XILINX
ALL PROGRAMMABLE™

NEON/VFPU Talk

Jim Wu, Senior Staff SAE

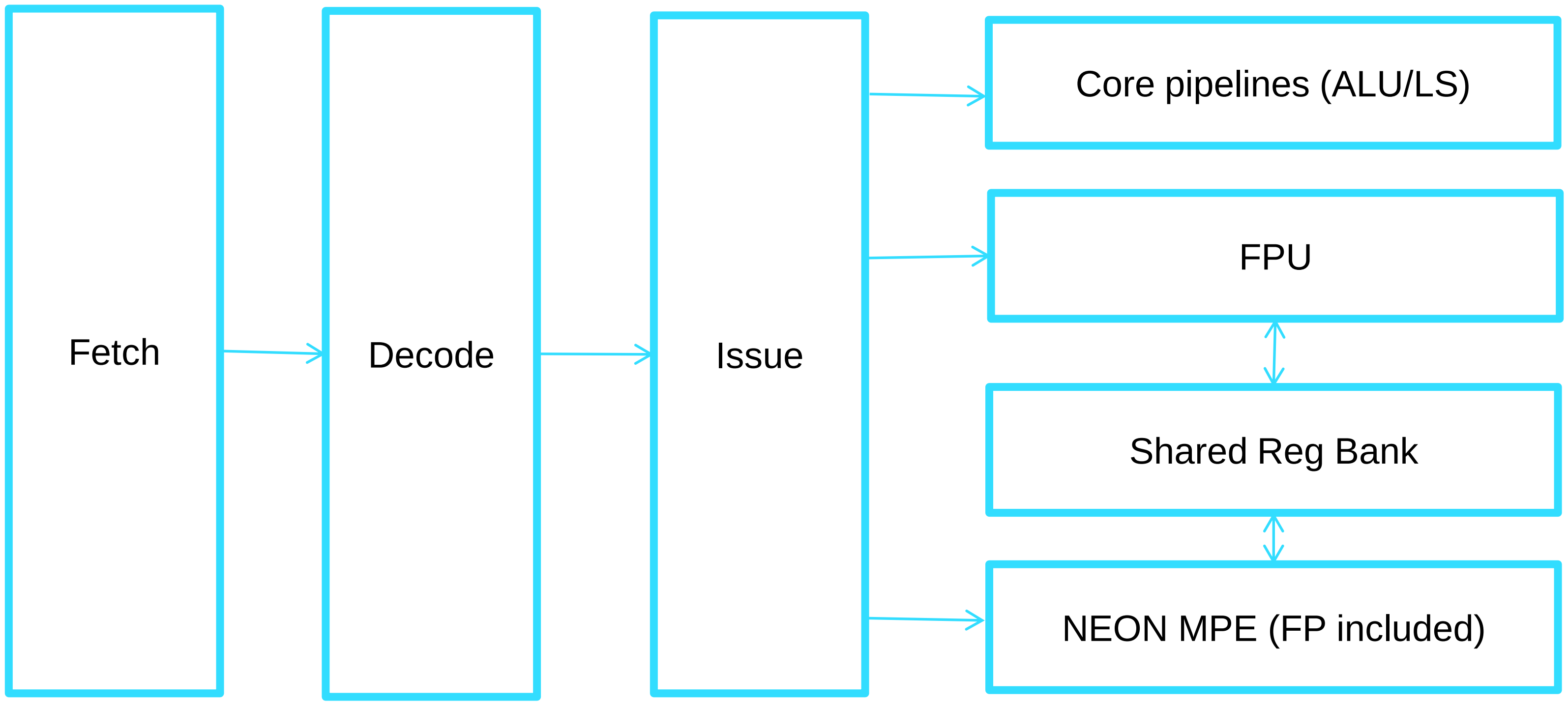
November, 2012

Outlines

- Overview of NEON/FPU
- NEON/FPU Usage in Xilinx Tool Chain
- NEON Debug
- Hardware Acceleration Using HLS
- Libraries Optimized for NEON

VFPU and NEON MPE Overview

- Zynq includes both VFPU and NEON MPE
- Instructions for FPU and NEON issued in parallel with core pipelines
- FPU and NEON share common register bank



FPU Features

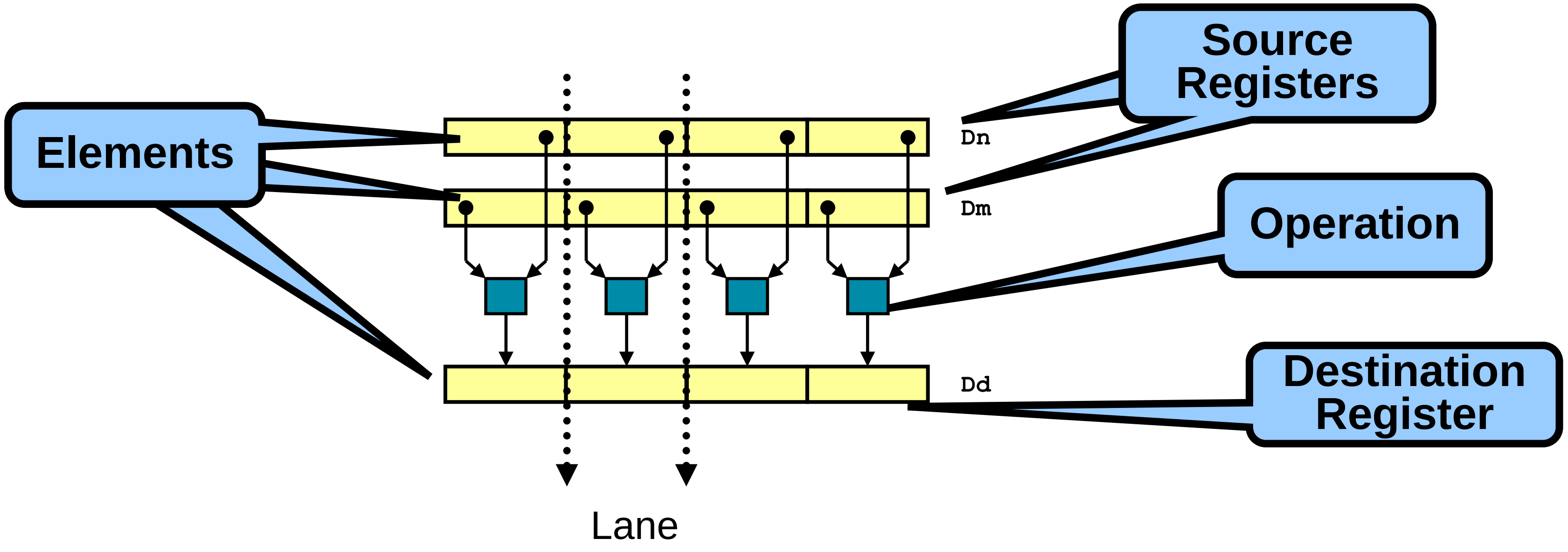
- Single and **double precision** VFPv3 FPU
- **VFP “Vector” feature not supported by A9 FPU hardware**
  - 2-sp and 4-sp vectorization available when using NEON
- Register set shared with NEON
- New half-precision conversions (FP16) useful for graphics & audio
- Mode supported: Flush-to-Zero, Default NaN, **full compliance with IEEE-754 standard**
- Rounding modes supported: All
- Trapped exceptions disabled (but can interrupt CPU)
- Denormals handled in hardware

FPU instruction throughput and latency cycles (partial)

