

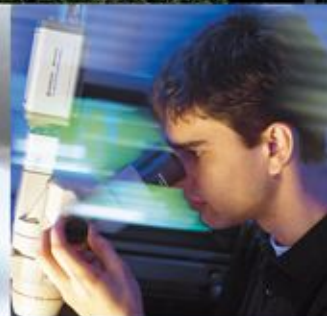
THE INSIDER'S GUIDE TO PLANNING XC2200 FAMILY DESIGNS

An Engineer's Introduction To The XC2200 Family

MICHAEL BEACH MIET

DAVID GREENHILL MIET

www.hitex.com



Embedding Software Quality

Published by Hitex (UK) Ltd.

ISBN 0-9549988 8

First published November 2007

Hitex (UK) Ltd.
Sir William Lyons Road
University Of Warwick Science Park
Coventry, CV4 7EZ
United Kingdom

Credits

Authors: Michael Beach
David Greenhill
Editor: Alison Wenlock

© Copyright Hitex (UK) Ltd. 20/11/2007

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system or transmitted in form or by any means, electronic, mechanical or photocopying, recording or otherwise without the prior written permission of the Publisher.

Contents

1.	The XC2200 Enhanced 16/32-Bit CPU	10
1.1	XC2200 On-Chip Memory.....	10
1.1.1	SBRAM Standby RAM.....	10
1.1.2	Marker Memory	10
1.1.3	DSRAM Data RAM	10
1.1.4	DPRAM Dual Port RAM.....	10
1.1.5	PSRAM Program SRAM.....	10
1.1.6	Automatic RAM Parity Checking.....	10
1.2	FLASH Memory	11
1.2.1	FLASH Memory Overview	11
1.2.2	Endurance And Data Retention	11
1.2.3	Intellectual Property And Data Protection In FLASH	11
1.2.4	EEPROM Emulation	12
1.2.5	FLASH Low Power Mode.....	12
1.3	32-bit Operations	12
2.	The System Control Unit	14
2.1	Clock Generator Unit	15
2.1.2	The Phase-Locked Loop (PLL)	17
2.1.3	Oscillator Watchdog.....	19
2.2	Synchronous CAPCOM2 & CAPCOM6 timer start.....	19
2.3	Watchdog	19
2.4	IO Port Driver Temperature Compensation.....	19
2.5	Reset.....	20
2.5.2	Register Access Control	21
2.6	Power Management.....	22
2.6.1	Powering The XC2200.....	22
2.6.2	The Embedded Voltage Regulators	23
2.6.3	Power Domains Summary	24
2.6.4	EVR Voltage Control.....	24
2.6.5	Core Power Validation System	24
2.7	Power Management.....	25
2.7.1	Supply Watchdog For DMP_IO_0/1 Sub-Domains.....	26
3.	ADC0 & ADC1	28
3.1	Introduction.....	28
3.1.1	Analog Features	28
3.1.2	Digital Features	31
3.1.3	Conversion Request Sources	31
4.	The MultiCAN Peripheral	34
4.1	Introduction.....	34
4.2	MultiCAN	34
4.2.1	Message Objects.....	34
4.3	MultiCAN Module Modes and Features	36
4.3.1	The CAN Frame Counter	36
4.3.2	Analyse Mode	36
4.3.3	Loop Back Mode.....	36
4.3.4	Single Transmit Trial.....	36
4.3.5	Enhanced FIFO Feature	37

4.3.6	Gateway Mode	37
4.3.7	CAN Interrupt Nodes	38
5.	Universal Serial Interface Channel	40
5.1.1	USIC Channel Configurations	40
5.1.2	Typical USIC Applications.....	41
6.	The CAPCOM6E Motor Controller	44
6.1	CAPCOM6E Operation	44
6.2	Other CAPCOM6E Applications.....	45
7.	XC2200 Software Development	47
7.1	Outline	47
7.2	The Development Tools.....	47
7.2.1	HiTOP Debugger & IDE	47
7.2.2	Which Compiler?	47
7.2.3	Other Recommended Tools	48
7.3	Startup Code	48
7.4	Interrupt / Trap Vector Table	50
7.5	Stack Usage	50
7.6	Memory Models	51
7.7	Accessing Peripherals	51
7.8	Interrupt Service Routines	52
7.9	In-Line Functions	52
7.9.1	Inline Assembler	53
7.10	Linker Script Files	53
7.11	Hardware Debugging Tools	54
7.12	Using HiTOP.....	55
7.13	Extended Features	56
7.13.1	Time Analyser.....	57
7.13.2	Execution Profile / Performance Analysis	57
7.13.3	Trace Features	59
7.14	Summary	60

Introduction

The XC2000 series is the 4th variation of the Infineon C166 family, that has been a leader in automotive systems since 1990. Three different applications will be addressed.

The XC2200 is the first family designed for body systems control like BCM (Body Control Module), Gateway, HVAC (Heating ventilation air-condition) and door modules. The XC2300 family will cover safety applications like Airbag, EPS and ABS, where as the XC2700 family will address power train systems.

Whilst the C166 and XC166 have been extensively used in such systems in the past, the huge increase in vehicle networking, increased energy efficiency and the adoption of IEC61508 SIL3 for safety-critical automotive systems like ABS and power steering demands major enhancements to the CPU core features and peripheral set. For simpler components like lamp control, window actuators, CPU cost is paramount. In seat actuators, airbag, power steering and ABS, cost is still important but long term reliability is crucial due to their safety-critical nature. To cover this range of applications the XC2000 includes family members with as few as 64 pins but with up to 176 pins for complex systems that for example, may include the need to control 4 separate brushless DC motors. As members of the same family they are binary compatible, share the same development tools and the peripheral set is in most cases 100% compatible with the XC16x so existing investment in tools and software development can be re-used. An AUTOSAR library is available that allows easy integration of existing code into XC2000 programs is available. AUTOSAR is rapidly becoming the standard for body systems application development.

The feature size has shrunk to 130nm and this has allowed an 80MHz clock and reduced power consumption. However over and above this manufacturing process-lead reduction, many advanced power saving techniques have been introduced such as variable low voltage operation, various levels of standby for major peripherals like the ADC plus some novel power-down/wakeup facilities, with the result that current demand ranges from 50uA in standby mode to 60mA at full CPU performance. Device power is derived from a single 3V or 5V input with on-board regulators providing the core voltage. By splitting the chip area into independent “power domains”, it is possible to run with a mixed 3V and 5V IO system so that for example, the ADC can measure across a full 5V range. Power supply integrity is monitored by brown-out and Vdd spike detection.

The XC2000 sits on the boundary between high-end 16-bit and low-end 32-bit CPUs. Although it is primarily a 16-bit instruction set machine, it can manipulate 32-bit quantities and via the MAC effectively has a second 32-bit ALU. This incorporates a 40-bit multiply-and-accumulate function that is primarily used for DSP-like operations but which is also used seamlessly for normal arithmetic functions by the Keil and Tasking C166 compilers.

Besides a conventional crystal-based external clock source, the XC2000 can run from a purely internal oscillator to reduce component count. This can be trimmed to provide improved accuracy through software.

Software security both in terms of IP protection and operational reliability are addressed through a number of measures. The FLASH EPROM includes a memory protection unit that allows protection against unauthorised READs both by probe programs bootloaded into internal RAM and even electron beam scanning of the FLASH array.

With body systems being very communications-intensive, the XC2000 now has the MultiCAN CAN module from the TC1796. This offers up to 6 CAN modules with 256 message objects with a TTCAN (Time-Triggered CAN) mode and a CPU-transparent gateway mode.

The USIC peripheral is a completely flexible serial communications system that can be configured to provide up to 4 channels of asynchronous (UART, LIN), synchronous (SPI, I2C, i2S) channels, or any combination thereof.

Finally, there are two synchronizable ADCs with a total of 30 channels that can yield 10-bit results in 1.2us and which have a basic signal pre-processing capability.

1. The XC2200 Enhanced 16/32-Bit CPU

The XC2200 CPU is closely based on the XC166's C166SV2 core but with a major increase in on-chip memory. The emphasis is on long-term reliability and security with parity checking on RAM areas and completely a revised FLASH protection mechanism to prevent unauthorised access to program code and constants.

1.1 XC2200 On-Chip Memory

1.1.1 SBRAM Standby RAM

This is a new RAM area and unlike other on-chip memories, this 1k RAM does not appear in the memory map. It is addressed via 4 SFRs in a serial fashion so it has only virtual addresses and requires a software driver to utilize (Infineon will release a suitable driver). It is powered from wake-up power domain and is thus non-volatile.

1.1.2 Marker Memory

There are two “marker_memory” registers that provide two additional words of standby data storage.

1.1.3 DSRAM Data RAM

There is 16k of DSRAM at 0xA000 for general purpose data storage (no code execution possible). As the range 0xA000 – 0xBFFF is not in the DPP3 page, this area is best reserved for infrequently used data unless a spare DPP is available.

1.1.4 DPRAM Dual Port RAM

There is 2k of dual ported RAM at 0xF600 which includes a bit-addressable area at 0xFD00. Code execution from this RAM is not possible.

1.1.5 PSRAM Program SRAM

The 64k PSRAM is located at 0xE0000 and provides program code storage that has the same fast timing characteristics as the FLASH modules but is relatively slow for READ/WRITE data access. It is commonly used to hold bootloaded programs that perform tasks like FLASH programming. It has its own parity generation and checking plus write protection. The PSRAM can be split into READ/WRITE and READ-only sections that are always a multiple of 4k bytes. Illegal writes into a READ-only area force a trap.

1.1.6 Automatic RAM Parity Checking

In avionics software written to DO178B/B and A, the Continuous Background Integrity Test (BIT) is mandatory. Such tests constantly check CPU functionality and configuration. A major component of BIT is the continuous RAM integrity test. This is a non-destructive WRITE/READ test of each RAM location to verify its continued correct operation. The XC2200 allows a further level of security by performing an automatic parity check of any data READ from RAM. A parity error causes a trap to be generated to deal with it safely. RAM parity errors are extremely rare and result from extreme events like Electromagnetic Pulse (EMP) or penetration by alpha particles and other nuclear radiations, the latter being more likely in aircraft. However in a safety-critical system they must be allowed for as they are theoretically possible.

Note: All parity-protected RAM should be initialised by the user application at power-up.

1.2 FLASH Memory

1.2.1 FLASH Memory Overview

Current members of the XC2200 family have FLASH EPROM sizes up to 768KB, split into 4KB sectors. Note: one 4KB sector is reserved for XC2200 system configuration purposes (FLASH, USIC, oscillator trim, voltage regulator adjustments etc.) and is not available to the user. A similar hardware-based error correction system is used to that found in the XC16x and TriCore families. Thus the ability to automatically correct single-bit errors (SEC) and detect double-bit errors (DED) and read the FLASH cell voltage margins to extend data retention is preserved, making the device especially suitable for long service life applications or products that operate at extremes of temperature.

Relying on the error correction system and with no active FLASH management in the user's program, worst-case FLASH data retention is stated as 20 years with a total endurance of 1000 updates or 5 years with an endurance of 15000 updates at max. $T_j = 85^{\circ}\text{C}$.

The projected failure rate is less than 1 defective part per million per year. Thus 1000 hours of device operation per year equates to 1 FIT i.e. 1 failure per billion hours of operation or a 1 in $1\text{E}9$ chance of a random failure if a device had a typical 12 months use. This is significantly better than some other automotive microcontrollers which have a typical FIT of 8 i.e. 8 failures per billion hours of operation and underlines the XC2200's high-reliability credentials.

1.2.2 Endurance And Data Retention

It should be noted that endurance and data retention are related in that in a high endurance requirement usually implies a lower retention capability as the time between updates is short. For example, adaptive data such as a last entry key code record may only need to be retained between vehicle starts (updated twice per day), with perhaps a worst case of say 24 months since the last activation if the vehicle is stored. In contrast, the main program code must last for say 20 years, with or only one or two updates.

1.2.3 Intellectual Property And Data Protection In FLASH

A new protection mechanism disallows unauthorised access to FLASH contents to prevent IP theft.

The flash memory is protected against accidental data changes by requiring the execution of special command sequences to change data. A crashed program randomly writing data all over the memory space has a very small chance of accidentally triggering these command sequences. Indeed there is a much greater likelihood that it could randomly jump to the section of the user's program that contains the functions that disable write protection but even in this case, any illegal command sequence will flip the FLASH controller back into READ mode to limit any damage. The importance of conducting either a power-up or continuous FLASH contents checksum or CRC cannot be over-emphasised!

READ protection is the major defence against unauthorised disassembly of program code. Here any attempt to boot the CPU from external bus or from internal RAM (possibly via bootstrap loading) will result in READs to the FLASH being inhibited and in addition, the OCDS debug interface is disabled.

By default the READ protection also enables a full FLASH memory write protection to prevent code dump programs being covertly programmed into a vacant FLASH area that subsequently try to READ the FLASH contents out via a serial port. Programming and

erasure of the FLASH can only occur if the user deliberately disables the WRITE protection via the right passwords.

Individual sectors can be WRITE protected to prevent unauthorised alteration of code or data. READ and WRITE protection is configured by the user by installing passwords into two special security pages in the reserved configuration sector in the FLASH. A 64-bit password is used to temporarily remove READ and/or WRITE protection so that the protected command sequences become active. If command sequences are not completed by a special single-cycle command, the unprotected state ends with the next RESET.

The 4x 16-bit keywords (i.e. 64-bits) are programmed by the user into the Security Page 0 in the reserved configuration sector. Special measures are taken to protect this information against power-cycling or electron beam attacks by making the protected state the default. This is done by encoding the configuration bits in bit pairs with the protected state being triggered by “00” or “11” combinations. Attack by X-Rays designed to remove FLASH cell charge to yield a “0” value will inevitably trigger full protection.

1.2.4 EEPROM Emulation

The XC2200 FLASH can be used as a substitute for EEPROM through the use of a special driver software, available from Infineon. Such algorithms are relatively commonplace and indeed some are proprietary and require a licence. They all seek to extend the endurance of the FLASH to around the 1E5 level by the use of multiple pages or copies of stored data. However they all have fundamental problems when power is removed part-way through the programming or erase phases. Previous data is usually preserved until the programming algorithm is certain that the writing of the next page is complete but where a power failure occurs during programming of the new page, the next CPU boot-up might find an apparently correctly programmed new page. However the FLASH cells may not be fully programmed and the data retention be drastically reduced. With a conventional FLASH array, this situation cannot be detected and a failure is possible. The XC2200 FLASH has the ability to check the voltage margins in FLASH cells so that incomplete programming can be detected and suspect data re-written to achieve normal margins and thus yield the expected data retention.

1.2.5 FLASH Low Power Mode

By default the CPU reads the FLASH continuously by issuing a continuous stream of sequential addresses, much like external burst-mode FLASH devices. This means that the XC2200 FLASH is never idle. When combined with the 5-stage pipeline system, this can give a significant performance boost but it does increase power consumption. If low power operation is crucial, this feature can be disabled. Where the program consists of short linear sequences then this will not result in a significant performance loss.

1.3 32-bit Operations

The XC2200 sits between the 16-bit XC16x and the 32-bit TC1796. Although it is strictly speaking still a 16-bit machine, it has an arithmetic performance greater than some low-end 32-bit devices. The CPU core has a mix of 1, 8, 16 and 32-bit instructions and both the Keil and Tasking compilers make full use of these to achieve a balance of high code density by using 8-bit operations where possible and fast arithmetic performance for 32-bit operations. Depending on the type of calculation appearing in C source, the compilers will decide to use either the 32/16-bit ALU or the 40-bit MAC unit.

