XInC2 Users Guide
Version 1.5

Created by
Phillip Jacobsen

Last Modified
October 22, 2008

10150 - 100 Street, Suite 900
Edmonton, Alberta, Canada, T5J 0P6
Phone: (780) 425-6511
Fax: (780) 425-7006
Inquiries: support@eleveneng.com
web: www.elevenengineering.com

XInC2 Summary

XInC2 (pronounced “zinc”) is a cost-effective and flexible multithreaded processor. XInC2 has a low-power, RISC processor architecture supporting an extensive peripheral library. The multithreaded, pipelined design of the XInC2 processor core executes concurrent, real-time programs with guaranteed performance, giving it a marked advantage over conventional serial processor implementations. The multithreaded programming model simplifies software development and testing, significantly reducing time to market. Firmware is also more intuitive to understand, enabling easy maintenance. The XInC2 hardware approach to real-time event handling eliminates the need for a real-time operating system (RTOS) scheduling kernel.

XInC2 is a 16-bit pipelined RISC processor with 8 hardware threads. The 8 threads behave as 8 independent processors, each with access to main memory and the peripheral bus. With XInC2 the disadvantages of serial interrupt-based processors such as context swapping, task scheduling, unpredictable execution times, and RTOS overheads are avoided. The 8 thread processors share hardware resources with the exception of each thread’s dedicated register set. Thus for the hardware cost of one processor, XInC2 provides 8. This approach results in outstanding MIPS/gate efficiency.

Each hardware thread is scheduled to execute at 1/8 of the system clock, thus removing the overhead of an RTOS. Firmware can be written as 8 independent programs. Each program runs on its own thread processor. Using simple interface conventions, programs on individual threads execute independently from each other. This allows for easy integration of firmware from different providers or team members.

Features The XInC2 processor include the following features:

- Independent execution of threads allows easy delineation and allocation of available processor cycles to unrelated applications.
- One or more threads can execute operations on a common data set
- Each thread has independent access to the peripheral bus.
- Firmware can be designed to assign one thread to service one or more Input/Output peripherals.
- Guaranteed response time to real-time events is realized via the elimination of interrupts and the division of task among the thread processors.
- A hardware peripheral library is provided that contains many common peripherals.
- Mathematical functions including multiplication, population count, bit reverse, and pack operations.
- Hardware semaphore mechanism for shared resource management.
- Inter-thread communication through a shared memory structure.

Target Applications XInC2 is well suited for many potential applications such as:

- Short-haul wireless applications using Eleven Engineering’s SPIKE technology.
- Bluetooth, 802.11, and ZIGBEE baseband processor applications.
- Cordless phones
- Consumer electronics - digital cameras, handheld computers, game controllers, MP3 players, smart cards
- Office Products - printers, fax machines
- Net infrastructure - USB and Ethernet hubs, Voice over IP telephones
- General purpose embedded processors for applications having multiple threads e.g. audio compression, protocol conversion, security and home control, industrial control, automotive sensors and actuators, and motor control.

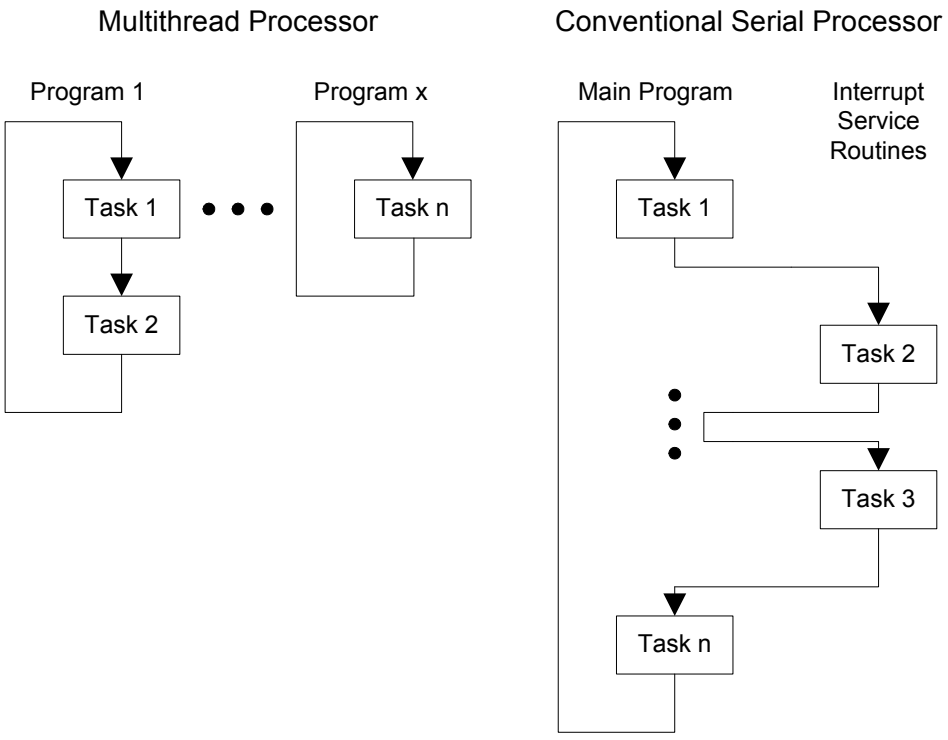
0100 Introduction / Description

Multithread versus Serial Microcontrollers

Typically focusing on input/output intensive applications, microcontrollers need to execute multiple tasks in a near simultaneous fashion. XInC2 and conventional serial microcontrollers take different approaches to solving this problem. Multithreaded processors divide applications into tasks that can run independently on separate processor threads. Conventional serial processors divide the execution time of the processor core through the use of interrupts.

Conventional serial processors handle time-critical, near-concurrent or priority tasks by interrupting(stopping) the current processor program and handing control over to an interrupt service routine. These interrupt service routines each perform one or more of the tasks required to support a given application. Each interrupt and hence each interrupt service routine has a different priority level. The processor allows higher priority interrupt routines to pre-empt lower priority interrupt routines. The most time-critical routines get the highest priority.

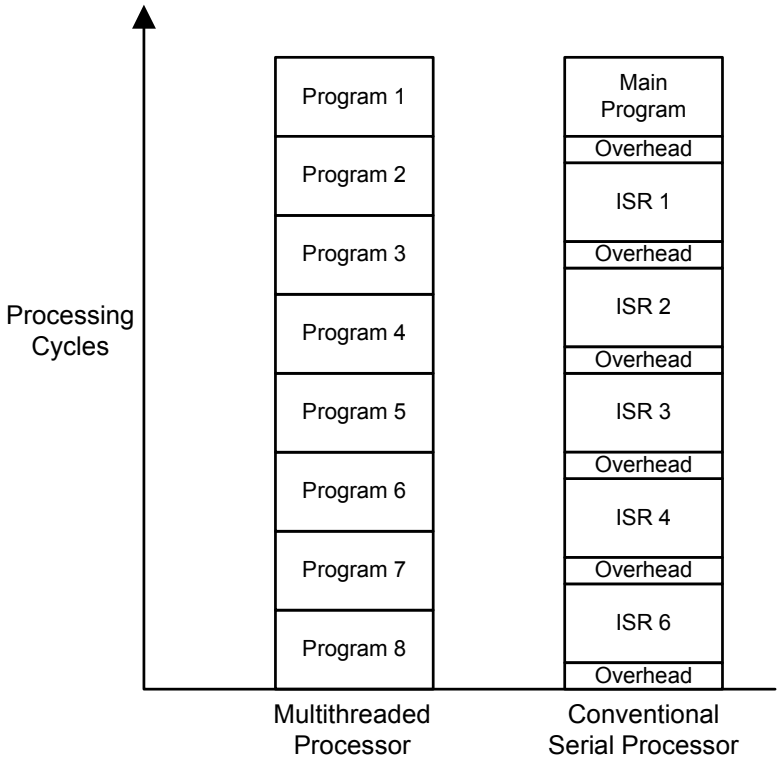
Conventional serial processors have inherent context switching overhead and verification issues. The diagram below illustrates the operational differences between XInC2 and conventional serial processors when executing multiple tasks. The tasks performed by XInC2 are simply divided concurrently into different threads. Separate tasks are performed by the conventional serial processor in a sequential manner as part of the main program (Task 1 and Task n in the diagram) or in interrupt service routine programs (Task 2 and Task 3). Each program segment has a certain percentage of overhead associated with it. This overhead comes from the context switching of program data that occurs when an interrupt happens and the previous program segment execution is temporarily halted. Context switching thus reduces the effective processing capacity of conventional serial processors when servicing inherently concurrent input/output tasks.



The XInC2 processor has other advantages. Whereas time critical programs can be designed to execute independently in multithreaded processors, overall reliable program execution is often difficult to verify for conventional serial processors. As the number of interrupt levels increase it becomes complex and difficult to verify that multiple interrupt service routines execute reliably. Unforeseen input conditions, such as surges in data traffic, or other system conditions, such as delayed responses from peripherals or error conditions, can cause one interrupt service routine to prevent the successful execution of another interrupt service routine in a timely fashion. These kinds of conditions cause the system to fail. Such situations can be somewhat mitigated by the costly addition of real time operating systems and over-provisioning processor capacity, but this may not reliably resolve all system problems. The developer thus ends up spending time optimizing resource management rather than implementing functions for the target application.

Unlike the conventional serial processor, XInC2 simply and reliably supports the concurrent, input/output processing required for microcontroller applications. Simple looping programs, each running on separate processor hardware threads, perform one or more of the tasks required to support an application. These looping programs execute in parallel on different XInC2 thread processors. The execution time of a given looping program is not impacted by the execution times of other programs since no context switching is required. This also decouples the timing dependencies between programs and the tasks performed by these programs. Not only does this greatly simplifies the verification of reliable program execution, but it also eliminates the need for a conventional real time operating system scheduler kernel as there are no interrupts to manage.

XInC2 programs, carefully allocated across the hardware threads of the XInC2 processor, can perform more work since they are not burdened with context switching overhead. The increased overhead of conventional serial processors versus the XInC2 processor can be seen in the following diagram which represents the total processing cycles over a given time period. The context switching processing time for the conventional serial processor is essentially lost processing time.



XInC2 has further benefits for applications having parallel or parallel and serial program structures. Certain algorithms such as audio and graphic compression or multithreaded operating systems such as Java can benefit from the multithreaded architecture. Each thread of the XInC2 processor can be configured to operate either independently or in a coordinated fashion. The XInC2 processor can be configured to execute a single instruction stream on multiple data streams (SIMD) or multiple instruction streams on multiple data streams (MIMD). One or more threads can operate in either SIMD or MIMD mode simultaneously. Sets of programs running one or more XInC2 threads can be optimized to operate in a fashion that best suits the nature of the target application or applications.

0200 XInC2 System Overview

MTA Processor

The XInC2 microcontroller has a multithreaded architecture (MTA) with a shared memory model. The programming model for the MTA is equivalent to a symmetric multiprocessor (SMP) with 8 heads, however the hardware complexity for the MTA is comparable to that of a single, conventional microcontroller. Only the register set is replicated. The balance of the hardware cost for the MTA processor is shared by all 8 heads. This approach results in outstanding MIPS/gate efficiency.

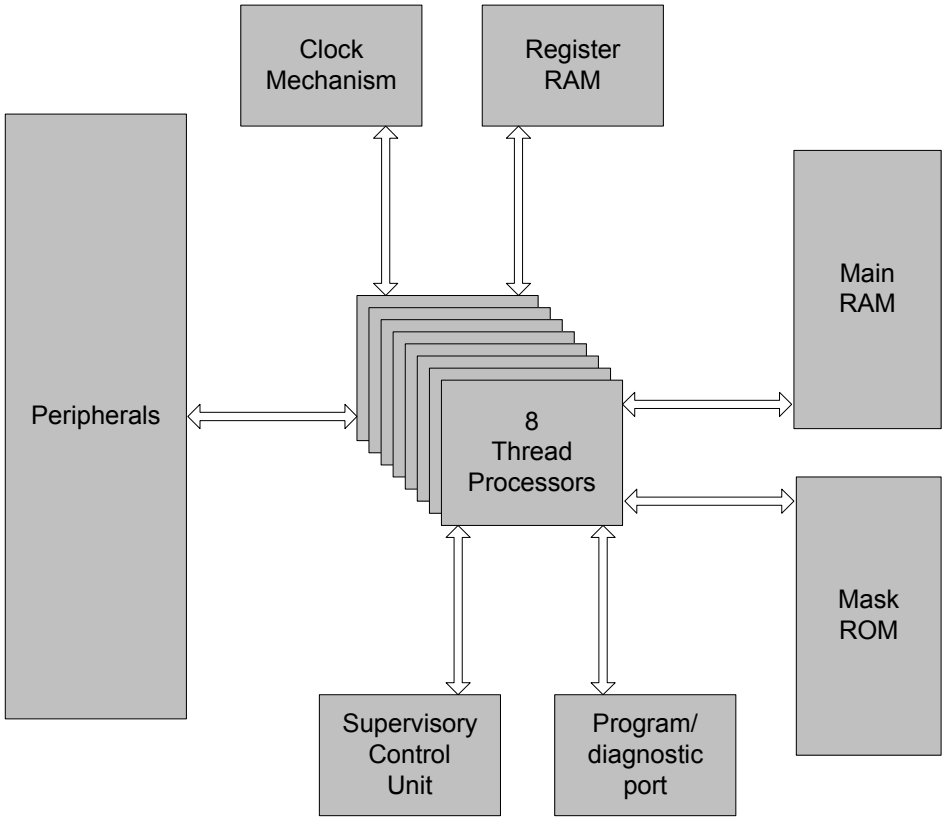
Distributed Processing Architecture

XInC2 is designed to perform multiple, concurrent I/O operations over both hardware and firmware based peripherals. As such, XInC2 is seen as a micro-mainframe architecture.

Multithreaded firmware guarantees that the real-time requirements of all peripheral devices are met concurrently. Note that a hardware thread which is dedicated to peripheral support, is equivalent to an I/O processor within a more general mainframe architecture.

Key XInC2 Components

As shown in the diagram below, XInC2 consists of an eight-thread processor core, a System clock distribution mechanism, internal memory components shown as register RAM, mask ROM, and main RAM; a supervisory control unit ("SCU"), a peripheral adaptor for reading to and writing from peripherals, and a program/diagnostic port for system programming and debug/test.



The XInC2 processor core is multithreaded, having an eight-stage pipeline capable of executing eight concurrent program threads simultaneously. The processor is described in more detail in the next section.

Various types of memory operate with the processor core. A register RAM module comprising eight sets of eight words is used for registers R0 to R7 for each of the eight processor threads. A mask ROM memory stores the boot loader and system routines. Main RAM is used as a system data store and program memory. Program code, data tables, temporary variables, and other modifiable parameters and system data are stored there in segmented program/data blocks. In the future as an alternative to main RAM, flash memory is planned to contain executable program code. In the base ASIC configuration, ROM is configured with 16k words and main RAM memory with 16k words.

0300 MTA Processor

Multi-Thread Architecture

XInC2 employs synchronous pipelining techniques to efficiently process multiple threads concurrently. A single 16-bit instruction is executed in an 8-stage pipeline. XInC2 also supports 32-bit instructions, i.e. 2-word instruction formats. Therefore a 2-word instruction requires 2 passes through the pipeline to process. In general, each thread processes 1 word of instruction stream per 8 ticks of the processor system clock. The timing model for the thread processor is therefore very simple. This simple timing model allows the firmware programmer to very easily calculate the timing of routines.

Private Thread State

The private state {PC,CC,R0:R7} of each thread processor includes.

- 16-bit program counter (PC) register.
- 4-bit condition code (CC) register, with bits named n,z,v, and c.
- 8 16-bit general purpose registers (R0:R7).

The XInC2 thread processor is a pure 16-bit machine. The program counter (PC) register and the 8 general purpose registers (R0:R7) are all 16 bits in size. The smallest addressable unit of main RAM is the 16-bit word, and the address word is also 16 bits wide.

Operand access in XInC2 is summarized as follows:

- XInC2 provides uniform access to all 8 general purpose registers, i.e. no individual general purpose register has any special properties
- Some instruction formats specify immediate operands
- Memory operands are accessed via load and store instructions
- For classification purposes, targets of branch instructions are treated as memory operands
- The only implicit operand is the condition code (CC) register

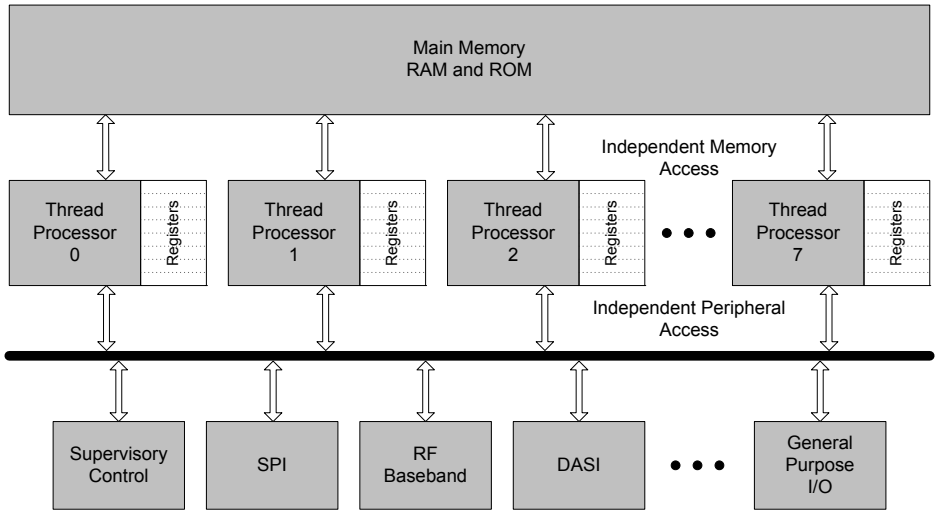
Shared Resources

All 8 thread processors share access to the RAM, and to the full peripheral set. Generally speaking, threads communicate with one another through the RAM.

0400 XInC2 Software Architecture

The XInC2 processor is organized into 8 independent thread processors which, from a firmware perspective, function independently. System software is distributed across 8 different program threads. Processing resource allocation is maintained simply through the hardware structure of XInC2. Using this approach the programmer or programming team can divide program segments among the threads so that sufficient time is available to complete each thread's program in a defined period of time. This segmentation of threads simplifies the allocation and integration of programs when teams of developers are working on a given project. Errant code segments in one thread can be isolated from impacting other threads.

System resources are shared. Main memory is shared among the 8 thread processors. Private memory segments such as stacks, buffers and variables can be allocated to each thread through programming conventions and semaphores. Data is shared between threads using common memory addresses. Peripherals access is shared. The following diagram illustrates the XInC2 programming model:



Flynn's Taxonomy Flynn first classified high performance computers based on the number of streams of instructions and data. He proposed four main architectural classes:

SISD Single instruction stream, Single data stream. These are conventional systems that contain one CPU and can accommodate one instruction stream that is executed serially.

SIMD Single instruction stream, Multiple data streams. The same instruction is executed by multiple threads operating on different data streams in lock-step. A single instruction manipulates many data items in parallel.

MISD Multiple instruction streams, Single data stream. There are no commercial machines of this type.

MIMD Multiple instruction stream, Multiple data streams. Each thread has its own instruction, and operates on its own set of data.

The XInC2 multithreaded architecture supports both SIMD and MIMD operation.

0401 Machine Language

The XInC2 RISC-based machine language consists of 18 instructions, and a total of 6 address modes. XInC2 supports 26 instruction/address-mode combinations. The instruction set is designed for efficient decode and high code density. Instructions are either 1 word or 2 words in length.

XInC2 defines a minimal set of 18 instructions, which provide:

- Efficient control flow for loops, conditionals, and subroutine calls
- Fundamental bitwise operations
- Elementary arithmetic
- Load/store access to main RAM
- I/O operations

The XInC2 instruction set architecture is summarized below:

Instruction	Description	Available Address Modes
bc	conditional branch	PC relative
bra	unconditional branch	PC relative
jsr	jump to subroutine	register indirect, absolute
thrd	get thread number	register
ld	load from RAM	base displacement, absolute
st	store to RAM	base displacement, absolute
inp	read input port	immediate
outp	write output port	immediate
mov	move immediate	immediate
add	2's complement add	register, immediate
sub	2's complement subtract	register
rol	bitwise rotate left	register, immediate
and	bitwise and	register, immediate
ior	bitwise inclusive or	register, immediate
xor	bitwise exclusive or	register, immediate
bic	bit clear	immediate
bis	bit set	immediate
bix	bit change	immediate

As is true for every machine language, XInC2 instructions are used in combination to construct higher-level operations. For example, the bitwise rotate left instruction, combined with the bit clear instruction, gives a shift right operation.

XInC2 illustrates the principle that an instruction set should be as simple as possible, and no simpler. Only the most frequent and the most elementary operations require direct hardware implementation.

Condition Codes

Most XInC2 machine instructions generate a 4-bit condition code, which may be passed through the condition code (CC) register, to be interpreted by subsequent conditional branch instructions.

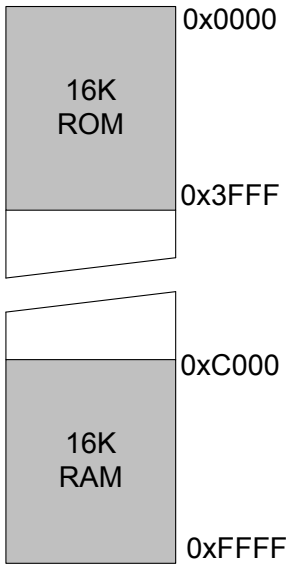
The n, z, v, and c bits of the condition code (CC) register have different interpretations, depending on the instruction that produced them. For the arithmetic operations, add and subtract, the CC bits respectively mean negative, zero, overflow, and carry. For other operations, the c bit means character, i.e. the result is in the interval [1:255]. The v bit has varying interpretations, usually indicating that the result is odd.

Register Set Each thread processor has its own, private set of 8 16-bit general purpose registers(R0:R7). XInC2 machine instructions can specify up to 3 register operands. Register operands may be specified in any combination.

While there is 8-way processing overlap in the XInC2 pipeline, each individual thread processor executes instructions serially. Within each thread, one instruction is executed to completion, and then the next instruction is executed, and so on. Results written to a register by one instruction, are available as source operands to a subsequent instruction.

0500 Memory Configuration

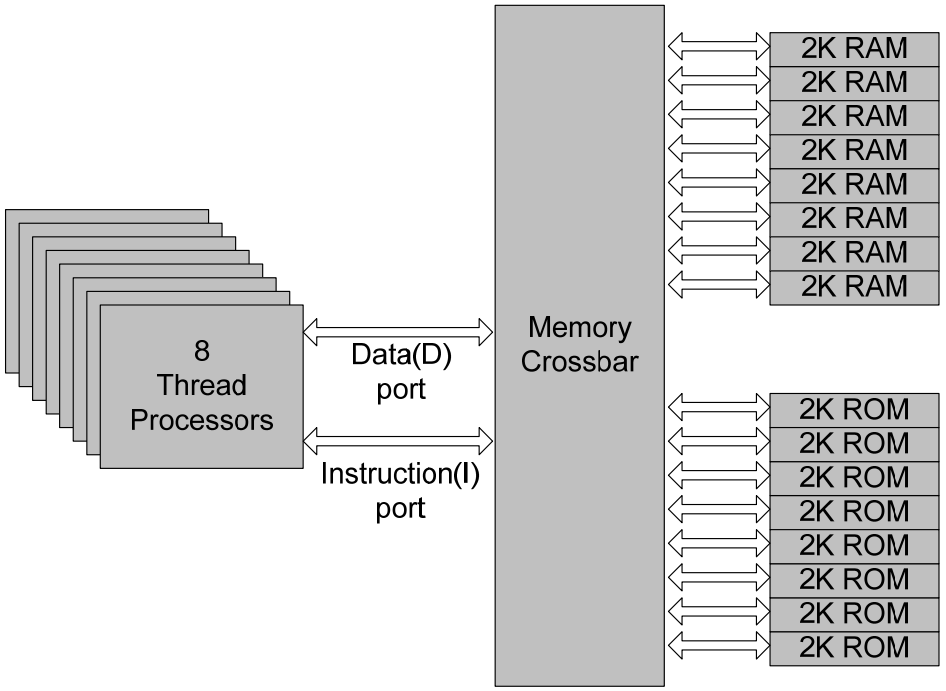
XInC2 Memory Model The 16 bit architecture allows for 64K words of addressable memory space. Physically, XInC2 has 16K words of mask ROM, and 16K words of RAM. Future expansion is possible. The physical memory map is shown below:



Internal Mask ROM The 16K words of internal Mask ROM begins at address 0x0000. The mask ROM contains the boot loader as well as some common routines that can be called by a programmer. This code area is normally reserved, but for high volume / low cost applications, portions of the ROM may be masked programmed with the customer application program during the fabrication of the microcontroller.

RAM The 16K words of RAM is used for program and data space. The RAM is organized into 8 blocks of 2k words each. This organization has two benefits. This architecture allows for data and code to placed in physically separate memories.
The second benefit is for power savings. The power consumption of memory can be significant. Therefore by only enabling small blocks of memory when it is accessed, allows for more optimal power usage.

Memory Crossbar The interface between the thread processors and the RAM/ROM is shown in the diagram below. The memory crossbar is responsible for routing the data to addressed memory blocks.



The thread processor hardware has 2 ports that are connected to the memory crossbar. The Instruction(I) port is used to fetch instructions. Every system clock(Sclk) cycle, the thread processor hardware can fetch one word of an instruction. If not all threads are running then some cycles will not contain any instruction fetches.
Every system clock(Sclk) cycle, the thread processor hardware may also read from memory or write to memory via ld/st instructions.
Therefore every Sclk cycle it is possible for two memory accesses to be requested from the thread processor; one from the I-port and one from the D-port.

Every Sclk cycle each block of memory(RAM or ROM) may have one memory access from either the I-port or the D-port. In the case of conflict, the I-port is given priority and the data read from the D-port will be invalid and a write will not occur.

Programming Rule In practice this conflict can be easily avoided. Data and Code in RAM/ROM should exist in separate 2K blocks. This will prevent both the D-port and I-port from being accessed at the same time on the same block of memory. If only one thread is running, it is acceptable for code and data to share the same memory block because one thread does not access the I-port and D-port in the same cycle.

Short Address Space The Format 1 LD and ST instructions allow for efficient addressing of 128 words at the end of RAM and 127 words at the beginning of the mask ROM.

0600 Clock Mechanism

The XInC2 processor core is a synchronous single clock design. All of the peripherals except for the Digital Audio Serial Interface are also synchronous to the processor core.
The System clock can be sourced from 3 different inputs. XInC2 includes 1 onboard oscillator, High frequency PLL, and an input designed for a low frequency external RC oscillator circuit. See the Supervisory Control Extensions peripheral(SCX) for more information on the clocking mechanism.

0700 System Reset and Initialization

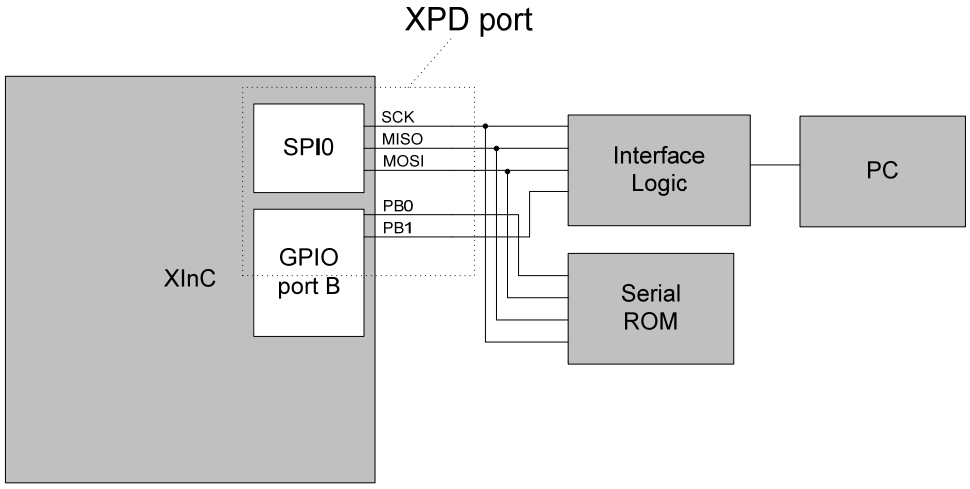
The XInC2 system has four sources of reset: External reset, Power-On reset delay, brownout detection, and a software initiated reset.

External Reset XInC2 is reset when a low level is present on the RESET pin for more than 2 System clock(Sclk) cycles.

Software Reset A software reset can be initiated by any thread calling the SoftReset routine.

0800 XInC2 Programming / Debug Port (XPD port)

The XPD port is the physical interface that Program code is loaded into XInC2 for execution and it also provides a path for debugging during code development and also as a diagnostic port in the field. The XPD port functions are accomplished by using a synchronous serial port(SPI0) and some general purpose I/O pins. The XPD port is diagramed below:



Programming XInC2 The XPD port provides an interface to the XInC2 platform and a PC running the XInC2 Development Environment (XDE). The XPD allows the programmer to download code into RAM.

Debugging XInC2 Once the firmware is executing, the XPD port can be used with XInC2 development Environment(XDE) for firmware debugging.

Real Time Diagnostic The XPD port can also be used for diagnostics in the field. It provides a convenient interface to a PC or other display device.

0900 BIOS

The BIOS resides in the mask ROM. The bios services include:

- System initialization
- Boot loader
- External EEPROM write routines
- Miscellaneous routines
- Diagnostic codes

0901 BIOS Subroutines

The BIOS contains routines that may be called by the firmware developer. The name and address of the available routines is shown below:

Name	Address
ShowTerminationCode	0x0006
ProgramEEPROM	0x0012
ProgramSPIEEPROM	0x0014
ProgramI2CEEPROM	0x0016
RunT0	0x0C7E
RunT1	0x0C9B
RunSIMD	0x0CB8

ShowTerminationCode

ShowTerminationCode is used by the bootloader to indicate error codes that can be used to debug problems. This routine can also be used by the developer to indicate application codes. This routine can be called from any thread but stops all other threads that are running. If the XPD module is connected, then the 16-bit hex error code will be sent once and then the 16-bit error code is transmitted indefinitely on the SPI0 bus. The error code can be read from a PC with a terminal program running or using an oscilloscope to decode the data using the SCK0 and MOSI0 pins.

The BIOS Termination Codes are shown below for reference. These error codes are shown when the corresponding routine fails.

Routine	Code
XXprogramEEPROM	0x0007
XXchecksum	0x0008
XXuartLoader	0x0009
XXxromLoader	0x000A
XXrunT0	0x000D
XXrunT1	0x000E
XXrunSIMD	0x000F
XXprogramI2CEEPROM	0x0011
XXi2cFailure	0x0013

ProgramSPIEEPROM ProgramSPIEEPROM must be called from Thread 0 and never returns. It checks for the following code sequence in RAM:

```
@ = 0xC000
                bra 0x0014      // programEEPROM vector
                0x4A55          // SPI0div/clkCfg word (example)
```

and transforms it into:

```
@ = 0xC000
                bra 0x0010      // checksum vector
                0x1234          // expected checksum (example)
```

The configuration word is shown below.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SCK2	SCK1	SCK0	PG1	PG0	SZ	CHK	UCK2	UCK1	UCK0	-	FEB_1	S1_1	S0_1	EB_1	E_1

- **Bit 15,14,13 - SCK2,SCK1,SCK0: SPI Clock Rate select for EEPROM**
  These three bits control the SPI0 clock(SCK) rate while reading from the EEPROM. See the SPI0 peripheral for more information.
- **Bit 12,11: PG1,PG0: EEPROM page size**
  These two bits control the write page size of the EEPROM.

PG	bytes
00	8
01	16
10	32
11	64

- **Bit 10: SZ: EEPROM addressing size**
  When this bit is zero then 16 bit addressing is used.
  When this bit is one then 24 bit addressing is used.
- **Bit 9: CHK: checksum disable**
  When this bit is zero then the program EEPROM call is transformed to a checksum routine that will verify the RAM contents after loading the code from EEPROM to RAM.
  When this bit is one then the program EEPROM call is transformed to a bra @+2 and the RAM contents is not verified.
- **Bit 8,7,6: UCK2,UCK1,UCK0: SPI Clock Rate select for uart**
  These three bits control the SPI0 clock(SCK) rate while reading from the UART. See the SPI0 peripheral for more information.
- **Bit 5: reserved**
  reserved
- **Bit 4,3,2,1,0: oscillator configuration**
  These five bits control the configuration of the oscillator.

The resulting RAM image is then written to external SPI EEPROM and SoftReset is called.

After SoftReset is called and the boot loader has finished loading the code image into RAM the first instruction that is executed is bra 0x0010. This is branch to a checksum routine that verifies that the image has been loaded successfully into RAM. If the checksum passes, program execution continues at 0xC000 + 3. If the checksum fails, the checksum termination code will be displayed on the XPD port.

RunT0 RunT0 can be called from any thread. It stops all threads except for Thread 0 and returns with Thread 0 running. (Thread 0 does not have to be running when RunT0 is called.). The routine requires the return address to be in R7.

RunT1 RunT1 can be called from any thread. It stops all threads except for Thread 1 and returns with Thread 1 running. (Thread 1 does not have to be running when RunT1 is called.). The routine requires the return address to be in R7.

RunSIMD RunSIMD may be called from Thread 1 only. The SIMD entry point of a routine is required in R7.

0902 Boot Loader

The main responsibility of the BIOS is to provide a boot loader. The boot loader functions in a manner very similar to the boot functions on a desktop PC. The loader attempts to load a program in sequence from several devices. If a load device fails due to a communication or checksum error, the associated termination code will be presented at the XPD port to aid in hardware diagnostics. If a load device is not present then the next load device in the sequence is accessed. Possible load devices include:

- development code download from a PC to XInC2 via the XDP port.
- external serial memory.
- program image in mask ROM.

The boot loader sequence is illustrated in the flow diagram on the following page.

Loader Sequence

The boot loader is executed when the following conditions occur:

- Power Up
- POR
- RESET pin is pulsed low
- SoftReset routine is executed

The boot loader begins execution with the System Clock(Sclk) source configured as the RCCLK. The default System clock source for developers is the output of Oscillator (CLK1). The boot loader speculatively configures the Oscillator with the following configuration: internal feedback resister enabled, fundamental mode, and 12-24 MHz range. External crystals can take a few milliseconds to become stable so the oscillator is enabled early to ensure that the oscillator is stable by the time that the loader switches the Sclk source to CLK1. The XPD port is now configured to enable communication to external devices. The clock divider configuration for SPI0 is Sclk/16.

The loader now attempts to load a Magic number from an external SPI interface ROM. The presence of the 16-bit Magic number is used by the boot loader to validate the existence of a code image. The absence of the Magic number indicates that an external memory is either not present or contains an invalid XInC2 code image. The loader is able to detect multiple types of memory addressing modes including 16 and 24 bit addressing. Contact Eleven Engineering for more information about supported memories.

If a Magic number is detected a second 16-bit word of data is read from the ROM. This word is a configuration word for SPI0 and Sclk source. This configuration allows the developer to configure the XPD port for a range of clock sources and speed.

The configuration word in the EEPROM is shown below.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SCK2	SCK1	SCK0	-	-	-	-	UCK2	UCK1	UCK0	-	FEB_1	S1_1	S0_1	EB_1	E_1

- **Bit 15,14,13 - SCK2,SCK1,SCK0: SPI Clock Rate select**
  These three bits control the SPI0 clock(SCK) rate while reading from the EEPROM. See the SPI0 peripheral for more information.

- **Bit 8,7,6: UCK2,UCK1,UCK0: SPI Clock Rate select**
  These three bits control the SPI0 clock(SCK) rate while reading from the UART. See the SPI0 peripheral for more information.

- **Bit 4,3,2,1,0: oscillator configuration**
  These five bits control the configuration of the oscillator.

If the configuration word is valid , the loader will configure SPI0 and the clock configuration register with the values retrieved from the ROM. If the configuration word is invalid or no Magic number is detected then SPI0 and the clock configuration register are setup with default values. The default values for SPI0 clock divider is Sclk/16. The default value for the oscillators was already setup prior to reading the Magic number. The loader will now switch the Sclk source to. The user must switch to Oscillator 1. Operating with only RCCLK is not valid.

The entire 16K words of RAM are now cleared.

The loader has now initialized the XInC2 chip for operation and begins a sequence to load program code into RAM for execution.

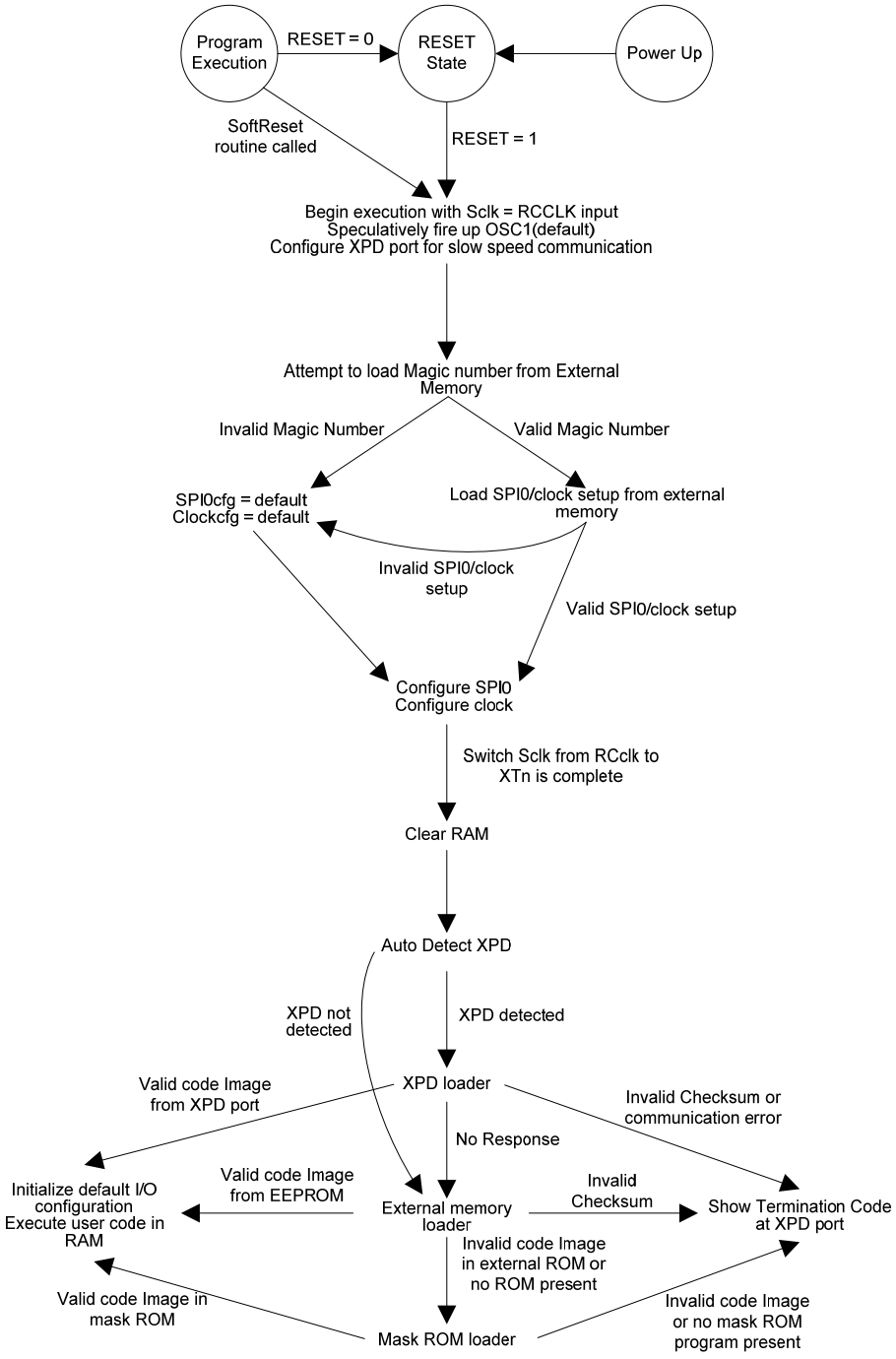
The first load device is the programming/development card attached to the XPD port. This card connects to a PC and allows in circuit programming of XInC2. Auto detection of the XPD is not performed. An internal pulldown on pin PB1 is enabled and the input state is read. If a '0' is read then no XPD is detected and this loader is skipped. If a '1' is read then the an attempt is made to look for a response from the PC to download firmware.

If there is no response from the PC the loader will attempt to load a firmware image from an external ROM. Once again the loader attempts to retrieve the Magic number from the external ROM.

If there is no Magic number the loader will search for a valid code image residing in the mask ROM. This is the last load device in the loader sequence. If no mask ROM image is available, the bootloader termination code will be executed.

After a successful load either from a SPI EEPROM or XPD download the I/O cells are initialized to the default configuration. See the SCX peripheral for details on the default state. Once the cells are initialized, execution at location 0xC000 with only thread0 running begins.

Boot loader Flow Diagram



1000 I/O Access

The XInC2 processor contains a bidirectional internal peripheral bus that allows the processor to send or receive data from peripherals using output and input instructions.

1001 I/O Paging

The instruction set allows for 7-bit addressing on the I/O bus. The XInC2 peripheral set requires more addresses then provided by the 7-bit addressing so an I/O paging mechanism is provided. The I/O paging is configurable for each thread and most of the peripherals are available on page0.

1002 Flow Control

The hardware peripheral interface provides a facility for peripherals to block access by the thread processors. If a thread issues an *inp* or *outp* instruction, and the peripheral is busy, it may assert a *wait* signal. Upon receipt of this wait signal, the thread processor continues executing through the pipeline and reissues the same instruction next instruction cycle as if the previous instruction had not been issued.
The use of the wait signal allows the firmware drivers to be very compact and responsive. The firmware routines do not have to spend time and code space polling a busy/done flag. For example the firmware required to initiate sending and receiving a byte to a device attached to the SPI port is:

```
outp    r0,SPI0tx       //start SPI transmit
inp     r0,SPI0rx       //get received byte
```

This peripheral interface is very well suited to a multi-threaded processor because often one thread is dedicated to interfacing to a specific I/O device. Many of the peripherals also provide registers that can be read that give status information without blocking the thread processor. This can be used if the programmer does not want the thread to be stalled.

1100 Critical Sections

The code segments within a program that access the same object from separate, concurrent threads are called *critical sections*.

A critical section is code that accesses a critical/shared resource. A shared resource could be a hardware peripheral, memory or combination.

Because all 8 threads share memory and peripheral resources, the programmer must be concerned with mutual exclusion and cooperating sequential processes. Care must be taken to ensure that no one thread disturbs the work of any other thread.

