ARM ISA SIMD Extensions

John.Rayfield@arm.com

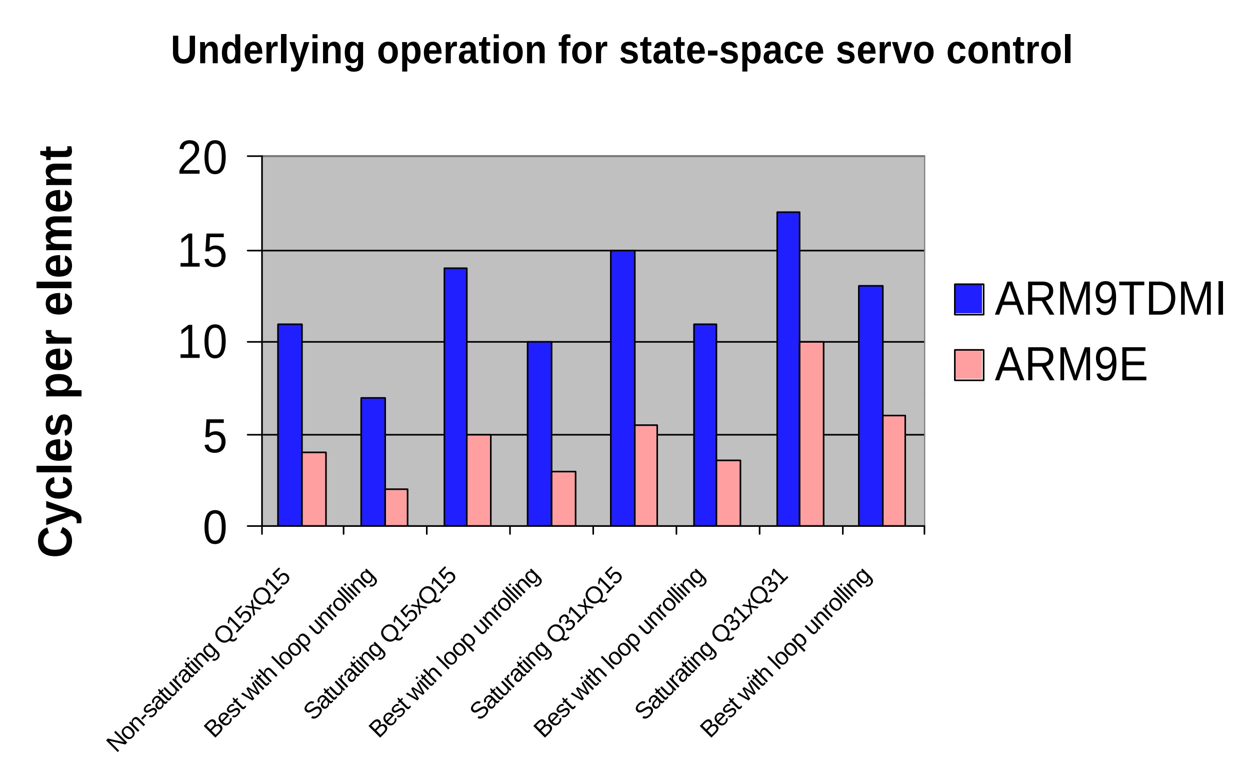
www.arm.com

Markets

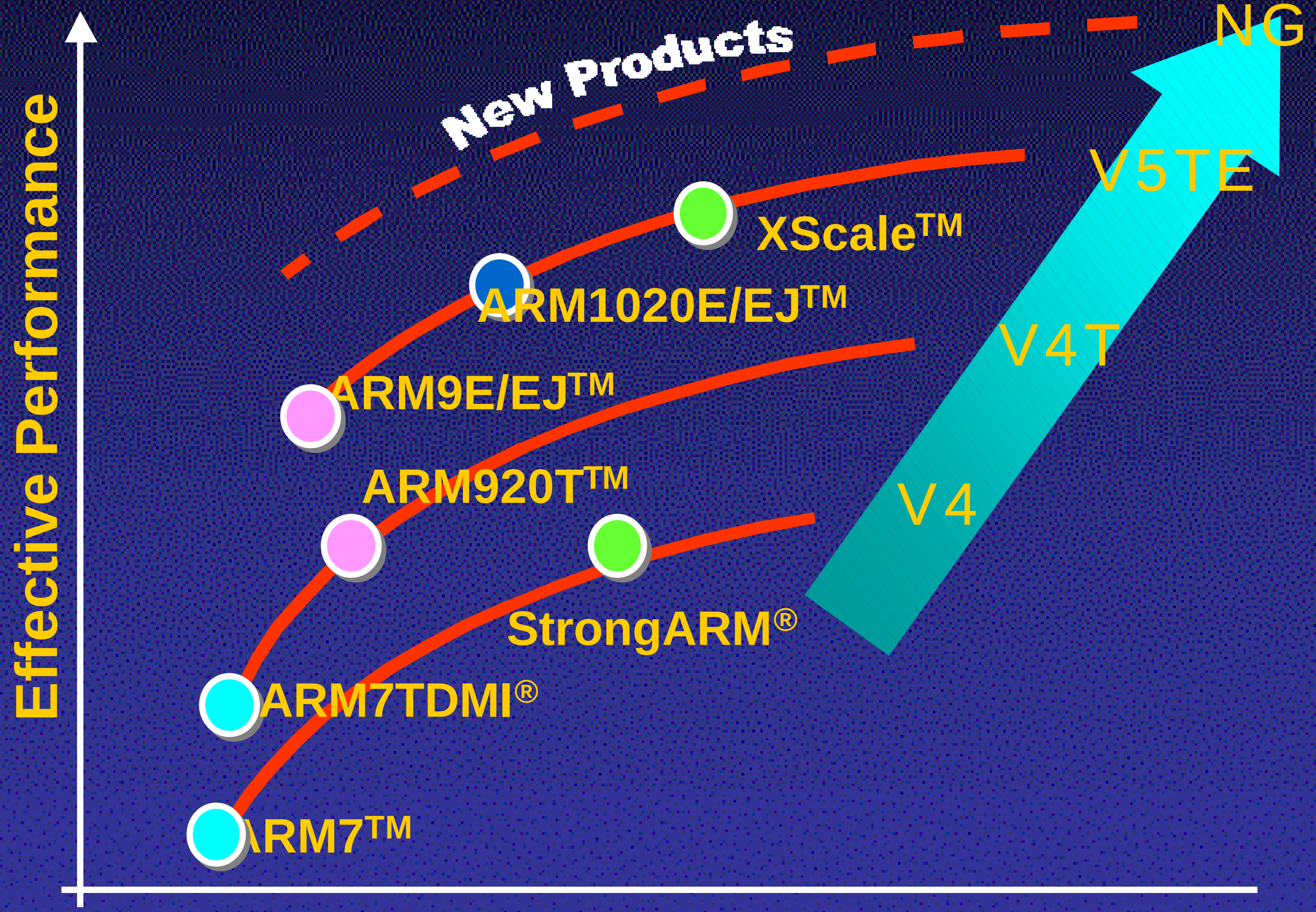
- ARM implementations are increasingly being used in a broad range of embedded and platform computing applications:
  - Wireless, mass-storage, automotive, industrial, digital imaging, secure, consumer entertainment
- Many of these are increasing the need for signal processing capabilities in the base architecture

The ARM ISA v5TE extensions

- ARM added considerable DSP functionality into the architecture with the introduction of ARM ISA V5TE
- More than 2x performance for some algorithms



ARM Architecture roadmap



Xscale is a trademark of Intel Corporation

Examples of new market needs:

- Video capabilities in 3G cellular
    - MPEG4 QCIF decode and encode at low power
- Audio encoders/decoders in PDA
    - MP3 and others
- Automotive collision avoidance systems
    - Radar systems with extensive FFT needs

- All pushing DSP/Media performance
    - and suffer from the 90/10 rule

Why SIMD?