The recently released Tasking VX compiler implements the C99 standard for the C166/XC16x & XC2000 families, adding support for “long long” 64-bit data types, commonly found on 32-bit CPUs. This assists in the porting of existing applications from older 32-bit CPUs.

2. The System Control Unit

The System Control Unit (SCU) contains the fundamental device settings that directly influence power consumption, CPU performance and system robustness in the presence of poor power supplies and clocks. The functions are grouped as follows:

- i. Clock Generation
- ii. Reset Operation
- iii. Power Supply
- iv. Global State Control
- v. Wake-up Timer
- vi. Register Access Control
- vii. Watchdog Timer
- viii. External Interrupts
- ix. Temperature Compensation

2.1 Clock Generator Unit

The clock generation is extremely flexible and allows configurations with optimal performance and power consumptions to be set. The following input clock sources are available:

- i. A high precision crystal oscillator of between 4 and 16MHz (OSC_HP)
- ii. A low-power but low accuracy (best case 3%), on-chip 5MHz oscillator (IOSC)
- iii. An ultra-low power 400kHz wake-up oscillator (WU_OSC)
- iv. An external directly-driven pin DRIN (oscillator module)
- v. An external directly-drive pin DRTC for real time clock use
- vi. VCO contained within the Phase-locked loop (PLL) with watchdog capability
- vii. The raw clock sources (not WU_OSC) can be multiplied up by a Phase Locked Loop (PLL) to yield system clock frequencies up to 80MHz.

The system clock (F_{SYS}) is derived from four selectable clocks:

F_{PLL} PLL clock

F_{WU} Wake-Up clock

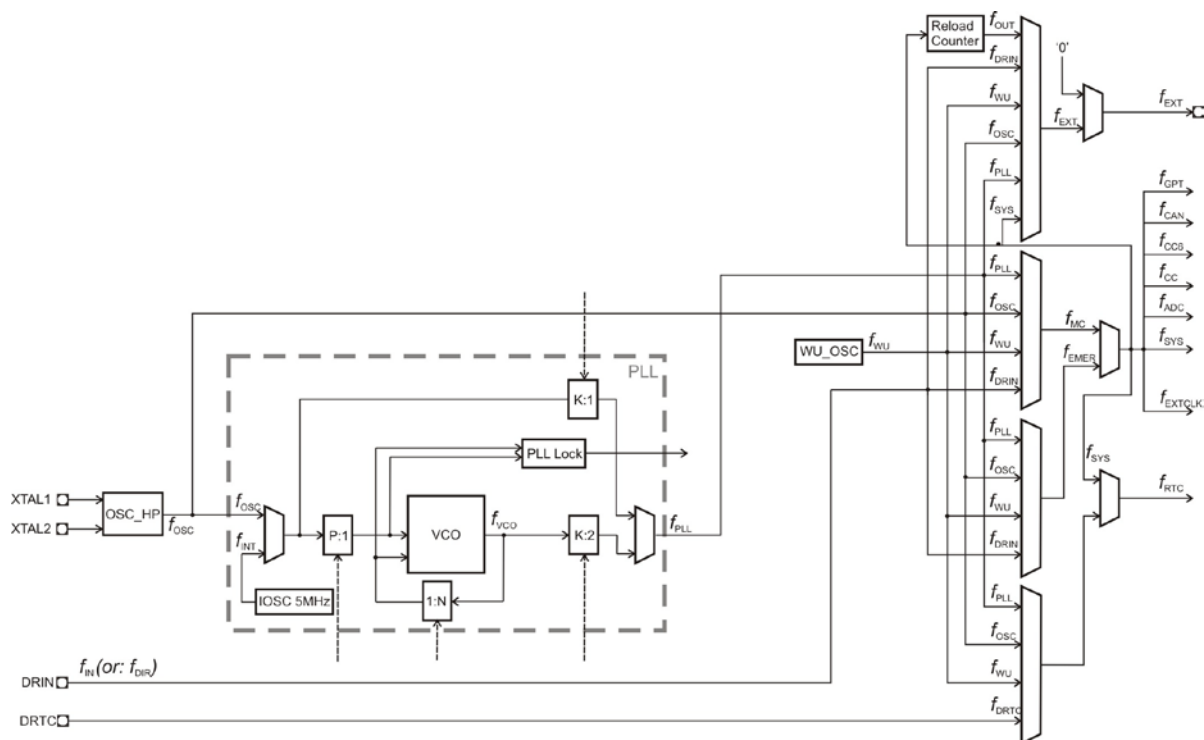
The Direct Clock F_{OSC} , from the XTAL1 Pin

F_{DIR} Input DIRIN as Direct Clock Input

For the exact characteristics of these inputs, please refer to the XC2200 datasheet.

2.1.1.1 Clock Distribution

The main clock fed to the CPU and the peripherals is F_{SYS} . This appears under various different names (F_{ADC} , F_{CC} , F_{CC6} etc.) as the input clock to the peripherals. In most cases, the peripheral will contain some sort of prescaler to reduce the F_{SYS} clock frequency down to something more appropriate to the peripheral's function. The RTC though has its own dedicated clock, F_{RTC} .



The Main XC2200 Internal Clock Distribution System

2.1.2 The Phase-Locked Loop (PLL)

The PLL takes the clock supplied either by the High-Precision Oscillator (OSC_HP) or the low-precision internal approximately 5MHz oscillator (IOSC) and multiplies it up to a frequency of typically 66 or 80MHz to drive the CPU and peripheral system. It has its own power supply to reduce jitter caused by disturbances from elsewhere in the device.

The OSC_HP is intended to use a crystal and provides a very accurate clock source. The requirements for the crystal selection and overall test of the oscillator components is covered in the XC2200 datasheet. The importance of selecting correct component values cannot be over-emphasised if the long term reliability of the system is to be guaranteed.

2.1.2.1 Internal PLL Clock

The internal PLL clock IOSC, does not use an external XTAL or module and runs at a nominal 5MHz. This can be used for the oscillator watchdog or driving the PLL. Its frequency can be tuned via software using the PLLOSCCON register in a range of up to +/- 30% according to:

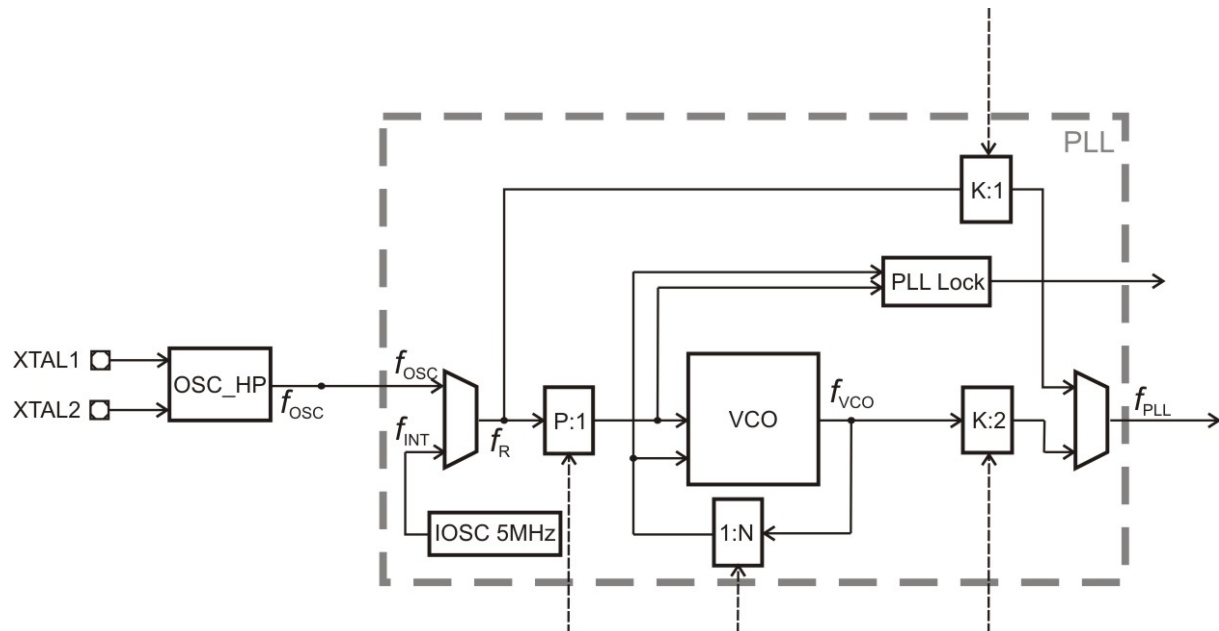
$$\text{Adjustment} = (\text{OSCTRIM}[6] \times 30) - (2 \times (\text{OSCTRIM}[5,1]/15))$$

The adjustment is made in software by comparing the period of a reference input waveform applied to for example, the CAPCOM unit with a target value. The IOSC frequency is then adjusted until the required period is measured by the CAPCOM, indicating that the ratio of the applied frequency and IOSC frequency is correct i.e. 5MHz.

2.1.2.2 PLL Operation

Normal PLL operation sees the FOSC input frequency divided by P by N and then divided by K2, according to:

$$F_{PLL} = (F_R * N) / (K2 * P)$$



2.1.2.3 Prescaler Mode

Prescaler mode takes clock from F_{OSC} to F_{PLL} with the K1 divider so that F_{PLL} is now given by:

$$F_{PLL} = F_R / K1$$

This mode allows the lowest possible F_{SYS} of $F_R/1024$ which can be down to 5kHz.

2.1.2.4 PLL Free-Running Mode

The PLL can free run i.e. the VCO runs with no crystal or IOSC reference. This is the mode used while the PLL achieves a lock state after power-up. The exact F_{VCO} depends on the selected VCO band gap (0 or 1). Typical values are $F_{VCO_BASE} = 26\text{MHz}$ with bandgap 0 and $F_{VCO_BASE} = 49\text{MHz}$ with bandgap 1.

The PLL frequency is then given by:

$$F_{PLL} = F_{VCO_base} / K2$$

2.1.2.5 Wake-Up Oscillator

The wake-up oscillator is an internal unit with a frequency of around 400kHz. The frequency can be selected from a range of 4 values between 150kHz and 560kHz with an accuracy of +/-10%. It requires just 1uA to operate.

2.1.3 Oscillator Watchdog

The oscillator watchdog monitors the incoming clock from OSC_HP and checks if it is in range ($\geq 300\text{kHz}$) for operation in PLL normal mode. The internal oscillator IOSC frequency F_{INT} is used as a reference so therefore the internal oscillator has to be active for the watchdog to function.

2.1.3.1 System Variable Clock Output

The system clock output EXTCLK can be also a programmable frequency output. It is derived from F_{SYS} . Through a reload-counter frequency divider.

2.2 Synchronous CAPCOM2 & CAPCOM6 timer start

All CAPCOMs can be started simultaneously (GLCCST bit) . This means T7/8 in the normal CAPCOM2 can be started simultaneously with the CAPCOM6s' T12 & T13.

2.3 Watchdog

Watchdog can create a pre-warning interrupt which occurs before a final watchdog timeout and reset. The CPU must enter IDLE state to wait for RESET.

In the pre-warning state, the CPU (if it is able) can save certain data that might record why the watchdog reset occurred. Traditionally this is usually very hard to do but on the XC2200 it is catered for.

Should two watchdog timeouts occur in succession a “double watchdog timeout” is detected and results in the CPU being kept in permanent reset once the pre-warning state has finished.

2.4 IO Port Driver Temperature Compensation

The RC oscillator (OSC_TC) generates a clock signal with a frequency of approximately 10 MHz, that is significantly influenced by the device temperature. The number of cycles of this per reference period (derived from F_{SYS}) is latched into TCLR.THCOUNT and this is compared by software to determine the IO port pad driver strength, set by TCLR.TCC. A software driver is needed to realise this functionality.

2.5 Reset

There are now 4 types of reset. System, debug, internal application, application

2.5.1.1 RESET Handling

An ESR0/ESR1/ESR2 reset request trigger leads to a configurable reset.

Software

A software reset request trigger leads to a configurable reset. The type of reset can be configured via RSTCON0.SW.

Watchdog Timer

A WDT reset request trigger leads to a configurable reset. The type of reset can be configured via RSTCON1.WDT. A WDT reset is requested on a WDT overflow event.

CPU

A CPU reset request trigger leads to a configurable reset. The type of reset can be configured via RSTCON0.CPU. A CPU reset is requested when instruction SRST is executed.

Memory Parity

A MP reset request trigger leads to a configurable reset. The type of reset can be configured via RSTCON1.MP.

The following sources can trigger a reset:

- i. External power-on hardware reset; PORST, (cold reset)
- ii. External System Request reset triggers; ESR0, ESR1, and ESR2, (cold/warm reset)
- iii. Supply Watchdog reset request trigger; (SWD), (cold reset)
- iv. Power Validation reset request triggers; (cold reset)
- v. Watchdog Timer (WDT) reset request trigger, (cold/warm reset)
- vi. Software reset (SW), (warm reset)
- vii. CPU reset (CPU), (warm reset)
- viii. Debug (OCDS) reset request trigger, (warm reset)
- ix. Memory parity and ECC (MP), (warm reset)
- x. JTAG reset (special reset)

2.5.1.2 Power-On Reset Pin PORST

The PORST input pin requests a power-on reset. Driving the PORST pin low causes a class 0 reset of the entire device. The PORST signal is forwarded to the system via the start-up reset signal.

Note: The slope of the trailing edge for the PORST pin activation should not be longer than 50 ms from de-asserted to asserted. The PORST pin itself can be asserted longer than 50ms.

PORST is equipped with a noise suppression filter that suppresses glitches below 10 ns pulse width. PORST pulses with a width above 100 ns are safely recognized.

2.5.1.3 Effects Of The Different Reset Types

Module / Function		Application Reset	Internal Application Reset	Debug Reset	System Reset
CPU Core		X	X	X	X
Peripherals (except SCU and RTC)		X	X	X	X
On-chip RAMs1)	Static	X	Not affected	Not affected	X
	RTC	Not affected	Not affected	X	X
	DPRAM	Not affected, reliable	Not affected, reliable	Not affected, reliable	Affected, undefined
	PSRAM	Not affected, reliable	Not affected, reliable	Not affected, reliable	Affected, undefined
	DSRAM	Not affected, reliable	Not affected, reliable	Not affected, reliable	Affected, undefined
Flash Memory		X 2)	X 2)	Not affected, reliable	X
JTAG Interface		Not affected	Not affected	Not affected	Not affected
OCDS		Not affected	Not affected	X	X
Oscillators, PLL		Not affected	Not affected	Not affected	X
Port Pins		Not affected	X	Not affected	X
Pins ESRx		Not affected	X	Not affected	X

2.5.2 Register Access Control

This goes beyond the EINIT protection found on other Infineon 16-bit microcontrollers. Certain registers have startup protection so that they cannot be written until STCON.STP is cleared by software. Others have write protection after EINIT. The secured mode uses a command (command 4) written into the SLC register to allow one write to a protected register whereupon write protection resumes. After writing command 4 in Secured Mode the lock mechanism remains disabled until the next write access to a register on the PD+ bus, i.e. accesses to registers outside this area do not re-activate the protection.

Startup, write protect, secured unprotected modes. Protected registers are always READable. Some SCU traps are shared with CAPCOM2 interrupt sources. The RAT trap occurs when a protected register is accessed.

