Interfacing to Asynchronous Inputs with the TMS32010

APPLICATION REPORT: SPRA006

Author: Jon Bradley
Digital Signal Processing – Semiconductor Group

Digital Signal Processing Solutions
1989

TEXAS INSTRUMENTS

IMPORTANT NOTICE

Texas Instruments (TI) reserves the right to make changes to its products or to discontinue any semiconductor product or service without notice, and advises its customers to obtain the latest version of relevant information to verify, before placing orders, that the information being relied on is current.

TI warrants performance of its semiconductor products and related software to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

Certain application using semiconductor products may involve potential risks of death, personal injury, or severe property or environmental damage ("Critical Applications").

TI SEMICONDUCTOR PRODUCTS ARE NOT DESIGNED, INTENDED, AUTHORIZED, OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT APPLICATIONS, DEVICES OR SYSTEMS OR OTHER CRITICAL APPLICATIONS.

Inclusion of TI products in such applications is understood to be fully at the risk of the customer. Use of TI products in such applications requires the written approval of an appropriate TI officer. Questions concerning potential risk applications should be directed to TI through a local SC sales office.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards should be provided by the customer to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance, customer product design, software performance, or infringement of patents or services described herein. Nor does TI warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used.

TRADEMARKS

TI is a trademark of Texas Instruments Incorporated.

Other brands and names are the property of their respective owners.

CONTACT INFORMATION

US TMS320 HOTLINE	(281) 274-2320
US TMS320 FAX	(281) 274-2324
US TMS320 BBS	(281) 274-2323
US TMS320 email	dsph@ti.com

Interfacing to Asynchronous Inputs with the TMS32010

Abstract

This report describes interface circuits for the TMS32010 to asynchronous inputs and to external memory devices, such as external ROM or RAM. A description of hardware peripheral interface device produced by Pacific Microcircuits is also included, which eases the TMS32010 interface to both external memory and codec devices.

Product Support on the World Wide Web

Our World Wide Web site at www.ti.com contains the most up to date product information, revisions, and additions. Users registering with TI&ME can build custom information pages and receive new product updates automatically via email.

INTRODUCTION

Interrupt ($\overline{INT}$), reset ($\overline{RS}$), and branch on I/O ($\overline{BIO}$) are inputs to the TMS32010 microprocessor that are typically provided from asynchronous sources within a system. These three inputs are subject to certain considerations in addition to those relevant to signals that are synchronized to TMS32010 operation. Observing these considerations is important to insure reliable operation of these three input functions.

INTERRUPT AND $\overline{BIO}$ INPUTS

Interrupt ($\overline{INT}$) and $\overline{BIO}$ are the two inputs on which synchronization is most important.

The $\overline{BIO}$ input provides a convenient approach to implementing polled I/O on the TMS32010. $\overline{BIO}$, unlike $\overline{INT}$, is not latched internally, so the state of the $\overline{BIO}$ input must be maintained while the BIOZ instruction is executing. The previous state of $\overline{BIO}$ is irrelevant. When properly synchronized to CLKOUT, $\overline{BIO}$ is recognized by the TMS32010 on the next CLKOUT cycle. It should also be noted that the cycle during which $\overline{BIO}$ is sampled by the BIOZ instruction is the first of the two cycles that it takes BIOZ to execute.

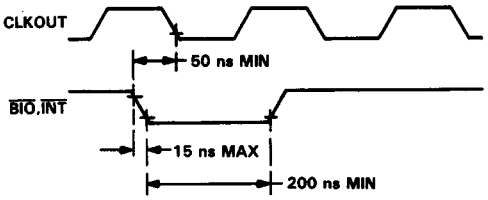
Several aspects of interrupt operation are important for proper implementation of the interrupt function within a system. These fall into three basic categories:

1. Interrupt and input synchronization
2. Internal interrupt control logic
3. Interrupt programming.

Interrupt and $\overline{BIO}$ Input Synchronization

Synchronization of the interrupt and $\overline{BIO}$ inputs is absolutely critical in maintaining proper operation of these functions. This must be accomplished externally since the TMS32010 does not contain the necessary synchronizing logic. This requirement results from the fact that these inputs are sampled within the TMS32010 on the falling edge of CLKOUT. If the input level is changing at this point, the internal circuitry may receive an invalid logic level, causing unpredictable operation. Specifically, the input must be at a stable, valid logic level at least 50 ns before the falling edge of the CLKOUT signal, and must remain stable for at least one full CLKOUT cycle (200 ns for a 20-MHz TMS32010). These timing relationships are shown graphically in Figure 1.

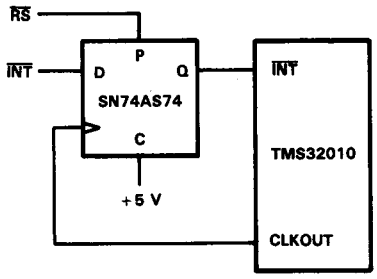
The issue of synchronization has been addressed at length in numerous publications,[1] so details of the subject are not presented here. It can be shown that adequate synchronization for the TMS32010 may be accomplished by simply passing the interrupt input through a sufficiently fast



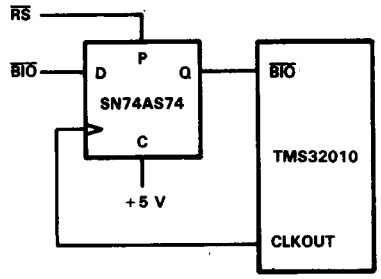
NOTE: Timing measurements are referenced to and from a low voltage of 0.8 V and a high voltage of 2.0 V.

Figure 1. Input Timing Relationships

positive edge-triggered flip-flop clocked by CLKOUT. The flip-flop used must have a clock-rising to output-valid delay of no more than 15 ns. Examples of circuits that accomplish input synchronization using a flip-flop are shown in Figure 2.



a. $\overline{INT}$ Synchronization Circuit



b. $\overline{BIO}$ Synchronization Circuit

Figure 2. Sample Input Synchronization Circuits

All further references to the interrupt or $\overline{BIO}$ functions in this application report assume that this synchronization has been provided. These functions should never be implemented without synchronization; behavior of

unsynchronized inputs is unpredictable and not guaranteed. For example, the interrupt input signal travels through two different paths with unequal propagation delays. If asynchronous interrupts are allowed, interrupts may become disabled before an interrupt can be accepted. Since the pending interrupt is not accepted, the interrupt service routine is not executed. Therefore, probably no EINT instruction will be executed, since this instruction is unlikely to be included in normal program flow and interrupts will remain disabled unless the system is reset.

Internal Interrupt Control Logic

The interrupt input on the TMS32010 is both edge and level triggered, i.e., a low-going pulse on the interrupt input can generate an interrupt (provided the pulse is at least as long as one CLKOUT cycle). However, if the interrupt input goes low and remains low, the interrupt will remain pending until the input goes high and the interrupt is cleared (see Figure 3). This results in several distinct situations which can occur with respect to how interrupts are serviced. Three specific cases are presented for illustration.

In the first case, shown in Figure 4, a low pulse occurs on the interrupt input. The falling edge of this pulse causes IFLAG to be set to one, resulting in IREQ and IACT going high (since $\overline{\text{INTM}}$ is high). Then, when the $\overline{\text{INT}}$ input returns to a one, IFLAG (and therefore, IREQ and IACT) remain high until the interrupt is accepted (according to the rules described in the next section). When the interrupt is accepted, the interrupt service routine is entered, and the IACK signal is pulsed, which disables the interrupt by setting the INTM bit. At the end of the interrupt service routine, interrupts are enabled using the EINT instruction, which clears the INTM bit. The important point in this case is that, once the interrupt input goes high and the IFLAG flip-flop is cleared, control returns directly to the mainline program following the interrupt service routine.

