nFlex:
A Scalable Broadband Communications Processor

Sanjay Vishin
Srinivas Lingam

nBand Communications, Inc.
Overview

• nFlex design goals
• Architecture Overview
• Vector / Scalar processor details
• Dataflow unit (SFA) details
• Programming model
• Conclusion
nFlex Design Goals

• A scalable PHY/MAC architecture
  – To support multi-function/multi-protocol, flexibility and futureproofing of technology needed in broadband communications

• A programmable architecture that pushes solutions closer to ASIC power and size

• A design that keeps the PHY and MAC both under the control of a single instruction stream, for ease of programming

• A design that addresses the entire broadband communications system (PHY+MAC) as a total solution
  – In addition to looking at the dominant kernels from generic communications apps
nFlex Applications

- Broadband fixed wireless systems
  - i.e., BWIF, 802.16
- Wireless local area networks
  - i.e., 802.11[ab], HiperLAN II
- Multi-protocol access points and bridges
- Multiple service convergence applications
  - Broadband WAN + IAD
  - Broadband WAN + WLAN
Programmer’s View of nFlex

Vector State
- Vector register file (16)
- Vector control registers (32)
  - VREV
  - VMASK
  - VCOMM
  - VPSR
  - VLEN
  - VMCR
  - VCBB
  - VCBE
  - RESERVED

Shared Memory

Onchip Memory

Scalar State (1 or more)
- 32 GPRs
- Control Registers
- r0
- r31

SFA State

Control Registers

Programmer’s View of nFlex
Vector/Scalar Processor ISA

• Scalar processor is standard RISC ISA with 32 Regs
  – Added scalar bit manipulation, looping and arithmetic ops (~11 ops)
  – Added the vector as a coprocessor (~42 ops)

• Vector ISA designed using Apps/Kernels e.g., FFT, FIR, FEC, modulation, vector, etc.
  – 7 vector load/store ops (2D, 1D strided loads/stores, etc.)
  – 5 vector multiply ops (cmlpx mult, mult-acc, galois-mult, etc.)
  – 26 vector arithmetic ops (compare, add/sub, viterbi, etc.)
  – 4 intra-vector arithmetic ops (min/max, accumulate, etc.)
Vector Processor Pipeline

- Fetch
- Decode
- Execute
- Memory
- Wback

- (vector load/store pipe)
- (vector load pipe)
- (vector alu pipe) Lazy add!
- (vector alu pipe)
- (vector mul pipe)
Vector Datapath (nFlex, 4 lanes)

LANE #0

Odd Vector regs

Even vector regs

Conflict detect

Instruction Issue

Latency= 4 clock cycles

...
Vector Scaling (4->8 lanes)
Vector Arithmetic Units

- vs1/vs2 and vs1/rs, operands supported for most operations.
- [Un]Signed, Sizes (.w,.h,.b), Saturation, Round Modes (R-, R+, RN)
- SIMD arithmetic within 32b elements
- Intra-vector operations in [vlen/4]+6 cycles -(e.g. vwminx, vwacc[x])

<table>
<thead>
<tr>
<th>LANE3</th>
<th>LANE2</th>
<th>LANE1</th>
<th>LANE0</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>--</td>
<td>3</td>
<td>--</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>7</td>
<td>6</td>
</tr>
<tr>
<td>(c+4)</td>
<td>8</td>
<td>(b+3)</td>
<td>(a+2)</td>
</tr>
<tr>
<td>(10+8)</td>
<td>(c+4)</td>
<td>(f+7)</td>
<td>(e+6)</td>
</tr>
<tr>
<td>(10+8)</td>
<td>(10+e)</td>
<td>(10+e)</td>
<td>(c+a)</td>
</tr>
<tr>
<td>(10+8)</td>
<td>(18+16)</td>
<td>(18+16)</td>
<td>(d+5)</td>
</tr>
</tbody>
</table>

Vector Reduction example
(Add 16 elem in v0=0x1,…,0x10)

(88)
Vector Load/Store Units

- 1D and 2D strided loads and stores with circular buffer addressing mode
- No indexed load/stores. Capability provided by vseed instruction. Limited indirect load on a 4Kbyte memory region (Mostly RO)
  - Divide, sqroot, sin(x), log(x), QAM lookup
- Small strided load/stores run at full speed, larger at _ and _ - zero latency chaining
- Software barriers needed between S/V and V/V memory instructions to satisfy RAW dependencies
Vector Cache

