User’s Manual

IP2022 Internet Processor™

Revision History

Revision	Release Date	Summary of Changes
1.0	January 22, 2001	Original issue.
1.1	April 13, 2001	New section for the LFSR peripheral.
1.2	May 27, 2001	New section for external memory interface. New **`ireadi`** and **`iwritei`** instructions

Ubicom, Inc.
1330 Charleston Road
Mountain View, CA 94043
tel 650 210 1500
fax 650 210 8715
www.ubicom.com

Table of Contents

List of Figures

Tables

Preface

This manual describes the architecture and instruction set of the IP2022 Internet Processor. Much of the information in this manual overlaps with the material in the data sheet, however this document presents a more detailed description of the instruction set architecture for the benefit of programmers. Refer to the data sheet for the electrical specifications, package mechanical drawing, and ordering information.

Related Documentation

- *IP2022 Data Sheet*, available from Ubicom.
- *IP2022 Quick Reference Guide*, available from Ubicom.
- *IP2022 User's Primer*, available from Ubicom.

1.0 Overview

The Ubicom IP2022 Internet Processor™ combines support for communication physical layer, Internet protocol stack, device-specific application and device-specific peripheral software modules in a single chip, and is reconfigurable over the Internet.

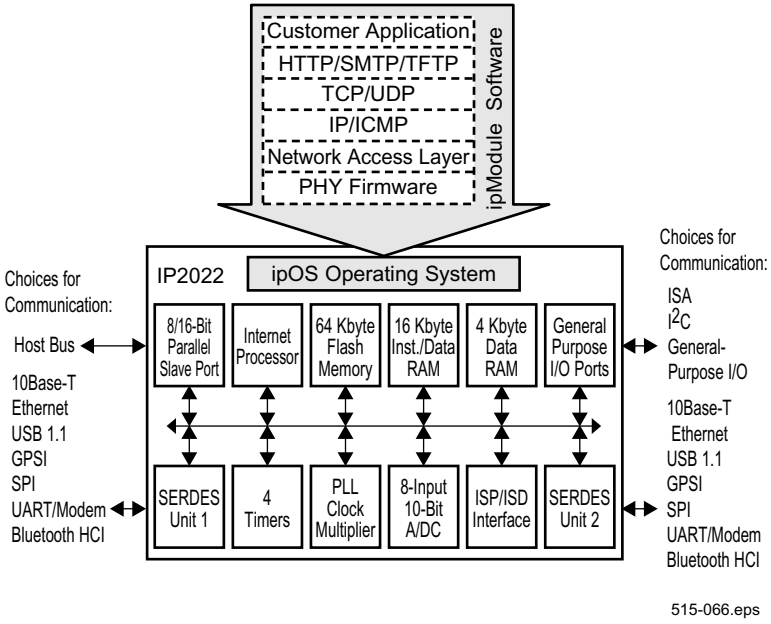


Figure 1-1 IP2022 Block Diagram

It can be programmed, and reprogrammed, using pre-built software modules and configuration tools to create true single-chip solutions for a wide range of device-to-device and device-to-human communication applications. Fabricated in an advanced 0.25-micron process, its RISC-based deterministic architecture provides high-speed computation, flexible I/O control, efficient data manipulation, in-system programming, and in-system debugging.

Two hardware serializer/deserializer (SERDES) units give the IP2022 the ability to directly connect to a variety of common network interfaces. This function provides the ability to implement on-chip 10Base-T Ethernet (MAC and PHY), USB, and a variety of other fast serial protocols. The inclusion of two SERDES units facilitate translation from one format to another, allowing the IP2022 to be used as a protocol converter. The 100 MHz operating frequency, with most instructions executing in a single cycle, delivers the throughput needed for emerging network connectivity applications, and a flash-based program memory allows both in-system and on-the-fly reprogramming. The IP2022 implements peripheral, communications and control functions as software modules (ipModule™ software), replacing traditional hardware for maximum system design flexibility. This approach allows rapid, inexpensive product design and, when needed, quick and easy reconfiguration to accommodate changes in market needs or industry standards.

On-chip dedicated hardware also includes a PLL, an 8-channel 10-bit ADC, general-purpose timers, single-cycle multiplier,

analog comparator, LFSR units, external memory interface, brown-out power voltage detector, watchdog timer, low-power support, multi-source wakeup capability, user-selectable clock modes, high-current outputs, and 52 general-purpose I/O pins.

A TCP/IP network protocol stack is available, and a variety of additional software that is necessary to form a complete end-to-end connectivity solution is being developed. Tools for developing with and using the IP2022, including the complete Red Hat GNUPro tools, are available from leading suppliers.

1.1 Key Features

- **CPU Features**
- RISC engine core with DC to 100 MHz operation
- 10 ns instruction cycle
- Compact 16-bit fixed-length instructions
- Single-cycle instruction execution on most instructions (3 cycles for jumps and calls)
- 16-level hardware stack for high-performance subroutine linkage
- 8 × 8 signed/unsigned single-cycle multiply
- Pointers and stack operation optimized for C compiler
- Uniform, linear address space (no register banks)

On-Chip Memory

- 64 Kbyte (32K × 16) program flash memory
- 16 Kbyte (8K × 16) program/data RAM
- 4 Kbyte linear-addressed data RAM
- Self-programming with built-in charge pump: instructions to read, write, and erase flash memory
- Ability to address up to 128K bytes of external memory

Fast and Deterministic Program Execution and Interrupt Response

- Predictable execution rate for real-time applications
- Fast and deterministic 3-cycle interrupt response
- 30 ns internal interrupt response at 100 MHz including context save
- Hardware save/restore of register context (PC, W, STATUS, MULH, SPDREG, IPH, IPL, DPH, DPL, SPH, SPL, ADDRSEL, DATAH, DATAL)

Multiple Networking Protocols and Physical Layer Support Hardware

- Two full-duplex serializer/deserializer (SERDES) channels for 10Base-T (MAC/PHY), USB, and other fast serial protocols
  - Embedded connectivity nodes
  - Two channels for protocol bridging
  - 10Base-T, GPSI, SPI, UART, USB protocols
  - Squelch function for 10Base-T Ethernet

- Four hardware LFSR units
  - CRC generation/checking
  - Data whitening
  - Encryption

General-Purpose Hardware Peripherals

- Two 16-bit timers with 8-bit prescalers supporting:
  - Timer mode
  - PWM mode
  - Capture/Compare mode
- Parallel host interface, 8/16-bit selectable for use as a communications coprocessor
- One 8-bit timer with programmable 8-bit prescaler
- One 8-bit real-time clock/counter with programmable 15-bit prescaler and 32 kHz crystal input
- Watchdog timer with prescaler
- On-chip PLL clock multiplier with pre- and post-divider
  - 100 MHz on-chip clock from 2 MHz external crystal
- 10-bit, 8-channel ADC with 1/2 LSB accuracy
- Analog comparator with hysteresis enable/disable
- Brown-out minimum supply voltage detector
- External interrupt inputs on 8 pins (Port B)

Sophisticated Power and Frequency/Clock Management Support

- Operating voltage of 2.3V to 2.7V
- Switching the system clock frequencies between different clock sources
- Changing the core clock using a selectable divider
- Shutting down the PLL and/or the OSC input
- Dynamic CPU speed control with **speed** instruction
- Power-On-Reset (POR) logic

Flexible I/O

- 52 I/O Pins
- 2.3V to 3.3V symmetric CMOS output drive
- 5V-tolerant inputs
- Port A pins capable of sourcing/sinking 24 mA
- Optional I/O synchronization to CPU core clock

Programming and Debugging Support

- Updateable application program
  - Run-time self programming
- On-chip in-system programming interface
- On-chip in-system debugging support interface
- Debugging at full IP2022 operating speed
- Programming at device supply voltage level
- Real-time emulation, program debugging, and integrated software development environment offered by leading third-party tool vendors

Pre-Built Software Modules

- Selection of physical interfaces:
  - 10Base-T Ethernet
  - Bluetooth baseband (1 Mbps bit bang)
  - GPSI
  - HomePlug
  - HomePNA
  - I^2C
  - SPI
  - UART
  - USB
  - Parallel host interface
- Complete TCP/IP stack implementation

Software Support

- Unity Integrated Development Environment (IDE)
  - Editor
  - Project manager
  - Graphical User Interface (GUI) to GNU debugger
  - Device programmer
  - ipModule configuration tool
- Nohau in-circuit debugger
  - Seehau interface
  - USB-based debug hardware
  - Assembler
- Library of off-the-shelf ipModule software (Ethernet, serial interfaces, USB interface, etc.)

- Connectivity kits for Internet and communication-intensive applications

1.2 Architecture

1.2.1 CPU

The IP2022 implements an enhanced Harvard architecture (i.e. separate instruction and data memories) with independent address and data buses. The 16-bit program memory and 8-bit dual-port data memory allow instruction fetch and data operations to occur in parallel. The advantage of this architecture is that instruction fetch and memory transfers can be overlapped by a multistage pipeline, so that the next instruction can be fetched from program memory while the current instruction is executed with data from the data memory.

Ubicom has developed a revolutionary RISC-based architecture that is deterministic, jitter free, and completely reprogrammable.

The IP2022 implements a four-stage pipeline (fetch, decode, execute, and write back). At the maximum operating frequency of 100 MHz, instructions are executed at the rate of one per 10 ns clock cycle.

1.2.2 Serializer/Deserializer Units (SERDES)

One of the key elements in optimizing the IP2022 for device-to-device and device-to-human communication is the inclusion of two

on-chip serializer/deserializer (SERDES) units. These units support popular communication protocols such as 10Base-T Ethernet, GPSI, SPI, UART, and USB, allowing the IP2022 to be used as a protocol converter in bridge and gateway applications.

By performing data serialization and deserialization in hardware, the CPU bandwidth needed to support serial communications is greatly reduced, especially at high baud rates. Providing two units allows easy implementation of protocol conversion or bridging functions, such as a USB-to-Ethernet bridge.

1.2.3 Low-Power Support

Particular attention has been paid to minimizing power consumption. For example, an on-chip PLL allows use of a lower-frequency external source (e.g., an inexpensive 2 MHz crystal can be used to produce a 100 MHz on-chip clock), which reduces both power consumption and EMI. In addition, software can change the execution speed of the CPU to reduce power consumption, and a mechanism is provided for automatically changing the speed on entry and return from an interrupt service routine. The **speed** instruction specifies power-saving modes that include a clock divisor between 1 and 128. This divisor only affects the clock to the CPU core, not the timers. The **speed** instruction also specifies the clock source (OSC1 clock, RTCLK oscillator, or PLL clock multiplier), and whether to disable the OSC1 clock oscillator or the PLL. The **speed** instruction executes using the current clock divisor.

1.2.4 Memory

The IP2022 CPU executes from a 32K × 16 flash program memory and an 8K × 16 RAM program/data memory. In addition, the ability to write into the program flash memory allows flexible non-volatile data storage. An interface is available for up to 128K bytes of external memory. The maximum execution rate is 30 MIPS from flash memory and 100 MIPS from RAM. Speed-critical routines can be copied from the flash memory to the RAM for faster execution. The IP2022 has a mechanism for in-system programming of its flash and RAM program memories through a four-wire SPI interface, and software has the ability to reprogram the program memories at run time. This allows the functionality of a device to be changed in the field over the Internet.

1.2.5 Instruction Set

The IP2022 instruction set, using 16-bit words, implements a rich set of arithmetic and logical operations, including signed and unsigned 8-bit × 8-bit integer multiply with a 16-bit product.

1.2.6 The ipModule Concept

The ipModule concept enables the "software system-on-a-chip" approach. An ipModule is a software implementation of an interface, protocol, or other function that replaces traditional hardware. This takes advantage of the Ubicom architecture's high performance and deterministic nature to produce the same results as hardware, but with much greater system design flexibility.

Having functionality implemented as pre-built software modules allows the IP2022 to be programmed and reprogrammed at any time in the design and manufacturing cycle, and even in the field over the Internet.

The speed and flexibility of the Ubicom architecture, together with the availability of Internet connectivity software modules, simultaneously address a wide range of engineering and product development concerns. They decrease the product development cycle dramatically, shortening time to production to as little as a few weeks.

Ubicom's timesaving ipModule software gives system designers a choice of ready-made solutions and a head start on developing their own peripherals. With ipModule software handling established functions, design engineers can concentrate on adding value to other areas of the application.

Overall, the ipModule concept provides such benefits as simpler hardware architecture, reduced component count, fast time to market, increased flexibility in design, application customization, and overall system cost reduction.

Some examples of ipModule software are:

- Ethernet and USB network interfaces
- Communication interfaces such as GPSI, I^2C™, Microwire™, SPI, and UART
- Internet connectivity protocols, such as UDP, TCP/IP, ARP, DHCP, HTTP, SMTP, and POP3

1.2.7 Programming and Debugging Support

The IP2022 is supported by leading third-party tool vendors. On-chip in-system debug capabilities allow these tools to provide an integrated software development environment that includes editor, assembler, debugger, simulator, and programmer tools. For example, the complete Red Hat GNUPro tools, including C compiler, assembler, linker, utilities and GNU debugger, supports the IP2022. Likewise, the Seehau interface, high-end debugger, assembler, and USB debug hardware from Nohau can be used with the IP2022.

In addition, Ubicom offers an integrated graphical development environment which includes an editor, project manager, graphical user interface for the GNU debugger, device programmer, and ipModule configuration tool.

Unobtrusive in-system programming is provided through the ISP interface. There is no need for a bond-out chip for software development. This eliminates concerns about differences in electrical characteristics between a bond-out chip and the actual chip used in the target application. Designers can test and revise code on the same part used in the actual application.

1.2.8 Applications

The IP2022 Internet Processor™ is optimized for network connectivity applications, and is ideally suited for use in the node and bridge/gateway portions of the Internet infrastructure.

Node device applications are those that are commonly associated with the "embedded Internet," such as home appliances, medical devices, vending machines, and remote monitoring and control systems. These nodes are frequently interconnected by local-area networks (LANs). Bridge/gateway devices provide the functions that are required to connect the nodes, and their related LANs, to the Internet, such as protocol conversion, IP address routing, and firewall functions. The IP2022 enables true single-chip device and bridge/gateway connectivity implementations at a consumer price point. The library of ipModule software, including the Internet protocol stack and communication interfaces, allows design engineers to embed Internet connectivity in all of their products at low cost with very fast time-to-market.

