AN10902

Using the LPC32xx VFP

Rev. 01 — 9 February 2010 Application note

Document information

Info	Content
Keywords	Vector Floating Point (VFP) coprocessor, LPC32x0, LPC3180
Abstract	This application note describes how to use LPC32x0 Vector Floating Point (VFP) coprocessor.

Revision history

Rev	Date	Description
01	20100209	Initial version.

Contact information

For additional information, please visit: http://www.nxp.com

For sales office addresses, please send an email to: salesaddresses@nxp.com

1. Introduction

The ARM processor core does not contain floating-point hardware. In typical ARM systems where floating point support is needed, software emulation is used to provide the floating point capability. Although software emulation works well, it does offer good CPU performance.

ARM offers a hardware based Vector Floating Point (VFP) coprocessor that accelerates floating point operation. The ARM VFP supports single-precision and double-precision add, subtract, multiply, divide, multiply-accumulate operations and divide/square root at CPU clock speeds. The ARM VFP can be used to increase performance of imaging applications such as scaling, 2D and 3D transforms, font generation, digital filters, or any application that uses floating point operations. Because the ARM VFP is a coprocessor developed and supported by ARM, it has support in various tool chains, RTOS's, and operating systems such as the Keil MDK development environment or Linux. The ARM VFP complies with the IEEE 754 standard.

Many real-time control applications in the industrial and automotive fields benefit from the dynamic range and precision of floating-point offered by the ARM VFP. Automotive powertrain, anti-lock braking, traction control, and active suspension systems are all mission-critical applications where precision and predictability are essential requirements. The next generation of consumer products such as Internet appliances, set-top boxes, and home gateways, can directly benefit from the ARM VFP.

The LPC3180 and LPC32x0 series devices from NXP offer an ARM926EJS core with the ARM VFP coprocessor. Use of the ARM VFP instead of software emulated floating point improves system performance by reducing CPU load for floating point operations. More CPU work can be done in the same time reducing system power usage and allowing the CPU to better handle other tasks.

The LPC32x0 can be used in the following example applications:

- Medical monitors
- Portable diagnostics
- Portable GPS
- Fingerprint/image identification
- Security systems
- Automotive control applications
- ABS, traction control, and active suspension
- 3D graphics

Fig 1 shows the connection of the VFP in the LPC32x0 CPU subsystem.

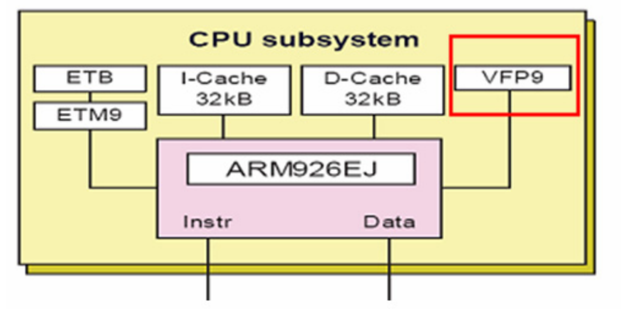


Fig 1. LPC32x0 CPU subsystem block diagram

1.1 VFP architecture

The ARM VFP coprocessor uses coprocessor 10 for single-precision instructions and coprocessor 11 for double-precision instructions. In some cases, such as mixed-precision instructions, the coprocessor ID represents the destination precision. Fig 2 shows the internal connection between ARM926E core and VFP.

