ABOUT GCC LINARO 4.6.2 MULTILIB TOOLCHAIN

1 WHAT'S NEW

New features of this toolchain include:

- GCC 4.6.2 released from Linaro:
  - Add cortex-a9 support by -mcpu=cortex-a9.
  - Support VFPv3 and NEON.
- Multilib cross toolchain
  - The library compiled separately for different CPU model.
- Optimized library.
  - Support hard float and cortex-a9.
  - Some common library routines are optimized to improve application performance.
- Application debug tools:
  - Provide some debug tools to trace and detect application bugs.

2 WHAT'S INSIDE

The whole toolchain contains:

· binutils 2.21

· gcc linaro 4.6.2 with multilib support

· glibc 2.13

· glibc-ports 2.13 (some routines are optimized with neon and arm instructions)

· gdb and gdbserver linaro 7.2

· other debug tools and some companion libraries

Toolchain directory structure is as follow:

|-- bin // toolchain with prefix, such as arm-none-linux-gnueabi-gcc etc.

|-- lib // library files used for toolchain itself, not for application

|-- arm-fsl-linux-gnueabi

|-- bin // toolchain without prefix, such as gcc.

|-- debug-root // all debug tools

|-- multi-libs // all libraries and headers.

|-- armv5 // library for armv5 (i.mx 2xx). only support soft float point

|-- armv6 // library for armv6 (i.mx 3xx), soft fpu version

|-- armv7-a // library for armv7-a (i.mx5xx and i.mx6xx), hardware fpu version

|--arm // library for armv7-a, using arm instruction

|--hard // library for armv7-a, using hardware fp

|--a9 // library for armv7-a, use a9 optimization

|-- neon // library for armv7-a, use neon as fpu

|-- vfpv3 // library for armv7-a, use vfpv3 as fpu.

|--softfp // library for armv7-a, using soft fp

|--a9 // library for armv7-a, use a9 optimization

|-- neon // library for armv7-a, use neon as fpu

|-- vfpv3 // library for armv7-a, use vfpv3 as fpu.

|-- thumb // library for armv7-a, using thumb-2 instruction instead of arm.

|-- lib //default library. It can be used for armv4t and above.

|-- usr

|-- include //header files for the application development

|-- lib //three-part library and static built library Freescale

3 HOW TO USE

3.1 GCC

As a multilib toolchain, the different gcc options link to different binary library. For example, Compile test.c with:

a) arm-none-linux-gnueabi-gcc -Wall test.c -o test

b) arm-none-linux-gnueabi-gcc -Wall -march=armv7-a -mfpu=neon -mfloat-abi=softfp –mcpu=cortex-a9 test.c -o test

case a) will link to default binary lib directory, which located at: multi-libs/lib

case b) will link to armv7-a neon binary lib directory, which locate at: multi-libs/armv7-a/arm/softfp/a9/lib

You can check multilib toolchain information by:

a) arm-none-linux-gnueabi-gcc -print-multi-lib

b) arm-none-linux-gnueabi-gcc “your gcc options” -print-multi-directory

c) arm-none-linux-gnueabi-gcc “your gcc options” -print-search-dirs

command a) shows all the multilib support.

command b) shows the library directory corresponding to the gcc options you provided. But it cannot handle options alias.

For example, “-mcpu=cortex-a9” is set as the same effect to -march=armv7-a. But the result above command shows is not what expected.

Although the information it shows is not correct, gcc will link to the correct library when the alias options are used, you can confirm the behavior by command c).

command c) shows the library search patch corresponding to the gcc options you provided.

Gcc options for different library directory:

Directory path	Target CPU model	GCC options
lib, usr/lib	armv4t	
armv5/lib armv5/usr/lib	armv5te, i.mx2xx	-march=armv5te
armv6/lib armv6/usr/lib	armv6, i.mx3xx	-march=armv6 –mfpu=vfp –mfloat-abi=softfp
armv7-a/default/lib armv7-a/default/usr/lib	armv7-a, i.mx5xx, i.mx6xx	-march=armv7-a
armv7-a/thumb/a9/lib armv7-a/thumb/a9/usr/lib	armv7-a, i.mx6xx	-march=armv7-a –mfpu=neon –mthumb –mfloat-abi=softfp –mcpu=cortex-a9
armv7-a/thumb/hard/lib armv7-a/thumb/hard/usr/lib	armv7-a, i.mx5xx, i.mx6xx	-march=armv7-a –mfpu=neon –mthumb –mfloat-abi=hard
armv7-a/thumb/softfp/lib armv7-a/thumb/softfp/usr/lib	armv7-a, i.mx5xx, i.mx6xx	-march=armv7-a –mfpu=neon –mthumb –mfloat-abi=softfp
armv7-a/arm/hard/a9/lib armv7-a/arm/hard/a9/usr/lib	armv7-a, i.mx6xx	-march=armv7-a –mfpu=neon –mfloat-abi=hard –mcpu=cortex-a9