2.0 System Architecture

The IP2022 CPU executes from a 32K × 16 flash program memory and an 8K × 16 RAM program memory. The maximum execution rate is 30 MIPS from flash and 100 MIPS from RAM. Speed-critical routines can be copied from the flash memory to the RAM for faster execution. The CPU operates on 8-bit data in 128 special-purpose registers, 128 global registers, and 3840 bytes of data memory. The special-purpose registers hold control and status bits used for CPU control and for interface with hardware peripherals (timers, I/O ports, A/D converter, etc.).

Although the philosophy followed in the design of Ubicom products emphasizes the use of fast RISC CPUs with predictable execution times to emulate peripheral devices in software (called ipModule™ software), there are a few hardware peripherals which are difficult to emulate in software alone (e.g. an A/D converter) or consume an excessive number of instruction cycles when operating at high speed (e.g. data serialization/deserialization). The design of the IP2022 incorporates only those hardware peripherals which can greatly accelerate or extend the reach of the ipModule™ concept.

The hardware peripherals included on-chip are:

- 52 I/O port pins
- Watchdog Timer
- 2 Real-time 8-bit timers
- 2 Multifunction 16-bit timers with compare and capture registers
- 2 Serializer/Deserializer (SERDES) channels
- 10-bit, 8-channel A/D converter
- Analog comparator
- Parallel slave peripheral interface

There is a single interrupt vector which can be reprogrammed by software. On-chip peripherals and up to 8 external inputs can raise interrupts.

There are five sources of reset:

- $\overline{\text{RST}}$ external reset input
- Power-On Reset (POR) logic
- Brown-Out Reset (BOR) logic (detects low DVdd condition)
- Watchdog Timer
- Reset from SPI port programming interface

The reset vector is fixed at word address 0xFFF0.

An on-chip PLL clock multiplier enables high-speed operation (up to 100 MHz) from a slow-speed external clock input, crystal, or ceramic resonator. A CPU clock-throttling mechanism allows fine control over power consumption in modes that do not require maximum speed, such as waiting for an interrupt.

The IP2022 has a mechanism for in-system programming of its flash and RAM program memories through a four-wire SPI interface. This provides easy programming and reprogramming of devices on assembled circuit boards. In addition, the flash memory can be programmed by software at run time, for example to store user-specific data such as phone numbers and to receive software upgrades downloaded over the Internet. The IP2022 also has an on-chip debugging facility which makes the internal operation of the chip visible to third-party debugging tools.

2.1 CPU Registers

Figure 2-1 shows the CPU registers, which consist of seven 8-bit registers, seven 16-bit registers, and one 24-bit register. The 16-bit registers are formed from pairs of 8-bit registers, and the 24-bit register is formed from three 8-bit registers. See Appendix C for a complete list of CPU and peripheral registers.

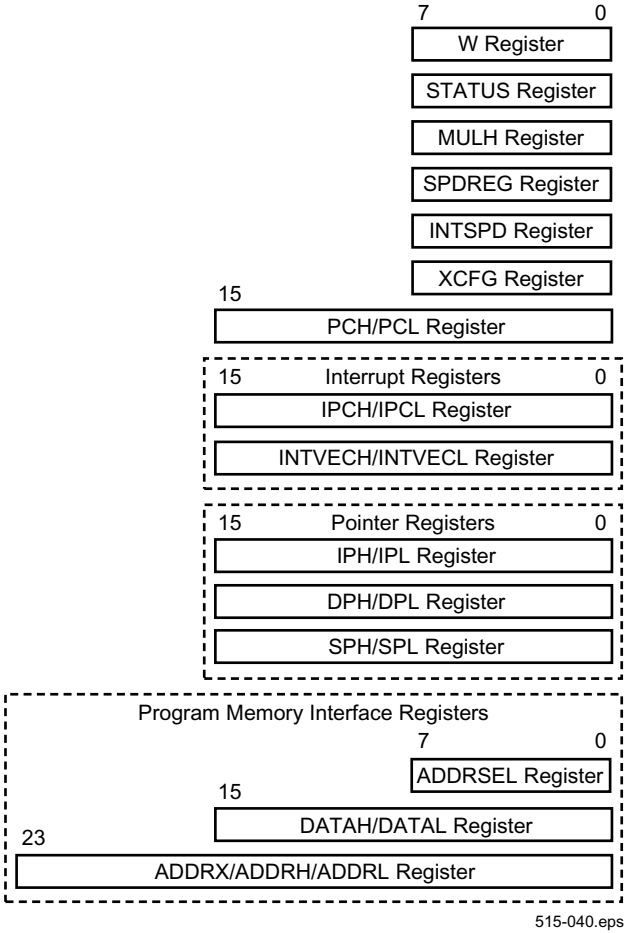


Figure 2-1 CPU Registers

- *W or Working register*—the source or destination for most arithmetic and logical instructions.
- *STATUS register*—condition flags for the results of arithmetic and logical operations, the page bits (used for jumps and subroutine calls), and bits which indicate the cause of the last reset (watchdog timer overflow or brown-out voltage detector). Section 2.1.1 shows the assignment of the bits in the STATUS register.
- *MULH register*—receives the upper 8 bits of the 16-bit product from signed or unsigned multiplication. The lower 8 bits are loaded into the W register.
- *SPDREG register*—holds bits that indicate the CPU speed and clock source settings loaded by the **`speed`** instruction, as shown in Section 2.1.2. For more information about the **`speed`** instruction and the clock throttling mechanism, see Section 2.4
- *INTSPD register*—holds bits that control the CPU speed and clock source during interrupt service routines. It has the same format as the SPDREG register.
- *XCFG register*—holds additional control and status bits, as shown in Section 2.1.3.
- *PCH/PCL register*—16-bit program counter.
- *IPCH/IPCL register*—specifies the return address when a **`reti`** instruction is executed. On returning from the interrupt service routine, an option of the **`reti`** instruction allows software to save the incremented value of the program counter in the INTVECH and INTVECL registers.

- *INTVECH/INTVECL register*—specifies the interrupt vector. It has a default value of 0x0000 following reset.
- *IPH/IPL register*—indirect pointer for indirect addressing. For more information about indirect addressing, see Section 3.1.3.
- *DPH/DPL register* and *SPH/SPL register*—data pointer and stack pointer for indirect-with-offset addressing. For more information about indirect-with-offset addressing, see Section 3.1.4. The SPH and SPL registers are automatically postdecremented when storing to memory with a **push** instruction, and they are automatically preincremented when reading from memory with a **pop** instruction.
- *ADDRSEL register*—holds an index to one of eight 24-bit pointers used to address program memory.
- *ADDRX/ADDRH/ADDRL register*—the current pointer selected by the ADDRSEL register is accessable in the ADDRX (bits 23:16), ADDRH (bits 15:8), and ADDRL (bits 7:0) registers.
- *DATAH/DATAL register*—program memory is always read or written as 16-bit words. On reads, the data from program memory is loaded into the DATAH/DATAL register. On writes, the contents of the DATAH/DATAL register pair are loaded into the program memory.

2.1.1 STATUS Register

7 5	4	3	2	1	0
PA2:0	WD	BO	Z	DC	C

Name	Description
PA2:0	Program memory page select bits. Used to extend the 13-bit address encoded in jump and call instructions. Modified using the **page** instruction.
WD	Watchdog time-out bit. Set at reset, if reset was triggered by Watchdog Timer overflow, otherwise cleared. 0 = Last reset was not caused by a Watchdog Timer overflow. 1 = Last reset was caused by a Watchdog Timer overflow.
BO	Brown-out reset bit. Set at reset, if reset was triggered by brown-out voltage level detection. 0 = Last reset was not caused by the brown-out voltage detector sensing a low-voltage condition. 1 = Last reset was caused by the brown-out voltage detector sensing a low-voltage condition.
Z	Zero bit. Affected by most logical, arithmetic, and data movement instructions. Set if the result was zero, otherwise cleared. 0 = Result of last ALU operation was non-zero. 1 = Result of last ALU operation was zero.

Name	Description
DC	Digit Carry bit. After addition, set if carry from bit 3 occurred, otherwise cleared. After subtraction, cleared if borrow from bit 3 occurred, otherwise set. 0 = Last addition did not generate carry out of bit 3, or last subtraction generated borrow out of bit 3. 1 = Last addition generated carry out of bit 3, or last subtraction did not generate borrow out of bit 3.
C	Carry bit. After addition, set if carry from bit 7 of the result occurred, otherwise cleared. After subtraction, cleared if borrow from bit 7 of the result occurred, otherwise set. After rotate (**`rr`** or **`rl`**) instructions, loaded with the LSB or MSB of the operand, respectively. 0 = Last addition did not generate carry out of bit 7, last subtraction generated borrow out of bit 7, or last rotate loaded a 0. 1 = Last addition generated carry out of bit 7, last subtraction did not generate borrow out of bit 7, or last rotate loaded a 1.

2.1.2 SPDREG Register

7	6	5 4	3 0
$\overline{\text{PLL}}$	$\overline{\text{OSC}}$	CLK1:0	CDIV3:0

Name	Description
$\overline{\text{PLL}}$	Controls PLL clock multiplier operation. If the PLL is not required, power consumption can be reduced by disabling it. 0 = PLL clock multiplier enabled 1 = PLL clock multiplier disabled
$\overline{\text{OSC}}$	Controls OSC oscillator operation. If the oscillator is not required, power consumption can be reduced by disabling it. 0 = OSC oscillator enabled 1 = OSC oscillator disabled
CLK1:0	Selects the system clock source. 00 = PLL clock multiplier 01 = OSC oscillator/external clock on OSC1 input 10 = RTCLK oscillator/external clock on RTCLK1 input 11 = System clock disabled (off)

Name	Description
CDIV3:0	Selects the system clock divisor. 0000 = 1 0001 = 2 0010 = 3 0011 = 4 0100 = 5 0101 = 6 0110 = 8 0111 = 10 1000 = 12 1001 = 16 1010 = 24 1011 = 32 1100 = 48 1101 = 64 1110 = 128 1111 = System clock disabled (off)

2.1.3 XCFG Register

7	6	5	4	3	2	1	0
GIE	$\overline{\text{FWP}}$	RTEOS	$\overline{\text{RTOSC\_EN}}$	INT_EN	Reserved		FBUSY

Name	Description
GIE	Global interrupt enable bit. For more information about interrupt processing, see Section 2.5. 0 = Interrupts disabled 1 = Interrupts enabled
$\overline{\text{FWP}}$	Flash write protect bit. This bit only affects operation of the self-programming instructions, not programming through the ISD/ISP interface. For more information about programming the flash memory, see Section 3.7. 0 = Writes to flash memory disabled 1 = Writes to flash memory enabled
RTEOS	Real-time timer oversampling bit. For more information about the real-time timer, see Section 4.3. 0 = Oversampling disabled 1 = Oversampling enabled
$\overline{\text{RTOSC\_EN}}$	RTCLK oscillator enable bit 0 = RTCLK oscillator is operational 1 = RTCLK oscillator turned off

Name	Description
INT_EN	**`int`** instruction interrupt enable bit 0 = **`int`** instructions only increment the PC, like **`nop`** 1 = **`int`** instructions cause interrupts
FBUSY	Flash memory busy bit (read-only). For more information about programming the flash memory, see Section 3.7. 0 = Flash memory is idle 1 = CPU is fetching an instruction out of flash memory or processing an **`iread`**, **`fwrite`**, or **`ferase`** instruction

2.2 Data Memory

Figure 2-2 is a map of the data memory.

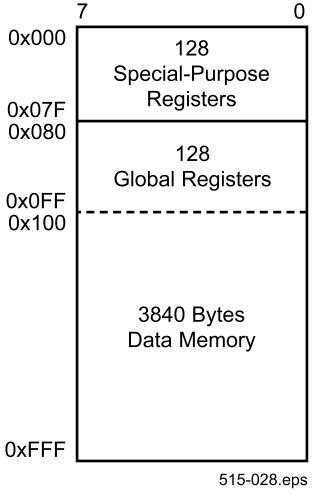


Figure 2-2 Data Memory Map

The special-purpose registers and the first 128 data memory locations (between addresses 0x080 and 0x0FF) can be accessed with a direct addressing mode in which the absolute address of the operand is encoded within the instruction. The remaining 3840 bytes of data memory (between addresses 0x100 and 0xFFF) must be accessed using indirect or indirect-with-offset addressing modes. There is one 16-bit register for the indirect addressing pointer, and two 16-bit registers for indirect-with-offset address pointers. The offset is a 7-bit value encoded within the

instruction. For more information about the addressing modes, see Section 3.1.

2.3 Program Memory

Figure 2-3 is a map of the program memory. A program memory address in the INTVECH/INVECL, IPCH/IPCL, or PCH/PCL registers or on the hardware stack is a word address. *However, the GNU software tools require byte addresses when referring to locations in program memory.* An address loaded in the ADDRX/ADDRH/ADDL register is a byte address.

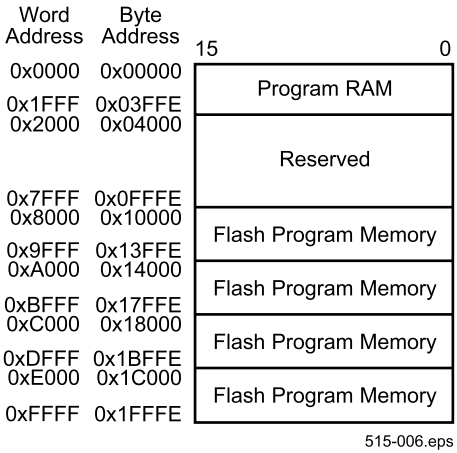


Figure 2-3 Program Memory Map

The program memory is organized as 8K-word pages (16K bytes). Single-instruction jumps and subroutine calls are restricted to be

within the same page. Longer jumps and calls require using a **`page`** instruction to load the upper address bits into the PA2:0 bits of the STATUS register. The **`page`** instruction must immediately precede the jump or call instruction. The PA2:0 bits should not be modified by writing directly to the STATUS register, because this may cause a mismatch between the PA2:0 bits in the STATUS register and the current program counter (see Section 2.3.2). For more information about program flash memory, see Section 3.7.

External memory is not shown in Figure 2-3 because the CPU cannot execute instructions directly out of external memory. For more information about external memory, see Section 4.11.