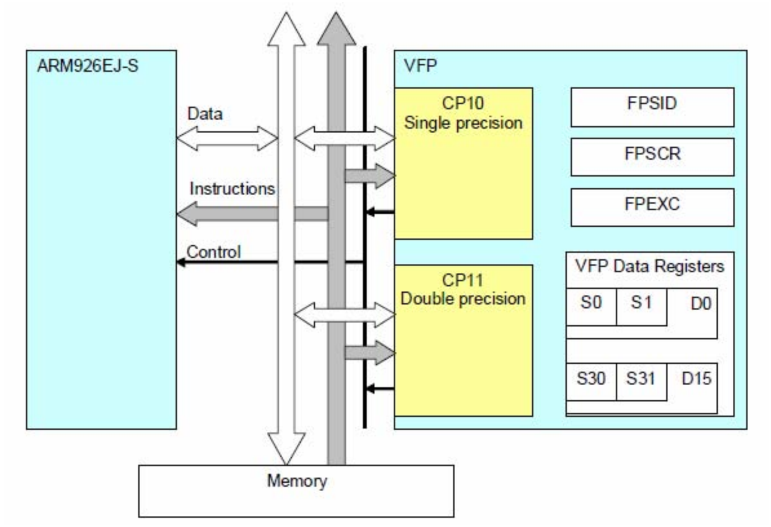


Fig 2. LPC32x0 VFP internal connection

The ARM VFP contains thirty-two 32-bit registers. Each register can store either a single-precision floating-point number or an integer. Any consecutive pair of registers can store a double-precision floating-point number. Because a load or store operation does not modify the data, the registers can also be used as secondary data storage by another application that does not use floating-point values. Fig 3 shows the overlapping of single-precision and double precision. For example, D0 uses the same memory as S0/S1.

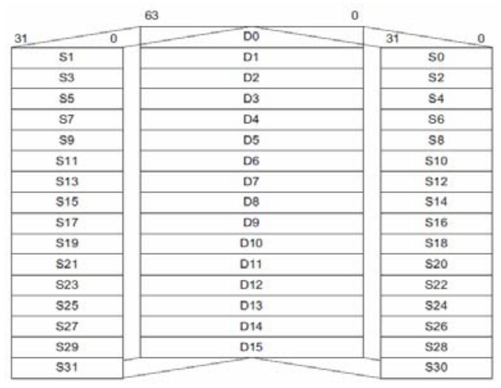


Fig 3. LPC32x0 VFP data register

The VFP registers can be accessed through the ARM core instructions MRC (Move to Register form Coprocessor), MCR (Move to Coprocessor from Register), MCRR (double precision MRC), or MRCC (double precision MCR). ARM compilers offer a number of assembly language instruction aliases that handle these operations such as FMSR (Move from ARM register to VFP single precision register), FMRS (Move from VFP single precision register to ARM register), etc. See the *ARM VFP9-S Technical Reference Manual* for a complete list of instructions.

The VFP9-S coprocessor provides floating-point functionality through a combination of hardware and software support. Normally, the ARM VFP hardware executes floating point instructions completely in hardware. However, the ARM VFP can, under certain circumstances, refuse to accept a floating point instruction, causing the ARM Undefined Instruction exception. This is known as bouncing the instruction.

There are two main reasons for bouncing an instruction:

- potential floating point arithmetic exceptions
- illegal instructions.

When an instruction bounces, software installed on the ARM Undefined instruction vector determines why the VFP9 coprocessor rejected the instruction and takes appropriate remedial action. This software is called the VFP support code. The support code has two components:

- a library of routines that perform floating-point arithmetic functions
- a set of exception handlers that process exceptional conditions.

See the Firmware Suite Reference Guide (AFS) for details of support code.

2. How to use VFP

Using VFP for floating point operations requires a compiler that supports ARM VFP instructions, the implementation of a library to handle the bounced instruction trap, initialization of the VFP, and VFP context switching support (if used in an OS).

Operating systems such as Linux and tool chains such as Keil MDK provide the support necessary to seamlessly support VFP in software. Complete initialization of the library for bounced instructions and VFP initialization is beyond the scope of this document. More information about VFP can be found in the *ARM VFP9-S Technical Reference Manual* if you need to develop new support for the ARM VFP.