2.6 Power Management

For a detailed examination of the hardware aspects of the XC2200 power supply system, please refer to Infineon Application Note: AP16103

Although the XC2000 family has a number of different power domains and a flexible configuration, the processor can be run from a single external power supply of between 3.0V and 5.5V. As far as the power supply is concerned, the core is split into two power domains. Most of the on-chip logic is located in first core domain (DMP_1) whilst the wake-up mechanism and other important device infrastructure and a Standby RAM are located in DMP_M.

2.6.1 Powering The XC2200

The XC2200 power is applied as shown below:

Symbol	Pin	Function
VDDPA	20	Digital Pad Supply Voltage for Domain A
VDDPB	2, 36, 38, 72, 74, 108, 110, 144	Digital Pad Supply Voltage for Domain B (EVRs fed from VDDPB)
VSS	1, 37, 73, 109	Digital Ground
VDDIM	15	Digital Core Supply Voltage for Domain M
VDDI1	54, 91, 127	Digital Core Supply Voltage for Domain 1
VAREF0		Reference Voltage for A/D Converter ADC0
VAREF1	29	Reference Voltage for A/D Converter ADC1
VAGND	31	Reference Ground for A/D Converters ADC0/1

2.6.1.1 Single Supply Designs

Typically most systems need only one power supply and with the Embedded Voltage Regulators (“EVRs”) and the supply watchdog, a low cost 3-pin voltage regulator is all that is required.

A suitable voltage regulator is the TLE 7274. This is a monolithic integrated low-drop voltage regulator for load currents up to 300 mA. An input voltage of up to 42 Volts is regulated to a nominal 5 Volts, with a precision of $\pm 2\%$. The standby current consumption is typically just 20 μ A, making it suitable for use in applications which are permanently connected to a battery supply. Where the application allows the system to be switched off completely, the TLE 7276 is an alternative. Here there is an additional control pin (Inhibit) that disables the 5V output.

2.6.1.1.1 XC2200 Single Supply V_{DD} Connections

The following pins must connected as shown for a single 5V supply.

5V: V_{DDPA} , V_{DDPE} , V_{AGND} , V_{AREF1} , V_{AREF2} , V_{DDI} , V_{DDM} , T_{REF} , $/TESTM$, $/PORST$

2.6.1.2 Dual Voltage Systems

Some applications require two external voltage domains, typically 3.3V for off-chip memory devices and 5V for analog sensors. The XC2200 multiple power domain concept supports this.

The 3.3V digital supply voltage for all the ports except Port5, Port6 & Port 15 (i.e. the analog dual function ports) whilst the analog GND and reference pins are supplied with 5V. For the 3.3V digital supply, a TLE4266-2-GV33 150mA regulator would be suitable and for the analog supply, the TLE4296-2-GV50 will deliver 5V at 30mA with a 4% accuracy. To make sure that the two regulators power-up and power-down together, their INHIBIT pins should both be connected together.

2.6.1.2.1 XC2200 Dual Supply V_{DD} Connections

The following pins must be connected as shown for a dual 5V/3V3 supply.

5V: V_{AGND} , V_{AREF1} , V_{AREF2} ,
 3.3V: V_{DDPA} , V_{DDPE} , V_{DDI} , V_{DDM} , T_{REF} , $/TESTM$, $/PORST$

2.6.2 The Embedded Voltage Regulators

These core domain voltages can either be generated by the on-chip EVRs or from external voltage regulators. Even if the on-chip regulators are used, the V_{DDI1} and V_{DDIM} pins need to be externally de-coupled (see note on capacitor selection below). Either or both of the core domains can be switched off (by disabling the respective EVR) or lowered to 1.0V. This will save leakage current within that area. The two core domains are supervised by the Power Validation Circuit (PVC) that provides two monitoring levels that can request an interrupt or a reset. These mechanisms should prevent the processor from damage due, for instance, to external short circuits.

The EVR's for each of the core domains can be controlled independently although some combinations are not allowed. Each EVR can supply a reference voltage from a Low Power Reference or from a High Precision Bandgap, as well as being switched off altogether. If the EVR is switched off, an external supply can be connected via the V_{DDI} pins.

The Low Power Reference (LPR) is used for two purposes:

- i. For operation in a Power State other than Full Active
- ii. For special power saving in the Full Active Power State
- iii. The High Precision Bandgap (HPB) is also used for two purposes
- iv. To provide a very stable reference for the two EVR's when operating Full Active
- v. To provide a reference for the flash memory

As well as the core domains, the I/O is also divided in two parts: DMP_IO_0 contains all the ADC related I/O's and DMP_IO_1 contains the remaining system and communication I/O's.

2.6.3 Power Domains Summary

- i. DMP_1 is main power domain for core and most peripherals.
- ii. DMP_M is a small power domain for SBRAM, WDOG, power controller, clock control, ERU & ESR (wake-up domain)
- iii. DMP_IO_0 is the IO port sub-domain for the ADC. DMP_IO_0 has its own supply watchdog
- iv. DMP_IO_1 ("pad sub-domain") for all other IO peripherals.

2.6.4 EVR Voltage Control

There is a dedicated EVR voltage regulator for each core power domain, EVR_M and EVR_1. The two regulators can be referenced either to an internal High-Precision Bandgap ("HPB") source or a Low Power Reference ("LPR"). The core voltage can be adjusted in +/- 0.1mV/degC steps to compensate for temperature variations. The core voltage can be either 1.5 or 1.0V. If the voltage drops below 1.5V, an interrupt occurs that allows an orderly transfer to a low-power 1.0V reference.

Supply management allows the temporary reduction of the supply voltage of major parts of the logic, or even its complete disconnection. This drastically reduces the power consumed because of leakage current, which is a particular problem at high temperatures. Several power reduction modes provide the optimal balance of power

To ensure the stability of the voltage regulators, each core voltage domain must be buffered with a ceramic capacitor. For DMP_M a 680 nF capacitor is recommended and for DMP_1, three 220 nF capacitors, one for each V_{DDI} pin. There is a maximum value of 4.7uF for each voltage domain which must not be exceeded. All buffer capacitors must be connected as close to the V_{DDI} pins as possible to keep the resistance of the board tracks below 2 Ω .

2.6.5 Core Power Validation System

There are two core Power Validation Circuits ("PVC") implemented that continuously monitor the voltages of the two core power domains. If the core power supply doesn't reach the value required for proper functioning of the main power domain (DMP_M), PVC_M issues a reset request trigger that can be forwarded to the RESET system. Likewise, if the core power supply doesn't reach the value required for proper functionality of the application power domain (DMP_1), PVC_1 generates a similar reset request trigger.

When used with a suitable timer from say the GPT12E unit, the two voltage monitors allow the detection of potentially damaging conditions such as over-current due to a short-circuit.

The PVC_M and PVC_1 can be configured to monitor two independent and programmable voltage levels and the relative values of these and the core clock mode determine the overall operating mode of the CPU core.

Active Mode: DMP_M and DMP_1 are both at 1.5V, and use F_{PLL} clock source or direct drive from XTAL1 pin with no PLL.
The peripherals and CPU are both active at running at full speed.

Stopover Mode: DMP_M = 1.5V, DMP_1 = 0V, DMP_M runs from wake-up clock, DMP_1 not powered
DMP_1 modules' values lost

Other modes are possible but have not been validated and so are not recommended:

- Stopover Mode: DMP_M = 1.5V, DMP_1 = 1.5V, DMP_M runs from wake-up clock, DMP_1 clock off
DMP_1 modules' values preserved, wake-up self-timed or externally triggered
- Standby Mode: DMP_M = 1.0V, DMP_1 = 1.0V, DMP_M not clocked
DMP_M and DMP_1 modules' values preserved but not active
- Standby Mode: DMP_M = 1.0V, DMP_1 = 0V, DMP_M not clocked
DMP_M modules' values preserved but not active, DMP_1 modules' values lost

2.7 Power Management

There are many different ways to use the power system. The XC2000 family has three mechanisms to control power management that can partly be used in parallel. These are:

- i. Supply Voltage Management. This allows the supply voltage to be reduced temporarily, or even completely disconnected from major parts of the logic. This can have quite an effect on power consumption because of leakage current, particularly at high temperatures.
- ii. There are several power reduction modes to provide a balance of wake-up time and power reduction.
- iii. Clock Generation Management. This controls the frequency and distribution of the internal and external clock signals. The clocks to inactive areas of logic are disabled automatically, but the user can also reduce the CPU clock which will also reduce the power that is consumed.
- iv. A programmable output clock (FOUT) can be used to control external circuitry.
- v. Peripheral Management. This allows peripheral modules to be disabled temporarily. The CPU can also be switched off while the peripherals continue to operate.

Either external signals, or the on-chip wake-up timer can be used to return from the power reduction modes. This internal timer can generate cyclic wake-up signals allowing intermittent operation of the processor.

It is important that the software configuration is correct to avoid unintended states. There are recommended sequences to ensure that the power supply and clock system function as intended.

There are two basic power transfers that cover all of the transitions from one Power State to another, a Ramp-up Power Transfer, or a Ramp-down Power Transfer. A power transfer has to be requested by certain triggers from the ESR Pin(s), the Wake Up Timer, software or a Power-on Reset. The power-on reset starts a power transfer based on the default values from the appropriate registers leading to Full Active Mode with the Low-Power Reference

active. Power state I using the High-Precision Bandgap is automatically configured and entered.

A Ramp-down trigger can only be requested from software and requests a transition into a power saving mode that is not power state I. Ramp-up triggers are used to return from a power saving mode to Full Active Mode.

2.7.1 Supply Watchdog For DMP_IO_0/1 Sub-Domains

2.7.1.1 Overview

The supply voltage of DMP_IO_0 is monitored by a Supply Watchdog (SWD). The Supply Watchdog has three threshold values. The first is fixed at the minimum operating voltage for the IO domain (V_{VAL}). If this threshold is not reached, the device is held in reset. Two adjustable levels are available and configured to request an interrupt or a reset. Both threshold compare levels should not be used as a reset level at the same time.

2.7.1.2 Software-Driven /PORST Emulation

The supply watchdog is a completely independent voltage monitoring system that provides information on power supply integrity and provides a means of giving an orderly shutdown in the event of a power failure. It can be used to emulate the operation of a conventional 4- or 5- terminal voltage regulator. It continuously compares the supply voltage for the DMP_IO_0 & _1 power domains against three thresholds. The lowest one (V_{VAL}) is determined by Infineon and represents the minimum possible operating voltage and falling below this causes the device to enter a RESET state to prevent erroneous operation. V_{VAL} is typically 2.0V but the current XC2200 datasheet should be consulted for the correct value for the device in use.

The other two thresholds are user-defined for voltage ranges between 2.9V and 5.5V in 16 steps. The exact voltages are distributed non-linearly within this range and are chosen by IFX to match the most common real-life situations. Although the minimum operating voltage detection cannot be disabled, the two configurable thresholds can be disabled thus saving power.

Should the supply voltage cross these programmable thresholds a flag is set or an interrupt requested which allows software to detect for example, an imminent power-fail situation i.e. the voltage passes in a downwards direction through both thresholds. At power-up, once the voltage exceeds V_{VAL} , the ramp-up of the supply can be monitored as it crosses both thresholds in the upwards direction, allowing software to decide when to start the application. In addition to setting a flag or requesting an interrupt, crossing the thresholds can also force an immediate reset however in this situation, setting both thresholds to the same value should be avoided. Overall, the supply watchdog allows an orderly start and power-down, without the need for a power-on reset (/PORST pin = low). However, if the system design incorporates traditional /PORST generation then this replaces completely the supply watchdog function.

3. ADC0 & ADC1

3.1 Introduction

The XC2200 is equipped with two independent 10-bit, 16-channel analog to digital convertors (ADC), based on the successive approximation principle. They can be operated autonomously or as a synchronized pair to perform simultaneous current and voltage measurements, for example. Each ADC has a variety of conversion triggering methods, data handling processes and storage options. The ADC is a very powerful sub-systems and has a very large number of potential operating modes to solve all common analog sampling tasks. The most important of these are covered in the following sections.

3.1.1 Analog Features

The XC2200 ADCs are able to utilise a voltage range of $V_{DDPA} = 3.3V$ (min) to 0-5V (nominal), regardless of the voltage levels used by other IO ports. This is by virtue of the XC2200's split power domains. Here, the ADCs operate in Domain A and 5V, whilst the remaining digital IO is in Domain B at 3.3V. Thus the measurable input range is 0V to V_{DDPA} .

The sample time can be adjusted to suit different signal source impedances. Two different sample times can be defined for each ADC and each channel can be assigned to use any one of them. Typical leakage current is +/-200nA.

3.1.1.1 Dual 3.3V & 5V References

When a particular analog input has a voltage equal to the reference, a full scale binary value is generated (0x3FF for a 10-bit conversion). On the XC2200, in addition to the normal V_{AREF} reference voltage pin each ADC can use a second reference voltage, applied to it Channel 0. If VAREF is set to 5V and Channel 0 is hard-wired to a 3.3V supply, the user can choose which reference particular channel will use. Thus even on the same ADC, both 5V and 3.3V signal sources can be assured of reaching a full scale binary value.

3.1.1.2 Conversion & Sample Time Tuning

The minimum possible 10-bit conversion time (including the sample time) is approximately 1.15us with a total unadjusted error (TUE) of +/-2LSB when running with VDDPA = 5V and a 66MHz system frequency (a.k.a F_{ADC} or F_{SYS}). At 80MHz, the minimum conversion time is 1.5us. This value is valid for the ADC running in an optimised electrical environment, e.g. careful routing of analog and digital signals; the isolation of analog and digital PCB areas; a low noise analog power supply (< 30 mV ripple); infrequent switching of digital pins near to the analog PCB area.