UAL	Single Precision		Double Precision	
	Throughput	Latency	Throughput	Latency
VADD VSUB	1	4	1	4
VMUL	1	5	2	6
VMLA	1	8	2	9
VDIV	10	15	20	25
VSQRT	13	17	28	32

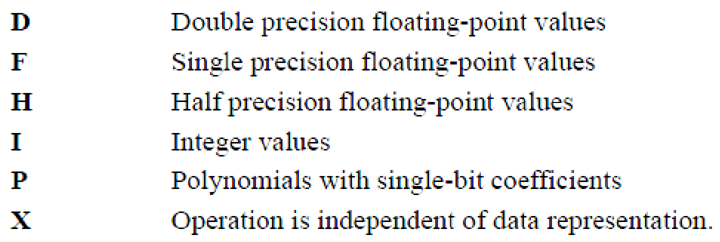
NEON MPE

- Packed SIMD processing
  - Registers considered as vectors of elements of the **same** data type
  - Instructions performing the **same** operation in all lanes



NEON Supported Data Types

- Double precision floating point NOT supported
- Data type convention: .(format letter)(bit width). e.g.
  - .F32 = 32 bit single precision floating point



D	Double precision floating-point values
F	Single precision floating-point values
H	Half precision floating-point values
I	Integer values
P	Polynomials with single-bit coefficients
X	Operation is independent of data representation.

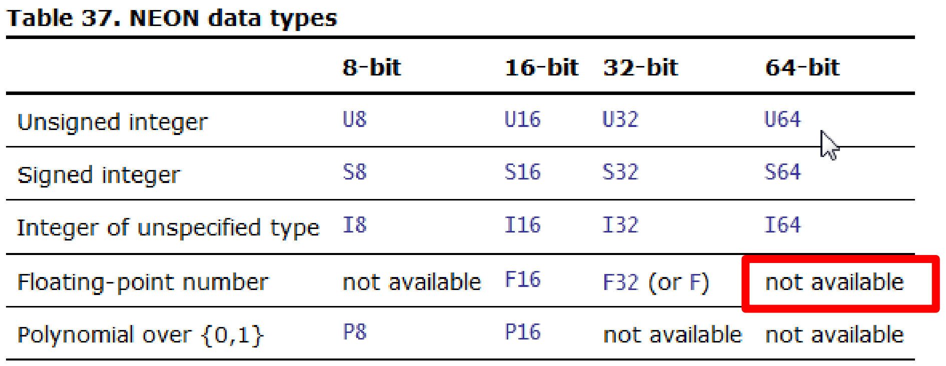


Table 37. NEON data types

	8-bit	16-bit	32-bit	64-bit
Unsigned integer	U8	U16	U32	U64
Signed integer	S8	S16	S32	S64
Integer of unspecified type	I8	I16	I32	I64
Floating-point number	not available	F16	F32 (or F)	not available
Polynomial over {0,1}	P8	P16	not available	not available

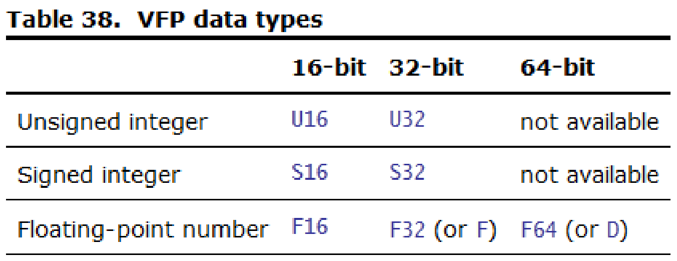


Table 38. VFP data types

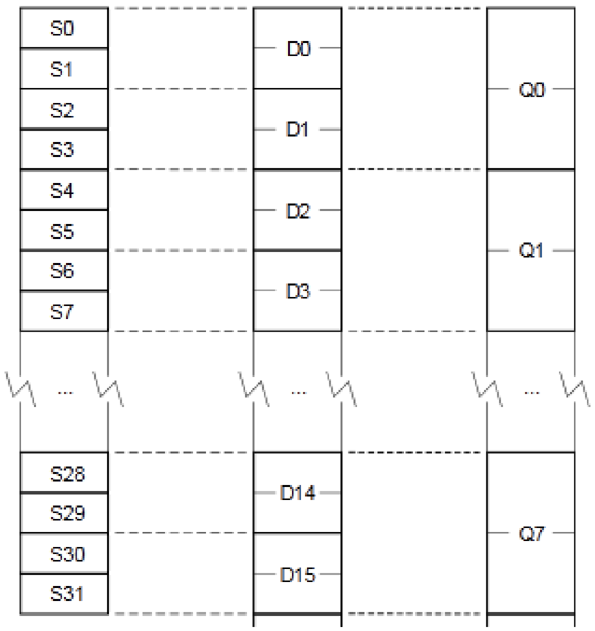
	16-bit	32-bit	64-bit
Unsigned integer	U16	U32	not available
Signed integer	S16	S32	not available
Floating-point number	F16	F32 (or F)	F64 (or D)

XILINX ALL PROGRAMMABLE.

NEON/VFPU Registers

- NEON and VFPU use the same extension register bank
- NEON View: 32x64b registers(D0-D31) dual viewed as 16x128b registers (Q0-Q31)
- VFPU View: 32x64b registers(D0-D31) dual viewed as 32x32b registers(S0-S31)
- Registers hold one or more elements of the same data type
- Scalar elements are referenced using the array notation Vn[x]

Figure 2. Extension register bank



- The mapping between registers
  - S<2n> maps to the least significant half of D<n>
  - S<2n+1> maps to the most significant half of D<n>
  - D<2n> maps to the least significant half of Q<n>
  - D<2n+1> maps to the most significant half of Q<n>

NEON Instruction Syntax

V`{<mod>}<op>{<shape>}{<cond>}{.<dt>}(<dest>}, src1, src2`

`<mod>` - **Instruction Modifiers**

Q indicates the operation uses saturating arithmetic (e.g. VQADD)
H indicates the operation halves the result (e.g. VHADD)
D indicates the operation doubles the result (e.g. VQDMUL)
R indicates the operation performs rounding (e.g. VRHADD)

`<op>` **- Instruction Operation (e.g. ADD,MUL, MLA, MAX, SHR, SHL, MOV)**

`<shape>` **- Shape**

L – The result is double the width of both operands
W – The result and first operand are double the width of the last operand
N – The result is half the width of both operands

`<cond>` **- Conditional, used with IT instruction**

`<.dt>` **- Data type**

`<dest>` **- Destination,** `<src1>` **- Source operand 1,** `<src2>`

NEON FP vs VFPv3

- NEON FP supports single precision floating point numbers only
- NEON handling of denormalized numbers and NaNs is not IEEE754 compliant. It operates Flush-to_Zero mode, which is compliant with the standards of most modern programming languages, including C and C++.

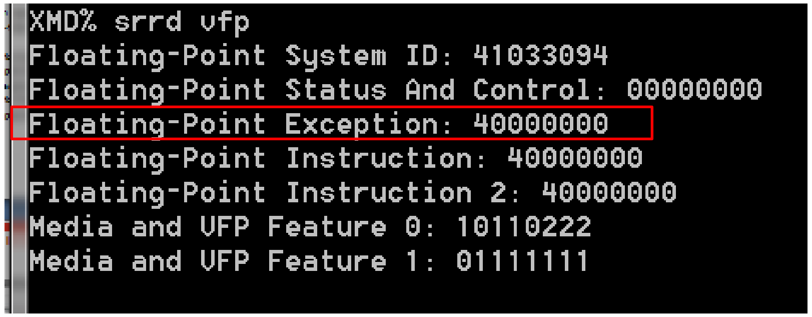
- VFPv3 supports single precision and double precision floating point numbers.
- VFPv3 is fully compliant to IEEE754 in hardware.