2.1 VFP support with various tool chains and operating systems

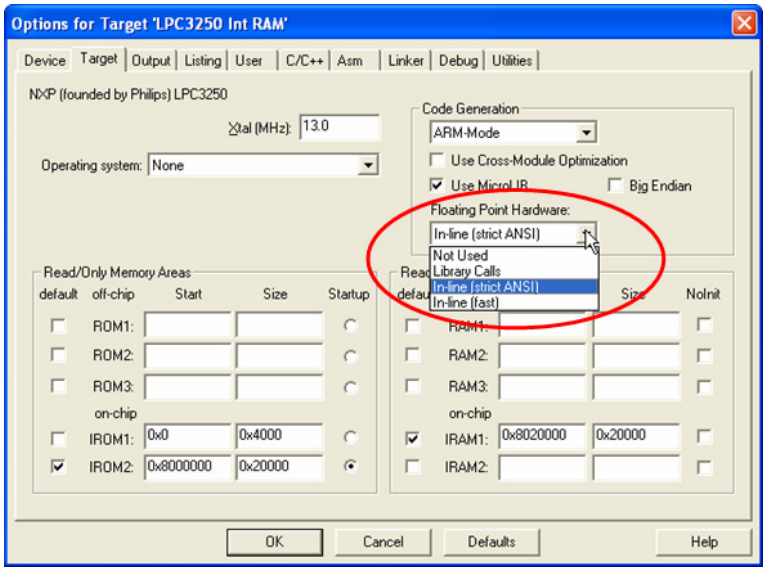
2.1.1 For Keil MDK

The Keil MDK development environment uses the ARM RealView compiler. More detail can be found in the *RealView Compiler User Guide.* Table 1 contains a description of the VFP compiler options available.

Table 1. MDK VFP compiler options

Option	Description
--fpu=vfp	This is a synonym for vfpv2.
--fpu=vfpv2	Selects a hardware vector floating-point unit conforming to architecture VFPv2.
--fpu=softvfp	Selects the software floating-point library fplib. This is the default if you select a CPU that does not have a FPU.
--fpu=softvfp+vfpv2	Selects a floating-point library with software floating-point linkage that can use VFPv2 instructions. Select this option if you are interworking Thumb code with ARM code on a system that implements a VFP unit.

More detailed information about floating-point linkage and computations for different compiler options can be found in the *RealView Compiler User Guide.* To use the LPC32x0 VFP, users can follow the setting as described in Fig 4 and Fig 5.


Options for Target 'LPC3250 Int RAM'
Device | Target | Output | Listing | User | C/C++ | Asm | Linker | Debug | Utilities
OK
Cancel
Defaults
Help

NXP (founded by Philips) LPC3250

Xtal (MHz): 13.0

Operating system: None

Code Generation: ARM-Mode

Use Cross-Module Optimization [ ]

Use MicroLIB [✓]

Big Endian [ ]

Floating Point Hardware: In-line (strict ANSI)

Not Used
Library Calls
[✓ In-line (strict ANSI)]
In-line (fast)

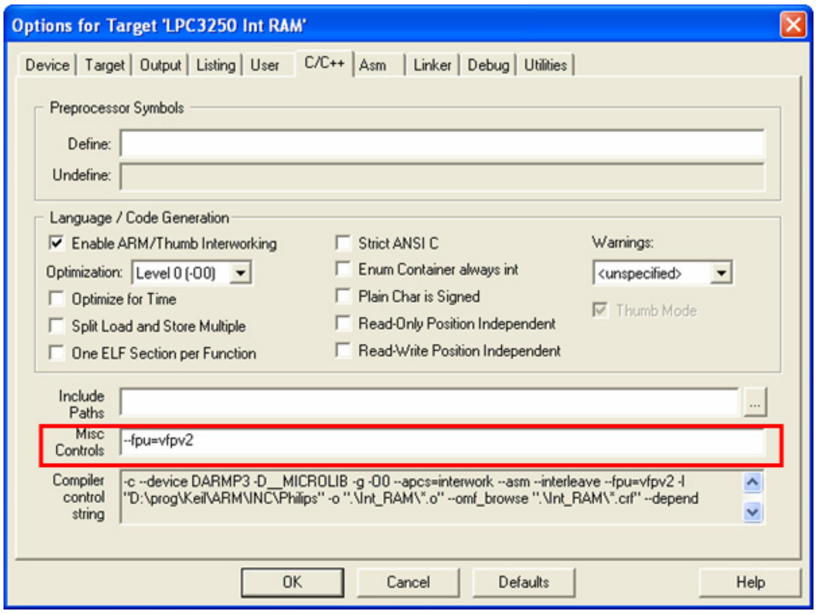
Read/Only Memory Areas

default	off-chip	Start	Size	Startup
☐	ROM1:			○
☐	ROM2:			○
☐	ROM3:			○
	on-chip			
☐	IROM1:	0x0	0x4000	○
✓	IROM2:	0x8000000	0x20000	●