- With short data-types, most of a 32-bit data-path is left computing on sign-extension bits
- SIMD is the most power and area efficient architectural model if you have data regularity
  - 8-bit and 16-bit data can be more efficiently manipulated in the 32-bit pipeline
- Very carefully architected so as not to compromise ARM implementations

Summary of the new instructions

- SIMD arithmetic
  - 16-bit addition and subtraction ADD16,SUB16,ADDSUB,SUBADD
  - 8-bit addition and subtraction ADD8, SUB8
  - Selection SEL8, SEL16

- Multiply instructions
  - Dual 16-bit multiply SMLAD, SMLSD, SMLALD, SMLSLD
  - 32-bit media multiplies SMMLA, SMMLAR SMMLS, SMMLSR

- Flexible saturation **SSAT, USAT**

Detail of the new instructions

New SIMD status flags

31….27	24	19….16		7	6	5	4….0
Flags		GE[3:0]		I	F	T	Mode

- Four new flags GE[3:0] added into the CPSR
  - GE[3] = ‘greater than or equal’ for bit 31 (8-bit,16-bit SIMD)
  - GE[2] = ‘greater than or equal’ for bit 23 (8-bit SIMD)
  - GE[1] = ‘greater than or equal’ for bit 15 (8-bit,16-bit SIMD)
  - GE[0] = ‘greater than or equal’ for bit 7 (8-bit SIMD)

- Flags affected by all SIMD operations

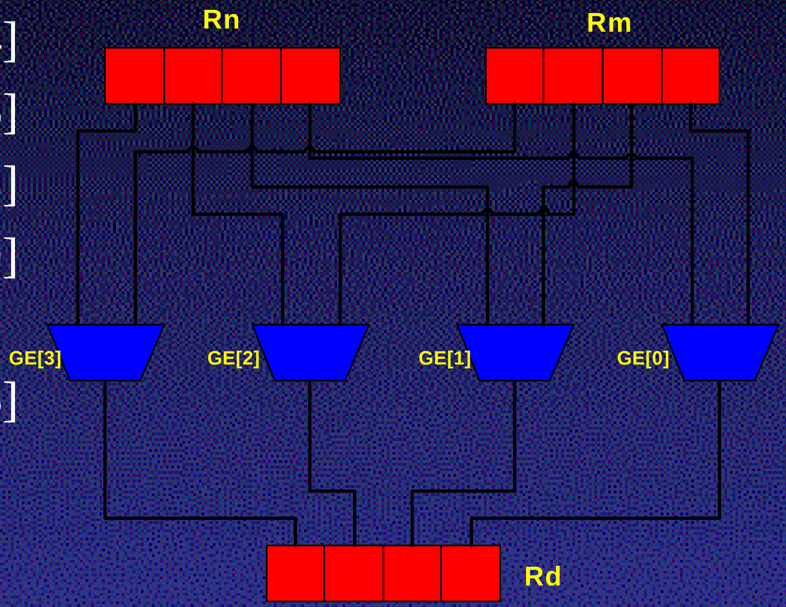
- Flags used by SIMD select operations

SIMD Add and Subtract

- ADD16 Rd,Rn,Rm, SUB16 Rd,Rn,Rm
- ADDSUB Rd,Rn,Rm, SUBADD Rd,Rn,Rm
- ADD8 Rd,Rn,Rm, SUB8 Rd,Rn,Rm
- Other variants
  - saturating
  - versions that swaps two halves of one source (16-bit)
  - version that halve the results to prevent overflow
  - unsigned versions
- Applications
  - General use in all algorithms using 16-bit precision
  - 16-bit FFT - audio, speech recognition
  - DCT video and still-image coding

SIMD Selection Operations

- SEL8 Rd,Rn,Rm
  - Rd[31:24] = GE[3] ? Rn[31:24] : Rm[31:24]
  - Rd[23:16] = GE[2] ? Rn[23:16] : Rm[23:16]
  - Rd[15:08] = GE[1] ? Rn[15:08] : Rm[15:08]
  - Rd[07:00] = GE[0] ? Rn[07:00] : Rm[07:00]
- SEL16 Rd,Rn,Rm
  - Rd[31:16] = GE[3] ? Rn[31:16] : Rm[31:16]
  - Rd[15:00] = GE[1] ? Rn[15:0] : Rm[15:0]
- Applications
  - 8-bit and 16-bit SIMD
    - minimum, maximum, absolute, Viterbi (ACS)



