Application Note, V 1.0, Aug. 2004

AP16083

16 Bit CMOS Microcontroller Product

Interrupt Response Time of the XC16x Family

Microcontrollers

Never stop thinking.

Revision History:	2004-08	**V 1.0**
Previous Version:	-	

Page	Subjects (major changes since last revision)

Controller Area Network (CAN): License of Robert Bosch GmbH

We Listen to Your Comments

Any information within this document that you feel is wrong, unclear or missing at all? Your feedback will help us to continuously improve the quality of this document. Please send your proposal (including a reference to this document) to:
mcdocu.comments@infineon.com

Edition 2004-08

Published by
Infineon Technologies AG
81726 München, Germany

LEGAL DISCLAIMER

THE INFORMATION GIVEN IN THIS APPLICATION NOTE IS GIVEN AS A HINT FOR THE IMPLEMENTATION OF THE INFINEON TECHNOLOGIES COMPONENT ONLY AND SHALL NOT BE REGARDED AS ANY DESCRIPTION OR WARRANTY OF A CERTAIN FUNCTIONALITY, CONDITION OR QUALITY OF THE INFINEON TECHNOLOGIES COMPONENT. THE RECIPIENT OF THIS APPLICATION NOTE MUST VERIFY ANY FUNCTION DESCRIBED HEREIN IN THE REAL APPLICATION. INFINEON TECHNOLOGIES HEREBY DISCLAIMS ANY AND ALL WARRANTIES AND LIABILITIES OF ANY KIND (INCLUDING WITHOUT LIMITATION WARRANTIES OF NON-INFRINGEMENT OF INTELLECTUAL PROPERTY RIGHTS OF ANY THIRD PARTY) WITH RESPECT TO ANY AND ALL INFORMATION GIVEN IN THIS APPLICATION NOTE.

Information

For further information on technology, delivery terms and conditions and prices please contact your nearest Infineon Technologies Office (**www.infineon.com**).

Warnings

Due to technical requirements components may contain dangerous substances. For information on the types in question please contact your nearest Infineon Technologies Office.

Infineon Technologies Components may only be used in life-support devices or systems with the express written approval of Infineon Technologies, if a failure of such components can reasonably be expected to cause the failure of that life-support device or system, or to affect the safety or effectiveness of that device or system. Life support devices or systems are intended to be implanted in the human body, or to support and/or maintain and sustain and/or protect human life. If they fail, it is reasonable to assume that the health of the user or other persons may be endangered.

Table of Contents **Page**

1 Introduction

The architecture of the XC16x supports several mechanisms for fast and flexible response to service requests from various sources internal or external to the microcontroller. Different kinds are handled in a similar way:

- Interrupts generated by the Interrupt Controller (ITC)
- DMA transfer issued by the Peripheral Event Controller (PEC)
- Traps caused by the Trap instruction or issued by faults or specific system states

The XC16x family fits perfectly in embedded applications. The target of this application note is to supply detailed information about the real time capabilities of the interrupt architecture.

For more detailed information about the functionallity of the interrupt architecture please refer to the corresponding user manual.

1.1 General Conditions

The following calculations are only valid for the conditions below:

- All memory accesses are done without delay
  - Code and Interrupt Vector Table located in a fast internal Memory
  - (PRAM / Flash), accessed within 1 CPU clock cycle
  - The stack is located in DPRAM
- The Peripheral Bus clock speed is the same as CPU clock speed
- No previous interrupt request is still processed
- No stall or cancellation condition of the pipeline is valid

2 Interrupt response time

2.1 Definition of Interrupt Response time

In this document the Interrupt Response Time is defined as the time between an active request signal being generated and the first instruction of the associated interrupt process entering the pipline of the CPU.

2.2 Interrupt Flow

The interrupt flow is divided in the following sections.