Outlines

- Overview of NEON/FPU
- **NEON/FPU Usage in Xilinx Tool Chain**
- NEON Debug
- Hardware Acceleration Using HLS
- Libraries Optimized for NEON

NEON/VFPU State on ZC702 after Power on/Boot

- Bare-Metal: NEON and VFPU are enabled after power on
  - FPEXC[30] = 1



```
XMD% srrd vfp
Floating-Point System ID: 41033094
Floating-Point Status And Control: 00000000
Floating-Point Exception: 40000000
Floating-Point Instruction: 40000000
Floating-Point Instruction 2: 40000000
Media and VFP Feature 0: 10110222
Media and VFP Feature 1: 01111111
```

- 14.2 Linux image from Xilinx wiki: VFPU and NEON are enabled on Linux configuration
  - **CONFIG_NEON** and **CONFIG_VFP**, **CONFIG_VFPv3** are enabled when building the Linux image.

- **Users DO NOT need to enable NEON and VFPU to execute NEON instructions on ZC702**

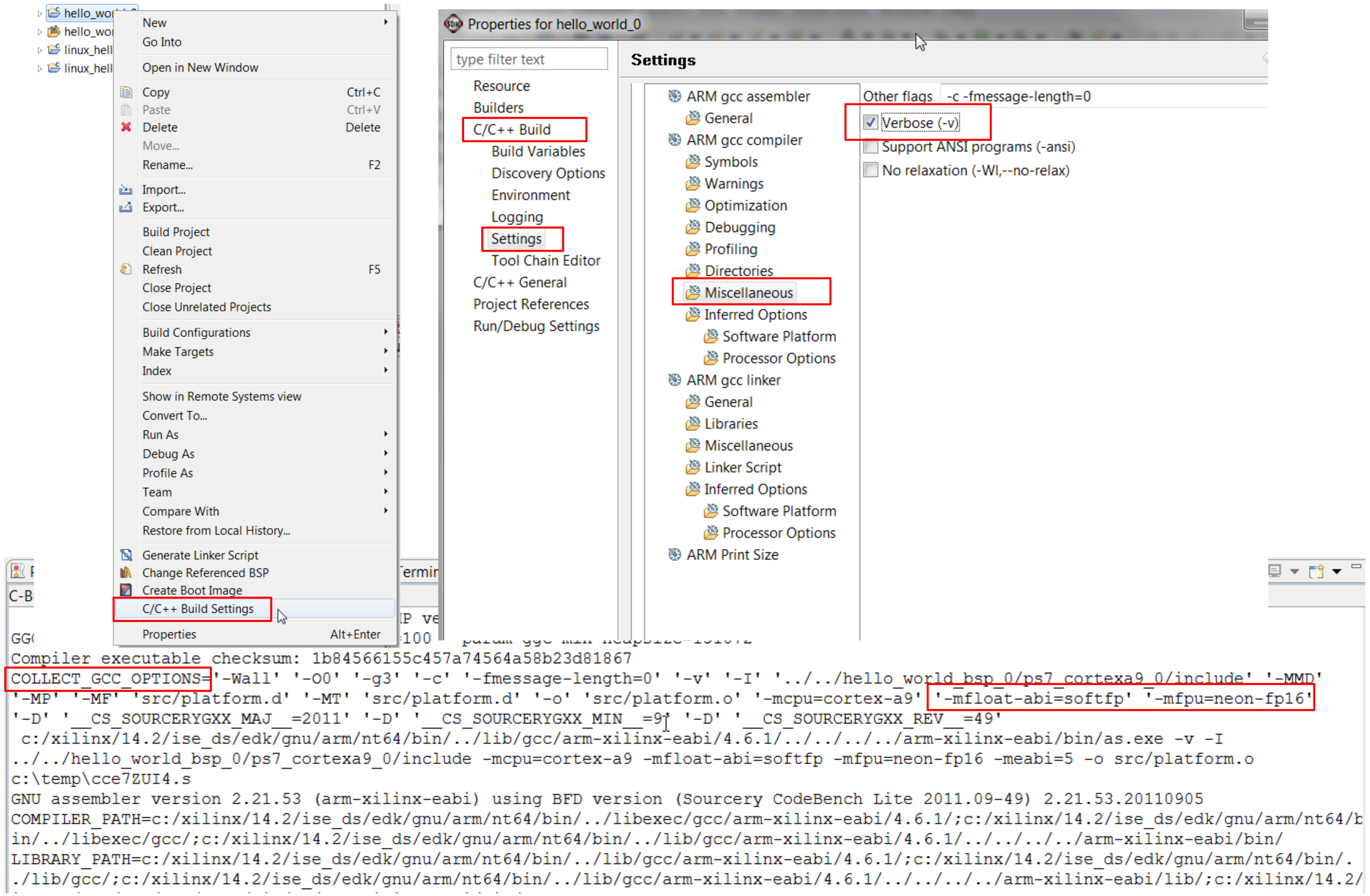
GNU (gcc/g++) Compiler Options for NEON/VFPU

- **-mfpu=**
  - **neon**: select NEON as FPU
  - **vfpv3**: select VFPU as FPU
- **-mfloat-abi=**
  - **soft**: No HW FP support
  - **softfp**: soft linkage. Compiler can generate HW FP instructions supported by FPU
  - **hard**: hard linkage. Compiler can generate HW FP instructions supported by FPU. Note: all code must be compiled with this option, including libraries
- **-ftree-vectorize** : enable NEON vectorization (automatically turned on by –O3)
- **-mvectorize-with-neon-quad**: use Q registers for NEON instructions
- **-ffast-math**: Some floating-point operations are not vectorized by default due to possible loss of precision. Use this option to enable vectorization of floating point operations.
- **-ftree-vectorizer-verbose=n**
  - n: verbose level. Higher values add more information the vectorizations the compiler is performing or unable to perform

Default NEON/VFPU Compiler Options in Xilinx ARM GNU Toolchain (Sourcery CodeBench Lite)

- Run SDK and open a workspace
- Select a project, right click to select **C/C++ Build Settings** (see snapshots on next slide)
- Select **C/C++ Build->Settings->ARM gcc compiler->Miscellaneous**
- Check **Verbose(-v)**
- Click OK to close the properties window
- Build the project
- Find “COLLECT_GCC_OPTIONS” in the Console, which all compiler options used during compilation

Default NEON/VFPU Compiler Options in Xilinx ARM GNU Toolchain (cont'd)



