced RISC Machines (ARM) spins out of Acorn and Apple Computer's collaboration efforts with a charter to create a new microprocessor standard. VLSI Technology becomes an investor and the first licensee
- **1998** – ARM Partners shipped more than 50 million ARM Powered products
- **2002** – ARM announced that it had shipped over one billion of its microprocessor cores to date
- **2008** – ARM announces 10 billionth processor shipment

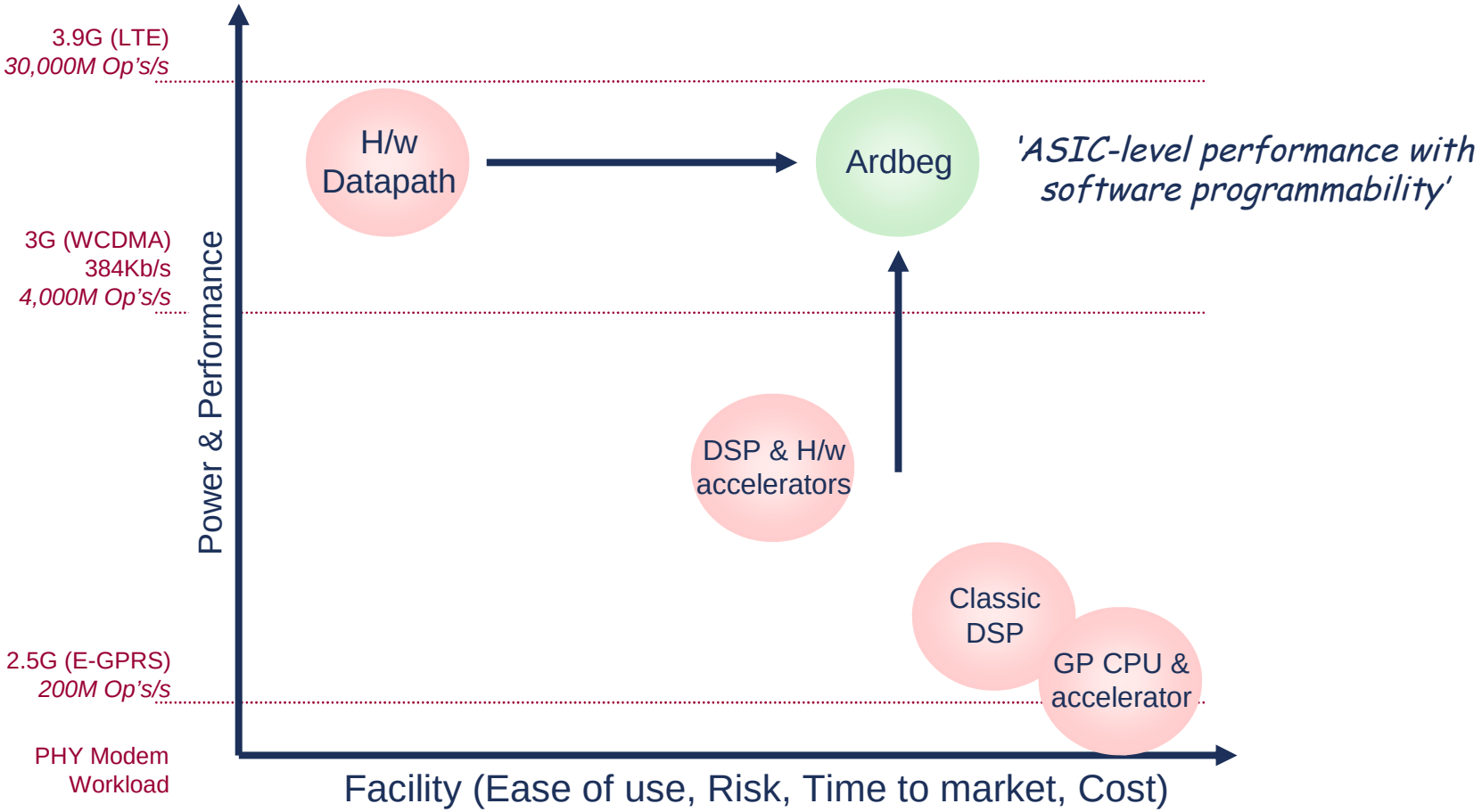
Ardbeg Vector Processor

Motivation

Industry trends

- Ever increasing numbers of radio standards
  - Cellular
  - Connectivity
  - Broadcast
- Multiple radios per handset driving up the BOM
- Wireless standards continuously evolve
  - This includes both standard changes and new (improved) algorithms
- High data rate standards require high performance
  - But mobile devices require low power
- Latest generation standards are typically implemented in hardware to meet performance and power requirements