Read/Write Memory Areas

default	off-chip	Start	Size	NoInit
☐	RAM1:			☐
☐	RAM2:			☐
☐	RAM3:			☐
	on-chip			
✓	IRAM1:	0x8020000	0x20000	☐
☐	IRAM2:			☐

Fig 4. MDK setting



Define:

Undefine:

[✓] Enable ARM/Thumb Interworking

Optimization: Level 0 (-O0)

[ ] Optimize for Time

[ ] Split Load and Store Multiple

[ ] One ELF Section per Function

[ ] Strict ANSI C

[ ] Enum Container always int

[ ] Plain Char is Signed

[ ] Read-Only Position Independent

[ ] Read-Write Position Independent

Warnings: <unspecified>

[✓] Thumb Mode

Include Paths:

Misc Controls: --fpu=vfpv2

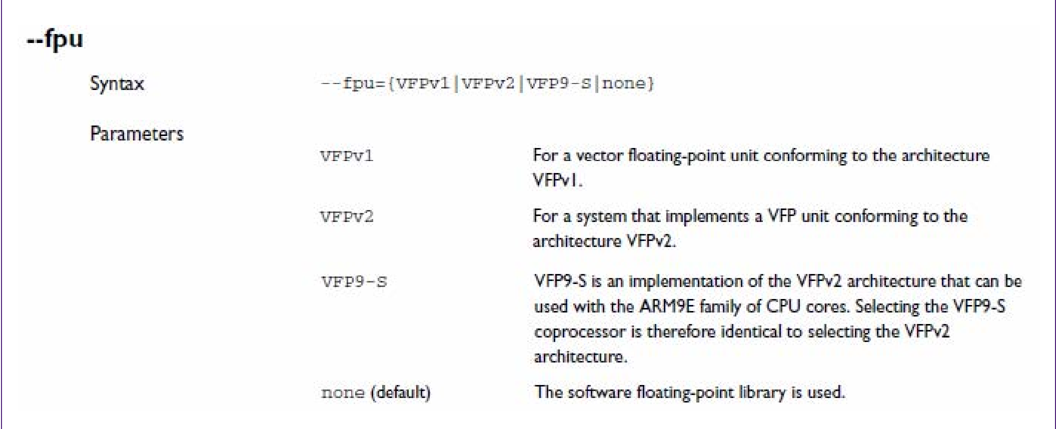
Compiler control string:

```
-c --device DARMP3 -D__MICROLIB -g -O0 --apcs=interwork --asm --interleave --fpu=vfpv2 -I
"D:\prog\Keil\ARM\INC\Philips" -o ".\Int_RAM\*.o" --omf_browse ".\Int_RAM\*.crf" --depend
```

Fig 5. MDK setting

2.1.2 For IAR

More information on the IAR compiler can be found in the ***IAR C/C++ Development Guide-Compiling and linking***. The following is the VFP compiler option description.