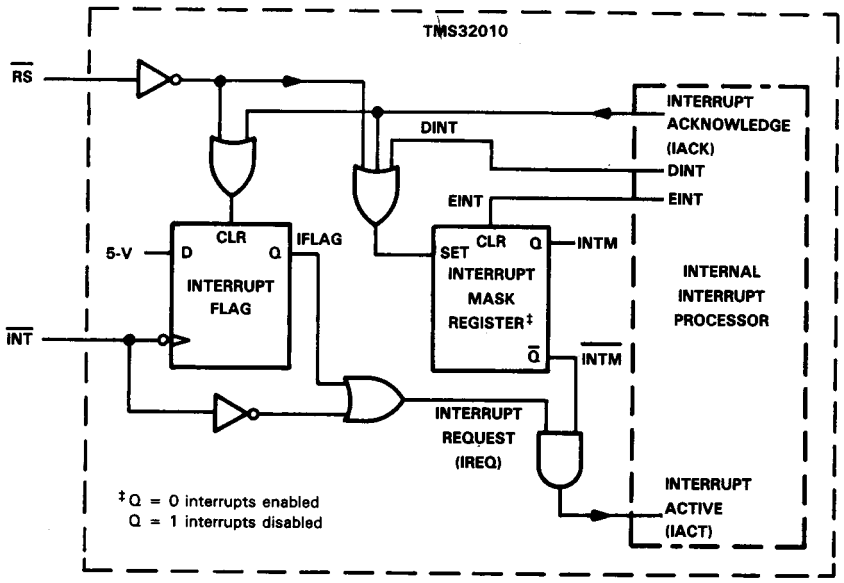


Figure 3. Internal Interrupt Control Logic

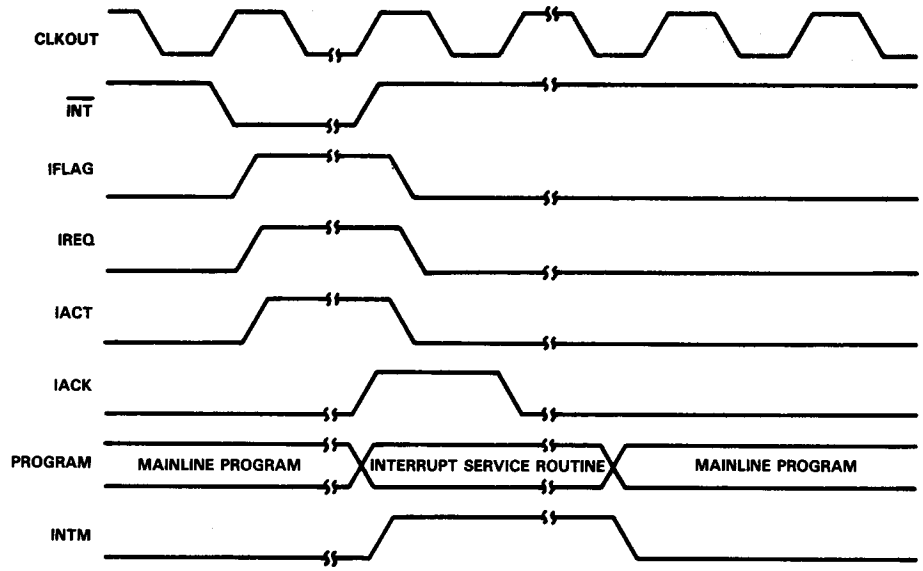


Figure 4. Pulse-Triggered Interrupt

In the second case, $\overline{INT}$ goes low and stays low for an indefinite period (see Figure 5). Here IFLAG is set to one, and IREQ and IACT go high as they did in the case of a pulsed interrupt. However, when the interrupt service routine is entered and the IACK pulse is generated, IFLAG is cleared, but IREQ remains high since $\overline{INT}$ is still low. The result of this is that, when interrupts are reenabled at the end of the interrupt service routine, the interrupt will be accepted again, and the interrupt routine will be reentered without returning to the mainline program. This will continue until the interrupt is removed.