A new component



Performance figures are approximate and represent part of physical layer processing

- Combining performance and energy efficiency of hardware datapath with software programmability

Ardbeg research goals

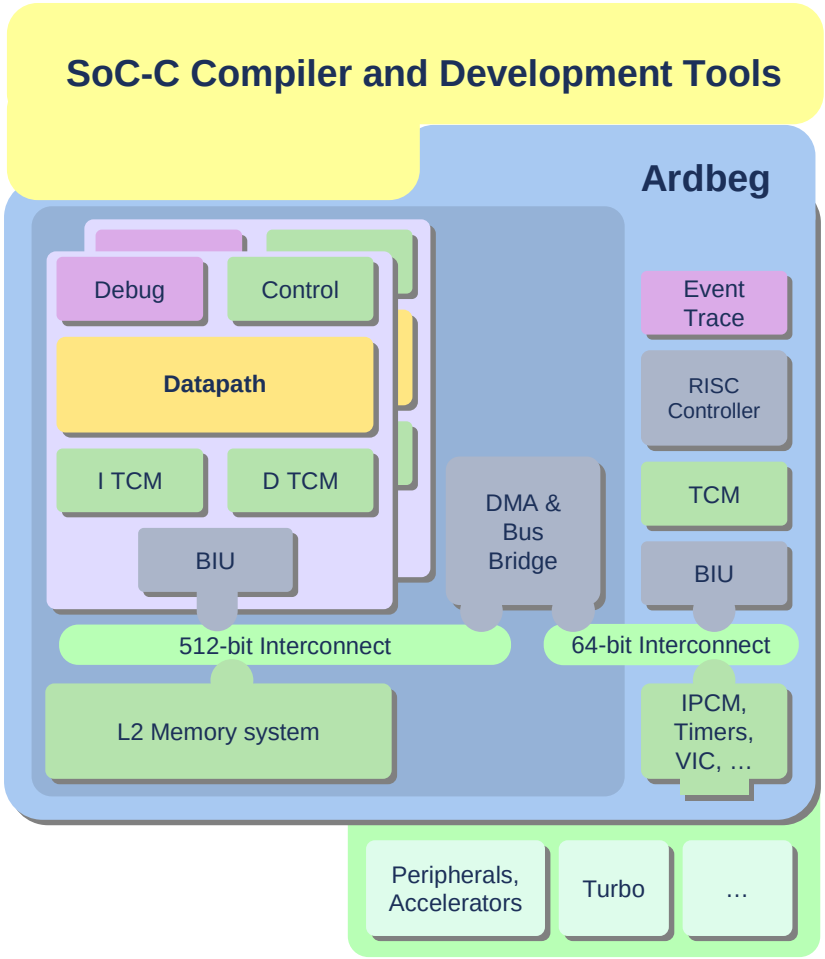
To investigate programmable solutions for high-performance and low-power signal processing applications.

- History
  - 2004 Funded research with UoM
    - Defined SODA architecture
  - 2005 Ardbeg central ARM R&D research program est.
    - Architecture and micro-architecture
    - Parallelising compiler
    - SOC compiler for system-level programming
    - SOC debug and visualisation tools