armv7-a/arm/hard/neon/lib armv7-a/arm/hard/neon/usr/lib	armv7-a, i.mx5xx, i.mx6xx	-march=armv7-a –mfpu=neon –mfloat-abi=hard
armv7-a/arm/hard/vfpv3/lib armv7-a/arm/hard/vfpv3/usr/lib	armv7-a, i.mx5xx, i.mx6xx	-march=armv7-a –mfpu=vfpv3 –mfloat-abi=hard
armv7-a/arm/softfp/a9/lib armv7-a/arm/softfp/a9/usr/lib	armv7-a, i.mx6xx	-march=armv7-a –mfpu=neon –mfloat-abi=softfp –mcpu=cortex-a9
armv7-a/arm/softfp/neon/lib armv7-a/arm/softfp/neon/usr/lib	armv7-a, i.mx5xx, i.mx6xx	-march=armv7-a –mfpu=neon –mfloat-abi=softfp
armv7-a/arm/softfp/vfpv3/lib armv7-a/arm/softfp/vfpv3/usr/lib	armv7-a, i.mx5xx, i.mx6xx	-march=armv7-a –mfpu=vfpv3 –mfloat-abi=softfp

Notes: hard float is not compatible with soft float(compiled with -mfloat-abi=softfp or -mfloat-abi=soft)

“Directory path” is the relative path to:

“gcc-4.6.2-glibc-2.13-multilib-2011.12/fsl-linaro-toolchain/arm-fsl-linux-gnueabi/multi-libs”

“Target CPU model” means the cpu model and above version, not just only the specific model.

“GCC options” means the options must be provided, the other options can add base on this.

3.2 APPLICATION DEBUG TOOLS

strace and strace-graph:

strace is used to trace system call invoked by the application and library.

It shows the system call parameters and results. It can can also provide some performance statistic among system calls.

ltrace:

ltrace is similar to strace, except ltrace trace library call.

duma:

duma is a memory access debug tool. It helps to solve memory access error (segmentation fault) and memory leak problem.dmalloc

dmalloc can be used to debug application memory leak. It needs the library(glibc) support.

You can check this reference for more:
http://dmalloc.com/docs/latest/online/dmalloc_toc.html

4 APPENDIX

4.1 TOOLCHAIN TEST RESULT

Some basic tests have be done to validate the toolchain, the below is the result.

4.1.1 TEST ON MX6Q (792 MHZ)

4.1.1.1 MEMCPY TEST:

The meaning of x/y is:

X is the test result of:

source and destination memory address changes at every test loop, and X is the average value of the results.

in code, it says:

```
for (i=0; i < loops; i++)
for (j=0; j < offset1; j++)
for (k=0; k < offset2; k++)
        {  memcpy(dest + j, src + k, size);        memcpy(src + j, dest + k, size);    }
```

Y stand for:

test with same source and destination address every loop, and caculate the average value of them.

in code, it says:

```
for (i=0; i < loops; i++)
for (j=0; j < offset1; j++)
for (k=0; k < offset2; k++)
        {       memcpy(dest , src, size);        memcpy(src, dest, size);    }
```

copy size	gcc 4.4.4 multilib (fsl)	android 2.3	gcc 4.6.2 multilib (linaro source, fsl built)
3 Bytes	103.3 MB/s / 107.3 MB/s	107.1 MB/s / 107.2 MB/s	107.1 MB/s / 107.3 MB/s