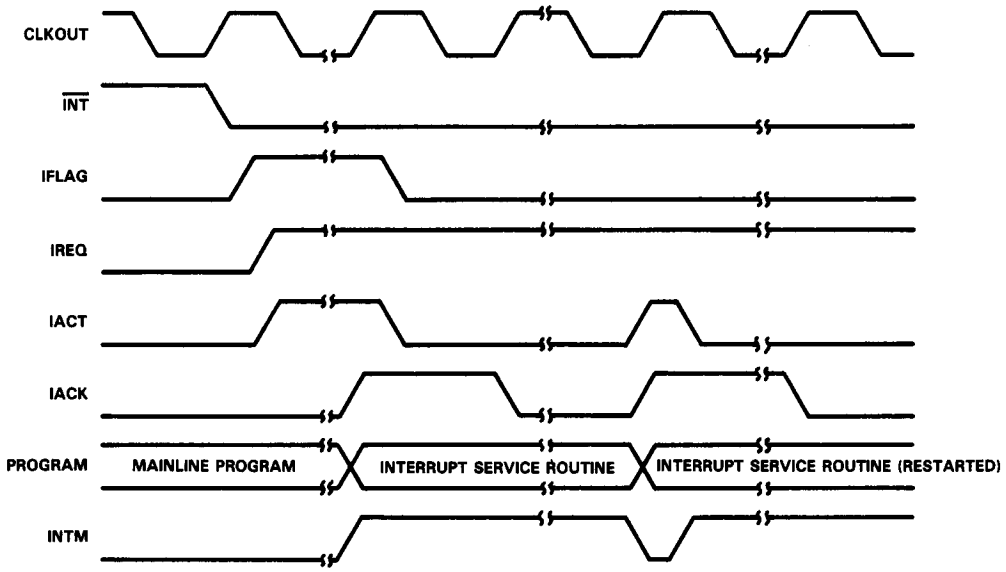


Figure 5. Level-Triggered Interrupt

The third case is a situation where, after a pulse has occurred on $\overline{INT}$ and the normal interrupt processing has begun, another interrupt occurs before servicing of the first interrupt is complete (see Figure 6). It is evident from the timing diagram that, once the first interrupt is cleared by the IACK pulse, another $\overline{INT}$ pulse may cause a second interrupt to become pending. This interrupt will be serviced immediately following the return from the first interrupt service routine (the mainline program will not be reentered until the second interrupt is serviced).