Ardbeg Vector Processor

R&D prototype

Ardbeg System



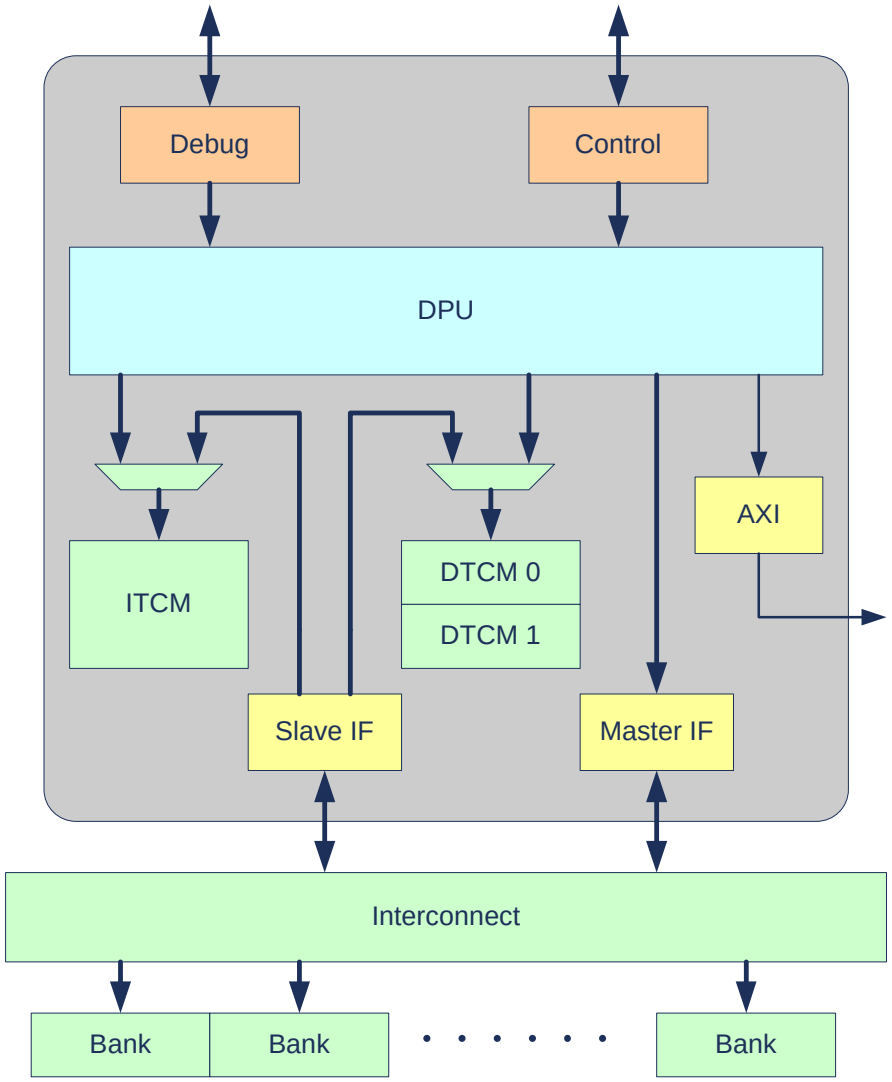
Tools to provide a simple programmers view

- C development flow

Ardbeg Vector Processor

- Control Data Decoupled Architecture
- Control plane
  - RISC control core
  - DMA, Timers, I/O control …
  - Debug
- Data plane – vector processor
  - One or more Ardbeg data engines
  - Hierarchical memory system
  - Dedicated DMA controller