Other flags: -c -fmessage-length=0

- [✓] Verbose (-v)
- [ ] Support ANSI programs (-ansi)
- [ ] No relaxation (-Wl,--no-relax)

```
Compiler executable checksum: 1b84566155c457a74564a58b23d81867
COLLECT_GCC_OPTIONS='-Wall' '-O0' '-g3' '-c' '-fmessage-length=0' '-v' '-I' '../../hello_world_bsp_0/ps7_cortexa9_0/include' '-MMD'
'-MP' '-MF' 'src/platform.d' '-MT' 'src/platform.d' '-o' 'src/platform.o' '-mcpu=cortex-a9' '-mfloat-abi=softfp' '-mfpu=neon-fp16'
'-D' '__CS_SOURCERYGXX_MAJ__=2011' '-D' '__CS_SOURCERYGXX_MIN__=9' '-D' '__CS_SOURCERYGXX_REV__=49'
 c:/xilinx/14.2/ise_ds/edk/gnu/arm/nt64/bin/../lib/gcc/arm-xilinx-eabi/4.6.1/../../../../arm-xilinx-eabi/bin/as.exe -v -I
../../hello_world_bsp_0/ps7_cortexa9_0/include -mcpu=cortex-a9 -mfloat-abi=softfp -mfpu=neon-fp16 -meabi=5 -o src/platform.o
c:\temp\cce7ZUI4.s
GNU assembler version 2.21.53 (arm-xilinx-eabi) using BFD version (Sourcery CodeBench Lite 2011.09-49) 2.21.53.20110905
COMPILER_PATH=c:/xilinx/14.2/ise_ds/edk/gnu/arm/nt64/bin/../libexec/gcc/arm-xilinx-eabi/4.6.1/;c:/xilinx/14.2/ise_ds/edk/gnu/arm/nt64/b
in/../libexec/gcc/;c:/xilinx/14.2/ise_ds/edk/gnu/arm/nt64/bin/../lib/gcc/arm-xilinx-eabi/4.6.1/../../../../arm-xilinx-eabi/bin/
LIBRARY_PATH=c:/xilinx/14.2/ise_ds/edk/gnu/arm/nt64/bin/../lib/gcc/arm-xilinx-eabi/4.6.1/;c:/xilinx/14.2/ise_ds/edk/gnu/arm/nt64/bin/.
./lib/gcc/;c:/xilinx/14.2/ise_ds/edk/gnu/arm/nt64/bin/../lib/gcc/arm-xilinx-eabi/4.6.1/../../../../arm-xilinx-eabi/lib/;c:/xilinx/14.2/
```

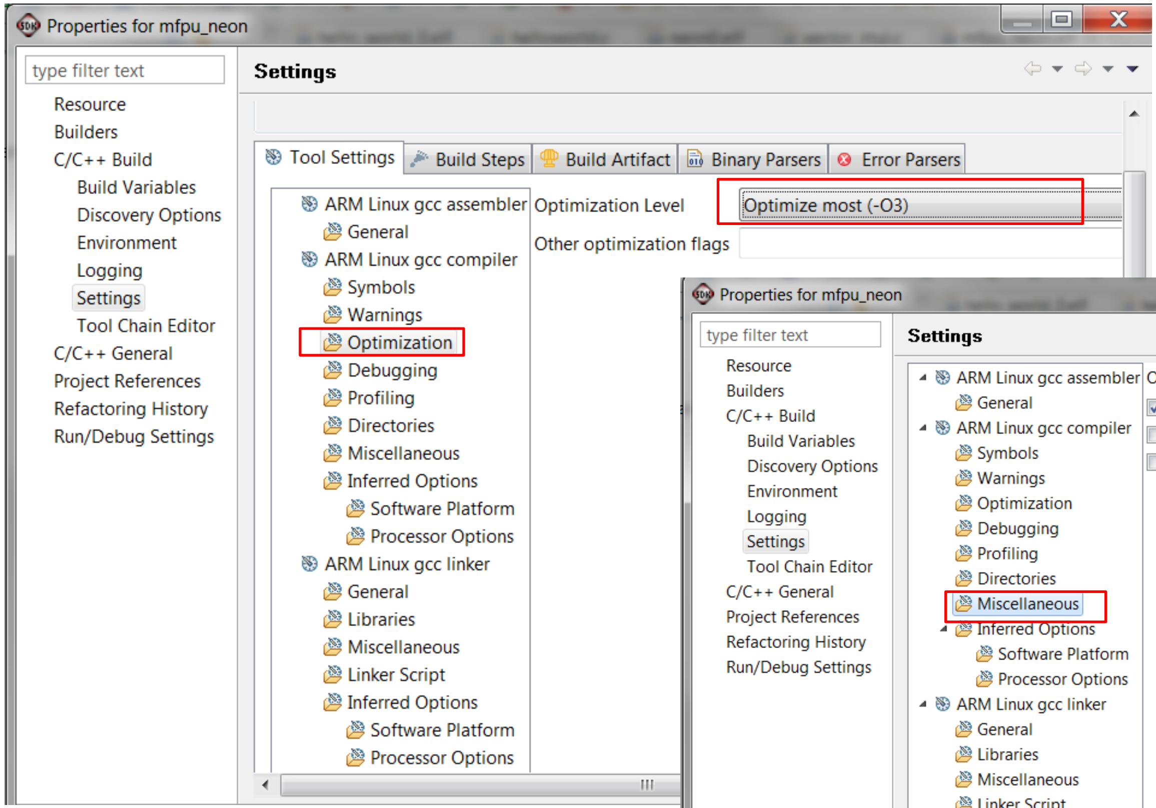
Default NEON/VFPU Compiler Options in Xilinx ARM GNU Toolchain (cont'd)

	-O	-mfpu	-mfloat-abi	-ftree-vectorize
arm-xilinx-eabi-gcc	-O0	neon-fp16	softfp	off
	-O2	neon-fp16	softfp	off
	-O3	neon-fp16	softfp	on
arm-xilinx-eabi-g++	-O0	neon-fp16	softfp	off
	-O2	neon-fp16	softfp	off
	-O3	neon-fp16	softfp	on
arm-xilinx-linux-gnueabi-gcc	-O0	neon-fp16	softfp	off
	-O2	neon-fp16	softfp	off
	-O3	neon-fp16	softfp	on
arm-xilinx-linux-gnueabi-g++	-O0	neon-fp16	softfp	off
	-O2	neon-fp16	softfp	off
	-O3	neon-fp16	softfp	on

Automatic Vectorization in Xilinx ARM GNU Toolchain

- Xilinx ARM GNU Toolchain use the options below by default
  - -mfloat-abi=softfp
  - -mfpu=neon-fp16
- -O0 turns off automatic vectorization regardless additional compiler options.
- -O1 or -O2: Add options below:
  - -ftree-vectorize
  - -ffast-math (optional. required for floating point vectorization)
  - -mvectorize-with-neon-quad
  - -ftree-vectorizer-verbose=n (optional. for debug purpose)
- -O3: automatically turns on -ftree-vectorize. Add options below:
  - -ffast-math (optional. required for floating point vectorization)
  - -mvectorize-with-neon-quad
  - -ftree-vectorizer-verbose=n (optional. for debug purpose)

Automatic Vectorization in Xilinx ARM GNU Toolchain (cont'd)