This approach can be likened to an office where the workers operate independently and in parallel except when they need to use shared office equipment or communicate with each other. A worker can talk to another worker only if the other worker is listening carefully. Also, a worker can't use a copy machine until it is free and in a finished state. Consider what would happen if a worker preempts another worker making copies halfway through a job.

Mutual Exclusion (Semaphores)

The problem of guarantying that a critical resource / region is accessed by only one thread at a time is accomplished through a semaphore mechanism. The semaphore mechanism is based on Edsger W. Dijkstra's ideas as discussed in "Cooperating Sequential Processes" , in Programming Languages, F. Genuys, editor, Academic Press 1968.
A semaphore is basically a switch that indicates whether it is safe to proceed.

When a thread wants to access a critical resource, it asserts a "down" flag to indicate that the resource is being used. When the thread is finished with the resource it asserts an "up" flag to indicate that the resource is released.

Consumer-Producer

Another common use of semaphores is in solving the "consumer-producer problem." This problem occurs when one thread is producing work for another thread to use. The producing thread must have some method of telling the consumer thread when the data is ready to be processed. The consumer thread initially blocks on the semaphore by issuing two "down flags" . When the producer is ready to pass the "work unit" on it releases the semaphore by issuing an "up flag". Once the consumer is finished processing the "work unit" it issues another "down flag" waiting for another unit. This consumer-producer relationship is often used when one thread is responsible for retrieving/sending real-time data via a hardware peripheral and passes/retrieves the data to another thread for processing.

Deadlock

Dijkstra also created the dining philosophers problem to illustrate problems that can occur with multithreaded systems. There are five philosophers sitting around a table. In front of them are five bowls and only five chopsticks. The philosophers are bound by the following rules:

- Each philosopher thinks for a while, eats for a while, and then waits for a while.
- To eat, the philosopher must hold both his right and left chopsticks.
- The philosophers communicate only be lifting and lowering of the chopsticks.(no talking allowed).

If all of the philosophers pick up their right chopstick, then no one can pick up their left one. If they all hold their right and wait for the left one to become available, no one will eat. This is a classic multithreading problem called deadlock . If some of them decide to put down their chopsticks then the others can eat, but that particular philosopher might never eat. This is called starvation. Another possibility is that all of the philosophers eat, but some might eat more than others. This is called lack of fairness.

Although difficult to detect and debug, by following a few simple rules, deadlock can be eliminated.

- Group multiple critical resources together under one semaphore. Using this protocol ensures that the system will never enter a deadlock state.

Deadlock detection The Supervisory Control unit provides a facility to detect deadlock.

Using semaphores in XInC2 See the Supervisory Control Unit section for more details on using the XInC2 semaphore mechanism.

1200 Supervisory Control Unit (SCU)

Introduction The thread processor architecture is augmented by a peripheral device known as the Supervisory Control Unit (SCU). The Supervisory functions include read/write access to the state of each thread processor, start/stop control of each thread, detection of unusual conditions (e.g. I/O lock ups/ tight loops), semaphore-based management of critical resources, a timer facility, and control of the IO page for each thread.

1201 Thread Run/Stop Control

The SCUstop register controls the run/stop state of a thread. SCUstop is a write only register. Setting STPn to one stops thread processor n. Setting STPn to zero starts thread processor n. When a thread is stopped it finishes executing the current instruction in the pipeline.

Note! There is no problem for a thread to turn itself off. However it is impossible for a thread to turn itself on. Therefore it is possible for all threads to be turned off with no mechanism to restart without reset.

SCU Thread Run Control SCUstop Address: 0x04, IOpage: 0/1
Access Mode: Write only

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	STP7	STP6	STP5	STP4	STP3	STP2	STP1	STP0

- **Bit 15..8 - Res: Reserved**
  These bits are reserved
- **Bit7 – STP7: Thread7 run/stop**
  When this bit is set, Thread7 is stopped. When this bit is cleared, Thread7 is running.
- **Bit6 – STP6: Thread6 run/stop**
  When this bit is set, Thread6 is stopped. When this bit is cleared, Thread6 is running.
- **Bit5 – STP5: Thread5 run/stop**
  When this bit is set, Thread5 is stopped. When this bit is cleared, Thread5 is running.
- **Bit4 – STP4: Thread4 run/stop**
  When this bit is set, Thread4 is stopped. When this bit is cleared, Thread4 is running.
- **Bit3 – STP3: Thread3 run/stop**
  When this bit is set, Thread3 is stopped. When this bit is cleared, Thread3 is running.
- **Bit2 – STP2: Thread2 run/stop**
  When this bit is set, Thread2 is stopped. When this bit is cleared, Thread2 is running.
- **Bit1 – STP1: Thread1 run/stop**
  When this bit is set, Thread1 is stopped. When this bit is cleared, Thread1is running.
- **Bit0 – STP0: Thread0 run/stop**
  When this bit is set, Thread0 is stopped. When this bit is cleared, Thread0 is running.

1202 IO Page Control

The large number of I/O peripheral addresses exceeds the 7-bit addressing range provided by the instruction set. An I/O paging mechanism programmable for each thread is provided to allow for 256 addresses.

The SCUiopageWR, and SCUiopageRD registers are used to control the page setup for each thread. SCUiopageWR is a write only register. Setting IOPn to one sets the page to 1 for thread processor n. Setting IOPn to zero sets the page to 0 for thread processor n. The SCUiopageRD register is a read only register and is used to read back the configuration that was previously written to the SCUiopageWR register.

SCU IO page Write SCUiopageWR

Address: 0x05, IOpage: 0/1
Access Mode: Write only

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	IOP7	IOP6	IOP5	IOP4	IOP3	IOP2	IOP1	IOP0

- **Bit 15..8 - Res: Reserved**
  These bits are reserved
- **Bit7 – IOP7: Thread7 I/O page**
  When this bit is set, Thread7 uses page 1. When this bit is cleared, Thread7 uses page 0.
- **Bit6 – IOP6: Thread6 I/O page**
  When this bit is set, Thread6 uses page 1. When this bit is cleared, Thread7 uses page 0.
- **Bit5 – IOP5: Thread5 I/O page**
  When this bit is set, Thread5 uses page 1. When this bit is cleared, Thread7 uses page 0.
- **Bit4 – IOP4: Thread4 I/O page**
  When this bit is set, Thread4 uses page 1. When this bit is cleared, Thread7 uses page 0.
- **Bit3 – IOP3: Thread3 I/O page**
  When this bit is set, Thread3 uses page 1. When this bit is cleared, Thread7 uses page 0.
- **Bit2 – IOP2: Thread2 I/O page**
  When this bit is set, Thread2 uses page 1. When this bit is cleared, Thread7 uses page 0.
- **Bit1 – IOP1: Thread1 I/O page**
  When this bit is set, Thread1 uses page 1. When this bit is cleared, Thread7 uses page 0.
- **Bit0 – IOP0: Thread0 I/O page**
  When this bit is set, Thread0 uses page 1. When this bit is cleared, Thread7 uses page 0.

SCU IO page Read Address: 0x05, IOpage: 0/1
SCUiopageRD Access Mode: Read only

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	IOP7	IOP6	IOP5	IOP4	IOP3	IOP2	IOP1	IOP0

- **Bit 15..8 - Res: Reserved**
  These bits are reserved
- **Bit7 – IOP7: Thread7 I/O page**
  When this bit is set, Thread7 uses page 1. When this bit is cleared, Thread7 uses page 0.
- **Bit6 – IOP6: Thread6 I/O page**
  When this bit is set, Thread6 uses page 1. When this bit is cleared, Thread7 uses page 0.
- **Bit5 – IOP5: Thread5 I/O page**
  When this bit is set, Thread5 uses page 1. When this bit is cleared, Thread7 uses page 0.
- **Bit4 – IOP4: Thread4 I/O page**
  When this bit is set, Thread4 uses page 1. When this bit is cleared, Thread7 uses page 0.
- **Bit3 – IOP3: Thread3 I/O page**
  When this bit is set, Thread3 uses page 1. When this bit is cleared, Thread7 uses page 0.
- **Bit2 – IOP2: Thread2 I/O page**
  When this bit is set, Thread2 uses page 1. When this bit is cleared, Thread7 uses page 0.
- **Bit1 – IOP1: Thread1 I/O page**
  When this bit is set, Thread1 uses page 1. When this bit is cleared, Thread7 uses page 0.
- **Bit0 – IOP0: Thread0 I/O page**
  When this bit is set, Thread0 uses page 1. When this bit is cleared, Thread7 uses page 0.

1203 Processor State Access

The SCUreg, SCUpc, SCUcc, and SCUpntr registers are used to read from and write to the instruction register set, program counter, and condition code registers of individual threads as addressed by the SCUpntr register. The procedure for reading or modifying a processor state is as follows:

1. Stop the thread processor for the thread of interest.
2. Setup the SCUpntr register:
    a. RP_2, RP_1, RP_0 bits of the SCUpntr register are used to address one of the 8 registers of a thread processor.
    b. TP_2, TP_1, TP_0 bits of the SCUpntr register are used to address one of the 8 thread processors.
3. Read/Write to SCUreg, SCUpc, or SCUcc to read/set the desired processor state.

Note! Care must be taken when reading SCUpc and SCUcc. After setting up the SCUpntr register, a single nop must be inserted before a read of the SCUpc and SCUcc registers is done. Reading or writing to SCUreg or writing to SCUpc or SCUcc can be done immediately after setting SCUpntr.

SCU register SCUreg Address: 0x00, IOpage: 0/1
Access Mode: Read/Write

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
REG15	REG14	REG13	REG12	REG11	REG10	REG9	REG8	REG7	REG6	REG5	REG4	REG3	REG2	REG1	REG0

- **Bit 15..0 - REG: Instruction Register**
  These bits represent the current value of the 16-bit instruction register as addressed by the SCUpntr register.

SCU Program Counter SCUpc Address: 0x01, IOpage: 0/1
Access Mode: Read/Write

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PC15	PC14	PC13	PC12	PC11	PC10	PC9	PC8	PC7	PC6	PC5	PC4	PC3	PC2	PC1	PC0

- **Bit 15..0 - PC: Program counter**
  These bits represent the current value of the 16-bit program counter for the thread addressed by the SCUpntr register.

SCU Condition Code Address: 0x02, IOpage: 0/1
SCUcc Access Mode: Read/Write

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	N	Z	V	C

- **Bit 15..4 - Res: Reserved**
  These bits are reserved.
- **Bit 3 - N: N Condition code**
  This bit is the 'N' condition code.
- **Bit 2 - Z: Z Condition code**
  This bit is the 'Z' condition code.
- **Bit 1 - V: V Condition code**
  This bit is the 'V' condition code.
- **Bit 0 - C: C Condition code**
  This bit is the 'C' condition code.

SCU Pointer Address: 0x03, IOpage: 0/1
SCUpntr Access Mode: Write only

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	TP 2	TP 1	TP 0	RP 2	RP 1	RP 0

- **Bit 15..6 - Res: Reserved**
  These bits are reserved.
- **Bit 5..3 - TP: thread pointer**
  These bits are used to address a thread. The addressing is shown in the table below:

TP2	TP1	TP0	THREAD
0	0	0	0
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	4
1	0	1	5
1	1	0	6
1	1	1	7

- **Bit 2..0 - RP: instruction register pointer**
  These bits are used to address an instruction register. The addressing is shown in the table below:

RP2	RP1	RP0	REGISTER
0	0	0	0
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	4
1	0	1	5
1	1	0	6
1	1	1	7

1204 Unusual Condition Detection

The SCUbkpt register is a read only register that gives indication of whether a thread is currently at a breakpoint. A breakpoint is defined as a tight loop (branch to thyself). This function is most useful as a debugging tool. If bit BPn is a one then thread n is currently executing a break point.

SCU Thread BreakPoint Status SCUbkpt

Address: 0x04, IOpage: 0/1
Access Mode: Read only

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	BP7	BP6	BP5	BP4	BP3	BP2	BP1	BP0

- **Bit 15..8 - Res: Reserved**
  These bits are reserved
- **Bit7 – BP7: Thread7 breakpoint**
  When this bit is set, Thread7 is at a breakpoint. When this bit is cleared, Thread7 is not at a breakpoint.
- **Bit6 – BP6: Thread6 breakpoint**
  When this bit is set, Thread6 is at a breakpoint. When this bit is cleared, Thread6 is not at a breakpoint.
- **Bit5 – BP5: Thread5 breakpoint**
  When this bit is set, Thread5 is at a breakpoint. When this bit is cleared, Thread5 is not at a breakpoint.
- **Bit4 – BP4: Thread4 breakpoint**
  When this bit is set, Thread4 is at a breakpoint. When this bit is cleared, Thread4 is not at a breakpoint.
- **Bit3 – BP3: Thread3 breakpoint**
  When this bit is set, Thread3 is at a breakpoint. When this bit is cleared, Thread3 is not at a breakpoint.
- **Bit2 – BP2: Thread2 breakpoint**
  When this bit is set, Thread2 is at a breakpoint. When this bit is cleared, Thread2 is not at a breakpoint.
- **Bit1 – BP1: Thread1 breakpoint**
  When this bit is set, Thread1 is at a breakpoint. When this bit is cleared, Thread1is not at a breakpoint.
- **Bit0 – BP0: Thread0 breakpoint**
  When this bit is set, Thread0 is at a breakpoint. When this bit is cleared, Thread0 is not at a breakpoint.

The SCUwait register is a read only register that gives indication of whether a thread is currently waiting on I/O. This function is useful as a debugging tool and can also be used by a supervisor thread to detect I/O lock ups. If bit WAn is one then thread n is currently stalled on an inp or outp instruction.

SCU Thread I/O Wait Status
SCUwait

Address: 0x05, IOpage: 0/1
Access Mode: Read only

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	WA7	WA 6	WA 5	WA 4	WA 3	WA 2	WA 1	WA 0

- **Bit 15..8 - Res: Reserved**
  These bits are reserved
- **Bit7 – BP7: Thread7 wait**
  When this bit is set, Thread7 is waiting on I/O. When this bit is cleared, Thread7 is not waiting on I/O.
- **Bit6 – BP6: Thread6 wait**
  When this bit is set, Thread6 is waiting on I/O. When this bit is cleared, Thread6 is not waiting on I/O.
- **Bit5 – BP5: Thread5 wait**
  When this bit is set, Thread5 is waiting on I/O. When this bit is cleared, Thread5 is not waiting on I/O.
- **Bit4 – BP4: Thread4 wait**
  When this bit is set, Thread4 is waiting on I/O. When this bit is cleared, Thread4 is not waiting on I/O.
- **Bit3 – BP3: Thread3 wait**
  When this bit is set, Thread3 is waiting on I/O. When this bit is cleared, Thread3 is not waiting on I/O.
- **Bit2 – BP2: Thread2 wait**
  When this bit is set, Thread2 is waiting on I/O. When this bit is cleared, Thread2 is not waiting on I/O.
- **Bit1 – BP1: Thread1 wait**
  When this bit is set, Thread1 is waiting on I/O. When this bit is cleared, Thread1 is not waiting on I/O.
- **Bit0 – BP0: Thread0 wait**
  When this bit is set, Thread0 is waiting on I/O. When this bit is cleared, Thread0 is not waiting on I/O.

1205 Critical Resource Management

The semaphore mechanism in XInC2 allows for 16 user defined semaphores. The critical resource associated with each semaphore is done by firmware convention. The firmware developer can associate any resource or combination of resources to a semaphore. All threads share this convention.

The SCUrsrc, SCUup, SCUdown registers are used for semaphore-based management of critical resources.
Writing a one to bit DNn of the SCUdown register will grant the issuing thread access to the associated critical resource if it is available. If it is not, the issuing thread will stall until the resource is available.
Writing a one to bit UPn of the SCUup register will release access to the critical resource.
Reading SCUrsrc register vector gives indication of which resources are currently being used. Bit RSCn equal to one indicates that the associated resource is currently in use.

For shared resources like memory or hardware peripherals(e.g. SPI) the normal use of the semaphore mechanism is to wrap the code that accesses the resource with writes to the SCUdown and SCUup registers.

SCU Resource Vector Address: 0x06, IOpage: 0/1
SCUrsrc Access Mode: Read only

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RSC15	RSC14	RSC13	RS12	RSC11	RSC10	RSC9	RSC8	RSC7	RSC6	RSC5	RSC4	RSC3	RSC2	RSC1	RSC0

- **Bit 15..0 - RSC: resource vector**
  These bits represent the state of the 16 resources. When RSCn is set, the associated resource is currently being used. When RSCn is cleared, the associated resource is available.

SCU Up Vector Address: 0x06, IOpage: 0/1
SCUup Access Mode: Write only

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
UP15	UP14	UP13	UP12	UP11	UP10	UP9	UP8	UP7	UP6	UP5	UP4	UP3	UP2	UP1	UP0

- **Bit 15..0 - UP: UP vector**
  Bit UPn should be set one when the programmer wants to release access of the associated critical resource.

SCU Down Vector Address: 0x07, IOpage: 0/1
SCUdown Access Mode: Write only

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DN15	DN14	DN13	DN12	DN11	DN10	DN9	DN8	DN7	DN6	DN5	DN4	DN3	DN2	DN1	DN0

- **Bit 15..0 - DN: DOWN vector**
  Bit DNn should be set one when the programmer wants to access the associated critical resource. The thread will stall on access to this register if the resource n is currently assigned to another thread. Access will be granted once the previous thread has released the semaphore by writing to the SCUup register.

1206 SCU Timer

The SCU contains a free running 16-bit counter that is clocked by the processor system clock(Sclk). Reading SCUtime register reads the current state of the counter.

SCU 16-bit counter SCUtime

Address: 0x03, IOpage: 0/1
Access Mode: Read only

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TM15	TM14	TM13	TM12	TM11	TM10	TM9	TM8	TM7	TM6	TM5	TM4	TM3	TM2	TM1	TM0

- **Bit 15..0 - TM: current time**
  These bits represent the current value of the timer

1300 Supervisory Control Extensions (SCX)

The Supervisory Control Extensions are peripheral devices that allow control over system operations such as I/O cell configuration, Clock control and Memory Collision Detection.

1301 Programmable I/O Cell Configuration

All of the I/O cells associated with the GPIO pins as well as the I/O cells dedicated to peripheral pins are programmable via firmware. The programmable features solve the problems generated from variable loading, impedance mismatch, noise immunity, and configuration for variable systems.

The input configuration options are:

- Pull up / Pull down
- Schmitt trigger

The output configuration options are:

- Slew rate control
- 2mA ~ 16mA driving strength

All of the pins on GPIO Ports B, C, D and G are individually configurable. GPIO Ports A, E, F, H, I and J each have only one configuration register to program all pins the same.

Two I/O registers are used to configure the I/O cells. The SCXioCfgP register is used to set a pointer to the I/O cell(s) that are to be configured, and the SCXioCfgD is used to set the configuration.

The procedure for setting an I/O configuration is:

1. Write the pointer to the SCXioCfgP register.
2. Read/Write the configuration to the SCXioCfD register.

Default I/O Configuration The BIOS configures the I/O to the following default configuration:

I/O cell	pullup/pulldown	Input Type	Drivestrength	SlewRate
PA	Disabled	Normal	4 mA	slow
PB0	Disabled	Normal	16 mA	slow
PB1	Disabled	Normal	16 mA	slow
PB2	Disabled	Normal	4 mA	slow
PC0	Disabled	Normal	4 mA	slow
PC1	Disabled	Normal	4 mA	slow
PC2	Disabled	Normal	4 mA	slow
PC3	Disabled	Normal	4 mA	slow
PC4	Disabled	Normal	4 mA	slow
PC5	Disabled	Normal	4 mA	slow
PC6	Disabled	Normal	4 mA	slow
PC7	Disabled	Normal	4 mA	slow
PD0	Disabled	Normal	4 mA	slow
PD1	Disabled	Normal	4 mA	slow
PD2	Disabled	Normal	4 mA	slow
PD3	Disabled	Normal	4 mA	slow
PD4	Disabled	Normal	4 mA	slow
PD5	Disabled	Normal	4 mA	slow
PD6	Disabled	Normal	4 mA	slow
PD7	Disabled	Normal	4 mA	slow
PE	Disabled	Normal	4 mA	slow
PF	Disabled	Normal	4 mA	slow
PG0	Disabled	Schmitt Trigger	16 mA	slow
PG1	Disabled	Normal	16 mA	slow
PG2	Disabled	Normal	16 mA	slow
PG3	Disabled	Normal	4 mA	slow
PH	Disabled	Normal	4 mA	slow
PI	Disabled	Normal	4 mA	slow
PJ	Disabled	Normal	4 mA	slow
TMRB	Disabled	Normal	4 mA	slow

SCX I/O Configuration Pointer Address: 0x08
SCXioCfgP Access Mode: Read/Write

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	CP4	CP3	CP2	CP1	CP0

- **Bit 14..6 - Res: Reserved**
  These bits are reserved.
- **Bit 4..0 - CP: CP4,CP3,CP2,CP1,CP0: Configuration Pointer**
  These bits are used to set the pointer to the I/O cell(s) of interest. See the table below for configuration pointer specifics.

CP0	I/O cell referenced	I/O cell type
0	PA	bidirectional
1	PB0	bidirectional
2	PB1	bidirectional
3	PB2	bidirectional
4	PC0	bidirectional
5	PC1	bidirectional
6	PC2	bidirectional
7	PC3	bidirectional
8	PC4	bidirectional
9	PC5	bidirectional
10	PC6	bidirectional
11	PC7	bidirectional
12	PD0	bidirectional
13	PD1	bidirectional
14	PD2	bidirectional
15	PD3	bidirectional
16	PD4	bidirectional
17	PD5	bidirectional
18	PD6	bidirectional
19	PD7	bidirectional
20	PE	bidirectional
21	PF	bidirectional
22	PG0	bidirectional
23	PG1	bidirectional
24	PG2	bidirectional
25	PG3	bidirectional
26	PH	bidirectional
27	PI	bidirectional
28	PJ	bidirectional
29	TMRB	bidirectional

SCX I/O Configuration Data Address: 0x09
SCXioCfgD Access Mode: Read/Write

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	PUDE	PUD	SMT	DS2	DS1	DS0	SR

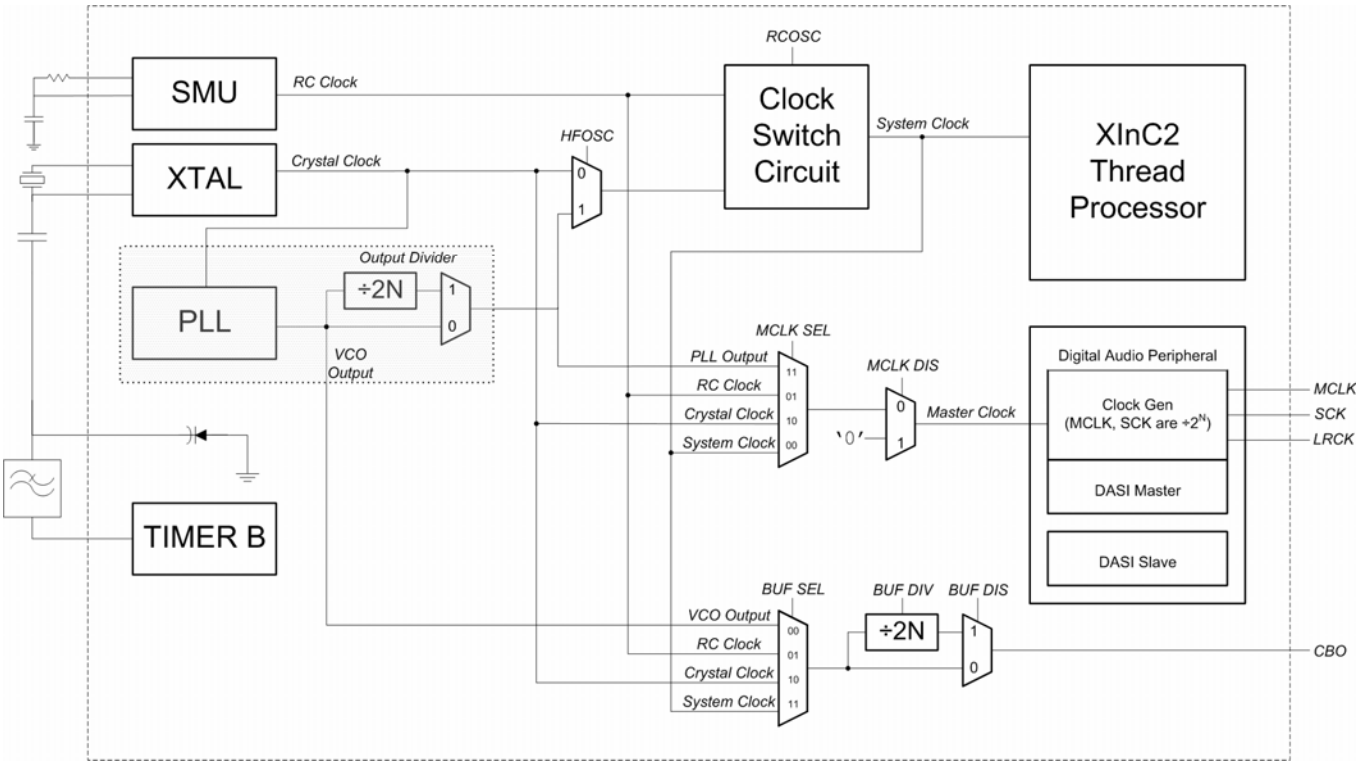
- **Bit 15..7 - Res: Reserved**
  These bits are reserved.
- **Bit 6 - PUDE: Pullup/Pulldown enable**
  When the pin(s) are configured as an output, the pullup/pulldown feature is disabled. When this bit is set(one), a pullup/pulldown is enabled. When this bit is cleared(zero), pullup/pulldown is disabled.
- **Bit 5 - PUD: Pullup/Pulldown select**
  When the pin(s) are configured as an output the pullup/pulldown feature is disabled. When this bit is set(one) a pullup is selected. When this bit is cleared(zero) a pulldown is selected.
- **Bit 4 - SMT: Schmitt Trigger**
  This bit is ignored when the pin(s) are configured as an output. When this bit is set(one) Schmitt Trigger inputs are enabled. When this bit is cleared(zero) Schmitt Trigger inputs are disabled.
- **Bit 3,2,1 - DS8,DS4,DS2: Output Driving Capability**
  These bits are ignored when the pin(s) are configured as an input. These three bits control the drive strength of the output buffer. The configuration is shown in the table below:
  (Where X=1 at V_{DD}=3.3V, X=0.55 at V_{DD}=2.5V, X=0.35 at V_{DD}=1.8V.)

DS2	DS1	DS0	Driving Strength
0	0	0	2*X mA
0	0	1	4*X mA
0	1	0	6*X mA
0	1	1	8*X mA
1	0	0	10*X mA
1	0	1	12*X mA
1	1	0	14*X mA
1	1	1	16*X mA

- **Bit 0 - CP: SR: Output Slew Rate**
  This bit is ignored when the pin(s) are configured as input. When this bit is set(one), fast slew is configured. When this bit is cleared(zero), slow slew is configured.

1302 Clock Configuration

A block diagram of the XInC2 clock system is shown below.



The System Clock is the XInC2 global clock. The processor core and most peripherals are run from the system clock. The system clock may be selected from the RC clock, the crystal oscillator, or the Phase Locked Loop (PLL) source.

RCCLK This is the startup clock for XInC2. Normally it is run from a low-speed RC clock (~32kHz) external to XInC2.

XTCLK This is the crystal clock, used as a timing source for the PLL, external ICs and the processor core. This is normally intended to be a stable reference frequency for the DASI or an RF module and an intermediate frequency for the XInC2 core.

PLLCLK This is the Phase-Locked Loop oscillator. It can be used to synthesize clocks for the processor core and external ICs.

"Sleepy Mode" XInC2 has several low-power modes. The simplest is called "Sleepy Mode". To enter this mode the developer can stop all threads except for one and then switch the system clock to the low speed RC oscillator input. The high speed oscillator can then be turned off. In this mode the system current consumption is reduced to just a few tens of uA. The thread that continues to run can poll for a triggering event, at which point it can setup and enable the other threads to resume operation.

For more information on the other low-power modes, see the Sleep Mode Unit (SMU) section of this document.

DASI Peripheral The logic for the Digital Audio Serial Interface (DASI) is on a separate clock domain. The DASI clock may be selected by the appropriate bits in the SCXaltCfg register (this section). See the DASI section of this document for more detail on the functionality of the Digital Audio Serial Peripheral.

Clock Buffered Output The CBO allows a clock to be supplied to an external circuit from any of the internal clocks through an independent clock divider.

SCX Clock Configuration Address: 0x72
SCXclkCfg Access Mode: Read/Write

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CST	RC	0	0	HFOSC	RCOSC	0	0	0	0	0	FEB	S1	S0	EB	E

- **Bit 15 - CST: RC/OSCn status**
  This bit is read only. This bit gives indication of when a clock switch has finished. When this bit is set(one), the system clock(Sclk) source is OSCn clock. When this bit is cleared(zero), the Sclk source is RC clock.
- **Bit 14 - RC: RC clock value**
  This bit is a read only. This bit is the RC clock input sampled by the system clock.
- **Bit 13,12 - Res: Reserved**
  These bits are reserved
- **Bit 11 - HFOSC: High Frequency Oscillator select**
  When this bit is clear(zero), the crystal clock source is selected. When this bit is set(one), the PLL clock source is selected.
- **Bit 10 - RCOSC: RC/HFOSC select**
  When this bit is set(one), the system clock source is switched to the clock source configured by the HFOSC bit. When this bit is cleared(zero), the system clock source is switched to the RC clock source. The clock switching mechanism takes 2 RC clock cycles + 2 OSCn clock cycles to switch. The CST bit gives indication of when the clock switching has completed.
- **Bit 9..5 - Reserved**
  These bits are reserved.
- **Bit 4 - FEB: Xtal osc feedback resistor select**
  When this bit is set(one), the internal feedback resistor is disabled. When this bit is cleared(zero), a 1Meg internal feedback resistor is enabled.
- **Bit 3 - S1: Xtal osc low / high frequency mode select**
  When this bit is set(one), the oscillator is configured for high frequency operation. When this bit is cleared(zero), the oscillator is configured for low frequency operation.
- **Bit 2 - S0: Xtal osc fundamental / overtone mode select**
  When this bit is set(one), the oscillator is configured for 3rd overtone oscillation. When this bit is cleared(zero), the oscillator is configured for fundamental oscillation.
- **Bit 1 - EB: Xtal osc tri-state disable**
  When this bit is set(one), the oscillator is disabled and the output is tri-stated. When this bit is cleared(zero), the oscillator output is enabled and the operation of the oscillator is dependent on the E bit.
- **Bit 0 - E: Xtal osc enable**
  When this bit is set(one), the oscillator is enabled. When this bit is cleared(zero), the oscillator is disabled.

SCX Alternate Clock Configuration Address: 0x73
SCXaltCfg Access Mode: Read/Write

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MCLK SEL1	MCLK SEL0	DCLK DIS	BUF DIV4	BUF DIV3	BUF DIV2	BUF DIV1	BUF DIV0	BUF SEL1	BUF SEL0	BUF DIS	SR	DS2	DS1	DS0	E

- **Bit 15..14 – MCLK SEL: MCLK Select**
  These bits select the clock source for the I2S peripheral as shown below:

MCLK_SEL	Clock Source
00	System Clock
01	RC
10	Crystal Oscillator
11	PLL Divided Out

- **Bit 13 – DASI CLK DIS: Disable DASI Logic Clock**
  When this bit is set(one), the DASI master mode logic clock source is disabled. When this bit is clear(zero), the DASI master mode logic clock source is enabled. (Enabled by default at boot)
- **Bit 12..8 – BUF DIV: Buffered Clock Divider**
  These five bits control the clock divider before the output buffer. The buffered clock is divided by 2*BUF_DIV. If BUF_DIV is zero, the output clock is the same frequency as the input clock.
- **Bit 7..6 – BUF_SEL: Buffered Clock Select**
  These bits select the clock source for the buffered clock output as shown below:

BUF_SEL	Clock Source
00	PLL VCO
01	RC
10	Crystal Oscillator
11	System Clock

- **Bit 5 – BUF DIS: Buffered Clock Divider Disable**
  When this bit is set(one), the buffered divider clock source is disabled. When this bit is clear(zero), the buffered divider clock source is enabled. (Enabled by default at boot)
- **Bit 4 – SR: Buffered Clock Slew Rate**
  When this bit is set(one) fast slew is configured for Oscillator1 output buffer. When this bit is cleared(zero) slow slew is configured.
- **Bit 3,2,1 - DS2,DS1,DS0: Buffered Clock drive strength**
  These three bits control the drive strength of the output buffer. The configuration is shown in the table below:

DS2	DS1	DS0	Driving Strength
0	0	0	2 mA
0	0	1	4 mA
0	1	0	6 mA
0	1	1	8 mA
1	0	0	10 mA
1	0	1	12 mA
1	1	0	14 mA
1	1	1	16 mA

- **Bit 0 - E: Buffered Clock Output enable**
  When this bit is set(one), Oscillator1 output buffer is enabled. When this bit is cleared(zero), the buffer is disabled.

The output frequency of the PLL is determined by the following formula:

$$f_{OUT} = \frac{f_{REF} \bullet FB}{REF \bullet OD}$$

The values of REF, FB and OD are programmed into the SCXpllCfg registers below:

SCX Phase Locked Loop Configuration SCXpllCfg0

Address: 0x74
Access Mode: Read/Write

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	RNG	FB5	FB4	FB3	FB2	FB1	FB0	REF5	REF4	REF3	REF2	REF1	REF0	EN

- **Bit 15..14: Reserved**
  These bits are reserved.
- **Bit 13 – RNG: PLL Range Select**
  This bit selects the frequency range of the internal PLL. If clear(zero), low frequency mode is selected (20MHz to 100MHz). If set(one), high frequency mode is selected(100MHz to 300MHz).
- **Bit 12..7 – FB: PLL Feedback Divider**
  These bits control the feedback divider for the PLL. Valid values for the divider are 1 ≤ N ≤ 63.
  The VCO frequency must be in the range
  20MHz ≤ f_{VCO} ≤ 300 MHz for proper operation.
- **Bit 6..1 – REF: – PLL Input Reference Divider**
  These bits control the PLL's reference clock divider. Valid values for the divider are 1 ≤ N ≤ 63.
  The limits on frequencies output from the divider are 5MHz ≤ f_{ref} ≤ 300MHz.
- **Bit 0 - EN: PLL Enable**
  When this bit is set(one), the PLL is enabled. When this bit is cleared(zero), the PLL is disabled.

SCX Phase Locked Loop Configuration SCXpllCfg1

Address: 0x75
Access Mode: Read/Write

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	OD5	OD4	OD3	OD2	OD1	OD0

- **Bit 15..6: Reserved**
  These bits are reserved.
- **Bit 5..0 – ODx: PLL Output Divider**
  These bits control the PLL output divider. Valid values for the divider are: 1 ≤ OD ≤ 63, for a divider range of 2 to 126 (ie. This is a divide-by-2N divider.)
  The output frequency is in the range 20MHz ≤ f_{out} ≤ 300MHz.

SCX RC Clock Counter SCXcount

Address: 0x76
Access Mode: Read Only

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	RC11	RC10	RC9	RC8	RC7	RC6	RC5	RC4	RC3	RC2	RC1	RC0

- **Bit 15..0 – RCx: Current Value of RC Counter**
  This register contains a 12-bit free running counter that increments with each rising edge of the RC clock. The RC clock is sampled by the system clock, so the system clock must be at least twice as fast as the RC clock.

1303 Memory Collision

The SCXmemcol register is a read/write register that gives indication of whether a memory collision between the Instruction Port and Data Port has occurred. This function is useful as a debugging tool. If bit MCn is one then a collision has occurred on the associated block of memory. Writing to the SCXmemcol register will clear all bits of the register.

SCX Memory Collision Detect SCXmemcol

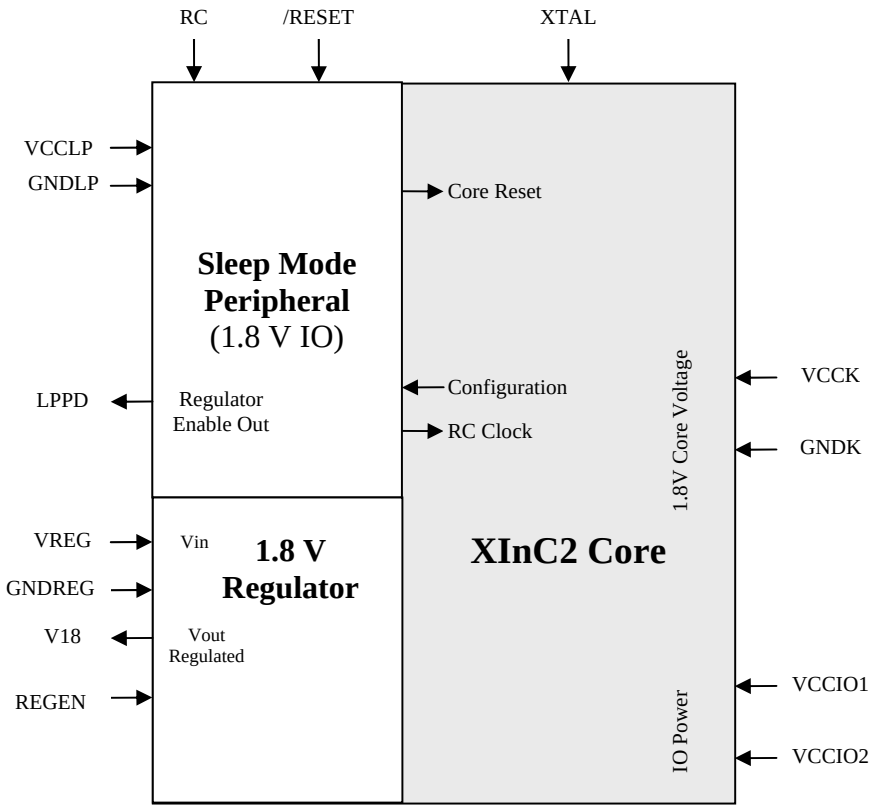
Address: 0x05
Access Mode: Read/Write

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MC15	MC14	MC13	MC12	MC11	MC10	MC9	MC8	MC7	MC6	MC5	MC4	MC3	MC2	MC1	MC0

- **Bit15 – MC15:** RAM block7.
  When this bit is set, a collision has occurred on RAM block7.
- **Bit14 – MC14:** RAM block6.
  When this bit is set, a collision has occurred on RAM block6.
- **Bit13 – MC13:** RAM block5.
  When this bit is set, a collision has occurred on RAM block5.
- **Bit12 – MC12:** RAM block4.
  When this bit is set, a collision has occurred on RAM block4.
- **Bit11 – MC11:** RAM block3.
  When this bit is set, a collision has occurred on RAM block3.
- **Bit10– MC10:** RAM block2.
  When this bit is set, a collision has occurred on RAM block2.
- **Bit9 – MC9:** RAM block1.
  When this bit is set, a collision has occurred on RAM block1.
- **Bit8 – MC8:** RAM block0.
  When this bit is set, a collision has occurred on RAM block0.
- **Bit7 – MC7:** ROM block7.
  When this bit is set, a collision has occurred on ROM block7.
- **Bit6 – MC6:** ROM block6.
  When this bit is set, a collision has occurred on ROM block6.
- **Bit5 – MC5:** ROM block5.
  When this bit is set, a collision has occurred on ROM block5.
- **Bit4 – MC4:** ROM block4.
  When this bit is set, a collision has occurred on ROM block4.
- **Bit3 – MC3:** ROM block3.
  When this bit is set, a collision has occurred on ROM block3.
- **Bit2 – MC2:** ROM block2.
  When this bit is set, a collision has occurred on ROM block2.
- **Bit1 – MC1:** ROM block1.
  When this bit is set, a collision has occurred on ROM block1.
- **Bit0 – MC0:** ROM block0.
  When this bit is set, a collision has occurred on ROM block0.

1400 Sleep Mode Unit

The Sleep Mode Unit (SMU) is a special peripheral provided in XInC2 for low power functionality. It operates on RC clock at 1.8 V and is powered independently from the XInC2 core. It controls the regulator from which XInC2 is powered. This enables Sleep Mode Unit to configure XInC2 core in various low power modes. It can be configured for desired mode by the firmware. It also provides basic watchdog timer functionality. It is directly connected to RC clock and Chip/System Reset input pins and provides gated RC clock and Reset to XInC2.



1401 Sleep Mode Peripheral Functions

- RC Oscillator
- Internal 16-bit counter provides wakeup timer function.
- Internal 10-bit counter provides Reset Delay Timer (previously referred as POD)
- Provides RC clock to system (Capable of gating the system RC clock)
- Output pin for enable of external 3.3V regulator
- Communication with XInC2 Core for configuration and status information.
- Separate clock domains while configuring for Watchdog
- XINC2 switches to RC clock before going to any of the Sleep Mode
- Sleep Mode vs. States

Sleep Mode	Regulator Enable	RC Oscilltor	RC Clock Output	Core Reset
Default /Power On	Hi-Z	On	RC Clock	High
Low Power	Active	On	Low	High
Very Low Power	Active	On	Low	Low
Extremely Low Power	Non-active	On	Low	Low

1402 SMU Modes

Default Mode This the default power on mode in which XInC2 boots up. In this mode SMU passes RC clock to the system. ”Core Reset” is high and Regulator enable is in Hi-Z (high impedance) state. This enables configuring external pull up or pull down to enable the regulator. (more could be described).

Low Power Mode This mode is reached by configuring **SMUcfg0** and **SMUcfg1** registers in the SMU from XInC2. In this mode XInC2 is not supplied RC clock, ”Core Reset” is high and Regulator enable is in Hi-Z state. Before switching to this mode user must make sure that XInC2 is running on the RC clock and not on crystal or PLL. In this mode memory information in the registers and RAM will be retained. The user must take caution of not loading any value in the RDT (Reset Delay Timer). Else “Core Reset” will be low and system will get reset for the time period specified in RDT.

Very Low Power Mode This mode is reached by configuring **SMUcfg0** and **SMUcfg1** registers in the SMU from XInC2. In this mode XInC2 is not supplied RC clock, ”Core Reset” is low and Regulator enable is in Hi-Z state. Before switching to this mode user must make sure that XInC2 is running on the RC clock and not on crystal or PLL. In this mode memory information in the registers and RAM will be lost. This mode could be used to provide software reset internally.

Extremely Low Power Mode This mode is reached by configuring **SMUcfg0** and **SMUcfg1** registers in the SMU from XInC2.In this mode XInC2 is not supplied RC clock, ”Core Reset” is low and Regulator enable is not-active(If the Regulator enable for external regulator is active-high this pin will be low and if it is low this pin will be high). The polarity needs to be appropriately configured with Bit 11 in **SMUcfg1** register. Before switching to this mode user must make sure that XInC2 is running on the RC clock and not on crystal or PLL. In this mode XInC2 is completely powered down and hence memory information in the registers and RAM will be lost. This mode could be used to provide minimum power consumption. The power dissipation is only due to leakage.