- Peripheral / Fast External Interrupt
- Interrupt Controller / Arbitration
- CPU Core

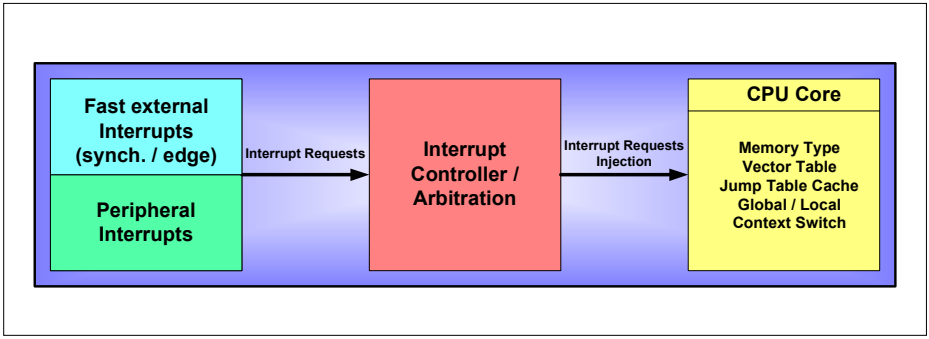


Figure 1 Interrupt Flow

2.2.1 Peripheral / External Interrupt

Interrupt requests may be triggered either by the on-chip peripherals or by external inputs. From the point when an interrupt occurs, until the interrupt request flag is set takes 2 or 3 peripheral clock cycles.

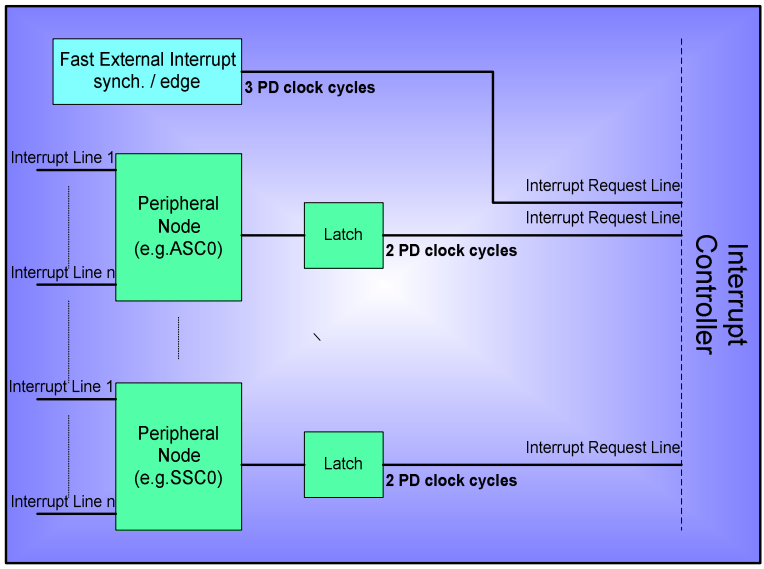


Figure 2 Peripherals / Fast external Interrupts

2.2.2 Interrupt Controller

The interrupt controller is arbitrating all pending interrupts according to a programmable prioritization schema.