2.3.1 Loading the Program RAM

Software loads the program RAM from program flash memory using the **`iread`**/**`ireadi`** and **`iwrite`**/**`iwritei`** instructions. The **`iread`** instruction reads the 16-bit word specified by the address held in the ADDRX/ADDRH/ADDRL register. This word can be in program flash memory, program RAM, or external memory. When the **`iread`** instruction is executed, bits 15:8 of the word are loaded into the DATAH register, and bits 7:0 are loaded into the DATAL register. The address is a word-aligned byte address (i.e. an address that is zero in its LSB). The **`ireadi`** instruction is identical to the **`iread`** instruction, except that it also increments the address by 2.

The **`iwrite`** instruction writes the 16-bit word held in the DATAH/DATAL registers to the program RAM location specified by

the address held in the ADDRX/ADDRH/ADDRL register. The **`iwritei`** instruction is identical, except that it also increments the address by 2. For more information about the **`iread`**/**`ireadi`** and **`iwrite`**/**`iwritei`** instructions, see Section 3.7.

2.3.2 Program Counter

The program counter holds the 16-bit address of the instruction to be executed. The lower eight bits of the program counter are held in the PCL register, and the upper eight bits are held in the PCH register. A write to the PCL register will cause a jump to the 16-bit address specified by the PCH and PCL registers. If the PCL register is written as the destination of an **`add`** or **`addc`** instruction and carry occurs, the PCH register is automatically incremented. (This may cause a mismatch between the PA2:0 bits in the STATUS register and the current program counter, therefore it is strongly recommended that direct modification of the PCL register is only used for jumps within a page.) The PCH register is read-only.

The PA2:0 bits in the STATUS register are not used for address generation, except when a jump or subroutine call instruction is executed. However, when an interrupt is taken, the PA2:0 bits are automatically updated with the upper three bits of the interrupt vector. These bits are restored from the STATUS shadow register when the interrupt service routine returns (i.e. executes a **`reti`** instruction).

2.4 Low Power Support

Software can change the execution speed of the CPU to reduce power consumption. A mechanism is provided for automatically changing the speed on entry and return from the interrupt service routine. The **speed** instruction specifies power-saving modes that include a clock divisor between 1 and 128. This divisor only affects the clock to the CPU core, not the timers or ADC (see Figure 2-12). The **speed** instruction also specifies the clock source (OSC clock, RTCLK oscillator, or PLL clock multiplier) and whether to disable the OSC clock oscillator or the PLL. The next two instructions after switching the clock source will be run at the old speed.

The **speed** instruction executes using the current clock divisor. The new clock divisor takes effect with the following instruction, as shown in the following code example.

```
nop                ;assume divisor is 4, so this
                   ;instruction takes 4 cycles
speed   #$06       ;change the divisor to 8,
                   ;instruction takes 4 cycles
nop                ;instruction takes 8 cycles
speed   #$0D       ;change the divisor to 1,
                   ;instruction takes 8 cycles
nop                ;instruction takes 1 cycle
```

Before executing the **speed** instruction, check that the FBUSY bit in the XCFG register is clear and that the FCFG register has appropriate settings for the new clock frequency.

The SPDREG register holds the current settings for the clock divisor, clock source, and disable bits. These settings can be explicitly changed by executing a speed instruction, and they change automatically on interrupts. The SPDREG register is read-only, and its contents may only be changed by executing a **`speed`** instruction, taking an interrupt, or returning from an interrupt. Two consecutive **`speed`** instructions are not allowed. The INTSPD register specifies the settings used during execution of the interrupt service routine. The INTSPD register is both readable and writeable.

On return from interrupts, the **`reti`** instruction includes a bit that specifies whether the pre-interrupt speed is restored or the current speed is maintained.

The actual speed of the CPU is indicated by the SPDREG register unless the specified speed is faster than the flash access time and the program is executing out of flash. When program execution moves from program RAM to program flash memory, the new clock divisor will be the greater (slower) of the clock divisor indicated by the SPDREG register and the clock divisor required to avoid violating the flash memory access time. The SPDREG register does not indicate if the flash clock divisor is being used. The value indicated by the SPDREG will be overridden only if the speed is too fast for the flash memory.

The FCFG register holds bits that specify the minimum number of system clock cycles for each flash memory cycle (see Section 3.7).

2.4.1 Speed Change Delay

The automatic speed changes require a certain amount of delay to take effect:

- *Changing the Clock Divisor*—there is no delay when the clock divisor is changed.
- *Changing the Clock Source*—the delay is up to one cycle of the slower clock. For example, changing between 32 kHz and 100 MHz could require up to 31.25 microseconds.
- *Turning on the OSC Clock Oscillator (clearing the $\overline{OSC}$ bit in the SPDREG register)*—the system clock suspend time is specified in the WUDX2:0 bits in the FUSE0 register.
- *Turning on the PLL Clock Multiplier (clearing the $\overline{PLL}$ bit in the SPRDREG register)*—the system clock suspend time is specified in the WUDP2:0 bits in the FUSE0 register.

If both the OSC oscillator and PLL are re-enabled simultaneously, the delay is controlled by only the WUDX2:0 bits. Bits in the FUSE0 register are flash memory cells which cannot be changed dynamically during program execution (see Section 6.2).

2.4.2 Instruction Timing

All instructions that perform branches take 3 cycles to complete, consisting of 1 cycle to execute and 2 cycles to load the pipeline.

Table 2-1 Branch Timing

Instruction	Execution Time	Pipeline Load Time
`jmp`	1	2
`call`	1	2
`ret`	1	2
`reti`	1	2

In the case of an automatic speed change, the execution time will be with respect to the original speed and the pipeline load time will be with respect to the new speed.

Conditional branching is implemented in the IP2022 by using conditional skip instructions to branch over an unconditional jump instruction. To support conditional branching to other pages, the conditional skip instructions will skip over two instructions if the first instruction is a **`page`** instruction. The **`loadh`** and **`loadl`** instructions also cause an additional instruction to be skipped. When any of these conditions occur, it is called an *extended skip instruction*.

Skip instructions take 1 cycle if they do not skip, or 2 cycles if they skip over one instruction. An extended skip instruction may skip over more than one **`loadh`**, **`loadl`**, or **`page`** instruction,

however this operation is interruptible and does not affect interrupt latency.

The **`iread`** and **`iwrite`** instructions take 4 cycles. The multiply instructions take 1 cycle.

2.5 Interrupt Support

There are three types of interrupt sources:

- *On-Chip Peripherals*–the serializer/deserializer (SERDES) units, real-time timer, timer 0, timer 1, and timer 2 are capable of generating interrupts. The Parallel Slave Peripheral does not generate interrupts on its own, however it requires programming one of the Port B external interrupt inputs to generate interrupts on its behalf.
- *External Interrupts*–the eight pins on Port B can be programmed to generate interrupts on either rising or falling edges (see Section 4.1.1).
- ***`int`*** *Instruction*–the **`int`** instruction can be executed by software to generate an interrupt. The INT_EN bit in the XCFG register must be set to enable the **`int`** instruction to trigger an interrupt. Because the **`reti`** instruction returns to the **`int`** instruction, the INT_EN bit must be cleared in the interrupt service routine (ISR) before returning.

Figure 2-4 shows the system interrupt logic. Each interrupt source has an interrupt enable bit. To be capable of generating an interrupt, the interrupt enable bit and the global interrupt enable (GIE) bit must be set.

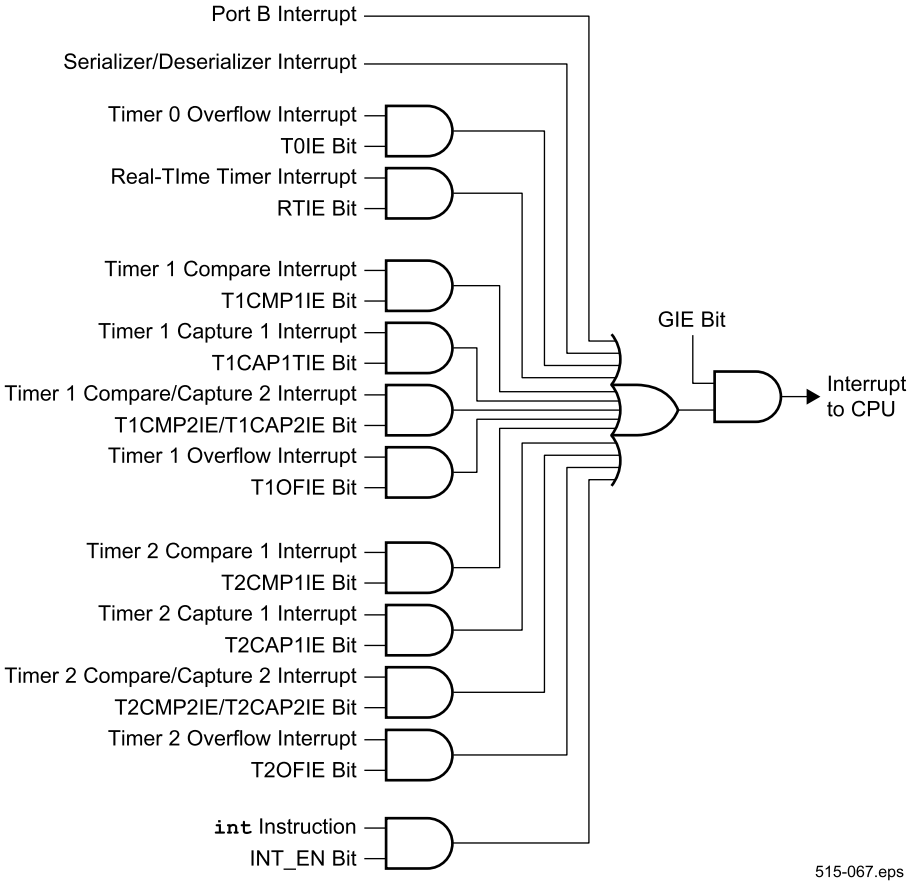


Figure 2-4 System Interrupt Logic

2.5.1 Interrupt Processing

There is one interrupt vector held in the INTVECH and INTVECL registers, which is reprogrammable by software. When an interrupt is taken, the current PC is saved in the IPCH and IPCL registers. On return from interrupt (i.e. execution of the **`reti`** instruction), the PC is restored from the IPCH and IPCL registers. Optionally, the **`reti`** instruction may also copy the incremented PC to the INTVECH and INTVECL registers before returning. This has the effect of loading the INTVECH and INTVECL registers with the address of the next instruction following the **`reti`** instruction. This option can be used to directly implement a state machine, such as a simple round-robin scheduling mechanism for a series of interrupt service routines (ISRs) in consecutive memory locations.

If multiple sources of interrupts have been enabled, the ISR must check the interrupt flags of each source to determine the cause of the interrupt. The ISR must clear the interrupt flag for the source of the interrupt to prevent retriggering of the interrupt on completion of the ISR (i.e. execution of the **`reti`** instruction). Because the interrupt logic adds a 2-cycle delay between clearing an interrupt flag and deasserting the interrupt request to the CPU, the flag must be cleared at least 2 cycles before the **`reti`** instruction is taken.

When an interrupt is taken, the registers shown in Figure 2-5 are copied to a shadow register set. Each shadow register is actually a 2-level push-down stack, so one level of interrupt nesting is supported in hardware. The interrupt processing mechanism is

completely independent of the 16-level call/return stack used for subroutines.

The contents of the DATAH and DATAL registers are pushed to their shadow registers 4 cycles after the interrupt occurs, to protect the result of any pending **`iread`** instruction. Therefore, software should not access the DATAH or DATAL registers during the first instruction of an ISR.

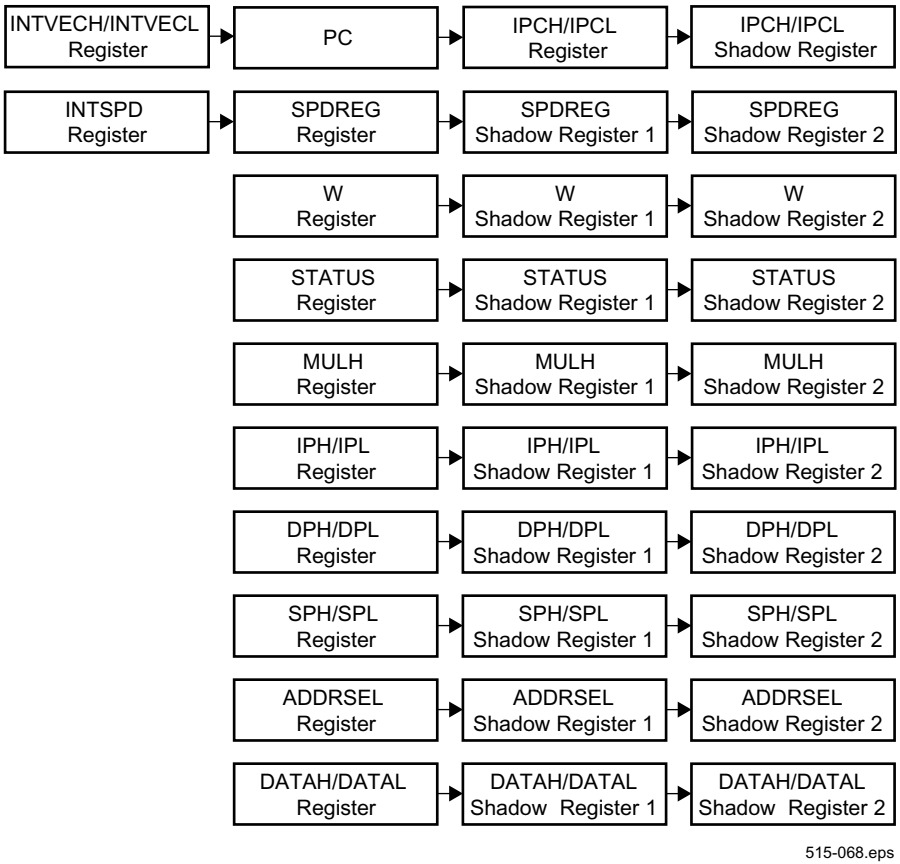


Figure 2-5 Interrupt Processing (On Entry to the ISR)

On return from the ISR, these registers are restored from the shadow registers, as shown in Figure 2-6.