1403 SMU Configuration

NOTE: The necessary sequence of configuration (loading these registers) is first SMUcfg0 and then SMUcfg1.This will ensure that the counter starts decrementing only with SMUcfg1 is loaded appropriately.

SMU Configuration0 Address: 0x78
SMUcfg0 Access Mode: Read/Write

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
T15	T14	T13	T12	T11	T10	T9	T8	T7	T6	T5	T4	T3	T2	T1	T0

- **Bit 15..0 T: Timer Value**
  These bits are the timer value for the Sleep Mode Timer. The timer decrements with every second RC clock tick when SME bit is set (one). This allows the counter to count 131070 RC ticks before wrapping. Assuming ~32 KHz RC clock, this allows XInC2 to sleep up to ~4.0959 seconds before waking up from the sleep.

SMU Configuration1 Address: 0x79
SMUcfg1 Access Mode: Read/Write

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
WDE	SM1	SM0	SME	REP	RD10	RD9	RD8	RD7	RD6	RD5	RD4	RD3	RD2	RD1	RD0

- **Bit 15 - WDE: Watch Dog Enable**
  This bit when set (one), enables the watchdog timer function. If the Sleep Mode Timer wraps, the system will be reset. To prevent the system reset, “FFFE” must be written to SMUcfg0 before the counter can wrap. This gives the firmware up to ~4.0959 seconds between writes to SMUcfg0.

- **Bit 14-13 - SM: Sleep Mode**
  These bits configure the desired of sleep mode.
  - “00” – Default Mode (XInC2 normal operation)
  - “01” – Low Power Mode
  - “10” – Very Low Power Mode
  - “11” – Extremely Low Power Mode

- **Bit 12 - SME: Sleep Mode Enable**
  The Sleep Mode Enable bit when set (one) causes XInC2 to enter the currently configured Sleep Mode**.** Once set there is no option to reset XInC2 and it will be reset at the end of the sleep time.

- **Bit 11 - RP: Regulator Polarity**
  The Regulator Enable Polarity bit is the active (enabled) polarity of the external regulator. Configuring this bit will allow proper operation of the “Extremely Low Power” power-saving mode.

- **Bit 10..0 – RD: Reset Delay Timer**
  The Reset Delay Timer decrements with every second RC clock tick. This allows the counter to count 4094 RC ticks before wrapping. Assuming ~32kHz RC clock, this allows up to ~127.93 ms delay before Reset to the core (“Core Reset”) goes high again and XInC2 starts. The user should make sure that this is loaded zero while configuring for Default or Low Power Modes and not loaded zero if the user wants to reset the system for the given amount of time (Time after the reset timer in SMUcfg0) for Very Low Power and Extremely Low Power modes. This is

due to the fact that Reset is low for *n* number of ticks, where *n* is the decimal equivalent number in this counter.

Invalid States

The following states are not supported.

- WDE=1 and SME=1 Watchdog and sleep Mode enabled at the same time.
- SMN and SME with following states.

SMN1	SMN0	SME
0	1	0
1	0	0
1	1	0

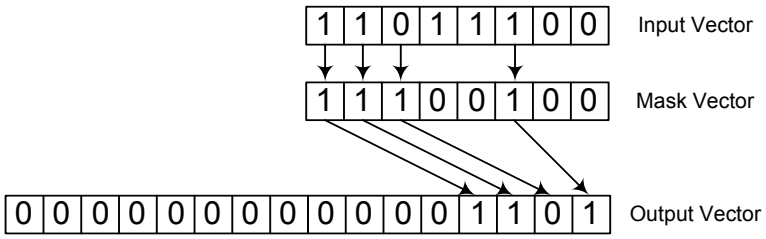
- Counter value in RDT in Low Power Mode.
- Counter value in RDT in Default Mode.
- Any other configuration apart from the specified.

1500 Shared Functional Units (SFU)

Introduction The Shared Functional Units (SFU) are peripherals that perform mathematical or bit manipulation functions that can be thought of as extensions to the Instruction set of XInC2. These units provide simple hardware solutions that would otherwise take many software instructions to complete. The Pack Bits, Population Count, Least Significant One, and Bit Reverse units have a built in semaphore mechanism that allows for multiple threads to share access to the peripherals without the overhead of using the semaphore based mechanism found in the SCU. The built in semaphore mechanism stalls a thread from initiating a write to a functional Unit until the result of the previous operation has been read.

1501 Pack Bits

The Pack Bits Functional Unit take an 8-bit vector and an 8-bit mask as inputs and returns an 8-bit vector packed with the bits of interest indicated by the mask The resulting vector is right justified. The Pack operation computes in a single cycle so the result is available to be read immediately following a write.
The 8-bit mask and 8-bit vector are written to the SFUpack register. Reading SFUpack returns the packed result.



SFU Pack Bits Address: 0x39, IOpage: 0,1
SFUpack Access Mode: Write only

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
M7	M6	M5	M4	M3	M2	M1	M0	IV7	IV6	IV5	IV4	IV3	IV2	IV1	IV0

- **Bit 15..8 - M: mask vector**
  These bits represent the 8 - bit input mask.
- **Bit 7..0 - IV: input vector**
  These bits represent the 8 - bit input vector to be packed.

SFU Pack Bits Result Address: 0x39, IOpage: 0,1
SFUpack Access Mode: Read only

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	OV7	OV6	OV5	OV4	OV3	OV2	OV1	OV0

- **Bit 15..8 - Res: Reserved**
  These bits are reserved.
- **Bit 3..0 - OV: output result vector**
  These bits represent the packed result.

1502 Population Count

The Population Count Functional Unit counts the number of bits that are set(one) in a 16-bit input vector. The count operation computes in a single cycle so the result is available to be read immediately following a write.
The input vector to be counted is written to the SFUpop register. Reading SFUpop register returns the number of ones in the input vector.

SFU Population Count SFUpop

Address: 0x3A, IOpage: 0,1
Access Mode: Write only

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IV15	IV14	IV13	IV12	IV11	IV10	IV9	IV8	IV7	IV6	IV5	IV4	IV3	IV2	IV1	IV0

- **Bit 15..0 - IV: input vector**
  These bits represent the 16 - bit input vector to be counted.

SFU Population Count Result SFUpop

Address: 0x3A, IOpage: 0,1
Access Mode: Read only

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	PC4	PC4	PC2	PC1	PC0

- **Bit 15..5 - Res: Reserved**
  These bits are reserved.
- **Bit 4..0 - PC: population count**
  These bits represent the number of ones in the input vector.

1503 Least Significant One

The Least Significant One Functional Unit returns the bit position of the first bit set(one) in a 16-bit input vector. The operation computes in a single cycle so the result is available to be read immediately following a write.
The input vector is written to the SFUls1 register. Reading SFUls1 register returns the bit position(0-15) of the first one. If the input vector contains no ones then the zero flag(ZF) bit will be set.

SFU Least Significant One SFUls1

Address: 0x3B, IOpage: 0,1
Access Mode: Write only

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IV15	IV14	IV13	IV12	IV11	IV10	IV9	IV8	IV7	IV6	IV5	IV4	IV3	IV2	IV1	IV0

- **Bit 15..0 - IV: input vector**
  These bits represent the 16 - bit input vector to be searched for the least significant one.

SFU Least Significant One Result SFUls1

Address: 0x3B, IOpage: 0,1
Access Mode: Read only

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ZF	0	0	0	0	0	0	0	0	0	0	0	BP3	BP2	BP1	BP0

- **Bit 15 - ZF: Zero flag**
  This bit is set when the input vector is zero.
- **Bit 14..4 - Res: Reserved**
  These bits are reserved.
- **Bit 3..0 - BP: LS one bit position**
  These bits represent the bit position of the first '1' found in the input vector.

1504 Bit Reverse

The Bit Reverse Functional Unit performs a bit reversal operation on a 16-bit input vector. The reversal operation computes in a single cycle so the result is available to be read immediately following a write.
The input vector to be reversed is written to the SFUrev register. Reading SFUrev register will return the input vector reversed.

SFU Bit Reverse SFUrev Address: 0x38, IOpage: 0,1
Access Mode: Write only

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IV15	IV14	IV13	IV12	IV11	IV10	IV9	IV8	IV7	IV6	IV5	IV4	IV3	IV2	IV1	IV0

- **Bit 15..0 - IV: input vector**
  These bits represent the 16 - bit input vector to be bit reversed.

SFU Bit Reverse Result SFUrev Address: 0x38, IOpage: 0,1
Access Mode: Read

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OV15	OV14	OV13	OV12	OV11	OV10	OV9	OV8	OV7	OV6	OV5	OV4	OV3	OV2	OV1	OV0

- **Bit 15..0 - OV: output vector**
  These bits represent the 16 - bit reversed result.

1600 Vector Processing Unit (VPU)

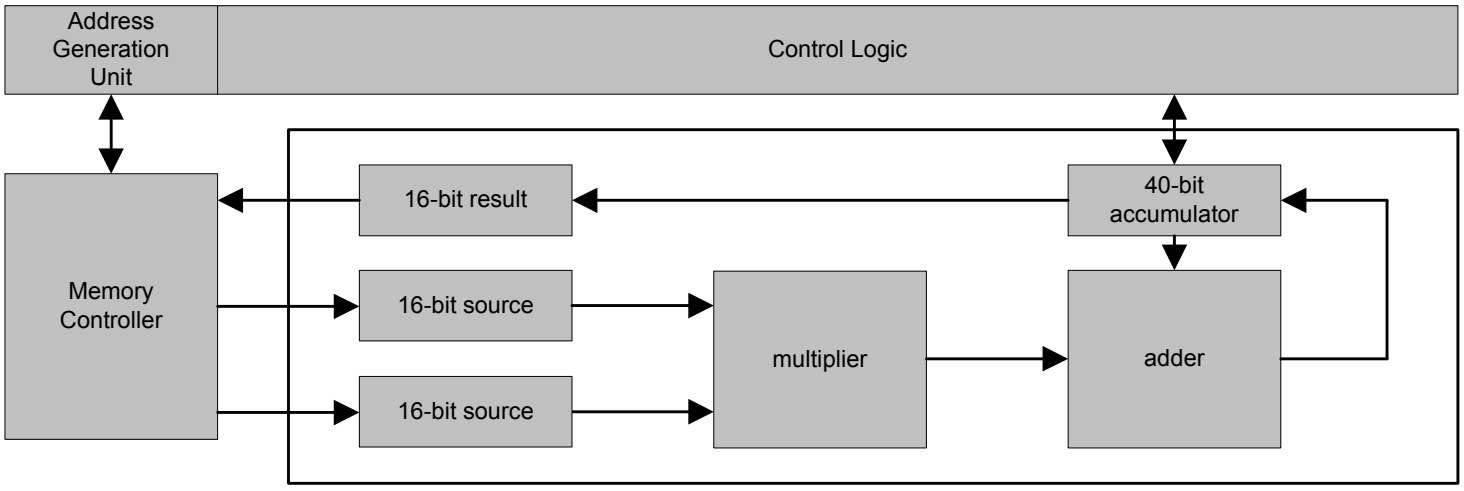
The Vector Processing Unit is a peripheral designed to provide XInC2 processor with enhanced Digital Signal Processing (DSP) capabilities. Once configured, VPU can operate in several different modes making it a versatile signal processing tool.

1601 Introduction

The block diagram below shows the basic building blocks of the VPU unit. It consists of a MAC unit as well as the necessary control logic for memory operations such as fetching and storing. It uses a DMA controller for fast memory access. However, VPU memory accesses are done when the memory accesses by the threads are idle. Hence memory access priority is given to threads. VPU also utilizes a 128 word long separate register file where the coefficients of one operand (Operand B) can be stored. The register file is provided for storing the coefficients of filters of lengths up to 128, however it can also be used for repetitive operations where the values of one operand are constant.

Although VPU operates on 16 bit 2's complements numbers, the precision for the intermediate results are kept to 40 bit with its 40 bit accumulator. This allows 256 MAC operations before accumulator overflows become an issue. If needed, the intermediate 40 bit results can also be read by the thread processor in three loads (16, 16 and 8 bits).

.



MAC Implementation The multiplier has 16-bit operands and a 32-bit result. The multiplier is pipelined.

Accumulator The accumulator is 32 bits + 8 extension overflow bits = 40-bits so integer and fractional operations can be performed without intermediate rounding errors.
Any multiplication of two N-bit values can produce a 2N bit result and any addition of two N bit values can produce an (N+1) bit result. The resolution of the required accumulator for the sum of M multiplications is shown below:
$N = K1 + K2 + \log_2(M) = K1 + K2 + \log(M) / \log(2)$ where,
N=required number of bits for accumulator
K1 and K2 are the resolution of each operand to multiply and M is the number of accumulations.
$\text{Log}(M) = (N - K1 - K2)*\log(2)$
$M=10^{\wedge((N - K1 - K2)*\log(2))}$

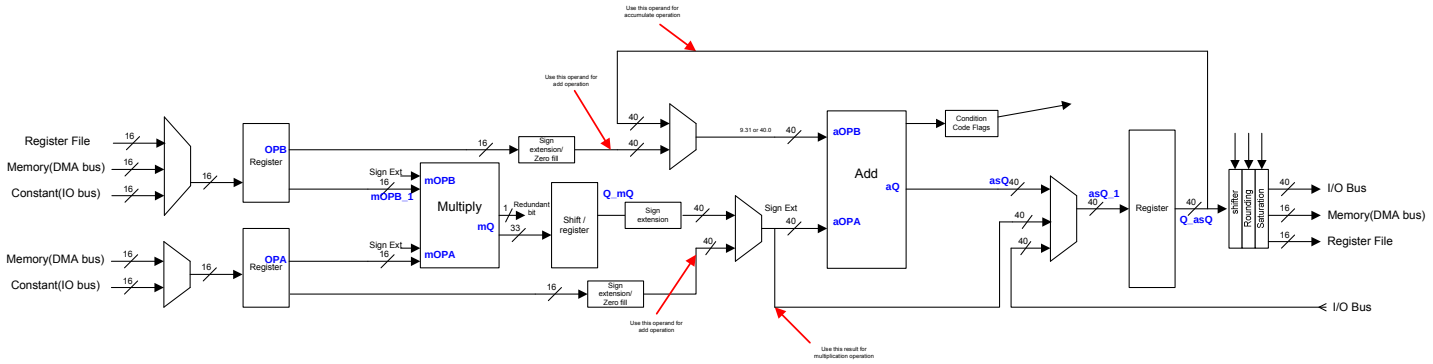
Example:
16 bit operands
40 bit accumulator
$M=10^{\wedge((40 - 16 - 16)*\log(2))} = 256$

So a 40 bit accumulator will allow for an FIR filter of 256 taps without worrying about overflow of intermediate results.

1602 MAC Architecture

The Diagram below shows a more detailed architecture of the MAC unit. The input to the multiplier, adder and the accumulator can be configured by the multiplexers shown in the figure. The configurations are entered through the VPUconfig0 register. The type of operation is specified by configuring the VPUconfig0 and VPUconfig1 registers. The VPU configuration settings for the multiplexers are also shown in the figure. It can be seen that the add or multiply and add operations are supported for both scalar (single) or vector operands. The results of the operation can either be fed back to the adder for accumulation or stored in a memory. The data source for the Operand A is either stored in memory or can be directly entered to I/O bus. The data source for the Operand B can be Memory, I/O bus or the VPU register file. Bits 0 and 1 control the input source paths for the Operand A and B.

.



Data after passing through Mux A and Mux B are fed to the OPA and OPB registers then into the 16 x 16 multiplier. They are also sign extended and fed to the Mux C and Mux D. In this way, if desired, the multiplier is taken out of the circuit and the unit operate as an accumulator operating on the sign extended (to 40 bits) OPA and OPB. Mux E is supplied to allow OPA result to be written to the register file. The constants can be stored in a memory location and transferred to register file by correctly setting the Mux B, Mux D and Mux E.

Data shifter / Scaler / Limiter

The shifter/scaler/limiter block provides special post processing on data reads from the accumulator.

The following shifting can be done:

- Shift data one bit to the left (fractional mode)
- No shifting(integer mode)

Saturation

The destination operand size is normally 16 bits which is smaller then the accumulator.
If the result of the accumulator can be represented without overflow in the destination operand size then the result is not modified. If overflow occurs then the data limiter will saturate the result.

Control logic

Control logic is required to provide an interface to the peripheral bus for configuration and to the memory bus for reading operands and writing results to the shared RAM. There is also logic required for the address generation unit that is responsible for incrementing/decrementing address pointers.

Memory Interface The DMA interface in the VPU allows shared access to the RAM. The DMA interface connects to the D-port of the memory cross-bar. The VPU has a Memory-to-Memory architecture. For one MAC operation two 16-bit operands are read from RAM and one 16-bit result is written to RAM.

Scalar / Vector mode The MAC can be configured for vector mode or scalar mode.

Tables below shows the VPU configuration settings for various operations

VPU config0 Bit #	Setting	Operation	Mux	Mode
1..0	00	Source Data for OPA and OPB are from IO bus	A,B	Scalar
	01	Source Data for OPA and OPB are from RAM		
	10	Source Data for OPA from RAM, OPB from Register file		
	11	Reserved		
2	0	Results will be stored to RAM		
	1	Results will be stored to Register file		
4..3	00	Results will not be saved (no memory write)		
	01	LSW of the results will be stored in memory		
	10	MSW of the results will be stored in memory		
	11	8 extended bits will be stored in memory		
5	0	Source Data for the Adder aOPA comes from multiplier	D	
	1	Source Data for the Adder aOPA comes from OPA register		
6	0	Source Data for the Adder aOPB comes from Accumulator	C	
	1	Source Data for the Adder aOPB comes from OPB register		
7	0	Source Data for the Accumulator comes from Adder	E	
	1	Source Data for the Accumulator is from output of Mux D		
8	0	Base-address pointer of OPA will **not** be incremented after a MAC operation		
	1	Base-address pointer of OPA will be incremented after a MAC operation		
9	0	Base-address pointer of OPA will be incremented by **one** after a MAC operation only if bit 8 is set		
	1	Base-address pointer of OPA will be incremented by **two** after a MAC operation only if bit 8 is set		
10	0	Base-address pointer of OPAB will **not** be incremented after a MAC operation		
	1	Base-address pointer of OPB will be incremented after a MAC operation		
11	0	Base-address pointer of Results will **not** be incremented after a MAC operation		
	1	Base-address pointer of Results will be incremented after a MAC operation		
12	0	Results will **not** be shifted by one before being stored		
	1	Results will **be** shifted by one before being stored		
13	0	ADD operation is chosen		
	1	Subtract operation is chosen		
14	0	Adder input OPB will **not** be cleared after each MAC operation		
	1	Adder input OPB will be cleared after each MAC operation		
15	0	Do not clear Accumulator		
	1	Clear Accumulator		

VPU config1 Bit #	Setting	Operation	Mode
0	0	OPA source type is 16 bit signed	
	1	OPA source type is 16 bit unsigned (17 bit signed)	
1	0	OPB source type is 16 bit signed	
	1	OPB source type is 16 bit unsigned (17 bit signed)	
2	0	Multiplier result will not be shifted by one	Integer
	1	Multiplier result will be shifted by one	Fractional
3	0	Operands will be fed into LSW of the adder	
	1	Operands will be fed into MSW of the adder	
4	0	Result will be truncated	Fractional
	1	Result will be rounded1.31 or 9.31 to 1.15 or 9.15	Fractional
6..5	00	Result will be saturated 9.31 to 1.31	Fractional
	01	40.0 to 16 bits	Integer
	10	No saturation	
	11	reserved	
7	0	Pipeline operations	
	1	Do not pipeline operations	
15	0	VPU is not busy doing operation	
	1	VPU is busy doing operation	

1603 Arithmetic Data Representations

The Vector Processing Unit supports Unsigned or Signed Two's complement formats. Signed-magnitude or other formats are not supported. The operands for arithmetic operations are 16 bits.

Fractional and integer data formats are supported. In an integer the radix point is assumed to be to the right of the LSB so that all magnitude bits have a weight of 1 or greater. In a fractional format the assumed radix point lies within the number so that some or all of the magnitude bits have a weight of less then 1. The VPU has support for a pure fractional representation where all of the bits have a weight of less then 1.

The notation used to represent the number uses two numbers separated by a period. The first number is the number of bits to the left of the radix point. The second number is the number of bits to the right of the radix point.

Unsigned Integer (16.0 format) Unsigned integers lie in the range of 0 -> 65535. The radix point is interpreted to be after the LSB.

Two's Complement Integer (16.0 format) Signed integers lie in the range of -32768 -> 32767. The radix point is interpreted to be to the right of the LSB.

Unsigned Fraction (0.16 format) Unsigned fractions lie in the range of 0.0 -> 2.0 - 2^-15.

Two's Complement Fraction (1.15 format) Signed fractions lie in the range of -1.0 -> 1.0 – 2^-15. The radix point is interpreted to be to the right of the MSB.

Mixed Integer/Fraction Numbers mixed with integer and fractions can be used but scaling and alignment of the radix point must be done manually.

Multi-precision Numbers Operations on multi-precision numbers can be done be using the unsigned and signed representations.

1604 Arithmetic operations

Each of the operands can be 16 bit signed or 16 bit unsigned. Two's complement representation is assumed for 16 bit signed operations. Integer or fractional representations are available for both signed and unsigned operands.

Addition and subtraction operations are performed identically for fractional and integer arithmetic. For mixed integer/fraction numbers the radix point must be lined up to do addition or subtraction. Shifting can be done on both integer and fractional data values. An arithmetic left shift of 1 bit is a multiplication by two. An arithmetic right shift of 1 bit is a division of a signed value by two. A logic right shift of 1 bit is a division of an unsigned value by two.

Multiplication operations are not performed the same for integer and fractional numbers. The result of multiplication of two 16.0 operands is a 32.0 number. In the integer mode, no shifting should be done.

The result of multiplication of two 1.15 operands is a 2.30 number. In the fractional mode, the MAC will shift the result left one bit. This produces a 1.31 number.

Standard VPU operations

For filtering applications the numbers are represented in Q15 format. In this format the number after the Q represents the quantity of fractional bits. Therefore each number has a sign bit as well as 15 fractional bits and no integer bits. If Q14 format is desired where we have one integer bit then bit12 of VPUcf0 should be set. This will ensure that the result will be shifted by one value. Following figure shows Q15 format representation of max and minimum values.

MSB															LSB	Value
sign	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}	2^{-8}	2^{-9}	2^{-10}	2^{-11}	2^{12}	2^{13}	2^{-14}	2^{-15}	
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	32767
Pos(0)	0.5	0.25	0.125	0.0625	0.03125	0.015625	0.0078125	etc	etc.	etc.	etc.	etc.	etc.	etc.	etc.	0.999969
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	32768
Neg (-1)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1

Hence the largest 16 bit positive value is 32767 corresponds to 0.999969 whereas largest 16 bit negative value, 32768 correspond to -1. To convert decimal value to Q15 format, multiply the value with 2^{15} and round to the next nearest integer value.

0.3456 in Q15 format 0.3456 x2^{15}= 11325

23456 in Q15 format = 23456/2^{15}= 0.71582

For Q14 Format

1.5476 in Q14 format 1.5476 x2^{14}= 25355

1605 Control Unit

In fractional mode, VPU fetches the values that are stored in the Base Addresses of OPA and OPB for multiplication. Then the base addresses for the multiplicand and the multiplier are recalculated for the multiplication based on the number of samples to be processed, address masks used as well as the length of the filter.

VPU can be set up to utilize a circular buffer so that once the address pointers are supplied, the multiplication will be done automatically. However, some care needs to be given how the data boundaries are chosen for the calculation.

For circular buffer usage the base addresses of OPA, OPB and Result should start at 2^N Address boundaries. In the following example we have two vectors. OPB is composed of a0…a2 and OPA consists of b0…b7. We would like to do a calculation similar to the one shown in the following equation for 8 samples.

$$y(n)= \sum_{k=0}^{k=2} a(k)xb(n-k)$$

We will use circular buffer for the OPA and Result here. The multiplications will assume Q15 format.
The data are would look like below:
//the following expression is used to force the data at 2^N boundary, in this case 2^N=16
@ = (@ + 0x010-1) & -0x010
//OPA
OPA_Base_Address:
b0: @ = @ + 1
b1: @ = @ + 1
b2: @ = @ + 1
b3: @ = @ + 1
b4: @ = @ + 1
b5: @ = @ + 1
b6: @ = @ + 1
b7: @ = @ + 1

@ = (@ + 0x010-1) & -0x010
//OPB
OPB_Base_Address:
a0: @ = @ + 1
a1: @ = @ + 1
a2: @ = @ + 1

//Results will be stored below reserve 8 words for results
Result_Base_Address: @ = @ + 8

//The VPU settings are as follows
VPU_Conf0:(1<<clrA | 1<<clraaOPB | 0<<addnsubCfg | 0<<RSh | 1<<Rinc | 0<<Binc | 0<<ADec | 1<<Ainc | 0<<ACcfg | 0<<aB | 0<<aA | 0b10<<DFW | 0<<Rdest | 0b01<<OPsrc)

VPU_Conf1: (0<<Pipe) | (0b00<<RS) | (1<<RR) | (0<<AddAlign) | (1<<MS) | (0<<Btyp) | (0<<Atyp)

```
//VPU setting for the base addresses as well as length of the filter etc.
//load the configuration settings to VPU configuration registers.
ld      r0, VPU_Conf0
outp    r0,VPUcfg0

ld      r0, VPU_Conf1
outp    r0,VPUcfg1

//the following will force the address pointer to wrap every 8 samples
//we are not using a circular buffer for the OPB so mask can be FF
mov     r0,0xFF07
outp    r0,VPUopadrmsk

//Use circular buffer for the results
mov     r0,0x007F
outp    r0,VPUrsadrsmk

//Provide pointers for the base addresses of  the OPA,OPB and the Results

mov     r0, OPA_Base_Address
outp    r0,VPUopAba

mov     r0,OPB_Base_Address
outp    r0,VPUopBba

mov     r0, Result_Base_Address
outp    r0,VPUrsba

//provide details for the filter length and number of samples to be processed
//each MAC is 3 multiplication and we will process 8 samples
mov     r0,8<<8 | 3
outp    r0,VPUfilterLength

//VPU seetings are complete and dummy writing to the VPUopA register
//will start the process.

outp    r0,VPUopA
```

The coefficient and the data alignment will be as follows

Address		OPA	Address	OpB	
0xXX10	←Base_Address_Pointer OPA	bo	0xXX20	a0	
0xXX11		b1	0xXX21	a1	
0xXX12		b2	0xXX22	a2	←Base_Address_pointer OpB
0xXX13		b3			
0xXX14		b4			
0xXX15		b5			
0xXX16		b6			
0xXX17		b7			

Given all the information above the calculation will proceed as follow
Y(1) = a2xb0 + a1xb7 + a0xb6 1st MAC
After first MAC the Address Pointer of OPA and Result will be incremented and OPB

Will be reset to point a2

Y(2) = a2xb1 + a1xb0 + a0xb7 2nd MAC
Similarly
Y(3) = a2xb2 + a1xb1 + a0xb0
Y(4) = a2xb3 + a1xb2 + a0xb1
Y(5) = a2xb4 + a1xb3 + a0xb2
Y(6) = a2xb5 + a1xb4 + a0xb3
Y(7) = a2xb6 + a1xb5 + a0xb4
Y(8) = a2xb7 + a1xb6 + a0xb5

Example of a single Multiplication:

In the following example we would like to multiple two numbers (Q15 format), saturate the result and read the result from VPU's internal register. Since this is a single multiplication we do not need to provide any address information or number of samples to be multiplied etc. However VPU needs to be set-up to read data from IO lines.
The VPU configuration settings :

Mult_Cf0: (1<<clrA | 0<<clraaOPB | 0<<addnsubCfg | 0<<RSh | 0<<Rinc | 0<<Binc | 0<<ADec | 0<<Ainc | 1<<ACcfg | 0<<aB | 0<<aA | 0b10<<DFW | 0<<Rdest | 0b00<<OPsrc)

Mult_Cf1: (0<<Pipe) | (0b00<<RS) | (1<<RR) | (0<<AddAlign) | (1<<MS) | (0<<Btyp) | (0<<Atyp)

```
ld      r0, Mult_Cf0
outp    r0,VPUcfg0

ld      r0, Mult_Cf1
outp    r0,VPUcfg1

//move the first integer to VPUopB register
mov     r0,16384
outp    r0,VPUopB
//move the second integer to VPUopA to start
//multiplication
mov     r0, 16384
outp    r0,VPUopA
//result is available after one cycle
inp     r0,VPUrsmsw
```

Note that Integer multiplication is the default format. Therefore if you will be using VPU only for integer multiplication then, you do not need to set-up VPU for the integer operation since it is the default format. However if you are going to use a mix of fractional and integer arithmetic then you need to make sure the settings are correct. If you set the bits 6 and 5 of the Configuration register 1 for VPU as 01 then the results will be saturated to 40->16 bit before they are stored in memory. However you also have the option of reading the VPU registers. In this way you have an access to all 40 bits.

Example of the Implementation of Digital Cross-Over Using XInC2

In this example we will develop a digital crossover using third order Butterworth filters. The final structure will be implemented using a first order stage cascaded with a second order stage. The sampling rate is 48000 Hz and the Crossover frequency is 3000 Hz.

VPU is designed to operate in Q15 format so if the filter coefficients are in the range of

-1 to $1-2^{-15}$ (-32768 to 32767)

Setting for the VPU is as follows:
VPU_DigiCross_Cf1: (0<<Pipe) | (0b00<<RS) | (1<<RR) | (0<<AddAlign) | (1<<MS) | (0<<Btyp) | (0<<Atyp)

Busy								pipe	RS	RS	RR	ADDalign	MS	Btyp	Atype
								0	0	0	1	0	1	0	0

The important settings are: Result saturation (RS bits 5 and 6) is set to 00 which implies that we are using Q15 format numbers; MS is set to 1, this is the operation that converts the numbers to its original format. If Q15 format is being used, we should have the MS set to 1.

VPU_DigiCross_Cf0 : (1<<clrA | 1<<clraaOPB | 0<<addnsubCfg | 0<<RSh | 1<<Rinc | 0<<Binc | 0<<ADec | 1<<Ainc | 0<<ACcfg | 0<<aB | 0<<aA | 0b10<<DFW | 0<<Rdest | 0b01<<OPsrc)

ClrA	ClraaOPB	AddnS ubCfg	RSh	Rinc	Binc	ADec	Ainc	ACcfg	aB	aA	DFW	DFW	Rdest	OPsrc	OPsrc
1	1	0	0	1	0	0	1	0	0	0	1	0	0	0	1

The important parameters are:
ClrA (bit15): is set to clear the accumulator before the MAC operation
ClraaOpB : is set to clear the Operand B of the adder input. This is important if you are processing multiple samples. After each sample adder input is cleared so that accumulator value is not fed back to the adder input.

0<<RSh: is cleared, this is the default operation for the Q15 arithmetic. If you are using Q14 then this bit should be set.

The following is the Cross-Over implementation: The Coefficients are calculated using the following formulas:
Low- Pass Filter:

$\gamma=\cos\theta c/(1+\sin\theta),\ \alpha=(1-\gamma)/2$

$\beta k=1/2((1-0.5*\sin\theta)/(1+0.5*\sin\theta))$
$\gamma k=(0.5+\beta k)\cos\theta c;\ \alpha k=(0.5+\beta k-\gamma k)/4$

$dk=2*\sin((2k-1)\pi/(4M+0.5))$

$\theta c=2\pi fc/fs \quad k=1..M$

difference equation has the form and this form will be implemented in XInC2

$y(n)=\alpha[x(n)+x(n-1)]+\gamma y(n-1)$
$y1(n)=2\{\alpha 1[x1(n)+2x1(n-1)+x1(n-2)]+\gamma 1y1(n-1)-\beta 1y1(n-2)\}$

where x1(n)=y(n)
The high pass filter has the following coefficient formulas:

$\gamma=\cos\theta c/(1+\sin\theta),\ \alpha=(1+\gamma)/2$

$\beta k=1/2((1-0.5* \sin\theta)/ (1+0.5* \sin\theta))$
$\gamma k=(0.5+\beta k)\cos\theta c;\ \ \alpha k=(0.5 + \beta k +\gamma k)/4$

$dk=2*\sin((2k-1)\pi /(4M+0.5))$

$\theta c=2\pi fc/fs \quad k=1..M$

The difference equation is:

$y(n)=\alpha[x(n)-x(n-1)] + \gamma y(n-1)$
$y1(n) = 2\{\alpha 1[x1(n)-2x1(n-1)+x1(n-2)] + \gamma 1 y1(n-1) – \beta 1 y1(n-2)\}$

where $x1(n)=y(n)$

The coefficients that are calculated from above formulas were scaled to get the Q15 format as well as scaling by 2 in front of x1(n-1) terms are done before entering to the data place.

The low pass coefficients for the first are:

0	0	α	α	γ	0
0	0	5437	5437	21895	0

The alignment of the delay line with the coefficients are as follows:

0	0
0	0
x(n-1)	5437
x(n)	5437
y(n-1	21895
y(n)	0

Note that the above structure is chosen due the fact that the second stage has six multiplications. By doing this above we are having three redundant multiplications but we do not reset the filter length. The multiplication y(n) with 0 can be eliminated but again then the base address for the OPA and Result will be different then you will require one more instruction to set the base address of the result. Hence although above implementation does have more multiplication, it is an efficient way of implementing the filter.

The second order stage alignment is as follows:

x(n-2)	523
x(n-1)	1046
x(n)	523
y(n-2)	-11121
y(n-1	25411
y(n)	0

For the High Pass filter we have the following

First Stage Delay Line	First Stage Coefficients		Second Stage Delay Line	Second Stage Coefficients	
0	0		x(n-2)	13229	
0	0		x(n-1)	-26458	
x(n-1)	-27331		x(n)	13229	
x(n)	27331		y(n-2)	-11121	
y(n-1	21895		y(n-1	25411	
y(n)	0		y(n)	0	

The following is the XInC assembly code:

Digital_CrossOver_Init:

```
        ld      r0,r7, VPU_DigiCross_Cf1
        outp    r0,VPUcfg1
        mov     r0,0x1F1F
        outp    r0,VPUopadrmsk
        mov     r0,0x001F
        outp    r0,VPUrsadrsmk
        jsr     r6,r6
```

This subroutine is used to initialize the VPU for the filtering operations that are common to each sample processing.

```
//the following is the actual filtering code
//Digital_CrossOver:

        st      r0,AC0_DCross_Xn
//before calling this routine make sure that r0 has the sample to be filtered, we
//store it in the delay line expressed as before (x(n))
//setting of the VPU for filtering
        ld      r0,r7, VPU_DigiCross_Cf0
        outp    r0,VPUcfg0
        mov     r0,1<<8 | 6
        outp    r0,VPUfilterLength
//filter length corresponds to the number of MAC operations it is equivalent to
//number of filter coefficients, we are setting it to be six, as well as specifying that
//we will process only one sample
        mov     r0,AC0_DCross_FOS_BaseAddress + 5       //results
        outp    r0,VPUrsba
//the results are going to be written to this memory location
        mov     r0,AC0_DCross_FOS_BaseAddress + 5
        outp    r0,VPUopAba
//the operand A will be fetched from this location (in the memory model above
//it corresponds to the location of y(n)

        mov     r0,AC0_DCross_FOS_Coeff_BaseAddress + 5
        outp    r0,VPUopBba
//the other operand OPB will be fetched from this location
/start the VPU process
        outp    r0, VPUopA

//dummy write to this register starts the MAC process

//1 MAC operation is as follows
//refer to the memory and coefficient model for the first stage above

y(n) =y(n)*0 + y(n-1) * 21895 + x(n)*5437 +x(n-1)*5437 +0*0 +0*0
```

```
//now while VPU is doing its multiplications we can preset some of the registers
//for the next VPU operation

//we do preset the registers for the next VPU operation

        mov     r2,AC0_DCross_SOS_BaseAddress+5
        mov     r3,AC0_DCross_SOS_Coeff_BaseAddress + 5
        ld      r4,r7,VPU_Digital_Cross2_Cf0

//here we are redefining the all the addresses as well as changing the configuration
WaitUntil_AC0_DCross_Done:
        inp     r0,VPUcfg1
        bc      NS, WaitUntil_AC0_DCross_Done

//we wait until VPU is finished its operation once it is done we load the VPU registers
//for the next operation. However since the result of the first stage is the input for the next
//stage we store the result to the input of the second stage

/first stage filtering is done we can do the second
//stage now we need to store the result into appropriate place
        ld      r0,AC0_DCross_FOS_BaseAddress + 5
        st      r0,AC0_DCross_X1n

//now continue with set-up

        outp    r2,VPUopAba
        outp    r3,VPUopBba
        outp    r2,VPUrsba
        outp    r4,VPUcfg0
//start the VPU process
        outp    r0, VPUopA

//now wait until the VPU is done, we could again set-up for the next operation

    WaitUntil_AC0_DCross2_Done:
        inp     r2,VPUcfg1
        bc      NS, WaitUntil_AC0_DCross2_Done

        ld      r0,AC0_DCross_SOS_BaseAddress+5

    //this is the final result and we are assuming that result
    //will be returned in register r0


    //now we need to update the delay lines and we do that by VPU

        ld      r2,r7,AC0_Biquad_DelayCf0
        outp    r2,VPUcfg0
        mov     r2,BiQuad_Delay_Scaling_Coefficient
        outp    r2,VPUopBba
        mov     r2, AC0_DCross_FOS_BaseAddress + 1
        outp    r2,VPUopAba
        mov     r2, AC0_DCross_FOS_BaseAddress
        outp    r2,VPUrsba


        mov     r2,11<<8 | 1
        outp    r2,VPUfilterLength
    //wait until the delay line is updated
```

```
        outp    r2, VPUopA
//it basically does 11 multiplications but after each multiplication
//it increments the results base address by one
//we are multiplying it with 1 you can also do a copy operation

WaitUntil_DCross_DelayLine_Done:
        inp     r2,VPUcfg1
        bc      NS, WaitUntil_DCross_DelayLine_Done
```

once it is done we are done with the filtering operations

//above example is one channel only but can easily be expanded to two channel

1606 Processing Bandwidth

1607 VPU Configuration

VPU General Configuration0 Address: 0x03
VPUcfg0 Access Mode: Read/Write

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CA	COPB	0	0	0	0	0	0	0	0	0	0	0	0	0	0

- **Bit 15 – clearAccum: Clear accumulator(WRITE ONLY)**
  0=do not clear
  1=immediately clear the accumulator(all 40 bits)
- **Bit 14 – clearaOPB: auto clear adder input aOPB on new MAC operation**
  0=do not auto clear
  1=auto clear aOPB
- **Bit 13 addnsub: select add or subtract operation**
  0=add
  1=subtract
- **Bit 12 – ResultShiftCfg: Result shift config**
  0=pass through
  1=arithmetic shift left 1
- **Bit 11 – Resultinc: Result outer loop counter increment**
  0=do not auto increment base address on outer loop count
  1=increment base address on outer loop count
- **Bit 10 – OPBinc: OPB outer loop counter increment**
  0=do not auto increment base address on outer loop count
  1=increment base address on outer loop count
- **Bit 9 – OPAdec: OPA decimation mode**
  0=increment base address by one if OPAinc='1'

1= increment base address by two if OPAinc='1'

- **Bit 8 – OPAinc: OPA outer loop counter increment**
  0=do not auto increment base address on outer loop count
  1=increment base address on outer loop count
- **Bit 7 - ACcfg: AccumulatorReg source configuration**
  0=asQ
  1=Q_mQ sign
- **Bit 6 – aB: adder operand B config**
  0 = aOPB <= accQ
  1 = aOPB <= OPB
- **Bit 5 – aA: adder operand A config**
  0 = aOPA <= mQ
  1 = aOPA <= OPA
- **Bit 4..3 – DFW: data field to write**
  00 = no memory write
  01 = write LSW
  10 = write MSW
  11 = write EXT
- **Bit 2 – Rdest: Result Memory destination**
  0=DMA
  1=Regfile (Regfile cannot be used for source operand)
- **Bit 1..0 – OPsrc: OPA and OPB source**
  00 = OPA < IObus, OPB < IObus (SCALAR MODE)
  01 = OPA < DMA, OPB < DMA
  10 = OPA < DMA, OPB < REG
  11 = RESERVED

VPU General Configuration1 Address: 0x03
VPUcfg1 Access Mode: Read/Write

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

- **Bit 15 – BUSY: Operation busy flag (READ ONLY)**
  0 = not busy
  1 = busy
- **Bit 7 – P: Operation pipelining configuration**
  0 = pipeline operations
  1 = do not pipeline operations
- **Bit 6..5 – RS: Result saturation(all modes are signed)**
  00 = 9.31->1.31 saturation(fractional)
  01 = 40.0->16.0 saturation(integer)
  10 = no saturation
  11 = reserved
- **Bit 4 – RR: result rounding mode(fractional mode only)**
  0=no rounding(truncation)
  1=rounding, 1.31or 9.31 -> 1.15 or 9.15
- **Bit 3 addalign: adder operand alignment**
  0=feed operands into LSW
  1=feed operands int MSW
- **Bit 2 – MS: multiplier result shift(integer/fractional mode)**
  0 = no shift
  1 = shift left
- **Bit 1 – Btyp: OPB source type**
  0=16 bit signed
  1=16 bit unsigned(17 bit signed)

- **Bit 0 – Atyp: OPA source type**
  0=16 bit signed
  1=16 bit unsigned(17 bit signed)

Result is written to memory when inner loop counter counts down to X"0000"
OPA baseAddress = OPA baseAddress or OPA baseAddress + 1 when

VPU OPB|OPA base address mask Address: 0x03
VPUopadrmsk Access Mode: Read/Write

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OPB7	OPB6	OPB5	OPB4	OPB3	OPB2	OPB1	OPB0	OPA7	OPA6	OPA5	OPA4	OPA3	OPA2	OPA1	OPA0

- **Bit 15..8 - OPB: OperandB buffer mask**
  8 bit vector address mask of Operand B
- **Bit 7..0 - OPA: OperandA buffer mask**
  8 bit vector address mask of Operand A

VPU OPA buffer base address Address: 0x03
VPUopAba Access Mode: Read/Write

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BA15	BA14	BA13	BA12	BA11	BA10	BA9	BA8	BA7	BA6	BA5	BA4	BA3	BA2	BA1	BA0

- **Bit 15..0 - BA: OperandA buffer base address**
  16 bit base address of Operand A

VPU OPA scalar Address: 0x03
VPUopA Access Mode: write only

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OPA15	OPA14	OPA13	OPA12	OPA11	OPA10	OPA9	OPA8	OPA7	OPA6	OPA5	OPA4	OPA3	OPA2	OPA1	OPA0