Properties for mfpu_neon

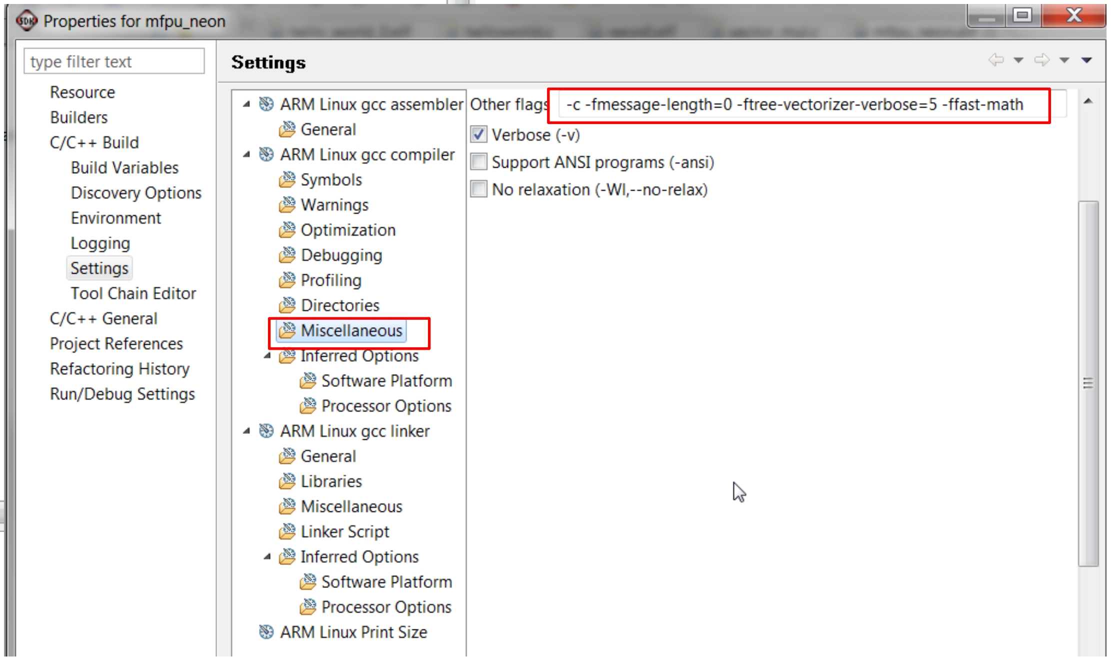
Settings

Tool Settings | Build Steps | Build Artifact | Binary Parsers | Error Parsers

ARM Linux gcc compiler > Optimization

Optimization Level: Optimize most (-O3)

Other optimization flags:



Properties for mfpu_neon

Settings

ARM Linux gcc compiler > Miscellaneous

Other flags: -c -fmessage-length=0 -ftree-vectorizer-verbose=5 -ffast-math

[✓] Verbose (-v)

[ ] Support ANSI programs (-ansi)

[ ] No relaxation (-Wl,--no-relax)

Optimization for Automatic Vectorization

- Indicate knowledge of loop count: e.g. mask lower 2 bits of loop count to indicate a loop of multiple of 4
- Remove inner-loop dependencies: result of one iteration not dependent on previous iteration
- Avoid conditions inside loop: avoid if-else
- Use the **restrict** keyword: no overlap on memory space for variables
- Use the smallest data type possible. e.g.
  - 2x instructions on 8-bit data than 16-bit data
  - No vectorization for double precision data. Use single precision if possible
- Use the same data types for operations
- Use **-ffast-math** compiler option for automatic vectorization for floating point data

Automatic Vectorization Example

```
void vector_mul_f32a(float * __restrict a, float * __restrict b, float * __restrict p)
{
    int i;

    for (i=0; i<VECTOR_LENGTH; i++) {
       p[i] = a[i] * b[i];
    }
}
```

```
 100574:  f469378f  vld1.32   {d19}, [r9]
 100580:  f42b178f  vld1.32   {d1}, [fp]
 1005ac:  f3433d91  vmul.f32  d19, d19, d1
 1005e4:  f445378f  vst1.32   {d19}, [r5]
```

NEON C Intrinsics

- C functions providing access to low level NEON operations
- NEON vectors defined as variables and passed as arguments or return values
- **#include <arm_neon.h>**
- List of Intrinsics
- Note: compilers may still apply different optimizations

NEON C Intrinsics Example

```
void vector_mul_f32i(float *a, float *b, float *p)
{
    int i;

    float32x4_t a4, b4, p4;
    float32_t   *pa4 = a;
    float32_t   *pb4 = b;
    float32_t   *pp4 = p;

    for (i=0; i<VECTOR_LENGTH/4; i++) {
        a4 = vld1q_f32(pa4);
        b4 = vld1q_f32(pb4);
        p4 = vmulq_f32(a4, b4);
        vst1q_f32(pp4, p4);
        pa4+=4;
        pb4+=4;
        pp4+=4;
    }
}
```

```
  1004bc: edd30b08  vldr      d16, [r3, #32]
  1004c0: edd31b0a  vldr      d17, [r3, #40]      ; 0x28
  1004c4: edd32b04  vldr      d18, [r3, #16]
  1004c8: edd33b06  vldr      d19, [r3, #24]
  1004cc: f3400df2  vmul.f32  q8, q8, q9
  1004d0: edc30b0c  vstr      d16, [r3, #48]      ; 0x30
  1004d4: edc31b0e  vstr      d17, [r3, #56]      ; 0x38
```

No Support for Double Precision Floating Point in NEON

- Double precision floating point NOT supported

```
//double precision floating point
void vector_mul_f64a(double * __restrict a, double * __restrict b, double *
__restrict p)
{
    int i;

    for (i=0; i<VECTOR_LENGTH; i++) {
       p[i] = a[i] * b[i];
    }
}
```

```
../src/vector_mul.c:72: note: not vectorized: no vectype for stmt: D.17521_90
= *D.17520_89;
 scalar_type: double
```

Execution Time Comparison

- Execute vector_mul functions 200 times

	Automatic Vectorization w/o Q-reg SP	Automatic Vectorization w/ Q-reg SP	Intrinsics SP	DP
-O0	9.06ms	9.06ms	10.67ms	9.07ms
-O3 with all applicable options	1.74ms	1.15ms	1.28ms	2.12ms

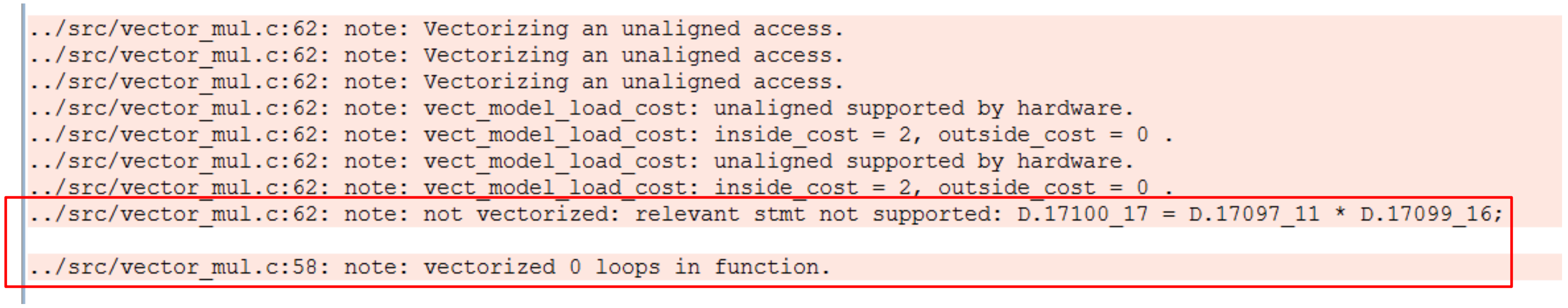
Outlines

- Overview of NEON/FPU
- NEON/FPU Usage in Xilinx Tool Chain
- **NEON Debug**
- Hardware Acceleration Using HLS
- Libraries Optimized for NEON

NEON Debug: Tree Vectorizer

- -ftree-vectorizer-verbose=n
  - n: verbose level. Higher value generates more verbose messages