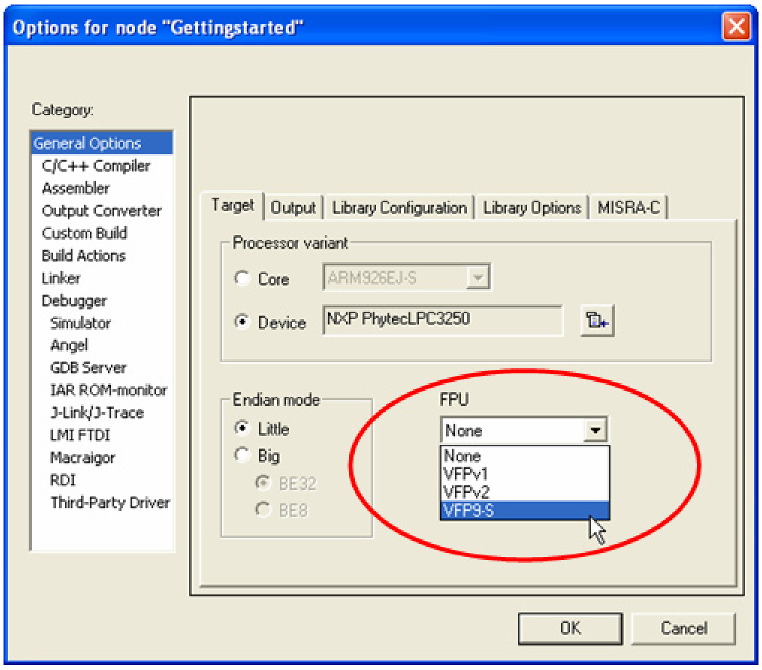


--fpu

Syntax	`--fpu={VFPv1|VFPv2|VFP9-S|none}`	
Parameters		
	`VFPv1`	For a vector floating-point unit conforming to the architecture VFPv1.
	`VFPv2`	For a system that implements a VFP unit conforming to the architecture VFPv2.
	`VFP9-S`	VFP9-S is an implementation of the VFPv2 architecture that can be used with the ARM9E family of CPU cores. Selecting the VFP9-S coprocessor is therefore identical to selecting the VFPv2 architecture.
	`none` (default)	The software floating-point library is used.

Fig 6. IAR VFP compiler options

Use this option to generate code that carries out floating-point operations using a Vector Floating Point (VFP) coprocessor. By selecting a VFP coprocessor, you will override the use of the software floating-point library for all supported floating-point operations. To use LPC3200 VFP, users can follow the setting as described in Fig 7.



Options for node "Gettingstarted"

Category: General Options

Target | Output | Library Configuration | Library Options | MISRA-C

Processor variant: Device NXP PhytecLPC3250

Endian mode: Little

FPU: None

Fig 7. IAR setting

2.1.3 For GNU compiler (GCC)

The GNU compiler and assembler have some ARM Dependent Features. The following description of the VFP compiler option is from *Using as, the GNU Assembler Version 2.18.50*.

-mfpu=*floating-point-format*

This option specifies the floating point format to assemble for. The assembler will issue an error message if an attempt is made to assemble an instruction which will not execute on the target floating point unit.

The *floating-point-format* options recognized are: softfpa, fpe, fpe2, fpe3, fpa, fpa10, fpa11,arm7500fe, softvfp, softvfp+vfp, vfp, vfp10, vfp10-r0, vfp3 vfp9, vfpxd,vfpv2 vfpv3 vfpv3-d16 arm1020t, arm1020e, arm1136jf-s, maverick and neon.

In addition to determining which instructions are assembled, this option also affects the way in which the ***.double*** assembler directive behaves when assembling little-endian code.

The default is dependent on the processor selected. For Architecture 5 or later, the default is to assembler for VFP instructions; for earlier architectures the default is to assemble for FPA instructions.

2.1.4 Linux applications with VFP support

To use VFP with Linux, you need to use a version of GCC that has support for the ARM VFP coprocessor. In addition to GCC, ARM VFP support will also need to be enabled in the Linux kernel configuration.

GCC with VFP support will enable the floating point instructions in the assembled Linux code. For floating point operations that cannot be handled by the VFP, a software floating point library and suitable software float ABI must also be supplied. The following compiler options enable VFP support with software handled through the softfp ABI:

-mfpu=vfp –mfloat-abi=softfp

Under Linux, VFP performs about 5x to 6x faster in single precision operations when compared to software emulated floating point. For double precision operations, VFP performs about 11x to 12x faster than software emulated floating point. This benchmark can vary based on selected kernel and system options, network load, and all types of system factors.

2.1.5 WinCE applications with VFP support

Because the Platform Builder 6.0 compiler does not support VFP, users have to write their own VFP software library. ARM VFPv2 Floating Point Support Library for Windows Embedded CE 6.0 can be downloaded at http://www.nxp.com/redirect/arm.com. Based on the library, VFP arithmetic can be implemented.

2.2 Enable VFP

There are 5 control and status registers associated with the VFP9-S

• Floating-point system ID register (FPSID)

• Floating-point status and control register (FPSID)

• Floating-point exception register (FPEXC).