GLOBCTR.DIVA: (Bits 5 to 0)	Clock Period		Sample Time							
	1/Fadci	INPCR.STC: (Bits 7-0)	0.0000	1.0000	3.0000	7.0000	15.0000	31.0000	63.0000	127.0000
0	0.0152		0.0303	0.0455	0.0758	0.1364	0.2576	0.5000	0.9848	1.9545
1	0.0303		0.0606	0.0909	0.1515	0.2727	0.5152	1.0000	1.9697	3.9091
3	0.0606	Fadci <= 16.5MHz	0.1212	0.1818	0.3030	0.5455	1.0303	2.0000	3.9394	7.8182
7	0.1212		0.2424	0.3636	0.6061	1.0909	2.0606	4.0000	7.8788	15.6364
15	0.2424		0.4848	0.7273	1.2121	2.1818	4.1212	8.0000	15.7576	31.2727
31	0.4848		0.9697	1.4545	2.4242	4.3636	8.2424	16.0000	31.5152	62.5455
63	0.9697		1.9394	2.9091	4.8485	8.7273	16.4848	32.0000	63.0303	125.0909
GLOBCTR.DIVA = 000100 INPCR.STC =			00000000	00000001	00000011	00000111	00011111	00111111	11111111	STC
Sample Time (us)			0.1212	0.1818	0.3030	0.5455	1.0303	2.0000	3.9394	7.8182 us
Overall Conversion Time (us)			1.1515	1.2121	1.3333	1.5758	2.0606	3.0303	4.9697	8.8485 us
Varef Source Resistance (Ohms)			117	117	117	117	117	117	117	Ohms
Signal Source Resistance (Ohms)			998	2247	4745	9742	19734	39720	79691	159632 Ohms
GLOBCTR.DIVA = 000111 INPCR.STC =			00000000	00000001	00000011	00000111	00011111	00111111	11111111	STC
Sample Time (us)			0.2424	0.3636	0.6061	1.0909	2.0606	4.0000	7.8788	15.6364 us
Overall Conversion Time (us)			1.2727	1.3939	1.6364	2.1212	3.0909	5.0303	8.9091	16.6667 us
Varef Source Resistance (Ohms)			485	485	485	485	485	485	485	Ohms
Signal Source Resistance (Ohms)			3496	5995	10991	20984	40969	80940	160881	320764 Ohms
GLOBCTR.DIVA = 001111 INPCR.STC =			00000000	00000001	00000011	00000111	00011111	00111111	11111111	STC
Sample Time (us)			0.4848	0.7273	1.2121	2.1818	4.1212	8.0000	15.7576	31.2727 us
Overall Conversion Time (us)			1.5152	1.7576	2.2424	3.2121	5.1515	9.0303	16.7879	32.3030 us
Varef Source Resistance (Ohms)			1219	1219	1219	1219	1219	1219	1219	Ohms
Signal Source Resistance (Ohms)			8493	13489	23482	43467	83438	163379	323263	643029 Ohms
GLOBCTR.DIVA = 011111 INPCR.STC =			00000000	00000001	00000011	00000111	00011111	00111111	11111111	STC
Sample Time (us)			0.9697	1.4545	2.4242	4.3636	8.2424	16.0000	31.5152	62.5455 us
Overall Conversion Time (us)			2.0000	2.4848	3.4545	5.3939	9.2727	17.0303	32.5455	63.5758 us
Varef Source Resistance (Ohms)			2688	2688	2688	2688	2688	2688	2688	Ohms
Signal Source Resistance (Ohms)			18485	28478	48463	88434	168376	328259	648025	1287557 Ohms
GLOBCTR.DIVA = 111111 INPCR.STC =			00000000	00000001	00000011	00000111	00011111	00111111	11111111	STC
Sample Time (us)			1.9394	2.9091	4.8485	8.7273	16.4848	32.0000	63.0303	125.0909 us
Overall Conversion Time (us)			2.9697	3.9394	5.8788	9.7576	17.5152	33.0303	64.0606	126.1212 us
Varef Source Resistance (Ohms)			5627	5627	5627	5627	5627	5627	5627	Ohms
Signal Source Resistance (Ohms)			38471	58456	98427	178368	338252	658018	1297550	2576615 Ohms

Typical Conversion Times Possible For a 66MHz System Clock

3.1.1.3 Debugger Support

One common problem when debugging ADC software is that should the debugger interrogate a results register to display the converted value on the host PC, the ADC will consider the data to have been read and in the case of the wait-for-read and FIFO modes, the state of the ADC will be have been disturbed. To overcome this, there are alternative “VIEW” results registers “RESRVx” that allow the result to be displayed by a debugger without changing the ADC state.

3.1.1.4 Limit Checking

Each channel's conversion result can be compared with an upper and lower limit register to determine whether the CPU needs to be interrupted. This can significantly reduce CPU loading where channels have to be continuously monitored for values above or below a limit value or within a range.

Typical applications for limit checking are temperature monitoring or over-current sensing. As long as the measured temperature value is below a predefined value, the CPU does not need to be interrupted. In this case, a channel event should be generated only if the conversion result is above the limit value to indicate an over-temperature condition. If the conversion of the analog temperature input signal is part of a time-based auto-scan sequence, the CPU load for over-temperature monitoring is zero until the over-temperature condition is actually detected. In the case of an over-current protection, the channel event created can be used to disable PWM generation to reduce the current. This makes use of the link between the ADC module and the CAPCOM6 PWM units to allow a fast response without CPU intervention.

3.1.1.5 Channel Aliasing

Channel 0 and 1 can be ganged with other channels to allow a single signal to be measured on two different channels. This is equivalent to routing one signal to two pins (e.g. CH0 and CH2) but here without any external connection or additional leakage current inaccuracies. CH0 (and CH1) can be setup in a completely different way to its partner channel having an alternative trigger source or limit checking. The conversion result appears in the results registers for both CH0 and the partner channel. This is particularly useful where a certain signal has to be read on a regular timed basis in say CH2 but occasionally be read into CH0 in response to some occasional asynchronous event.

Another example would be in an AC drive with a common current sensor for all phases, parallel phase current measurements are required whereby two of the three phase currents have to be measured together, e.g. phases 1-2, then 1-3, and finally 2-3. Here the trigger for the aliased conversions would be the PWM drivers for each phase pair.

3.1.1.6 Synchronised ADCs

By using the synchronised ADC mode, truly simultaneous sampling of two channels can take place. Here one ADC is nominated as the master and one or more channels can be coupled to their equivalent number in the slave ADC. Any request from any source for a conversion on the master ADC's channel will now cause a simultaneous request to be seen by the slave. Channels in the slave can be used as normal regardless of whether they are coupled to their equivalent in the master ADC but any slave channel will have a running conversion abandoned should a master conversion request arrive before it has completed.

ADC modules that are to have synchronised channels must have the same F_{ACDI} and F_{ADCD} clocks so that there are no timing differences. However synchronised channels can have different sampling times (STC). The channel alias feature (see above) can still be used when channel synchronisation is enabled. Thus a conversion request from a master can be a trigger for say CH0, which is paired with CHx.

The only restriction on synchronised channels sampling is that a target results register for any channel that is part of a master-slave pairing should not also be used by a channel that uses the data reduction mode.

3.1.2 Digital Features

3.1.2.1 Conversion Results Handling

There are a total of 8 conversion result registers, each of which can be assigned to any channel in the ADC via a simple pointer mechanism. This is especially useful for auto-scan modes of up to 7 channels (one results register used for other modes) where the results of each conversion in the sequence will be left in sequential locations where a single PEC (peripheral event controller) channel can move them automatically into RAM for processing once the scan sequence has ended.

Each result register has a Data Valid flag to show that data has not been read since it was converted. It is possible to gang otherwise unused results registers together to form a FIFO. This is suitable for buffer non-time-critical data pending CPU service.

A degree of filtering is built into the ADC through the “data reduction” filter. Up to 4 sequential results can be summed automatically, with an interrupt being triggered on the last addition. By dividing the accumulated value by the number of samples, a simple average can be calculated with very little CPU overhead. Data loss is prevented by the wait-for-read mechanism which applies to each result register. No new conversion from whatever trigger event will begin until the last result has been read from the results register.

3.1.2.2 ADC Power Saving Modes

In common with other major XC2200 peripherals, the ADC contains a number of power-saving features, ranging from a complete power-off to a variety of standby modes. After reset the ADC is in the off state and no conversions are possible. The required mode (Dual parallel ADCs or synchronised mode) must be set before switching the ADCs on. It takes around 10µs for the ADC to become ready for the first conversion.

The lowest power consumption operating mode is Slow-Standby Mode. Here the ADC enters a low-power state after each conversion however an extra 10µs wake-up time is added to the user-defined sample time.

Fast standby mode is similar but takes only 3µs to wake-up but the power saving is less significant. Normal mode means that the ADC is always active and will make conversions in using the sample time pre-set by the user in the INPCR.STC register field.

3.1.3 Conversion Request Sources

3.1.3.1 Overview

There are a wide variety of events that can cause an analog to digital conversion each of which can be assigned a priority. Within the ADC system, conversions can be triggered by auto-scan sequences of a number of channels plus simple software requests for a spot conversion. An internal scheduler scans the request sources regularly to find the channel with the highest priority. The priority of each source can be programmed individually. The current conversion is aborted immediately if a conversion with a higher priority is found. Equidistant sampling can be achieved by connecting external timers through the external triggering system.

3.1.3.2 Conversion Requests From Outside The ADC

External triggers for conversions can come directly from specific port pins or via hard-wired links between the ADC and peripherals like the CAPCOM or CAPCOM6E.. Alternatively, conversion requests can come from the Event Request Unit (ERU) via the ERU_GUOTx and ERU_TOUTx signals that allow many additional port pins and other peripherals (MultiCAN, USiC, pattern detection logic etc.) to request conversions. Thus conversions can be requested from almost any CPU or peripheral state. The DAVE tools is especially useful for configuring this aspect of the ADC.

Request Source 0	ADC0	ADC1
REQGT0A	ERU_GOUT0	ERU_GOUT0
REQGT0B	ERU_GOUT1	ERU_GOUT1
REQGT0C	P6.0	P6.0
REQGT0D	0	0
REQTR0A	ERU_TOUT1	ERU_TOUT1
REQTR0B	CCU60_T13_PM	CCU61_T13_PM
REQTR0C	P6.1	P6.1
REQTR0D	P6.3	P6.3
Request Source 1		
REQGT1A	ERU_GOUT0	ERU_GOUT0
REQGT1B	ERU_GOUT1	ERU_GOUT1
REQGT1C	P6.0	P6.0
REQGT1D	0	0
REQTR1A	ERU_TOUT1	ERU_TOUT1
REQTR1B	CCU60_T13_PM	CCU61_T13_PM
REQTR1C	P6.1	P6.1
REQTR1D	P6.3	P6.3
Request Source 2		
REQGT2A	ERU_GOUT0	ERU_GOUT0
REQGT2B	ERU_GOUT1	ERU_GOUT1
REQGT2C	P6.0	P6.0
REQGT2D	0	0
REQTR2A	ERU_TOUT1	ERU_TOUT1
REQTR2B	CCU62_T13_PM	CCU63_T13_PM
REQTR2C	P6.1	P6.1
REQTR2D	P6.3	P6.3

Possible external conversion
request sources for ADCs

ERU_GOUTx = pattern match or no match on port pins

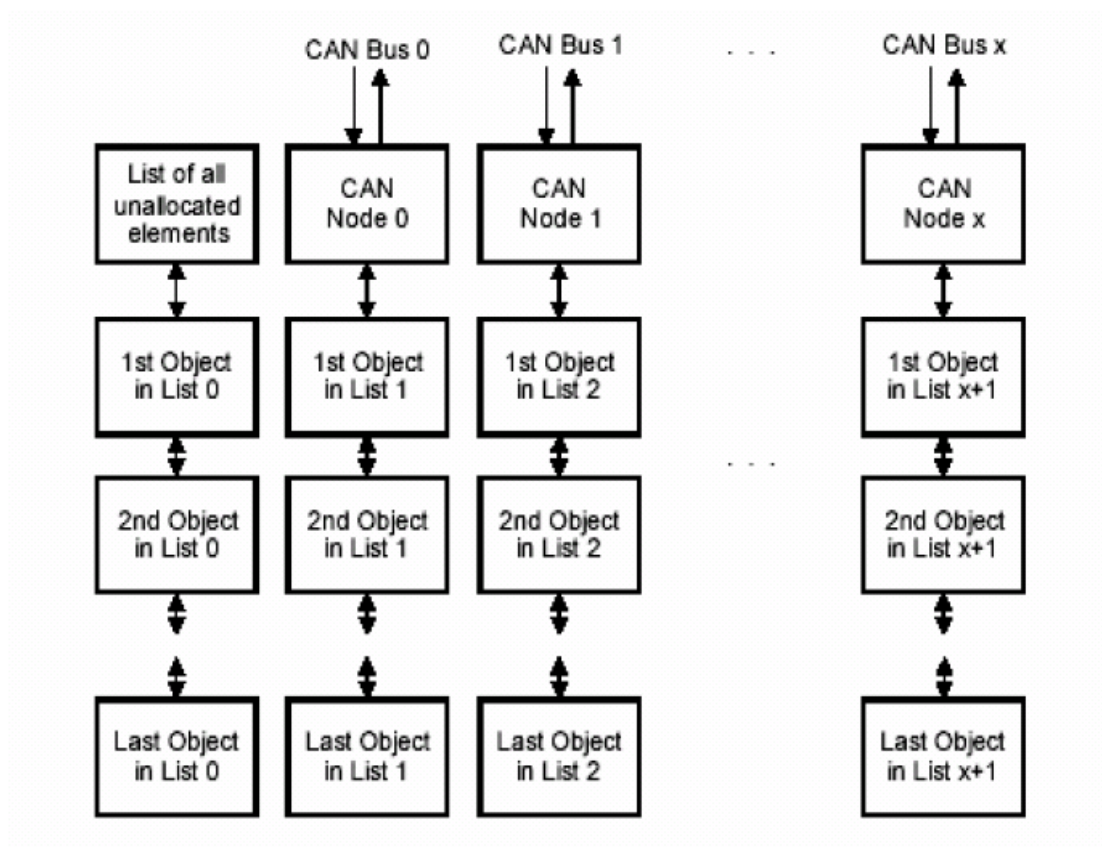
ERU_TOUTx = combination of pattern match or triggers from other peripherals

The connections to the CAPCOM and CAPCOM6 units are particularly important for current control in power inverters as they allow the ADC to take current (or voltage) samples at crucial points in the PWM waveform. ADC0 is directly coupled to CAPCOM6_0 & _2 whilst ADC1 is connected with CAPCOM6_1 & _3 (where fitted).

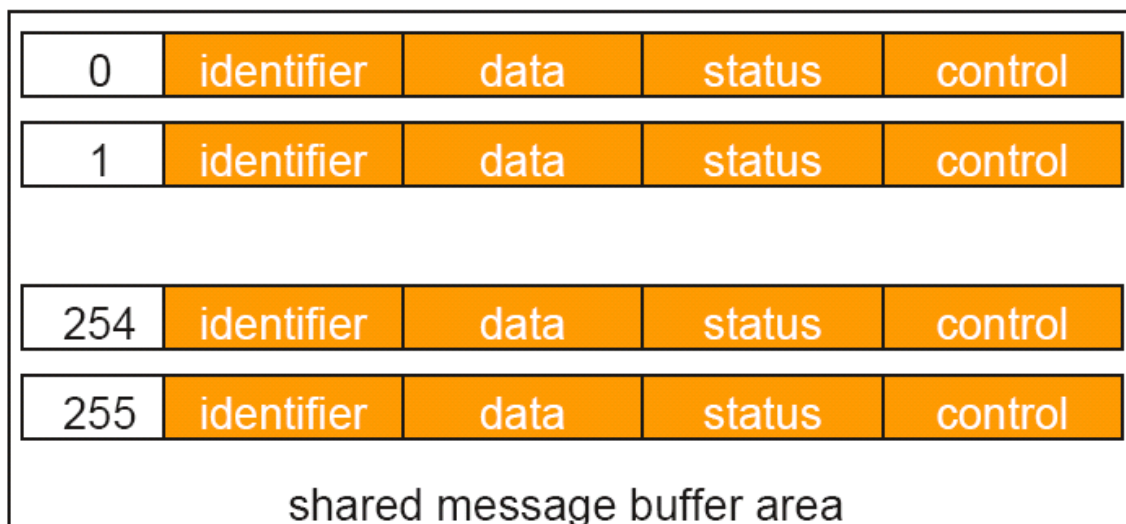
4. The MultiCAN Peripheral

4.1 Introduction

The CAN peripheral used in the XC2200 family differs from the TwinCAN Module that is presently used by the XC16x Family. The MultiCAN Module contains 5 (or 6) independent CAN nodes which can share the 128 (or 256) independent message objects. The number of CAN modules available on the different members of the XC2200 family varies between 2 and 5. The Module conforms to the CAN v2.0 B active specification and can receive and transmit standard frames with 11 and 29-bit identifiers up to a baud rate of 1Mbit/s.



All changes relating to lists are made using the Command Panel. There are 7 commands to initialise and modify lists and these command codes and any arguments are written to the Panel Control Register. The message object registers are located at consecutive 32-bit addresses.