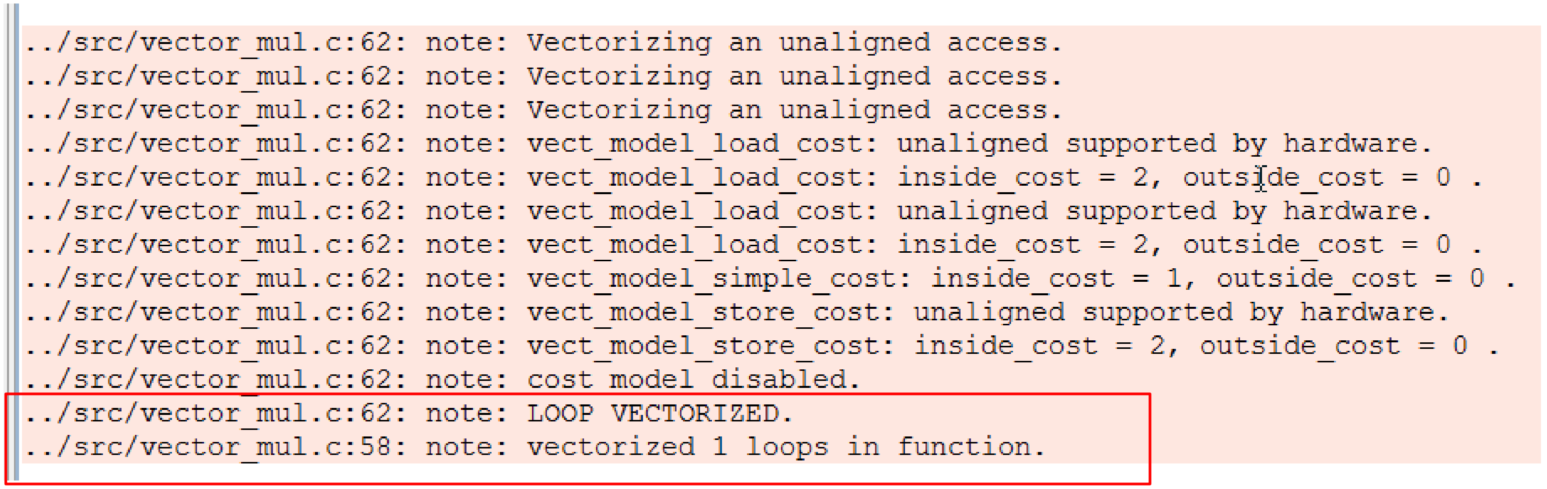
Without -ffast-math



```
../src/vector_mul.c:62: note: Vectorizing an unaligned access.
../src/vector_mul.c:62: note: Vectorizing an unaligned access.
../src/vector_mul.c:62: note: Vectorizing an unaligned access.
../src/vector_mul.c:62: note: vect_model_load_cost: unaligned supported by hardware.
../src/vector_mul.c:62: note: vect_model_load_cost: inside_cost = 2, outside_cost = 0 .
../src/vector_mul.c:62: note: vect_model_load_cost: unaligned supported by hardware.
../src/vector_mul.c:62: note: vect_model_load_cost: inside_cost = 2, outside_cost = 0 .
../src/vector_mul.c:62: note: not vectorized: relevant stmt not supported: D.17100_17 = D.17097_11 * D.17099_16;

../src/vector_mul.c:58: note: vectorized 0 loops in function.
```

With -ffast-math



```
../src/vector_mul.c:62: note: Vectorizing an unaligned access.
../src/vector_mul.c:62: note: Vectorizing an unaligned access.
../src/vector_mul.c:62: note: Vectorizing an unaligned access.
../src/vector_mul.c:62: note: vect_model_load_cost: unaligned supported by hardware.
../src/vector_mul.c:62: note: vect_model_load_cost: inside_cost = 2, outside_cost = 0 .
../src/vector_mul.c:62: note: vect_model_load_cost: unaligned supported by hardware.
../src/vector_mul.c:62: note: vect_model_load_cost: inside_cost = 2, outside_cost = 0 .
../src/vector_mul.c:62: note: vect_model_simple_cost: inside_cost = 1, outside_cost = 0 .
../src/vector_mul.c:62: note: vect_model_store_cost: unaligned supported by hardware.
../src/vector_mul.c:62: note: vect_model_store_cost: inside_cost = 2, outside_cost = 0 .
../src/vector_mul.c:62: note: cost model disabled.
../src/vector_mul.c:62: note: LOOP VECTORIZED.
../src/vector_mul.c:58: note: vectorized 1 loops in function.
```

XILINX ALL PROGRAMMABLE.

NEON Debug: disassemble code

- arm-xilinx-linux-gnueabi-objdump -S bm_neon_benchmark.elf > dump.txt
  - -S annotates source code in the disassembled code.
  - SDK runs thiw command when opening elf file
  - Not work well with –O3
- arm-xilinx-linux-gnueabi-objdump -d bm_neon_benchmark.elf > dump.txt
  - -d only generates disassembly.

NEON Debug: display NEON/VFPU registers on Linux

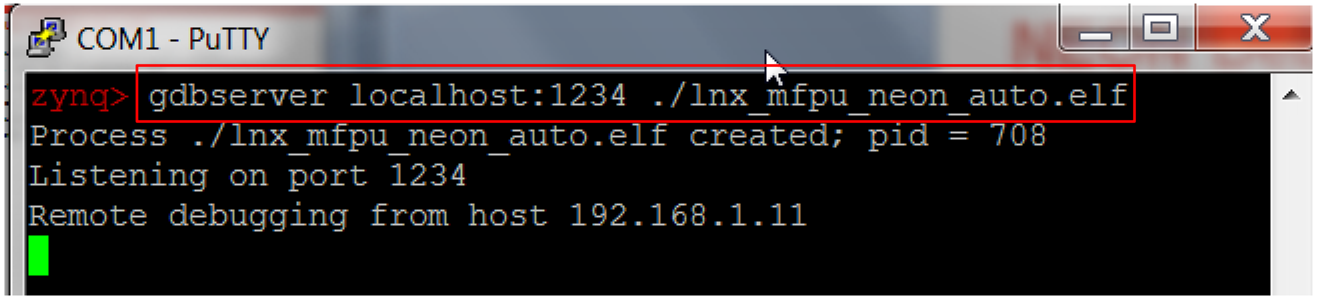
- On a terminal running on ZC702 run gdbserver (see the snapshot on the next slide) “zynq>” is the command prompt

```
zynq> gdbserver localhost:1234 ./lnx_mfpu_neon_auto.elf
```