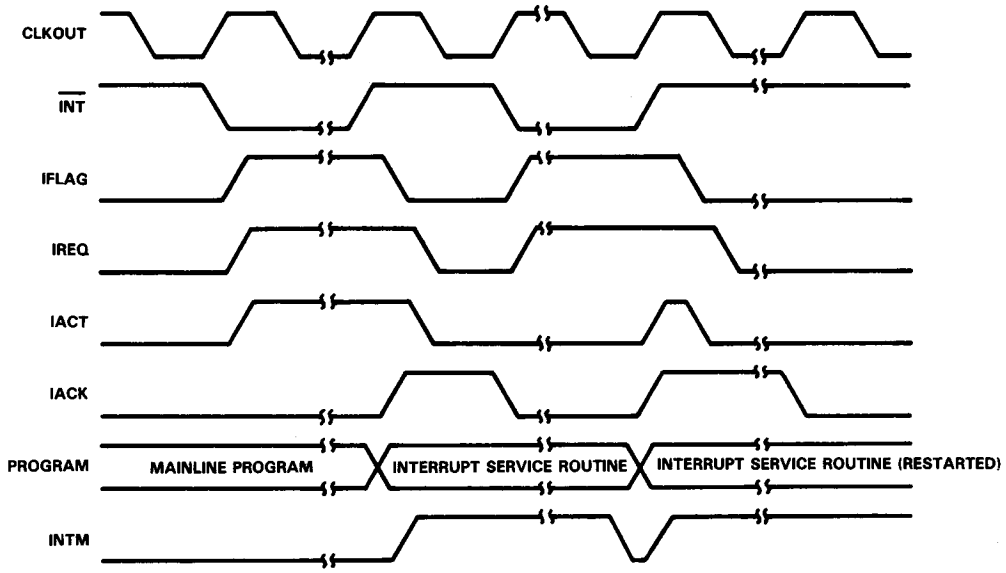


Figure 6. Multiple-Pulse Interrupts

The following general rule describes all of these situations: An interrupt may be generated either by the falling edge of a low-going pulse or by a low level on the $\overline{INT}$ input. It should be noted that this is distinctly different from many other types of systems in which an actual edge is required to produce an interrupt, and a level alone has no effect. Level triggering is provided on the TMS32010 in order to allow multiple external devices to interrupt the processor and be handled in a polled fashion.

Interrupt Programming

From a software standpoint, operation of interrupts on the TMS32010 is quite straightforward and very similar to interrupts on most other microprocessors, with only two exceptions.

Interrupt operation in general can be characterized by the following rules:

1. If an interrupt occurs while interrupts are enabled, it is accepted following completion of the instruction currently executing, even if the current instruction is a multicycle instruction.
2. If an interrupt occurs while interrupts are disabled, the interrupt remains pending and will be accepted as soon as interrupts are enabled, unless cleared in the meantime.
3. Pending interrupts are cleared by $\overline{RS}$ or by entry into the interrupt service routine unless $\overline{INT}$ has remained low.

Table I shows a summary of interrupt operation depending upon the circumstances under which the interrupt occurred. It should be emphasized that, although it may not be immediately obvious, most of these cases are a direct consequence of standard interrupt processing. The two exceptions to the general rules are those marked with an asterisk.

As an example of standard interrupt processing, consider the situation in which interrupts are enabled, and an interrupt occurs during the execution of a DINT (disable interrupts) instruction. Even though interrupts are enabled, the general rules state that an interrupt will not be accepted until the execution of the current instruction is completed. In this case, since the completion of this instruction results in interrupts being disabled, the interrupt will not be accepted.

Also, it should be noted that, even though some instructions take more than one cycle to execute, interrupt acceptance will always be delayed until the instruction has fully completed execution.

The first of the two exceptions to the general rules is the situation in which interrupts are enabled and an interrupt occurs during an MPY (or an MPYK) instruction. In this case, acceptance of the interrupt is delayed one more instruction cycle, until after the instruction following the MPY(K) has been executed. The reason for this is that the contents of the P Register (the results of the multiply) are very difficult to recreate. Delaying acceptance of interrupts an additional cycle guarantees that the program may safely store the results of the operation before trapping into the interrupt service routine, which might alter the contents of the P Register. (Thus, the user would normally code an instruction which stores the P Register into the accumulator immediately following a multiply instruction if interrupts are present in the system). However, if the interrupt service routine does not require the use of the P Register, it is not necessary that its contents be stored.

The second exception to the general rules occurs in the situation where interrupts are disabled (such as in an interrupt service routine), and an interrupt occurs during the execution of an EINT (enable interrupts) instruction. Here also, acceptance of the interrupt is delayed one additional instruction cycle, until after the instruction following the EINT has been executed. The reason for this is that, if the RET

Table 1. Interrupt Operation in Software

If Interrupt Occurs:	And Interrupts Are:	
	Enabled	Disabled
During DINT	The interrupt is ignored (but remains pending until cleared or an EINT is executed)	The interrupt is ignored (but remains pending until cleared or an EINT is executed)
During EINT	The interrupt is accepted after the EINT is executed	*The instruction following the EINT is completed and then the interrupt is accepted (unless the instruction is a DINT)
During MPY or MPYK	*The instruction following the MPY(K) is completed and then the interrupt is accepted (unless the instruction is a DINT)	The interrupt is ignored (but remains pending until cleared or an EINT is executed)
During any other instruction	The interrupt is accepted after the instruction executes (even if the instruction is multicycle)	The interrupt is ignored (but remains pending until cleared or an EINT is executed)