Ardbeg Data Engine



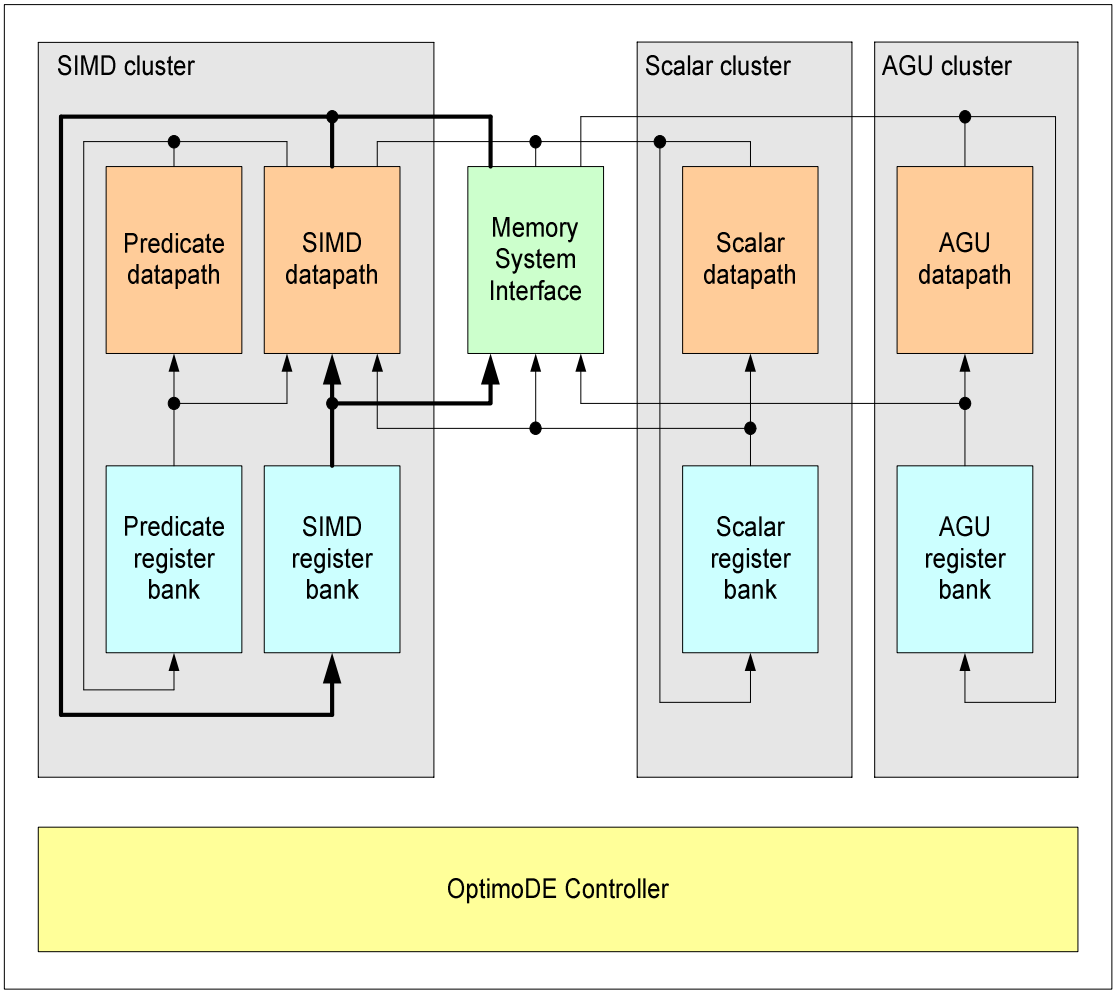
DE consists of:

- 512-bit vector DPU
  - VLIW architecture
- L1 memory system
  - Instruction and data TCMs
- L2 memory system interfaces
  - One 512-bit master port
  - Two 512-bit slave ports
- AXI master port
  - 32-bit port for IO
- Control and debug ports

Optimised for energy efficiency

- Simple pipeline and control logic
- Small L1 memory (32-64 kB)
- Clock and power gating

Ardbeg DPU – logical diagram



Built using OptimoDE technology

512 bit wide SIMD

- 64 x 8-bit concurrent operations
- 32 x 16-bit concurrent operations
- 16 x 32-bit concurrent operations
- Block floating point support
- SIMD permutation functional unit
- 15 x 512 bit registers in SIMD register file
- 15 x 64 bit registers in SIMD predicate register file