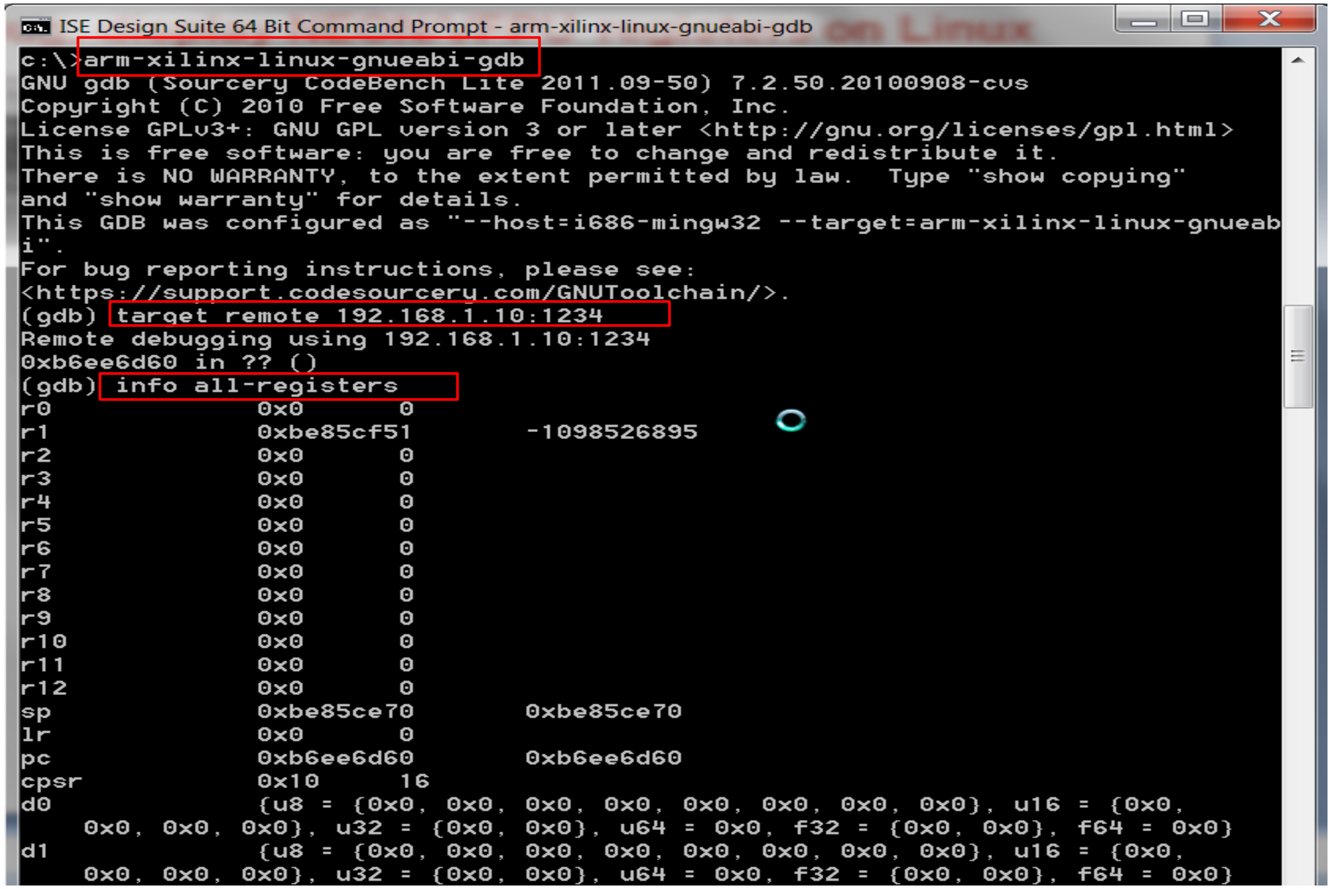
- On PC run gdb (C:\> and (gdb) are the command prompt)

```
c:\>arm-xilinx-linux-gnueabi-gdb
(gdb) target remote 192.168.1.10:1234
Remote debugging using 192.168.1.10:1234
0xb6ee6d60 in ?? ()
(gdb) info all-registers
d0             {u8 = {0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0}, u16 = {0x0,
    0x0, 0x0, 0x0}, u32 = {0x0, 0x0}, u64 = 0x0, f32 = {0x0, 0x0}, f64 = 0x0}
d1             {u8 = {0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0}, u16 = {0x0,
    0x0, 0x0, 0x0}, u32 = {0x0, 0x0}, u64 = 0x0, f32 = {0x0, 0x0}, f64 = 0x0}
```

NEON Debug: display NEON/VFPU registers on Linux



```
zynq> gdbserver localhost:1234 ./lnx_mfpu_neon_auto.elf
Process ./lnx_mfpu_neon_auto.elf created; pid = 708
Listening on port 1234
Remote debugging from host 192.168.1.11
```



ISE Design Suite 64 Bit Command Prompt - arm-xilinx-linux-gnueabi-gdb

```
c:\>arm-xilinx-linux-gnueabi-gdb
GNU gdb (Sourcery CodeBench Lite 2011.09-50) 7.2.50.20100908-cvs
Copyright (C) 2010 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "--host=i686-mingw32 --target=arm-xilinx-linux-gnueab
i".
For bug reporting instructions, please see:
<https://support.codesourcery.com/GNUToolchain/>.
(gdb) target remote 192.168.1.10:1234
Remote debugging using 192.168.1.10:1234
0xb6ee6d60 in ?? ()
(gdb) info all-registers
r0             0x0      0
r1             0xbe85cf51       -1098526895
r2             0x0      0
r3             0x0      0
r4             0x0      0
r5             0x0      0
r6             0x0      0
r7             0x0      0
r8             0x0      0
r9             0x0      0
r10            0x0      0
r11            0x0      0
r12            0x0      0
sp             0xbe85ce70       0xbe85ce70
lr             0x0      0
pc             0xb6ee6d60       0xb6ee6d60
cpsr           0x10     16
d0             {u8 = {0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0}, u16 = {0x0,
    0x0, 0x0, 0x0}, u32 = {0x0, 0x0}, u64 = 0x0, f32 = {0x0, 0x0}, f64 = 0x0}
d1             {u8 = {0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0}, u16 = {0x0,
    0x0, 0x0, 0x0}, u32 = {0x0, 0x0}, u64 = 0x0, f32 = {0x0, 0x0}, f64 = 0x0}
```

XILINX ALL PROGRAMMABLE.

NEON Debug in ARM Development Suite

- RealView Development Suite(RVDS)
  - RVDS Can display Neon instructions in the RVD disassembly view
  - RVDS Profiler support profiling NEON/VFP code
- Note: Development Suite 5(DS-5) replaces RVDS
  - DS-5 supports ZC702

Outlines

- Overview of NEON/FPU
- NEON/FPU Usage in Xilinx Tool Chain
- NEON Debug
- **Hardware Acceleration Using HLS**
- Libraries Optimized for NEON

Hardware Acceleration Using HLS

- **HLS C/C++ code can be executed in ARM with little changes**
  - Add "C:\Xilinx\Vivado_HLS\2012.2\include“ to include directories in SDK
  - Easy to evaluate performance/resource difference between SW and HW

- **8x8 QRD Run Time**

	QRD SP in ARM @667MHz	QRD DP in ARM @667MHz	QRD SP in Fabric @250MHz	QRD DP in Fabric @250MHz
Run Time	34us	49us	7us	

Outlines

- Overview of NEON/FPU
- NEON/FPU Usage in Xilinx Tool Chain
- NEON Debug
- Hardware Acceleration Using HLS
- **Libraries Optimized for NEON**