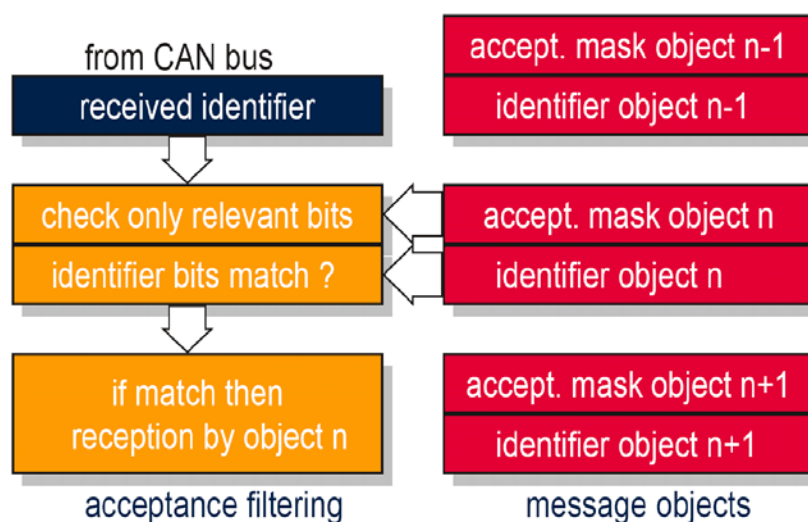
Each message object consists of the following sub-registers:

- MO Control
- MO Data
- MO Arbitration ID
- MO Acceptance Mask
- MO Interrupt Pointer
- MO FIFO/Gateway Pointer
- MO Function Control

As per the earlier C & XC16x processors, the acceptance mask of a message object can be programmed to accept or reject individual or groups of identifiers.

The received identifier from the bus is compared to the programmed identifiers in the message objects:

- i. Each identifier bit can be individually tagged “don’t care” by an acceptance mask, groups of identifiers can be received by the same message object.
- ii. Each message object has its own identifier and acceptance mask



4.3 MultiCAN Module Modes and Features

4.3.1 The CAN Frame Counter

This can be setup to be in one of three modes by the CFSEL Bit field in the NFCR register for each CAN node.

These modes are:

- i. Frame Count Mode. The counter is incremented after successful transmission and / or reception of a frame.
- ii. Time Stamp Mode. “The frame counter is incremented with the beginning of a new bit time. When the transmission / reception of a frame starts, the value of the frame counter is captured and stored to the CFC field of the register NFCR. After the successful transfer of the frame the captured value is copied to the CFC field of the Interrupt Pointer Register of the message object involved in the transfer”!!
- iii. Bit Timing Mode. Used for baud rate detection and analysis of the bit timing.

4.3.2 Analyse Mode

In this mode, the CAN node will receive messages but its transmit pin is held at the recessive level which means that a CAN node in Analyse Mode will not send an acknowledge frame. It is just listening to the bus without interacting with it. User software can be written to monitor the traffic on the bus without any danger of changing bus behaviour.

4.3.3 Loop Back Mode

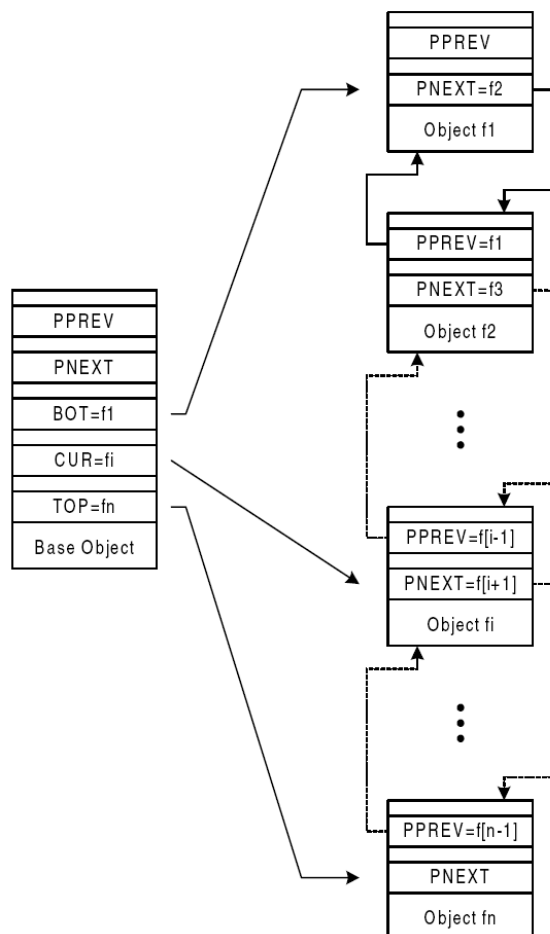
The MultiCAN Module includes an internal CAN bus to which CAN nodes during (for instance) development can be connected. A switch is used to connect each node to either this internal bus or to the external bus via the transceivers.

4.3.4 Single Transmit Trial

In this mode, enabled when the STT bit of the message object function register is set, a message object will only be transmitted once. The CAN node will clear the transmission request flag will be cleared even if the transmission was not successful due to bus errors. This is the kind of feature that is important in MIL CAN applications.

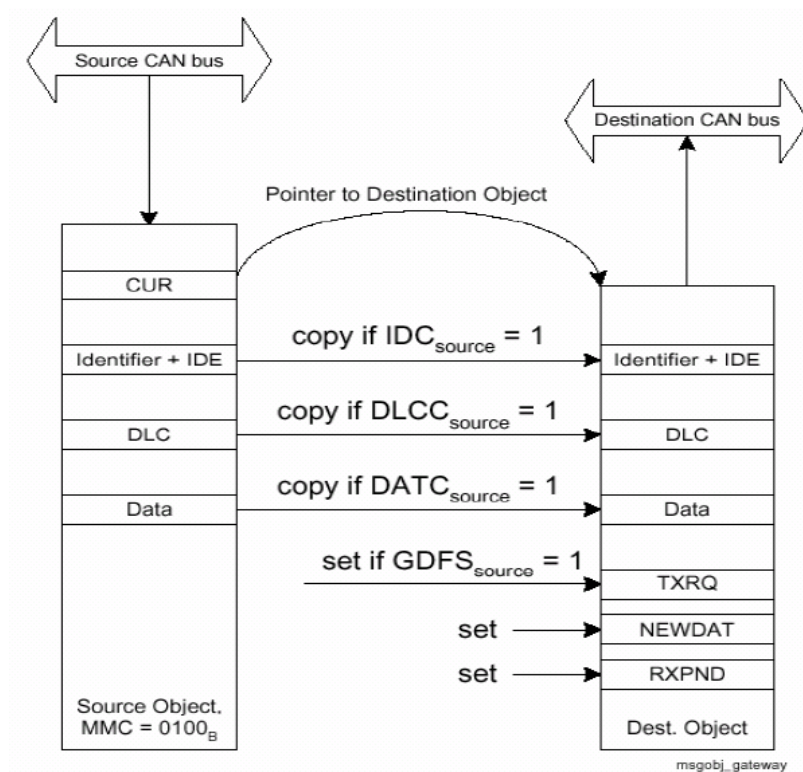
4.3.5 Enhanced FIFO Feature

The CAN message objects can be linked together to create one or more FIFO buffers. The smallest FIFO would be a single message object that is both the FIFO base and a slave – this would have no practical use! The biggest FIFO would use all of the message objects in the MultiCAN Module. Any number of FIFO's of any size is possible, the limitation being the number of message objects available. An interrupt can be linked to an object within the FIFO that can be used to signal the end of a predefined series of message transfers or to issue a warning to the CPU when the FIFO is full.



4.3.6 Gateway Mode

The Gateway mode allows an automatic transfer between two CAN networks running, at any baud rates, without CPU interaction.

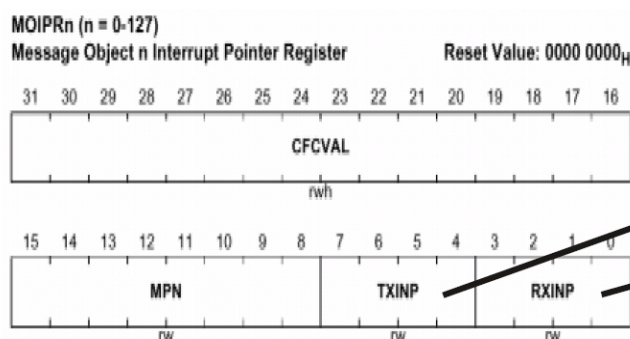


4.3.7 CAN Interrupt Nodes

The MultiCAN module includes 16 individually programmable interrupt outputs. Most interrupt requests can be individually routed to one of the 16 interrupt output lines. Each CAN node has four interrupt sources:

- Correct Message Object Transfer
- CAN Error
- List Length or Object Error
- Frame Counter Overflow / Event

Interrupt source allocation to any of 16 interrupt nodes is via SFRs and the Message Object 'n' Interrupt Pointer Register MOIPRn.



selection via node pointer

interrupt x

interrupt y

Interrupt generation is extremely flexible with the separation of tasks being supported and different request priorities programmable.

5. Universal Serial Interface Channel

Communications are an important part of any application and it is not uncommon for systems to require many different types of serial interface. UARTs are often used for diagnostic terminals or connection to legacy devices; I2C and SPI are commonly used for connections to EEPROMs, real time clocks, control blocks for complex peripheral devices; LIN is widely used for small CPU networks. Many microcontrollers include such peripherals but in many cases the number of each type of interface does not quite fit the needs of the application.

To address this problem, the XC2200 includes a completely new type of serial interface device. The USIC (Universal Serial Interface Channel) replaces multiple dedicated serial communications peripherals with 6 (XC2287) completely reconfigurable serial channels that can each act as an UART, SSC/SPI, I2C, I2S or LIN interface, depending on how they are programmed. It is therefore possible to create up to 6 independent UARTs, something that is required in a surprising large number of applications.

An USIC module contains two independent serial channels and 3 such modules are implemented on the XC2287, giving a total of 6 channels. The USIC registers are located at 0x204000 in the XC2200 memory space.

5.1.1 USIC Channel Configurations

Each USIC channel can be configured to handle UART, SSC, LIN, IIC and IIS serial protocols with individually-set Baudrates. Full duplex data transfers are supported and each channel can be reprogrammed without chip reset.

The mode of each USIC channel is set in software as shown below. Here USIC0 channel 0 is set up for UART mode and channel 1 for I2C operation:

```

/// -----
/// Configuration of the U0C0 Channel Control Register (Mode & Interrupts):
/// -----
/// - ASC (SCI, UART) Protocol is selected
/// - The parity generation is disabled

U0C0_CCR      =  0x6002;      // load U0C0 channel control register

/// -----
/// Configuration of the U0C1 Channel Control Register (Mode & Interrupts):
/// -----
/// - IIC protocol is selected
/// - The parity generation is disabled

U0C1_CCR      =  0x0004;      // load U0C1 channel control register

```

Once the mode of the individual channels have been set, the baudrate, number of data, start and stop bits are configured much like on a conventional serial port e.g.:

```

//  initializes the Universal Serial Interface Channel (USIC) U0C0
U0C0_ASC_vInit();

//  initializes the Universal Serial Interface Channel (USIC) U0C1
U0C1_IIC_vInit();

```

5.1.1.1 USIC Mode: UART

The UART (ASC, asynchronous serial channel) can receive and /transmit at a maximum baud rate of $F_{SYS}/4$, although the usable baud rate range is 1.2kBaud to 3.5MBaud. It supports between 1 and 63 bits per data frame, processed either MSB or LSB first.

5.1.1.2 USIC Mode: LIN

The LIN mode allows a low-cost network to be created at up to 20kBaud. Data transfers are based in the UART protocol. Baud rate detection is made possible by a built-in capture event of the baud rate generator. The LIN checksum is generated under SW control to give greater flexibility.

5.1.1.3 USIC Mode: SSC/SPI

The USIC SSC/SPI (synchronous serial channel) mode is supported with or without slave chipselect lines. The max Baud rate in slave mode is based on F_{SYS} . In master mode the maximum Baud rate is $F_{SYS}/2$, although the usable Baud rate range is 2kBaud to 10MBaud. The number of data bits per data frame is 1 to 63 (more with an explicit stop condition), sent MSB or LSB first.

5.1.1.4 USIC Mode: I2C (Inter-IC Bus)

The I2C channel supports the standard application Baud rates of up to 100kBaud and up to 400kBaud ("FAST mode").

5.1.1.5 USIC Mode: IIS (infotainment audio bus)

The Infotainment audio bus can receive at up to a maximum Baud rate of F_{SYS} and transmit at up to a maximum baud rate of $F_{SYS} / 2$. The usable Baud rate range is up to 26MBaud.

5.1.2 Typical USIC Applications

With the large number of serial communications channels available, the XC2200 can cope with the demands of most applications. Some common USIC configurations from real products are listed in the following:

5.1.2.1 Example 1: Precision Instrument Controller

- 2x UART for connection to legacy measurement devices
- 1x UART for RS485 transceiver
- 1x SPI for connection to Ethernet transceiver
- 1x UART for connection to FDTI serial to USB interface
- 1x I2C for 14-bit ADC

5.1.2.2 Example 2: Industrial Controller

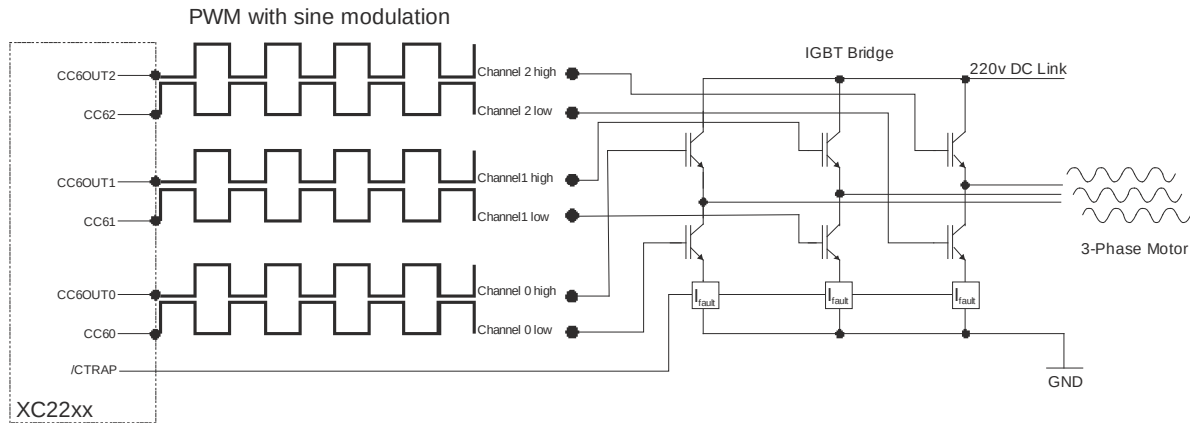
- 1x UART for connection to dumb terminal
- 1x UART for RS485 transceiver
- 1x SPI for connection to Ethernet transceiver
- 1x UART for connection to FDTI serial to USB interface
- 1x UART for Profibus interface module
- 1x I2C for simple IO expansion

5.1.2.3 Example 3: RS232-USB Data Concentrator

- 5x UART for connection to legacy devices
- 1x UART to connect to FDTI USB interface

6. The CAPCOM6E Motor Controller

The XC2200 family is intended for motor drive applications, having up to four identical dedicated motor control peripheral, giving the potential for controlling four AC or DC motors. This peripheral is of the “CAPCOM6E” type, found on the XC16x and features three special CAPCOM channels with *six* outputs (high-low pairs) which make the implementation of high



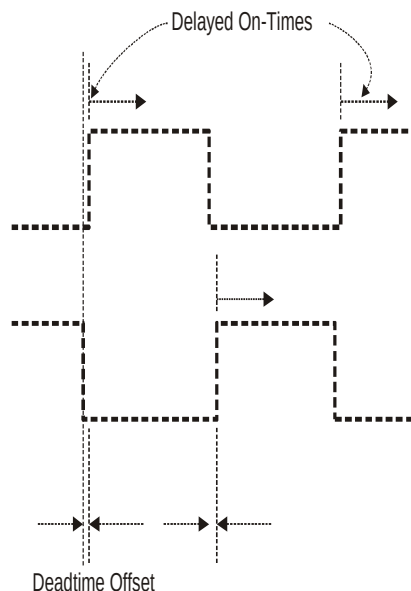
Connecting CAPCOM6E To Half-Bridges For 3-Phase Induction Motors (Simplified)

performance controllers easier. Typical motor types supported are; DC-brushless (with Hall-effect or back-EMF position sensors), AC induction (open-loop and vector), switched-reluctor plus up to 4 independent brushed permanent magnet DC motors. The control of general purpose (i.e. non-motor) power inverters is of course also possible.