AGU cluster

- Address generation
- Loop control

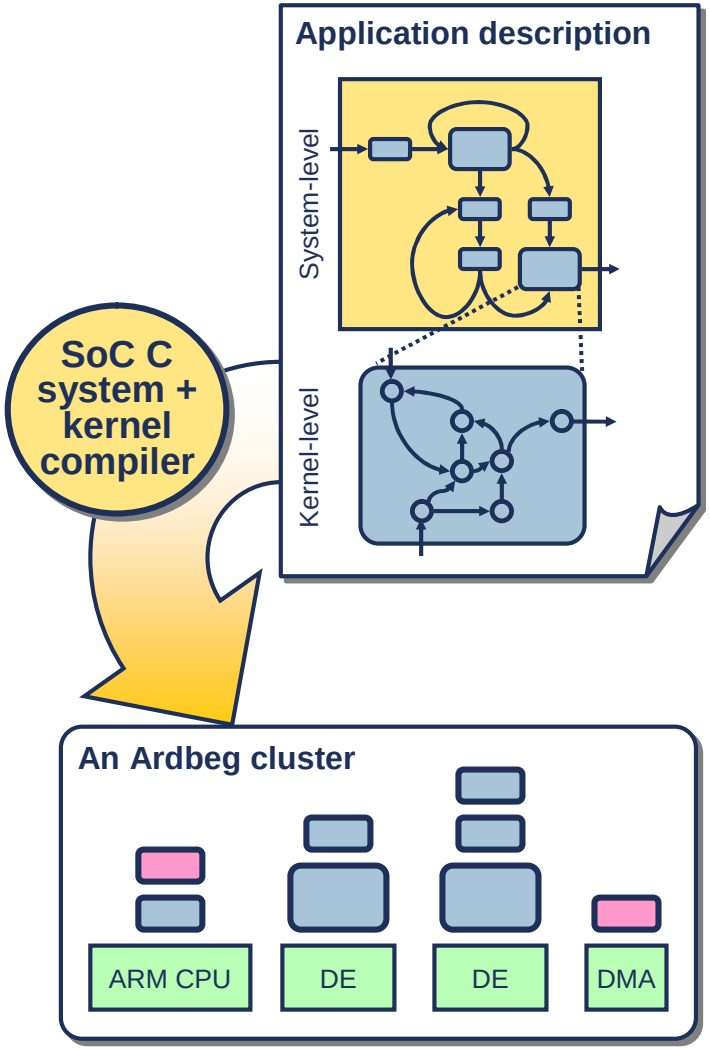
Parallel execution

- Concurrent SIMD, scalar, AGU and memory operations in one instruction.

Fully compiler exposed μ-architecture

- Leads to efficient hardware implementation
- Compiler scheduled operations reduce hardware complexity

Programming the Ardbeg cluster



Two level application description

- **System-level**: describes inter-kernel communications, timing constraints, mapping attributes, assertions

  Concurrent tasks extracted from “C + channels + attributes” description
- **Kernel-level**: “C + primitives + vectors”

System compilation

- Generates tasks, schedules, communication stubs, DMA requests, timing assertions, synchronization, debug support
- Iterative compilation: marks up program with increasingly detailed attributes; fully attributed program required for final mapping

Kernel compilation (i.e. DE programming)

- Standard “C + primitives + vectors” converted to machine code

DE (kernel) programming

- Ardbeg DE Instruction Set Architecture (ISA) is defined as a set of C data types and primitives
  - NEON™ based ISA
  - SIMD, scalar and predicate data types used for data processing
  - Integer data types used for control and memory access
  - SIMD width and data type width are implementation defined
- Kernel development in C/C++ using the OptimoDE tools
  - Compiler performs register allocation, instruction scheduling, etc.
  - Micro-architecture is fully exposed to the compiler
  - Programmer not exposed to Binary Instruction Format
  - Leads to efficient hardware implementations
- Software development environment
  - Optimising C compiler with profiling information
  - ISS model
  - Debugger

Vector data types

- Vector data types, shown for 128-bit wide vector
  - 8, 16 and 32-bit element support

127															0	bit number
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	vint8_t elements
7		6		5		4		3		2		1		0		vint16_t elements
3				2				1				0				vint32_t elements

- Predicate vector – 1, 2 or 4 bits per vector element

15															0	bit number
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	vbool8_t elements
7		6		5		4		3		2		1		0		vbool16_t elements
3				2				1				0				vbool32_t elements

- Accumulator vector – support for 16-bit multiplies only

255								0	bit number
7	6	5	4	3	2	1	0	vint32L_t elements	

Architecture - primitives

- Ardbeg primitives (operations) can be broadly grouped into
  - Arithmetic operations (unary, binary, conditional, fused)
  - Logic operations
  - Multiply operations
  - Permute and data transfer operations
- There are a relatively small number of basic primitives
  - Goal is energy efficiency and high-performance
    - most commonly used operations
  - Some primitives have several variations
    - added functionality, such as block-float
    - improved performance, such as pair-wise operations