4 Bytes	144.7 MB/s / 159.2 MB/s	123.9 MB/s / 125.9 MB/s	123.9 MB/s / 125.8 MB/s
5 bytes	161.8 MB/s / 182.8 MB/s	164.8 MB/s / 171.1 MB/s	164.8 MB/s / 171.1 MB/s
7 bytes	243.8 MB/s / 273.4 MB/s	228.4 MB/s / 261.9 MB/s	228.4 MB/s / 261.9 MB/s
8 bytes	269.8 MB/s / 359.1 MB/s	266.8 MB/s / 262.1 MB/s	266.8 MB/s / 262.1 MB/s
11 bytes	338.2 MB/s / 467.3 MB/s	341.8 MB/s / 411.3 MB/s	341.8 MB/s / 411.3 MB/s
12 bytes	380.4 MB/s / 608.0 MB/s	325.6 MB/s / 343.5 MB/s	325.6 MB/s / 343.6 MB/s
15 bytes	440.2 MB/s / 760.1 MB/s	385.6 MB/s / 469.0 MB/s	385.6 MB/s / 468.2 MB/s
16 bytes	315.6 MB/s / 475.4 MB/s	254.6 MB/s / 360.4 MB/s	252.1 MB/s / 360.5 MB/s
24 bytes	433.8 MB/s / 740.9 MB/s	373.8 MB/s / 505.1 MB/s	366.9 MB/s / 505.2 MB/s
31 bytes	528.8 MB/s / 995.5 MB/s	463.4 MB/s / 611.1 MB/s	468.5 MB/s / 611.1 MB/s
4096	1533.9 MB/s / 1650.3 MB/s	1525.1 MB/s / 1604.9 MB/s	1525.7 MB/s / 1604.9 MB/s
6144	1541.6 MB/s / 1636.9 MB/s	1535.9 MB/s / 1614.4 MB/s	1536.5 MB/s / 1614.4 MB/s
64k	801.4 MB/s / 855.6 MB/s	808.1 MB/s / 876.5 MB/s	808.1 MB/s / 876.5 MB/s
96k	801.3 MB/s / 856.6 MB/s	793.2 MB/s / 870.9 MB/s	793.2 MB/s / 870.8 MB/s
128k	803.2 MB/s / 859.3 MB/s	786.0 MB/s / 870.2 MB/s	786.1 MB/s / 870.2 MB/s
256k	807.5 MB/s / 864.5	779.3 MB/s / 869.9	779.3 MB/s / 869.9 MB/s

	MB/s	MB/s	
1M	353.2 MB/s / 449.7 MB/s	396.2 MB/s / 445.4 MB/s	396.2 MB/s / 445.4 MB/s
2M	356.6 MB/s / 483.0 MB/s	407.6 MB/s / 468.4 MB/s	407.6 MB/s / 468.4 MB/s

FLOAT POINT TEST:

soft float point (-mfloat-abi=softfp)

##

Single Precision C/C++ Whetstone Benchmark

Calibrate

0.03 Seconds 1 Passes (x 100)

0.16 Seconds 5 Passes (x 100)

0.81 Seconds 25 Passes (x 100)

4.05 Seconds 125 Passes (x 100)

Use 3087 passes (x 100)

Single Precision C/C++ Whetstone Benchmark

Loop content	Result	MFLOPS	MOPS	Seconds
N1 floating point	-1.12475013732910156	124.257		0.477
N2 floating point	-1.12274742126464844	103.983		3.990
N3 if then else	1.00000000000000000		101.109	3.160
N4 fixed point	12.00000000000000000		162.610	5.980
N5 sin,cos etc.	0.49911010265350342		12.081	21.260
N6 floating point	0.99999982118606567	48.960		34.010
N7 assignments	3.00000000000000000		47.540	12.000
N8 exp,sqrt etc.	0.75110864639282227		6.012	19.100
MWIPS		308.771	99.977	

hard float point (-mfloat-abi=hard)

root@freescale /work$ ./whet_hard

##

Single Precision C/C++ Whetstone Benchmark

Calibrate

0.02 Seconds 1 Passes (x 100)

0.16 Seconds 5 Passes (x 100)

0.81 Seconds 25 Passes (x 100)

4.05 Seconds 125 Passes (x 100)

Use 3087 passes (x 100)

Single Precision C/C++ Whetstone Benchmark

Loop content	Result	MFLOPS	MOPS	Seconds
N1 floating point	-1.12475013732910156	124.257		0.477
N2 floating point	-1.12274742126464844	103.983		3.990
N3 if then else	1.00000000000000000	100.790		3.170
N4 fixed point	12.00000000000000000	162.610		5.980
N5 sin,cos etc.	0.49911010265350342	12.420		20.680
N6 floating point	0.99999982118606567	48.461		34.360
N7 assignments	3.00000000000000000	47.540		12.000
N8 exp,sqrt etc.	0.75110864639282227	5.975		19.220
MWIPS		309.080	99.877	

4.1.2 TEST ON MX53 SMD (1 GHZ) , 2.6.35 KERNEL

4.1.2.1 MEMORY TEST:

copy size	gcc 4.4.4 multilib (fsl)	android 2.3	gcc 4.6.2 multilib (linaro source, fsl build)
3 Bytes	166.0 MB/s / 197.7 MB/s	105.4 MB/s / 124.0 MB/s	120.4 MB/s / 124.0 MB/s
4 Bytes	177.7 MB/s / 263.6 MB/s	140.7 MB/s / 144.2 MB/s	140.2 MB/s / 144.3 MB/s
5 bytes	219.8 MB/s / 328.9 MB/s	91.7 MB/s / 89.6 MB/s	91.8 MB/s / 89.5 MB/s
7 bytes	281.3 MB/s / 461.3 MB/s	131.0 MB/s / 125.4 MB/s	131.3 MB/s / 125.4 MB/s
8 bytes	278.6 MB/s / 527.2	281.6 MB/s / 288.8	281.2 MB/s / 288.9 MB/s