- **Bit 15..0 - OPA: OperandA scalar**
  16 bit scalar OperandA

VPU OPB buffer base address Address: 0x03
VPUopBba Access Mode: Read/Write

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BA15	BA14	BA13	BA12	BA11	BA10	BA9	BA8	BA7	BA6	BA5	BA4	BA3	BA2	BA1	BA0

- **Bit 15..0 - BA: OperandB buffer base address**
  16 bit base address of OperandB

VPU OPB scalar Address: 0x03
VPUopB Access Mode: Write only

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OPB15	OPB14	OPB13	OPB12	OPB11	OPB10	OPB9	OPB8	OPB7	OPB6	OPB5	OPB4	OPB3	OPB2	OPB1	OPB0

- **Bit 15..0 - OPB: OperandB scalar**
  16 bit scalar operandB

VPU result buffer base address Address: 0x03
VPUrsba Access Mode: Read/Write

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BA15	BA14	BA13	BA12	BA11	BA10	BA9	BA8	BA7	BA6	BA5	BA4	BA3	BA2	BA1	BA0

- **Bit 15..0 - BA: result buffer base address**
  16 bit base address of the arithmetic result

VPU result base address mask Address: 0x03
VPUresultMsk Access Mode: Read/Write

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BA15	BA14	BA13	BA12	BA11	BA10	BA9	BA8	BA7	BA6	BA5	BA4	BA3	BA2	BA1	BA0

- **Bit 15..0 - BA: Result buffer mask**
  8 bit vector address mask of Result

VPU accumulator lsw Address: 0x03
VPUrslsw Access Mode: Read/Write

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
LW15	LW14	LW13	LW12	LW11	LW10	LW9	LW8	LW7	LW6	LW5	LW4	LW3	LW2	LW1	LW0

- **Bit 15..0 - LW: least significant word of result**
  Bits 15..0 of the accumulator register

VPU accumulator msw Address: 0x03
VPUrsmsw Access Mode: Read/Write

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MW15	MW14	MW13	MW12	MW11	MW10	MW9	MW8	MW7	MW6	MW5	MW4	MW3	MW2	MW1	MW0

- **Bit 15..0 - MW: most significant word of result**
  Bits 31..16 of the accumulator register

VPU accumulator Ext Address: 0x03
VPUrsext Access Mode: Read/Write

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	EB7	EB6	EB5	EB4	EB3	EB2	EB1	EB0

- **Bit 15..0 - EB: extension byte of result**
  Bits 39..32 of the accumulator register

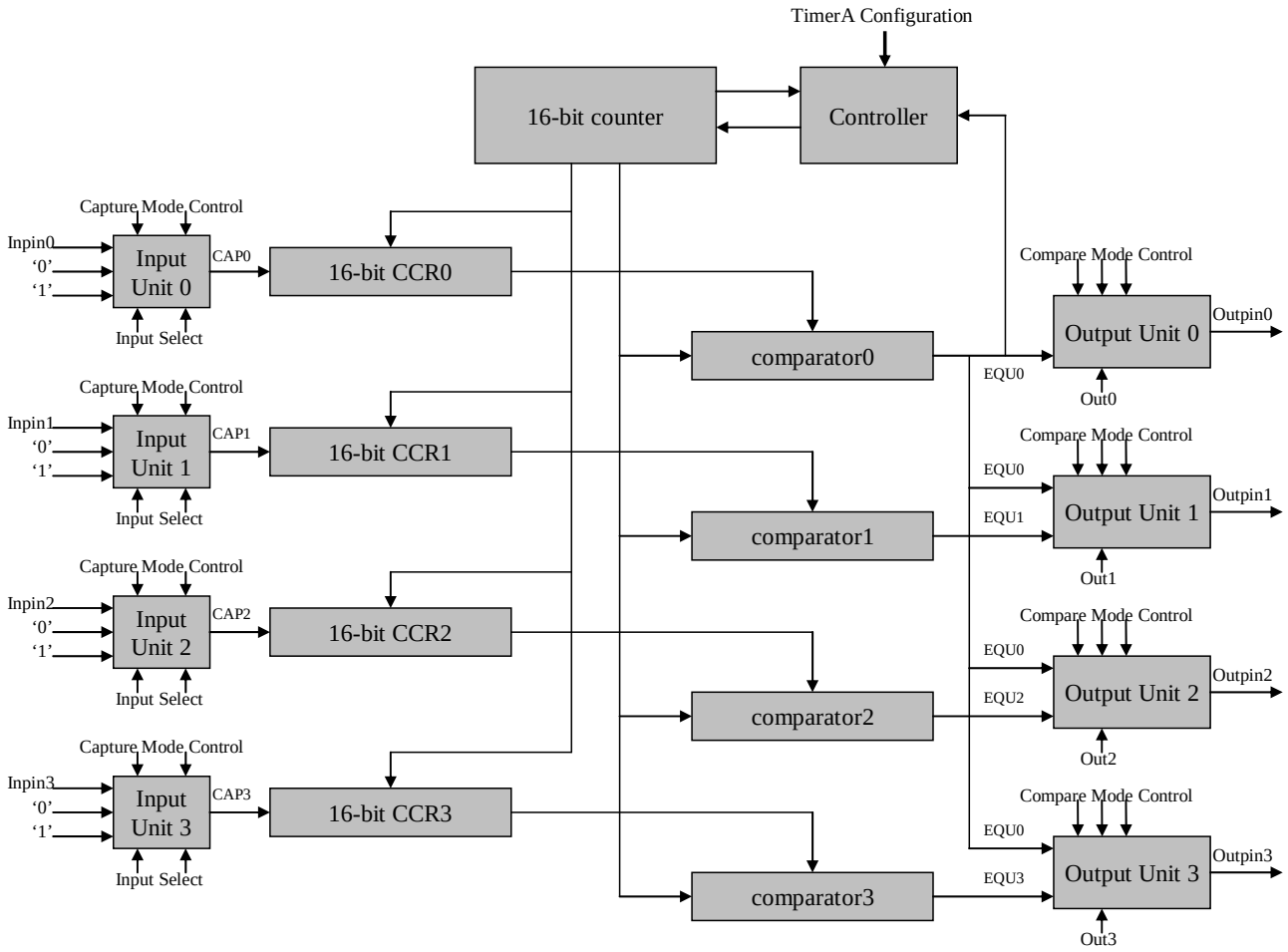
Length1 | Length0 Address: 0x03
VPUalgLength Access Mode: Read/Write

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OPB7	OPB6	OPB5	OPB4	OPB3	OPB2	OPB1	OPB0	OPA7	OPA6	OPA5	OPA4	OPA3	OPA2	OPA1	OPA0

- **Bit 15..8 – L1: Algorithm Length 1**
  8 bit "outer loop" counter
- **Bit 7..0 – L0: Algorithm Length 2**
  8 bit "inner loop" counter

1700 TimerA

TimerA is a general purpose 16 bit timer. The timer consists of multiple operating modes with selectable clock source, four configurable capture/compare registers, configurable input modules, and configurable output modules.
The timer can be used to generate output waveforms such as a PWM signals, capture input signals on compare, trigger an event on a input signal transition, or be used as a configurable source of time.
Input signals are synchronized with the XInC2 system clock.
The basic block diagram of the timer is shown below.



1701 Timer Mode Control

TimerA has 4 modes of operation:

- **Stopped**
- **Up**: count up to CR0 and restart at 0
- **Continuous Up**: timer counts up to 0xFFFF and restarts at 0
- **Up/Down**: timer counts continuously up to CR0 and back down to 0

1702 Capture / Compare Modules

There are four capture/compare modules.
Each module can be configured independently for capture or compare mode.

Capture mode Capture mode is used to latch the timer value in the CCRx register when an event on an input signal occurs.

Compare mode The input signal may be latched with compare signal EQUx occurs. This mode is used to generate events at specific time intervals or to generate output signals.

1703 General Timer Configuration & Status

TimerA configuration Address: 0x3E, IOpage: 0,1
TMRAcfg Access Mode: Read/Write

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OFflg	DIR	0	0	0	0	0	0	OFfc	CLK3	CLK2	CLK1	CLK0	Mcfg1	Mcfg0	E

- **Bit 15 – OFflg: Counter overflow flag**
  Offlg bit is set when the counter counts to zero. This bit is cleared by reading the register or writing a '0' to OFflg bit.
- **Bit 14 – DIR: Counter direction – read only**
  When the counter is incrementing then DIR bit is set. When the counter is decrementing then DIR bit is cleared
- **Bit 7– OFfc: Counter overflow flow-control**
  Setting the OFfc bit will enable blocking flow control on reads of the timerAcfg register.
- **Bit 6,5,4,3 – CLK: Timer Clock configuration**
  These bits are used to configure clock rate for the timer. The relationship between the timer clock and the system clock(Sclk) frequency is shown below.

CLK3	CLK2	CLK1	CLK0	CLK Frequency
0	0	0	0	Sclk / 2
0	0	0	1	Sclk / 4
0	0	1	0	Sclk / 8
0	0	1	1	Sclk / 16
0	1	0	0	Sclk / 32
0	1	0	1	Sclk / 64
0	1	1	0	Sclk / 128
0	1	1	1	Sclk / 256
1	0	0	0	Sclk / 512
1	0	0	1	Sclk / 1024
1	0	1	0	Sclk / 2048
1	0	1	1	Sclk / 4096
1	1	0	0	Sclk / 4096
1	1	0	1	Sclk / 4096
1	1	1	0	Sclk / 4096
1	1	1	1	Sclk / 4096

- **Bit 2,1 – Mcfg: Timer Mode configuration**
  Thse bits are used to configure the timer mode.

Mcfg1	Mcfg0	mode	Description
0	0	Halt	Timer halted
0	1	Up	count up to CR0 and restart at 0
1	0	Continuous	count up to 0xFFFF and restarts at 0
1	1	Up/Down	Count up to CR0 and back down to 0

- **Bit 0 – E: Timer enable**
  Setting this bit to one enables the timer.

TimerA 16-bit counter Address: 0x3F, IOpage: 0,1
TMRAcount Access Mode: Read Only

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TM15	TM14	TM13	TM12	TM11	TM10	TM9	TM8	TM7	TM6	TM5	TM4	TM3	TM2	TM1	TM0

- **Bit 15..0 - TM: current counter**
  These bits represent the current value of the counter.

1704 Capture/Compare Module0

TimerA Cap/Comp Module0 Config Address: 0x41, IOpage: 0,1
TMRACCM0cfg Access Mode: Read/Write

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CCflg	0	0	0	EFfc	CMc1	CMc0	IMc1	IMc0	OMc2	OMc1	OMc0	OPinE	OUTx	CCM	IN

- **Bit 15 – CCflg: Capture or Compare event flag**
  In compare mode the CCflg bit is set when the EQU0 event occurs. In capture mode the CCflg bit is set when the capture event occurs. This bit is cleared by reading the register or writing a '0' to CCflg bit.
- **Bit 14..12 – RES: Reserved**
  Reserved.
- **Bit 11– EFfc: event flag flow-control**
  Setting the EFfc bit will enable blocking flow control on reads of the TMRACCM0cfg register. Writes to this register are never blocked. The reads will block while CCflg = '0'.
- **Bit 10,9 – CMc: Capture Module configuration**
  These bits are used to configure the Capture module. This configures what type of signal will trigger a capture event.

CMc1	CMc0	mode
0	0	Disabled
0	1	Rising edge
1	0	Falling edge
1	1	Rising or falling edge

- **Bit 8,7 – IMc: Input module configuration**
  These bits are used to configure the input module.

IMc1	IMc0	Mode
0	0	disabled
0	1	inPin
1	0	zero
1	1	one

- **Bit 6,5,4 – OMc: Output module configuration.**
  These bits are used to set the mode of operation for OutputModule0.

OMc2	OMc1	OMc0	Mode	Description
0	0	0	Output	At EQU0, Opin0 = OUTx
0	0	1	Set	EQU0 sets Opin0
0	1	0	Reset	EQU0 resets Opin0
0	1	1	Toggle	EQU0 toggles Opin0
1	0	0	Toggle/Set	EQU0 toggles Opin0
1	0	1	Toggle/Reset	EQU0 toggles Opin0
1	1	0	Set/Reset	EQU0 sets Opin0
1	1	1	Reset/Set	EQU0 resets Opin0

- **Bit 3 – OPinE: Output pin enable**
  In compare mode setting this bit will enable the output pin buffer. Clearing this bit will tristate the output pin.
  In capture mode the output pin buffer is tristate.
- **Bit 2 – OUTx: OUTx signal on compare event**
  In compare mode the value of OUTx is sent to the the output unit.
- **Bit 1 – CCM: Capture/Compare mode select**
  When this bit is set compare mode is selected. When this bit is cleared capture mode is selected.
- **Bit 0 – IN: input signal latched on event (READONLY)**
  In compare mode this is the synchronized Input signal latched with EQU0 event.
  In capture mode this is the synchronized input signal latched with CAP0 event.

TimerA Cap/Comp Register0 Address: 0x40, IOpage: 0,1
TMRACCR0 Access Mode: Read/Write

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TM15	TM14	TM13	TM12	TM11	TM10	TM9	TM8	TM7	TM6	TM5	TM4	TM3	TM2	TM1	TM0

- **Bit 15..0 - CCR: Capture/Compare Register**
  In Capture mode these bits are the 16 bit timer value latched on a capture event.
  In Compare mode these bits are the 16 bit compare register values.

1705 Capture/Compare Module1

TimerA Cap/Comp Module1 Config Address: 0x43, IOpage: 0,1
TMRACCM1cfg Access Mode: Read/Write

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CCflg	0	0	0	EFfc	CMc1	CMc0	IMc1	IMc0	OMc2	OMc1	OMc0	OPinE	OUTx	CCM	IN

- **Bit 15 – CCflg: Capture or Compare event flag**
  In compare mode the CCflg bit is set when the EQU1 event occurs. In capture mode the CCflg bit is set when the capture event occurs. This bit is cleared by reading the register or writing a '0' to CCflg bit.
- **Bit 14..12 – RES: Reserved**
  Reserved.
- **Bit 11– EFfc: event flag flow-control**
  Setting the EFfc bit will enable blocking flow control on reads of the TMRACCM1cfg register. Writes to this register are never blocked. The reads will block while CCflg = '0'.
- **Bit 10,9 – CMc: Capture Module configuration**
  These bits are used to configure the Capture module. This configures what type of signal will trigger a capture event.

CMc1	CMc0	mode
0	0	Disabled
0	1	Rising edge
1	0	Falling edge
1	1	Rising or falling edge

- **Bit 8,7 – IMc: Input module configuration**
  These bits are used to configure the input module.

IMc1	IMc0	mode
0	0	disabled
0	1	inPin
1	0	zero
1	1	one

- **Bit 6,5,4 – OMc: Output module configuration.**
  These bits are used to set the mode of operation for OutputModule1.

OMc2	OMc1	OMc0	Mode	Description
0	0	0	Output	At EQU1, Opin1 = OUTx
0	0	1	Set	EQU1 sets Opin1
0	1	0	Reset	EQU1 resets Opin1
0	1	1	Toggle	EQU1 toggles Opin1
1	0	0	Toggle/Set	EQU1 toggles Opin1, EQU0 sets Opin1
1	0	1	Toggle/Reset	EQU1 toggles Opin1, EQU0 resets Opin1
1	1	0	Set/Reset	EQU1 sets Opin1, EQU0 reset Opin1
1	1	1	Reset/Set	EQU1 resets Opin1, EQU0 sets Opin1

- **Bit 3 – OPinE: Output pin enable**
  In compare mode setting this bit will enable the output pin buffer. Clearing this bit will tristate the output pin.
  In capture mode the output pin buffer is tristate.
- **Bit 2 – OUTx: OUTx signal on compare event**
  In compare mode the value of OUTx is sent to the the output unit.
- **Bit 1 – CCM: Capture/Compare mode select**
  When this bit is set compare mode is selected. When this bit is cleared capture mode is selected.
- **Bit 0 – IN: input signal latched on event (READONLY)**
  In compare mode this is the synchronized Input signal latched with EQU1 event.
  In capture mode this is the synchronized input signal latched with CAP1 event.

TimerA Cap/Comp Register1 Address: 0x42, IOpage: 0,1
TMRACCR1 Access Mode: Read/Write

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TM15	TM14	TM13	TM12	TM11	TM10	TM9	TM8	TM7	TM6	TM5	TM4	TM3	TM2	TM1	TM0

- **Bit 15..0 - CCR: Capture/Compare Register**
  In Capture mode these bits are the 16 bit timer value latched on a capture event.
  In Compare mode these bits are the 16 bit compare register values.

1706 Capture/Compare Module2

TimerA Cap/Comp Module2 Config Address: 0x45, IOpage: 0,1
TMRACCM2cfg Access Mode: Read/Write

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CCflg	0	0	0	EFfc	CMc1	CMc0	IMc1	IMc0	OMc2	OMc1	OMc0	OPinE	OUTx	CCM	IN

- **Bit 15 – CCflg: Capture or Compare event flag**
  In compare mode the CCflg bit is set when the EQU2 event occurs. In capture mode the CCflg bit is set when the capture event occurs. This bit is cleared by reading the register or writing a '0' to CCflg bit.
- **Bit 14..12 – RES: Reserved**
  Reserved.
- **Bit 11– EFfc: event flag flow-control**
  Setting the EFfc bit will enable blocking flow control on reads of the TMRACCM2cfg register. Writes to this register are never blocked. The reads will block while CCflg = '0'.
- **Bit 10,9 – CMc: Capture Module configuration**
  These bits are used to configure the Capture module. This configures what type of signal will trigger a capture event.

CMc1	CMc0	mode
0	0	Disabled
0	1	Rising edge
1	0	Falling edge
1	1	Rising or falling edge

- **Bit 8,7 – IMc: Input module configuration**
  These bits are used to configure the input module.

IMc1	IMc0	mode
0	0	disabled
0	1	inPin
1	0	zero
1	1	one

- **Bit 6,5,4 – OMc: Output module configuration.**
  These bits are used to set the mode of operation for OutputModule2.

OMc2	OMc1	OMc0	Mode	Description
0	0	0	Output	At EQU2, Opin2 = OUTx
0	0	1	Set	EQU2 sets Opin2
0	1	0	Reset	EQU2 resets Opin2
0	1	1	Toggle	EQU2 toggles Opin2
1	0	0	Toggle/Set	EQU2 toggles Opin2, EQU0 sets Opin2
1	0	1	Toggle/Reset	EQU2 toggles Opin2, EQU0 resets Opin2
1	1	0	Set/Reset	EQU2 sets Opin2, EQU0 reset Opin2
1	1	1	Reset/Set	EQU2 resets Opin2, EQU0 sets Opin2

- **Bit 3 – OPinE: Output pin enable**
  In compare mode setting this bit will enable the output pin buffer. Clearing this bit will tristate the output pin.
  In capture mode the output pin buffer is tristate.
- **Bit 2 – OUTx: OUTx signal on compare event**
  In compare mode the value of OUTx is sent to the the output unit.
- **Bit 1 – CCM: Capture/Compare mode select**
  When this bit is set compare mode is selected. When this bit is cleared capture mode is selected.
- **Bit 0 – IN: input signal latched on event (READONLY)**
  In compare mode this is the synchronized Input signal latched with EQU2 event.
  In capture mode this is the synchronized input signal latched with CAP2 event.

TimerA Cap/Comp Register2 Address: 0x44, IOpage: 0,1
TMRACCR2 Access Mode: Read/Write

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TM15	TM14	TM13	TM12	TM11	TM10	TM9	TM8	TM7	TM6	TM5	TM4	TM3	TM2	TM1	TM0

- **Bit 15..0 - CCR: Capture/Compare Register**
  In Capture mode these bits are the 16 bit timer value latched on a capture event.
  In Compare mode these bits are the 16 bit compare register values.

1707 Capture/Compare Module3

TimerA Cap/Comp Module3 Config Address: 0x47, IOpage: 0,1
TMRACCM3cfg Access Mode: Read/Write

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CCflg	0	0	0	EFfc	CMc1	CMc0	IMc1	IMc0	OMc2	OMc1	OMc0	OPinE	OUTx	CCM	IN

- **Bit 15 – CCflg: Capture or Compare event flag**
  In compare mode the CCflg bit is set when the EQ3 event occurs. In capture mode the CCflg bit is set when the capture event occurs. This bit is cleared by reading the register or writing a '0' to CCflg bit.
- **Bit 14..12 – RES: Reserved**
  Reserved.
- **Bit 11– EFfc: event flag flow-control**
  Setting the EFfc bit will enable blocking flow control on reads of the TMRACCM3cfg register. Writes to this register are never blocked. The reads will block while CCflg = '0'.
- **Bit 10,9 – CMc: Capture Module configuration**
  These bits are used to configure the Capture module. This configures what type of signal will trigger a capture event.

CMc1	CMc0	mode
0	0	Disabled
0	1	Rising edge
1	0	Falling edge
1	1	Rising or falling edge

- **Bit 8,7 – IMc: Input module configuration**
  These bits are used to configure the input module.

IMc1	IMc0	mode
0	0	disabled
0	1	inPin
1	0	zero
1	1	one

- **Bit 6,5,4 – OMc: Output module configuration.**
  These bits are used to set the mode of operation for OutputModule3.

OMc2	OMc1	OMc0	Mode	Description
0	0	0	Output	At EQU3, Opin3 = OUTx
0	0	1	Set	EQU3 sets Opin3
0	1	0	Reset	EQU3 resets Opin3
0	1	1	Toggle	EQU3 toggles Opin3
1	0	0	Toggle/Set	EQU3 toggles Opin3, EQU0 sets Opin3
1	0	1	Toggle/Reset	EQU3 toggles Opin3, EQU0 resets Opin3
1	1	0	Set/Reset	EQU3 sets Opin3, EQU0 reset Opin3
1	1	1	Reset/Set	EQU3 resets Opin3, EQU0 sets Opin3

- **Bit 3 – OPinE: Output pin enable**
  In compare mode setting this bit will enable the output pin buffer. Clearing this bit will tristate the output pin.
  In capture mode the output pin buffer is tristate.
- **Bit 2 – OUTx: OUTx signal on compare event**
  In compare mode the value of OUTx is sent to the the output unit.
- **Bit 1 – CCM: Capture/Compare mode select**
  When this bit is set compare mode is selected. When this bit is cleared capture mode is selected.
- **Bit 0 – IN: input signal latched on event (READONLY)**
  In compare mode this is the synchronized Input signal latched with EQU3 event.
  In capture mode this is the synchronized input signal latched with CAP3 event.

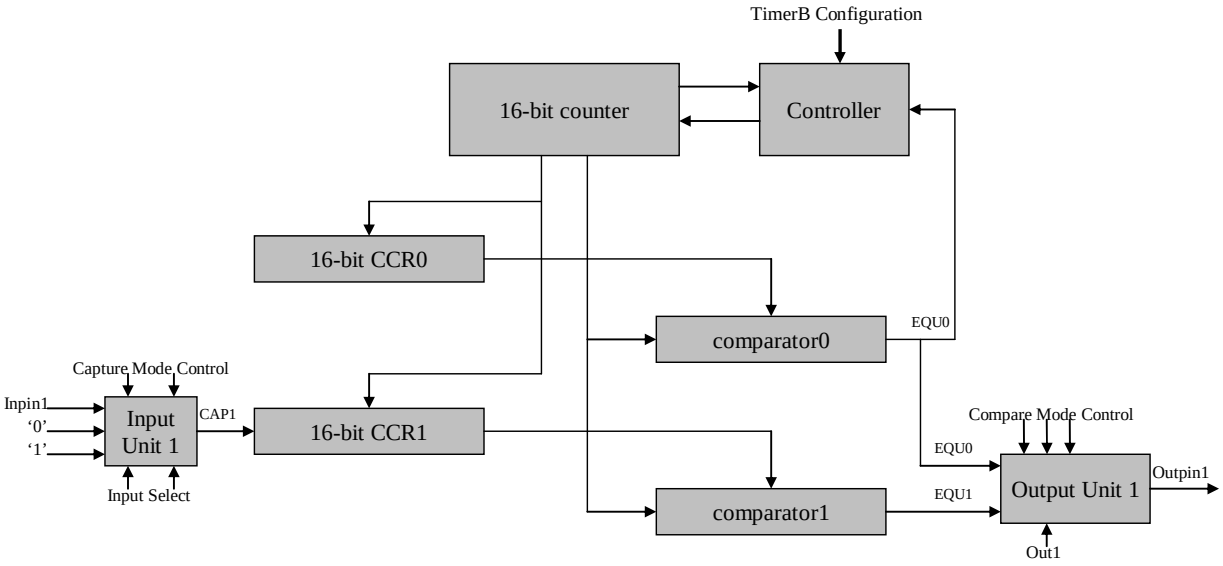
TimerA Cap/Comp Register3 Address: 0x46, IOpage: 0,1
TMRACCR3 Access Mode: Read/Write

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TM15	TM14	TM13	TM12	TM11	TM10	TM9	TM8	TM7	TM6	TM5	TM4	TM3	TM2	TM1	TM0

- **Bit 15..0 - CCR: Capture/Compare Register**
  In Capture mode these bits are the 16 bit timer value latched on a capture event.
  In Compare mode these bits are the 16 bit compare register values.

1800 TimerB

The TimerB is a general purpose 16 bit timer. The timer consists of multiple operating modes with selectable clock source, one configurable capture/compare registers, a configurable input module, and a configurable output module.
The timer can be used to generate output waveforms such as a PWM signals, capture input signals on compare, trigger an event on a input signal transition, or be used as a configurable source of time.
Input signals are synchronized with the XInC2 system clock.
TimerB is very similar to TimerA except that it only has two capture/compare registers and only one input and one output module.
The basic block diagram of the timer is shown below.



1801 Timer Mode Control

TimerB has 4 modes of operation:

- **Stopped**
- **Up**: count up to CR0 and restart at 0
- **Continuous Up**: timer counts up to 0xFFFF and restarts at 0
- **Up/Down**: timer counts continuously up to CR0 and back down to 0

1802 Capture / Compare Modules

There is one configurable capture/compare modules and a second compare module used for Up or Up/Down count mode.

Capture mode Capture mode is used to latch the timer value in the CCRx register when an event on an input signal occurs.

Compare mode The input signal may be latched with compare signal EQUx occurs. This mode is used to generate events at specific time intervals or to generate output signals.

1803 General Timer Configuration & Status

TimerB configuration Address: 0x48, IOpage: 0
TMRBcfg Access Mode: Read/Write

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OFflg	DIR	0	0	0	0	0	0	OFfc	CLK3	CLK2	CLK1	CLK0	Mcfg1	Mcfg0	E

- **Bit 15 – OFflg: Counter overflow flag**
  Offlg bit is set when the counter counts to zero. This bit is cleared by reading the register or writing a '0' to OFflg bit.
- **Bit 14 – DIR: Counter direction – read only**
  When the counter is incrementing then DIR bit is set. When the counter is decrementing then DIR bit is cleared
- **Bit 7– OFfc: Counter overflow flow-control**
  Setting the OFfc bit will enable blocking flow control on reads of the timerBcfg register.
- **Bit 6,5,4,3 – CLK: Timer Clock configuration**
  These bits are used to configure clock rate for the timer. The relationship between the timer clock and the system clock(Sclk) frequency is shown below.

CLK3	CLK2	CLK1	CLK0	CLK Frequency
0	0	0	0	Sclk / 2
0	0	0	1	Sclk / 4
0	0	1	0	Sclk / 8
0	0	1	1	Sclk / 16
0	1	0	0	Sclk / 32
0	1	0	1	Sclk / 64
0	1	1	0	Sclk / 128
0	1	1	1	Sclk / 256
1	0	0	0	Sclk / 512
1	0	0	1	Sclk / 1024
1	0	1	0	Sclk / 2048
1	0	1	1	Sclk / 4096
1	1	0	0	Sclk / 4096
1	1	0	1	Sclk / 4096
1	1	1	0	Sclk / 4096
1	1	1	1	Sclk / 4096

- **Bit 2,1 – Mcfg: Timer Mode configuration**
  Thse bits are used to configure the timer mode.

Mcfg1	Mcfg0	mode
0	0	Halt
0	1	Up
1	0	Continuous
1	1	Up/Down

- **Bit 0 – E: Timer enable**
  Setting this bit to one enables the timer.

TimerB 16-bit counter Address: 0x49, IOpage: 0
TMRBcount Access Mode: Read Only

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TM15	TM14	TM13	TM12	TM11	TM10	TM9	TM8	TM7	TM6	TM5	TM4	TM3	TM2	TM1	TM0

- **Bit 15..0 - TM: current counter**
  These bits represent the current value of the counter.

1804 Capture/Compare Module0

TimerB Cap/Comp Module0 Config
TMRBCCM0cfg

Address: 0x4B, IOpage: 0
Access Mode: Read/Write

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CCflg	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

- **Bit 15 – CCflg: Compare event flag**
  The CCflg bit is set when the EQ0 event occurs. This bit is cleared by reading the register or writing a '0' to CCflg bit.
- **Bit 14..0 – RES: Reserved**
  Reserved.

TimerB Cap/Comp Register0
TMRBCCR0

Address: 0x4A, IOpage: 0
Access Mode: Read only

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TM15	TM14	TM13	TM12	TM11	TM10	TM9	TM8	TM7	TM6	TM5	TM4	TM3	TM2	TM1	TM0

- **Bit 15..0 - CCR: Compare Register**
  These bits are the 16 bit compare register values.

1805 Capture/Compare Module1

TimerB Cap/Comp Module1 Config Address: 0x4D, IOpage: 0
TMRBCCM1cfg Access Mode: Read/Write

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CCflg	0	0	0	EFfc	CMc1	CMc0	IMc1	IMc0	OMc2	OMc1	OMc0	OPinE	OUTx	CCM	IN

- **Bit 15 – CCflg: Capture or Compare event flag**
  In compare mode the CCflg bit is set when the EQ1 event occurs. In capture mode the CCflg bit is set when the capture event occurs. This bit is cleared by reading the register or writing a '0' to CCflg bit.
- **Bit 12..14 – RES: Reserved**
  Reserved.
- **Bit 11– EFfc: event flag flow-control**
  Setting the EFfc bit will enable blocking flow control on reads of the TMRBCCM1cfg register. Writes to this register are never blocked. The reads will block while CCflg = '0'.
- **Bit 10,9 – CMc: Capture Module configuration**
  These bits are used to configure the Capture module. This configures what type of signal will trigger a capture event.

CMc1	CMc0	mode
0	0	Disabled
0	1	Rising edge
1	0	Falling edge
1	1	Rising or falling edge

- **Bit 8,7 – IMc: Input module configuration**
  These bits are used to configure the input module.

IMc1	IMc0	mode
0	0	disabled
0	1	inPin
1	0	zero
1	1	one

- **Bit 6,5,4 – OMc: Output module configuration.**
  These bits are used to set the mode of operation for OutputModule0.

OMc2	OMc1	OMc0	Mode	Description
0	0	0	Output	At EQU1, Opin1 = OUTx
0	0	1	Set	EQU1 sets Opin1
0	1	0	Reset	EQU1 resets Opin1
0	1	1	Toggle	EQU1 toggles Opin1
1	0	0	Toggle/Set	EQU1 toggles Opin1, EQU0 sets Opin1
1	0	1	Toggle/Reset	EQU1 toggles Opin1, EQU0 resets Opin1
1	1	0	Set/Reset	EQU1 sets Opin1, EQU0 reset Opin1
1	1	1	Reset/Set	EQU1 resets Opin1, EQU0 sets Opin1

- **Bit 3 – OPinE: Output pin enable**
  In compare mode setting this bit will enable the output pin buffer. Clearing this bit will tristate the output pin.
  In capture mode the output pin buffer is tristate.
- **Bit 2 – OUTx: OUTx signal on compare event**
  In compare mode the value of OUTx is sent to the the output unit.
- **Bit 1 – CCM: Capture/Compare mode select**
  When this bit is set compare mode is selected. When this bit is cleared capture mode is selected.
- **Bit 0 – IN: input signal latched on event (READONLY)**
  In compare mode this is the synchronized Input signal latched with EQU1 event.
  In capture mode this is the synchronized input signal latched with CAP1 event.

TimerB Cap/Comp Register1 Address: 0x4C, IOpage: 0
TMRBCCR1 Access Mode: Read only

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TM15	TM14	TM13	TM12	TM11	TM10	TM9	TM8	TM7	TM6	TM5	TM4	TM3	TM2	TM1	TM0

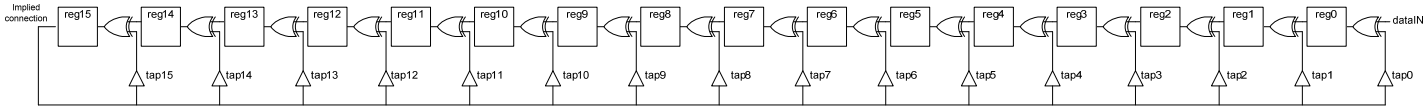
- **Bit 15..0 - CCR: Capture/Compare Register**
  In Capture mode these bits are the 16 bit timer value latched on a capture event.
  In Compare mode these bits are the 16 bit compare register values.

1900 Linear Feedback Shift Registers (LFSR)

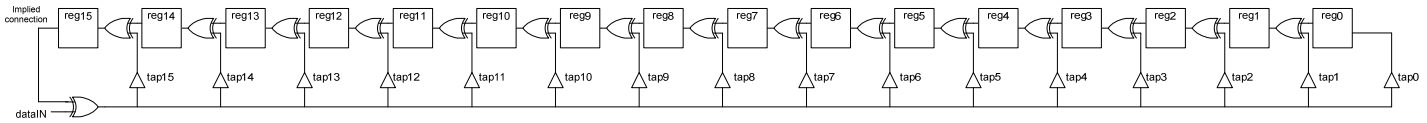
The Linear Feedback Shift Register(LFSR) can be used for polynomial multiplication and division over GF(2). LFSRs can be used to generate cyclic code encoders and decoders(CRC and Hamming codes), counters, compute parity and generate pn-sequences for scrambling, random numbers and direct sequence spread spectrum applications.

There are two identical LFSR peripherals. Each is individually configurable. The hardware implementation uses the Galois form. There are two configuration for variations on the feedback structure. The length(# of stages) of the LFSR can be configured from 1 to 16. The structure of the LFSR for 16 stages is shown below.
Data fed into the LFSR is fed in MSB first.

Feedback Configuration Mode0



Feedback Configuration Mode1



1901 LFSR0

LFSR configuration Address: 0x08, IOpage: 0/1
LFSR0cfg Access Mode: Read/Write

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	FB	DL3	DL2	DL1	DL0	L3	L2	L1	L0

- **Bit 15..9 - RES: Reserved**
  unused.
- **Bit 8 FB: LFSR feedback configuration mode**
  This bit is used to configure the feedback mode.
- **Bit 7..4 - DL: LFSR Data length configuration**
  These bits are used to configure the size of the data input to the LFSR, or the number of ticks to clock the LFSR.(# of ticks = DL + 1)
- **Bit 3..0 - L: LFSR length configuration**
  These bits are used to configure the length of the LFSR or which bit is the output or feedback. (# of FFs = L + 1)

LFSR tap configuration Address: 0x09, IOpage: 0/1
LFSR0tapcfg Access Mode: Read/Write

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TAP15	TAP14	TAP13	TAP12	TAP11	TAP10	TAP9	TAP8	TAP7	TAP6	TAP5	TAP4	TAP3	TAP2	TAP1	TAP0

- **Bit 15..0 - TAP: LFSR Tap Configuration**
  These bits configure the feedback taps.

LFSR register Address: 0x0A, IOpage: 0/1
LFSR0reg Access Mode: Read/Write

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
LR15	LR14	LR13	LR12	LR11	LR10	LR9	LR8	LR7	LR6	LR5	LR4	LR3	LR2	LR1	LR0

- **Bit 15..0 - LR: LFSR Register**
  These bits are the contents of the 16 bit LFSR register
  Note: Reading or Writing this register always accesses all 16 bits regardless of the configuration of LFSR0cfg.

LFSR data Address: 0x0B, IOpage: 0/1
LFSR0data Access Mode: Read/Write

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
LD15	LD14	LD13	LD12	LD11	LD10	LD9	LD8	LD7	LD6	LD5	LD4	LD3	LD2	LD1	LD0

- **Bit 15..0 - LD: LFSR data**
  These bits are the data bits to feed into the LFSR if written and the data bits that come out of the LFSR if read.
  Note! Data is fed in MSB first and data written must be left justified.

1902 LFSR1

LFSR configuration Address: 0x0C, IOpage: 0/1
LFSR1cfg Access Mode: Read/Write

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	FB	DL3	DL2	DL1	DL0	L3	L2	L1	L0

- **Bit 15..9 - RES: Reserved**
  unused.
- **Bit 8 FB: LFSR feedback configuration mode**
  This bit is used to configure the feedback mode.
- **Bit 7..4 - DL: LFSR Data length configuration**
  These bits are used to configure the size of the data input to the LFSR, or the number of ticks to clock the LFSR.(# of ticks = DL + 1)
- **Bit 3..0 - L: LFSR length configuration**
  These bits are used to configure the length of the LFSR or which bit is the output or feedback.

LFSR tap configuration Address: 0x0D, IOpage: 0/1
LFSR1tapcfg Access Mode: Read/Write

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TAP15	TAP14	TAP13	TAP12	TAP11	TAP10	TAP9	TAP8	TAP7	TAP6	TAP5	TAP4	TAP3	TAP2	TAP1	TAP0

- **Bit 15..0 - TAP: LFSR Tap Configuration**
  These bits configure the feedback taps.

LFSR register Address: 0x0E, IOpage: 0/1
LFSR1reg Access Mode: Read/Write

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
LR15	LR14	LR13	LR12	LR11	LR10	LR9	LR8	LR7	LR6	LR5	LR4	LR3	LR2	LR1	LR0

- **Bit 15..0 - LR: LFSR Register**
  These bits are the contents of the 16 bit LFSR register.
  Note! Reading or Writing this register always accesses all 16 bits regardless of the configuration of LFSR1cfg.

LFSR data Address: 0x0F, IOpage: 0/1
LFSR1data Access Mode: Read/Write

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
LD15	LD14	LD13	LD12	LD11	LD10	LD9	LD8	LD7	LD6	LD5	LD4	LD3	LD2	LD1	LD0

- **Bit 15..0 - LD: LFSR data**
  These bits are the data bits to feed into the LFSR if written and the data bits that come out of the LFSR if read.
  Note! Data is fed in MSB first and data written must be left justified.

2000 Accumulators

Two independent accumulators are provided to allow for summing or subtraction of 32 bit operands with a 32 bit result.
The operands are assumed to be of integer 32.0 format.
The accumulators also support a signed 16 bit saturated mode. In this mode the accumulator results will be saturated to -32768 -> + 32767. (16.0 format)
The accumulation is done with a nibble-serial algorithm. The accumulation takes 9 system clock ticks to complete.
Read and write access to all I/O registers is blocked while the accumulator is busy doing an accumulation.

2001 Accumulator0

accumulator0 Configuration Address: 0x68, IOpage: 1
Accum0cfg Access Mode: Read/Write

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SOFL	OFL	OF16	0	0	0	0	0	0	0	0	0	0	Aclr	OPM	SAT

- **Bit 15 - SOFL: sticky overflow flag**
  The SOFL bit will be set when an accumulation results in a 32bit overflow.
  The SOFL flag is cleared by writing zero to OFL.
- **Bit 14 - OFL: overflow flag**
  The OFL bit will be set when a single addition results in a 32bit overflow.
  This OFL flag is cleared after starting another accumulation operation or by writing zero to OFL.
- **Bit 13 – OF16: signed 16 bit overflow flag (read only)**
  The OF16 bit will be set when the accumulator contents contains a signed value larger then can be represented in 16 bits. This bit is clear if Accumhigh = X"0000" or X"FFFF" else it is set.
- **Bit 2 – Aclr: Accumulator Clear**
  Writing a one to the Accumulator Clear bit will clear the 32 bits of the accumulator.
- **Bit 1 – OPM: 16 bit signed operand mode**
  Writing a one to the to the OPM bit will put the accumulator into 16 bit signed operand mode. In this mode the Accumaddsubhigh register(most significant word of the operand to add or subtract) is ignored and the 16 bit value in Accumaddlow or Accum0sublow is interpreted as a signed 16 bit value and is sign extended to the Accumaddsubhigh value.
  When the OPM bit is zero the accumulator operates in 32 bit signed operand mode.
- **Bit 0 - SAT: saturation mode**
  When the SAT bit is set(one), the accumulator will operate in signed 16 bit saturated mode. When the SAT bit is cleared(zero), the accumulator will operate in signed 32 bit mode.
  In saturation mode the accumulator retains the 32 bit result but reading the accumulator will give a 16 bit saturated result. This enables both results to be read as an option.

Accumulator0 high word Address: 0x69, IOpage: 1
Accum0high Access Mode: Read/Write

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AH15	AH14	AH13	AH12	AH11	AH10	AH9-	AH8	AH7	AH6	AH5	AH4	AH3	AH2	AH1	AH0

- **Bit 15..0 - AH: accumulator high**
  These bits are the most significant word of the accumulator. Reading this will block until the accumulation is complete.
  Writing this will block until the previous accumulation is complete.

Accumulator0 low word Address: 0x6A, IOpage: 1
Accum0low Access Mode: Read/Write

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AL15	AL14	AL13	AL12	AL11	AL10	AL9	AL8	AL7	AL6	AL5	AL4	AL3	AL2	AL1	AL0

- **Bit 15..0 - AL: accumulator low**
  These bits are the least significant word of the accumulator. Reading this will block until the accumulation is complete.
  Writing this will block until the previous accumulation is complete.

accumulator0 add/sub high word Address: 0x6B, IOpage: 1
Accum0addsubhigh Access Mode: Write only

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ASH15	ASH14	ASH13	ASH12	ASH11	ASH10	ASH9	ASH8	ASH7	ASH6	ASH5	ASH4	ASH3	ASH2	ASH1	ASH0

- **Bit 15..0 - ASH: accumulator add/sub high**
  These bits are the most significant word of the value to be added to or subtracted from the current accumulator value.
  Writing this will block until the previous accumulation is complete.

accumulator0 add low word Address: 0x6C, IOpage: 1
Accum0addlow Access Mode: Write only

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AL15	AL14	AL13	AL12	AL11	AL10	AL9	AL8	AL7	AL6	AL5	AL4	AL3	AL2	AL1	AL0

- **Bit 15..0 - AL: accumulator add low**
  These bits are the least significant word of the value to be added to the current accumulator value.
  Writing this will block until the previous accumulation is complete.
  Writing to this will start the accumulation process.

accumulator0 sub low word Address: 0x6D, IOpage: 1
Accum0sublow Access Mode: Write only

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SL15	SL14	SL13	SL12	SL11	SL10	SL9	SL8	SL7	SL6	SL5	SL4	SL3	SL2	SL1	SL0

- **Bit 15..0 - SL: accumulator sub low**
  These bits are the least significant word of the value to be subtracted from the current accumulator value.
  Writing this will block until the previous accumulation is complete.
  Writing to this will start the accumulation process.