Programming example - FIR

```
void fir_simd(
    const vint16_t in[(SIZE+TAPS+ELEMENTS16-1)/ELEMENTS16],
    vint16_t       out[SIZE/ELEMENTS16],
    const int16_t  h[TAPS])
{
#pragma OUT out
    vint16_t v;
    vint16_t w = in[0];
    int i,j;
    vint32L_t acc;

    for (i=0; i<SIZE/ELEMENTS16; ++i) {
        v = w;
        w = in[i+1];
        acc = vqdmull_n_s16(v,h[0]);
        v = vdown_n_s16(v,vget_lane_s16(w,0));

        for (j=1; j<TAPS-1; ++j)
        {
            acc = vqdmlal_n_s16(acc,v,h[j]);
            v = vdown_n_s16(v,vget_lane_s16(w,j));
        }
        out[i] = vqrdmlah_n_s16(acc,v,h[TAPS-1]);
    }
}
```

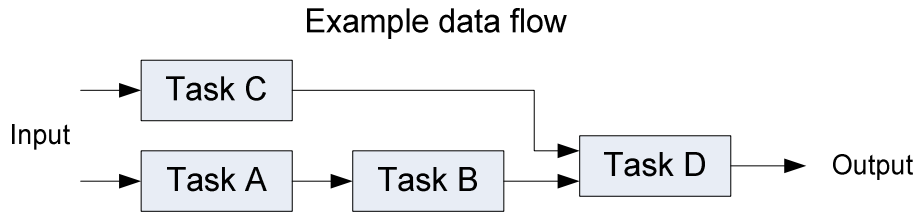
Compiler output

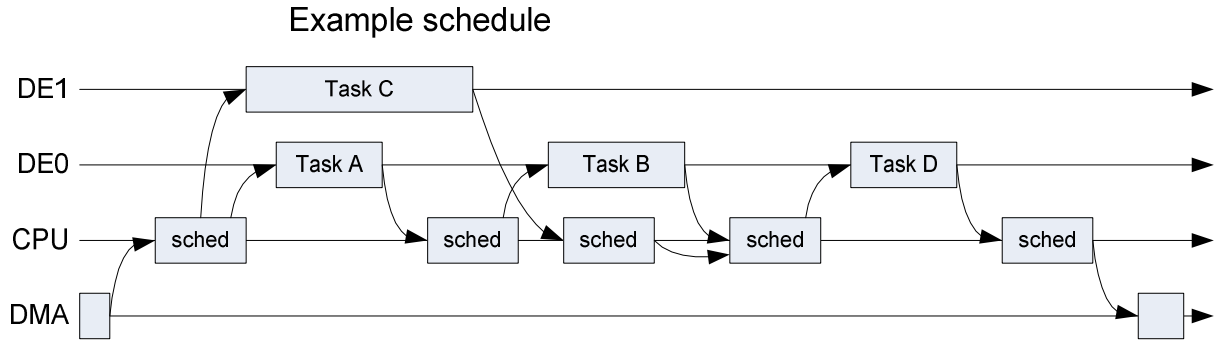
There is no assembler, we have the schedule report instead

```
 pot   | pc    |  operation   | operation description
=======+=======+==============+================================================
       |       | fir_simd     | FUNCTION
       |       |              | BEGIN
 0     | 0     | RT96         |    addr$333:agu_regbank:7:d0
       |       |              |    <-
       |       |              |       | immediate:immediate=<#00000000000000000000000100000000>;
       |       |              |
-------+-------+--------------+------------------------------------------------
 1     | 1     | RT93:0       |    p0reg:p0reg
       |       |              |    <- addr$333:agu_regbank:7:dout6
       |       |              |       | dtcm_ram:WSelect=b64,
       |       |              |         dtcm_ram:CReq=active,
       |       |              |         dtcm_ram:Write=read;
       |       |              |
 1     | 1     | RT105        |    addr$321:agu_regbank:4:d0
       |       |              |    <-
       |       |              |       | immediate:immediate=<#00000000000000000000000101000000>;
       |       |              |
-------+-------+--------------+------------------------------------------------
 2     | 2     | RT104        |    addr$332:agu_regbank:7:d0
       |       |              |    <-
       |       |              |       | immediate:immediate=<#00000000000000000000000000000000>;
       |       |              |
 2     | 2     | RT94:1       |    p1reg:p1reg
       |       |              |    <- p0reg:p0reg
       |       |              |       | ;
```

