ANALOG DEVICES **Engineer-To-Engineer Note EE-136**

Technical Notes on using Analog Devices' DSP components and development tools
Phone: (800) ANALOG-D, FAX: (781) 461-3010, EMAIL: dsp.support@analog.com, FTP: ftp.analog.com, WEB: www.analog.com/dsp

Using the Programmable I/O FLAGS and IOFLAG register on the ADSP-21161

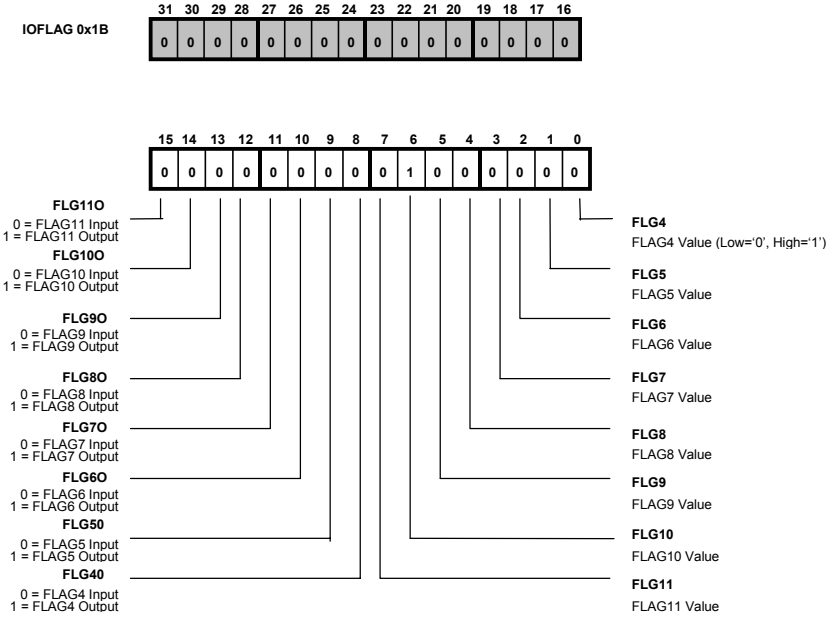
Last modified: 7/26/01 Contributed By: S.B.

This engineer-to-engineer note will discuss the programmable input/output flags pins on the new ADSP-21161 SIMD SHARC. The IOFLAG register is a new IOP register created specifically for controlling these additional input/output flag pins. Some example assembly instructions demonstrating the use of these pins are included as well.

The ADSP-21161 processor consists of twelve general-purpose programmable input/output flag pins - FLAG[11:4]. As outputs, these pins can signal peripheral devices; as inputs, these pins can provide the test for conditional branching. These pins correspond to the FLAG0-FLAG11 pins listed in the datasheet of the device.

The functionality of the FLAG 11-4 pins is similar to that of the FLAG3-0 except for the status and control information both are into one register IOFLAG, vs. the FLAG register. The control and status bits for the FLAG 3-0 are in the MODE2 register and FLAGS register, respectively. Bits 0-7 of IOFLAG reflect the status of the FLAG pins and Bits 15-8 control the direction (input or output) of these flags. A value of '0' programs the flag as an input and a value of '1' programs it as an output. You cannot execute bit wise operations such as BIT TST, BIT CLR, on these flags directly in memory, but you can execute bit wise operations; by first writing to system register such as the USTAT1 - USTAT4 registers.

The following shows the diagram of the IOFLAG Register:



The latest DEF21161.H file included with the VisualDSP ++ 1.0 tools (VisualDSP\211xx\include\ def21161.h) contains newly defined macros (using #define) to program the programmable flags defined in the IOFLAG register. The #define command is used to for two functions: defining symbolic constants and defining macros. For more details on #defines please refer to the Assembler Manual for 21XXX Family DSP's. The "def21161.h" file contains the symbolic names for all the system registers bits for the ADSP-21161. The following has been included in the "def21161.h" file, which defines all the bits in the IOFLAG register:

```
/* IOFLAG - programmable I/O status macro definitions */

#define FLG4   0x00000001     /* FLAG4 value (Low = '0', High = '1')              */
#define FLG5   0x00000002     /* FLAG5 value (Low = '0', High = '1')              */
#define FLG6   0x00000004     /* FLAG6 value (Low = '0', High = '1')              */
#define FLG7   0x00000008     /* FLAG7 value (Low = '0', High = '1')              */
#define FLG8   0x00000010     /* FLAG8 value (Low = '0', High = '1')              */
#define FLG9   0x00000020     /* FLAG9 value (Low = '0', High = '1')              */
#define FLG10  0x00000040     /* FLAG10 value (Low = '0', High = '1')             */
#define FLG11  0x00000080     /* FLAG11 value (Low = '0', High = '1')             */

/* IOFLAG - programmable I/O control macro definitions */

#define FLG4O  0x00000100     /* FLAG4 control ('0' = flag input, '1' = flag output) */
#define FLG5O  0x00000200     /* FLAG5 control ('0' = flag input, '1' = flag output) */
#define FLG6O  0x00000400     /* FLAG6 control ('0' = flag input, '1' = flag output) */
#define FLG7O  0x00000800     /* FLAG7 control ('0' = flag input, '1' = flag output) */
#define FLG8O  0x00001000     /* FLAG8 control ('0' = flag input, '1' = flag output) */
#define FLG9O  0x00002000     /* FLAG9 control ('0' = flag input, '1' = flag output) */
#define FLG10O 0x00004000     /* FLAG10 control ('0' = flag input, '1' = flag output) */
#define FLG11O 0x00008000     /* FLAG11 control ('0' = flag input, '1' = flag output) */
```

Some example assembly language instructions demonstrating the use of these macros are shown below. The instructions show how to configure the flags as output flags and set the flag pins high. After setting the bits, the bits are written to the IOFLAG register.

```
ustat2 = 0x00000000;
bit set ustat2  FLG9O|FLG8O|FLG7O|FLG6O|FLG5O|FLG4O;
dm(IOFLAG) = ustat2;
```

After writing to the register you can toggle the flags with the `bit tgl` command:

```
bit tgl ustat2  FLG9|FLG8|FLG7|FLG6|FLG5|FLG4;
```

The next example shows how we configure the flags as input flags, clear the flag pins, then writing the modified flag settings to the IOFLAG register.

.

```
ustat2 = 0x00000000;
bit clr ustat2 FLG9O|FLG8O|FLG7O|FLG6O|FLG5O|FLG4O;
dm(IOFLAG) = ustat2;
```

System Design Considerations

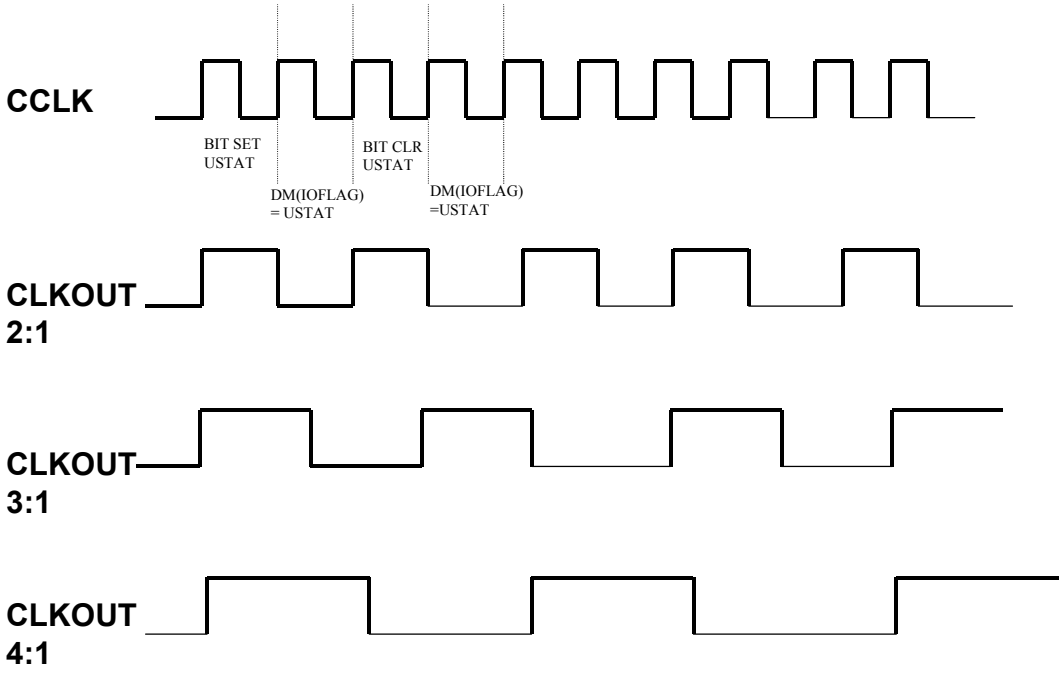
> ***** IMPORTANT NOTE**: All FLAG pins are configured as inputs on reset. When configuring an IOFLAG register flag pin (FLAG11-4) as outputs, you cannot set the FLGx bits 0 to 7 high in the same instruction cycle in which the flag is configured as an output (by setting the FLGxO bits 8 to15 high in the IOFLAG register). If your application requires the flags to be set after they are configured as outputs, then two writes to the IOFLAG register are required, one to configure the flag pin as an output, and another write to set the flag pin high.