• Floating-point instruction register (FPINST)

• Floating-point instruction register 2 (FPINST2)

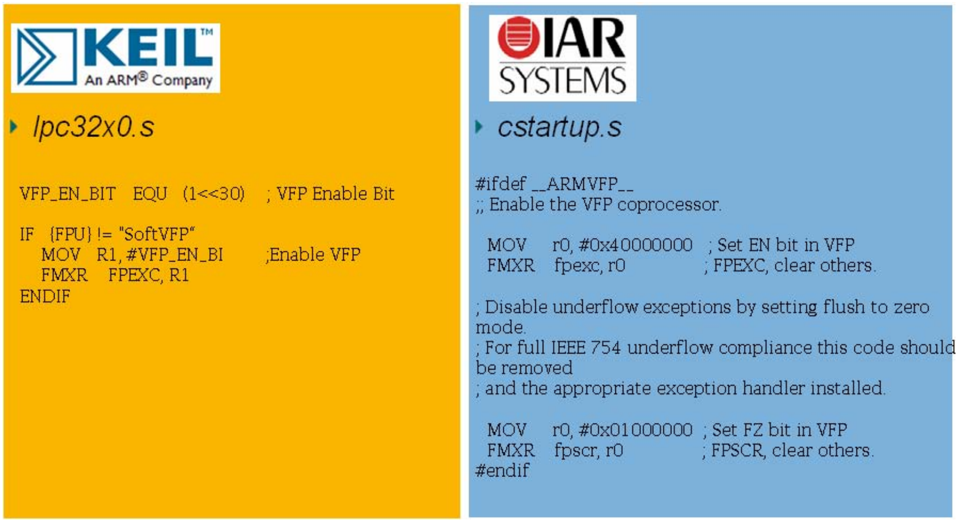
More details about these registers can be found in the *VFP9 Technical Reference Manual.*

On all systems, it is necessary to enable the VFP by setting the VFPEnable (EN) bit in the VFP’s FPEXC register. Until this is done, the VFP coprocessor is disabled and any other access to the VFP causes an undefined instruction exception. On pre-v7 cores, it is necessary to reset the EX bit in this register to clear any pending exceptions. This operation must be carried out in a privileged mode.

In the Keil/IAR startup source file, the code for VFP Enable is as follows:

```
VFPEnable EQU 0x40000000
Enable_VFP FUNCTION   ; Enable VFP itself
    MOV r0,#VFPEnable
    FMXR FPEXC, r0     ; FPEXC = r0
    BX LR
ENDFUNC
```

Fig 8 shows the VFP enable function from Keil/IAR startup source code.



lpc32x0.s

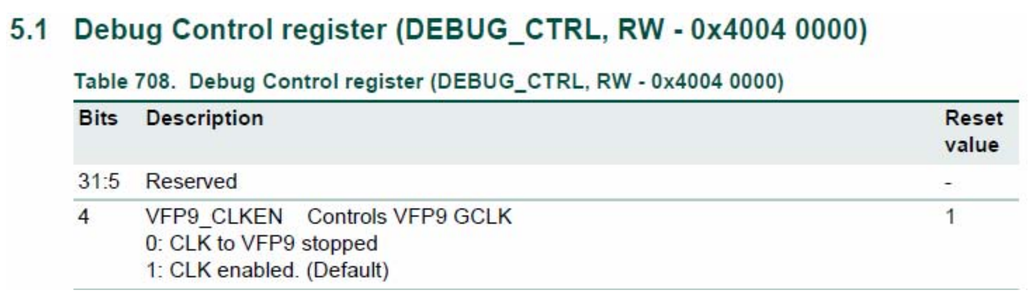
```
VFP_EN_BIT   EQU   (1<<30)    ; VFP Enable Bit

IF   {FPU} != "SoftVFP"
   MOV   R1, #VFP_EN_BI       ;Enable VFP
   FMXR   FPEXC, R1
ENDIF
```

cstartup.s

```
#ifdef __ARMVFP__
;; Enable the VFP coprocessor.

  MOV     r0, #0x40000000   ; Set EN bit in VFP
  FMXR    fpexc, r0         ; FPEXC, clear others.

; Disable underflow exceptions by setting flush to zero
mode.
; For full IEEE 754 underflow compliance this code should
be removed
; and the appropriate exception handler installed.

  MOV     r0, #0x01000000 ; Set FZ bit in VFP
  FMXR    fpscr, r0         ; FPSCR, clear others.
#endif
```

Fig 8. VFP enable in startup

When debugging with VFP, take note of the state of the DEBUG_CTRL register. Bit4 (VFP9_CLKEN) of DEBUG_CTRL controls VFP9 GCLK. If VFP9_CLKEN =0, then VFP clock is stopped. If VFP9_CLKEN =1, VFP clock is enabled. The default is enabled.