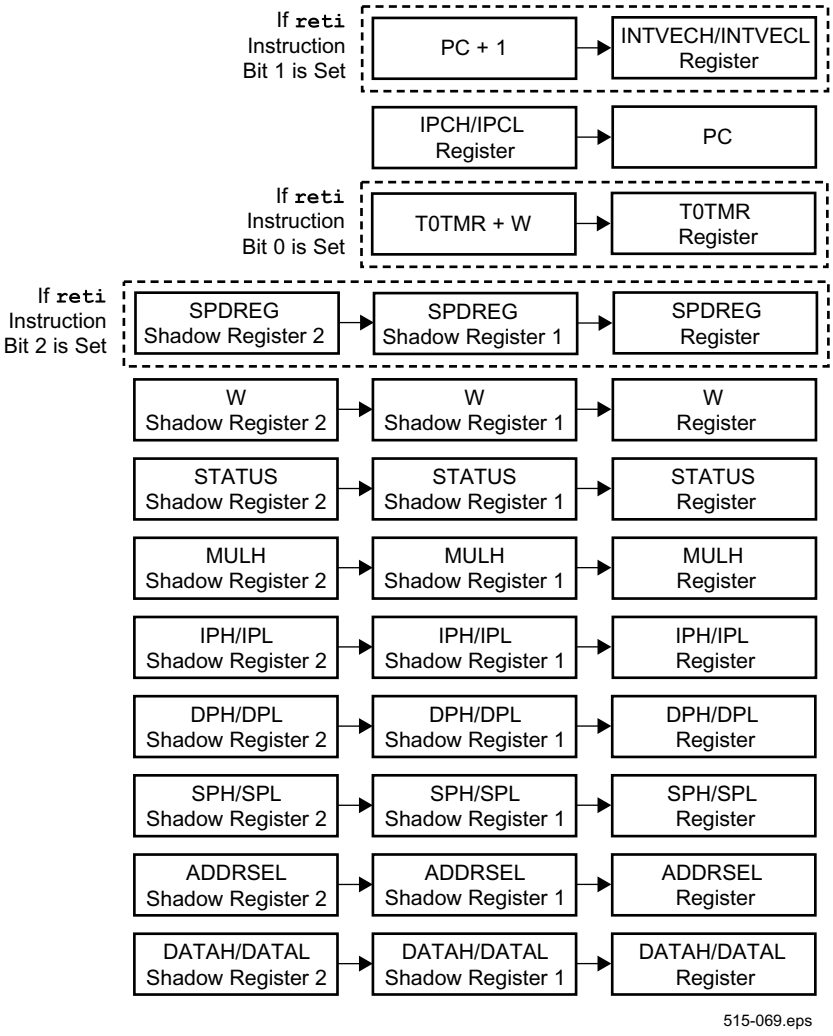


Figure 2-6 Interrupt Processing (On Return from the ISR)

2.5.2 Global Interrupt Enable Bit

The GIE bit serves two purposes:

- Preventing an interrupt in a critical section of mainline code
- Supporting nested interrupts

The GIE bit is automatically cleared when an interrupt occurs, to disable interrupts while the ISR is executing. The GIE bit is automatically set by the **`reti`** instruction to re-enable interrupts when the ISR returns.

Table 2-2 GIE Bit Handling

Event	Effect
Enter ISR (interrupt)	GIE bit is cleared
Exit ISR (**`reti`** instruction)	GIE bit is set
`setb xcfg,7` instruction (inside ISR)	Enable interrupts for nested interrupt support
`clrb xcfg,7` instruction (inside ISR)	Nothing, the GIE bit is already clear
`setb xcfg,7` instruction (mainline code)	Enable interrupts
`clrb xcfg,7` instruction (mainline code)	Disable interrupts

To re-enable interrupts during ISR execution, the ISR code must first clear the source of the first interrupt. It may also be desirable

to disable specific interrupts before setting the GIE bit to provide interrupt prioritization. Caution must be taken not to exceed the interrupt shadow register stack depth of 2.

Clearing the GIE bit in the ISR cannot be used to globally disable interrupts so that they remain disabled when the ISR returns, because the **`reti`** instruction automatically sets the GIE bit. To disable interrupts in the ISR so that they remain disabled after the ISR returns, the individual interrupt enable bits for each source of interrupts must be cleared.

2.5.3 Interrupt Latency

The interrupt latency is the time from the interrupt event occurring to first ISR instruction being latched from the decode to the execute stage. If the interrupt comes from a Port B input and the $\overline{\text{SYNC}}$ bit in the FUSE1 register is 0, an additional two cycles of synchronization delay are added to the interrupt latency.

The **`iread`** or **`iwrite`** instructions are blocking (i.e. prevent other instructions from being executed) for 4 cycles. If these instructions are used in mainline code, interrupt latency may be increased by an additional 3 cycles.

When an interrupt event is triggered, the CPU speed is changed to the speed specified by the INTSPD register. The SPDREG register is copied to a shadow register, then loaded with the value from the INTSPD register.

If the clock source for the system clock before the interrupt is the same as after the interrupt (i.e. only the core divider is modified), then the interrupt latency is deterministic with respect to the post-interrupt CPU clock. The interrupt latency is 3 cycles for synchronous interrupts.

For example, if the clock divisor is changed from 128 to 1 due to an interrupt, then the interrupt latency is 3 cycles with respect to the post-interrupt clock.

As another example, if the INTSPD register is configured such that the system clock comes from the PLL and the clock divisor is 1 (100 MIPS from 2 MHz), then the mainline code can reduce the clock divisor to a slower speed (e.g. a clock divisor of 128) without affecting the interrupt latency.

If the clock source is changed, then the delay to change the system clock will be up to one cycle of the slower of the pre- and post-interrupt clocks. The total interrupt latency will be this delay plus the normal interrupt latency (with respect to the new core clock).

If the interrupt speed change requires re-enabling the clock multiplier PLL or crystal oscillator, then the interrupt latency will be extended by the PLL or oscillator startup stabilization period. These delay times are programmed in the WUDP2:0 and WUDX2:0 fields of the FUSE0 register, respectively.

2.5.4 Return From Interrupt

The **`reti`** instruction word includes three bits which control its operation.

Table 2-3 `reti` Instruction Options

Bit	Function
2	Reinstate the pre-interrupt speed 1 = enable, 0 = disable
1	Store the PC+1 value in the INTVECH and INTVECL registers 1 = enable, 0 = disable
0	Add W to the T0TMR register 1 = enable, 0 = disable

Updating the interrupt vector allows the programmer to implement a sequential state machine. The next interrupt will resume the code directly after the previous **`reti`** instruction.

The **`reti`** instruction takes 1 cycle to execute, and there is a further delay of 2 cycles at the mainline code speed to load the pipeline before the mainline code is resumed.

2.5.5 Disabled Resources

If a peripheral is disabled, it does not have the ability to set an interrupt flag. The interrupt flag, however, is still a valid source of interrupt.

If software sets an interrupt flag, the corresponding interrupt enable bit is set, and the GIE bit is set, then the CPU will be interrupted whether or not the peripheral is enabled or disabled.

If a peripheral is disabled inside the ISR, then its interrupt flag must be cleared to prevent a spurious interrupt from being taken when the ISR completes.

2.5.6 Clock Stop Mode

When a speed change occurs, it is possible for the CPU clock source to be disabled. The clock to the CPU core may be disabled while the system clock is left running, or the system clock may be disabled which also disables the CPU core clock. When the system clock is disabled, the interrupt logic continues to function, and the watchdog timer and real-time timer may be enabled to keep running. (For minimum power consumption in clock stop mode, disable these timers if they are not needed.)

Recovery from clock stop mode to normal execution is possible from these sources:

- External interrupts (i.e. Port B interrupts)
- Real-time timer interrupts
- Watchdog timer overflow reset

- Brown-out voltage reset
- $\overline{\text{RST}}$ external reset

The first two sources listed above do not reset the chip, so program execution continues from where it was stopped. The last three sources reset the chip, so software must perform all of its reset initialization tasks to recover. This usually requires additional time, as compared to recovery through an external interrupt.

2.6 Reset

There are five sources of reset:

- Power-On Reset
- Brown-Out Reset
- Watchdog Reset
- External Reset (from the $\overline{\text{RST}}$ pin)
- Target Reset (from the debugging interface)

Each of these reset conditions causes the program counter to branch to the top of the program memory (word address FFF0).

The IP2022 incorporates a Power-On Reset (POR) detector that generates an internal reset as IOVDD rises during power-up. Figure 2-7 is a block diagram of the reset logic. It includes a 10-bit startup timer and a reset latch. The startup timer controls the reset time-put delay. The reset latch controls the internal reset signal. On power-up, the reset latch is set (CPU held in reset), and the startup timer starts counting once it detects a valid logic high signal on the $\overline{\text{RST}}$ pin. Once the startup timer reaches the end of the

timeout period, the reset latch is cleared, releasing the CPU from reset.

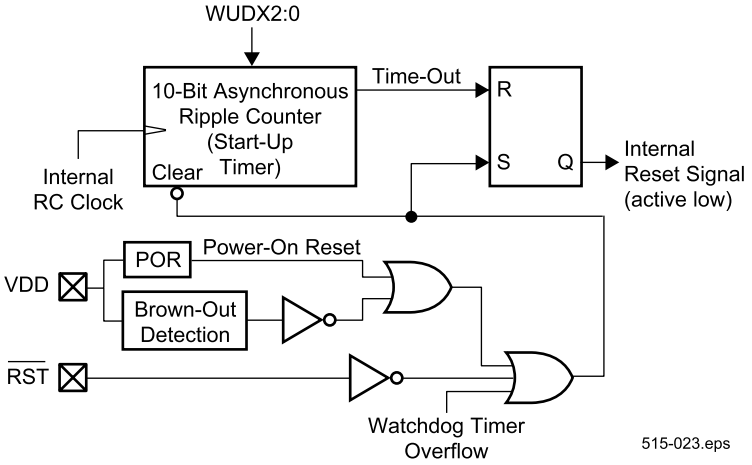


Figure 2-7 On-Chip Reset Circuit Block Diagram

The status register contains two bits to indicate the source of the reset, WD and BO. The WD bit is cleared on reset unless the reset was caused by the watchdog timer, in which case the WD bit is set. The BO bit is cleared on reset unless the reset was caused by the brown-out logic, in which case, the BO bit is set.

Figure 2-8 shows a power-up sequence in which $\overline{\text{RST}}$ is not tied to the IOVDD pin and the IOVDD signal is allowed to rise and stabilize before $\overline{\text{RST}}$ pin is brought high. The device will actually come out of reset after the startup stabilization period (Tstartup) from $\overline{\text{RST}}$ going high. The WUDX2:0 bits of the FUSE0 register specify the length of the stabilization period.

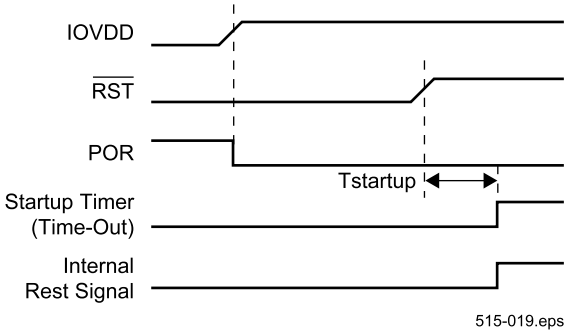


Figure 2-8 Power-On Reset Timing, Separate $\overline{\text{RST}}$ Signal

The brown-out circuitry resets the chip when device power (AVdd) dips below its minimum allowed value, but not to zero, and then recovers to the normal value.

Figure 2-9 shows the on-chip Power-On Reset sequence in which the $\overline{\text{RST}}$ and IOVDD pins are tied together. The IOVDD signal is stable before the startup timer expires. In this case, the CPU receives a reliable reset.

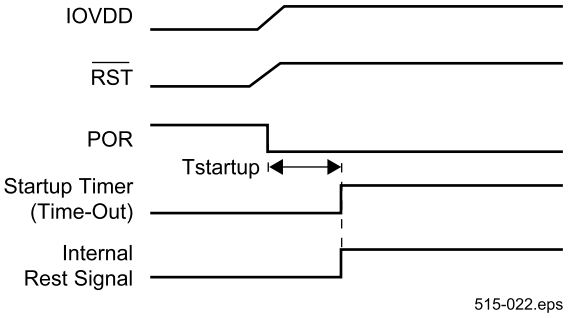


Figure 2-9 Power-On Reset, $\overline{\text{RST}}$ Tied to IOVDD

However, Figure 2-10 depicts a situation in which IOVDD rises too slowly. In this scenario, the startup timer will time out prior to IOVDD reaching a valid operating voltage level (IOVDD min). This means the CPU will come out of reset and start operating with the supply voltage below the level required for reliable performance. In this situation, an external RC circuit is recommended for driving $\overline{\text{RST}}$. The RC delay should exceed the time period required for IOVDD to reach a valid operating voltage.

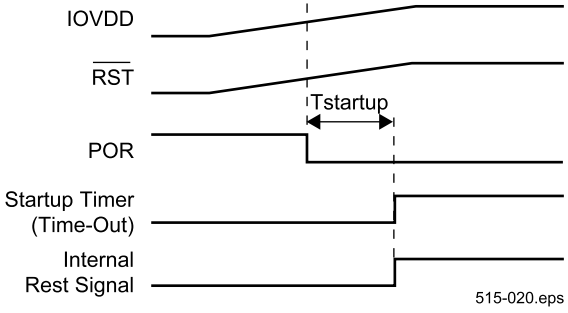


Figure 2-10 IOVDD Rise Time Exceeds Tstartup

Figure 2-11 shows the recommended external reset circuit. The external reset circuit is required only if the IOVDD rise time has the possibility of being too slow.

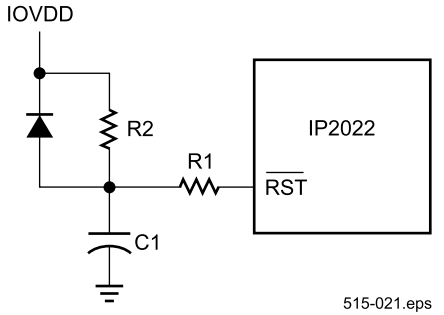


Figure 2-11 External Reset Circuit

The diode D discharges the capacitor when IOVDD is powered down.