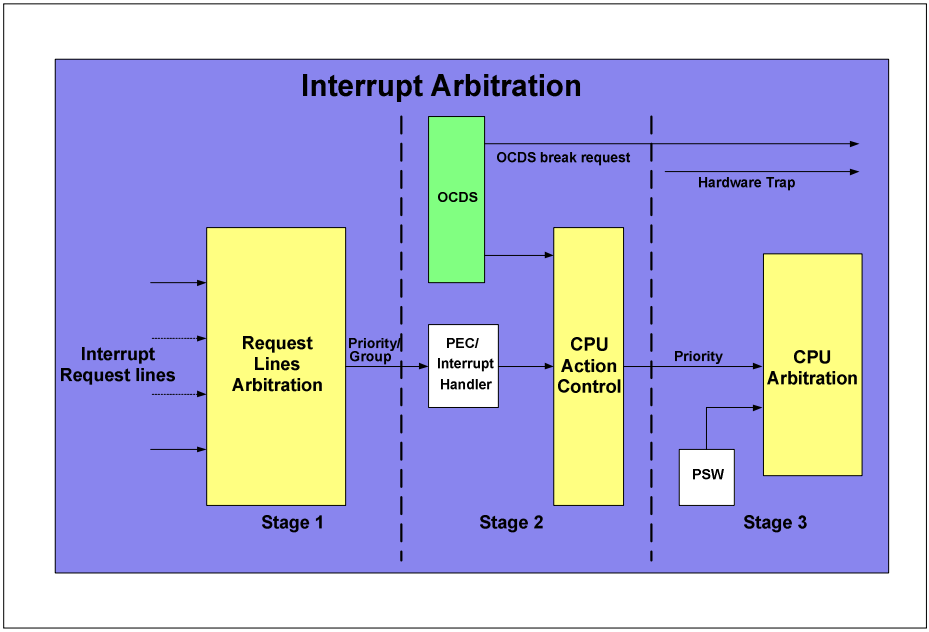


Figure 3 Interrupt Arbitration

The interrupt arbitration is done in three stages:

The first arbitration stage – all active requests are compared against their priorities from the respective xxIC registers.

The second arbitration stage – the first stage winner is arbitrated against the OCDS service requests; an interrupt-injection is requested to the CPU.

The third arbitration stage – the upcoming request is examined inside the CPU against the current value from PSW – priority level of the present task and global interrupt enable flag.

2.2.3 CPU / PEC

Whenever a request is accepted the ITRAP instruction is injected. The preparation and the execution take 5 CPU clock cycles. Servicing an interrupt request via the vector table requires two subsequent branches. The first one includes the vector location; the second one includes the address to the actual service routine.

The interrupt service time can be reduced by 4 cycles using the Jump Table Cache feature.

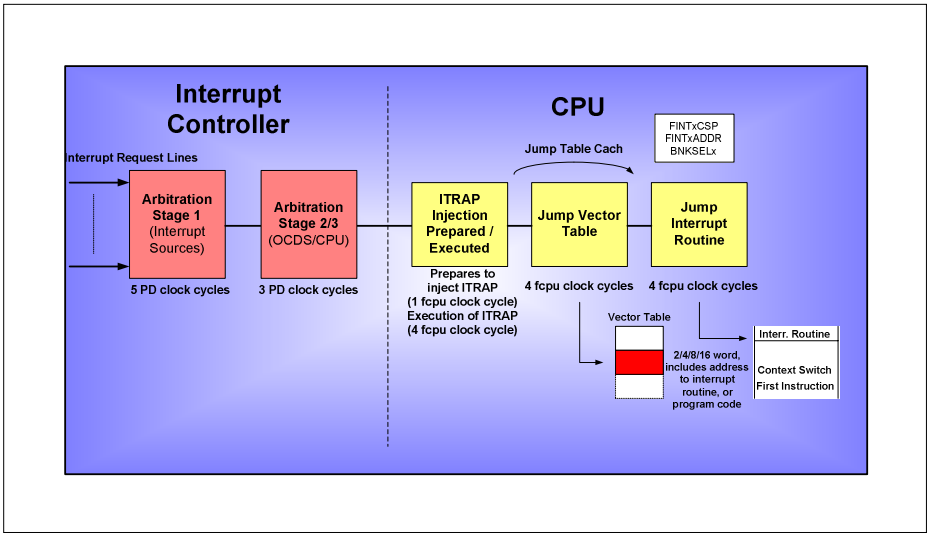


Figure 4 Interrupt processing

Before the first instruction of an interrupt service routine is executed, a context switch is mandatory. There are two ways to switch the context in the XC16x core.

- Switching between Global Register Banks

One single dedicated instruction (SCXT) is used to change the Context Pointer Register, to save the old and to load the new GPR-content to/from Dual Ported Memory. The execution of the SCXT instruction takes 19 clock cycles.

- Switching to a local Register Bank

For interrupt priority levels 15..12 the two local register banks can be pre-selected and can then be switched automatically. In this case, no SCXT instruction is executed.

In the case when the selection of a local register bank is done at the starting point of the interrupt service routine, a cancellation of the complete pipeline is caused and an additional minimum delay of 6 clock cycles is added.

In the case of a PEC transfer being processed, a part of all the interrupt related process is required. PEC transfers are generally a faster method of interrupt services. For PEC transfers, the arbitration process works in the same manner. After the request is accepted by the CPU, a special instruction is injected and it passes through the pipeline until the execute stage is reached. Figure 5 shows the flow in detail.