Dual 16x16 multiply

- Result = [Accumulator +] 16-bits x 16-bits +/- 16-bits x 16-bits

SMUA{X}D Rd, Rm, Rs	Rd = Rmlo*Rslo/hi + Rmhi*Rshi/lo
SMUS{X}D Rd, Rm, Rs	Rd = Rmlo*Rslo/hi - Rmhi*Rshi/lo
SMLA{X}D Rd, Rm, Rs, Rn	Rd = Rn + Rmlo*Rslo/hi + Rmhi*Rshi/lo
SMLS{X}D Rd, Rm, Rs, Rn	Rd = Rn + Rmlo*Rslo/hi - Rmhi*Rshi/lo
SMLALD Rlo, Rhi, Rm, Rs	Rhi.Rlo += Rmlo*Rslo + Rmhi*Rshi

- Option to swap one 16-bit input operand postion
- Option on 32-bit or 64-bit extended accumulation
- Applications
  - 16-bit FFT
  - 16-bit real/complex filters/dot-products with 32-bit accumulator
  - 16-bit filters/dot-products with 64-bit accumulator

32x32 fractional multiplies

- Result = [Accumulator +/-] ((32-bits x 32-bits + round)>>32)

```
– SMMLA{R} Rd, Rm, Rs, Rn      ;Rd = Rn + ((Rm*Rs + round)>>32
– SMMLS{R} Rd, Rm, Rs, Rn      ;Rd = Rn - ((Rm*Rs + round)>>32)
– SMMUL{R} Rd, Rm, Rs          ;Rd = ((Rm*Rs) + round)>>32
```

- Applications
  - 32-bit FFT, complex filters
  - 32-bit real/complex filters with 32-bit accumulator
  - 32-bit filters with 64-bit accumulator

Saturation operation

- SSAT Rd,#BitPosition,Rm,{Shift}
- USAT Rd,#BitPosition,Rm,{Shift}

- Flexible saturation in signed and unsigned form at arbitrary bit-position in the source word (Rm)

- Applications
  - Graphics, pixel plotting, audio algorithms ISO/ETSI standards

Example uses of the new instructions

16-bit Complex multiply

- ;ARM ISA v5TE implementation

```
SMULTT    Real,Ra,Rb     ;Real = Ra.real*Rb.real
SMULBB    Temp,Ra,Rb     ;Temp = Ra.imag*Rb.imag
SUB Real,Real,Temp       ;Real = Ra.real*Rb.real - Ra.imag*Rb.imag
SMULTB    Imag,Ra,Rb     ;Imag = Ra.real*Rb.imag
SMLABT    Imag,Ra,Rb     ;Imag = Ra.real*Rb.imag + Ra.imag*Rb.real
;Total of 5-cycle in a single-cycle implementation
```

- ;New implementation

```
SMUSD Real,Ra,Rb         ;Real = Ra.real*Rb.real - Ra.imag*Rb.imag
SMUADX Imag,Ra,Rb        ;Imag = Ra.real*Rb.imag + Ra.imag*Rb.real
```