5.1 Debug Control register (DEBUG_CTRL, RW - 0x4004 0000)

Table 708. Debug Control register (DEBUG_CTRL, RW - 0x4004 0000)

Bits	Description	Reset value
31:5	Reserved	-
4	VFP9_CLKEN Controls VFP9 GCLK 0: CLK to VFP9 stopped 1: CLK enabled. (Default)	1

Fig 9. DEBUG_CTRL register

2.3 Programming hints

The following are some programming hints when using the LPC3200 VFP unit.

When using the VFP with an RTOS such as uCos-II or FreeRTOS, the RTOS must preserve the VFP register states during the task or interrupt context switch. If the VFP register states aren't saved, only one task is allowed to use VFP. An OS such as Linux will save the VFP states during a context switch as part of the task context block.

An interrupt route which uses VFP should save VFP registers on the stack if the VFP is also used in normal mode or other interrupts.

A floating point operation that cannot be handled by the VFP will cause an undefined instruction. A handler needs to be installed to process to floating point operation in software when this happens. The Keil and Realview toolchains install the software floating point handler as part of their library startup when VFP support is enabled.

The VFP can be powered off through a control register. With the clocks disabled, this unit will consume zero dynamic power.

Avoid cases where two consecutive floating-point instructions use the same floating-point data registers.

3. VFP performance

VFP provides floating-point computation suitable for a wide spectrum of applications such as voice compression and decompression, three-dimensional graphics and digital audio, printers, set-top boxes, and automotive applications. The VFP architecture also supports execution of short vector instructions allowing a single instruction to execute on multiple data at the same time. This is useful in graphics and signal-processing applications by reducing code size and increasing throughput.

John Walker's Floating-point benchmark (Fbench) method is available on the website at http://www.nxp.com/redirect/fourmilab.ch/fbench/fbench.html. The results in (1) are based on that Fbench code running on the LPC3180 internal RAM. From the table, we can see VFP increases performance by about 5x, reduces Code size by about 25 % and also reduces power consumption.

(1) Fbench test results

Micro	Frequency (MHz)	I-Cache	VFP9	Time Normalized	Icore mA	Code & constants kB
LPC3180	200	No	No	48	77.66	35.2
LPC3180	200	Yes	No	28	91.73	35.3
LPC3180	200	No	Yes	8	76.81	26.8
LPC3180	200	Yes	Yes	6	88.8	26.8

To characterize the microcontroller, we selected a set of performance tests from the AutoBench suite of the Embedded Microprocessor Benchmark Consortium (EEMBC).

Fig 10 shows a diagram of the AutoBench/EnergyBench test setup. It consists of three parts: the data acquisition system (DAC), the software development environment, and the target under test. The DAC, from National Instruments, is connected to a PC running the Energy-Bench power and energy measurement software. The software development environment used the Keil™ Integrated Development tools to compile, download, and run the AutoBench benchmarks. We isolated the three power supply voltages to the microprocessor so EnergyBench could measure the power consumed during the Autobench benchmark testing and calculate the total energy consumed during each test.