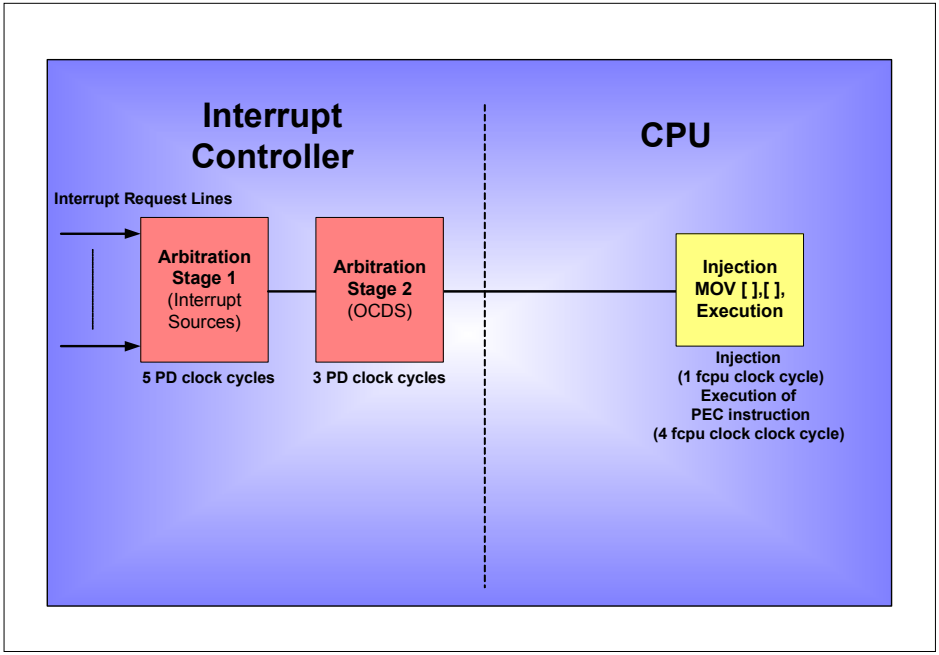


Figure 5 PEC Transfer

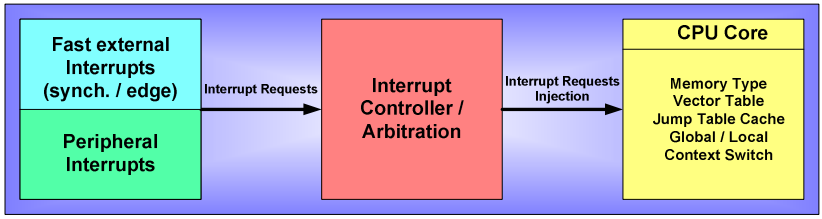
An additional delay can be caused if:

- Interrupt Controller is busy
- Pipline stalled
- Pipline cancelled

2.3 Summary

Figure 6 summarizes the interrupt response time for interrupts in general and for PEC transfers. The numbers of cycles (inside the brackets) that have been added are dependent of the following reasons.

- Interrupt Controller busy
- Pipeline stalled
- Pipeline cancelled
- Latency of the memory
- Context switching



Interrupt	2 (3) PD cycle	5 + 3 PD cycle	1 + 4 + 4 + (4) + (...) fcpu cycle
PEC	2 (3) PD cycle	5 + 3 PD cycle	1 + 4 +(...) fcpu cycle

Figure 6 Interrupt Response Time

Please refer to the XC16x user manual for detailed information about additional delays.

The following example visualise the best case interrupt response time:

The interrupt service routine (ISR) is executed from the internal program memory (PM). The jump table cache feature is used. The local register bank is initialized before the interrupt is enabled. If the general condition as described in chapter 1.1 are valid the interrupt response time take 21 CPU cycles (525ns @ 40 MHz).

The following conditions describe some conditions for add on to the existing interrupt response time:

The interrupt response time is extended by 19 CPU cycles if a global register bank is used.

The interrupt response time can be extended by up to 9 CPU cycles if another interrupt request is started arbitration.