R1 = 100 Ω to 1K Ω will limit any current flowing into $\overline{\text{RST}}$ from external capacitor C1. This protects the $\overline{\text{RST}}$ pin from breakdown due to Electrostatic Discharge (ESD) or Electrical Overstress (EOS).

R2 < 40K Ω is recommended to make sure that voltage drop across R2 leaves the $\overline{\text{RST}}$ pin above a Vih level.

C1 should be chosen so that R2 × C1 exceeds the time period required for IOVDD to reach a valid operating voltage.

2.6.1 Brown-Out Detector

The on-chip brown-out detection circuitry resets the CPU when AVdd dips below the brown-out voltage level programmed in the BOR2:0 bits of the FUSE1 register. Bits in the FUSE1 register are flash memory cells which cannot be changed dynamically during program execution.

The device is held in reset as long as AVdd stays below the brown-out voltage. The CPU will come out of reset when AVdd rises above the brown-out voltage. The brown-out level can be programmed using the BOR2:0 bits in the FUSE register, as shown in Table 2-4.

Table 2-4 Brown-Out Voltage Levels

BOR2:0	Voltage
000	2.30V
001	2.25V
010	2.20V
011	2.15V
100	2.10V
101	2.05V
110	2.00V
111	Disabled

2.6.2 Reset and Interrupt Vectors

After reset, the PC is loaded with 0xFFF0, which is near the top of the program memory space. Typical activities for the reset initialization code include:

- Setting up the FCFG register with appropriate values for flash timing compensation.
- Issuing a **speed** instruction to initialize the CPU core clock speed.
- Checking for the cause of reset (brown-out voltage, watchdog timer overflow, or other cause). In some applications, a "warm" reset allows some data initialization procedures to be skipped.
- Copying speed-critical sections of code from flash memory to program RAM.
- Setting up data memory structures (stacks, tables, etc.).
- Initializing peripherals for operation (timers, etc.).
- Initializing the dynamic interrupt vector and enabling interrupts.

Because the default interrupt vector location is 0x0000, which is in shadow RAM, interrupts should not be enabled until the ISR is loaded in shadow RAM or the dynamic interrupt vector is loaded with the address of an ISR in flash memory. There is a single dynamic interrupt vector shared by all interrupts. The interrupt vector can be changed by loading the INTVECH and INTVECL registers, or by issuing a `reti` instruction with an option specifying that the interrupt vector should be updated with the current PC value.

2.6.3 Register States Following Reset

The effect of different reset sources on a register depends on the register and the type of reset operation. Some registers are initialized to specific values, some are left unchanged, and some have undefined reset states.

A register that starts with an undefined state should be initialized by the software to a known value. Do not simply test the initial state and rely on it starting in that state consistently. Table 2-5 lists the IP2022 registers and shows the state of each register upon reset, with a different column for each type of reset. See Table C-1 and Table C-2 for more detailed information.

Table 2-5 Register States Following Reset

Register	Power-On	$\overline{\text{RST}}$ Asserted	Brown-Out Voltage	Watchdog Timer Overflow
PCH	0xFF	0xFF	0xFF	0xFF
PCL	0xF0	0xF0	0xF0	0xF0
CALLH	0xFF	0xFF	0xFF	0xFF
CALLL	0xFF	0xFF	0xFF	0xFF
STATUS	Bits 7:5 - 111 Bits 4:3 - 00 Bits 2:0 - 000	Bits 7:5 - 111 Bits 4:3 - 00 Bits 2:0 - 000	Bits 7:5 - 111 Bits 4:3 - 01 Bits 2:0 - 000	Bits 7:5 - 111 Bits 4:3 - 10 Bits 2:0 - 000
SPDREG	0x93	0x93	0x93	0x93
XCFG	0x01*	0x01*	0x01*	0x01*
Global registers	Undefined	Unchanged	Undefined	Unchanged

Table 2-5 Register States Following Reset

Register	Power-On	$\overline{\text{RST}}$ Asserted	Brown-Out Voltage	Watchdog Timer Overflow
Data memory	Undefined	Unchanged	Undefined	Unchanged
All I/O port registers except RxDIR	0x00	0x00	0x00	0x00
RxDIR registers	0xFF	0xFF	0xFF	0xFF
All other registers	Undefined	Unchanged	Undefined	Unchanged

* The FBUSY bit in the XCFG register is set while an instruction is fetched from flash memory and while the flash memory is busy with a read, write, or erase operation.

2.7 Clock Oscillator

There are two clock oscillators, the OSC oscillator and the RTCLK oscillator. The OSC oscillator is capable of operating at 1 to 6 MHz using an external crystal or ceramic resonator. Using the PLL clock multiplier, the OSC clock is intended to provide the time base for running the CPU core at speeds up to 100 MHz. The RTCLK oscillator operates at 32.768 kHz using an external crystal. This oscillator is intended for running the real-time timer when the OSC oscillator and PLL clock multiplier are turned off. Either clock source can be driven by an external clock signal, up to 150 MHz for the OSC1 input and up to 100 MHz for the RTCLK1 input.

Figure 2-12 shows the clock logic. The PLL clock multiplier has a fixed multiplication factor of 50. The PLL is preceded by a divider capable of any integer divisor between 1 and 8, as controlled by the PIN2:0 bits of the FUSE0 register. The PLL is followed by a second divider capable of any integer divisor between 1 and 4, as controlled by the POUT1:0 bits of the FUSE0 register. A third divider which only affects the clock to the CPU core is controlled by the speed change mechanism described in Section 2.4. If both the OSC oscillator and PLL are re-enabled simultaneously, the delay is controlled by only the WUDX2:0 bits. Bits in the FUSE0 register are flash memory cells which cannot be changed dynamically during program execution. For more information about the FUSE0 register, see Section 6.2.

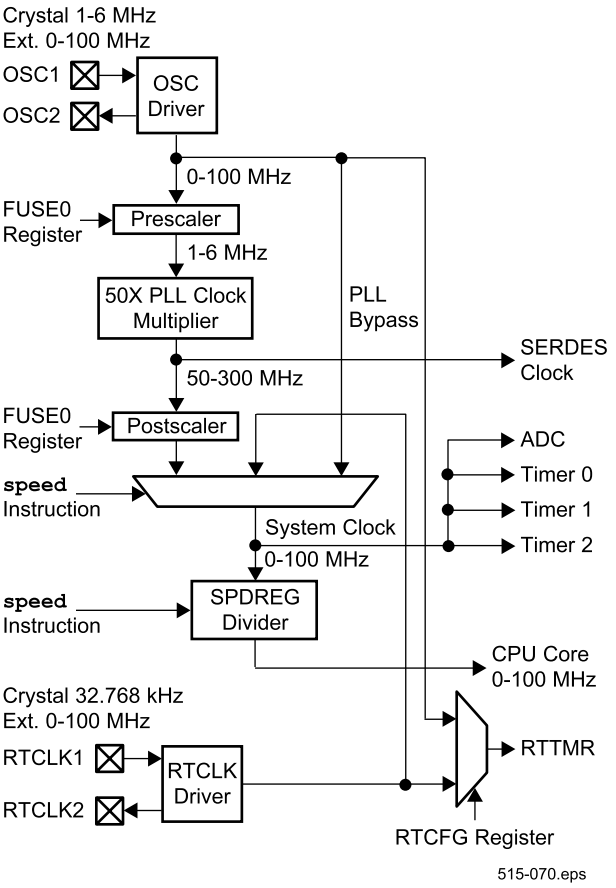


Figure 2-12 Clock Logic

2.7.1 External Connections

Figure 2-13 shows the connections for driving the OSC or RTCLK clock sources with an external signal. To drive the OSC clock source, the external clock signal is driven on the OSC1 pin and the OSC2 pin is left open. The external clock signal driven on the OSC1 pin may be any frequency up to 150 MHz. To drive the RTCLK clock source, the external clock signal is driven on the RTCLK1 input and the RTCLK2 output is left open. The external clock signal driven on the RTCLK1 pin may be any frequency up to 100 MHz.

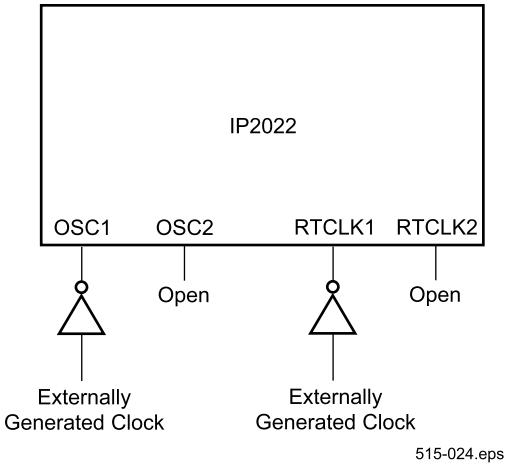


Figure 2-13 External Clock Input

Figure 2-14 shows the connections for attaching an external crystal to the OSC or RTCLK oscillator. For the OSC oscillator, a crystal between 1 MHz and 6 MHz is connected across the OSC1 and OSC2 pins. For the RTCLK oscillator, a 32.768 kHz crystal is connected across the RTCLK1 and RTCLK2 pins. No external capacitors are required.

For proper operation of the crystal or resonator, the total printed circuit board trace length for the OSC1 and OSC2 signals must be kept to less than 1 inch (2.5 cm) each, and the capacitive loading must be kept to less than 3 picofarads. Routing should be direct and no vias should be used.

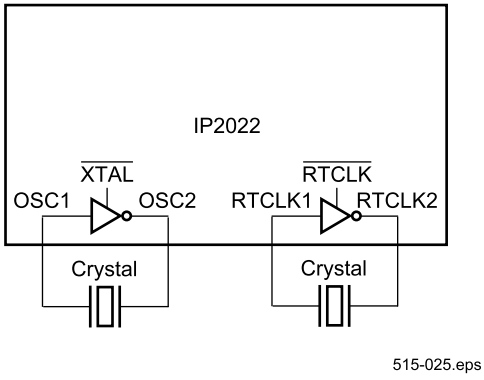


Figure 2-14 Crystal Connection

Figure 2-15 shows the connections for attaching an external ceramic resonator to the OSC oscillator. The frequency of the resonator must be between 1 MHz and 6 MHz. The value of the external capacitors C1 and C2 is 5 pF. The RTCLK oscillator may not be used with an external ceramic resonator.

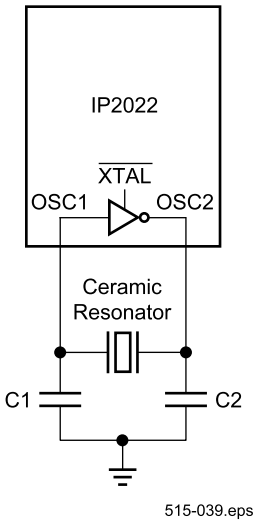


Figure 2-15 Ceramic Resonator Connection

3.0 Instruction Set Architecture

The IP2022 implements a powerful load-store RISC architecture with a rich set of arithmetic and logical operations, including signed and unsigned 8-bit × 8-bit integer multiply with a 16-bit product.

The CPU operates on data held in 128 special-purpose registers, 128 global registers, and 3840 bytes of data memory, shown in Figure 3-1. The special-purpose registers are dedicated to control and status functions for the CPU and peripherals. The global registers and data memory may be used for any functions required by software, the only distinction among them being that the 128 global registers (addresses 0x080 to 0x0FF) can be accessed using a direct addressing mode. The remaining 3840 bytes of data memory (between addresses 0x100 and 0xFFF) must be accessed using indirect or indirect-with-offset addressing modes. The IPH/IPL register is the pointer for the indirect addressing mode, and the DPH/DPL and SPH/SPL registers are the pointers for the indirect-with-offset addressing modes.

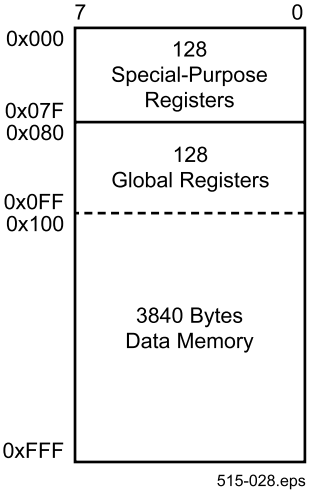


Figure 3-1 Data Memory Map

3.1 Addressing Modes

The arithmetic and logical instructions have one or two operands. The two-operand instructions implicitly use the W register as one of the operands. The one-operand instructions and one of the two operands used by two-operand instructions access data memory using addressing modes. The addressing mode identifies the location in the data memory that is being accessed. A 9-bit field within the instruction, called the "fr" field, specifies the addressing mode and the address (in the case of direct addressing) or the address offset (in the case of indirect-with-offset addressing), as shown in Table 3-1.

Table 3-1 Addressing Mode Summary

"fr" Field	Mode	Assembler Syntax	Effective Address (EA) and Range Restrictions
000000000	Indirect	**mov w,(ip)** **mov (ip),w**	EA = IPH || IPL 0x020 ≤ EA ≤ 0xFFF
00nnnnnnn	Direct, special-purpose registers	**mov w,fr** **mov fr,w**	EA = nnnnnnn 0x002 ≤ EA ≤ 0x07F
01nnnnnnn	Direct, global registers	**mov w,fr** **mov fr,w**	EA = 0x080 + nnnnnnn 0x080 ≤ EA ≤ 0x0FF
10nnnnnnn	Indirect with off-set, data pointer	**mov w,offset(dp)** **mov offset(dp),w**	EA = DPH || DPL + nnnnnnn 0x000 ≤ nnnnnnn ≤ 0x07F 0x020 ≤ EA ≤ 0xFFF
11nnnnnnn	Indirect with off-set, stack pointer	**mov w,offset(sp)** **mov offset(sp),w**	EA = SPH || SPL + nnnnnnn 0x000 ≤ nnnnnnn ≤ 0x07F 0x020 ≤ EA ≤ 0xFFF

3.1.1 Pointer Registers

When an addition or increment instruction (i.e. **add**, **addc**, **inc**, **incsz**, or **incsnz**) on the low byte of a pointer register (i.e. IPL, DPL, SPL, or ADDRL) generates a carry, the high part of the register is incremented. For example, if the IP register holds 0x00FF and an **inc ipl** instruction is executed, the register will hold 0x0100 after the instruction. When a subtraction or decrement instruction (i.e. **sub**, **subc**, **dec**, **decsz**, or

decsnz) generates a borrow, the high part of the register is decremented. Because carry and borrow are automatically handled, the **addc** and **subc** instructions are not needed for arithmetic on pointer registers.