Libraries Optimized for NEON

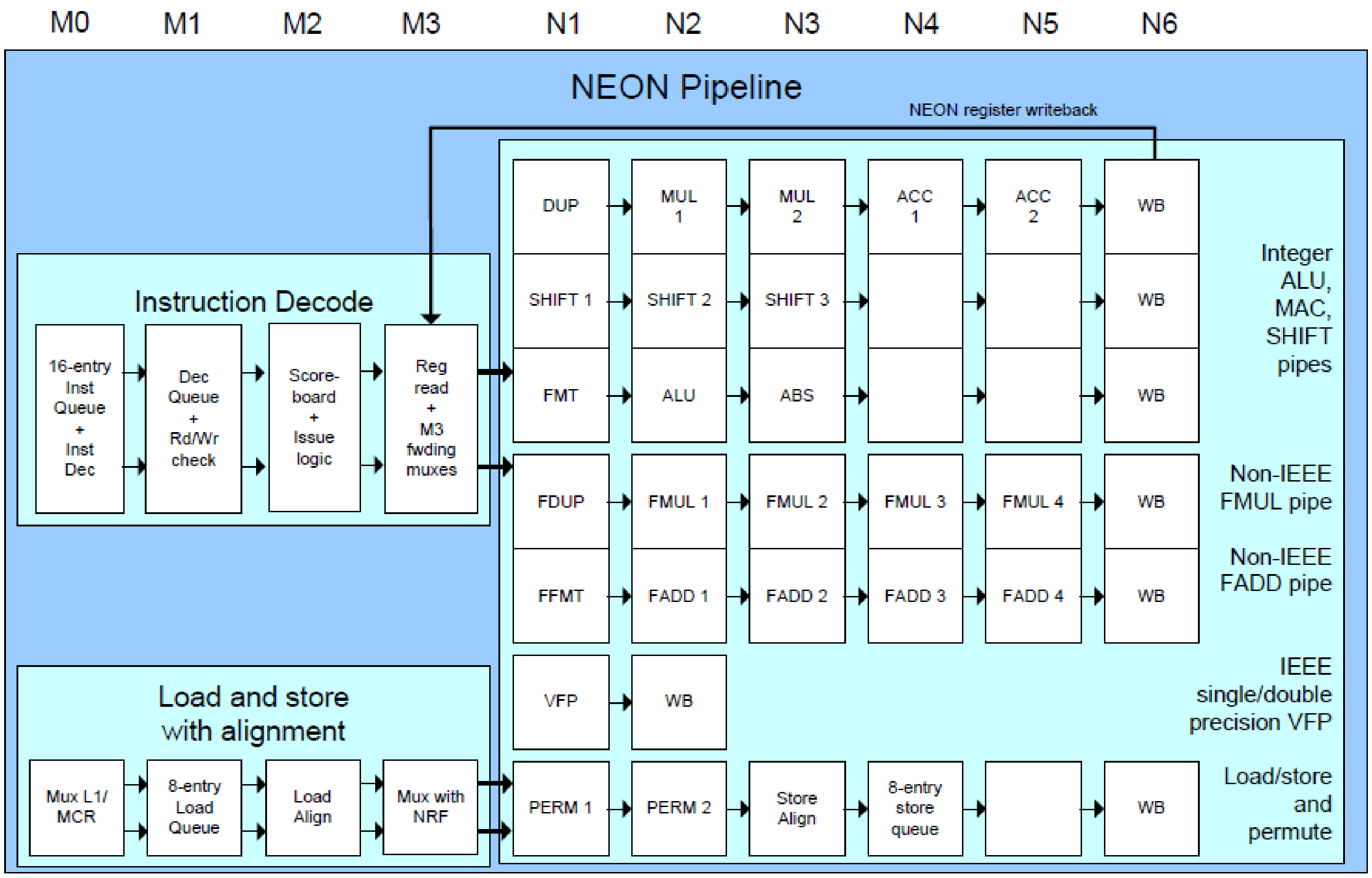
- Project Ne10: open source library. A small set of floating-point, vector arithmetic, and matrix manipulation functions
- OpenMAX DL: royalty-free and cross-platform library of low-level multimedia kernels or media processing building blocks to accelerate media codecs. ARM has created a reference implementation of the OpenMAX DL API, as well as hand-optimized ports for the NEON general-purpose SIMD engine found in ARM Cortex-A series.
  - Video Domain
  - Still Image Domain
  - Image Processing Domain
  - Audio Domain
  - Signal Processing Domain

Backup Slides

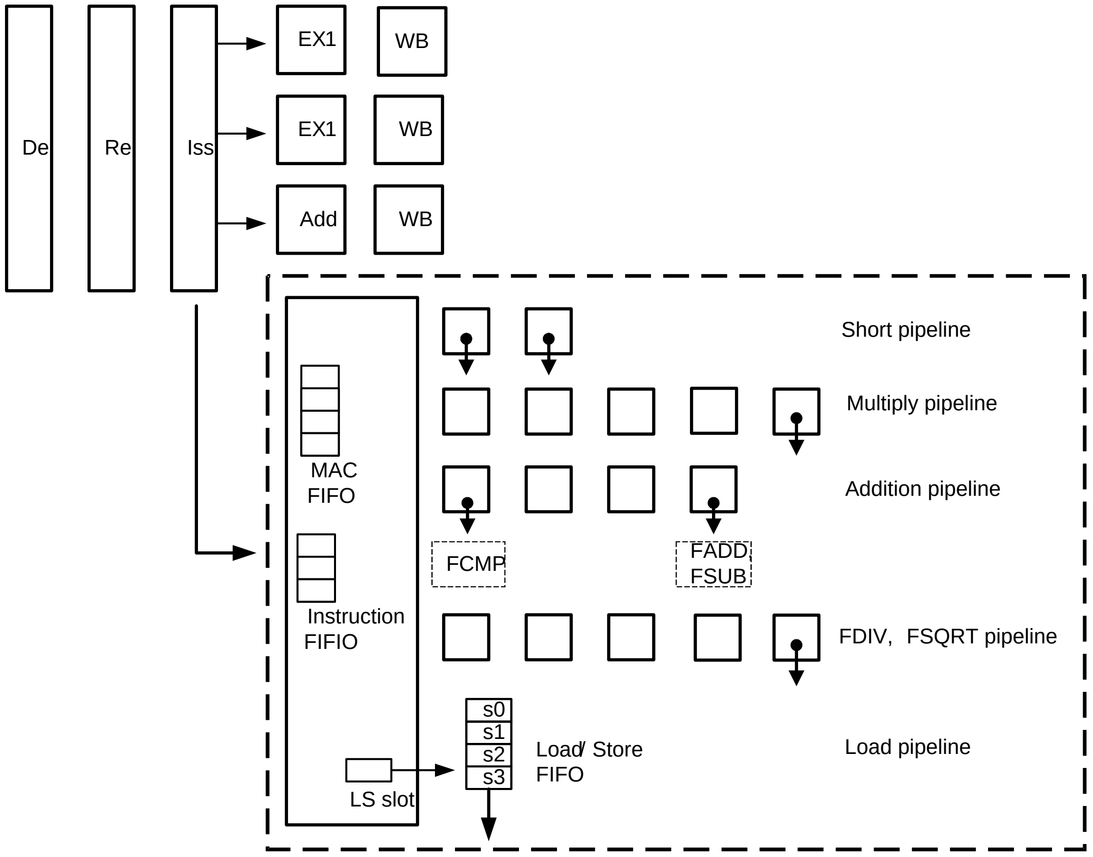
ARM References

- ARM Info Center for all documents
- Cortex™-A9 Technical Reference Manual r4p1
- Cortex-A9 NEON Media Processing Engine Technical Reference Manual r4p1
- Cortex™-A9 Floating-Point Unit Technical Reference Manual r4p1

NEON Pipeline



FPU Pipeline



NEON in opensource

- **Google – WebM – 11,000 lines NEON assembler!**
- **Bluez – official Linux Bluetooth protocol stack**
  - NEON sbc audio encoder
- **Pixman (part of cairo 2D graphics library)**
  - Compositing/alpha blending
- **ffmpeg – libavcodec**
  - LGPL media player used in many Linux distros
  - NEON Video: MPEG-2, MPEG-4 ASP, H.264 (AVC), VC-1, VP3, Theora
  - NEON Audio: AAC, Vorbis, WMA
- **x264 – Google Summer Of Code 2009**
  - GPL H.264 encoder – e.g. for video conferencing
- **Android – NEON optimizations**
  - **Skia** library, S32A_D565_Opaque **5x** faster using NEON
  - Available in Google Skia tree from 03-Aug-2009
- **Eigen2 – C++ vector math / linear algebra template library**
- **Theorarm – libtheora NEON version (optimized by Google)**

NEON in opensource (cont’d)

- Theorarm – libtheora NEON version (optimized by Google)
- libjpeg – optimized JPEG decode (IJG library)
- FFTW – NEON enabled FFT library
- LLVM – code generation backend used by Android Renderscript