Application Report
SPRAAJ3–July 2008

Configuring External Interrupts on TMS320C672x Devices

Giancarlo Parodi

ABSTRACT

Every real-time embedded system needs to be able to interact to events associated with the external world in a timely manner; for this, typically every digital signal processor (DSP) device features one or more dedicated pins, which are used for carrying these external signals, called interrupts, inside the device for further processing by the application. The TMS320C672x generation of processors has this functionality, although it does not feature any dedicated interrupt lines. This application report details how to configure the application, using the C672x chip support library, in order to setup the device to process external interrupts by providing two simple example projects: one of them being based on the DSP/BIOS™ software kernel foundation. The sample code described in this application report can be downloaded from http://www.ti.com/lit/zip/SPRAAJ3.

Contents

List of Figures

1 Introduction

The C672x CPU does not feature any dedicated interrupt lines associated directly with external pins.

This implies that, compared with other architectures belonging to the C6000™ platform of processors, there are no device pins which are reserved and used exclusively for this purpose.

However, the direct memory access (DMA) engine (named dMax) has the possibility of generating internal interrupts to the central processing unit (CPU), and for this a special dMax type of event has been reserved in the dma architecture.

To enable this, put a specific setup in place on the dMax side so that the status of specific external pins is monitored and latched by the dMax; in response to changes on this status, the dMax can propagate the event by issuing an interrupt to the CPU and enable the application code to react to the external stimulus.

The C672x external interrupts are routed through dMAX after an initial mux selector.

2 Prerequisites

This application report is based upon usage of the C672x chip support library. The *TMS320C672x Chip Support Libraries (CSL)* (SPRC223) can be downloaded from the TI website (http://www.ti.com/).

Use a custom or evaluation module based on this processor since the C672x simulator provided with Code Composer Studio™ software does not support all the device peripherals, but rather the CPU core implying that the code referenced in this application report cannot be used on a simulator.

Use Code Composer Studio 3.1, including installation of the C672x chip support package, to rebuild the code and test it on the target board. The C672x chip support package can be downloaded from the updated advisor tool included in Code Composer Studio. Note that in Code Composer Studio 3.3 or later, the C672x devices are natively supported and do not require installing any specific chip support package.

This example code was tested on the low cost evaluation module (EVM) manufactured by DSP Weuffen (http://www.dsp-weuffen.de/). Any board-specific settings (commented in the code) might have to be modified depending on the custom board configuration.

3 Configuration Steps Needed

There are several device-specific aspects that need to be evaluated in order to properly configure the system:

1. Decide which pin should be used as external interrupt source. The C672x devices have the possibility of configuring, via a multiplexer, one of the serial pins from multichannel audio serial port (McASP), serial peripheral interface (SPI), and IIC to be used as a source input. In this case, the mute input signal of the McASP is selected as the multiplexed input and the CFGMCASPx register details which external pin is used.
   Below is a table taken from the device datasheet, which details how to configure the CFGMCASPx register to select the external pin to be used.

   AMUTEIN0 Selects the source of the input to the McASP0 mute input.

   000 = Select the input to be a constant '0'

   001 = Select the input from AXR0[7]/SPI1_CLK

   010 = Select the input from AXR0[8]/AXR1[5]/SPI1_SOMI

   011 = Select the input from AXR0[9]/AXR1[4]/SPI1_SIMO

   100 = Select the input from AHCLKR2

   101 = Select the input from SPI0_SIMO

   110 = Select the input from SPI0_SCS/I2C1_SCL

   111 = Select the input from SPI0_ENA/I2C1_SDA

   AMUTEIN1 Selects the source of the input to the McASP0 mute input.

   000 = Select the input to be a constant '0'

   001 = Select the input from AXR0[7]/SPI1_CLK

   010 = Select the input from AXR0[8]/AXR1[5]/SPI1_SOMI

   011 = Select the input from AXR0[9]/AXR1[4]/SPI1_SIMO

   100 = Select the input from AHCLKR2

   101 = Select the input from SPI0_SIMO

   110 = Select the input from SPI0_SCS/I2C1_SCL

   111 = Select the input from SPI0_ENA/I2C1_SDA

   AMUTEIN2 Selects the source of the input to the McASP2 mute input.