3.1.2 Direct Addressing Mode

Figure 3-2 shows the direct addressing mode used to reference the special-purpose registers. Seven bits from the "fr" field allow addressing up to 128 special-purpose registers. (Not all 128 locations in this space are implemented in the IP2022; several locations are reserved for future expansion.)

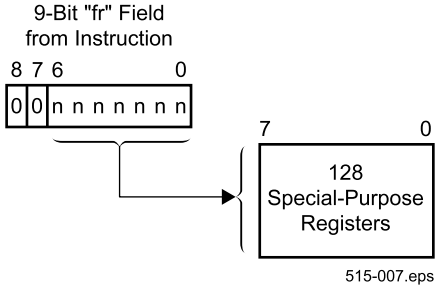


Figure 3-2 Direct Addressing, Special-Purpose Registers

The following code example uses direct addressing.

```
mov    w,0x012     ;load W with the contents of
                   ;the memory location at 0x012
                   ;(the DATAL register)
```

Figure 3-3 shows the direct addressing mode used to reference the global registers. This mode is distinguished from the mode used to access the special-purpose registers with bit 7 of the "fr" field. Because these registers have this additional addressing mode not available for the other registers, they are especially useful for holding global variables and frequently accessed data.

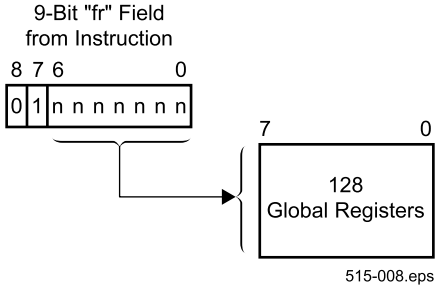


Figure 3-3 Direct Addressing, Global Registers

3.1.3 Indirect Addressing

Indirect addressing is used when all of the bits in the "fr" field are clear. The location of the operand is specified by a 12-bit pointer in the IPH and IPL registers. The upper four bits of the IPH register are not used. Addresses from 0x000 to 0x01F cannot be accessed reliably with this addressing mode, therefore it must not be used for this purpose. (Direct addressing should be used instead.) Figure 3-4 shows indirect addressing.

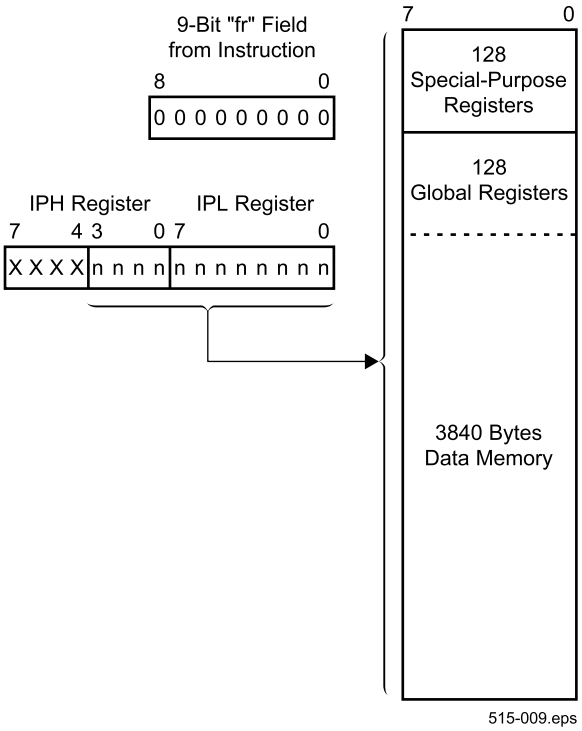


Figure 3-4 Indirect Addressing

The following code example uses indirect addressing.

```
mov    w,#0x03     ;load W with 0x03
mov    iph,w       ;load the high byte of the
                   ;indirect pointer from W
mov    w,#0x85     ;load W with 0x85
mov    ipl,w       ;load the low byte of the
                   ;indirect pointer from W
mov    w,(ip)      ;load W with the contents of the
                   ;memory location at address 0x385
```

3.1.4 Indirect-with-Offset Addressing

Indirect-with-offset addressing is used when bit 8 of the "fr" field is set. The location of the operand is specified by a 7-bit unsigned immediate from the "fr" field added to a 12-bit base address in a pointer register. Addresses from 0x000 to 0x01F cannot be accessed reliably with this addressing mode, therefore it must not be used for this purpose. (Direct addressing should be used instead.)

When bit 7 of the "fr" field is clear, the DPH/DPL register (i.e. the data pointer) is selected as the pointer register. The upper four bits of the DPH register are not used. Figure 3-5 shows indirect-with-offset addressing using the DPH/DPL register as the pointer register.

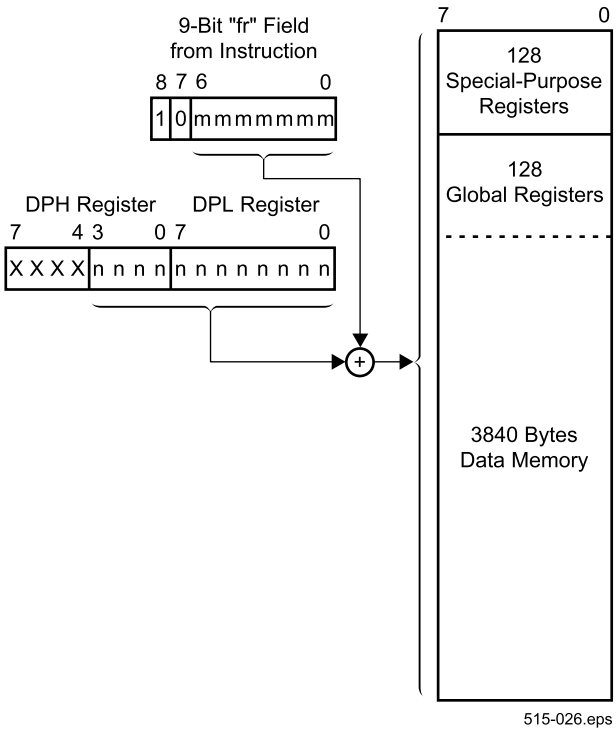


Figure 3-5 Indirect-with-Offset Addressing, Data Pointer

The following code example uses indirect-with-offset addressing.

```
MyStuff=  0x0385      ;define address MyStuff
loadh     MyStuff     ;load the high byte of the
                      ;DPH/DPL register with 0x03
loadl     MyStuff     ;load the low byte of the
                      ;DPH/DPL register with 0x85
mov       w,8(dp)     ;load W with the contents of
                      ;the memory location at
                      ;address 0x38D
                      ;(i.e. 0x385 + 0x008)
```

When bit 7 of the "fr" field is set, the SPH/SPL register (i.e. the stack pointer) is selected as the pointer register. The upper four bits of the SPH register are not used. Figure 3-6 shows indirect-with-offset addressing using the SPH/SPL register as the pointer register. In addition to this indirect-with-offset addressing mode, there are also **push** and **pop** instructions which automatically increment and decrement the SPH/SPL register while performing a data transfer between the top of stack and a data memory location specified by the "fr" field. Stacks grow down from higher addresses to lower addresses. This stack addressing mechanism is independent from the hardware stack used for subroutine call and return.

When a **pop** instruction is used with the indirect-with-offset addressing mode, the address calculation for the "fr" operand is made using the value in the SPH/SPL register *before* the automatic increment, even though the stack operand itself is addressed using the value *after* the automatic increment.

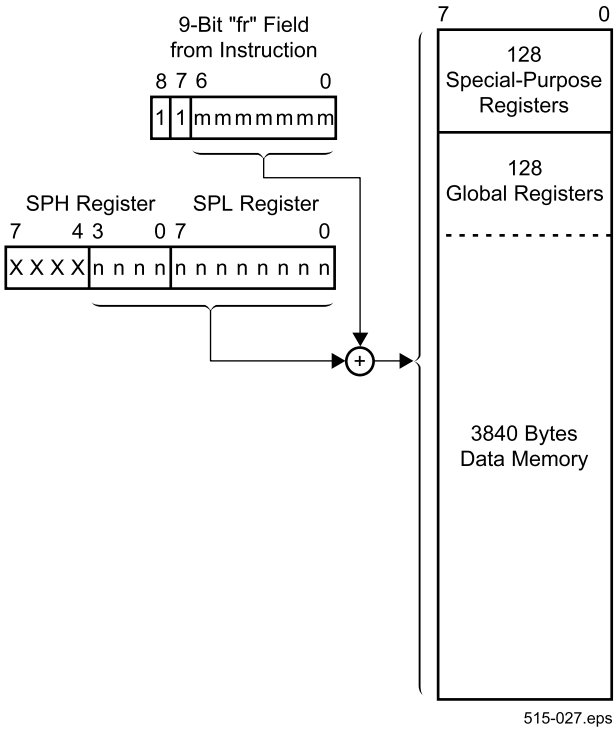


Figure 3-6 Indirect-with-Offset Addressing, Stack Pointer

3.2 Instruction Set

The instruction set consists entirely of single-word (16-bit) instructions, most of which can be executed at a rate of one instruction per clock cycle, for a throughput of up to 100 MIPS when executing out of program RAM.

Assemblers may implement additional instruction mnemonics for the convenience of programmers, such as a long jump instruction which compiles to multiple IP2022 instructions for handling the page structure of program memory. Refer to the assembler documentation for more information about any instruction mnemonics implemented in the assembler.

3.2.1 Instruction Formats

There are five instruction formats:

- Two-operand arithmetic and logical instructions
- Immediate-operand arithmetic and logical instructions
- Jumps and subroutine calls
- Bit operations
- Miscellaneous instructions

Figure 3-7 shows the two-operand instruction format. The two-operand instructions perform an arithmetic or logical operation between the W register and a data memory location specified by the "fr" field. The D bit indicates the destination operand. When the D bit is clear, the destination operand is the W register. When the D bit is set, the destination operand is specified by the "fr" field.

There are some exceptions to this behavior. The multiply instructions always load the 16-bit product into the MULH and W registers. The MULH register receives the upper 8 bits of the product, and the W register receives the lower 8 bits.

Traditionally single-operand instructions, such as increment, are available in two forms distinguished by the D bit. When the D bit is clear, the source operand is specified by the "fr" field and the destination operand is the W register. When the D bit is set, the data memory location specified by the "fr" field is both the source and destination operand.

Also, there are a few cases of unrelated instructions, such as **`clr`** and **`cmp`**, which are distinguished by the D bit.

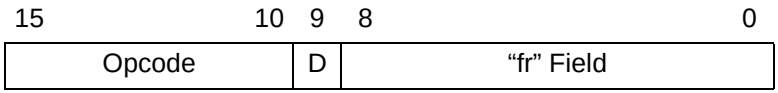


Figure 3-7 Two-Operand Instruction Format

Figure 3-8 shows the immediate-operand instruction format. In this format, an 8-bit literal value is encoded in the instruction field. Usually the W register is the destination operand, however this format also includes instructions that use the top of the stack or a special-purpose register as the destination operand.

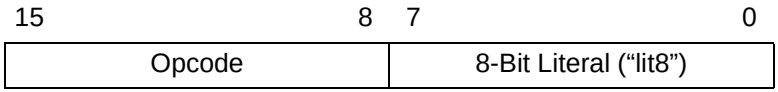


Figure 3-8 Immediate-Operand Format

Figure 3-9 shows the format of the jump and subroutine call instructions. 13 bits of the entry point address are encoded in the instruction. The remaining three bits come from the PA2:0 bits of the STATUS register.

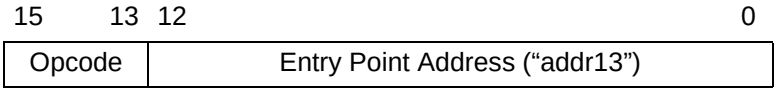


Figure 3-9 Jump and Call Instruction Format

Figure 3-10 shows the format of the instructions that clear, set, and test individual bits within registers. The register is specified by the "fr" field, and a 3-bit field in the instruction selects one of the eight bits in the register.

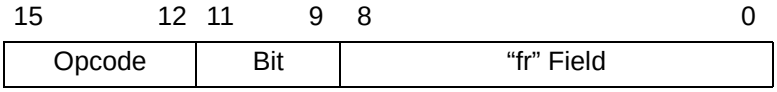


Figure 3-10 Bit Operation Instruction Format

Figure 3-11 shows the format of the remaining instructions.

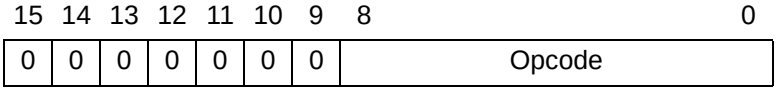


Figure 3-11 Miscellaneous Instruction Format

3.2.2 Instruction Types

The instructions are grouped into the following functional categories:

- Logical instructions
- Arithmetic and shift instructions
- Bitwise operation instructions
- Data movement instructions
- Program control instructions
- System control instructions

The following sections describe the characteristics of the instructions in these categories.

3.2.3 Logical Instructions

Each logic instruction performs a standard logical operation (AND, OR, exclusive OR, or logical complement) on the respective bits of the 8-bit operands. The result of the logic operation is written to W or to the data memory location specified by the "fr" field.

All of these instructions take one clock cycle for execution.

3.2.4 Arithmetic and Shift Instructions

Each arithmetic or shift instruction performs an operation such as add, subtract, add with carry, subtract with carry, rotate left or right through carry, increment, decrement, clear to zero, or swap high/low nibbles. The compare (`cmp`) instruction performs the same operation as subtract, but it only updates the C, DC, and Z

flags of the STATUS register; the result of the subtraction is discarded.

There are instructions available that increment or decrement a register and simultaneously test the result. If the 8-bit result is zero, the next instruction in the program is skipped. These instructions can be used to make program loops. There are also compare-and-skip instructions which perform the same operation as subtract, then perform a conditional skip without affecting either operand or the condition flags in the STATUS register.

All of the arithmetic and shift instructions take one clock cycle for execution, except in the case of the test-and-skip instructions when the tested condition is true and a skip occurs, in which case the instruction takes two cycles.