2002 Accumulator1

Accumulator1 Configuration Address: 0x6E, IOpage: 1
Accum1cfg Access Mode: Read/Write

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SOFL	OFL	OF16	0	0	0	0	0	0	0	0	0	0	Aclr	OPM	SAT

- **Bit 15 - SOFL: sticky overflow flag – not complete**
  The SOFL bit will be set when an accumulation results in a 32bit overflow.
  The SOFL flag is cleared by writing zero to OFL or writing to either accum0high or accum0low.
- **Bit 14 - OFL: overflow flag – not complete**
  The OFL bit will be set when a single addition results in a 32bit overflow.
  This OFL flag is cleared after starting another accumulation operation or by writing zero to OFL.
- **Bit 13 – OF16: signed 16 bit overflow flag (read only)**
  The OF16 bit will be set when the accumulator contents contains a signed value larger then can be represented in 16 bits. This bit is clear if Accumhigh = X"0000" or X"FFFF" else it is set.
- **Bit 2 – Aclr: Accumulator Clear**
  Writing a one to the Accumulator Clear bit will clear the 32 bits of the accumulator.
- **Bit 1 – OPM: 16 bit signed operand mode**
  Writing a one to the to the OPM bit will put the accumulator into 16 bit signed operand mode. In this mode the Accumaddsubhigh register(most significant word of the operand to add or subtract) is ignored and the 16 bit value in Accumaddlow or Accum0sublow is interpreted as a signed 16 bit value and is sign extended to the Accumaddsubhigh value.
  When the OPM bit is zero the accumulator operates in 32 bit signed operand mode.
- **Bit 0 - SAT: saturation mode**
  When the SAT bit is set(one), the accumulator will operate in signed 16 bit saturated mode. When the SAT bit is cleared(zero), the accumulator will operate in signed 32 bit mode.
  In saturation mode the accumulator retains the 32 bit result but reading the accumulator will give a 16 bit saturated result. This enables both results to be read as an option.

Accumulator1 high word Address: 0x6F, IOpage: 1
Accum1high Access Mode: Read/Write

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AH15	AH14	AH13	AH12	AH11	AH10	AH9-	AH8	AH7	AH6	AH5	AH4	AH3	AH2	AH1	AH0

- **Bit 15..0 - AH: accumulator high**
  These bits are the most significant word of the accumulator. Reading this will block until the accumulation is complete.
  Writing this will block until the previous accumulation is complete.

Accumulator1 low word Address: 0x70, IOpage: 1
Accum1low Access Mode: Read/Write

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AL15	AL14	AL13	AL12	AL11	AL10	AL9	AL8	AL7	AL6	AL5	AL4	AL3	AL2	AL1	AL0

- **Bit 15..0 - AL: accumulator low**
  These bits are the least significant word of the accumulator. Reading this will block until the accumulation is complete.
  Writing this will block until the previous accumulation is complete.

Accumulator1 add/sub high word Address: 0x71, IOpage: 1
Accum1addsubhigh Access Mode: Write only

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ASH15	ASH14	ASH13	ASH12	ASH11	ASH10	ASH9	ASH8	ASH7	ASH6	ASH5	ASH4	ASH3	ASH2	ASH1	ASH0

- **Bit 15..0 - ASH: accumulator add/sub high**
  These bits are the most significant word of the value to be added to or subtracted from the current accumulator value.
  Writing this will block until the previous accumulation is complete.

Accumulator1 add low word Address: 0x72, IOpage: 1
Accum1addlow Access Mode: Write only

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AL15	AL14	AL13	AL12	AL11	AL10	AL9	AL8	AL7	AL6	AL5	AL4	AL3	AL2	AL1	AL0

- **Bit 15..0 - AL: accumulator add low**
  These bits are the least significant word of the value to be added to the current accumulator value.
  Writing this will block until the previous accumulation is complete.
  Writing to this will start the accumulation process.

Accumulator1 sub low word Address: 0x73, IOpage: 1
Accum1sublow Access Mode: Write only

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SL15	SL14	SL13	SL12	SL11	SL10	SL9	SL8	SL7	SL6	SL5	SL4	SL3	SL2	SL1	SL0

- **Bit 15..0 - SL: accumulator sub low**
  These bits are the least significant word of the value to be subtracted from the current accumulator value.
  Writing this will block until the previous accumulation is complete.
  Writing to this will start the accumulation process.

2100 ADPCM Difference Quantizers

Four independent ADPCM difference quantizers are provided to allow for quantization of signed 16 bit audio samples to 1->8 bit values.
The quantized output for N-bit quantization consists of a sign bit followed by N-1 bits of quantized value.
The quantizer takes a sample, prediction and a step size and then produces a quantized output based on the difference between the sample and prediction and the step size.

2101 ADPCMQuantizer0

Quantizer0 Configuration Address: 0x48, IOpage: 1
Quant0cfg Access Mode: Read/Write

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	BD2	BD1	**BD0**

- **Bit 2..0 - BD: quantizer output bit depth**
  These bits determine the bit depth of the output of the quantization.
  The following table shows the configuration.

BD2	BD1	BD0	Bit depth
0	0	0	1
0	0	1	2
0	1	0	3
0	1	1	4
1	0	0	5
1	0	1	6
1	1	0	7
1	1	1	8

Quantizer0 step Address: 0x4A, IOpage: 1
Quant0step Access Mode: Write only

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ST15	ST14	ST13	ST12	ST11	ST10	ST9	ST8	ST7	ST6	ST5	ST4	ST3	ST2	ST1	ST0

- **Bit 15..0 - ST: quantizer step**
  These bits are the 16 bit unsigned quantizer step size.
  Writing this will block until the previous quantization is complete.
  Writing to this will start the quantization process.

Quantizer0 sample Address: 0x4B, IOpage: 1
Quant0sam Access Mode: Write only

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ST15	ST14	ST13	ST12	ST11	ST10	ST9	ST8	ST7	ST6	ST5	ST4	ST3	ST2	ST1	ST0

- **Bit 15..0 - ST: quantizer sample**
  These bits are the 16 bit signed sample value to quantize.
  Writing this will block until the previous quantization is complete.

Quantizer0 predictor Address: 0x49, IOpage: 1
Quant0pred Access Mode: Write only

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ST15	ST14	ST13	ST12	ST11	ST10	ST9	ST8	ST7	ST6	ST5	ST4	ST3	ST2	ST1	ST0

- **Bit 15..0 - ST: predictor sample**
  These bits are the 16 bit signed prediction value for the current sample.
  Writing this will block until the previous quantization is complete.
  After a quantization is complete the value of the predictor will become the same value as the quantizer decoded register.

Quantizer0 decoded
Quant0dec

Address: 0x49, IOpage: 1
Access Mode: Read only

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ST15	ST14	ST13	ST12	ST11	ST10	ST9	ST8	ST7	ST6	ST5	ST4	ST3	ST2	ST1	ST0

- **Bit 15..0 - ST: decoded sample**
  These bits are the 16 bit signed “decoded” value for the current sample and quantizer bit depth.
  Reading this will block until the previous quantization is complete.
  The internal quantizer predictor will use this value for the next quantization if no new predictor is written to adpcmQuant0pred.

Quantizer0 delta
Quant0delta

Address: 0x4A, IOpage: 1
Access Mode: Read only

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ST15	ST14	ST13	ST12	ST11	ST10	ST9	ST8	ST7	ST6	ST5	ST4	ST3	ST2	ST1	ST0

- **Bit 15..0 - D: delta**
  These bits are the 1->8 bit quantized delta output.
  Reading this will block until the previous quantization is complete.

2102 ADPCMQuantizer1

Quantizer1 Configuration Address: 0x4C, IOpage: 1
Quant1cfg Access Mode: Read/Write

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	BD2	BD1	**BD0**

- **Bit 2..0 - BD: quantizer output bit depth**
  These bits determine the bit depth of the output of the quantization.
  The following table shows the configuration.

BD2	BD1	BD0	Bit depth
0	0	0	1
0	0	1	2
0	1	0	3
0	1	1	4
1	0	0	5
1	0	1	6
1	1	0	7
1	1	1	8

Quantizer1 step Address: 0x4E, IOpage: 1
Quant1step Access Mode: Write only

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ST15	ST14	ST13	ST12	ST11	ST10	ST9	ST8	ST7	ST6	ST5	ST4	ST3	ST2	ST1	ST0

- **Bit 15..0 - ST: quantizer step**
  These bits are the 16 bit unsigned quantizer step size.
  Writing this will block until the previous quantization is complete.
  Writing to this will start the quantization process.

Quantizer1 sample Address: 0x4F, IOpage: 1
Quant1sam Access Mode: Write only

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ST15	ST14	ST13	ST12	ST11	ST10	ST9	ST8	ST7	ST6	ST5	ST4	ST3	ST2	ST1	ST0

- **Bit 15..0 - ST: quantizer sample**
  These bits are the 16 bit signed sample value to quantize.
  Writing this will block until the previous quantization is complete.

Quantizer1 predictor Address: 0x4D, IOpage: 1
Quant1pred Access Mode: Write only

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ST15	ST14	ST13	ST12	ST11	ST10	ST9	ST8	ST7	ST6	ST5	ST4	ST3	ST2	ST1	ST0

- **Bit 15..0 - ST: predictor sample**
  These bits are the 16 bit signed prediction value for the current sample.
  Writing this will block until the previous quantization is complete.
  After a quantization is complete the value of the predictor will become the same value as the quantizer decoded register.

Quantizer1 decoded Address: 0x49, IOpage: 1
Quant1dec Access Mode: Read only

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ST15	ST14	ST13	ST12	ST11	ST10	ST9	ST8	ST7	ST6	ST5	ST4	ST3	ST2	ST1	ST0

- **Bit 15..0 - ST: decoded sample**
  These bits are the 16 bit signed “decoded” value for the current sample and quantizer bit depth.
  Reading this will block until the previous quantization is complete.
  The internal quantizer predictor will use this value for the next quantization if no new predictor is written to adpcmQuant0pred.

Quantizer1 delta Address: 0x4E, IOpage: 1
Quant1delta Access Mode: Read only

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ST15	ST14	ST13	ST12	ST11	ST10	ST9	ST8	ST7	ST6	ST5	ST4	ST3	ST2	ST1	ST0

- **Bit 15..0 - D: delta**
  These bits are the 1->8 bit quantized delta output.
  Reading this will block until the previous quantization is complete.

2103 ADPCMQuantizer2

Quantizer2 Configuration
Quant2cfg

Address: 0x50, IOpage: 1
Access Mode: Read/Write

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	BD2	BD1	**BD0**

- **Bit 2..0 - BD: quantizer output bit depth**
  These bits determine the bit depth of the output of the quantization.
  The following table shows the configuration.

BD2	BD1	BD0	Bit depth
0	0	0	1
0	0	1	2
0	1	0	3
0	1	1	4
1	0	0	5
1	0	1	6
1	1	0	7
1	1	1	8

Quantizer2 step
Quant2step

Address: 0x52, IOpage: 1
Access Mode: Write only

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ST15	ST14	ST13	ST12	ST11	ST10	ST9	ST8	ST7	ST6	ST5	ST4	ST3	ST2	ST1	ST0

- **Bit 15..0 - ST: quantizer step**
  These bits are the 16 bit unsigned quantizer step size.
  Writing this will block until the previous quantization is complete.
  Writing to this will start the quantization process.

Quantizer2 sample
Quant2sam

Address: 0x53, IOpage: 1
Access Mode: Write only

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ST15	ST14	ST13	ST12	ST11	ST10	ST9	ST8	ST7	ST6	ST5	ST4	ST3	ST2	ST1	ST0

- **Bit 15..0 - ST: quantizer sample**
  These bits are the 16 bit signed sample value to quantize.
  Writing this will block until the previous quantization is complete.

Quantizer2 predictor
Quant2pred

Address: 0x51, IOpage: 1
Access Mode: Write only

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ST15	ST14	ST13	ST12	ST11	ST10	ST9	ST8	ST7	ST6	ST5	ST4	ST3	ST2	ST1	ST0

- **Bit 15..0 - ST: predictor sample**
  These bits are the 16 bit signed prediction value for the current sample.
  Writing this will block until the previous quantization is complete.
  After a quantization is complete the value of the predictor will become the same value as the quantizer decoded register.

Quantizer2 decoded Address: 0x51, IOpage: 1
Quant2dec Access Mode: Read only

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ST15	ST14	ST13	ST12	ST11	ST10	ST9	ST8	ST7	ST6	ST5	ST4	ST3	ST2	ST1	ST0

- **Bit 15..0 - ST: decoded sample**
  These bits are the 16 bit signed “decoded” value for the current sample and quantizer bit depth.
  Reading this will block until the previous quantization is complete.
  The internal quantizer predictor will use this value for the next quantization if no new predictor is written to adpcmQuant0pred.

Quantizer2 delta Address: 0x52, IOpage: 1
Quant2delta Access Mode: Read only

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ST15	ST14	ST13	ST12	ST11	ST10	ST9	ST8	ST7	ST6	ST5	ST4	ST3	ST2	ST1	ST0

- **Bit 15..0 - D: delta**
  These bits are the 1->8 bit quantized delta output.
  Reading this will block until the previous quantization is complete.

2104 ADPCMQuantizer3

Quantizer3 Configuration Address: 0x54, IOpage: 1
Quant3cfg Access Mode: Read/Write

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	BD2	BD1	**BD0**

- **Bit 2..0 - BD: quantizer output bit depth**
  These bits determine the bit depth of the output of the quantization.
  The following table shows the configuration.

BD2	BD1	BD0	Bit depth
0	0	0	1
0	0	1	2
0	1	0	3
0	1	1	4
1	0	0	5
1	0	1	6
1	1	0	7
1	1	1	8

Quantizer3 step Address: 0x56, IOpage: 1
Quant3step Access Mode: Write only

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ST15	ST14	ST13	ST12	ST11	ST10	ST9	ST8	ST7	ST6	ST5	ST4	ST3	ST2	ST1	ST0

- **Bit 15..0 - ST: quantizer step**
  These bits are the 16 bit unsigned quantizer step size.
  Writing this will block until the previous quantization is complete.
  Writing to this will start the quantization process.

Quantizer3 sample Address: 0x57, IOpage: 1
Quant3sam Access Mode: Write only

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ST15	ST14	ST13	ST12	ST11	ST10	ST9	ST8	ST7	ST6	ST5	ST4	ST3	ST2	ST1	ST0

- **Bit 15..0 - ST: quantizer sample**
  These bits are the 16 bit signed sample value to quantize.
  Writing this will block until the previous quantization is complete.

Quantizer3 predictor Address: 0x55, IOpage: 1
Quant3pred Access Mode: Write only

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ST15	ST14	ST13	ST12	ST11	ST10	ST9	ST8	ST7	ST6	ST5	ST4	ST3	ST2	ST1	ST0

- **Bit 15..0 - ST: predictor sample**
  These bits are the 16 bit signed prediction value for the current sample.
  Writing this will block until the previous quantization is complete.
  After a quantization is complete the value of the predictor will become the same value as the quantizer decoded register.

Quantizer3 decoded Address: 0x55, IOpage: 1
Quant3dec Access Mode: Read only

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ST15	ST14	ST13	ST12	ST11	ST10	ST9	ST8	ST7	ST6	ST5	ST4	ST3	ST2	ST1	ST0

- **Bit 15..0 - ST: decoded sample**
  These bits are the 16 bit signed “decoded” value for the current sample and quantizer bit depth.
  Reading this will block until the previous quantization is complete.
  The internal quantizer predictor will use this value for the next quantization if no new predictor is written to adpcmQuant0pred.

Quantizer3 delta Address: 0x56, IOpage: 1
Quant3delta Access Mode: Read only

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ST15	ST14	ST13	ST12	ST11	ST10	ST9	ST8	ST7	ST6	ST5	ST4	ST3	ST2	ST1	ST0

- **Bit 15..0 - D: delta**
  These bits are the 1->8 bit quantized delta output.
  Reading this will block until the previous quantization is complete.

2200 ADPCM Inverse Difference Quantizers

Four independent ADPCM Inverse difference quantizers are provided to allow for decoding of 1->8 bit values to 16 bit difference signals.
The difference signal is obtained from the code word and quantization set size scale factor.

2201 ADPCMInverseQuantizer0

InverseQuantizer0 Configuration Address: 0x58, IOpage: 1
InvQuant0cfg Access Mode: Read/Write

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	BD2	BD1	**BD0**

- **Bit 2..0 - BD: quantizer input bit depth**
  These bits determine the bit depth of the input of the inverse quantization.
  The following table shows the configuration.

BD2	BD1	BD0	Bit depth
0	0	0	1
0	0	1	2
0	1	0	3
0	1	1	4
1	0	0	5
1	0	1	6
1	1	0	7
1	1	1	8

InverseQuantizer0 step Address: 0x5A, IOpage: 1
InvQuant0step Access Mode: Write only

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
QT15	QT14	QT13	QT12	QT11	QT10	QT9	QT8	QT7	QT6	QT5	QT4	QT3	QT2	QT1	QT0

- **Bit 15..0 - QT: quantizer step**
  These bits are the 16 bit unsigned quantizer step size.
  Writing this will block until the previous inverse quantization is complete.
  Writing to this will start the inverse quantization process.

InverseQuantizer0 delta Address: 0x59, IOpage: 1
InvQuant0delta Access Mode: Write only

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0

- **Bit 15..0 - D: delta**
  These bits are the encoded quantized data.
  Writing this will block until the previous inverse quantization is complete.

InverseQuantizer0 difference Address: 0x59, IOpage: 1
InvQuant0dif Access Mode: Read only

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0

- **Bit 15..0 - D: difference**
  These bits are unsigned 16 bit difference.
  Reading this will block until the previous inverse quantization is complete.
  Note: the sign of the input delta is the sign of the difference

2202 ADPCMInverseQuantizer1

InverseQuantizer1 Configuration Address: 0x5C, IOpage: 1
InvQuant1cfg Access Mode: Read/Write

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	BD2	BD1	**BD0**

- **Bit 2..0 - BD: quantizer input bit depth**
  These bits determine the bit depth of the input of the inverse quantization.
  The following table shows the configuration.

BD2	BD1	BD0	Bit depth
0	0	0	1
0	0	1	2
0	1	0	3
0	1	1	4
1	0	0	5
1	0	1	6
1	1	0	7
1	1	1	8

InverseQuantizer1 step Address: 0x5E, IOpage: 1
InvQuant1step Access Mode: Write only

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
QT15	QT14	QT13	QT12	QT11	QT10	QT9	QT8	QT7	QT6	QT5	QT4	QT3	QT2	QT1	QT0

- **Bit 15..0 - QT: quantizer step**
  These bits are the 16 bit unsigned quantizer step size.
  Writing this will block until the previous inverse quantization is complete.
  Writing to this will start the inverse quantization process.

InverseQuantizer1 delta Address: 0x5D, IOpage: 1
InvQuant1delta Access Mode: Write only

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0

- **Bit 15..0 - D: delta**
  These bits are the encoded quantized data.
  Writing this will block until the previous inverse quantization is complete.

InverseQuantizer1 difference Address: 0x5D, IOpage: 1
InvQuant1dif Access Mode: Read only

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0

- **Bit 15..0 - D: difference**
  These bits are unsigned 16 bit difference.
  Reading this will block until the previous inverse quantization is complete.
  Note: the sign of the input delta is the sign of the difference

2203 ADPCMInverseQuantizer2

InverseQuantizer2 Configuration Address: 0x60, IOpage: 1
InvQuant2cfg Access Mode: Read/Write

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	BD2	BD1	**BD0**

- **Bit 2..0 - BD: quantizer input bit depth**
  These bits determine the bit depth of the input of the inverse quantization.
  The following table shows the configuration.

BD2	BD1	BD0	Bit depth
0	0	0	1
0	0	1	2
0	1	0	3
0	1	1	4
1	0	0	5
1	0	1	6
1	1	0	7
1	1	1	8

InverseQuantizer2 step Address: 0x62, IOpage: 1
InvQuant2step Access Mode: Write only

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
QT15	QT14	QT13	QT12	QT11	QT10	QT9	QT8	QT7	QT6	QT5	QT4	QT3	QT2	QT1	QT0

- **Bit 15..0 - QT: quantizer step**
  These bits are the 16 bit unsigned quantizer step size.
  Writing this will block until the previous inverse quantization is complete.
  Writing to this will start the inverse quantization process.

InverseQuantizer2 delta Address: 0x61, IOpage: 1
InvQuant2delta Access Mode: Write only

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0

- **Bit 15..0 - D: delta**
  These bits are the encoded quantized data.
  Writing this will block until the previous inverse quantization is complete.

InverseQuantizer2 difference Address: 0x61, IOpage: 1
InvQuant2dif Access Mode: Read only

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0

- **Bit 15..0 - D: difference**
  These bits are unsigned 16 bit difference.
  Reading this will block until the previous inverse quantization is complete.
  Note: the sign of the input delta is the sign of the difference

2204 ADPCMInverseQuantizer3

InverseQuantizer3 Configuration Address: 0x64, IOpage: 1
InvQuant3cfg Access Mode: Read/Write

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	BD2	BD1	**BD0**

- **Bit 2..0 - BD: quantizer input bit depth**
  These bits determine the bit depth of the input of the inverse quantization.
  The following table shows the configuration.

BD2	BD1	BD0	Bit depth
0	0	0	1
0	0	1	2
0	1	0	3
0	1	1	4
1	0	0	5
1	0	1	6
1	1	0	7
1	1	1	8

InverseQuantizer3 step Address: 0x66, IOpage: 1
InvQuant3step Access Mode: Write only

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
QT15	QT14	QT13	QT12	QT11	QT10	QT9	QT8	QT7	QT6	QT5	QT4	QT3	QT2	QT1	QT0

- **Bit 15..0 - QT: quantizer step**
  These bits are the 16 bit unsigned quantizer step size.
  Writing this will block until the previous inverse quantization is complete.
  Writing to this will start the inverse quantization process.

InverseQuantizer3 delta Address: 0x65, IOpage: 1
InvQuant3delta Access Mode: Write only

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0

- **Bit 15..0 - D: delta**
  These bits are the encoded quantized data.
  Writing this will block until the previous inverse quantization is complete.

InverseQuantizer3 difference Address: 0x65, IOpage: 1
InvQuant3dif Access Mode: Read only

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0

- **Bit 15..0 - D: difference**
  These bits are unsigned 16 bit difference.
  Reading this will block until the previous inverse quantization is complete.
  Note: the sign of the input delta is the sign of the difference

2300 Serial Peripheral Interface

The serial peripheral interface (SPI) is a high-speed synchronous serial port. It allows a serial bit stream of 8 bits to be shifted into and out of the device attached. Data is clocked in and out most significant bit first.

XInC2 contains two SPI peripherals. SPI0 and SPI1 function identically but they have different uses in a XInC2 system. SPI0 is an integral part of the XPD port. In the normal case a developer should use SPI1 to communicate to additional SPI devices. This allows the XPD port to be dedicated to debug and diagnostics.

The SPI interface normally requires 3 or 4 pins. There is a clock(SCK), and two data pins: MISO(master in slave out) and MOSI(master out slave in). A chip select pin is also normally required to select one of multiple slave devices attached.

Each serial port has a control register, status register, and data register.
The SPI can operate in two modes; Master or Slave.

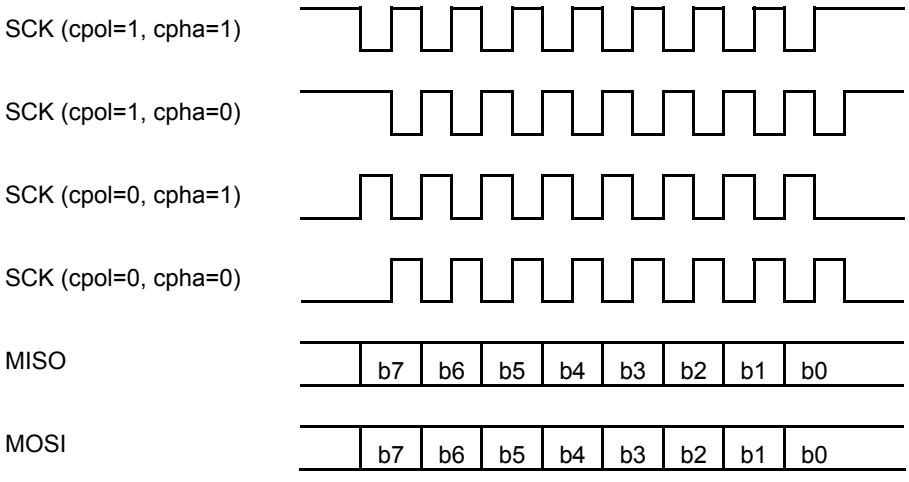
Master Mode

In Master Mode the maximum SCK clock rate is Sclk / 2. When the SPI is enabled in master mode, the data direction of the pins are configured as: SCK and MOSI pins are outputs and the MISO pin is an input.

When the SPI is configured as master, data is clocked using one of four programmable clock phase and polarity variations.

CPOL	CPHA	setup SCK edge	latch SCK edge	SCK idle state
0	0	falling	rising	low
0	1	rising	falling	low
1	0	rising	falling	high
1	1	falling	rising	high

The relationship between SCK, MISO, and MOSI is shown below for the four phase/polarity configurations.



Writing to the data register starts data transmission. When phase(CPHA) = 0, the MSB is output when a thread loads the transmitted data. When phase = 1, the MSB is output on the first SCK edge.

Slave Mode In Slave Mode the slave ignores communication unless the SEN pin is low. If SEN is brought high during a transaction the communication is aborted and the data received is invalid.

In Slave mode the SCK and MOSI pins are inputs. The MISO pin is switched to an output while the SEN pin is low and an input when SEN pin is high.

In Slave Mode the shift register is asynchronous to Sclk so the maximum SCK frequency is not dependent on Sclk.

In slave mode the polarity and phase control bits are ignored. The clocking mode is hard coded as phase = 1 and polarity = 1;

I/O blocking The data registers of the SPI peripheral utilize peripheral blocking.

Master Mode: A thread writing to SPItx or reading from SPIrx will be blocked while the SPI shift register is in the middle of a transaction.

Slave Mode: A thread reading from SPIrx will be blocked while the SPI shift register is in the middle of a transaction and the SEN pin is low. Writing to the SPItx register will never be blocked.

2301 SPI0

SPI0 is an integral part of the XInC2 programming/diagnostics(XPD) port. The I/O pins associated with the SCK0,MISO0,MOSI0, and SEN0 pins are dedicated to SPI0. The I/O cells can be individually configured in the SCX peripheral.

SPI0 Configuration
SPI0cfg

Address: 0x35, IOpage: 0/1
Access Mode: Read/Write

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CS	0	0	0	0	0	0	0	OD	CPOL	CPHA	SCK2	SCK1	SCK0	MSTR	SPE

- **Bit 15 - CS: Slave Select state**
  This is a read only bit. This bit is the registered value present at CS0 pin. In Slave mode if the CS0 pin goes high in the middle of a transaction the internal shift registers are reset and the data received is invalid.
- **Bit 14..8 - Res: Reserved**
  These bits are reserved.
- **Bit 7 - OD: Output Disable**
  If this bit is set(one), the output pin (either MOSI or MISO, depending whether the SPI is master or slave) will be under the control of the GPIO port. The SPI transmit pin is effectively disabled. If this bit is clear(zero), the pin is a SPI data output.
- **Bit 6 - CPOL: Clock Polarity**
  The clock polarity can be configured for master mode only. In slave mode the polarity is set at 1 and this bit is ignored.
  When this bit is set(one), SCK idles high. When cleared(zero), SCK idles low.
- **Bit 5 - CPHA: Clock Phase**
  The clock phase can be configured for master mode only. This bit controls the clock/data phase relationship. In slave mode the polarity is set at 1 and this bit is ignored.
- **Bit 4,3,2 - SCK2,SCK1,SCK0: SPI Clock Rate select**
  These three bits control the SPI clock(SCK) rate of the device when configured in master mode. In slave mode these bits are ignored. The relationship between SCK and the system clock(Sclk) frequency is shown below:

SCK2	SCK1	SCK0	SCK Frequency
0	0	0	Sclk / 2
0	0	1	Sclk / 4
0	1	0	Sclk / 8
0	1	1	Sclk / 16
1	0	0	Sclk / 32
1	0	1	Sclk / 64
1	1	0	Sclk / 128
1	1	1	Sclk / 256

- **Bit 1 - MSTR: Master/Slave mode select**
  This bit selects Master SPI mode when set(one), and Slave SPI mode when cleared(zero).

- **Bit 0 - SPE: SPI enable**
  When the SPE bit is set(one), the SPI is enabled. When the SPI is disabled, all of the associated I/O pins are set to inputs.

SPI0 Transmit Data Address: 0x34, IOpage: 0/1
SPI0tx Access Mode: Write only

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	TD7	TD6	TD5	TD4	TD3	TD2	TD1	TD0

- **Bit 15..8 - Res: Reserved**
  These bits are reserved.
- **Bit 7..0 - TD: Transmit data byte**
  Data byte to transmit

SPI0 Receive Data Address: 0x34, IOpage: 0/1
SPI0rx Access Mode: Read only

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CS	0	0	0	0	0	0	0	RD7	RD6	RD5	RD4	RD3	RD2	RD1	RD0

- **Bit 15 - CS: Slave Select state**
  This is a read only bit. In Master mode this bit is ignored and reads back zero. In Slave mode this bit is the registered value present at CS0 pin. In Slave mode if the CS0 pin goes high in the middle of a transaction the internal shift registers are reset and data received is invalid.
- **Bit 14..8 - Res: Reserved**
  These bits are reserved.
- **Bit 7..0 - RD: Receive data byte**
  Data byte received

2302 SPI1

The I/O pins associated with the SCK1,MISO1, and MOSI1, SEN1 are shared with GPIO port D pins.

SPI1 Configuration
SPI1cfg

Address: 0x37, IOpage: 0/1
Access Mode: Read/Write

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CS	0	0	0	0	0	0	0	OD	CPOL	CPHA	SCK2	SCK1	SCK0	MSTR	SPE

- **Bit 15 - CS: Slave Select state**
  This is a read only bit. This bit is the registered value present at CS1 pin. In Slave mode if the CS1 pin goes high in the middle of a transaction the internal shift registers are reset and the data received is invalid.
- **Bit 14..8 - Res: Reserved**
  These bits are reserved.
- **Bit 7 - OD: Output Disable**
  If this bit is set(one), the output pin (either MOSI or MISO, depending whether the SPI is master or slave) will be under the control of the GPIO port. The SPI transmit pin is effectively disabled. If this bit is clear(zero), the pin is a SPI data output.
- **Bit 6 - CPOL: Clock Polarity**
  The clock polarity can be configured for master mode only. In slave mode the polarity is set at 1 and this bit is ignored.
  When this bit is set(one), SCK idles high. When cleared(zero), SCK idles low.
- **Bit 5 - CPHA: Clock Phase**
  The clock phase can be configured for master mode only. In slave mode the polarity is set at 1 and this bit is ignored.
  When this bit is set(one), SCK is delayed one half cycle.
- **Bit 4,3,2 - SCK2,SCK1,SCK0: SPI Clock Rate select**
  These three bits control the SPI clock(SCK) rate of the device when configured in master mode. In slave mode these bits are ignored. The relationship between SCK and the system clock(Sclk) frequency is shown below:

SCK2	SCK1	SCK0	SCK Frequency
0	0	0	Sclk / 2
0	0	1	Sclk / 4
0	1	0	Sclk / 8
0	1	1	Sclk / 16
1	0	0	Sclk / 32
1	0	1	Sclk / 64
1	1	0	Sclk / 128
1	1	1	Sclk / 256

- **Bit 1 - MSTR: Master/Slave mode select**
  This bit selects Master SPI mode when set(one), and Slave SPI mode when cleared(zero).

- **Bit 0 - SPE: SPI enable**
  When the SPE bit is set(one), the SPI is enabled. Enabling the SPI sets the function multiplexer for SCK1, MISO1 and MOSI1 pins.

SPI1 Transmit Data Address: 0x36, IOpage: 0/1
SPItx Access Mode: Write only

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	TD7	TD6	TD5	TD4	TD3	TD2	TD1	TD0

- **Bit 15..8 - Res: Reserved**
  These bits are reserved.
- **Bit 7..0 - TD: Transmit data byte**
  Data byte to transmit

SPI1 Receive Data Address: 0x36, IOpage: 0/1
SPIrx Access Mode: Read only

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CS	0	0	0	0	0	0	0	RD7	RD6	RD5	RD4	RD3	RD2	RD1	RD0

- **Bit 15 - CS: Slave Select state**
  This is a read only bit. In Master mode this bit is ignored and reads back zero. In Slave mode this bit is the registered value present at CS1 pin. In Slave mode if the CS1 pin goes high in the middle of a transaction the internal shift registers are reset and data received is invalid.
- **Bit 14..8 - Res: Reserved**
  These bits are reserved.
- **Bit 7..0 - RD: Receive data byte**
  Data byte received

2400 Baseband Units (BBU)

The Base Band Unit (BBU) is designed to interface to many RF sections that use a digital bit stream for transmit and receive data. The BBU provides the following features:

- Configurable bit and word synchronization
- Automatic baud rate synchronization to the incoming bit stream
- Very fine baud rate generation
- RZ and NRZ encoding and decoding
- Asynchronous and synchronous clocking of data
- Optional 0 DC bias coding over 16 bits
- Optional 4 bit run length coding
- Automatic error correction of 3, 2, or 1 bit errors in every 16 bit code word, depending on data rate used.
- Firmware interface has the ability to gather statistics on the number of bit errors

XInC2 contains two BaseBand Units. BBU1 is similar to BBU0 except that it does not have the built in run length / error correction logic.

2401 Basic Theory of Operation

The BaseBand Unit has four major functional blocks.

- Bit synchronization
- Word synchronization
- Baud Rate Generation
- Error correction
- Line Signal Coding
- Bit timer

Bit synchronization The Bit Synchronizer performs clock recovery and generates the bit timing. The bit synchronizer requires a sequence of alternating '1's and '0's to reset the Baseband unit and correctly extract the bit edges. Deviations of up to 150ppm in the crystal of the receiver and transmitter can be compensated for by the bit synchronizer. A minimum of 16 bits of alternating '1's and '0's is required by the bit synchronizer.

Word Synchronization The Word Synchronizer obtains synchronization at correct word boundaries after bit synchronization has been achieved and a configurable 16-bit "start" word has been received. The synchronizer is able to correct up to 3 bit errors in the start word.

BaudRate Generation The maximum bit rate over the RF link is limited to 1/8 of the system clock. The bit rate is configurable using two choices of baud rate generators. A wide range of baud rates are possible, for example at 12MHz the baud rate could be between 22.9 – 1.5 Mbps in 22.9 bps steps.

Error Correction The bit stream is created with 16 bit words, each word represents a certain amount of data. Two encoding ratios are possible 4/16, and 6/16.The ratios describe the number of data bits represented by each 16 bit encoded word. The code words have been specially chosen, and meet the above mentioned run length and dc bias characteristics. The actual data rate of the payload is the bit stream rate multiplied by the encoding ratio, for example a 1.5 Mbps bit stream would result in a data rate of 562 kbps.

Each code rate has a certain tolerance for bit errors. Data bits are successfully decoded if there are no more than the following number of bit errors within each 16 bit code word received.

Code Rate	Bit errors per 16 bit word
4/16	3
6/16	2

Baud Rate Synchronization The receive side of the BBU automatically adjusts it bit clock to the incoming data stream. This adjustment is allows the system to use very low tolerance crystals as the system clock source.

Line Signal Coding Non Return to Zero (NRZ) signal coding and Return to Zero (RZ) signal coding are supported. Asynchronous and synchronous clocking of data is supported.

Bit Timer A 16-bit counter is provided that increments at the operating baudrate if asynchronous clocking is used and at the operating bitrate if synchronous clocking is used.

2402 BBU0 (Baseband Unit 0)

The BBU requires very little configuration for normal operation. The BBUcfg0 and BBUcfg1 registers are used to configure the BBU.

Note! If he Clock pin enable bit is set then the enable on the i/o pin is enabled and the output comes from the BBU peripheral and not the associated GPIO. When the enable bit is cleared the pin function/state is determine by the GPIO configuration.
The Transmit pin enable bit operates the same as the clock pin enable.
The control of the rx I/O pin is always controlled entirely by the associated GPIO configuration. The data received by the bbu peripheral is the value at the pin. The GPIO could configure the pin to be an output and could be changing the value. In this case the bbu rx data would receive this data.

2403 BBU0 Configuration

BBU Configuration0 Address: 0x58, IOpage: 0
BBU0cfg0 Access Mode: Read/Write

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ROM	BG	TM	BI	RXm	TXm	CII	COI	CE	CDE	CM	LP	H	RAW	TXE	E

- **Bit 15 - ROM: ROM enable**
  When the ROM bit is cleared(zero), the ROM memory will not be enabled. When the ROM bit is set(one), the ROM memory will be enabled.
- **Bit 14 – BG: baudRateGeneratorSelect:**
  When the this bit is cleared(zero), the old Baudrate generator is used. When the ROM bit is set(one), the new baudrate generator is used.
  Bits (7-0) of the BBUbrg register is used for new generator.
- **Bit 13 - TM: Time clock source**
  When the TMsrc bit is cleared(zero), the BBUtimer will use the internal baud rate clock to increment. When the TMsrc bit is set(one), the BBUtimer will use the external clock source to increment.
- **Bit 12 - BI: Bidirectional data mode**
  Sets unidirectional or bidirectional data mode. When the BI bit is cleared then the BBO pin is transmit data and BBI is received data.
  When the BI bit is set the BBO pin is the bidirectional data pin and the BBI pin is an input and controls the tristate enable of the BBO I/O cell. When the BBI pin is zero the BBO pin is tristate. When the BBI pin is one the BBO pin is enabled.
  In bidirectional data mode the function of the output configuration of the BBI pin is controlled by the associated GPIO.
- **Bit 11 - RXm: RX mode**
  When the RXm bit is cleared(zero), the receiver will decode a non-return-to-zero signal. When the RXm bit is set(one), the receiver will decode a return-to-zero signal.
- **Bit 10 - TXm: TX mode**
  When the TXm bit is cleared(zero), the transmitter will generate a non-return-to-zero signal. When the TXm bit is set(one), the transmitter will generate a return-to-zero signal.
- **Bit 9 - CII: CLK input invert**
  When the CLKIi bit is cleared(zero), the receiver will sample data on the rising edge of the input CLK and the transmitter will setup the data on the falling edge of the clk. When the CLKi bit is set(one), the CLK signal will be inverted.

- **Bit 8 - COI: CLK output invert**
  When the CLKOi bit is cleared(zero), the output CLK signal is not inverted(data transitions on falling edge of clock).
  When the CLKOi bit is set(one), the output CLK signal is inverted(data transitions on rising edge of clock).
- **Bit 7 - CE: Clock pin enable**
  When the CLKE bit is set(one), the clock data pin is enabled. When the bit is cleared(zero), the pin function is configured according to the associated GPIO pin.
- **Bit 6 - CDE: Clock detect enable**
  When the CLKDE bit is set(one), the clock gone bit is set if the external clock source is not present as controlled by the clock detect window. If the glock gone bit is set the receive will go into "hunt" mode. In transmit the flow control on the BBUtx register will be disabled. This is used to avoid deadlock if the clock source is removed.
- **Bit 5 - CM: Clock mode (clock pin enable needs to be 0)**
  When the CLKm bit is cleared(zero), the transmitter and receiver will use the internally generated clock. When the CLKm bit is set(one), the transmitter and receiver will use the signal at the CLK_in pin as the bit clock.
- **Bit 4 - LP: Loop back mode**
  Sets internal loop back mode. The tx output is internally connected to the rx input.
- **Bit 3 - H: Force hunt mode**
  When the RAW bit is cleared, this bit has no meaning. When the RAW bit is set, setting the HUNT bit forces the BBU into preamble detection mode. When the HUNT bit is cleared, the preamble detection is disabled. This is useful when the user wants to send data without using the hardware error correction.
- **Bit 2 - RAW: Receive raw mode**
  Setting this bit one puts the BBU into raw data mode.
- **Bit 1 - TXE: Transmit pin enable**
  When the TXE bit is set(one), the transmit data pin is enabled. When the bit is cleared(zero) the pin function is configured according to the associated GPIO pin.
  Bit 0 - E: BBU enable
  When the E bit is set(one), the Baseband Unit is enabled.

BBU Configuration1 Address: 0x59, IOpage: 0
BBU0cfg1 Access Mode: Read/Write

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CDW7	CDW6	CDW5	CDW4	CDW3	CDW2	CDW1	CDW0	PS7	PS6	PS5	PS4	PS3	PS2	PS1	PS0

- **Bit 15..8 - CDW: Clock detect window**
  These bits are used to configure the window size for the external clock detect circuit. When a rising edge of the external clock is detected the 8 bit counter is loaded with the CDW value and the clock present signal is '1'. This counter is decremented with the system clock. If the counter = 0 then the clock present signal is '0'.
  Bit 7..0 - PS: Pulse stretcher counter
  These bits are used to configure the pulse stretching length used in RXm= '1' mode (RX decoding).

BBU StartWord Configuration Address: 0x5F, IOpage: 0
BBU0start Access Mode: Read/Write

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BR15	BR14	BR13	BR12	BR11	BR10	BR9	BR8	BR7	BR6	BR5	BR4	BR3	BR2	BR1	BR0

- **Bit 15..0 – SW: Start Word configuration**
  These bits represent the 16-bit start character that is used by the receiver to obtain word synchronization

BBU BaudRate Address: 0x5B, IOpage: 0
BBU0brg Access Mode: Read/Write

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BR15	BR14	BR13	BR12	BR11	BR10	BR9	BR8	BR7	BR6	BR5	BR4	BR3	BR2	BR1	BR0

- **Bit 15..0 – BR: Baud rate configuration**
  These bits represent a 16-bit baud rate setting for BBU.

  Bit rate = (sys_clk/8) * (1/(65535+1)) *(BBUbrg + 1)

2404 BBU Timer

The BBU contains a free running 16-bit counter that is clocked by the buadrate generator. Reading BBUtime register reads the current state of the counter.

BBU Timer
BBU0time

Address: 0x5C, IOpage: 0
Access Mode: Read only

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BT15	BT14	BT13	BT12	BT11	BT10	BT9	BT8	BT7	BT6	BT5	BT4	BT3	BT2	BT1	BT0

- **Bit 15..0 - BT: BBU time**
  These bits represent the current value of the 16-bit free running counter.