   000 = Select the input to be a constant '0'

   001 = Select the input from AXR0[7]/SPI1_CLK

   010 = Select the input from AXR0[8]/AXR1[5]/SPI1_SOMI

   011 = Select the input from AXR0[9]/AXR1[4]/SPI1_SIMO

   100 = Select the input from AHCLKR2

   101 = Select the input from SPI0_SIMO

   110 = Select the input from SPI0_SCS/I2C1_SCL

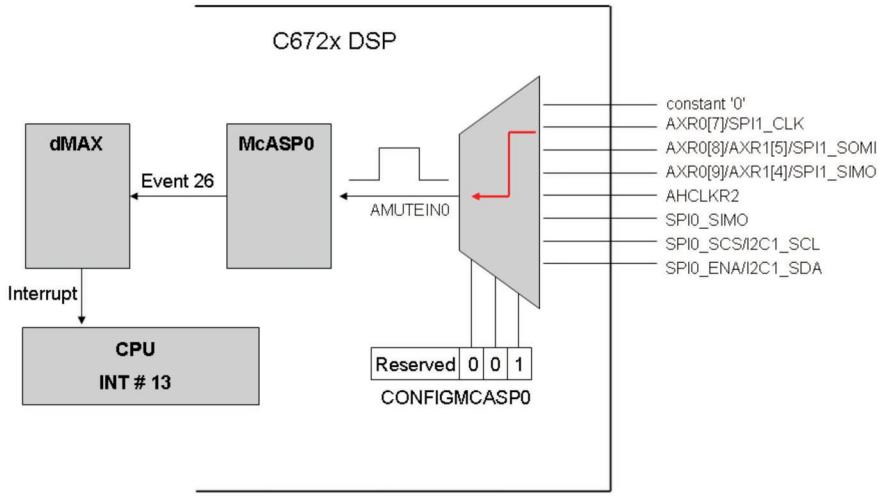
   111 = Select the input from SPI0_ENA/I2C1_SDA

Note: McASP2 is not available on C6720 and C6722.

Although the specific pin cannot be used on its McASP functionality anymore, this does not imply that the McASP peripheral needs to be kept in reset; it can still function as a McASP, provided that:

- The specific pin of choice is configured as general-purpose input/output (GPIO) in the McASP configuration
- This pin is configured as input

2. Disable the AMUTEIN signal within the error logic of the corresponding McASP for an external system to react to any peripheral errors related to the McASP functionality, if the AMUTE pin is used on the McASP at the system level. Do this to avoid AMUTE being driven when AMUTEIN, used as an external interrupt source, is active. This can be achieved by clearing the INEN bit of the AMUTE register inside the specific McASP.
   An additional caution is needed for pins which are also multiplexed with other peripherals like serial peripheral interface (SPI) or inter-integrated circuit (I2C).
   If pin driving conflicts arise again, the SPI or I2C devices have to be kept in reset. Given the low pin count and the purpose of each serial line, those ports will not have any practical usage without the specific pin functionality being available for the communication.
3. Configure the dMax engine to generate an interrupt to the processor in response to the external signal status changes.
   To do this, a specific dedicated event is being reserved in the event types and associated to the interrupt generation. These event are numbered 26, 27, and 28, respectively, for McASP0, McASP1, and McASP2.
   Note that the detection of the interrupt will be edge-triggered, but not registered (rising or falling). The external device that is generating it needs to toggle the signal properly for the next edge to be recognized. Not being registered implies that the event will not be held (flagged) in any register bit. Additionally, the pulse needs to be at least two SYSCLK2 cycles wide, where SYSCLK2 is the internal clock being used in the device for the dMax peripheral.
4. Determine which event on the McASP side of the interrupt service routine is triggering the interrupt if the McASP peripheral is being used and configured to generate CPU interrupts in case of peripheral errors (on the CPU side). This is done to make sure that the status is properly handled (eventual other interrupt flags are not being cleared or ignored inadvertently) and the application can react as desired. AMUTEIN events in the McASP architecture are logically ORed with the McASP transmit and receive error events within the dMax.



Interrupt chain setup

(default example configuration)

Figure 1. Interrupt Chain Setup

4 DSP/BIOS

DSP/BIOS is a royalty free, TI-developed real-time operating system that supports, among others, the complete platform of C6000™ devices including the new C672x generation. For additional details about the different modules available and a description of the features that this operating system (OS) can provide, (for example, interrupt processing, task management, data analysis and logging features, etc.), see the *TMS320 DSP/BIOS User's Guide* (SPRU423) and the *TMS320C6000 DSP/BIOS 5.32 Application Programming Interface (API) Reference Guide* (SPRU403).

Note: If you decide to use the DSP/BIOS libraries versions stored in the ROM, the BIOS version being tested in this example is 5.20.05 since it is the only version supported in the ROM code of the C672x. This is outside the scope of this application report. Fore more details on how to use it, see *Using ROM Contents on TMS320C672x* (SPRAAS8).