The interrupt response time is extended by 7 CPU cycles if the register bank is changed directly in the PSW register.

The interrupt response time is extended by 5 CPU cycles if a special function register is updated.

Note: These numbers of cycles are only valid for optimized bus timings and should be verified in either case.

3 Configuration of the Interrupt Handler

3.1 Potentials to influence the interrupt response time

The XC16x architecture offers a couple of dedicated registers to configure the interrupt handler. The configuration can be divided in two groups.

- Interrupt jump table cache
- Fast bank switching

3.1.1 Jump Table Cache (Fast Interrupt)

The interrupt servicing time can be reduced by the Interrupt Jump Table Cache. This feature eliminates the explicit branch to the ISR by directly providing the CPU with the service routine location.

The two pointers are each stored in a pair of interrupt jump table cache registers, which store an 8 bit pointer segment and a 16 bit offset along with the priority level (priority level 12-15). These features can be selected for two interrupt sources.

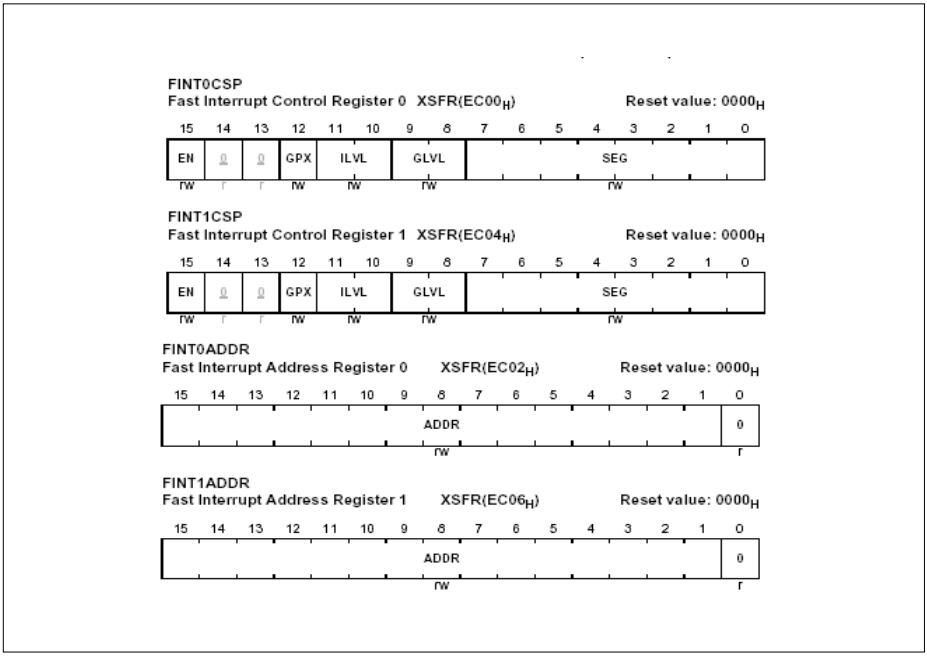


Figure 7 Jump Table Cache Registers

3.1.2 Fast Bank Switching

The XC16x architecture allows switching the selected physical register bank. By updating the bitfield BANK in register PSW the active register bank is switched. In case of an interrupt service, the bank switch is automatically executed by updating bitfield BANK from BNKSELx in the interrupt controller. For interrupt priority levels 12…15 the target register bank can be pre-selected. The registers BNKSELx provides a 2-bit field for each possible arbitration priority level. The respective bitfield is then copied to bitfield BANK in register PSW to select the register bank, as soon as the respective interrupt request is accepted. After a switch to a local register bank, the new bank is immediately available.