3.2.5 Bit Operation Instructions

There are four bit operation instructions:

- **`setb`**—sets a single bit in a data register without affecting other bits
- **`clrb`**—clears a single bit in a data register without affecting other bits
- **`sb`**—tests a single bit in a data register and skips the next instruction if the bit is set
- **`snb`**—tests a single bit in a data register and skips the next instruction if the bit is clear

Any bit at any location in data memory can be individually set, cleared, or tested.

All of the bitwise operation instructions take one clock cycle for execution, except in the case of the test-and-skip instructions when the tested condition is true and a skip occurs. If a skip instruction is immediately followed by a **`loadh`**, **`loadl`**, or **`page`** instruction (and the tested condition is true) then two instructions are skipped and the operation consumes three cycles. This is useful for skipping over a conditional branch to another page in which a **`page`** instruction precedes a **`jmp`** instruction. If several **`page`** or **`loadh`**/**`loadl`** instructions immediately follow a skip instruction then they are all skipped plus the next instruction and a cycle is consumed for each. These "extended skip" instructions are interruptible, so they do not affect interrupt latency.

3.2.6 Data Movement Instructions

A data movement instruction moves a byte of data from a data memory location to either the W register or the top of stack, or it moves the byte from either the W register or the top of stack to a data memory location. The location is specified by the "fr" field. The SPH/SPL register pair points to the top of stack. This stack is independent of the hardware stack used for subroutine call and return.

3.2.7 Program Control Instructions

A program control instruction alters the flow of the program by changing the contents of the program counter. Included in this category are the jump, call, return-from-subroutine, and interrupt instructions.

The **`jmp`** instruction has a single operand that specifies the entry point at which to continue execution. The entry point is typically specified in assembly language with a label, as in the following code example.

```
snb     status,0    ;test the carry bit
jmp     do_carry    ;jump to do_carry routine if C = 1

...
do_carry:           ;jump destination label
...                 ;execution continues here
```

If the carry bit is set to 1, the **`jmp`** instruction is executed and program execution continues where the **`do_carry`** label appears in the program.

The **`call`** instruction works in a similar manner, except that it saves the contents of the program counter before jumping to the new address. It calls a subroutine that is terminated by a **`ret`** instruction, as shown in the following code example.

```
call     add_2bytes    ;call subroutine
                       ;add_2bytes
nop                    ;execution returns to here
                       ;after subroutine is finished

...
add_2bytes:            ;subroutine label
...                    ;subroutine code goes here
ret                    ;return from subroutine
```

Returning from a subroutine restores the saved program counter contents, which causes program to resume execution with the instruction immediately following the **`call`** instruction (a **`nop`** instruction, in the above example)

A program memory address is a 16-bit word address. The **`jmp`** and **`call`** instructions specify only the lowest thirteen bits of the jump/call address. The upper 3 bits come from the PA2:0 bits of the STATUS register. An indirect relative jump can be accomplished by adding the contents of the W register to the PCL register (i.e. an **`add pcl,w`** instruction).

Program control instructions such as **`jmp`**, **`call`**, and **`ret`** alter the normal program sequence. When one of these instructions is executed, the execution pipeline is automatically cleared of pending instructions and refilled with new instructions, starting at the new program address. Because the pipeline must be cleared, three clock cycles are required for execution, one to execute the instruction and two to reload the pipeline.

3.2.8 System Control Instructions

A system control instruction performs a special-purpose operation that sets the operating mode of the IP2022 or reads data from the program memory. Included in this category are the following types of instructions:

- **`speed`**—changes the CPU core speed (for saving power)
- **`break`**—enters debug mode
- **`page`**—writes the PA2:0 bits in the STATUS register
- **`loadh`**/**`loadl`**—loads a 16-bit pointer into the DPH and DPL registers
- **`iread`**/**`ireadi`**—reads a word from external memory, program flash memory, or program RAM
- **`iwrite`**/**`iwritei`**—writes a word to the program RAM
- **`fwrite`**—writes the flash program memory
- **`ferase`**—erases the flash program memory
- **`cwdt`**—clears the watchdog timer

3.3 Instruction Pipeline

An instruction goes through a four-stage pipeline to be executed, as shown in Figure 3-2. The first instruction is fetched from the program memory on the first clock cycle. On the second clock cycle, the first instruction is decoded and a second instruction is fetched. On the third clock cycle, the first instruction is executed, the second instruction is decoded, and a third instruction is fetched. On the fourth clock cycle, the first instruction's results are written to its destination, the second instruction is executed, the

third instruction is decoded, and a fourth instruction is fetched. Once the pipeline is full, instructions are executed at the rate of one per clock cycle.

Table 3-2 Pipeline Execution

Stage	Cycle 1	Cycle 2	Cycle 3	Cycle 4
Fetch	Instruction 1	Instruction 2	Instruction 3	Instruction 4
Decode		Instruction 1	Instruction 2	Instruction 3
Execute			Instruction 1	Instruction 2
Write				Instruction 1

Instructions that directly affect the contents of the program counter (such as jumps and calls) require that the pipeline be cleared and subsequently refilled. Therefore, these instructions take two additional clock cycles.

3.4 Subroutine Call/Return Stack

A 16-level hardware call/return stack is provided for saving the program counter on a subroutine call and restoring the program counter on subroutine return. The stack is not mapped into the data memory address space except for the top level, which is accessible as the CALLH and CALLL registers. Software can read and write these registers to implement a deeper stack, in those cases which require nesting subroutines more than 16 levels deep. This stack is independent of the stack used with the **push** and **pop** instructions and the SPH/SPL register.

When a subroutine is called, the return address is pushed onto the subroutine stack, as shown in Figure 3-12. Specifically, each saved address in the stack is moved to the next lower level to make room for the new address to be saved. Stack 1 receives the contents of the program counter. Stack 16 is overwritten with what was in Stack 15. The contents of stack 16 are lost.

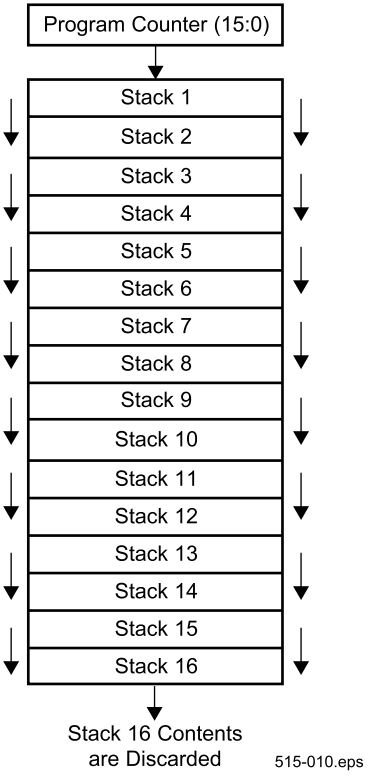


Figure 3-12 Stack Operation on Subroutine Call

When a return instruction is executed the subroutine stack is popped, as shown in Figure 3-13. Specifically, the contents of Stack 1 are copied into the program counter and the contents of each stack level are moved to the next higher level. When a value is popped off the stack, the bottom entry is initialized to 0xFFFF. For example, Stack 1 receives the contents of Stack 2, etc., until Stack 15 is overwritten with the contents of Stack 16. Stack 16 is initialized to 0xFFFF.

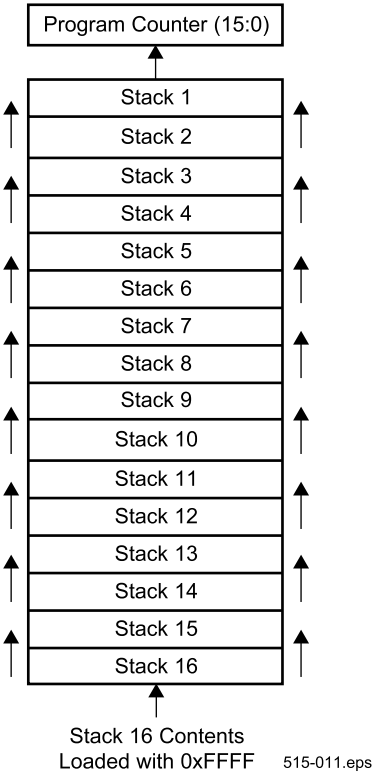


Figure 3-13 Stack Operation on Subroutine Return

For program bugs involving stack underflow, the instruction at word address 0xFFFF can be used to jump to an appropriate handler. For example, system recovery may be possible by jumping to the reset vector at word address 0xFFF0.

3.5 Key to Abbreviations and Symbols

Symbol	Description
W	Working register
fr	"File register" field (a 9-bit operand address specified in the instruction)
PCL	Virtual register for direct PC modification (special-purpose register 0x009)
STATUS	STATUS register (special-purpose register 0x00B)
IPH	Upper half of register for indirect addressing (special-purpose register 0x004)
IPL	Lower half of register for indirect addressing (special-purpose register 0x005)
DPH	Upper half of pointer register for indirect-with-offset addressing (special-purpose register 0x00C)
DPL	Lower half of pointer register for indirect-with-offset addressing (special-purpose register 0x00D)
SPH	Upper half of pointer register for indirect-with-offset addressing (special-purpose register 0x006)
SPL	Lower half of pointer register for indirect-with-offset addressing (special-purpose register 0x007)
C	Carry bit in the STATUS register (bit 0)
DC	Digit Carry bit in the STATUS register (bit 1)
Z	Zero bit in the STATUS register (bit 2
PD	Power Down bit in the STATUS register (bit 3)
TO	Watchdog Timeout bit in the STATUS register (bit 4)
PA2:0	Page bits in the STATUS register (bits 7:5)

Symbol	Description
WDT	Watchdog Timer counter and prescaler
rx	Port control register pointer (RA, RB, RC, RD, RE, RF, or RG)
f	"fr" field bit in opcode
k	Constant value bit in opcode
n	Numerical value bit in opcode
b	Bit position selector bit in opcode
,	Register/bit selector separator in assembly language instruction (e.g. **`clrb status,z`**)
#	Immediate literal designator in assembly language instruction (e.g. **`mov w,#0xff`**)
#lit8	8-bit literal value in assembly language instruction
addr13	13-bit address in assembly language instruction
addr16	16-bit address in assembly language instruction
(address)	Contents of memory referenced by address
|	Logical OR
||	Concatenation
^	Logical exclusive OR
&	Logical AND
!=	inequality

3.6 Instruction Set Summary Tables

Table 3-3 through Table 3-8 list all of the IP2022 instructions, organized by category. For each instruction, the table shows the instruction mnemonic (as written in assembly language), a brief

description of what the instruction does, the number of instruction cycles required for execution, the binary opcode, and the flags in the STATUS register affected by the instruction.

Although the number of clock cycles for execution is typically 1, for the skip instructions the exact number of cycles depends whether the skip is taken or not taken. Taking the skip adds 1 cycle. The effect of extended skip instructions (i.e. a skip followed by a **`loadh`**, **`loadl`**, or **`page`** instruction) is not shown.

Table 3-3 Logical Instructions

Assembler Syntax	Description	Cycles	Opcode	Flags Affected
`and fr,w`	AND fr,W into fr	1	`0001 011f ffff ffff`	Z
`and w,fr`	AND W,fr into W	1	`0001 010f ffff ffff`	Z
`and w,#lit8`	AND W,literal into W	1	`0111 1110 kkkk kkkk`	Z
`not fr`	Complement fr into fr	1	`0010 011f ffff ffff`	Z
`not w,fr`	Complement fr into W	1	`0010 010f ffff ffff`	Z
`or fr,w`	OR fr,W into fr	1	`0001 001f ffff ffff`	Z
`or w,fr`	OR W,fr into W	1	`0001 000f ffff ffff`	Z
`or w,#lit8`	OR W,literal into W	1	`0111 1101 kkkk kkkk`	Z

Table 3-3 Logical Instructions (continued)

Assembler Syntax	Description	Cycles	Opcode	Flags Affected
xor fr,w	XOR fr,W into fr	1	0001 101f ffff ffff	Z
xor w,fr	XOR W,fr into W	1	0001 100f ffff ffff	Z
xor w,#lit8	XOR W,literal into W	1	0111 1111 kkkk kkkk	Z

Table 3-4 Arithmetic and Shift Instructions

Assembler Syntax	Description	Cycles	Opcode	Flags Affected
add fr,w	Add fr,W into fr	1	**0001 111f ffff ffff**	C, DC, Z
add w,fr	Add W,fr into W	1	**0001 110f ffff ffff**	C, DC, Z
add w,#lit8	Add W,literal into W	1	**0111 1011 kkkk kkkk**	C, DC, Z
addc fr,w	Add carry,W,fr into fr	1	**0101 111f ffff ffff**	C, DC, Z
addc w,fr	Add carry,W,fr into W	1	**0101 110f ffff ffff**	C, DC, Z
clr fr	Clear fr	1	**0000 011f ffff ffff**	Z
cmp w,fr	Compare W,fr then update STATUS	1	**0000 010f ffff ffff**	C, DC, Z

Table 3-4 Arithmetic and Shift Instructions (continued)

Assembler Syntax	Description	Cycles	Opcode	Flags Affected
cmp w,#lit8	Compare W,literal then update STATUS	1	**0111 1001 kkkk kkkk**	C, DC, Z
cse w,fr	Compare W,fr then skip if equal	1/2	**0100 001f ffff ffff**	None
cse w,#lit8	Compare W,literal then skip if equal	1/2	**0111 0111 ffff ffff**	None
csne w,fr	Compare W,fr then skip if not equal	1/2	**0100 000f ffff ffff**	None
csne w,#lit8	Compare W,literal then skip if not equal	1/2	**0111 0110 kkkk kkkk**	None
cwdt	Clear Watchdog Timer	1	**0000 0000 0000 0100**	None
dec fr	Decrement fr into fr	1	**0000 111f ffff ffff**	Z
dec w,fr	Decrement fr into W	1	**0000 110f ffff ffff**	Z

Table 3-4 Arithmetic and Shift Instructions (continued)