- ;Total of 2-cycles in a single cycle implementation

Add-compare-select

- Underlying operation in the Viterbi algorithm

  Equalization, convolutional code decoding, recognition engines

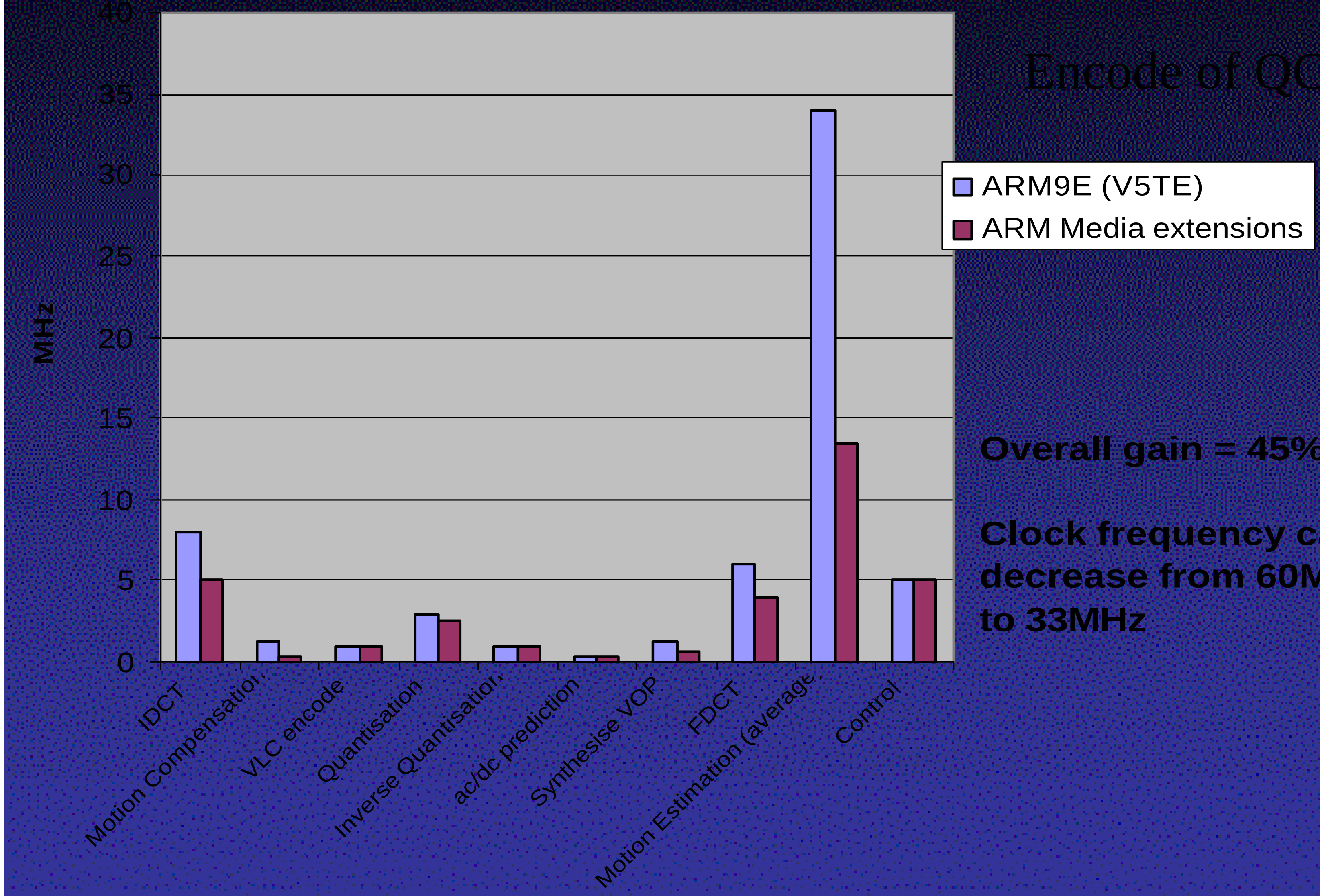
- ;New implementation

```
ADD8     Rp1, Rs1, Rb1     ; path 1 metric = state 1 metric + metric for branch 1
ADD8     Rp2, Rs2, Rb2     ; path 2 metric = state 2 metric + metric for branch 2
USUB8    Rt, Rp1, Rp2      ; compare metrics - setting the SIMD flags
SEL8     Rd, Rp2, Rp1      ; choose best (smallest) metric
```

- Total of 4-cycles for a single-cycle implementation

  Total of 12-cycles for previous ARM ISA v5TE implementation

MPEG4 performance on ARM



Encode of QCIF at 15fps

Overall gain = 45%

Clock frequency can decrease from 60MHz to 33MHz

Dominated by motion estimation

- Underling operation is a sum of absolute differences

```
;Implementation using the SIMD operations
LDR         Rx, [pRx], #Offset                        ;get x(i,j)
LDR         Ry, [pRy], #Offset                        ;get y(i+m, j+n)
USUB8       Rt1, Rx, Ry                  ;Rt1 = x(i,j) - y(i+m,j+n)
USUB8       Rt2, Ry, Rx                  ;Rt2 = y(i+m,j+n) - x(i,j)
SEL8        Rd, Rt2, Rt1                              ;ABS(x(i,j) - y(i+m,j+n))
```

- Taking 5-cycles in a single-cycle implementation
  - Previously on v5TE cores, this takes around 15-cycles

Conclusions

- Up to 4x performance at virtually zero power increment
- Single core solutions for high performance platform processing requirements such as wireless and portable applications
- Fully backward compatible with the ARM roadmap
  - Value of existing 3rd party software support is maintained
  - No OS changes required for applications to use the new extensions