BNKSELx
Register Bank Select Reg. x **XSFR (See Table 5-10)** **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
GPRSEL7		GPRSEL6		GPRSEL5		GPRSEL4		GPRSEL3		GPRSEL2		GPRSEL1		GPRSEL0	
rw		rw		rw		rw		rw		rw		rw		rw	

Field	Bits	Type	Description
GPRSELy (y = 7 … 0)	[2y+1 :2y]	rw	**Register Bank Selection** 00 Global register bank 01 Reserved 10 Local register bank 1 11 Local register bank 2

Figure 8 Registers BNKSEL0-3 for interrupt priority level 12-15

3.2 C-Compiler and configuration of the interrupt handler

Both Keil and Tasking Compilers support the enhanced interrupt handling of the XC16x architecture. DAvE (Digital Application Engineer) does not currently support the enhanced interrupt handling. The following hints may be useful to illustrate how the user may to influence the interrupt response time.

3.2.1 Keil

The Keil C-Compiler supports the XC16x architecture and the enhanced interrupt handling. For more detailed information please refer to the Keil C-Compiler manual or C166: USING XC16X FAST REGISTER BANK SWITCHING.

There are different register bank switching methods available. They can be controlled with different specifiers for 'rbank_id':

Omitting using: The compiler generates code to save (PUSH) and restore (POP) all registers that are used in this function to the system stack. Saving and restoring the register values takes time. However, if you have a very small interrupt function where only a few registers are used, this might be the most effective method.

Any Name: The compiler generates code to save (PUSH) the current context pointer register (CP) and loads it with the address of a dedicated register bank. At the end of the ISR the CP register is restored. The registers R0 to R15 don't need to be saved on the system stack in this case. Specifiying a register bank speeds up the execution of an ISR.

_FAST_BANK1_ or _FAST_BANK2_: The compiler generates code to switch to a fast register bank by modifying the BANK field of the program status word (PSW). The registers R0 to R15 don't need to be saved on the system stack in this case.

_FAST_ABANK1_ or _FAST_ABANK2_: The compiler does not generate code to switch to a different register bank or to save the current registers (R0 - R15). The interrupt controller (BNKSELx register) must be initialized to switch to a fast register bank automatically on entering the ISR by the user application.

The following C-Examples illustrate the configuration and the interrupt handling.

General Interrupt using global register bank

```
void CC1_vInit(void)
{ .......
  PSW_IEN    =  1;                    // set global interrupt enable
}
void CC2_viTmr7(void) interrupt CC2_T7INT
{.......
}//  End of function CC2_viTmr7
```

Interrupt Jump Table Cache

An interrupt service routine (ISR) can be defined with "<Name>=CACHED" instead of an interrupt vector number. In this case, no interrupt vector is generated. The application needs to program the interrupt controller registers FINTxCSP and FINTxADDR with the address of the ISR before the interrupt is enabled. The following example shows how to do it:

```
void CC2_viTmr7(void) interrupt CC2_T7INT =CACHED
{ .......
}//  End of function CC2_viTmr7
#define SEG(func) (unsigned int)(((unsigned long)((void (far
*)(void))func) >> 16))
#define SOF(func) (unsigned int)(((void (far *) (void))func))
void CC2_vInit(void)
{.......
  // Initialize fast interrupt register: EN=1, ILVL=15, GLVL=0, GPX=0
  FINT0CSP   = SEG(CC2_Tmr7) | 0x8C00;
  FINT0ADDR  = SOF(CC2_Tmr7);
  PSW_IEN    =  1;                    // set global interrupt enable
}
```

Fast Bank switching:

When using **_FAST_BANKx_** or **_FAST_ABANKx_** a separate user stack needs to be defined for these ISR's. This is done with **UST1SZ** and **UST2SZ** in the START_V2.A66 file. Be sure to define a range that is big enough to hold the local variables of the ISR.

```
UST1SZ    EQU    0x20      ; set User Stack Size to 20H Bytes
UST2SZ    EQU    0x20      ; set User Stack Size to 20H Bytes
```

When **_FAST_ABANKx_** is used, the interrupt controller register BNKSELx needs to be initialized before the interrupt is enabled. The following example shows how to do it:

```
void CC2__viCC2(void) interrupt CC2_T7INT using _FAST_ABANK2_
{......
}

void CC2_vInit(void)
{......
  BNKSEL1=0x0300;         // BNKSEL1.GPRSEL4=11 -> Local register bank 2
  PSW_IEN    =  1;        // set global interrupt enable
}
```

Fast Bank switching + Jump Table Cache:

```
void CC1_viCC1(void) interrupt CC2_T7INT =CACHED using _FAST_ABANK2_
{......
}
void CC1_vInit(void)
{......
  BNKSEL1=0x0300;         // BNKSEL1.GPRSEL4=11 -> Local register bank 2
  PSW_IEN    =  1;        // set global interrupt enable
}
```

3.2.2 Tasking

The Tasking C-Compiler supports the XC16x architecture and the enhanced interrupt handling. Included are some useful extensions to force fast register bank switching, cached interrupts, etc.. For more detailed information please refer to the Tasking C-Compiler manual.