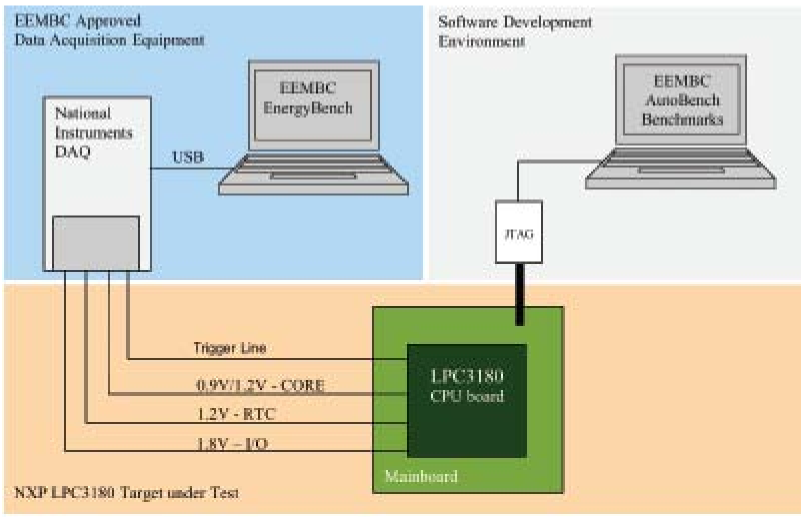


Fig 10. Testing result based on EEMBC

The test results shown in Fig 11 are based on data from http://www.nxp.com/redirect/eembc.org. The effect on performance and energy consumption by using the VFP coprocessor is shown in Fig 10. This graph is quite dramatic in demonstrating the performance effects of an integrated VFP coprocessor. At 208 MHz and with the Icache enabled, the microcontroller LPC3180 runs about 8500 iterations/sec using software floating point, but this value jumps to over 32500 iterations/sec using the VFP coprocessor – for a better than 280 % performance improvement.

The power chart in Fig 11 shows that VFP could reduce power consumption by approximately 15 %.

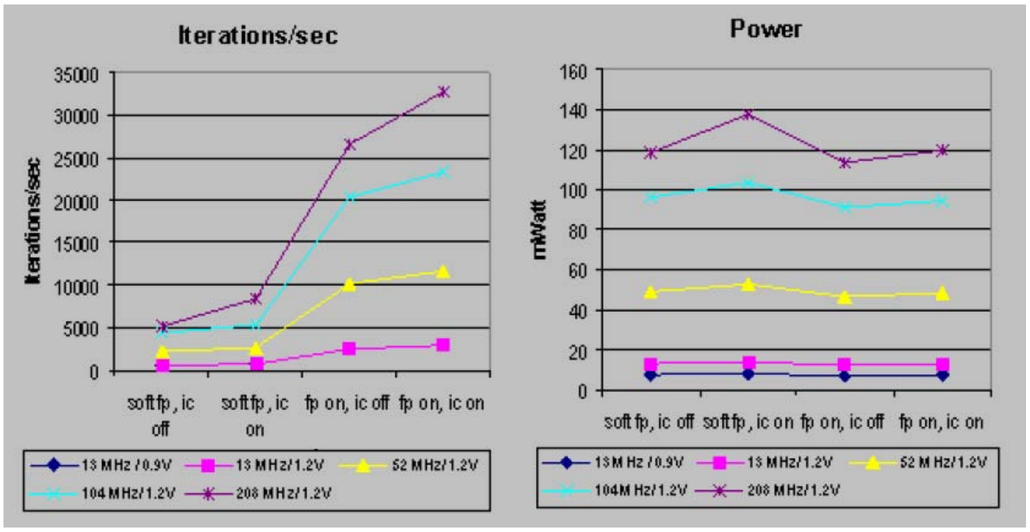


Fig 11. Testing result based on EEMBC

4. Conclusion

From the above test results, it can be seen that the VFP provides improved floating point performance and reduced power consumption and code size as compared to using software only floating point operations.

5. Reference

[1] LPC32x0 User manual, Rev. 1.0

[2] VFP9 Technical Reference Manual, r0p2, ARM Limited

[3] AN_Using VFP with RVDS, ARM Limited

[4] RealView Compilation Tools Compiler Reference Guide, Version 4.0, ARM Limited

[5] AN_Getting Started Linux with LPC3250

6. Legal information

6.1 Definitions

Draft — The document is a draft version only. The content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included herein and shall have no liability for the consequences of use of such information.

6.2 Disclaimers

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms and conditions of commercial sale of NXP Semiconductors.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in medical, military, aircraft, space or life support equipment, nor in applications where failure or malfunction of a NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors accepts no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on a weakness or default in the customer application/use or the application/use of customer's third party customer(s) (hereinafter both referred to as "Application"). It is customer's sole responsibility to check whether the NXP Semiconductors product is suitable and fit for the Application planned. Customer has to do all necessary testing for the Application in order to avoid a default of the Application and the product. NXP Semiconductors does not accept any liability in this respect.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from national authorities.

6.3 Trademarks

Notice: All referenced brands, product names, service names and trademarks are property of their respective owners.

7. Contents

Date of release: 9 February 2010

Document identifier: AN10902_1