6.1 CAPCOM6E Operation

The key CAPCOM6E feature is that each PWM channel has a slave second pin which provides the complementary output for driving power bridges. The required deadtime offset for the slave channel is automatically added in hardware rather than via software, as in the standard CAPCOM units. For safety/over current trips or emergency stop, the special /CTRAP pin forces all outputs into a predefined passive state immediately, essential for the protection of IGBTs and MOSFETs configured in a bridge formation.

The input to the CAPCOM6E unit is the CPU clock (or some multiple of it) for 3-phase and stepper motors but can optionally be accompanied by a three Hall effect position sensors that can be used in several ways to give block commutation in DC brushless applications. In the simple case, for a given pattern on the CC6POSx inputs, an user-defined output pattern can be placed on the output pins to advance the armature to the next position. The detection of Hall input transitions is detected automatically and duplicated Hall codes are rejected to prevent noise causing discontinuities in the code sequence. Alternatively the



Automatic Deadtime Offset Generation With CAPCOM6E

PWM duty ratio can be altered progressively to produce the trapezoidal waveform required by most DC brushless motors.

For DC drives where there are no Hall sensors (“sensorless”), channel 1 can be used to create a phase delay during which the back EMF can be measured to allow the armature position to be estimated. The triggering of the analog to digital converter to make this measurement can be entirely automatic.

The two CAPCOM6E timers, Timers 12 and 13, are usually used as the PWM carrier frequency generator and modulation timebase respectively. Timer 12 can generate compares with three compare registers that have direct connections to the CC6OUT0/1/2 pins. Timer 13 has a single but dedicated compare register and one output pin (CC63), sufficient to allow it to trigger an interrupt that hosts the code required to modulate the raw PWM produced by Timer 12.

Timers 12 and 13 can run at up to 80MHz, allowing carrier frequencies of up to 312.5kHz for an 8-bit PWM resolution or 78.125kHz for 10-bit resolution.

```
void timer12_13_init_XC2287(void)
{
    /* Timer 12 - base carrier generator */
    T120F = con.deadtime0 ; // Set deadtime between low and high
                                // side PWM outputs

    T12P = pwm_period ;      // Set period register to define carrier
                                // frequency

    CTCON = 0x0000 ;        // Timer 12 runs at 0.05us per count
    CTM = 1 ;               // Up/down mode for centre aligned PWM

    /*** Configure Timer13 For Modulation ***/

    CTCON |= 0x0300 ;      // Timer 13 runs at 0.100us per count

    T13P = pwm_period ;    // Set update rate of modulation

    Set_Priority_13(T13IC) ;

    T13IR = 0 ;            // Clear any spurious interrupts
    T13IE = 1 ;            // Enable timer 13 overflow interrupt
}
```

Setting Timers 12 & 13 For PWM Carrier & Modulation

6.2 Other CAPCOM6E Applications

Where the XC22xx application does not involve motor or inverter control, CAPCOM6E can also be used for general timebase generation, digital to analog conversion and input frequency measurement, much like the standard CAPCOM2 unit but at much higher frequencies.

```
void timer13_int(void) interrupt 0x4e using TIMER13_REGS {
    int CCx_temp ;

    #define CC8_image CCx_image /* define temporary registers */
    #define CC9_image CCx_image
    #define CC10_image CCx_image
    #define CC12_image CCy_image
    #define CC13_image CCy_image
    #define CC14_image CCy_image

    /* Calculate Values For Next Interrupt */

    CCx_temp = ((long)((int)VF_con.VF_scaling) *
                ref_sine_table[current_carrier][freq0.acc_bytes[1]])/VF_Div ;

    CC60 = CCx_temp/2 + table_offset ; // Remove LSB and add table offset

    CCx_temp = ((long)((int)VF_con.VF_scaling) *
                ref_sine_table[current_carrier][freq1.acc_bytes[1]])/VF_Div ;

    CC61 = CCx_temp/2 + table_offset ; // Remove LSB and add table offset

    CCx_temp = ((long)((int)VF_con.VF_scaling) *
                ref_sine_table[current_carrier][freq2.acc_bytes[1]])/VF_Div ;

    CC62 = CCx_temp/2 + table_offset ; // Remove LSB and add table offset

    STE12 = 1 ; // Enable loading of compare registers from
                // shadow latch for next cycle
    CT13P = 0 ;

    STE13 = 1 ; // Enable loading of compare registers from shadow
                // latch for next cycle
}
```

Applying Sinewave Modulation On Three Channels

7. XC2200 Software Development

7.1 Outline

Having read the previous chapters, you should now have an understanding of the XC2000 CPU and its major peripherals. In this chapter we will look at how to write and debug C code for this processor. The example programs in this chapter will concentrate on demonstrating how to setup the tool environment for exploiting the unique features of the XC2000 architecture.

7.2 The Development Tools

The two compiler vendors for this architecture are Keil and Altium, who both offer evaluation versions of their tool chains. In order to evaluate program code without any target hardware, these tool chains also offer powerful instruction set / peripheral simulators. This makes for a quick project start and the easy review of code generation for software evaluation exercises. When it is necessary to move the code to real hardware, a switch to an evaluation package including an evaluation board is recommended. Downloading and debugging program code on these boards uses the Infineon On Chip Debugging Support (OCDS) via the standardized JTAG interface.

To access the CPU via JTAG, a host-based interface is provided from Infineon via their "Device Access Server" ("DAS") package. This server provides a generic interface for tool vendors to access the on-chip debug system through a JTAG hardware tool. A widely used and DAS-compatible JTAG tool (USB-Wiggler / TantinoXC) comes from Hitex and will be the reference tool used in this guide. Due to the availability of a DAS server ("JTAG over USB Box") for this JTAG tool, a wide range of debuggers (μ Vision from Keil, CrossView from Altium, UDE from PLS) can also make use of this debugger hardware and adopted tools for this interface are available.

Some evaluation kits already contain a JTAG Wiggler as part of the board which can directly be connected via USB to the development host. This type of board just requires another DAS server (called "JTAG over USB Chip"). The debug environment can easily be switched to this interface.

7.2.1 HiTOP Debugger & IDE

HiTOP is the front end for all Hitex debugging tools and in-circuit emulators. In the case of the XC2000, HiTOP connects to the TantinoXC JTAG debugger. Using JTAG allows HiTOP to download code into the XC2000 FLASH or RAM and then debug the code as it runs on the microcontroller. In addition to its debugging features, HiTOP includes a programmers' editor and support for various compiler tools and MAKE utilities that allow you to maintain existing XC2000 programs.

7.2.2 Which Compiler?

The HiTOP development environment can be used with all the currently existing compiler tools from Keil and Altium (Tasking classic, Tasking VX). All three compilers are professional tools and offer a superior value in terms of code density and speed of execution. Some of the extended features of the Hitex TantinoXC like the program flow trace tool is only supported by Altium compilers and for this reason, we will use this compiler for demonstrating the examples within this chapter.

7.2.3 Other Recommended Tools

7.2.3.1 Development Assistant for C (DA-C)

“Development Assistant for C” (DA-C) is an advanced editor specifically targeted at embedded systems developers. As well as having all the facilities that you would expect in a programmer’s editor, DA-C includes a number of advanced features that help you to produce high quality and well-documented C source code. DA-C includes a static checker that will analyse your code for common programming errors, generate flow charts and calling hierarchies. DA-C also includes a code browser, so that you can easily navigate your code plus a metrics module so that the source code may be analysed using quality standard measures.

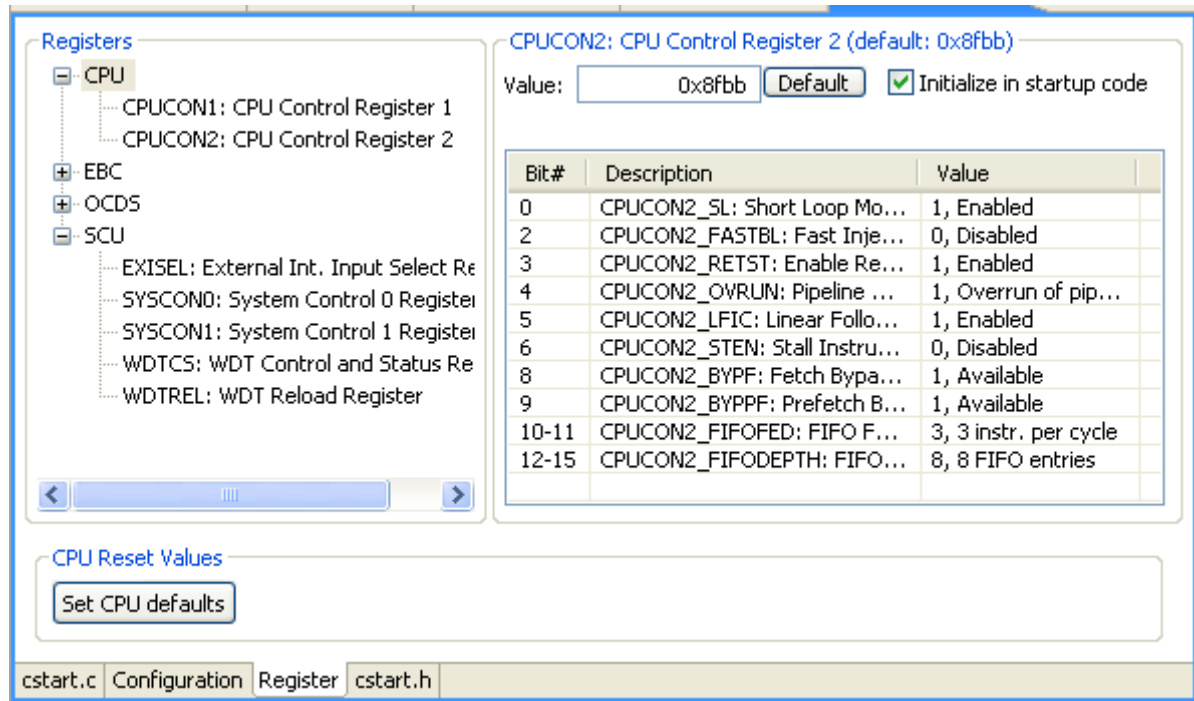
7.2.3.2 TESSY

The TESSY toolset automates the functional testing of embedded microcontrollers and their target hardware. In many industries (especially aerospace and medical), validation of microcontroller firmware is a lengthy and important process. TESSY is especially suitable for testing small footprint microcontrollers which only have limited amounts of on-chip memory. Rather than build a test harness that is downloaded into the target memory of the device under test, TESSY makes no changes to the code under test but builds its test harness in the HiScript language built into the debugger. This way the target application can be fully exercised without the loss of any on-chip resources.

7.3 Startup Code

The tool chain usually offers a dedicated standardized file which represents the startup code. For the Tasking VX this file is provided as a unique file *cstart.c* which in turn gets configured by a user-defined *cstart.h* file. Configuration is done from the Tasking IDE just by double-clicking on the *cstart.c* source file and selecting the appropriate tabs. The startup code is mainly used to setup the C-runtime environment according to the memory model selected. This means initializing RAM (moving constants from FLASH to RAM, clearing RAM) and setting up Heap, Stack and DPP-Registers.

The startup code additionally offers the possibility to maintain the most important registers for initializing the target processor, like watchdog, external bus setup and dedicated system configuration registers. The general recommendation is, to change only those registers that require a different value from their reset default value. For most applications the default reset values are suitable.

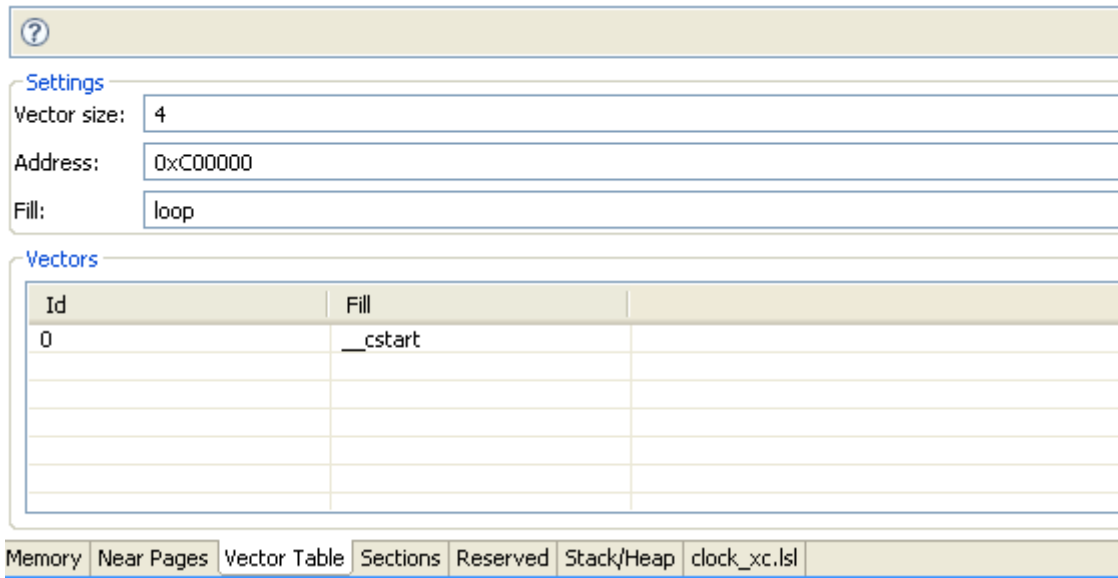


Usually the startup code ends up in setting the EINIT flag, globally enabling the interrupts and calling the *main()* function of the user's application. The startup code configuration leaves interrupts disabled and in the protected mode (EINIT not executed). By this it is possible to initialize additional protected peripherals in any of the user's application modules and delay the execution of EINIT until all these tasks have been finished. This reduces the overhead of always switching off/on the protection scheme during the user's application.

7.4 Interrupt / Trap Vector Table

The interrupt vector table contains the first executed instruction when an interrupt or trap occurs. The vector table thus should always be present, even if no interrupts are planned to be used. At least the trap vectors must be initialized with a simple loop to catch these (mostly unexpected) exceptions. The VECSEG register (default = 0xC00000) must point to the vector table (see startup configuration).

The location and content of the vector table can be controlled by a double-click on the `<project>.isl` file and selecting the appropriate tab. Additionally, dedicated entries in the vector table are generated by the compiler automatically when using the `__interrupt` attribute for a function.