_stacksize (num)	// specifies the userstack adjustment in byte
_localbank (num)	// local register bank switching (0,1,2,-1,-2) 0 = Global register bank -1 / -2 = local register bank1/2, BNKSEL0 should be used 1 / 2 = local register bank1/2, PSW is set in the ISR (not recommended !)
_cached	// bypasses the interrupt vector table
#pragma noframe	// omit the whole interrupt frame, allows you to make your own interrupt frame. Should be used carefully !

General Interrupt using global register bank:

The Timer T7 overflow interrupt is enabled. If a timer T7 overflow occurs, the service routine switches the context of the global register bank to get a new set of GPRs..

```
interrupt (CC2_T7INT) void CC2_viTmr7(void)
{ ......
}
```

Interrupt Jump Table Cache:

The Timer T7 overflow interrupt is enabled. If a timer T7 overflow occurs the service routine switches the context of the global register bank to get a new set of GPRs. Instead of using the vector table the CPU directly takes the addresses of the service routine.

```
interrupt (CC2_T7INT) _cached void CC2_viTmr7(void)
{......
}
void CC2_vInit(void)
{......
  FINT0CSP = 0x8000 | (((unsigned long)&(CC2_viTmr7))>>16);
  FINT0ADDR = (unsigned int)&(CC2_viTmr7);
  PSW_IEN    =  1;            // set global interrupt enable
}
```

Fast Bank switching + Jump Table Cache:

The Timer T7 overflow interrupt is enabled. If a timer T7 overflow occurs, the service routine switches automatically to the local register bank 1. Instead of using the vector table the CPU directly takes the addresses of the service routine.

```
interrupt (CC2_T7INT) _localbank(-1) _stacksize(50) _cached  void
CC2_viTmr7(void)
{ ......
}
void CC2_vInit(void)
{......
  FINT0CSP = 0x8000 | (((unsigned long)&(CC2_viTmr7))>>16);
  FINT0ADDR = (unsigned int)&(CC2_viTmr7);
  BNKSEL0 = 0x0002;    // Set local register bank 1 for Interr.level 12,
group 0
  PSW_IEN    =  1;            // set global interrupt enable
}
```

Fast Bank switching + Jump Table Cache (advanced):

The Timer T7 overflow interrupt is enabled. If a timer T7 overflow occurs, the service routine switches automatically to the local register bank 1. Instead of using the vector table the CPU directly takes the addresses of the service routine.

```
interrupt (CC2_T7INT) _localbank(-1) _stacksize(50) _cached void
CC2_viTmr7(void)
#pragma noframe        // omit the whole interrupt frame
{......
}
void CC2_vInit(void)
{......
  FINT0CSP = 0x8000 | (((unsigned long)&(CC2_viTmr7))>>16);
  FINT0ADDR = (unsigned int)&(CC2_viTmr7);
  BNKSEL0 = 0x0002;    // Set local register bank 1 for Interr.level 12,
                       // group 0
  PSW_IEN    =  1;     // set global interrupt enable
}
```

This is the fastest way to process an interrupt call because the extension **#pragma noframe** omits the whole interrupt frame. Using this extension makes the user responsible for storing all the controller specific registers, like data pointers, multiply registers, etc..

4 Conclusion

The architecture of the XC16x supports several powerful mechanisms for fast and flexible response to service requests from various sources. For optimized code and dataflow the customer has to analyze the real time requirements of their application. The interrupt architecture of the XC16x together with the different tool chains, perfectly supports these requirements.

http://www.infineon.com

Published by Infineon Technologies AG