2405 Data Transmitter

BBU Transmit Coded Data
BBU0tx

Address: 0x5A, IOpage: 0
Access Mode: Write only

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
T15	T14	T13	T12	T11	T10	T9	T8	T7	T6	T5	T4	T3	T2	T1	T0

- **Bit 15..0 – T: Coded transmit data**
  These bits represent the encoded 16-bits to transmit.

2406 BBU0 Receiver

BBU Rate 4 Receive Data Address: 0x5D, IOpage: 0
BBU0rx4 Access Mode: Read only

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
HUNT	ERR	DIST1	DIST0	CG	0	0	0	0	0	0	0	R4_3	R4_2	R4_1	R4_0

- **Bit 15- HUNT: Hunt mode status**
  This bit is set when the timing synchronizer sees preamble. It is used to detect a start of packet.
- **Bit 14 - ERR: Hard error status NOTE! requires ROM to be enabled**
  This bit is set when an uncorrectable codeword is received.
- **Bit 13,12 - DIST1,DIST0: Bits corrected NOTE! requires ROM to be enabled**
  These bits indicate the number of bits corrected in the received codeword.
- **Bit 11 - CG: Clock Gone**
  This bit is set when clock mode = '1' and the external clock is not present according to the clock detect window counter.
- **Bit 10..4 - Res: Reserved**
  These bits are reserved.
- **Bit 3,2,1,0 - R4_3,R4_2,R4_1, R4_0: Rate4 receive data**
  These 4 bits are the received data nibble.

BBU Rate 6 Receive Data Address: 0x5E, IOpage: 0
BBU0rx6 Access Mode: Read only

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
HUNT	ERR	DIST1	DIST0	CG	0	0	0	0	0	R6_5	R6_4	R6_3	R6_2	R6_1	R6_0

- **Bit 15- HUNT: Hunt mode status**
  This bit is set when the timing synchronizer sees preamble.
- **Bit 14 - ERR: Hard error status NOTE! requires ROM to be enabled**
  This bit is set when an uncorrectable codeword is received.
- **Bit 13,12 - DIST1,DIST0: Bits correction NOTE! requires ROM to be enabled**
  These bits indicate the number of bits corrected in the received codeword. DIST = 3 is invalid.
- **Bit 11 - CG: Clock Gone**
  This bit is set when clock mode = '1' and the external clock is not present according to the clock detect window counter.
- **Bit 10..4 - Res: Reserved**
  These bits are reserved.
- **Bit 5,4,3,2,1,0 - R6_5,R6_4,R6_3,R6_2,R6_1, R6_0: Rate6 receive data**
  These 6 bits are the received data.

BBU Receive Raw Data Address: 0x5A, IOpage: 0
BBU0rx Access Mode: Read only

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R15	R14	R13	R12	R11	R10	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0

- **Bit 15..0 – BR: Baud rate configuration**
  These bits represent the raw 16-bit data received.

2407 BBU1 (Baseband Unit 1)

The BBU1 has all of the features of BBU0 except for the built in rate 4:16 and 6:16 error correction.

The BBU requires very little configuration for normal operation. The BBUcfg0 and BBUcfg1 registers are used to configure the BBU.

Note! If he Clock pin enable bit is set then the enable on the i/o pin is enabled and the output comes from the BBU peripheral and not the associated GPIO. When the enable bit is cleared the pin function/state is determine by the GPIO configuration.
The Transmit pin enable bit operates the same as the clock pin enable.
The control of the rx I/O pin is always controlled entirely by the associated GPIO configuration. The data received by the bbu peripheral is the value at the pin. The GPIO could configure the pin to be an output and could be changing the value. In this case the bbu rx data would receive this data.

2408 BBU1 Configuration

BBU Configuration0 Address: 0x60, IOpage: 0
BBU1cfg0 Access Mode: Read/Write

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	BG	TM	BI	RXm	TXm	CII	COI	CE	CDE	CM	LP	H	RAW	TXE	E

- **Bit 15 - RS: Reserved**
- **Bit 14 – BG: baudRateGeneratorSelect:**
  When the this bit is cleared(zero), the old Baudrate generator is used. When the ROM bit is set(one), the new baudrate generator is used.
  Bits (7-0) of the BBUbrg register is used for new generator.
- **Bit 13 - TM: Time clock source**
  When the TMsrc bit is cleared(zero), the BBUtimer will use the internal baud rate clock to increment. When the TMsrc bit is set(one), the BBUtimer will use the external clock source to increment.
- **Bit 12 - BI: Bidirectional data mode**
  Sets unidirectional or bidirectional data mode. When the BI bit is cleared then the BBO pin is transmit data and BBI is received data.
  When the BI bit is set the BBO pin is the bidirectional data pin and the BBI pin is an input and controls the tristate enable of the BBO I/O cell. When the BBI pin is zero the BBO pin is tristate. When the BBI pin is one the BBO pin is enabled.
  In bidirectional data mode the function of the output configuration of the BBI pin is controlled by the associated GPIO.
- **Bit 11 - RXm: RX mode**
  When the RXm bit is cleared(zero), the receiver will decode a non-return-to-zero signal. When the RXm bit is set(one), the receiver will decode a return-to-zero signal.
- **Bit 10 - TXm: TX mode**
  When the TXm bit is cleared(zero), the transmitter will generate a non-return-to-zero signal. When the TXm bit is set(one), the transmitter will generate a return-to-zero signal.
- **Bit 9 - CII: CLK input invert**
  When the CLKIi bit is cleared(zero), the receiver will sample data on the rising edge of the input CLK and the transmitter will setup the data on the falling edge of the clk. When the CLKi bit is set(one), the CLK signal will be inverted.

- **Bit 8 - COI: CLK output invert**
  When the CLKOi bit is cleared(zero), the output CLK signal is not inverted(data transitions on falling edge of clock).
  When the CLKOi bit is set(one), the output CLK signal is inverted(data transitions on rising edge of clock).
- **Bit 7 - CE: Clock pin enable**
  When the CLKE bit is set(one), the clock data pin is enabled. When the bit is cleared(zero), the pin function is configured according to the associated GPIO pin.
- **Bit 6 - CDE: Clock detect enable**
  When the CLKDE bit is set(one), the clock gone bit is set if the external clock source is not present as controlled by the clock detect window. If the glock gone bit is set the receive will go into "hunt" mode. In transmit the flow control on the BBUtx register will be disabled. This is used to avoid deadlock if the clock source is removed.
- **Bit 5 - CM: Clock mode (clock pin enable needs to be 0)**
  When the CLKm bit is cleared(zero), the transmitter and receiver will use the internally generated clock. When the CLKm bit is set(one), the transmitter and receiver will use the signal at the CLK_in pin as the bit clock.
- **Bit 4 - LP: Loop back mode**
  Sets internal loop back mode. The tx output is internally connected to the rx input.
- **Bit 3 - H: Force hunt mode**
  When the RAW bit is cleared, this bit has no meaning. When the RAW bit is set, setting the HUNT bit forces the BBU into preamble detection mode. When the HUNT bit is cleared, the preamble detection is disabled. This is useful when the user wants to send data without using the hardware error correction.
- **Bit 2 - RAW: Receive raw mode**
  Setting this bit one puts the BBU into raw data mode.
- **Bit 1 - TXE: Transmit pin enable**
  When the TXE bit is set(one), the transmit data pin is enabled. When the bit is cleared(zero) the pin function is configured according to the associated GPIO pin.
  Bit 0 - E: BBU enable
  When the E bit is set(one), the Baseband Unit is enabled.

BBU Configuration1 Address: 0x61, IOpage: 0
BBU1cfg1 Access Mode: Read/Write

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CDW7	CDW6	CDW5	CDW4	CDW3	CDW2	CDW1	CDW0	PS7	PS6	PS5	PS4	PS3	PS2	PS1	PS0

- **Bit 15..8 - CDW: Clock detect window**
  These bits are used to configure the window size for the external clock detect circuit. When a rising edge of the external clock is detected the 8 bit counter is loaded with the CDW value and the clock present signal is '1'. This counter is decremented with the system clock. If the counter = 0 then the clock present signal is '0'.
  Bit 7..0 - PS: Pulse stretcher counter
  These bits are used to configure the pulse stretching length used in RXm= '1' mode (RX decoding).

BBU StartWord Configuration Address: 0x67, IOpage: 0
BBU1start Access Mode: Read/Write

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BR15	BR14	BR13	BR12	BR11	BR10	BR9	BR8	BR7	BR6	BR5	BR4	BR3	BR2	BR1	BR0

- **Bit 15..0 – SW: Start Word configuration**
  These bits represent the 16-bit start character that is used by the receiver to obtain word synchronization.

BBU BaudRate Address: 0x63, IOpage: 0
BBU1brg Access Mode: Read/Write

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BR15	BR14	BR13	BR12	BR11	BR10	BR9	BR8	BR7	BR6	BR5	BR4	BR3	BR2	BR1	BR0

- **Bit 15..0 – BR: Baud rate configuration**
  These bits represent a 16-bit baud rate setting for BBU.

  Bit rate = (sys_clk/8) * (1/(65535+1)) *(BBUbrg + 1)

2409 BBU Timer

The BBU contains a free running 16-bit counter that is clocked by the buadrate generator. Reading BBUtime register reads the current state of the counter.

BBU Timer Address: 0x64, IOpage: 0
BBU1time Access Mode: Read only

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BT15	BT14	BT13	BT12	BT11	BT10	BT9	BT8	BT7	BT6	BT5	BT4	BT3	BT2	BT1	BT0

- **Bit 15..0 - BT: BBU time**
  These bits represent the current value of the 16-bit free running counter.

2410 Data Transmitter

BBU Transmit Coded Data Address: 0x62, IOpage: 0
BBU1tx Access Mode: Write only

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
T15	T14	T13	T12	T11	T10	T9	T8	T7	T6	T5	T4	T3	T2	T1	T0

- **Bit 15..0 – T: Coded transmit data**
  These bits represent the encoded 16-bits to transmit.

2411 Receiver

BBU Rate Flow Control Address: 0x65, IOpage: 0
BBUflow Access Mode: Read only

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
HUNT	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

- **Bit 15- HUNT: Hunt mode status**
  This bit is set when the timing synchronizer sees preamble. It is used to detect a start of packet.
- **Bit 11 - CG: Clock Gone**
  This bit is set when clock mode = '1' and the external clock is not present according to the clock detect window counter.
- **Bit 10..4 - Res: Reserved**
  These bits are reserved.

BBU Receive Raw Data Address: 0x62, IOpage: 0
BBU1rx Access Mode: Read only

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R15	R14	R13	R12	R11	R10	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0

- **Bit 15..0 – BR: Baud rate configuration**
  These bits represent the raw 16-bit data received.

2500 Digital Audio Serial Interface

The Digital Audio Serial Interface (DASI) allows XInC2 to connect to digital audio devices conforming to the Inter-IC Sound (I2S) specification (and variations).

The DASI is equipped with three digital audio channels, each allowing the transfer of a stereo audio bit stream, for a total of 6 channels of audio (two per transceiver). Master and slave modes are available. This peripheral allows XInC2 to generate or receive arbitrary bit depth audio. The only constraint for bit depth lies in slave mode, where incomplete bytes are lost on receive. (if 20-bit data is received, the frames must be at least 24 bits long or the last nibble of the data will be lost.)

As master, the DASI outputs the digital audio master clock (MCLK), the bit clock (SCK) and the Left-Right channel clock (LRCK) or frame clock. As slave the SCK and LRCK pins become inputs, but the MCLK pin may still be configured to provide a clock to an external device. MCLK is not required for slave operation.

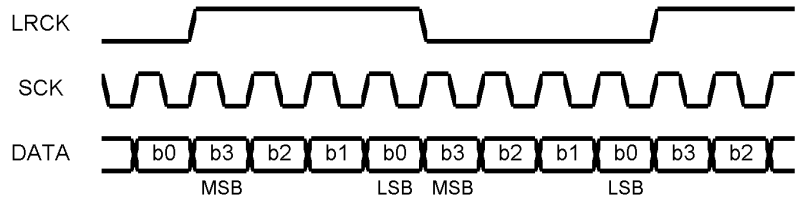
All transceivers share the bit clock (SCK) configuration, sample depth configuration, master/slave selection, and block/poll selection. Each transceiver is individually enabled and configured for transmit or receive and Left Justified or I2S format data.

A synchronization circuit will detect the first LRCK transition and “lock” the transceiver to it. In master mode this synchronization occurs only once (this gives the master more flexibility in the relationships between the data and the LRCK) but in slave mode, the start of each word is synchronized to the LRCK signal since the frame depth may not be constant.

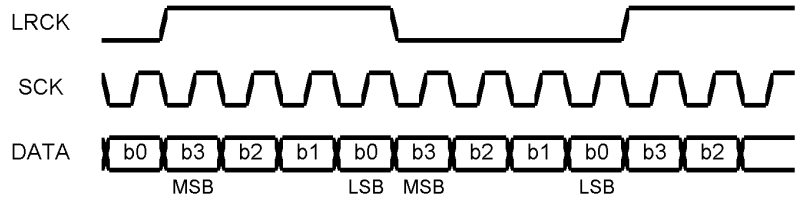
The SCK POL bit (DASIconfig0) selects the bit clock edge that the data and LRCK transition on. Transmitted data will change on the edge configured by the SCK POL bit. With receive data, however, the receive shift register latches incoming data one half-period of the SCK later (while the data is stable).

Typical waveforms are shown below, using only four bit frames for illustration purposes:

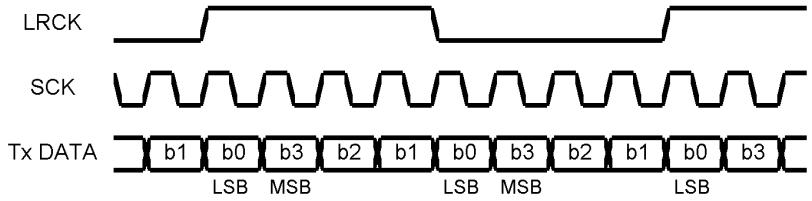
RE SCK, Left Justified format



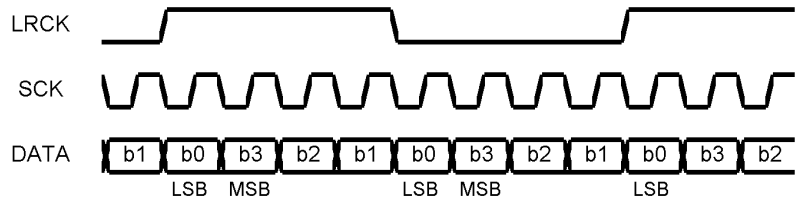
FE SCK, Left Justified format



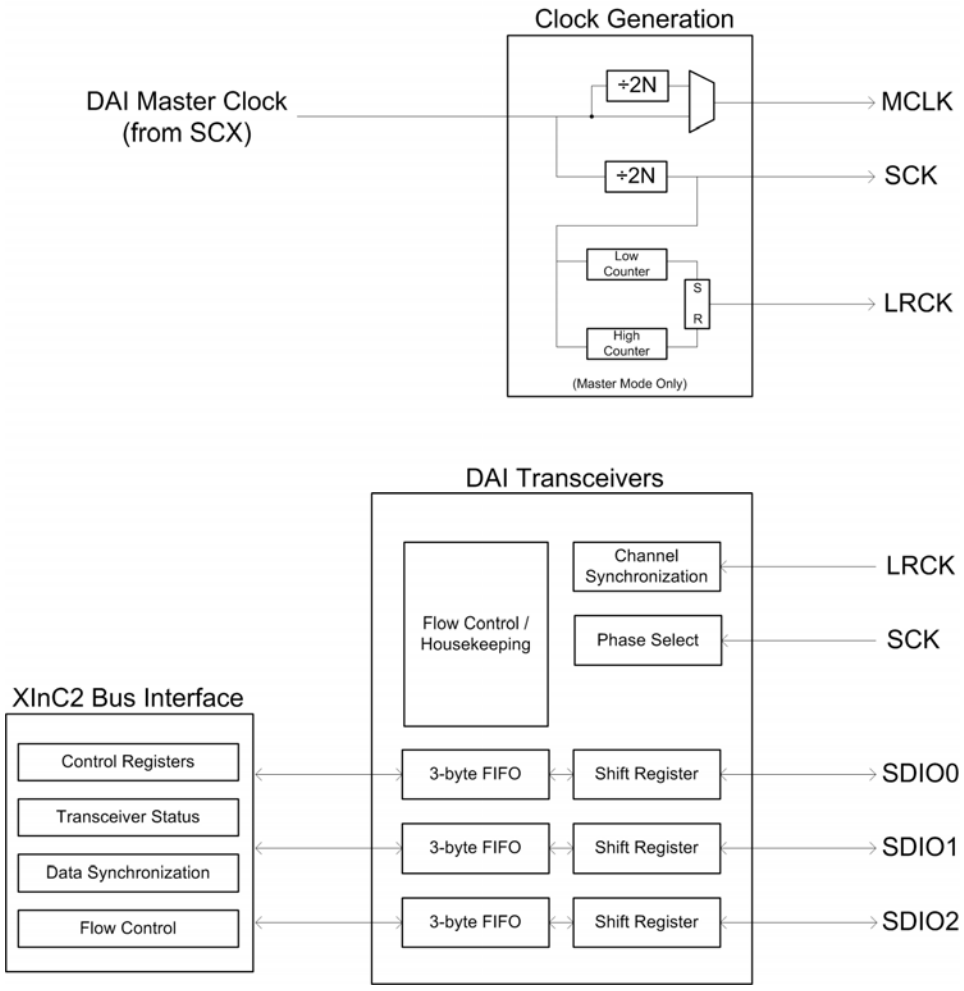
RE SCK, I2S format



FE SCK, I2S format



A high-level block diagram of the Digital Audio Interface is shown below:



2501 DASI Configuration

DASI Mode Configuration Address: 0x68
DASIconfig0 Access Mode: Read/Write

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	LRCK SH2	TX/ RX2	EN2	-	LRCK SH1	TX/ RX1	EN1	-	LRCK SH0	TX/ RX0	EN0	MSTR /SLV	SCK POL	WAIT EN	DASI EN

- **Bits 15, 11, 7 - Reserved**
  These bits are reserved.
- **Bits 14, 10, 6 – LRCK SHn: LRCK Shift**
  When this bit is set(one), the channel(n) will read or write data shifted right by one bit with respect to the LRCK transition chosen in 'LRCK SYNn'. When this bit is cleared(zero), the channel(n) will align the data to the selected LRCK transition in 'LRCK SYNn'.
- **Bits 13, 9, 5 – TX/RXn: Transmit/Receive selection**
  When this bit is set(one), the channel will be configured to transmit data. When this bit is cleared(zero), the channel will be configured to receive data.
- **Bits 12, 8, 4 – ENn: Enable transceiver**
  When this bit is set(one), the transceiver is enabled. When this bit is cleared, the pin will be tristated and the transceiver will be inactive.
- **Bit 3 – MSTR/SLV: Mode select**
  This bit selects whether the DASI peripheral is to be configured for master mode(supplies MCLK, SCK, LRCK) or slave mode(runs from externally supplied SCK, LRCK). If set(one), master mode is chosen. If cleared(zero), slave mode is chosen.
- **Bit 2 – SCK POL: SCK polarity**
  This bit selects the edge of the bit clock(SCK) that data transitions occur on. If set (one), the transmit data transitions on the rising edge of SCK, and data is latched into the receiver on the falling edge of SCK.
  If cleared(zero), transmit data transitions on the falling edge of SCK and receive data is latched on the rising edge.
- **Bit 1 – WAIT EN: Wait enable**
  This peripheral has configurable blocking. When this bit is set(one) the DASI peripheral will block read operations while the receive buffer is empty and will block write operations while the receive buffer is full. When this bit is cleared(zero), the peripheral does not block and the status of the buffers must be determined by polling.
- **Bit 0 – DASI EN: Enable the DASI peripheral**
  When this bit is cleared(zero), the DASI peripheral will be held in a reset state. All registers will be cleared and the logic will be inactive. When this bit is set(one), the peripheral will resume normal operation.

DASI General Configuration Address: 0x69
DASIconfig1 Access Mode: Read/Write

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MCLK EN	LRCK SYNC	LOOP 1	LOOP 0	BPF3	BPF2	BPF1	BPF0	MCLK DIV3	MCLK DIV2	MCLK DIV1	MCLK DIV0	SCK DIV3	SCK DIV2	SCK DIV1	SCK DIV0

- **Bit 15 – MCLK EN: MCLK enable**
  When set(one), this bit enables the MCLK and SCK generator. When cleared(zero), this bit disables the MCLK and SCK generator.
- **Bit 14 – LRCK SYNC: Left/Right Clock Sync**
  (Master mode only)
  When set(one), the first byte of data sent or received will be synchronized to the next rising edge of LRCK. When cleared(zero), the first byte of data will be synchronized to a rising edge of LRCK.
- **Bits 13, 12 – Loop 1,0: Loopback select**
  These bits select the function of the internal loopback multiplexers as shown in the table below:

LOOP1,0	Loopback
00	No loopback
01	SDIO0 <-> SDIO1
10	SDIO0 <-> SDIO2
11	SDIO1 <-> SDIO2

- **Bits 11..8 – BPF: Bytes Per Frame Select**
  (Slave mode only)
  These bits determine the number of bytes sent or received in each frame, according to the table below:

BPF	Bytes Per Frame
0000	1
0001	2
…	…
1110	15
1111	continuous mode

  If BPF = '1111', the slave will transmit or receive data in a continuous bit stream. No synchronization to LRCK will be performed.

- **Bits 7..4 – MCLK SEL: MCLK divider select**
  The value of these bits determines the divider used between the DASI master clock and the MCLK output pin.

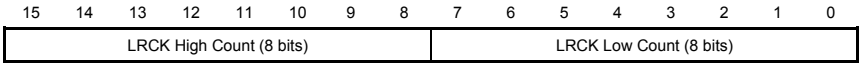
MCLK SEL3..0	Output Divider
0000	÷1
0001	÷2
0010	÷4
0011	÷8
0100	÷16
0101	÷32
0110	÷64
0111	÷128
1000	÷256
1001	÷512

- **Bits 3..0 – BUF SEL: Clock source select**

(Master mode only)
The value of these bits determines the divider used between the DASI master clock and the SCK output pin.

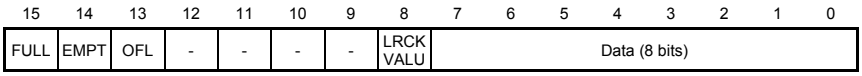
SCK SEL3..0	Output Divider
0001	÷2
0010	÷4
0011	÷8
0100	÷16
0101	÷32
0110	÷64
0111	÷128
1000	÷256
1001	÷512

DASI LRCK Configuration Address: 0x6A
DASIconfig2 Access Mode: Read/Write



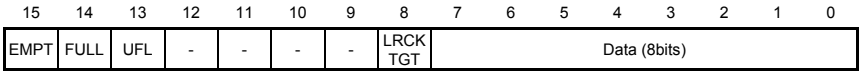
- **Bits 15..8 – LRCK High Count**
  This byte determines the number of SCK ticks that LRCK is high before a high-to-low transition occurs. (LRCK is high for n+1 SCK ticks)
- **Bits 7..0 – LRCK Low Count**
  This byte determines the number of SCK ticks that LRCK is low before a low-to-high transition occurs. (LRCK is low for n+1 SCK ticks)

DASI Channel 0 Receive Register Address: 0x6B
DASIrx0 Access Mode: Read/Write



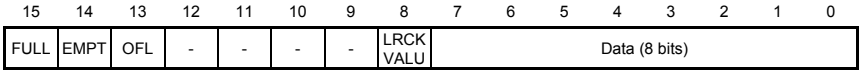
- **Bit 15 – FULL: FIFO full flag**
  When set(one), this bit indicates that the transmit or receive buffer is full. When cleared(zero), this bit indicates that the buffer is not full.
- **Bit 14 – EMPT: FIFO empty flag**
  When set(one), this bit indicates that the transmit or receive buffer is empty. When cleared(zero), this bit indicates that the buffer is not empty.
- **Bit 13 – OFL: Overflow detect flag**
  When set, an overflow condition has occurred (ie. Write when full.) This may occur in either receive mode (firmware not keeping up with incoming data) or transmit mode (developer error: replace and continue). This bit is sticky; the only way to clear this condition is to reset the DASI peripheral by using the "DASI EN" bit in DASIconfig0.
- **Bit 8 – LRCK VAL: LRCK Value (Read Only)**
  (Slave Only) This bit indicates the LRCK value at the second-last bit of the data byte.
- **Bits 7..0 – Audio Data**
  This is the last data byte received.

DASI Channel 0 Transmit Register Address: 0x6B
DASItx0 Access Mode: Read/Write



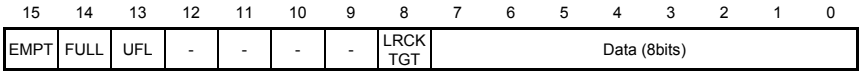
- **Bit 15 – EMPT: FIFO empty flag**
  When set(one), this bit indicates that the transmit or receive buffer is empty. When cleared(zero), this bit indicates that the buffer is not full.
- **Bit 14 – FULL: FIFO full flag**
  When set(one), this bit indicates that the transmit of receive buffer is full. When cleared(zero), this bit indicates that the buffer is not full.
- **Bit 13 – UFL: Underflow detect flag**
  When set, this bit indicates that an underflow condition has occurred. (ie. Read when empty.) The only way to clear this condition is to reset the DASI peripheral by using the "DASI EN" bit in DASIconfig0.
- **Bit 13 – LRCK TGT: LRCK Target (Write Only)**
  (Slave Only) This bit is written when data is output to the peripheral. The value of the bit indicates which polarity of LRCK the data will be sent on.
- **Bits 7..0 – Audio Data**
  This is the data to be sent.

DASI Channel 1 Receive Register Address: 0x6C
DASIrx1 Access Mode: Read/Write



- **Bit 15 – FULL: FIFO full flag**
  When set(one), this bit indicates that the transmit of receive buffer is full. When cleared(zero), this bit indicates that the buffer is not full.
- **Bit 14 – EMPT: FIFO empty flag**
  When set(one), this bit indicates that the transmit or receive buffer is empty. When cleared(zero), this bit indicates that the buffer is not full.
- **Bit 13 – OFL: Overflow detect flag**
  When set, an overflow condition has occurred (ie. Write when full.) This may occur in either receive mode (firmware not keeping up with incoming data) or transmit mode (developer error: replace and continue). The only way to clear this condition is to reset the DASI peripheral by using the "DASI EN" bit in DASIconfig0.
- **Bit 8 – LRCK VAL: LRCK Value (Read Only)**
  (Slave Only) This bit indicates the LRCK value at the second-last bit of the data byte.
- **Bits 7..0 – Audio Data**
  This is the last data byte received.

DASI Channel 1 Transmit Register Address: 0x6C
DASItx1 Access Mode: Read/Write



- **Bit 15 – EMPT: FIFO empty flag**
  When set(one), this bit indicates that the transmit or receive buffer is empty. When cleared(zero), this bit indicates that the buffer is not full.
- **Bit 14 – FULL: FIFO full flag**
  When set(one), this bit indicates that the transmit of receive buffer is full. When cleared(zero), this bit indicates that the buffer is not full.
- **Bit 13 – UFL: Underflow detect flag**
  When set, this bit indicates that an underflow condition has occurred. (ie. Read when empty.) The only way to clear this condition is to reset the DASI peripheral by using the "DASI EN" bit in DASIconfig0.
- **Bit 13 – LRCK TGT: LRCK Target (Write Only)**
  (Slave Only) This bit is written when data is output to the peripheral. The value of the bit indicates which polarity of LRCK the data will be sent on.
- **Bits 7..0 – Audio Data**
  This is the data to be sent.

DASI Channel 2 Receive Register Address: 0x6D
DASIrx2 Access Mode: Read/Write

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
FULL	EMPT	OFL	-	-	-	-	LRCK VALU	Data (8 bits)							

- **Bit 15 – FULL: FIFO full flag**
  When set(one), this bit indicates that the transmit of receive buffer is full. When cleared(zero), this bit indicates that the buffer is not full.
- **Bit 14 – EMPT: FIFO empty flag**
  When set(one), this bit indicates that the transmit or receive buffer is empty. When cleared(zero), this bit indicates that the buffer is not full.
- **Bit 13 – OFL: Overflow detect flag**
  When set, an overflow condition has occurred (ie. Write when full.) This may occur in either receive mode (firmware not keeping up with incoming data) or transmit mode (developer error: replace and continue). The only way to clear this condition is to reset the DASI peripheral by using the "DASI EN" bit in DASIconfig0.
- **Bit 8 – LRCK VAL: LRCK Value (Read Only)**
  (Slave Only) This bit indicates the LRCK value at the second-last bit of the data byte.
- **Bits 7..0 – Audio Data**
  This is the last data byte received.

DASI Channel 2 Receive Register Address: 0x6D
DASItx2 Access Mode: Read/Write

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EMPT	FULL	UFL	-	-	-	-	LRCK TGT	Data (8bits)							

- **Bit 15 – EMPT: FIFO empty flag**
  When set(one), this bit indicates that the transmit or receive buffer is empty. When cleared(zero), this bit indicates that the buffer is not full.
- **Bit 14 – FULL: FIFO full flag**
  When set(one), this bit indicates that the transmit of receive buffer is full. When cleared(zero), this bit indicates that the buffer is not full.
- **Bit 13 – UFL: Underflow detect flag**
  When set, this bit indicates that an underflow condition has occurred. (ie. Read when empty.) The only way to clear this condition is to reset the DASI peripheral by using the "DASI EN" bit in DASIconfig0.
- **Bit 13 – LRCK TGT: LRCK Target (Write Only)**
  (Slave Only) This bit is written when data is output to the peripheral. The value of the bit indicates which polarity of LRCK the data will be sent on.
- **Bits 7..0 – Audio Data**
  This is the data to be sent.

DASI Sample Counter DASIcount0 Address: 0x6E
Access Mode: Read/Write

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
LC15	LC14	LC13	LC12	LC11	LC10	LC9	LC8	LC7	LC6	LC5	LC4	LC3	LC2	LC1	LC0

- **Bit 15..0 – LCx: Left Channel Sample Count**
  When read, this register will return the count of all samples sent or received on the left LRCK channel (channel 0). A write to this register will simply reset the counter to zero.

DASI Sample Counter DASIcount1 Address: 0x6F
Access Mode: Read/Write

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RC15	RC14	RC13	RC12	RC11	RC10	RC9	RC8	RC7	RC6	RC5	RC4	RC3	RC2	RC1	RC0

- **Bit 15..0 – RCx: Right Channel Sample Count**
  When read, this register will return the count of all bytes sent or received on the right LRCK channel (channel 1). A write to this register will simply reset the counter to zero.

2502 DASI Sample Code (Master Mode)

The following code segment initializes the DASI as a master and transmits a previously encoded sine wave on the left channel, while sending zeroed samples on the right. If the DASI master clock is 49.152MHz, the output data will be 16-bit, 48kHz.

```
          mov       r0, 0xFE                        // Stop all threads but Thread 0
          outp      r0, SCUstop

          mov       r0, 0x0000                      // Make sure the DASI master clock is enabled
          outp      r0, SCXaltCfg

////////////////////////////////
// DASI Initialization        //
////////////////////////////////

          mov       r0, 0x060B                      // SDIO1 TX, I2S data format, enable DASI
          outp      r0, DASIconfig0
          mov       r0, 0x8025                      // Config1 : (MCLK enable, MCLK/SCK dividers)
          outp      r0, DASIconfig1
          mov       r0, 0x0F0F                      // Config2 : LRCK (24 bit frames)
          outp      r0, DASIconfig2

          inp       r0, DASIconfig0                 // Enable SDIO1
          bis       r0, r0, 8
          outp      r0, DASIconfig0

////////////////////////////////
// DASI Application           //
////////////////////////////////
                                                    // Sends 16-bit samples in a 24-bit frame. (LSbyte is zero)
          mov       r0, 0                           // Index into sample buffer
          st        r0, SamplePtr
   GetNewSample:
          ld        r1, r0, TableA

          rol       r0, r1, 8                       // MSB (bits 23..16)
          and       r0, r0, 0xFF
          outp      r0, DASItx1
          and       r0, r1, 0xFF                    // LSB (bits 15..8)
          outp      r0, DASItx1

          mov       r0, 0
          outp      r0, DASItx1                     // zero word for the second channel
          outp      r0, DASItx1

          ld        r0, SamplePtr
          add       r0, r0, 1
          and       r0, r0, 0x0F                    // Wrap index -> 64 word table
          st        r0, SamplePtr
          bra       GetNewSample


@ = (@ + 0x800-1) & -0x800 // Place data area in the next available 2K RAM block after the end of code
////////////////////////////////
// 1.5kHz tone at 48kHz rate  //
////////////////////////////////
TableA:
0
3205
6284
9124
11613
13654
15172
16107
16422
16107
15172
13655
11612
9123
6285
3203
0
-3204
-6285
-9124
-11612
-13655
-15172
-16107
-16422
```

```
-16107
-15172
-13655
-11613
-9123
-6284
-3204
-1
3204
6284
9123
11613
13655
15173
16106
16423
16107
15172
13655
11612
9124
6284
3204
0
-3203
-6285
-9123
-11612
-13655
-15172
-16107
-16422
-16106
-15173
-13655
-11613
-9124
-6284
-3204
0

//-------------------------------------------------------------------------------
// SHORT ADDRESS DATA SPACE
// - placed in last 128 words
//-------------------------------------------------------------------------------
@ = 65535 - 127

SamplePtr:          @ = @ + 1
```

2503 DASI Sample Code (Slave Mode)

This code segment initializes the DASI as a slave. Data received on the SDIO0 pin is stored to a buffer and retransmitted on SDIO1. A semaphore is used to control access to the buffer's read and write pointers. (The DASI is running from the system clock at 49.152MHz using a 1.536Mbps audio bitstream).

```
#define   BufferSemaphore     0         // Controls access to the circular buffer

          mov       r0, 0x0000                          // Make sure the DASI clock is enabled
          outp      r0, SCXaltCfg

          mov       r0, 0                               // Initialize the circular buffer
          st        r0, RD_ptr
          mov       r0, 1
          st        r0, WR_ptr
          mov       r0, BufferSemaphore
          outp      r0, SCUup                           // Initialize the semaphore

////////////////////////////////
//  DASI Initialization        //
////////////////////////////////
          mov       r0, 0x0207                          // Configure transceiver(s) without enabling
          outp      r0, DASIconfig0                     // SDIO1 is tx, SDIO0 is rx, slave mode, RE SCK, LJ data
          mov       r0, 0x8145                          // Enable MCLK output, MCLK = div by 16,
          outp      r0, DASIconfig1                     // SCK = div by 32, 2 bytes per frame(16 bit data)

////////////////////////////////
//  Initialize Application     //
////////////////////////////////
          mov       r0, 0x0010                          // Set up Thread2 to run tx code
          outp      r0, SCUpntr
          mov       r0, Thread2_Start
          outp      r0, SCUpc

          mov       r0, 0xFA                            // Run Threads 0 and 2
          outp      r0, SCUstop

          inp       r0, DASIconfig0
          bis       r0, r0, 4                           // Enable the receiver
          outp      r0, DASIconfig0

          mov       r0, 0
          jsr       r7, WriteBuffer

////////////////////////////////
// Receive Thread              //
////////////////////////////////
Thread0_Start:
          inp       r0, DASIrx0                         // Receive a byte from SDIO0
          jsr       r7, WriteBuffer               // Write to circular buffer
bra Thread0_Start

////////////////////////////////
// Transmit Thread            //
////////////////////////////////
Thread2_Start:
          ld        r5, WR_ptr                          // Wait for the buffer to have a few bytes of data
          ld        r4, RD_ptr                    // before enabling the transceiver.
          sub       r5, r5, r4
          and       r5, r5, 0x000F
          sub       r5, r5, 3
          bc        LT, Thread2_Start

          inp       r1, DASIconfig0
          bis       r1, r1, 8                           // Enable transceiver 1
          outp      r1, DASIconfig0

 T2_SDIO_Enabled:
          jsr       r7, ReadBuffer                      // Get a byte from the buffer
          outp      r0, DASItx1                         // Transmit the data to SDIO1
bra T2_SDIO_Enabled


////////////////////////////////
// Circular Buffer Routines    //
////////////////////////////////
WriteBuffer:
          // Data to be written in r0
          // Return address in r7
          // No overflow checking
          mov       r3, BufferSemaphore           // Get access to the buffer
          outp      r3, SCUdown
          ld        r4, RD_ptr
          ld        r5, WR_ptr
```

```
            add       r5, r5, 1                           // increment the pointer
            and       r5, r5, 0x000F                      // wrap the pointer
            st        r5, WR_ptr                          // store the new pointer value
            st        r0, r5, CircularBuffer        // store the data

            mov       r3, BufferSemaphore           // Release the semaphore
            outp      r3, SCUup
jsr         r7, r7

ReadBuffer:
            // Data from buffer returned in r0
            // doesn't touch r1-r2
            // Return address in r7
            // No underflow checking
            mov       r3, BufferSemaphore           // Get access to the buffer
            outp      r3, SCUdown
            ld        r4, RD_ptr
            ld        r5, WR_ptr

            add       r4, r4, 1                           // Read the next byte
            and       r4, r4, 0x000F                      // Wrap the pointer
            st        r4, RD_ptr                          // Update the read pointer
            ld        r0, r4, CircularBuffer        // Load the word and return in r0

            mov       r3, BufferSemaphore           // Release the semaphore
            outp      r3, SCUup
jsr         r7, r7
```

2600 GPIO Ports

Introduction The GPIO ports of XInC2 each contain 8 general purpose digital I/O lines and the capability to control input/output direction, output level, and to read the level applied at the pin associated with each I/O line. XInC2 has 10 GPIO ports(GPA - GPJ). Some of the pins on GPIO ports are multiplexed with other peripheral modules to reduce the overall pin count. See specific peripheral modules for control of the function multiplexer.

The I/O cells of all GPIO ports can be configured for many options such as slew rate, pull-ups, Schmitt trigger and drive strength. See the programmable I/O cell configuration section for more details. For many applications the default setup is adequate but the configuration options allow for fine system tuning.

Programming GPIO ports Two I/O address locations are allocated for each port, one for the data, and one for direction and output configuration. All individual I/O bits are independently programmable so any combination of input or output conditions is possible.

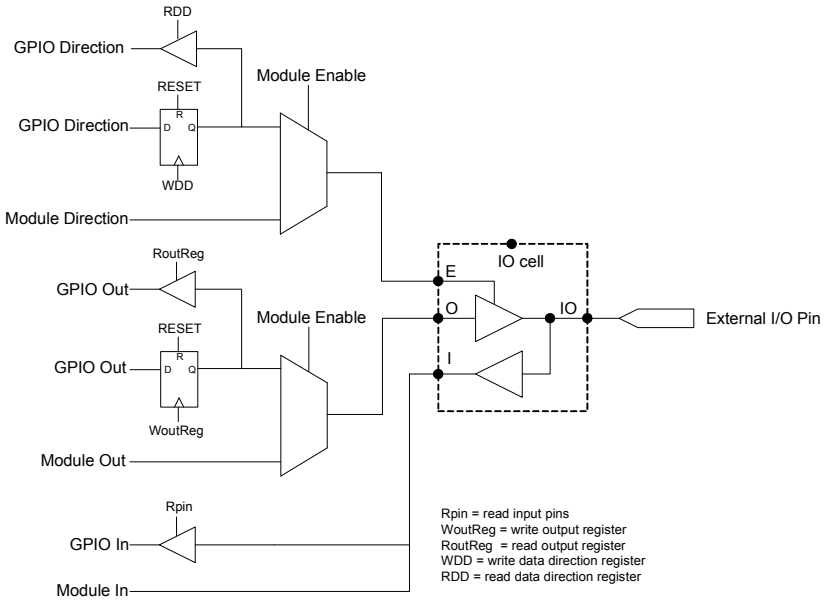
GPxcfg register is a read/write register that is used to set the direction of each pin and to set the value in the output pin register. The DDn bit in the GPxcfg register selects the direction of pin GPxn. If DDn bit is set(one), GPxn is configured as an output pin. If DDn bit is cleared(zero), GPxn is configured as an input pin.

GPxin register is a read only register that reflects the logical value present on each pin.

GPxout register is a write only register that is used to set the logic value on the output pin register. It is the latch for the data to be output.

Note! The GPIO ports are common resources that can be shared among threads. Reading the GPIO port pins can be safely done without the use of a semaphore mechanism. Care must be taken when multiple threads are modifying the GPIO registers. Conflict can be easily avoided by protecting port modifications with the SCU semaphore mechanism or by assigning pin functions to GPIO ports according to which thread accesses it.

Functional Diagram The functional diagram for the general case where a GPIO port pin is overloaded with an alternate peripheral function is shown below:



2601 GPIO Port A

The I/O cells associated with GPIO Port A are all configured with one register. See the programmable I/O cell configuration section for more details. Because all 8 pins are configured the same, this port is often used for functions that require similar configuration setup should the system require it.

Port A Input Pins Data Register GPAin

Address: 0x20, IOpage: 0/1
Access Mode: Read only

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	PIN7	PIN6	PIN5	PIN4	PIN3	PIN2	PIN1	PIN0

- **Bit 15..8 - Res: Reserved**
  These bits are reserved.
- **Bit 7..0 - PIN7, PIN6, PIN5, PIN4, PIN3, PIN2, PIN1, PIN0: Input pins value**
  These bits represent the logical level present at the external I/O pin.

Port A Output Data Register GPAout

Address: 0x20, IOpage: 0/1
Access Mode: Write only

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	OUT7	OUT6	OUT5	OUT4	OUT3	OUT2	OUT1	OUT0

- **Bit 15..8 - Res: Reserved**
  These bits are reserved.
- **Bit 7..0 - OUT7,OUT6,OUT5,OUT4,OUT3,OUT2,OUT1,OUT0: Output buffer register**
  The value of these bits are latched in the output buffer register.

Port A Direction/Data Register GPAcfg

Address: 0x21, IOpage: 0/1
Access Mode: Read/Write

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DD7	DD6	DD5	DD4	DD3	DD2	DD1	DD0	OUT7	OUT6	OUT5	OUT4	OUT3	OUT2	OUT1	OUT0

- **Bit 15..8 - DD7, DD6, DD5, DD4, DD3, DD2, DD1, DD0: Output pins**
  These bits define the direction of each I/O pin. When bit DDn is set(one), the port pin is an output. When bit DDn is cleared(zero), the port pin is an input.
- **Bit 7..0 - OUT7,OUT6,OUT5,OUT4,OUT3,OUT2,OUT1,OUT0: Output buffer register**
  These bits represent the contents of the output buffer register..

2602 GPIO Port B

The I/O cells associated with GPIO Port B are all individually configurable. See the programmable I/O cell configuration section for more details. Because all 3 pins are individually configurable for options such as pull-ups/pull-downs, and Schmitt triggered inputs, this port is often used for functions that require different configuration setup.