• Designed a low-cost and fast 2 ported (LD, LD/ST) vector cache for unit/short strides (like a stream buffer+cache)
  – 16 cache lines only, fully associative, LRU, 128 byte lines, WBack, 2 ports, 4x 256b fill

• Next-Line prefetching is triggered in 2 ways, through the vprefetch instruction and vld/vst.[p] flag
  – Vcache can keep 4 prefetches in flight
  – Flags for V$ management: prefetch, intent to modify, load/store final data, no snoop, etc.
Cache Coherency in nFlex

(27x4 A bits, 4 Reqs)

Vector Snoop Signals

Global Snoop Bus

Extensions
nFlex Implementation

- SFA
- I-Cache
- D-Cache
- vector Processor
- SPU
Dataflow/SFA System

- Non-vectorizable or continuously used operators in signal processing are scheduled on the SFA System for *Concurrency and Silicon/Power* efficiency.

- Synchronized through interrupts to the scalar processor and through FIFOs within the SFA system.

- Arbitrary *pipelined* SFA streams can be setup by programming the DMA Controller/Global Table (GT).

- Driven by a 16-Input/16-Output channel DMA controller.
SFA System

- SFA sub-blocks in first generation nflex
  - Programmable FIR SFA
  - Programmable Viterbi De/Encoder SFA
  - CRC/PNSeq SFA
  - Bit/Byte Interleaver SFA
  - PVL SFA connects to various streaming interfaces
nFlex I/O System

• PVL SFA block interfaces to various streaming interfaces [A/D, D/A, IF, MAC, etc.]
  – 8 separately clocked channels configured as I or O
  – Programmable data widths of 2,4,…,16b
  – TimeStamping on any I Channels
  – TimeScheduling on any O channel
  – Total of 80 pins to be distributed over the 8 channels

• Other I/O
  – 32b PCI interface
  – GPIO (32) / PWM
  – UART (2)
  – SPI
System Implementors View of nFlex (Application Mapping)
nFlex Software Development Environment

- Binary Code compatibility for future products
- Reduced programming complexity
**MATLAB Code**

function z = complex_FIR (x,y,M)

    % inputs
    %  x = [1,N] vector
    %  y = [1,N] vector
    %  M = # points starting from 0
    % output
    %  z = [1,M-1] vector

    % Initializations
    N = length(x); % Determine # elements of x

    % Main loop
    for i = 1:M
        r = y(i:i+N-1)*x(N:-1:1).'; % Compute sample
        z(i) = r; % Save result
    end

**Vector Oriented C Code**

```c
#include <vector.h>

void complex_FIR(complex *ComplexInput, complex *ComplexCoef, word16 Order)
{
    _vector TempVector,Result;
    _vector_pair TempVectorPair1,TempVectorPair2;
    word16 i;
    word32 Temp;

    /* Initialization */
    SetVectorLength(Order);
    SetProductShift(LEFT_SHIFT_1);
    TempVectorPair2 = _vmov_ext_from_scalar(0);

    for(i=0; i < Order; i++)
    {
        /* Read data from the memory starts here */
        TempVector = *(vector *)(ComplexInput+Order-i-1);
        Temp = *(word32 *)(ComplexCoef+i);
        /* Read data from the memory ends here */
        TempVectorPair1 = _vcmul_vs(TempVector,(unsigned int)Temp);
        TempVectorPair2 = _vaccx_w(TempVectorPair2,TempVectorPair1);
    }

    Result = _vmix_hh(_vget_high(TempVectorPair2),_vget_low(TempVectorPair2));
    return(Result);
}
```

Complex FIR Filter Example

Vector Oriented C Code
Conclusion

• Example of Application Performance/implementation speed
  – 802.11a is completely coded and uses ~70 % of an nFlex (@250 MHz)
  – 802.11a PHY took 4 months and 7000 lines of code to complete

• Kernel Performance
  – FFT 64pt, radix 4, 16b data (including bit reversal) 170 cycles
  – FFT 1024pt, radix 4, 16b data (including bit reversal) 4500 cycles
  – Complex FIR (N=256, Coeff=32, 16b elements) 2600 cycles