*Exception to standard interrupt servicing

instruction normally following an EINT in an interrupt service routine is not allowed to execute before another interrupt is accepted, a stack overflow can eventually result, destroying return linkage to the mainline program.

In these two exceptions to the general rules, note that if the instruction following an EINT (or MPY or MPYK) instruction is a DINT instruction, the interrupt will not be accepted.

The following sequence of instructions illustrates a combination of several of these concepts:

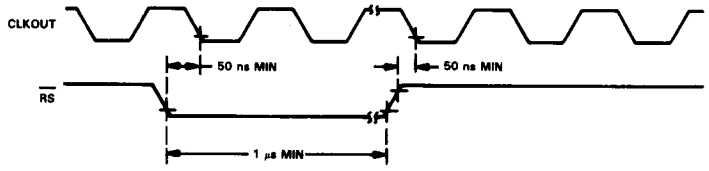
```
DINT
EINT
MPY     VAL1
MPY     VAL2
MPYK    CONST
DINT
```

An interrupt occurring at any point during this sequence of instructions would never be accepted (at least during this sequence). It would, however, remain pending until cleared or accepted in a later section of code following an EINT. The reasons for this are as follows: if the interrupt occurred during the DINT, it would wait until the instruction had executed before being considered for acceptance. Since the completion of this instruction results in interrupts being disabled, it would not be accepted at this point. If the interrupt occurred during the EINT or was still pending at this time, interrupts would still be disabled, and the interrupt would be ignored. After the EINT had executed (because of the two exceptions to the general rules), all of the multiplies would be executed. Then finally, after the MPYK, the DINT would also be executed (since it follows a multiply instruction) and would disable interrupts again.

$\overline{\text{RS}}$ INPUT

$\overline{\text{RS}}$ serves as the master reset for the TMS32010. Asynchronous inputs may be used on $\overline{\text{RS}}$ since this input is provided with an internal synchronizer circuit. However, if it is necessary to maintain synchronization of program execution to the reset input, the 50-ns setup time between $\overline{\text{RS}}$ and CLKOUT must be observed. This may be required, for example, when multiple TMS32010s are used concurrently in a system. Figure 7 shows the timing of the $\overline{\text{RS}}$ input.

It should be noted that reset should only be used to perform system initialization-type functions (such as would be used after powerup). Reset cannot be used to perform hold or interrupt functions, since it does not preserve any linkage to the previous program environment. Also, the state of the internal logic (including data RAM, status and general-purpose registers, accumulator contents, etc.) is not predictable after reset; once $\overline{\text{RS}}$ is driven low and then high, the results are simply that interrupts are disabled, the interrupt flag is cleared, and program execution is forced to location zero.



NOTE: Timing measurements are referenced to and from a low voltage of 0.8 V and a high voltage of 2.0 V.

Figure 7. $\overline{\text{RS}}$ Input Timing

SUMMARY

Adherence to the basic considerations described in this application report regarding these three inputs provides efficient and reliable system operation. Briefly restated, these considerations are as follows:

1. The interrupt and $\overline{\text{BIO}}$ inputs must be synchronized.
2. Interrupts are either level or edge triggered, and function according to standard interrupt processing, except for the special cases of the EINT, MPY, and MPYK instructions.
3. On the $\overline{\text{RS}}$ input, no synchronization is required, except in cases where precise timing immediately following reset is necessary.

REFERENCE

1. Stoll, Peter A., "How To Avoid Synchronization Problems," VLSI DESIGN, November/December 1982, pp. 56-59.