Note! Pins GPB2, GPB1, and GPB0 are normally reserved for chips selects of devices attached to the XPD port. See the XPD port description for more information.

Port B Input Pins Data Register GPBin

Address: 0x22, IOpage: 0/1
Access Mode: Read only

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	PIN2	PIN1	PIN0

- **Bit 15..3 - Res: Reserved**
  These bits are reserved.
- **Bit 2..0 - PIN2, PIN1, PIN0: Input pins**
  These bits represent the logical level present at the external pin.

Port B Output Data Register GPBout

Address: 0x22, IOpage: 0/1
Access Mode: Write only

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	-	-	-	-	-	OUT2	OUT1	OUT0

- **Bit 15..3 - Res: Reserved**
  These bits are reserved.
- **Bit 2..0 - OUT2,OUT1,OUT0: Output buffer register**
  The value of these bits are latched in the output buffer register.

Port B Direction/Data Register GPBcfg

Address: 0x23, IOpage: 0/1
Access Mode: Read/Write

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	DD2	DD1	DD0	-	-	-	-	-	OUT2	OUT1	OUT0

- **Bit 10..8 - DD2, DD1, DD0: Output pins**
  These bits define the direction of each I/O pin. When bit DDn is set(one), the port pin is an output. When bit DDn is cleared(zero), the port pin is an input.
- **Bit 2..0 - OUT2,OUT1,OUT0: Output buffer register**
  These bits represent the contents of the output buffer register.

2603 GPIO Port C

The I/O cells associated with GPIO Port C are all individually configurable. See the programmable I/O cell configuration section for more details. Because all 8 pins are individually configurable for options such as pull-ups/pull-downs, and Schmitt triggered inputs, this port is often used for functions that require different configuration setup.

Port C Input Pins Data Register GPCin

Address: 0x24, IOpage: 0/1
Access Mode: Read only

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	PIN7	PIN6	PIN5	PIN4	PIN3	PIN2	PIN1	PIN0

- **Bit 15..8 - Res: Reserved**
  These bits are reserved.
- **Bit 7..0 - PIN7, PIN6, PIN5, PIN4, PIN3, PIN2, PIN1, PIN0: Input pins**
  These bits represent the logical level present at the external pin.

Port C Output Data Register GPCout

Address: 0x24, IOpage: 0/1
Access Mode: Write only

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	OUT7	OUT6	OUT5	OUT4	OUT3	OUT2	OUT1	OUT0

- **Bit 15..8 - Res: Reserved**
  These bits are reserved.
- **Bit 7..0 - OUT7,OUT6,OUT5,OUT4,OUT3,OUT2,OUT1,OUT0: Output buffer register**
  The value of these bits are latched in the output buffer register.

Port C Direction/Data Register GPCcfg

Address: 0x25, IOpage: 0/1
Access Mode: Read/Write

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DD7	DD6	DD5	DD4	DD3	DD2	DD1	DD0	OUT7	OUT6	OUT5	OUT4	OUT3	OUT2	OUT1	OUT0

- **Bit 15..8 - DD7, DD6, DD5, DD4, DD3, DD2, DD1, DD0: Output pins**
  These bits define the direction of each I/O pin. When bit DDn is set(one), the port pin is an output. When bit DDn is cleared(zero), the port pin is an input.
- **Bit 7..0 - OUT7,OUT6,OUT5,OUT4,OUT3,OUT2,OUT1,OUT0: Output buffer register**
  These bits represent the contents of the output buffer register.

2604 GPIO Port D

The I/O cells associated with GPIO Port D are all individually configurable. See the programmable I/O cell configuration section for more details. Because all 8 pins are individually configurable for options such as pull-ups/pull-downs, and Schmitt triggered inputs, this port is often used for functions that require different configuration setup.

Port D Input Pins Data Register GPDin

Address: 0x26, IOpage: 0/1
Access Mode: Read only

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	PIN7	PIN6	PIN5	PIN4	PIN3	PIN2	PIN1	PIN0

- **Bit 15..8 - Res: Reserved**
  These bits are reserved.
- **Bit 7..0 - PIN7, PIN6, PIN5, PIN4, PIN3, PIN2, PIN1, PIN0: Input pins**
  These bits represent the logical level present at the external pin.

Port D Output Data Register GPDout

Address: 0x26, IOpage: 0/1
Access Mode: Write only

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	OUT7	OUT6	OUT5	OUT4	OUT3	OUT2	OUT1	OUT0

- **Bit 15..8 - Res: Reserved**
  These bits are reserved.
- **Bit 7..0 - OUT7,OUT6,OUT5,OUT4,OUT3,OUT2,OUT1,OUT0: Output buffer register**
  The value of these bits are latched in the output buffer register.

Port D Direction/Data Register GPDcfg

Address: 0x27, IOpage: 0/1
Access Mode: Read/Write

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DD7	DD6	DD5	DD4	DD3	DD2	DD1	DD0	OUT7	OUT6	OUT5	OUT4	OUT3	OUT2	OUT1	OUT0

- **Bit 15..8 - DD7, DD6, DD5, DD4, DD3, DD2, DD1, DD0: Output pins**
  These bits define the direction of each I/O pin. When bit DDn is set(one), the port pin is an output. When bit DDn is cleared(zero), the port pin is an input.
- **Bit 7..0 - OUT7,OUT6,OUT5,OUT4,OUT3,OUT2,OUT1,OUT0: Output buffer register**
  These bits represent the contents of the output buffer register.

Alternate Function of Port D pins

The alternate pin configuration is as follows:

- **PD0 - MISO1:**
  Master Data Input, Slave Data Output pin for SPI1. When SPI1 is enabled as a master, this pin is configured as an input. When SPI1 is enabled as a slave, the data direction of this pin is controlled by the level of the slave select. See the description of the SPI ports for further details.
- **PD1 - MOSI1:**
  Master Data Output, Slave Data Input pin for SPI1. When SPI1 is enabled as a master, this pin is configured as an output. When SPI1 is enabled as a slave,

this pin is configured as an input. See the description of the SPI ports for further details.

- **PD2 - SCK1:**
  Master Clock Output, Slave Clock Input pin for SPI1. When SPI1 is enabled as a master, this pin is configured as an output. When SPI1 is enabled as a slave, this pin is configured as an input. See the description of the SPI ports for further details.

2605 GPIO Port E

The I/O cells associated with GPIO Port E are all configured with one register. See the programmable I/O cell configuration section for more details. Because all 8 pins are configured the same, this port is often used for functions that require similar configuration setup should the system require it.

Port E Input Pins Data Register GPEin

Address: 0x28, IOpage: 0/1
Access Mode: Read only

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	PIN7	PIN6	PIN5	PIN4	PIN3	PIN2	PIN1	PIN0

- **Bit 15..8 - Res: Reserved**
  These bits are reserved.
- **Bit 7..0 - PIN7, PIN6, PIN5, PIN4, PIN3, PIN2, PIN1, PIN0: Input pins**
  These bits represent the logical level present at the external pin.

Port E Output Data Register GPEout

Address: 0x28, IOpage: 0/1
Access Mode: Write only

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	OUT7	OUT6	OUT5	OUT4	OUT3	OUT2	OUT1	OUT0

- **Bit 15..8 - Res: Reserved**
  These bits are reserved.
- **Bit 7..0 - OUT7,OUT6,OUT5,OUT4,OUT3,OUT2,OUT1,OUT0: Output buffer register**
  The value of these bits are latched in the output buffer register.

Port E Direction/Data Register GPEcfg

Address: 0x29, IOpage: 0/1
Access Mode: Read/Write

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DD7	DD6	DD5	DD4	DD3	DD2	DD1	DD0	OUT7	OUT6	OUT5	OUT4	OUT3	OUT2	OUT1	OUT0

- **Bit 15..8 - DD7, DD6, DD5, DD4, DD3, DD2, DD1, DD0: Output pins**
  These bits define the direction of each I/O pin. When bit DDn is set(one), the port pin is an output. When bit DDn is cleared(zero), the port pin is an input.
- **Bit 7..0 - OUT7,OUT6,OUT5,OUT4,OUT3,OUT2,OUT1,OUT0: Output buffer register**
  These bits represent the contents of the output buffer register.

2606 GPIO Port F

The I/O cells associated with GPIO Port F are all configured with one register. See the programmable I/O cell configuration section for more details. Because all 2 pins are configured the same, this port is often used for functions that require similar configuration setup should the system require it.

Port F Input Pins Data Register GPFin

Address: 0x2A, IOpage: 0/1
Access Mode: Read only

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	PIN1	PIN0

- **Bit 15..2 - Res: Reserved**
  These bits are reserved.
- **Bit 1..0 - PIN1, PIN0: Input pins**
  These bits represent the logical level present at the external pin.

Port F Output Data Register GPFout

Address: 0x2A, IOpage: 0/1
Access Mode: Write only

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	-	-	-	-	-	-	OUT1	OUT0

- **Bit 15..2 - Res: Reserved**
  These bits are reserved.
- **Bit 1..0 - OUT1,OUT0: Output buffer register**
  The value of these bits are latched in the output buffer register.

Port F Direction/Data Register GPFcfg

Address: 0x2B, IOpage: 0/1
Access Mode: Read/Write

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	DD1	DD0	-	-	-	-	-	-	OUT1	OUT0

- **Bit 9..8 - DD1, DD0: Output pins**
  These bits define the direction of each I/O pin. When bit DDn is set(one), the port pin is an output. When bit DDn is cleared(zero), the port pin is an input.
- **Bit 1..0 - OUT1,OUT0: Output buffer register**
  These bits represent the contents of the output buffer register.

2607 GPIO Port G

The I/O cells associated with GPIO Port G are all individually configurable. See the programmable I/O cell configuration section for more details. Because all 4 pins are individually configurable for options such as pull-ups/pull-downs, and Schmitt triggered inputs, this port is often used for functions that require different configuration setup.

Port G Input Pins Data Register GPGin

Address: 0x2C, IOpage: 0/1
Access Mode: Read only

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	PIN3	PIN2	PIN1	PIN0

- **Bit 15..4 - Res: Reserved**
  These bits are reserved.
- **Bit 3..0 - PIN3, PIN2, PIN1, PIN0: Input pins**
  These bits represent the logical level present at the external pin.

Port G Output Data Register GPGout

Address: 0x2C, IOpage: 0/1
Access Mode: Write only

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	-	-	-	-	OUT3	OUT2	OUT1	OUT0

- **Bit 15..4 - Res: Reserved**
  These bits are reserved.
- **Bit 3..0 - OUT3,OUT2,OUT1,OUT0: Output buffer register**
  The value of these bits are latched in the output buffer register.

Port G Direction/Data Register GPGcfg

Address: 0x2D, IOpage: 0/1
Access Mode: Read/Write

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	DD3	DD2	DD1	DD0	-	-	-	-	OUT3	OUT2	OUT1	OUT0

- **Bit 11..8 - DD3, DD2, DD1, DD0: Output pins**
  These bits define the direction of each I/O pin. When bit DDn is set(one), the port pin is an output. When bit DDn is cleared(zero), the port pin is an input.
- **Bit 3..0 - OUT3,OUT2,OUT1,OUT0: Output buffer register**
  These bits represent the contents of the output buffer register.

2608 GPIO Port H

The I/O cells associated with GPIO Port H are all configured with one register. See the programmable I/O cell configuration section for more details. Because all 3 pins are configured the same, this port is often used for functions that require similar configuration setup should the system require it.

Port H Input Pins Data Register GPHin

Address: 0x2E, IOpage: 0/1
Access Mode: Read only

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	PIN2	PIN1	PIN0

- **Bit 15..3 - Res: Reserved**
  These bits are reserved.
- **Bit 2..0 - PIN2, PIN1, PIN0: Input pins**
  These bits represent the logical level present at the external pin.

Port H Output Data Register GPHout

Address: 0x2E, IOpage: 0/1
Access Mode: Write only

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	-	-	-	-	-	OUT2	OUT1	OUT0

- **Bit 15..3 - Res: Reserved**
  These bits are reserved.
- **Bit 2..0 - OUT2,OUT1,OUT0: Output buffer register**
  The value of these bits are latched in the output buffer register.

Port H Direction/Data Register GPHcfg

Address: 0x2F, IOpage: 0/1
Access Mode: Read/Write

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	DD2	DD1	DD0	-	-	-	-	-	OUT2	OUT1	OUT0

- **Bit 10..8 - DD2, DD1, DD0: Output pins**
  These bits define the direction of each I/O pin. When bit DDn is set(one), the port pin is an output. When bit DDn is cleared(zero), the port pin is an input.
- **Bit 2..0 - OUT2,OUT1,OUT0: Output buffer register**
  These bits represent the contents of the output buffer register.

2609 GPIO Port I

The I/O cells associated with GPIO Port I are all configured with one register. See the programmable I/O cell configuration section for more details. Because all 3 pins are configured the same, this port is often used for functions that require similar configuration setup should the system require it.

Port I Input Pins Data Register GPIin

Address: 0x30, IOpage: 0/1
Access Mode: Read only

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	PIN2	PIN1	PIN0

- **Bit 15..7 - Res: Reserved**
  These bits are reserved.
- **Bit 2..0 - PIN2, PIN1, PIN0: Input pins**
  These bits represent the logical level present at the external pin.

Port I Output Data Register GPIout

Address: 0x30, IOpage: 0/1
Access Mode: Write only

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	-	-	-	-	-	OUT2	OUT1	OUT0

- **Bit 15..3 - Res: Reserved**
  These bits are reserved.
- **Bit 2..0 - OUT2,OUT1,OUT0: Output buffer register**
  The value of these bits are latched in the output buffer register.

Port I Direction/Data Register GPIcfg

Address: 0x31, IOpage: 0/1
Access Mode: Read/Write

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	DD2	DD1	DD0	-	-	-	-	-	OUT2	OUT1	OUT0

- **Bit 10..8 - DD2, DD1, DD0: Output pins**
  These bits define the direction of each I/O pin. When bit DDn is set(one), the port pin is an output. When bit DDn is cleared(zero), the port pin is an input.
- **Bit 2..0 - OUT2,OUT1,OUT0: Output buffer register**
  These bits represent the contents of the output buffer register.

2610 GPIO Port J

The I/O cells associated with GPIO Port J are all configured with one register. See the programmable I/O cell configuration section for more details. Because all 4 pins are configured the same, this port is often used for functions that require similar configuration setup should the system require it.

Port J Input Pins Data Register GPJin

Address: 0x32, IOpage: 0/1
Access Mode: Read only

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	PIN3	PIN2	PIN1	PIN0

- **Bit 15..4 - Res: Reserved**
  These bits are reserved.
- **Bit 3..0 - PIN3, PIN2, PIN1, PIN0: Input pins**
  These bits represent the logical level present at the external pin.

Port J Output Data Register GPJout

Address: 0x32, IOpage: 0/1
Access Mode: Write only

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	-	-	-	-	OUT3	OUT2	OUT1	OUT0

- **Bit 15..4 - Res: Reserved**
  These bits are reserved.
- **Bit 3..0 - OUT3,OUT2,OUT1,OUT0: Output buffer register**
  The value of these bits are latched in the output buffer register.

Port J Direction/Data Register GPJcfg

Address: 0x33, IOpage: 0/1
Access Mode: Read/Write

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	DD3	DD2	DD1	DD0	-	-	-	-	OUT3	OUT2	OUT1	OUT0

- **Bit 11..8 - DD3, DD2, DD1, DD0: Output pins**
  These bits define the direction of each I/O pin. When bit DDn is set(one), the port pin is an output. When bit DDn is cleared(zero), the port pin is an input.
- **Bit 3..0 - OUT3,OUT2,OUT1,OUT0: Output buffer register**
  These bits represent the contents of the output buffer register.

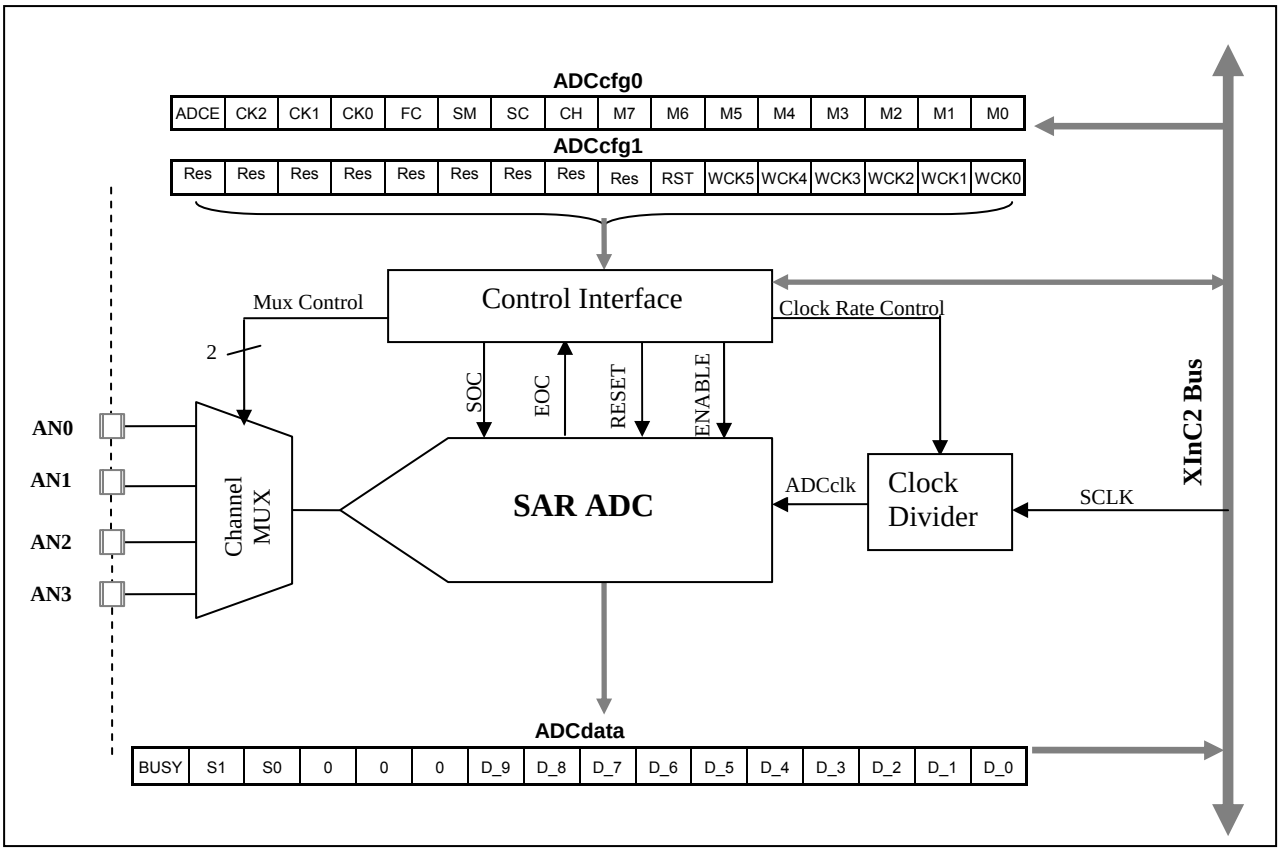
2700 ADC

Introduction XInC2 has an on chip 10-bit successive approximation analog-to-digital converter with all the biasing and sample and hold circuitry included, with only the need of an external reference voltage to define the full-scale range. The ADC can sample data from one of the four input channels. It has a built-in controller interface for continuous sampling of single or multiple channels. A power down mode is provided with less than 1uA of standby current. The result of conversion is stored in ADCdata register. The ADC is controlled through ADCcfg0 and ADCcfg1 registers.

Features

- 10-bit resolution
- 4 Input Channels
- Low Power Dissipation (2mW)
- Power-Down Capability (1uA)
- Up to 44.64 KHz Conversion rate
- Continuous sampling capability

The block diagram of the ADC is shown below.



ADC Functional Diagram

Conversion Clock The ADC conversion clock source (ADCclk) is configured in the ADCcfg0 register. The clock source is a divisor of the Sclk. The ADCclk is used for A/D conversion and to generate the sampling period. The ADCclk frequency must not exceed the maximum frequency of 625KHz as specified in the electrical data tables. A total of 14 ADCclks are required to complete a conversion and store the result in ADCdata register.

Power Down The ADC core is enabled by the ADCE bit. The ADC can be configured in power-down mode (<1uA), by clearing this bit.

Caution! ADC turn on time When the ADC is turned on with the ADCE bit, the turn on time of 2ms must be observed before any conversion is started. Otherwise, the result will be invalid. Apart from this, ADC needs to be reset for at least 3 ADC clock ticks before starting the first conversion after being turned on. Reset is applied by setting the RST bit in ADCcfg1 register.

ADC Parameters The A/D conversion is monotonic, meaning that the digital output value always increases (0-3FF) with increase in input (0-V_{REF}) analog voltage; and linear, meaning the ADC characteristic is close to ideal 10-bit ADC characteristic. INL and DNL values could be found in electrical characteristics data table. Thing to note here is that there is a voltage drop (V_{DI}) across the protection diode (this is on-chip) connected to the V_{REF} pin. This voltage is about 161mv for a V_{REF} of 1.826V. This drop reduces the input voltage range from V_{REF} to V_{REF}-$V_{ID.}$ If the input voltage is greater than V_{REF}-V_{ID} then a result 3FF is obtained in the ADCdata register. Similarly any analog input voltage less than GNDADC results in a 0 value in the ADCdata register.

A/D Conversion Setting the SC bit in the ADCcfg0 register starts the conversion process. Two modes for sampling are provided viz. single sampling mode (SSM) and continuous sampling mode (CSM). In SSM, when the SC bit is set, SOC (start-of-conversion) signal is provided to the ADC by the control interface and the ADC finishes the sampling in 14 clock ticks. The channel sampled is the one provided in bits M0 M1 of ADCcfg0 register. There are two methods to determine if the conversion process has been finished namely Flow Control (FC) and ADC Busy Flag (Busy). In FC the tread processor keeps on trying to read ADCdata register and is blocked till the conversion process is complete. This is achieved by setting (writing 1) the FC bit in ADCcfg0 register. This is an easier way of reading without much worrying about the conversion timings. Another approach is to clear (zero) the FC bit. This will not block the thread processor to read the ADCdata register. However the data in this register will only be valid when the Busy flag is cleared by the ADC control Interface (which indicates completion of the conversion). This is useful when the user wants to optimize the timings for reading. The thread processor does not have to wait (can execute other tasks) till the conversion is complete.

In CSM the ADC starts sampling the channels provided in bits M1M0, M3M2, M5M4, M7M6 in that order one after other continuously. Additional time delay between two consecutive sampling can be introduced by appropriately setting the wait clock rates in ADCcfg1 register. This facilitates multiple channel sampling without any intervening required by the thread processor. This is especially useful if the inputs on the channels are high frequency variable signals. Similar to that in SSM, Flow Control as well as Busy Flag can be used for reading the ADCdata register.

ADC Configuration Address: 0x24
ADCcfg0 Access Mode: Read/Write

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADCE	CK2	CK1	CK0	FC	SM	SC	CH	M7	M6	M5	M4	M3	M2	M1	M0

- **Bit 15 - ADCE: ADC Enable**
  When this bit is set (one), the ADC is enabled. When this bit is cleared (zero), the ADC is disabled. When the ADC is disabled, power consumption reduces drastically.
- **Bit 14,13,12 - CK2,CK1,CK0: ADC Clock Rate select**
  These three bits control the ADC clock rate of the device. The relationship between ADCclk and the system clock (Sclk) frequency is shown below:

CK2	CK1	CK0	ADCCIK
0	0	0	Sclk / 2
0	0	1	Sclk / 4
0	1	0	Sclk / 8
0	1	1	Sclk / 16
1	0	0	Sclk / 32
1	0	1	Sclk / 64
1	1	0	Sclk / 128
1	1	1	Sclk / 256

- **Bit 11 - FC: Flow control enable**
  When this bit is set (one), the ADC will use flow control. When this bit is cleared (zero), the flow control is disabled and the ADC busy flag can be polled for the end of conversion.
- **Bit 10 - SM: Sampling Mode**
  When this bit is set (one), the ADC will operate in continuous sampling mode. When this bit is cleared (zero), the ADC will operate in single conversion mode.
- **Bit 9 - SC: Start Conversion**
  When this bit is set (one), the ADC will start the conversion process. It is necessary to configure all the registers and this bit should be set in the end (This also means rewriting the other bits as before to ensure the right configuration for the whole register (ADCcfg0). Writing back a zero will do nothing in single conversion mode and will stop the conversion in continuous sampling mode.
- **Bit 8 - CH: Channel Type**
  When this bit is set (one), only three channels specified from bits 0 to 5 (2 bits for each channel) will be sampled in a continuous sampling mode. When this bit is cleared (zero)all the four channels from bits 0 to 7(2 bits for each channel) will be sampled. This bit is ignored in single sampling mode.
- **Bit 7,6,5,4,3,2,1,0 – M7..M0: Mux Select Lines**

  M1, M0 – Channel One
  M3, M2 – Channel Two
  M5, M4 – Channel Three
  M7, M6 – Channel Four
  These bits specify the channel numbers to be sampled in continuous sampling mode and will be sampled continuously from Channel one to four (M7..M0) when CH bit is cleared (zero) and from Channel one to three (M5..M0) when CH bit is set (one).
  In single sampling mode the Channel One specified in bits M1, M0 will be sampled.

ADC Configuration Address: 0x25
ADCcfg1 Access Mode: Read/Write

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res	Res	Res	Res	Res	Res	Res	Res	Res	RST	WCK5	WCK4	WCK3	WCK2	WCK1	WCK0

- **BBit 5,4,3,2,1,0 – WCK2,WCK1,WCK0: ADC Wait Clock Rate select**
  These four bits control the number of clocks that occur after a sample has been completed before the start of the next conversion cycle in a continuous sampling mode. These bits are ignored in single sampling mode.

WCK5	WCK4	WCK3	WCK2	WCK1	WCK0	Wait clocks
0	0	0	0	0	0	0
0	0	0	0	0	1	1
0	0	0	0	1	0	2
:	:	:	:	:	:	:
:	:	:	:	:	:	:
1	1	1	1	0	1	61
1	1	1	1	1	0	62
1	1	1	1	1	1	63

- **Bit 6 - RST: Reset**
  When the ADC has powered up it needs to be Reset for 3 ADC Clock cycles. Hence this bit should be set (one) and cleared(zero) before starting to sample first sample.
- **Bit 7,8,…15 - Res: Reserved**
  These bits are reserved.

ADC Data / Status Address: 0x25
ADCdata Access Mode: Read Only

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BUSY	S1	S0	0	0	0	D_9	D_8	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0

- **Bit 15 - BUSY: ADC busy flag**
  Once a conversion has started this bit will read one while the ADC is doing a conversion.
- **Bit 14..13 – Channel Number**
  In continuos sampling mode these bits will indicate the channel number of the converted sample.
- **Bit 12..10 - Res: Reserved**
  These bits are reserved.
- **Bit 9..0 – D_9, D_8, D_7, D_6, D_5, D_4, D_3, D_2, D_1, D_0: 10-bit converted sample**
  This is the10-bit conversion sample. If flow control is enabled then reading ADCdata will block until the conversion is complete.

2800 I/O Pin Function Overloading Summary

Many of the I/O pins are overloaded with a second peripheral functions.
Almost all of the I/O pins can be configured as General Purpose I/O (GPIO).

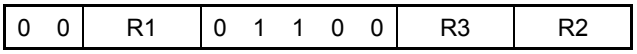
2900 Instruction Set

The XInC2 instruction set architecture is summarized below:

Instruction	Description	Available Address Modes
bc	conditional branch	PC relative
bra	unconditional branch	PC relative
jsr	jump to subroutine	register indirect, absolute
thrd	get thread number	register
ld	load from RAM	base displacement, absolute
st	store to RAM	base displacement, absolute
inp	read input port	immediate
outp	write output port	immediate
mov	move immediate	immediate
add	2's complement add	register, immediate
sub	2's complement subtract	register
rol	bitwise rotate left	register, immediate
and	bitwise and	register, immediate
ior	bitwise inclusive or	register, immediate
xor	bitwise exclusive or	register, immediate
bic	bit clear	immediate
bis	bit set	immediate
bix	bit change	immediate

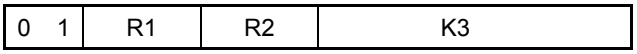
2901 add - 2's Complement Add

R1 = R2 + R3



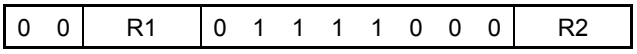
Format 1
register

R1 = R2 + K3

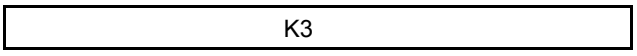


Format 2
immediate
K3 = [-128:127]

R1 = R2 + K3



Format 3
immediate



Instruction Fields

R1 3-bit specifier for destination register
R2 3-bit specifier for source register
R3 3-bit specifier for source register
K3 signed 8-bit or 16-bit literal source

Description

Add the source operands and write the result to the destination register R1. The address mode is either register (R3) or immediate (K3).

This instruction can be used to construct an immediate-mode subtract.

Note that for the singular case of the following signed subtraction:
R1 = R2 - (-32768)
the transformation into:
R1 = R2 + 32768
produces a misleading condition code, since the literal operand is unrepresentable as a 16-bit 2's complement number.

The format 1 add instruction can be used to construct a bitwise shift-left. The carry indication is useful for multiword shifts.

Condition Codes

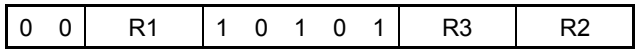
n Set if result is negative, i.e. msb is 1
z Set if result is zero
v Set if an overflow is generated
c Set if a carry is generated

Example

```
add  r0,r0,r1
add  r0,r1,16
add  r0,r1,0x1234
```

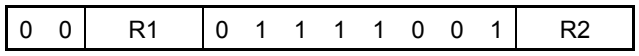
2902 and - Bitwise And

R1 = R2 & R3

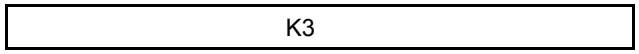


Format 1 register

R1 = R2 & K3



Format 2 immediate



Instruction Fields

R1 3-bit specifier for destination register
R2 3-bit specifier for source register
R3 3-bit specifier for source register
K3 16-bit literal source

Description

Bitwise-and the source operands and write the result to the destination register R1. The address mode is either register (R3) or immediate (K3).

Where the literal source K3 consists of a single 0 bit in a background of 1s, the bitwise-and instruction can be transformed into an equivalent bit-clear instruction.

Condition Codes

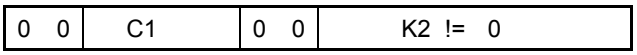
n Set if result is negative, i.e. msb is 1
z Set if result is zero
v Set if (R2 | R3) != R3, or alternatively if (R2 | K3) != K3
c Set if result is in the interval [1:255]

Example

```
and  r0,r0,r1
and  r0,r1,0x00FF
```

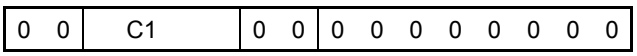
2903 bc - Conditional Branch

if (conditional (C1)) PC += K2



Format 1
PC relative
K2 = [-128:127]

if (conditional (C1)) PC += K2



Format 2
PC relative

Instruction Fields C1 4-bit specifier for branch condition
K2 signed 8-bit or 16-bit literal source

Description Evaluate the specified branch condition (C1) using the n, z, v, and c bits of the condition code (CC) register. See the interpretation of C1, as given below. If the specified branch condition is met, add the source operand to the program counter (PC) register. Otherwise, the program counter (PC) is not affected, i.e. it points at the next instruction. Treating the next fetched instruction as a memory operand, the address mode is PC relative.

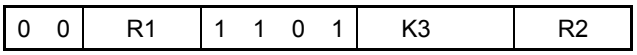
Interpretation of C1

C1	condition(C1)	Test	Signed Comparison	Unsigned Comparison
0x0	c			<
0x1	v			
0x2	z	==0	==	==
0x3	n	<0		
0x4	c | z			<=
0x5	n ^ v		<	
0x6	(n^v) | z		<=	
0x7	n | z	<=0		
0x8	!c			>=
0x9	!v			
0xA	!z	!=0	!=	!=
0xB	!n	>=0		
0xC	!(c | z)			>
0xD	!(n ^ v)		>=	
0xE	!((n^v) | z)		>	
0xF	!(n | z)	>0		

Condition Codes Not affected.

2904 bic - Bit Clear

R1 = R2 & ~(1<<K3)



Format 1
immediate
[K3 = [0:15]

Instruction Fields

R1 3-bit specifier for destination register
R2 3-bit specifier for source register
K3 4-bit literal source

Description

Select a single bit of the source operand R2 using the immediate operand K3, test the selected bit, clear the selected bit, and write the result to the destination register R1. The bits of R2 are numbered 15:0, with bit 0 the lsb. The address mode is immediate (K3).

Condition Codes

n Set if result is negative, i.e. msb is 1
z Set if result is zero
v Set if the selected bit was 1 when it was tested
c Set if result is in the interval [1:255]

Example

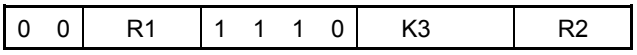
```
        bic   r0,r0,15   // clear msb

        bic   r0,r1,5    // test bit 5
        bc    VS,bitset
        …

bitset: …
```

2905 bis - Bit Set

R1 = R2 | (1<<K3)



Format 1
immediate
[K3 = [0:15]

Instruction Fields

R1 3-bit specifier for destination register
R2 3-bit specifier for source register
K3 4-bit literal source

Description

Select a single bit of the source operand R2 using the immediate operand K3, test the selected bit, set the selected bit, and write the result to the destination register R1. The bits of R2 are numbered 15:0, with bit 0 the lsb. The address mode is immediate (K3).

Condition Codes

n Set if result is negative, i.e. msb is 1
z Set if result is zero
v Set if the selected bit was 1 when it was tested
c Set if result is in the interval [1:255]

Example

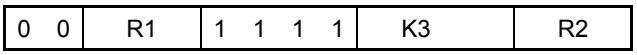
```
        bis   r0,r0,0   // set lsb

        bis   r0,r1,5     // test bit 5
        bc    VC,bitcleared
        …

bitclear: …
```

2906 bix - Bit Change

R1 = R2 ^ (1<<K3)



Format 1
immediate
[K3 = [0:15]

Instruction Fields

R1 3-bit specifier for destination register
R2 3-bit specifier for source register
K3 4-bit literal source

Description

Select a single bit of the source operand R2 using the immediate operand K3, test the selected bit, change the selected bit, and write the result to the destination register R1. The bits of R2 are numbered 15:0, with bit 0 the lsb. The address mode is immediate (K3).

Condition Codes

n Set if result is negative, i.e. msb is 1
z Set if result is zero
v Set if the selected bit was 1 when it was tested
c Set if result is in the interval [1:255]

Example

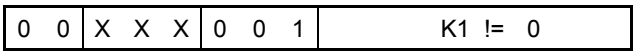
```
        bix   r0,r0,0   // change lsb

        bix   r0,r1,5     // test bit 5
        bc    VC,bitcleared
        …

bitclear: …
```

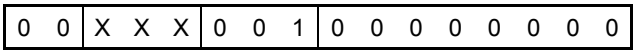
2907 bra - Unconditional Branch

PC += K1

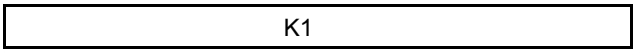


Format 1
PC relative
K1 = [-128:127]

PC += K1



Format 2
PC relative



Instruction Fields K1 signed 8-bit or 16-bit literal source

Description Add the source operand to the program counter (PC) register. Treating the next fetched instruction as a memory operand, the address mode is PC relative.

For each of formats 1 and 2, there are 3 bit positions defined as X (don't care).

Condition Codes Not affected

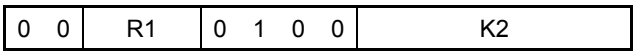
Example

```
        bra  there
        …

there:  …
```

2908 inp - Read Input Port

R1 = inp(K2)



Format 1
immediate
K2 = [0:127]

Instruction Fields

R1 3-bit specifier for destination register
K2 unsigned 7-bit literal source

Description

Read the input port at I/O address K2, and write the result to the destination register R1. The address mode is immediate (K2).

Some I/O devices can signal a “wait” condition. When a XInC2 I/O device signals “wait”, the I/O cycle is canceled. Both the thread processor and the selected device proceed as though the I/O cycle had never occurred.

Consequently, the thread processor retries the I/O instruction until the wait condition clears. This results in 1 I/O retry cycle every 8 clock ticks. Other threads and I/O processes proceed without interference.

Condition Codes

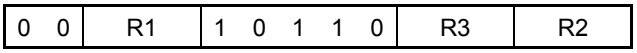
n Set if result is negative, i.e. msb is 1
z Set if result is zero
v Set if result is odd, i.e. lsb is 1
c Set if result is in the interval [1:255]

Example

```
inp   r0,SCUtime   //read the SCU timer register
```

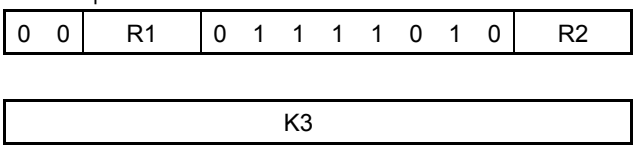
2909 ior - Bitwise Inclusive Or

R1 = R2 | R3



Format 1 register

R1 = R2 | K3



Format 2 immediate

Instruction Fields

R1 3-bit specifier for destination register
R2 3-bit specifier for source register
R3 3-bit specifier for source register
K3 16-bit literal source

Description

Bitwise-inclusive-or the source operands and write the result to the destination register R1. The address mode is either register (R3) or immediate (K3).

Where the literal source K3 consists of a single 1 bit in a background of 0s, the bitwise-inclusive-or instruction can be transformed into an equivalent bit-set instruction.

Condition Codes

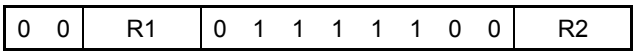
n Set if result is negative, i.e. msb is 1
z Set if result is zero
v Set if (R2 & R3) == R3, or alternatively if (R2 & K3) == K3
c Set if result is in the interval [1:255]

Example

```
ior   r0,r0,r1
ior   r0,r1,0x8001
```

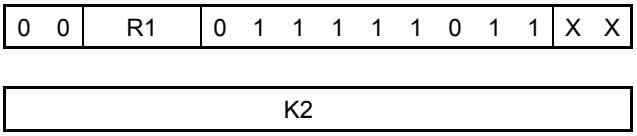
2910 jsr - Jump to Subroutine / Return from Subroutine

T = R2; R1 = PC; PC = T



Format 1
register indirect
T is a temporary

T = K2; R1 = PC; PC = T



Format 2
absolute

Instruction Fields R1 3-bit specifier for destination register
R2 3-bit specifier for source register
K2 16-bit literal source

Description Save the source operand in a temporary T, write the program counter (PC) to the destination register R1, and write the temporary T to the program counter (PC) register. Treating the next fetched instruction as a memory operand, the address mode is either register indirect (R2) or absolute (K2).

For format 2, there are 2 bit positions defined as X (don't care). One use for the X bits is the construction of copyright traps.

The jump-to-subroutine instruction subsumes jump instructions (discarding the R1 destination), and also the return instruction found in other architectures.

Condition Codes Not affected.

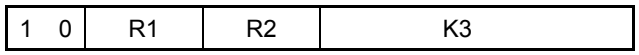
Example

```
        jsr   r6, routine    // jump to subroutine
        …

routine: …
        jsr   r6,r6          // return from subroutine
```

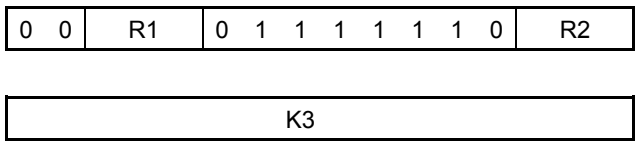
2911 ld - Load from RAM

R1 = *(R2 + K3)



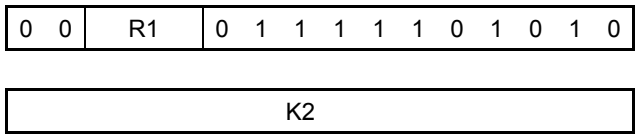
Format 1
base displacement
absolute indexed
K3 = [-128:127]

R1 = *(R2 + K3)



Format 1
base displacement
absolute indexed

R1 = *K2



Format 3
absolute

Instruction Fields

R1 3-bit specifier for destination register
R2 3-bit specifier for base register
K3 signed 8-bit or 16-bit displacement
K2 16-bit absolute address

Description

For formats 1 and 2, add the base register R2 and the displacement K3 to form the address of the RAM source. For format 3, K2 is the address of the RAM source. Read the RAM source and write the result to the destination register R1. The address mode is either base displacement (R2+K3) or absolute (K2).

Note that absolute indexed is a synonym for base displacement. The format 1 load instruction can efficiently express register indirect addressing.

Condition Codes

n Set if result is negative, i.e. msb is 1
z Set if result is zero
v Set if result is odd, i.e. lsb is 1
c Set if result is in the interval [1:255]

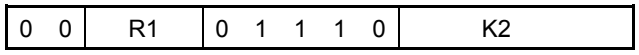
Example

```
ld   r0,r1,1

ld   r0,0xF000
```

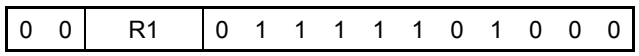
2912 mov - Move Immediate

R1 = K2

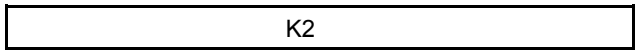


Format 1
immediate
K2 = [-32:31]

R1 = K2



Format 2
immediate



Instruction Fields R1 3-bit specifier for destination register
K2 signed 6-bit or 16-bit literal source

Description Write the source value K2 to the destination register R1. The address mode is immediate (K2).

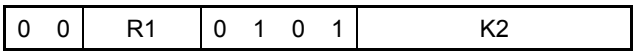
Condition Codes Not affected.

Example

```
mov  r0,15
mov  r0,0x1234
```

2913 outp - Write Output Port

R1 = inp(K2)



Format 1
immediate
K2 = [0:127]

Instruction Fields R1 3-bit specifier for source register
K2 unsigned 7-bit literal source

Description Read the source operand R1 and write the result to the output port at I/O address K2. The address mode is immediate (K2).

Some I/O devices can signal a “wait” condition. When a XInC2 I/O device signals “wait”, the I/O cycle is canceled. Both the thread processor and the selected device proceed as though the I/O cycle had never occurred.

Consequently, the thread processor retries the I/O instruction until the wait condition clears. This results in 1 I/O retry cycle every 8 clock ticks. Other threads and I/O processes proceed without interference.

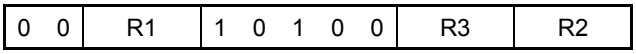
Condition Codes Not affected.