Program execution

- At the system level, the Ardbeg DE can be viewed as a simple programmable accelerator
  - First, load the program code and data
  - Then give the GO command
  - On completion, an interrupt request is asserted
- However, programmability and the peripheral port allow the DE to interact with the rest of the system
  - It is possible to move the control and scheduling task to the Ardbeg DE
- SoC-C tools automate task scheduling and communication





SoC-C example

```
void main() {
  initFFT();
  PIPELINE{
    for(i=0; i<100; ++i) {
      int buffer[N]@{M0,M1};
      readData(buffer);
      FIR(buffer)@P0;
      FIFO(buffer);
      FFT(buffer)@P1;
      QAM(buffer)@P1;
      SYNCH(buffer) @DMA3;
      FIFO(buffer);
      Viterbi(buffer)@P2;
    }
  }
}
```

Sequential program semantics

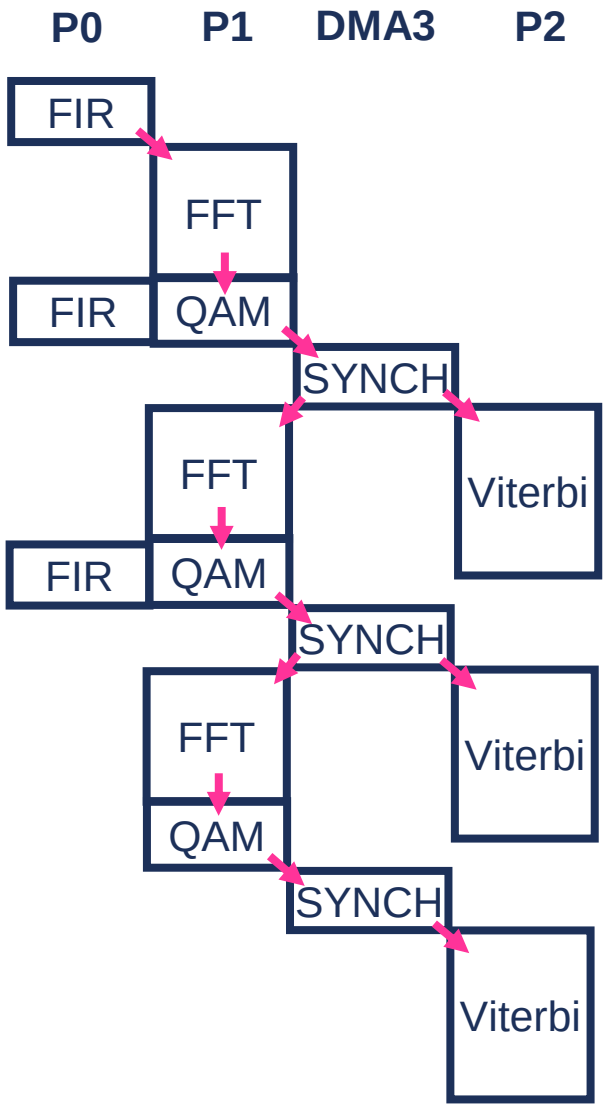
- Well-understood programming model
- Good at expressing complex control patterns
- Straightforward port of legacy codes
- Simple debug model