Settings

Vector size: 4

Address: 0xC00000

Fill: loop

Vectors

Id	Fill
0	___cstart

Memory Near Pages Vector Table Sections Reserved Stack/Heap clock_xc.isl

7.5 Stack Usage

Up to four different types of stack can be managed by the Tasking VX compiler: the system stack, the user stack and additionally two dedicated stacks for interrupts using register banks (`user_stack1` and `user_stack2`).

The system stack is the 'hardware' stack addressed by the register SP and is directly used by the processor to store the return address when calling a function. For an interrupt function or trap, it additionally stores the PSW. The compiler also makes use of the system stack to push/pop temporary Registers and SFRs. In the prologue/epilogue code of interrupt functions, registers can also be saved/restored using push/pop instructions.

The user stack is used for variables temporary storage and passing arguments. The user stack pointer is R15.

The compiler has a command line switch (`--user-stack`) to tell that it to avoid using the system stack for return addresses. Instead it calls functions using a branch instruction after pushing the return address onto the user stack. Such function returns by pushing the return address on the system stack directly followed by a return instruction, all in an atomic sequence.

The user-stack1 and user-stack2 are for local register banks, used for interrupts. The processor can be configured to switch to a new local register bank for an interrupt. In the startup code the user stack pointer register (R15) of each local register bank is set to its own user stack. This way the interrupt function does not need to load the user stack pointer in the prologue code, which saves time. If a local register bank is not used for interrupts, the size of the appropriate user-stack can safely be set to zero.

The compiler will calculate the minimum required size for system and user stacks and put these values into the map file. The linker will issue a warning if the stack size settings within the project are not sufficient for the calculated requirements.

7.6 Memory Models

The Tasking VX compiler offers four memory models, distinguished by the way data is used and accessed. Each memory model defines a default memory type for objects that do not have a memory type qualifier specified. By default, the C166 compiler uses the “near” memory model. With this memory model, the most efficient code is generated. With the compiler option **--model** you can specify another memory model.

The table below illustrates the meaning of each memory model:

Model	Memory type	Location	Pointer size	Pointer arithmetic
Near	__near	Near data pages defined at link time	16 bit	16 bit
Far	__far	Anywhere in memory	32 bit	14 bit
segmented huge	__shuge	Anywhere in memory	32 bit	16 bit
Huge	__huge	Anywhere in memory	32 bit	32 bit

The value of the predefined preprocessor symbol `__MODEL__` represents the memory model selected with this option. This can be very helpful in making conditional C code in one source module that will be used for different applications in different memory models.

Program code can always be anywhere in memory and function calls by default are always segmented calls, except for explicitly defined “near” functions, i.e. those contained in the same 64k segment.

7.7 Accessing Peripherals

Once we have built some code and got it running on an XC2000 device, it will at some point be necessary to access the special function registers (SFR) in the peripherals. As all the peripherals are memory-mapped, they can be accessed as normal memory locations. Each SFR location can be accessed by ‘hardwiring’ a volatile pointer to its memory location as shown below.

```
#define SFR    (*((volatile unsigned long *) 0xFFFFF000))
```

These defines are already predefined by the Tasking VX tool chain and are accessible automatically for the selected device within the project. There is even not the requirement to include any header file for this purpose!

7.8 Interrupt Service Routines

In addition to accessing the on-chip peripherals, your C code will have to service interrupt requests. It is possible to convert a standard function into an interrupt service routine (ISR) as shown below:

```
#define TIMER_ISR      20

__interrupt (TIMER_ISR) void msec5_isr(void) {
    T2 = RELOAD_5ms;
    inc_clock();
    signal_clock();
} // end msec5_isr
```

The function qualifier `__interrupt` tells the compiler to create code for proper interrupt handling and additionally include the entry into the interrupt vector table. The vector location is specified by the interrupt number (TIMER_ISR for the above example) and must be altered to the specific interrupt number.

Besides setting the proper interrupt priority level, different options exist to reduce interrupt latency times:

- i. Usage of device hardware by defining an interrupt as a “fast interrupt” with fast register bank switch. This is done by setting the SFRS FINT0ADDR / FINT1ADDR for up to two interrupts. As a result of this, the JMP to the vector location is omitted and the device directly passes control to the specified ISR-Address. Additionally it is possible to automatically switch to a certain register bank by setting one of the BNKSELx registers. This speeds up the register bank switch to the shortest possible time.
- ii. Usage of software controlled register bank switching which is supported from the Tasking VX compiler by the additional function qualifier `__registerbank`. Interrupts using their own local register bank can make use of the compiler option to automatically switch to a dedicated user-stack associated with the register bank.

7.9 In-Line Functions

It is also possible to increase the performance of your code by in-lining your functions. The inline keyword can be applied to any function as shown below:

```
inline void NoSubroutine (void)
{
    ...
}
```

When the inline keyword is used, the function will not be coded as a subroutine but the function code will be inserted at the point at which the function is called, every time it is called. This removes the prologue and epilogue code which is necessary for a subroutine, making its execution time faster. However, you are duplicating the function every time it is called so it is expensive in terms of your FLASH memory.

The Tasking VX compiler will also do a so called smart in-lining automatically for small functions, even if they were not defined as in-line functions. To avoid in-lining you can use the keyword `__noinline`. The compiler's behaviour for in-lining functions can additionally be controlled by command line options as well through pragmas.

7.9.1 Inline Assembler

The compiler also allows you to use Assembler instructions within a C file. This can be done as shown below:

```
__asm ( "NOP");
```

This can be useful if you need to use features that are not supported by the C language, for example save and disable interrupts and restore the interrupt flags.

```
// Clear / Restore interrupts
inline void ClearINT(void) {
    __asm ("PUSH PSW\n"
          "BCLR PSW.11\n");
}

inline void RestoreINT(void) {
    __asm ("POP PSW\n");
}
```

7.10 Linker Script Files

Once you have written your source code and compiled it to object files, these files must be linked together to make the final absolute file. The examples supplied contain linker files with the switches as well as a linker script file *clock_xc.lsl* that describes the target memory layout to the linker, so that it knows how to build the final application. The linker is invoked automatically as the last step of the build/rebuild operation.

The setup for the linker script can be controlled and modified within the Tasking VX interface by double-clicking on the appropriate *.lsl file. This will show up different tabs which contain various settings for the target memory layout and usage:

- **Memory** : shows type (RAM / ROM), start address and size for system and user memories. By default the system memory is initialized with the memory regions available for the selected derivative. Reserved regions are marked and the linker will only map the available internal resources to the application requirements. For applications using additional external memory, the table at the bottom must be filled in accordingly.
- **Near Pages**: shows the fixed values for the DPP-Registers initially set by the startup code. The compiler will use up to the first three DPPs to access near data pages in the near data model.
- **Vector Table**: setup the location and entries in the vector table.
- **Sections**: this table allows the definition of additional memory ranges that can be individually treated by the linker according to their attributes (locate order, initialized data, different run/load addresses).
- **Reserved**: entries in this table mark reserved memory regions that the linker will not occupy with code or data.
- **Stack/Heap**: sets up the sizes (and optionally also fixed addresses) for stack and heap. For near memory model the compiler uses a near heap but optionally a huge heap can be used. For all other memory models the compiler uses a huge heap.

Type	Name	Address	Size	Reserved	
ram	SFR_area	0x00FE00	512	☒	
ram	ESFR_area	0x00F000	512	☒	
ram	PSRAM	0xE00000	64k	☒	
rom	Program_Flash_C10...	0xC10000	704k	☐	
rom	Program_Flash_Con...	0xC0F000	4k	☒	
rom	Program_Flash_C00...	0xC00000	60k	☐	
ram	USIC_registers	0x204000	6k	☒	
ram	MultiCAN_registers	0x200000	16k	☒	
ram	DPRAM	0x00F600	2k	☐	
ram	XSFR_area	0x00E000	4k	☒	
ram	DSRAM	0x00A000	16k	☐	

7.11 Hardware Debugging Tools

Infineon has designed the XC2000 to have the maximum on-chip debug support offered via a standardized JTAG connector. The JTAG interface gives you basic run control of the chip i.e. you can single step lines of code, run and halt. Setting breakpoints and also viewing variables or memory locations is even possible while the system is running. Reading data while running causes only a very small real-time violation, due to the fact that the Infineon OCDS just injects one extra instruction into the execution pipeline in order to achieve this operation.

The JTAG connects to the target through a special debug port, which must be added to your target hardware. The JTAG socket has a standard layout which is common to all Infineon 16 / 32 – Bit devices. The JTAG is brought out of the XC2000 by at least four GPIO pins as secondary functions. This means that when you are using the JTAG, these pins will be unavailable for your application. The TantinoXC offers additional features like Time Analyser, Trace and Performance Measurement. Using these features requires dedicating an additional port pin to the extended debug features. A second extra port pin would be needed, if the user wishes to send a debugger break to an application while it is in the sleep/idle mode.

Layout of the Infineon JTAG connector:

Pin	Signal (*)	Dir (**)	Comment
1	TMS	I	
2	VDDP	-	I/O voltage of processor
3	TDO	O	
4	VSS	-	Signal ground
5	CLKOUT	O	optional
6	VSS	-	Signal ground
7	TDI	I	

8	RSTIN_N	I	Reset input of processor (open-drain)
9	TRST_N	I	
10	BRKOUT_N	O	Time Analyzer Signal Input (optional)
11	TCK	I	
12	VSS	-	Signal ground
13	BRKIN_N	I	Break signal (optional)
14		-	do not connect on target board
15	RES1	-	do not connect on target board
16	RES2	-	do not connect on target board

*) The suffix "_N" denotes a low-active signal.

**) The direction "I" denotes an input of the microcontroller, thus an output of the Tantino.
 "O" denotes the direction from a microcontroller output to the Tantino.

Target Design Rules

- To avoid malfunctions when no debugger is connected to the target, 10-K pull-up resistors should be provided for the JTAG inputs TMS, TCK and TDI.
- The Tantino uses the input pin TRST_N to enable the debug support of the processor. For normal operation a pull-down resistor is required which should not be less than 10-K to allow the Tantino to control this signal.
- The Tantino optionally uses the input pin BRKIN_N to asynchronously stop emulation during the debugging session. This signal should be connected to a 10-K pull-up resistor so that normal operation is assured if no debugger is present.
- Do not connect the output RSTIN_N of the Tantino and the output of the on-board reset logic directly if the on-board reset logic does not feature an open drain output. If the on-board logic features a push-pull output, connect the RSTIN_N signal of the Tantino to an input of the on-board reset logic instead.
- Some of the signals listed in the table are optional. Do not connect these signals to the debug connector if the appropriate pins are used for an alternate functions. If the use of the pin might change during development, use a jumper to disconnect the signals if necessary.

7.12 Using HiTOP

The HiTOP debugger installation includes several examples which might be used as an easy start and demonstration for gathering first experiences with it. These examples can be started from within HiTOP by just opening the appropriate HiTOP project file (extension **.htp**). A HiTOP project file saves the setup for a debugging session, including the current window layout, the watch- and breakpoints and the user's project settings for HiTOP. The hereby most important and essential project settings are:

- Selection of used derivative
- Definition of name and location of the user application(s) together with the used compiler
- Definition of FLASH configurations (if application runs from FLASH)

In order to create a HiTOP project for your application two methods are recommended:

- Create a new HiTOP project (Project New...): this can be done step by step assisted by the HiTOP project wizard or by just using a predefined project template.
- Load and modify an already existing project file: if a project file already has been set up for an existing target hardware this method offers also a quick solution. Just copy and rename the existing file to the new project directory and usually just change the name of the application to be loaded.

The first example you try should always be the “**clock_xc**” example, which Hitex always deliver in its installation for the most commonly used starter kits. This application is tailored to run with an evaluation license and mostly uses only a common set of internal chip resources, thus being able to run on almost every derivative. This example executes the startup and hardware initialization for a 5ms timer interrupt that just counts a clock and displays this clock in the watch window. The example code in the main loop executes some other functions just to demonstrate some of the usual debugger features provided by HiTOP.

As HiTOP also supports the Infineon DAS interface it can be used with both the DAS servers. The “JTAG over USB Box” server is required and used for accessing the target via the TantinoXC or the OEM USB-Wiggler Box provided by Infineon. The “JTAG over USB Chip” server is used for the Infineon MiniWiggler or Evaluation boards already equipped with an on-board USB to JTAG converter from FTDI, which will be true for upcoming EasyKits provided from Infineon via the Hitex WebShop.

7.13 Extended Features

The Hitex TantinoXC fully supports the XC16x and the XC2000 architecture like the Infineon USB-Wiggler just because the TantinoXC is a superset of this hardware. The additional benefits of the TantinoXC hardware are based on an extra FPGA which yields the following advantages:

- JTAG acceleration with frequencies up to 20 MHz (target dependent)
- Automatic sense and adaptation to the target voltage level in the range of 1.8V, 2.5V, 3.3V and 5V
- Follow target frequency option for JTAG frequency
- Support for hardware break request (BRKIN) during idle/sleep condition
- Support of interrupts during halted state
- Simple trace based on Infineon OCDS triggered transfer
- Extended program and data flow trace based on smart instrumentation
- Execution profile based on periodically sampling CS:IP
- Time Analyser and Performance Analyser based on smart instrumentation

A short description of these features is given below. Additional application notes are available from Hitex.

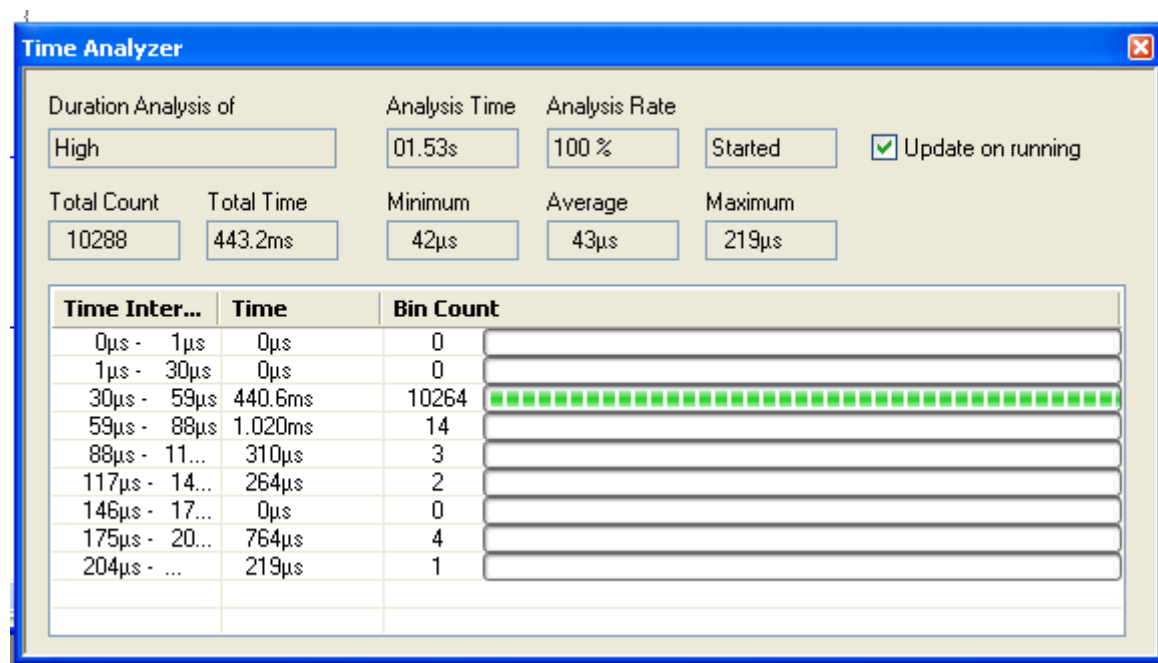
Time Analyser, Extended trace and Performance Analysis all require that the user dedicates one general purpose IO pin to the debugger to achieve a toggling signal used for generating the necessary time stamp pulses. This signal must be connected to Pin10 (BRKOUT) of the debug connector. At least to toggle this port pin, a smart instrumentation is always necessary to access these features.