Normally, if a flag is sampled or driven periodically, latency issues with respect to when flag pin change occurs are not a concern to the programmer. However, since the flag pins are sampled or driven with respect to the rising edge of CLKOUT (or CLKIN if CLKDBL~ is disabled), it is important that the application program allows enough time in certain programming situations for the flag state to be driven or sampled in the CLKOUT cycle boundary. This is especially true if the flags states are driven at a (core-clock) rate faster than the completion of the CLKOUT cycle boundary. The same also applies for the external device driving the flag pin as an input. The external device must drive the flag pin for at least 1 CLKOUT cycle to guarantee that it is latched properly by the DSP.

When setting (or toggling flag pins) in a loop, we therefore must insert extra NOPs to prevent an "overrun" of setting and clearing a flag pin every cycle (or every other cycle for the IOFLAG register flag pins). For example, if using 2:1 mode; the CCLK runs twice as fast as CLKOUT. Depending on where the CLKOUT cycle boundary is with respect to the instruction writing to I/O flag register it takes up to 2 CCLK cycles before that change is seen external to the processor based on the rising edge of CLKOUT. The same goes for the 3:1 and 4:1 ratios. For 3:1 it takes up to 3 CCLK cycles before the change is seen external to the processor based on the rising edge of CLKOUT and for 4:1 it takes up to 4 CCLK.

Since a core stall does not occur when writing to or reading from FLAG pins synchronized to the slower DSP system clock, NOPs are required. In this case, you need to write extra NOPs to ensure overruns do not occur in the higher clock rates.

This diagram shows the delay in setting (or toggling the flag pins) for different clock modes, specifically 2:1, 3:1, and 4:1.



Setting an IOFLAG output with 2:1 CCLK to CLKOUT ratio

```
LCNTR = 100, DO flag_toggle UNTIL LCE;
               bit tgl ustat1  FLG4O;
flag_toggle:   dm(IOFLAG) = ustat1;
```

In this case, since a CLKOUT transition occurs every two CCLK instruction cycles, no NOPs are required.

Setting an IOFLAG output with 3:1 CCLK to CLKOUT ratio

```
LCNTR = 100, DO flag_toggle UNTIL LCE;
               bit tgl ustat1  FLG4O;
               dm(IOFLAG) = ustat1;
flag_toggle:   nop;
```

In this case, since a CLKOUT transition occurs every three CCLK instruction cycles, one NOP instruction is required to prevent the flag output overrun.

Setting an IOFLAG output with 4:1 CCLK to CLKOUT ratio

```
LCNTR = 100, DO flag_toggle UNTIL LCE;
               bit tgl ustat1  FLG4O;
               dm(IOFLAG) = ustat1;
               nop;
flag_toggle:   nop;
```

In this case, since a CLKOUT transition occurs every four CCLK instruction cycles, two NOP instructions are required to prevent the flag output overrun.