	MB/s	MB/s	
11 bytes	358.2 MB/s / 723.9 MB/s	223.7 MB/s / 216.6 MB/s	223.8 MB/s / 216.5 MB/s
12 bytes	352.6 MB/s / 790.8 MB/s	381.6 MB/s / 390.8 MB/s	381.6 MB/s / 390.8 MB/s
15 bytes	425.3 MB/s / 981.6 MB/s	266.5 MB/s / 254.9 MB/s	266.3 MB/s / 254.9 MB/s
16 bytes	343.8 MB/s / 719.2 MB/s	207.0 MB/s / 467.0 MB/s	207.1 MB/s / 465.8 MB/s
24 bytes	476.5 MB/s / 1078.9 MB/s	307.0 MB/s / 635.4 MB/s	307.2 MB/s / 634.9 MB/s
31 bytes	623.1 MB/s / 1387.6 MB/s	407.3 MB/s / 450.3 MB/s	407.6 MB/s / 450.1 MB/s
4096	4372.3 MB/s / 4537.7 MB/s	4187.5 MB/s / 4379.5 MB/s	4241.3 MB/s / 4450.0 MB/s
6144	4486.9 MB/s / 4594.2 MB/s	4381.3 MB/s / 4538.0 MB/s	4312.9 MB/s / 4524.5 MB/s
64k	1968.4 MB/s / 1929.7 MB/s	1917.8 MB/s / 1689.9 MB/s	1903.7 MB/s / 1659.3 MB/s
96k	1885.2 MB/s / 1760.1 MB/s	1902.6 MB/s / 1675.9 MB/s	1880.9 MB/s / 1645.1 MB/s
128k	1155.4 MB/s / 1153.6 MB/s	1174.6 MB/s / 1129.5 MB/s	1149.1 MB/s / 1142.8 MB/s
256k	503.4 MB/s / 522.9 MB/s	507.9 MB/s / 524.6 MB/s	504.0 MB/s / 528.0 MB/s
1M	482.3 MB/s / 503.2 MB/s	466.1 MB/s / 483.6 MB/s	471.5 MB/s / 482.7 MB/s
2M	496.9 MB/s / 508.8 MB/s	496.1 MB/s / 504.4 MB/s	496.5 MB/s / 507.1 MB/s

4.1.2.2 SOFT FLOAT POINT TEST (SOFTFP)

Single Precision C/C++ Whetstone Benchmark

Calibrate

0.06 Seconds	1 Passes (x 100)
0.34 Seconds	5 Passes (x 100)
1.72 Seconds	25 Passes (x 100)
8.61 Seconds	125 Passes (x 100)

Use 1451 passes (x 100)

Single Precision C/C++ Whetstone Benchmark

Loop content	Result	MFLOPS	MOPS Seconds
N1 floating point	-1.12475013732910156	44.718	0.623
N2 floating point	-1.12274742126464844	42.302	4.610
N3 if then else	1.00000000000000000	158.083	0.950
N4 fixed point	12.00000000000000000	164.412	2.780
N5 sin,cos etc	0.49911010265350342	4.035	29.920
N6 floating point	0.99999982118606567	25.577	30.600
N7 assignments	3.00000000000000000	45.994	5.830
N8 exp,sqrt etc	0.75110864639282227	2.210	24.420
MWIPS	0.75110864639282227	145.488	99.733

4.1.2.3 HARD FLOAT POINT TEST(HARDFP)

Single Precision C/C++ Whetstone Benchmark

Calibrate

0.07 Seconds 1 Passes (x 100)

0.34 Seconds 5 Passes (x 100)

1.73 Seconds 25 Passes (x 100)

8.66 Seconds 125 Passes (x 100)

Use 1442 passes (x 100)

Single Precision C/C++ Whetstone Benchmark

Loop content	Result	MFLOPS	MOPS	Seconds
N1 floating point	-1.12475013732910156	44.728		0.619
N2 floating point	-1.12274742126464844	41.411		4.680
N3 if then else	1.00000000000000000		158.774	0.940
N4 fixed point	12.00000000000000000		163.982	2.770
N5 sin,cos etc.	0.49911010265350342		4.156	28.870
N6 floating point	0.99999982118606567	24.133		32.230
N7 assignments	3.00000000000000000		46.104	5.780
N8 exp,sqrt etc.	0.75110864639282227		2.255	23.790
MWIPS		144.664	99.679	