Annotations

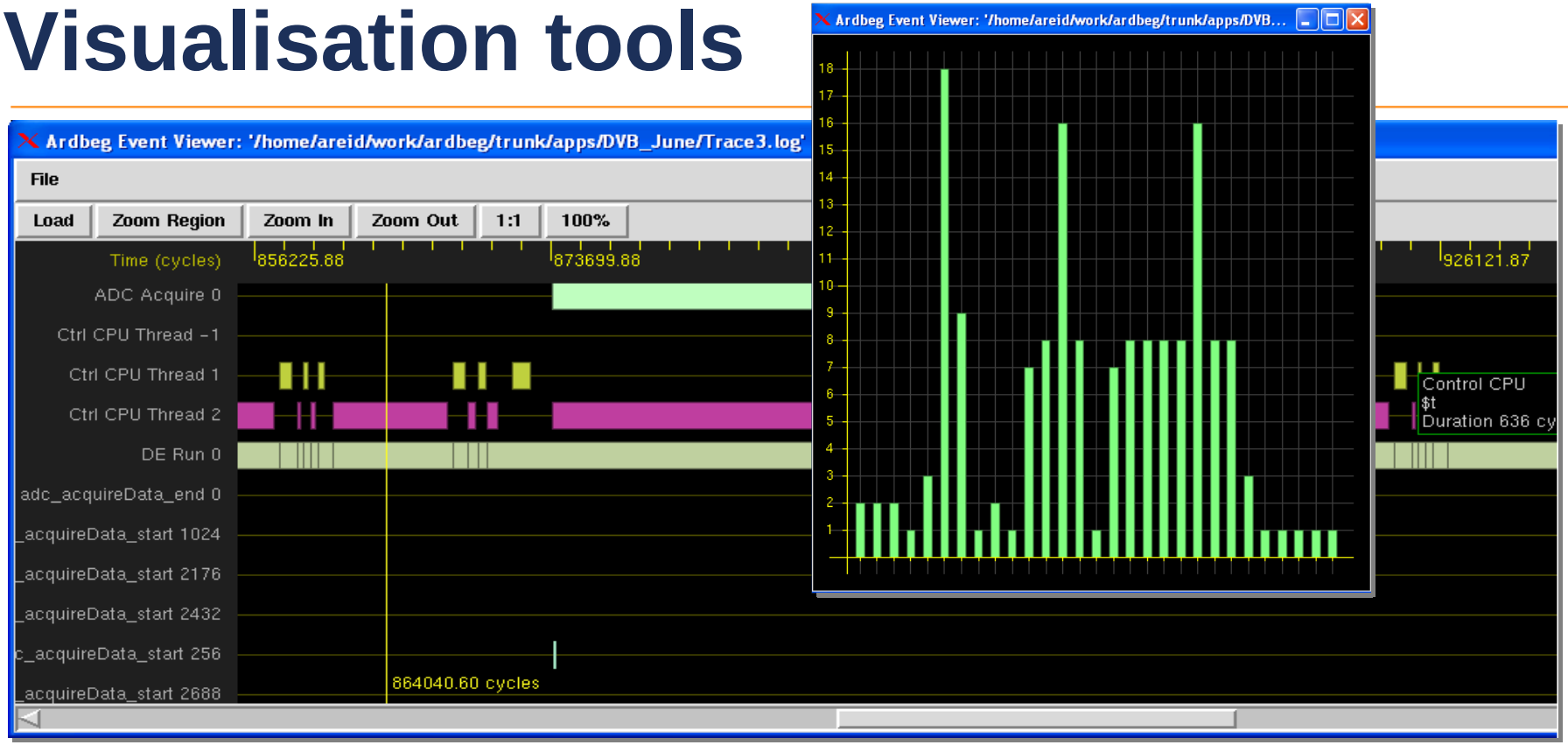
- Expose pipeline parallelism (`PIPELINE,FIFO`)
- Abstracts data copying (`SYNCH`)
- Express code/data placement (`…@P0, …@{M0,M1}`)
- Do not alter semantics
- Can be checked for consistency
- Can be inferred or automatically inserted

SoC-C Event-Driven Execution

```
void main() {
  initFFT();
  PIPELINE{
    for(i=0; i<100; ++i) {
      int buffer[N]@{M0,M1};
      readData(buffer);
      FIR(buffer)@P0;
      FIFO(buffer);
      FFT(buffer)@P1;
      QAM(buffer)@P1;
      SYNCH(buffer) @ DMA3;
      FIFO(buffer);
      Viterbi(buffer)@P2;
    }
  }
}
```



Visualisation tools



- SoC-C run-time generates trace of time-stamped events
- Viewer visualizes trace using various activity views:
  - Thread view: highlights sequence of execution
  - Device view: highlights parallelism for load balancing
  - Event view: correlation of system events
  - Code view: link events to/from source code

More info

- SODA: A low-power architecture for software radio.
  - Proc. 33rd Ann. Int. Symp. on Computer Architecture, Boston, MA USA, June 2006
  - Top Pick—selected as one of the 12 best papers in computer architecture for 2006
- From Soda to Scotch: The Evolution of a Wireless Baseband Processor
  - MICRO-41
- SoC-C: Efficient Programming Abstractions for Heterogeneous Multicore Systems on Chip
  - CASES-2008

Q&A