The simple trace feature is restricted to generate only one trace event and thus does not require any instrumentation. Instead it uses the OCDS feature to trigger the BRKOUT pin when the selected debug event occurs. Thus using this feature requires connection of the device’ BRKOUT pin to pin10 (BRKOUT) of the debug connector. As the BRKOUT is for all XC2000 derivatives an alternate function of a general purpose port, this port pin could also

be used for the previously described extended trace features. Therefore it is recommended to connect either P10.11, P6.0, P1.5 or P9.3 (only available for 144 pin derivatives) as BRKOUT signal to pin 10 at the debug connector. This allows the use of both the simple and extended trace features without changing the hardware.

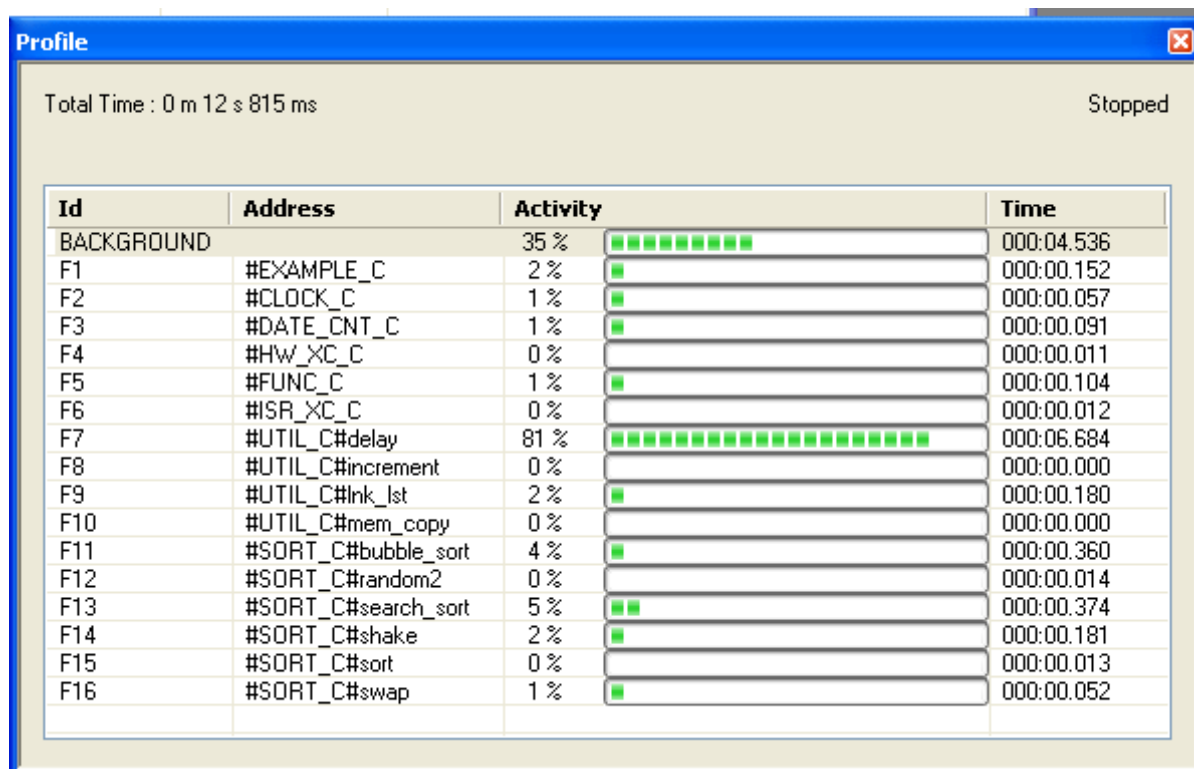
7.13.1 Time Analyser

The Time Analyser uses one port pin and requires a smart instrumentation to toggle this pin at the place at which a time measurement is required. Only one event can be measured and the measurement start signal is given by setting the port high. The measurement stop signal is given by setting the port to low. The TantinoXC hardware will sample this port and will count the occurrences and measure the high, the low and the period time for each signal toggle. HiTOP then offers the display of event count, minimum, maximum and average time of the selected signal phase (low, high, period). The measured values are additionally compared against given ranges and will be sorted as they match these ranges. This results in the following display:

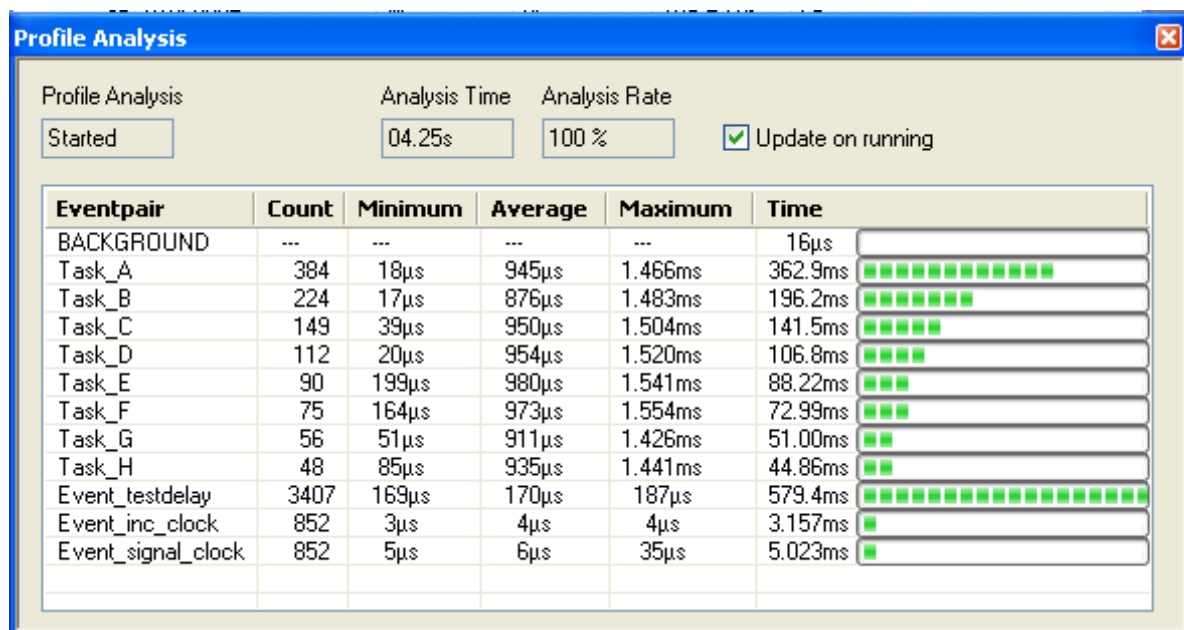


7.13.2 Execution Profile / Performance Analysis

Two methods are provided to measure the distribution of processor activities. The first method is called execution profiling and uses a periodic sample of the CS:IP value. Based on a sampling period of 1m,s this method will very quickly give (typically after some seconds) a relatively stable view of the distribution of processor activity. This view may easily be used to find out those program regions where a more detailed analysis makes sense and little optimization could lead to a maximum degree improvement. The advantage of this method is that it does not require any instrumentation and the observer objects can be changed without recompiling and even during a running system. The disadvantage of this method is that the result is only based on a statistical approach. Therefore this method is suitable for optimizing frequently called regions. It does not appropriate though for optimizing time-critical but very seldom executed ranges of code. A typical view of a measurement looks like this:



The second method supported by the TantinoXC is called Performance Analysis and this method should be used if the regions being investigated are already known. Typically the user wants to measure the execution time of a set of dedicated functions or the time consumed by a known task. This method uses the Time Analyser's port pin to toggle to signal a measurement point which is detected and stored as timestamp in the TantinoXC. As the time stamp signal does not bear any information on the location being measured, this is done additionally in a small trace event buffer located in the target system. This trace event buffer just remembers and identifies the occurred measurement points. As soon as the TantinoXC hardware has detected by counting the timestamps that the target's trace event buffer is filled to a certain level, it starts a quick OCDS read out of these event identifiers and matches them to the already-recorded timestamps. This permits the measurement and display for example of the following displayed scenario.



In order to achieve this measurement, the user has to include predefined function calls to a trace library provided by Hitex. This library call toggles the timestamp port and saves the trace event in the user-provided trace event buffer.

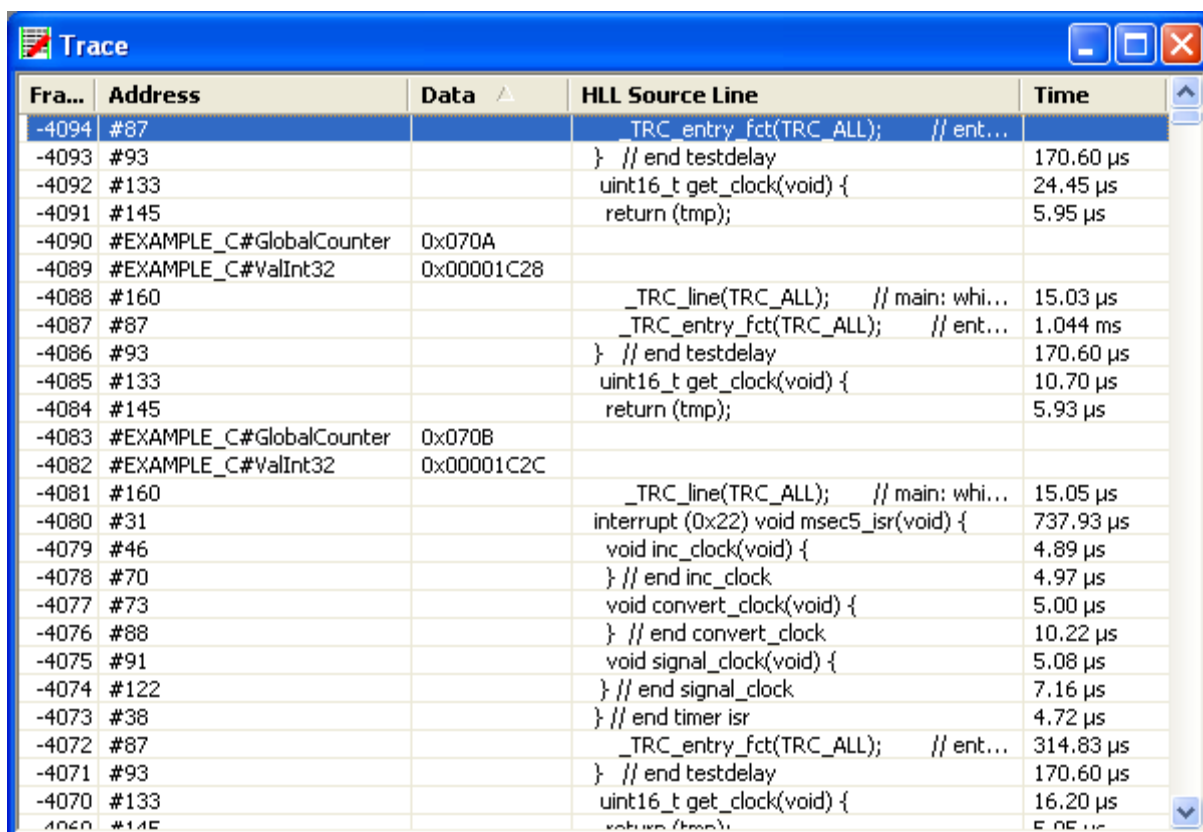
7.13.3 Trace Features

The possibility of transferring a single data value via JTAG based on an OCDS event (called triggered transfer) is used to realise the simple trace. Typically the user observes the write to a dedicated variable and the trace feature will transfer these written values to the external debugger via a JTAG transfer. The TantinoXC additionally records the timestamp when this transfer happened. This method does not require the use of instrumentation and is restricted to recording a single event (R/W data or executing a dedicated address).

The more powerful method is similar to the method used for gathering Performance analysis data, except that the data is not used for continuously calculating times but for displaying a program or data flow trace. Here we also need the instrumentation for the timestamp generation and the possibility of being able to identify different events, which is also done by the Hitex-supplied trace library.

As a special feature, the Tasking compilers offers the automatic generation of calls to these instrumentation libraries for function entry/exit or block entry thus reducing the overhead for the user in order to get a detailed program flow trace for selected program modules.

A typical view of a program and data trace is shown by this illustration:



Fra...	Address	Data	HLL Source Line	Time
-4094	#87		_TRC_entry_fct(TRC_ALL); // ent...	
-4093	#93		} // end testdelay	170.60 µs
-4092	#133		uint16_t get_clock(void) {	24.45 µs
-4091	#145		return (tmp);	5.95 µs
-4090	#EXAMPLE_C#GlobalCounter	0x070A		
-4089	#EXAMPLE_C#ValInt32	0x00001C28		
-4088	#160		_TRC_line(TRC_ALL); // main: whi...	15.03 µs
-4087	#87		_TRC_entry_fct(TRC_ALL); // ent...	1.044 ms
-4086	#93		} // end testdelay	170.60 µs
-4085	#133		uint16_t get_clock(void) {	10.70 µs
-4084	#145		return (tmp);	5.93 µs
-4083	#EXAMPLE_C#GlobalCounter	0x070B		
-4082	#EXAMPLE_C#ValInt32	0x00001C2C		
-4081	#160		_TRC_line(TRC_ALL); // main: whi...	15.05 µs
-4080	#31		interrupt (0x22) void msec5_isr(void) {	737.93 µs
-4079	#46		void inc_clock(void) {	4.89 µs
-4078	#70		} // end inc_clock	4.97 µs
-4077	#73		void convert_clock(void) {	5.00 µs
-4076	#88		} // end convert_clock	10.22 µs
-4075	#91		void signal_clock(void) {	5.08 µs
-4074	#122		} // end signal_clock	7.16 µs
-4073	#38		} // end timer_isr	4.72 µs
-4072	#87		_TRC_entry_fct(TRC_ALL); // ent...	314.83 µs
-4071	#93		} // end testdelay	170.60 µs
-4070	#133		uint16_t get_clock(void) {	16.20 µs
-4069	#145		return (tmp);	5.95 µs

7.14 Summary

By the end of this section you should be able to setup a project with HiTOP and Tasking VX, select the compiler options and the variant you want to use, configure the startup code and write C functions to handle exceptions.

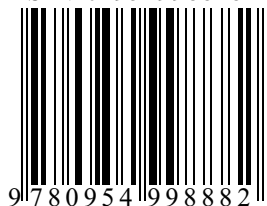
This guide contains important information that is useful when doing your first XC2200 family design or when evaluating it for a new project. The XC2200 series has many advanced power-saving features and special hardware facilities for supporting high-integrity software applications at SIL3/4. There are many simple facts which, if they are known at the outset, can save a lot of time and money and greatly improve system reliability.

Overall, it is intended to complement the user manuals by putting things into a practical context. Some of the material can be found in the XC2200 family databooks but most of it is simply the result of our practical experience and so is only to be found here.

www.hitex.com

hitex 
DEVELOPMENT TOOLS

ISBN 0-9549988-8-X



9 780954 998882

Hitex (UK) Ltd., Sir William Lyons Road, Science Park, Coventry, UK, CV4 7EZ.
Tel +44 (0) 2476 692066