Example

```
outp   r0,SPI0cfg   //write to the SPI0 configuration register
```

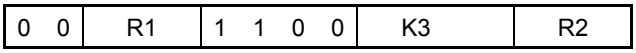
2914 rol - Bitwise Rotate Left

R1 = R2 << R3



Format 1
register

R1 = R2 << K3



Format 2
immediate
K3 = [0:15]

Instruction Fields

R1 3-bit specifier for destination register
R2 3-bit specifier for source register
R3 3-bit specifier for source register
K3 4-bit literal source

Description

Bitwise-rotate the source operand R2 left n positions, and write the result to the destination register R1. The amount n of the rotation is given by either R3 or K3, modulo 16. The address mode is either register (R3) or immediate (K3).

The bits of R2 are numbered 15:0, with bit 0 the lsb. For the rotate-left operation, each ith bit of the R2 source is copied to the ((i+n)%16)th bit position within the destination. The direction of the rotation can be changed from left to right by negating n.

The rotate-left instruction can be used to construct rotate-right, bitwise shifts, arithmetic shift-right, bit-field accesses, byte accesses, etc. It is also a building block for many arithmetic operations.

Condition Codes

n Set if result is negative, i.e. msb is 1
z Set if result is zero
v Set if result is odd, i.e. lsb is 1
c Set if result is in the interval [1:255]

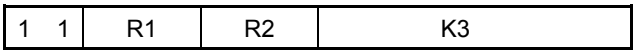
Example

```
rol   r0,r0,r1

rol   r0,r0,4      // rotate left 4
rol   r0,r0,-4     // rotate right 4
```

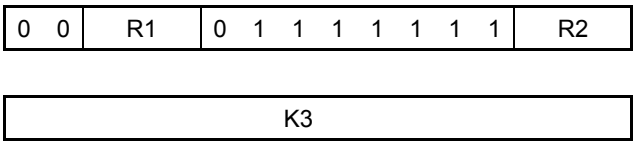
2915 st - Store to RAM

*(R2 + K3) = R1



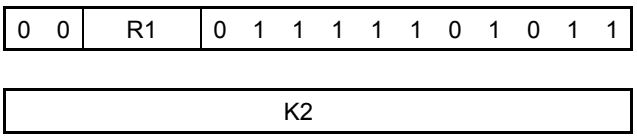
Format 1
base displacement
absolute indexed
K3 = [-128:127]

*(R2 + K3) = R1



Format 1
base displacement
absolute indexed

R1 = *K2



Format 3
absolute

Instruction Fields R1 3-bit specifier for destination register
R2 3-bit specifier for base register
K3 signed 8-bit or 16-bit displacement
K2 16-bit absolute address

Description For formats 1 and 2, add the base register R2 and the displacement K3 to form the address of the RAM destination. For format 3, K2 is the address of the RAM destination. Read the source register R1 and write the result to the RAM destination. The address mode is either base displacement (R2+K3) or absolute (K2).

Note that absolute indexed is a synonym for base displacement. The format 1 store instruction can efficiently express register indirect addressing.

Condition Codes Not affected.

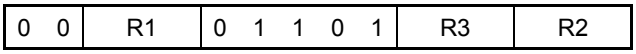
Example

```
st   r0,r1,1

st   r0,0xF000
```

2916 sub - 2's Complement Subtract

R1 = R2 - R3



Format 1 register

Instruction Fields

R1 3-bit specifier for destination register
R2 3-bit specifier for source register
R3 3-bit specifier for source register

Description

Subtract the source operands R2-R3 and write the result to the destination register R1. The address mode is register (R3).

Both signed and unsigned comparisons can be constructed by combining a subtract instruction with a conditional branch. An immediate-mode subtract can be constructed from the add instruction.

Condition Codes

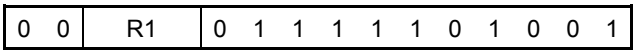
n Set if result is negative, i.e. msb is 1
z Set if result is zero
v Set if an overflow is generated
c Set if a carry is generated

Example

```
sub   r0,r1,r2
```

2917 thrd - Get Thread Number

R1 = thrd()



Format 1 register

Instruction Fields R1 3-bit specifier for destination register

Description Write the thread number to the destination register R1. The address mode is register (R1).

Each thread processor is assigned a unique number in the interval [0:7]. The thread number breaks the symmetry between thread processors.

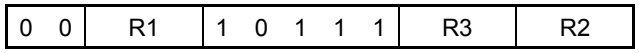
Condition Codes Not affected.

Example

```
thrd   r0
```

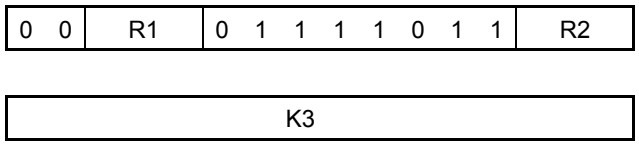
2918 xor - Bitwise Exclusive Or

R1 = R2 ^ R3



Format 1 register

R1 = R2 ^ K3



Format 2 immediate

Instruction Fields

R1 3-bit specifier for destination register
R2 3-bit specifier for source register
R3 3-bit specifier for source register
K3 16-bit literal source

Description

Bitwise-exclusive-or the source operands and write the result to the destination register R1. The address mode is either register (R3) or immediate (K3).

Where the literal source K3 consists of a single 1 bit in a background of 0s, the bitwise-exclusive-or instruction can be transformed into an equivalent bit-change instruction.

Condition Codes

n Set if result is negative, i.e. msb is 1
z Set if result is zero
v Set if (R2 & R3) == R3, or alternatively if (R2 & K3) == K3
c Set if result is in the interval [1:255]

Example

```
xor   r0,r0,r1
xor   r0,r1,0x8001
```

3000 Electrical Characteristics and Power

The normal I/O operating voltage is 3.3V plus or minus 10%. The preliminary maximum clock speed at 3.3V for commercial conditions is 100MHz.

The XInC2 has separate power and ground for the core, I/O and analog peripherals. This minimizes the core switching noise from the output I/O pins and helps to isolate sensitive peripherals from noise as much as possible.

With the exception of the SMU pins (LPPD, RCI, RCO, PORRES, PORDIS) and the crystal oscillator pins (XTI, XTO) all the IO cells are multivoltage IO – capable of operating at a range of 1.8V -> 3.6V. The I/O are characterized below for the operating points of 1.8V, 2.5V and 3.3V.

NOTE! **The IO cells are NOT tolerant to 5V input.**

3001 DC Characteristics for 3.3V I/O

SYMBOL	PARAMETER	CONDITIONS	MIN	TYP	MAX	UNITS
VCC18K	Core Power Supply	Core Area	1.65	1.8	1.89	V
VCCIO	IO Power Supply	Multivoltage Cells	2.97	3.3	3.63	V
Vil	Input Low Voltage	LVTTL			0.8	V
Vih	Input High Voltage	LVTTL	2.0			V
Vt	Switching Threshold	LVTTL		1.6		V
Vt-	Schmitt Trigger Negative Going Threshold Voltage	LVTTL	0.8	1.25		V
Vt+	Schmitt Trigger Positive Going Threshold Voltage	LVTTL		1.75	2.0	V
Vol	Output Low Voltage	|Iol| = 2 ~ 16 mA			0.4	V
Voh	Output High Voltage	|Ioh| = 2 ~ 16 mA	2.4			V
Rpu	Input Pull-Up Resistance	PU=high, PD=low	40	75	190	KΩ
Rpd	Input Pull-Down Resistance	PU=low, PD=high	40	75	190	KΩ
Iin	Input Leakage Current	Vin = VCC3I or 0	-10	±1	10	µA
Ioz	Tri-state Output Leakage Current		-10	±1	10	µA

3002 DC Characteristics for 2.5V I/O

SYMBOL	PARAMETER	CONDITIONS	MIN	TYP	MAX	UNITS
VCC18K	Core Power Supply	Core Area	1.65	1.8	1.89	V
VCCIO	IO Power Supply	Multivoltage Cells	2.25	2.5	2.75	V
Vil	Input Low Voltage	LVTTL			0.69	V
Vih	Input High Voltage	LVTTL	1.46			V
Vt	Switching Threshold	LVTTL		1.15		V
Vt-	Schmitt Trigger Negative Going Threshold Voltage	LVTTL	0.56	0.94		V
Vt+	Schmitt Trigger Positive Going Threshold Voltage	LVTTL		1.4	1.72	V
Vol	Output Low Voltage	|Iol| = 1.1 ~ 8.8 mA			0.4	V
Voh	Output High Voltage	|Ioh| = 1.1 ~ 8.8 mA	1.85			V
Rpu	Input Pull-Up Resistance	PU=high, PD=low	45	110	290	KΩ
Rpd	Input Pull-Down Resistance	PU=low, PD=high	45	110	290	KΩ
Iin	Input Leakage Current	Vin = VCC3I or 0	-10	±1	10	µA
Ioz	Tri-state Output Leakage Current		-10	±1	10	µA

3003 DC Characteristics for 1.8V I/O

SYMBOL	PARAMETER	CONDITIONS	MIN	TYP	MAX	UNITS
VCC18K	Core Power Supply	Core Area	1.62	1.8	1.98	V
VCCIO	IO Power Supply	Multivoltage Cells	1.62	1.8	1.98	V
Vil	Input Low Voltage	LVTTL			0.59	V
Vih	Input High Voltage	LVTTL	1.13			V
Vt	Switching Threshold	LVTTL		0.85		V
Vt-	Schmitt Trigger Negative Going Threshold Voltage	LVTTL	0.49	0.65		V
Vt+	Schmitt Trigger Positive Going Threshold Voltage	LVTTL		1.08	1.39	V
Vol	Output Low Voltage	|Iol| = 0.7 ~ 5.6 mA			0.4	V
Voh	Output High Voltage	|Ioh| = 0.7 ~ 5.6 mA	1.22			V
Rpu	Input Pull-Up Resistance	PU=high, PD=low	80	200	510	KΩ
Rpd	Input Pull-Down Resistance	PU=low, PD=high	80	200	510	KΩ
Iin	Input Leakage Current	Vin = VCC3I or 0	-10	±1	10	μA
Ioz	Tri-state Output Leakage Current		-10	±1	10	μA

3004 Drive Capability of I/O Cells

The DSx bits below are the drive strength configuration for the IO cells in the SCX.

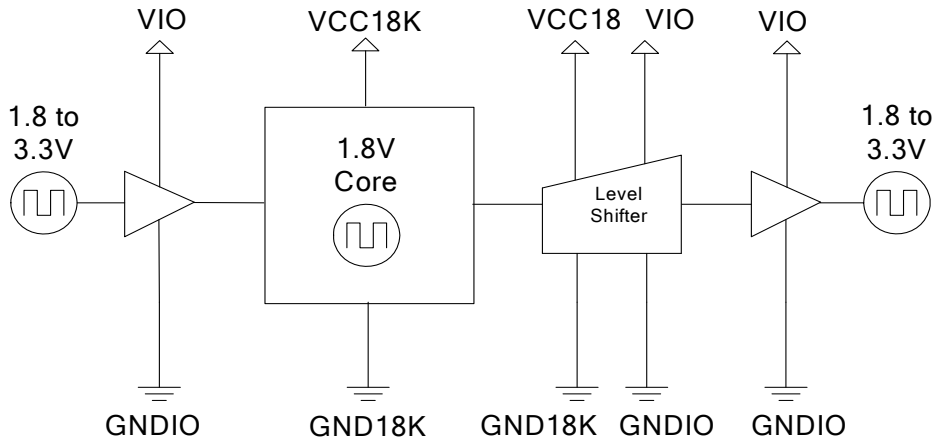
DS2	DS1	DS0	1.8V Drive Strength	2.5V Drive Strength	3.3V Drive Strength	UNITS
0	0	0	0.7	1.1	2.0	mA
0	0	1	1.4	2.2	4.0	mA
0	1	0	2.1	3.3	6.0	mA
0	1	1	2.8	4.4	8.0	mA
1	0	0	3.5	5.5	10.0	mA
1	0	1	4.2	6.6	12.0	mA
1	1	0	4.9	7.7	14.0	mA
1	1	1	5.6	8.8	16.0	mA

3005 ESD and Latchup Specifications

Description	Specification
Human Body Model	> ±2kV
Machine Model	> ±200V
Latch Up	> ±200mA

3006 Power Bussing Structure

The power bussing structure of the multivoltage IO cells are illustrated below:



Where:

- VIO: Power supply for multivoltage input buffers and output pre-drivers.
- VCC18K: Power supply for internal core.
- GNDIO: Ground for input buffers, output buffers, output pre-drivers.
- GNDK: Ground for internal core.

3100 Package Pin out

The XInC2 chip is currently available in a LQFP100 package.

Pin Number	Pin Name	Description
1	VADC	1.8 V ADC Power
2	AN0	Analog multiplexor input 0
3	AN1	Analog multiplexor input 1
4	AN2	Analog multiplexor input 2
5	AN3	Analog multiplexor input 3
6	GNDADC	ADC VSS
7	VREF	ADC Reference Voltage
8	VCC18ADC	ADC digital supply
9	GND18ADC	ADC digital supply
10	SDIO2/PF1	DASI data0/General Purpose IO
11	SDIO1/PF0	DASI data1/General Purpose IO
12	SDIO0/PJ0	DASI data2/General Purpose IO
13	LRCK/PJ1	DASI LR clock/General Purpose IO
14	SCLK/PJ2	DASI Bit clock/General Purpose IO
15	MCLK/PJ3	DASI M clock/General Purpose IO
16	GNDIOK	IO + Core Ground
17	VCCK	1.8 V Core Voltage
18	VCC3IO	3.3 V IO Voltage, Voltage Range 1.8 to 3.6
19	TMRA3/PD7	TimerA0/General Purpose IO
20	TMRA2/PD6	TimerA1/General Purpose IO
21	TMRA1/PD5	TimerA2/General Purpose IO
22	TMRA0/PD4	TimerA3/General Purpose IO
23	PC4	General Purpose IO
24	PC5	General Purpose IO
25	PC6	General Purpose IO
26	PC7	General Purpose IO
27	mbist_ps_sel	reserved
28	mbist_hold_l	reserved
29	NTC0	reserved
30	MISO0/PG0	SPI0 MISO/General Purpose IO
31	MOSI0/PG1	SPI0 MOSI/General Purpose IO
32	SCK0/PG2	SPI0 SCK/General Purpose IO
33	SEN0/PG3	SPI0 SEN/General Purpose IO
34	VCC3IO	3.3 V IO Voltage, Voltage Range 1.8 to 3.6
35	VCCK	1.8 V Core Voltage
36	GNDIOK	IO + Core Ground
37	PB0	General Purpose IO
38	PB1	General Purpose IO
39	PB2	General Purpose IO
40	PC0	General Purpose IO
41	PC1	General Purpose IO
42	PC2	General Purpose IO
43	PC3	General Purpose IO
44	CBO	Clock Buffer Output

45	VCC18XT	Oscillator 1.8V supply
46	XTI	Crystal Oscillator Input
47	XTO	Crystal Oscillator Output
48	GNDXT	Oscillator ground
49	VDC	Varactor Diode Cathode
50	VPLL	1.8 V PLL Supply
51	GNDPLL	PLL VSS
52	NTC1	reserved
53	TMRB	Timer B for Varactor PWM
54	MISO1/PD0	SPI1 MISO/General Purpose IO
55	MOSI1/PD1	SPI1 MOSI/General Purpose IO
56	SCK1/PD2	SPI1 SCK/General Purpose IO
57	SEN1/PD3	SPI1 SEN/General Purpose IO
58	PA7	General Purpose IO
59	PA6	General Purpose IO
60	PA5	General Purpose IO
61	PA4	General Purpose IO
62	PA3	General Purpose IO
63	PA2	General Purpose IO
64	PA1	General Purpose IO
65	PA0	General Purpose IO
66	VCC3IO	3.3 V IO Voltage, Voltage Range 1.8 to 3.6
67	VCCK	1.8 V Core Voltage
68	GNDIOK	IO + Core Ground
69	BBOCLK/PH0	BBU0 data clk/General Purpose IO
70	BB0O/PH1	BBU0 data out/General Purpose IO
71	BB0I/PH2	BBU0 data in/General Purpose IO
72	BB1CLK/PI0	BBU1 data clk/General Purpose IO
73	BB10/PI1	BBU1 data out/General Purpose IO
74	BB1I/PI2	BBU1 data in/General Purpose IO
75	scan_en	reserved
76	scan_test	reserved
77	TEST_IP	reserved
78	TEST_M	reserved
79	VCC3IO	3.3 V IO Voltage, Voltage Range 1.8 to 3.6
80	GNDIO	IO Ground
81	PE0/SCDO4	General Purpose IO
82	PE1/SCDI4	General Purpose IO
83	PE2/SCDO3	General Purpose IO
84	PE3/SCDI3	General Purpose IO
85	PE4	General Purpose IO
86	PE5	General Purpose IO
87	PE6	General Purpose IO
88	PE7	General Purpose IO
89	PORDIS	Prevent the POR reset signal from reseting XInC
90	PORRES	Power on Reset RES Input - Built in pull down

91	/RESET	Active Low Reset
92	RCO	RC Clock Output
93	RCI	RC Clock Input
94	GNDLP	Wakeup Circuit VSS
95	VLP	1.8 V Wakeup Circuit Power
96	LPPD	Power Down Output, Active Low
97	REGEN	Regulator Enable
98	V18	Internal Regulator Output
99	VREG	Internal Regulator Input Voltage
100	GNDREG	Internal Regulator GND

3200 Programming Hints & Common firmware bugs

This section is intended to guide the firmware developer in the process of developing code

Frequently used bits Frequently used bits should be bit 0 or 15. These bits can be tested immediately after an *inp* or *ld* instruction.

Conditional Assembly The use of conditional assembly can greatly reduce development time.

Common Errors The following are some common software bugs:

- Incorrect stack operations
- Real time processing: execution of subroutine is longer than allotted time slot.
- Unprotected resources
- Data space and code space not in separate blocks of RAM/ROM
- Deadlock
- I/O page not set correctly

3300 Firmware Application Examples

3301 Read/Write to General Purpose I/O (GPIO)

General purpose I/O are commonly used to read buttons or keypads, control switches, LEDS, and LCDS or create interfaces like SPI, I2C, or parallel communication.

Each GPIO port contains a configuration register that is used to configure the direction of the pin and set the output data register and a data register used to read the value at the pin or write a value to the output register.

Configuring GPIO ports

The following code configures GPIO port D pins 7-4 as outputs and pins 3-0 as inputs. It also configures pins 7 and 6 to drive high and pins 5 and 4 to drive low.
The upper byte written to GPDcfg set the direction of the pins and the lower byte sets the output register contents.

```
mov     r0,0xF0C0
outp    r0,GPDcfg
```

The following code reads back the configuration.

```
inp     r0,GPDcfg
```

The lower byte of the configuration register is the current contents of the output register, not the value at the pins.

Writing GPIO ports

The following code writes to the output register .

```
mov     r0,0xC0
outp    r0,GPDout
```

Reading GPIO ports

The following code reads the logical value present at the pins.

```
inp     r0,GPDin
```

Read/Modify/Write

Often it is necessary to modify one of the pins on a GPIO port without modifying the state of the other pins. The following code toggles pin 0.

```
inp     r0,GPDin
bix     r0,r0,0
outp    r0,GPDout
```

Firmware Based Peripherals

The XInC2 architecture is well suited to firmware based peripherals. High speed serial interfaces can be implemented with GPIO and a dedicated thread. For time critical serial interfaces use pin0 of a GPIO port for the received data pin because a test of bit 0 can be done by looking at the V-bit immediately after an inp instruction.

3302 Clock Configuration

The following code provides examples of configuring and changing the XInC2 system clock. The system clock source can be configured to be one of three sources. Oscillator1(CLK1), PLL(CLK2), or the slow external RC oscillator(RCCLK)

```
// XInC2 oscillator configuration definitions used by following routines
       XT1ccfg = 0b01001             // FEB, E
       RCXTsel = 1<<10               // RC/XTn select
       XT2Sel  = 1<<11               // XT1/PLL select
       XT1Sel  = 0<<11               // XT1/PLL select
```

Switch to RCCLK The following code switches the system clock source from CLK1 or CLK2 to RCCLK.

```
runRC:
       inp     r1,SCXclkCfg                  // obtain current setup
       and     r1,r1,~(0xF000 | RCXTsel)     // select RCCLK
       outp    r1,SCXclkCfg                  // backing off to RCCLK
runRC10:
       inp     r0,SCXclkCfg                  // wait for switch to occur
       bc      LT0,runRC10
```

Switch to CLK1 The following code switches the system clock source from CLK2 or RCCLK to CLK1.

```
runCLK1:
       inp     r1,SCXclkCfg                  // obtain current setup
       and     r1,r1,~(0xF000 | RCXTsel)     // select RCclk
       outp    r1,SCXclkCfg                  // backing off to RCclk
runCLK110:
       inp     r0,SCXclkCfg                  // wait for switch to occur
       bc      LT0,runCLK110
       mov     r0,XT1Sel | XT1ccfg
       outp    r0,SCXclkCfg                  // fire up XT1
       ior     r0,r0,XTnSel                  // RC/XTn select
       outp    r0,SCXclkCfg                  // switch to XT1
runCLK120:
       inp     r0,SCXclkCfg                  // wait for switch to occur
       bc      GE0,runCLK120
```

Switch to CLK2 The following code switches the system clock source from CLK1 or RCCLK to CLK2.

```
runCLK2:
       inp     r1,SCXclkCfg                  // obtain current setup
       and     r1,r1,~(0xF000 | RCXTsel)     // select RCclk
       outp    r1,SCXclkCfg                  // backing off to RCclk
runCLK210:
       inp     r0,SCXclkCfg                  // wait for switch to occur
       bc      LT0,runCLK210
       mov     r0,XT2Sel
       outp    r0,SCXclkCfg                  // fire up XT2
       ior     r0,r0,XTnSel                  // RC/XTn select
       outp    r0,SCXclkCfg                  // switch to XT2
runCLK220:
       inp     r0,SCXclkCfg                  // make sure we have switched
       bc      GE0,runCLK220
```

3303 I/O cell configuration

The BIOS configures the I/O cells to a default configuration at boot time. For many projects this default configuration may be acceptable. Normally the firmware developer will have a variety of system initialization routines to setup peripherals, and memory. It is at this point that the developer should also initialize the configurable I/O cells. The following code illustrates an I/O setup routine. Subroutine IOCellConfig configures all of the I/Ocells based on the table values.

```
//=============================================================
//              fill in I/O cell configuration table
//              r0-r1   = scratchpad regs
//              r6      = in: return address

IOCellConfig:
        mov     r0,0
IOCellConfig10:
        outp    r0,SCXioCfgP            // I/O config pointer
        ld      r1,r0,IOCbase
        outp    r1,SCXioCfgD            // I/O config data
        add     r0,r0,1
        sub     r1,r0,IOClen
        bc      NE, IOCellConfig10
        jsr     r6,r6                   // return

//The following table should be placed in data space
//=============================================================
//              XInC2 I/O cell configuration table
IOCbase:
//              Bit             Function                Meaning when 1
// ____________________________________________________
//              0               SR                      fast slew
//              1               E2                      +2ma
//              2               E4                      +4ma
//              3               E8                      +8ma
//              4               SMT                     Schmitt Trigger
//              5               PU/PD select            PU
//              6               PU/PD enable            enabled
// ____________________________________________________________
iPA:    0b0000011               // PA           0
iPB0:   0b0001111               // PB0          1
iPB1:   0b0001111               // PB1          2
iPB2:   0b0000011               // PB2          3
iPC0:   0b0000011               // PC0          4
iPC1:   0b0000011               // PC1          5
iPC2:   0b0000011               // PC2          6
iPC3:   0b0000011               // PC3          7
iPC4:   0b0000011               // PC4          8
iPC5:   0b0000011               // PC5          9
iPC6:   0b0000011               // PC6          10
iPC7:   0b0000011               // PC7          11
iPD0:   0b0000011               // PD0  MISO1   12
iPD1:   0b0000011               // PD1  MOSI1   13
iPD2:   0b0000011               // PD2  SCK1    14
iPD3:   0b0000011               // PD3  SEN0    15
iPD4:   0b0000011               // PD4  TMRA0   16
iPD5:   0b0000011               // PD5  TMRA1   17
iPD6:   0b0000011               // PD6  TMRA2   18
iPD7:   0b0000011               // PD7  TMRA3   19
iPE:    0b0000011               // PE           20
iPF:    0b0000011               // PF           21
iPG0:   0b0011111               // PG0  MISO0   22
iPG1:   0b0001111               // PG1  MOSI0   23
iPG2:   0b0001111               // PG2  SCK0    24
iPG3:   0b0000011               // PG3  SEN0    25
iPH:    0b0000011               // PH   BBU0    26
iPI:    0b0000011               // PI   BBU1    27
iPJ:    0b0000011               // PJ   DASI    28
iTMRB:  0b0000011               //      TMRB1   29
        IOClen  = @ - IOCbase
```

3304 SIMD

Certain algorithms can take advantage of the SIMD processing capabilities of XInC2. To enable SIMD processing the XInC2 thread processors must be setup such that they operate in lock step. The routine runSIMD shown below configures all 8 threads to run SIMD. The runSIMD routine takes a pointer to a SIMD algorithm as an input parameter.

runSIMD

```
//=============================================================
//      RunSIMD may be called from T1 only
//      r6      = in: return address
//      r5      = in: SIMD entry point, not preserved
//      r0-r1   = scratchpad registers
RunSIMD:
        mov     r0,~(1<<1)      // stop all threads except 1
        outp    r0,SCUstop
        mov     r1,RunSIMD10
        mov     r0,5            // thread 0
        outp    r0,SCUpntr
        outp    r5,SCUreg
        outp    r1,SCUpc
        add     r0,r0,2<<3      // thread 2
        outp    r0,SCUpntr
        outp    r5,SCUreg
        outp    r1,SCUpc
        add     r0,r0,1<<3      // thread 3
        outp    r0,SCUpntr
        outp    r5,SCUreg
        outp    r1,SCUpc
        add     r0,r0,1<<3      // thread 4
        outp    r0,SCUpntr
        outp    r5,SCUreg
        outp    r1,SCUpc
        add     r0,r0,1<<3      // thread 5
        outp    r0,SCUpntr
        outp    r5,SCUreg
        outp    r1,SCUpc
        add     r0,r0,1<<3      // thread 6
        outp    r0,SCUpntr
        outp    r5,SCUreg
        outp    r1,SCUpc
        add     r0,r0,1<<3      // thread 7
        outp    r0,SCUpntr
        outp    r5,SCUreg
        outp    r1,SCUpc
        mov     r0,0
        outp    r0,SCUstop
RunSIMD10:
        jsr     r5,r5           // jump to SIMD routine
        thrd    r0
        sub     r0,r0,1
        bc      NE,RunSIMD20    // check for Thread1
        jsr     r6,r6           // return
RunSIMD20:
        mov     r0,~(1<<1)      // stop all threads except 1
        outp    r0,SCUstop
        bra     @               // catch for runSIMD
```

RAM Initialization The following code can be used to initialize a block of RAM. Often a block a memory needs to be cleared or set to a know state. The initializeation is done by passing the pointer to InitializeRamSIMD to the RunSIMD routine.

```
        mov     r5,InitializeRamSIMD
        jsr     r6,RunSIMD              // SIMD clear


//============================================================
//      Subroutine: InitializeRamSIMD
//      This routine is used to initialize RAM in SIMD(8 thread) mode.
//      This routine is optimized to clear large blocks of memory.
//      Each pass through the loop clears 128 words.
//              r5      = in: return address
//              r4      = in: end address
//              r3      = in: start address
//              r2      = in: memory initialization value
//              r1      = scratchpad registers
//              r0      = scratchpad registers

InitializeRamSIMD:
        thrd    r0                      // SIMD mode
        add     r0,r0,r3                // compute start address
        mov     r1,0x80                 // clear 0x80 words per loop
InitializeRamSIMD10:
        st      r2,r0,0x00              // high-speed RAM clear
        st      r2,r0,0x08
        st      r2,r0,0x10
        st      r2,r0,0x18
        st      r2,r0,0x20
        st      r2,r0,0x28
        st      r2,r0,0x30
        st      r2,r0,0x38
        st      r2,r0,0x40
        st      r2,r0,0x48
        st      r2,r0,0x50
        st      r2,r0,0x58
        st      r2,r0,0x60
        st      r2,r0,0x68
        st      r2,r0,0x70
        st      r2,r0,0x78
        add     r0,r0,r1                //increment address pointer
        sub     r3,r4,r0                //check for end
        bc      UGE,InitializeRamSIMD10
        jsr     r5,r5                   // return to serial mode
```

3305 Using Semaphores

The following code illustrates examples of using semaphores to share a resource using semaphores to implement a consumer/producer system.

Using a semaphore to share SPI1

The following code shows how to use the XInC2 hardware semaphore mechanism to share the SPI1 serial port between multiple threads.

```
SPI1_semaphore = 0            //define semaphore bit

mov     r1,1<<SPI1_semaphore
outp    r1,SCUdown            //gain access to SPI1

mov     r0,0x12
outp    r0,SPI1tx             //transmit byte
inp     r0,SPI1rx             //receive byte

outp    r1,SCUup              //release access to SPI1
```

Using a semaphore for producer / consumer thread relationship

The following code illustrates an example of a producer/consumer thread relationship. See the section on Critical sections for more information.

The producer thread first captures a 10-bit sample from the ADC and then stores the value in a shared memory location. The consumer thread is then released by issuing a write to the SCUup I/O register. It is assumed that the consumer thread is waiting for a work unit from the producer.

```
        prod_con_sem    = 1             //define semaphore bit

//PRODUCER THREAD CODE

        startADC        = 0b1111000001100000   //setup ADCclock = Sclk /128

producerThread:
        mov     r1,startADC
        outp    r1,ADCcfg               //start conversion
ADCwait:
        inp     r1,ADCdata
        bc      NS,ADCwait              //wait for conversion to finish
        st      r1,ADCsample            //store sample in shared memory
        mov     r0,1<<prod_con_sem
        outp    r0,SCUup                //release work to consumer thread
        bra     producerThread          //get another sample
```

The consumer thread is responsible for converting the 10-bit sample to an 8 bit value and then transmitting the byte through the SPI1 serial port.

The consumer thread begins by configuring the SPI1 port and issuing a initial write to the SCUdown I/O register. Another write to the SCUdown register results in the consumer thread stalling until the producer thread issues the write to the SCUup register. One the consumer thread is released it continues its processing of the data sample and then loops waiting for another work unit.

```
//CONSUMER THREAD CODE

consumerThread:
        mov     r0,0b00000111  //setup SPI1 for Sclk/4,master,cpha,cpol=0
        outp    r0,SP1cfg
        mov     r0,1<<prod_con_sem
        outp    r0,SCUdown              //setup consumer state

consumerWait:
        outp    r0,SCUdown              //wait for producer to release work
        ld      r1,ADCsample            //get 10-bit ADCsample
        rol     r1,r1,-2                //convert to 8-bit
        and     r1,r1,0x00FF
        outp    r1,SPI1tx               //transmit 8-bit sample
        inp     r1,SPI1rx               //wait for transmit to finish
        bra     consumerWait


//SHARED DATA SPACE

ADCsample: @ = @ + 1;                   //reserve 1 word
```

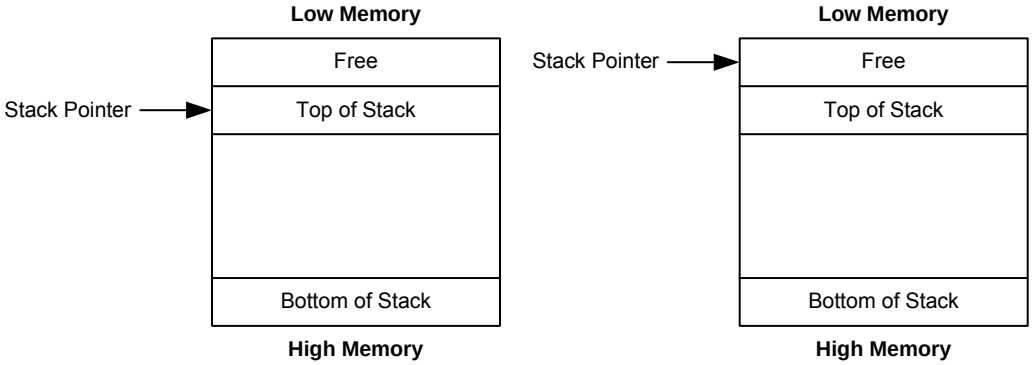
Note! This setup requires that the consumer thread completes its processing before the producer releases another work unit.

3306 User Stacks

User stacks can be designed to store data from low-to-high memory or from high-to-low memory.

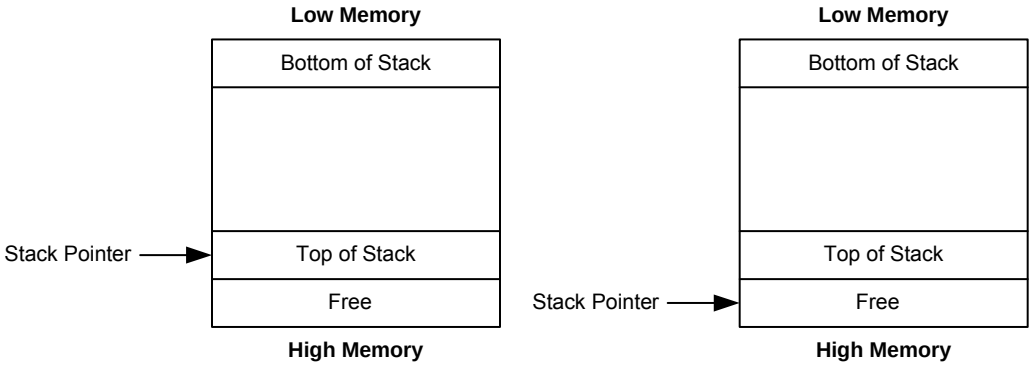
High-to-Low Memory Stack

The following diagrams illustrate two implementations of stack growth from high to low memory.
In the diagram on the left, the stack Pointer points to the top of the stack. In the diagram on the right, the stack Pointer points to the next free memory location on the stack.



Low-to-High Memory Stack

The following diagrams illustrate two implementations of stack growth from low to high memory.
In the diagram on the left, the stack Pointer points to the top of the stack. In the diagram on the right, the stack Pointer points to the next free memory location on the stack.



Example: The following code illustrates setting up a low-to-high memory stack with the Stack Pointer(SP) pointing to the next free location on the stack.
All 8 threads are setup to have their own independent stack.

The following code example uses the SCU peripheral to configure r7 to be the stack pointer(SP) for all threads: This code must be called from Thread 0 and assumes all other threads are stopped.

```
mov     r7,T0_SP        //set stack pointer for T0

mov     r0,T1_SP
mov     r2,1<<3 | 7     //set pointer to Thread1, Register7
outp    r2,SCUpntr
outp    r0,SCUreg       //set r7=stack pointer

mov     r0,T2_SP
mov     r2,2<<3 | 7     //set pointer to Thread2, Register7
outp    r2,SCUpntr
outp    r0,SCUreg       //set r7=stack pointer

mov     r0,T3_SP
mov     r2,3<<3 | 7     //set pointer to Thread3, Register7
outp    r2,SCUpntr
outp    r0,SCUreg       //set r7=stack pointer

mov     r0,T4_SP
mov     r2,4<<3 | 7     //set pointer to Thread4, Register7
outp    r2,SCUpntr
outp    r0,SCUreg       //set r7=stack pointer

mov     r0,T5_SP
mov     r2,5<<3 | 7     //set pointer to Thread5, Register7
outp    r2,SCUpntr
outp    r0,SCUreg       //set r7=stack pointer

mov     r0,T6_SP
mov     r2,6<<3 | 7     //set pointer to Thread6, Register7
outp    r2,SCUpntr
outp    r0,SCUreg       //set r7=stack pointer

mov     r0,T7_SP
mov     r2,7<<3 | 7     //set pointer to Thread7, Register7
outp    r2,SCUpntr
outp    r0,SCUreg       //set r7=stack pointer
```

//The following must be placed in data space

```
StackSize = 32          //reserve 32 words for each stack

T0_SP:  @ = @ + StackSize
T1_SP:  @ = @ + StackSize
T2_SP:  @ = @ + StackSize
T3_SP:  @ = @ + StackSize
T4_SP:  @ = @ + StackSize
T5_SP:  @ = @ + StackSize
T6_SP:  @ = @ + StackSize
T7_SP:  @ = @ + StackSize
```

The following code shows an example of using the stack:

Pushing data to stack:

```
st      r0,sp,0         //push r0 to stack
add     sp,sp,1         //increment stack pointer
```

Popping data from stack:

```
sub     sp,sp,1         //decrement stack pointer
ld      r0,sp,0         //pop r0 from stack
```

3400 XInC2 Development Tools

XInC2 is programmed using XInC2 assembly language. The assembler is know as XInA (pronounced “zina”).

Firmware development can be done using the XInC2 Development Environment (XDE). Command line versions of XInA are also available for Windows, Linux, and Mac systems.

3500 XInC2 Assembler User Guide

The XInC2 assembler known as XInA is used to translate assembly code into machine code.

XInA sources files include instruction mnemonics, labels, preprocessor directives, and comments.

An input line has one of the following forms:

- Directive [operands] [comment]
- [label] mnemonic [operands] []comment]
- comment
- white space

XInA is case sensitive.
There is not limit to line length.
Source files are processed until and end of file is encountered.
There are no restrictions on column placement.

3501 Label

Every input line can be preceded by a label. A label is an alphanumeric string terminated by a colon. The first character of the string can not be a numeric. A label can be used with the branch and jump instructions and as variable names. There is no limit on the length of labels.

3502 Comments

A comment has the following form:

// [text]

The text between the // and the end of line is ignored by XInA.

3503 Instruction mnemonics

The following mnemonics are supported:

Mnemonic	Description
bc	conditional branch
bra	unconditional branch
jsr	jump to subroutine
thrd	get thread number
ld	load from RAM
st	store to RAM
inp	read input port
outp	write output port
mov	move immediate
add	2's complement add
sub	2's complement subtract
rol	bitwise rotate left
and	bitwise and
ior	bitwise inclusive or
xor	bitwise exclusive or
bic	bit clear
bis	bit set
bix	bit change

3504 Preprocessor Directives

The Preprocessor performs the following functions:

- Removes comments
- Includes source code from other files
- Allows definition of constants and substitutes occurrences of defined constants with given text.
- Conditionally includes or ignores selected blocks of program text.
- Set the location counter

The following rules for using Preprocessor Directives should be followed:

- Only white spaces may precede the directive.(@ directive is an exception)
- Only end-of-line, white space, and comments may trail the directive.
- The same identifier name for preprocessor and assembler code is not permitted.

The following directives are supported:

- #include: insert text contained in another file
- #define: define a constant
- #undef: remove a constant
- #ifdef: either include or ignore or ignore the following lines in the program, depending on whether a name has a definition or not.
- #ifndef: similar to #ifdef, but with a reversed test.
- #if: either include or ignore the following lines depending on whether an expression evaluates to 0 or 1.
- #endif: close a #if, #ifdef or #ifndef construct.
- #else: terminate the text following #if, #ifdef or #ifndef construct and introduce a new block of text to be included if the test failed.
- @: sets the location counter to an absolute value

#include The #include directive is used to tell the assembler to start reading from text contained in another file. The assembler then assembles the specified file until end of file is encounterd. The following form is used:

```
#include “MyInclude.asm”
```

Full or relative path names may be specified. For example:

```
#include “..\includeFiles\MyInclude.asm”
```

Included files are shown fully expanded in the listing.

Files that are included by an #include directive may themselves contain #include directives resulting in nested includes.

#define The #define directive may be used to define a constant. All of the text to the right of the constant name is substituted for every occurrence of the constant name. For example:

```
#define PacketLength 10
mov     r0,PacketLength
```

results in the following transformation:

```
mov     r0,10
```

Constants appearing within a comment, a quoted string are not substituted.

#undef The #undef directive is used to undefine a constant. For example:

```
#undef   PacketLength
```

#ifdef The #ifdef directive instructs the preprocessor to process the following block of code only if an identifier has a definition made with a #define directive. If there has been no definition, all lines up to the matching #endif directive are ignored. For example:

```
#ifdef Debug
        inp     r0,GPDin
        bix     r0,r0,0
        outp    r0,GPDout
#endif
```

It is not necessary that the identifier by assigned a value. It is sufficient for the identifier to appear in a define directive. For example:

```
#define Debug
```

#ifndef The #ifndef directive is the reversed conditional test as performed by #ifdef directive.

#if The #if directive is used when an expression must be evaluated in order to determine conditional inclusion. For example:

```
#if Version == 3
…
…
#endif
```

#else The #else directive may be used with the #ifdef, #ifndef or #if directives to indicate code to be included if the test fails.

#ifdef, #ifndef, and #if directives may be nested.

@ The @ directive is used to set the location counter to an absolute value. This directive can be used to set the location of a code segment or a data segment. It can also be used to reserve data words to a variable.

The following syntax is used:

```
@ = [constant]
or
@ = [expression]
```

For example:

```
@ = 0xC800      //set the location for a code segment
mov     r0,0
add     r0,r0,5


@ = 0xE000      //set the location for a data segment
Table:  0x1234
        0x5678
```

```
0x9ABC
0xDEF0

TempVariable:@ = @ + 1  //reserve 1 word to variable
```

The @ directive may also be used with the unconditional branch instruction. The following two instructions are equivalent.

```
Label1:  bra     @       //branch to thyself

Label1:  bra     Label1  //branch to thyself
```

3505 Expressions

Expressions can consist of operands and operators. All expressions are internally 16 bits.

Operands The following operands are valid:

- Labels
- Constants defined by #define directive
- Integer constants in the following formats:
  a. Decimal (default): 15000
  b. Hexadecimal: 0x89AB
  c. Binary: 0b1101000010100001

Operators Many operators are supported: The following operators are listed in order of precedence. Expressions may include parentheses.

- Logical Not !
- Bitwise Not ~
- Unary Minus -
- Multiplication *
- Division /
- Addition +
- Subtraction -
- Shift Left <<
- Shift Right >>
- Less Than <
- Less or Equal <=
- Greater than >
- Greater or Equal >=
- Equal ==
- Not Equal !=
- Bitwise And &
- Bitwise Xor ^
- Bitwise Or |
- Logical And &&
- Logical Or ||

3600 Revision History

The revision history of the XInC2 users guide is:

Version 1.0 Released: Nov 16,2005
Initial release

Version 1.1 Released: March 10,2006
Modified:

- Improved description for ADC and fixed small errors throughout

Version 1.3 Released: September 26,2007
Modified:

- Changed TMRAcount register description to be read only
- Changed TMRBcount register description to be read only
- Changed VPU mask registers description to be read only
- Significant addition to VPU section
- Various other minor typos