5 Software Description

The two software examples accompanying this application report demonstrate the same system level of functionality on two different applications: one is based on DSP/BIOS and the other exclusively uses the Chip Support Library, including the INTC module for the interrupt setup configuration.

In the system, a periodic timer event is used to generate short pulses on one of the device pins configured as GPIO. In the example code, since the McASP functionality is not used, one of the available pins of the McASP device is chosen for this function. This pin is externally physically wired, as an external loopback, to the pin chosen as the source of the AMUTEIN signal.

In the DSP/BIOS example, this pulse on the external pin is generated via a periodic function configured in the BIOS operating system setup.

In the dMax interrupt service routine, the AMUTEIN event is being recognized and in response a status flag in the application is changed, or a semaphore is posted in the BIOS based example. This allows a polling routine or a blocked task to wake up and log on the terminal output/BIOS log a counter of the number of interrupts being processed so far.

Example 1. dMax Isr example

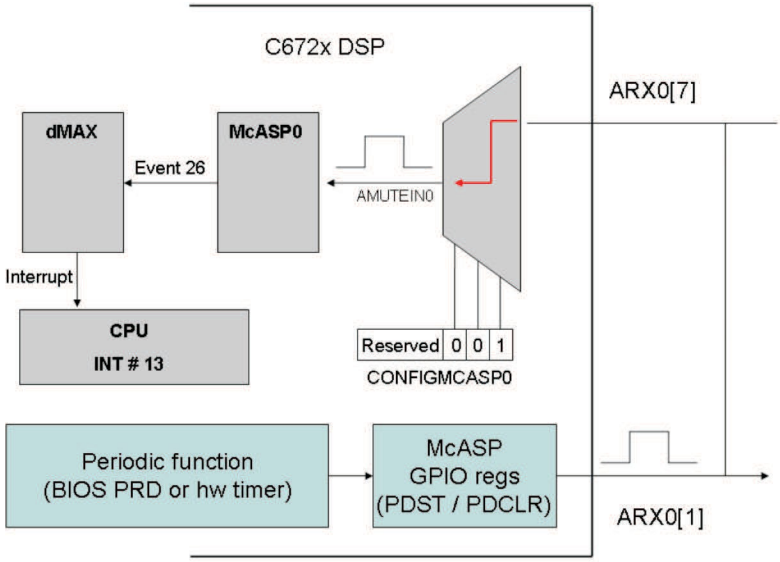
```
 void myIsr()
{
 numInterrupts++;

 // check the source of the dMax interrupt
 // you need to verify in the McASP register which flags are being raised

 // event 26 MCASP0ERR is due to either AMUTEIN0 - McASP0 TX INT - McASP0 RX
 // check RSTAX / XSTAT depending on RINTCTL / XINTCTL as needed

 // in this case the check is not done due to the simple example
 // McASP0 error ints are not enabled

 // release the cpu from waiting for the interrupt
 compInt0Event = TRUE;
}
```



Evm hw setup with external wiring

(default example configuration)

Figure 2. External Wiring Setup

6 Summary

This application report shows how to put together the information provided in the device data sheet, the dMax user's guide and the McASP user's guide to configure a C672x-based hardware and be able to recognize and process external interrupts.

As a final note on the example projects, it is not recommended to use printf() on the final application. The main reason being that the related string formatting and processing is done on the target (DSP) side before the data gets sent to the host for displaying the output in Code Composer Studio.

For a more lightweight and less computation intensive way to achieve the same result, it is advisable to use LOG_printf, when possible, if DSP BIOS is used. In this case, the required string processing is done on the host, therefore, the process consumes only a minimal set of DSP resources.

7 References

- *TMS320C6727, TMS320C6726, TMS320C6722 Floating-Point Digital Signal Processors Data Manual* (SPRS268)
- *C9230C100 TMS320C672x Floating-Point Digital Signal Processor ROM Data Manual* (SPRS277)
- *TMS320C672x DSP Dual Data Movement Accelerator (dMAX) Reference Guide* (SPRU795)
- *TMS320C672x DSP Multichannel Audio Serial Port (McASP) Reference Guide* (SPRU878)
- *TMS320C672x DSP Real-Time Interrupt Reference Guide* (SPRU717)
- *TMS320 DSP/BIOS User's Guide* (SPRU423)
- *TMS320C6000 DSP/BIOS 5.32 Application Programming Interface (API) Reference Guide* (SPRU403)
- *TMS320C672x Chip Support Libraries (CSL)* (SPRC223)

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, or other requirements. These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale (www.ti.com/legal/termsofsale.html) or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265