Assembler Syntax	Description	Cycles	Opcode	Flags Affected
decsnz fr	Decrement fr into fr then skip if not zero (STATUS not updated)	1/2	**0100 111f ffff ffff**	None
decsnz w,fr	Decrement fr into W then skip if not zero (STATUS not updated)	1/2	**0100 110f ffff ffff**	None
decsz fr	Decrement fr into fr then skip if zero (STATUS not updated)	1/2	**0010 111f ffff ffff**	None
decsz w,fr	Decrement fr into W then skip if zero (STATUS not updated)	1/2	**0010 110f ffff ffff**	None
inc fr	Increment fr into fr	1	**0010 101f ffff ffff**	Z
inc w,fr	Increment fr into W	1	**0010 100f ffff ffff**	Z

Table 3-4 Arithmetic and Shift Instructions (continued)

Assembler Syntax	Description	Cycles	Opcode	Flags Affected
`incsnz fr`	Increment fr into fr then skip if not zero (STATUS not updated)	1/2	`0101 101f ffff ffff`	None
`incsnz w,fr`	Increment fr into W then skip if not zero (STATUS not updated)	1/2	`0101 100f ffff ffff`	None
`incsz fr`	Increment fr into fr then skip if zero (STATUS not updated)	1/2	`0011 111f ffff ffff`	None
`incsz w,fr`	Increment fr into W then skip if zero (STATUS not updated)	1/2	`0011 110f ffff ffff`	None
`muls w,fr`	Signed 8 × 8 multiply (bit 7 = sign) W,fr into MULH || W	1	`0101 010f ffff ffff`	None

Table 3-4 Arithmetic and Shift Instructions (continued)

Assembler Syntax	Description	Cycles	Opcode	Flags Affected
`muls w,#lit8`	Signed 8 × 8 multiply (bit 7 = sign) W,literal into MULH || W	1	`0111 0011 kkkk kkkk`	None
`mulu w,fr`	Unsigned 8 × 8 multiply W,fr into MULH || W	1	`0101 000f ffff ffff`	None
`mulu w,#lit8`	Unsigned 8 × 8 multiply W,literal into MULH || W	1	`0111 0010 kkkk kkkk`	None
`rl fr`	Rotate fr left through carry into fr	1	`0011 011f ffff ffff`	C
`rl w,fr`	Rotate fr left through carry into W	1	`0011 010f ffff ffff`	C
`rr fr`	Rotate fr right through carry into fr	1	`0011 001f ffff ffff`	C
`rr w,fr`	Rotate fr right through carry into W	1	`0011 000f ffff ffff`	C

Table 3-4 Arithmetic and Shift Instructions (continued)

Assembler Syntax	Description	Cycles	Opcode	Flags Affected
sub fr,w	Subtract W from fr into fr	1	**0000 101f ffff ffff**	C, DC, Z
sub w,fr	Subtract W from fr into W	1	**0000 100f ffff ffff**	C, DC, Z
sub w,#lit8	Subtract W from literal into W	1	**0111 1010 kkkk kkkk**	C, DC, Z
subc fr,w	Subtract $\overline{\text{carry}}$,W from fr into fr	1	**0100 101f ffff ffff**	C, DC, Z
subc w,fr	Subtract $\overline{\text{carry}}$,W from fr into W	1	**0100 100f ffff ffff**	C, DC, Z
swap fr	Swap high,low nibbles of fr into fr	1	**0011 101f ffff ffff**	None
swap w,fr	Swap high,low nibbles of fr into W	1	**0011 100f ffff ffff**	None
test fr	Test fr for zero	1	**0010 001f ffff ffff**	Z

Table 3-5 Bit Operation Instructions

Assembler Syntax	Description	Cycles	Opcode	Flags Affected
`clrb fr,bit`	Clear bit in fr	1	`1000 bbbf ffff ffff`	None
`sb fr,bit`	Test bit in fr then skip if set	1/2	`1011 bbbf ffff ffff`	None
`setb fr,bit`	Set bit in fr	1	`1001 bbbf ffff ffff`	None
`snb fr,bit`	Test bit in fr then skip if clear	1/2	`1010 bbbf ffff ffff`	None

Table 3-6 Data Movement Instructions

Assembler Syntax	Description	Cycles	Opcode	Flags Affected
`mov fr,w`	Move W into fr	1	`0000 001f ffff ffff`	None
`mov w,fr`	Move fr into W	1	`0010 000f ffff ffff`	Z
`mov w,#lit8`	Move literal into W	1	`0111 1100 kkkk kkkk`	None
`push fr`	Move fr onto top of stack	1	`0100 010f ffff ffff`	None
`push #lit8`	Move literal onto top of stack	1	`0111 0100 kkkk kkkk`	None
`pop fr`	Move top of stack into fr	1	`0100 011f ffff ffff`	None

Table 3-7 Program Control Instructions

Assembler Syntax	Description	Cycles	Opcode	Flags Affected
`call addr13`	Call subroutine	3	`110k kkkk kkkk kkkk`	None
`jmp addr13`	Jump	3	`111k kkkk kkkk kkkk`	None
`int`	Software interrupt	3	`0000 0000 0000 0110`	None
`nop`	No operation	1	`0000 0000 0000 0000`	None
`ret`	Return from subroutine	3	`0000 0000 0000 0111`	PA2:0
`reti #lit3`	Return from interrupt	3	`0000 0000 0000 1nnn`	All
`retw #lit8`	Return from subroutine with literal into W	3	`0111 1000 kkkk kkkk`	PA2:0

Table 3-8 System Control Instructions

Assembler Syntax	Description	Cycles	Opcode	Flags Affected
break	Software break-point	1	**0000 0000 0000 0001**	None
ferase	Erase flash block	1	**0000 0000 0000 0011**	None
fread	Read from flash memory	1	**0000 0000 0001 1011**	None
fwrite	Write into flash memory	1	**0000 0000 0001 1010**	None
iread	Read external/program memory	4	**0000 0000 0001 1001**	None
ireadi	Read external/program memory and increment	4	**0000 0000 0001 1101**	None
iwrite	Write external/program RAM	4	**0000 0000 0001 1000**	None
iwritei	Write external/program RAM and increment	4	**0000 0000 0001 1100**	None

Table 3-8 System Control Instructions (continued)

Assembler Syntax	Description	Cycles	Opcode	Flags Affected
loadh addr16	Load high data address into DPH	1	**0111 0000 kkkk kkkk**	None
loadl addr16	Load addr16[7:0] into DPL register	1	**0111 0001 kkkk kkkk**	None
page addr16	Load addr16[15:13] into STATUS7:5 (i.e. PA2:0)	1	**0000 0000 0001 0nnn**	PA2:0
speed #lit8	Set speed mode	1	**0000 0001 nnnn nnnn**	None

3.7 Self-Programming Instructions

The IP2022 has several instructions used to read and write the program RAM and the program flash memory. These instructions allow the program flash memory to be read and written through special-purpose registers in the data memory space, which allows the flash memory to be used to store both program code and data.

Because no special programming voltage is required to write to the flash memory, any application may take advantage of this feature at run-time. Typical uses include saving phone numbers and passwords, downloading new or updated software, and logging infrequent events such as errors and Watchdog Timer resets.

The self-programming instructions are not affected by the code-protection flag (the $\overline{\text{CP}}$ bit of the FUSE1 register), so the entire program memory is visible to any software running on the IP2022.

There are five instructions used for self-programming, as shown in Table 3-9. Certain uses of the instructions are not valid. In these cases, the instruction is executed as though it were a **nop** instruction (i.e. the program counter is incremented, but no other registers or bits are affected).

Table 3-9 Instructions Used for Self-Programming

Instruction	Executed From RAM to Operate On Program RAM	Executed From RAM to Operate On Flash Memory	Executed From Flash to Operate On Program RAM	Executed From Flash to Operate On Flash Memory	Executed While $\overline{\text{FWP}}$ Bit in XCFG Register is Clear (FWP = 0)
iread ireadi	Blocking	Non-Blocking	Blocking	Blocking	N/A
iwrite iwritei	Blocking	**nop**	Blocking	**nop**	N/A
fwrite	**nop**	Non-Blocking	**nop**	**nop**	**nop**
ferase	**nop**	Non-Blocking	**nop**	**nop**	**nop**

Blocking instructions take 4 cycles to complete, and prevent other instructions from executing. Non-blocking instructions occupy the CPU pipeline for only one cycle, but they launch a multi-cycle operation which is not complete until indicated by the FBUSY bit in the XCFG register becoming clear.

Read and write operations that involve program memory use two registers. The DATAH/DATAL register is a 16-bit data buffer used for loading or unloading data in program memory. The

ADDRX/ADDRH/ADDRL register holds a 24-bit address used to specify a location in program memory. Like the other pointer registers (IPH/IPL, DPH/DPL, and SPH/SPL), addition to the low byte of the register that results in carry will cause the high part of the register (ADDRX/ADDRH) to be incremented. Subtraction from the low byte of the register that results in borrow will cause the high part of the register to be decremented.

Software can use the FBUSY bit to check that a previous flash memory operation has completed before executing another instruction that accesses flash memory, before jumping or calling program code in flash memory, and before changing the CPU core speed. Software must not attempt to execute out of flash memory while the FBUSY bit is set, because the flash memory is unreadable during that time. Code which reads, writes, or erases flash memory must execute from program RAM, not flash memory. The CPU core speed must not change while a flash memory read, write, or erase operation is in progress. Software must wait at least three cycles before checking the FBUSY bit for completion of the flash operation. It is not necessary to check the FBUSY bit if enough cycles are allowed for the flash operation to complete.

Unlike RAM, flash memory requires an explicit erase operation before being written. The **`ferase`** instruction is used to erase a 512-byte (256-word) block of flash memory. After the block has been erased, individual words can be written with the **`fwrite`** instruction. For example, an **`ferase`** instruction executed on any address from 0x10000 to 0x100FE erases the whole block spanning those addresses. The self-programming instructions

have no access to the flash memory bits used to implement the configuration block.

3.7.1 Interrupts During Flash Operations

Before starting a flash write or erase operation, the flash write timing compensation must be set up properly for the current speed. The CPU core clock is the time base for the flash write timing compensation, so it is critical that the CPU core clock speed is not changed during a flash write or erase operation. Interrupts may be taken during a flash write or erase operation, if the INTSPD register is set up so the speed does not change when an interrupt occurs.

If the flash read timing compensation is set up for a clock divisor of 1 (i.e. fastest speed), interrupts will not cause **`iread`** instructions to fail, so no special precautions need to be taken to avoid violating the flash read access time.

3.7.2 FCFG Register

7	6	5	4	3	2	1	0
FRDTS1:0		FRDTC1:0		FWRT3:0			

Name	Description
FRDTS1:0	The CPU core clock while executing from flash program memory must not exceed 30 MHz, otherwise unreliable operation may result. The IP2022 automatically increases the number of system clock cycles for each CPU core clock cycle when executing from flash, but it is the responsibility of software to load the FRDTS1:0 bits appropriately for the operating frequency, as shown below. The actual speed will be the slower of the speed indicated in the SPDREG register and the speed specified by the FRDTS1:0 bits. The previous CPU core clock divisor is reinstated when jumping back to program RAM from program flash memory. 00 = 1 cycle 0–30 MHz system clock frequency 01 = 2 cycles 30–60 MHz 10 = 3 cycles 60–90 MHz 11 = 4 cycles 90–1200 MHz

Name	Description
FRDTC1:0	The number of system clock cycles for reading the flash memory using an **`iread`** instruction must be specified to prevent the flash memory access time from being exceeded. Because the CPU core is subject to changes in speed, the value programmed in these bits should be appropriate for the fastest speed that might be used (typically, the faster of the main line code and the interrupt service routine). The FRDTC1:0 bits specify the number of CPU core clock cycles required for read access. 00 = 1 cycle 0–30 MHz system clock frequency 01 = 2 cycles 30–60 MHz 10 = 3 cycles 60–90 MHz 11 = 4 cycles 90–1200 MHz

Name	Description
FWRT3:0	The flash memory erase and write timing is derived from the CPU core clock through a programmable divider. The FWRT3:0 bits specify the divisor. The time base must be 1 to 2 microseconds. Below 1 microsecond, the flash memory will be underprogrammed, and data retention is not guaranteed. Above 2 microseconds, the flash memory will be overprogrammed, and reliability is not guaranteed. Because the minimum flash write clock divisor is 2, the minimum clock frequency for self-programming is 1 MHz. 0000 = 2 1–2 MHz CPU core clock frequency 0001 = 3 2–3 MHz 0010 = 4 3–4 MHz 0011 = 6 4–6 MHz 0100 = 8 6–8 MHz 0101 = 12 8–12 MHz 0110 = 16 12–16 MHz 0111 = 24 16–24 MHz 1000 = 32 24–32 MHz 1001 = 48 32–48 MHz 1010 = 64 48–64 MHz 1011 = 96 64–96 MHz 1100 = 128 96–100 MHz 1101 = 192 Reserved 1110 = 256 Reserved 1111 = 384 Reserved

3.8 Instruction Descriptions

This section provides a detailed description of each instruction.

ADD fr,W — Add fr,W into fr

Operation: fr = fr + W

Bits affected: C, DC, Z

Opcode: `0001 111f ffff ffff`

Description: This instruction adds the contents of W to the contents of the specified data memory location and writes the 8-bit result into the same data memory location. W is left unchanged. The register contents are treated as unsigned values.

If the result of addition exceeds 0xFF, the C bit is set and the lower eight bits of the result are written to the data memory location. Otherwise, the C bit is cleared.

If there is a carry from bit 3 to bit 4, the DC (digit carry) bit is set. Otherwise, the bit is cleared.

If the result of addition is zero, the Z bit is set. Otherwise, the Z bit is cleared. A sum of 0x100 is considered zero and therefore sets the Z bit.

Cycles: 1 (3, if jumping by using PCL as the destination)

Example: **`add 0x099,w`**

This example adds the contents of W to data memory location 0x099. For example, if the data memory location holds 0x7F and W holds 0x02, this instruction adds 0x02 to 0x7F, writes the result 0x81 into the data memory location, and clears the C and Z bits. It sets the DC bit because of the carry from bit 3 to bit 4.

ADD W,fr — Add W,fr into W

Operation: W = W + fr

Bits affected: C, DC, Z

Opcode: `0001 110f ffff ffff`

Description: This instruction adds the contents of the specified data memory location to the contents of W and writes the 8-bit result into W. The data memory location is left unchanged. The register contents are treated as unsigned values.

If the result of addition exceeds 0xFF, the C bit is set and the lower eight bits of the result are written to W. Otherwise, the C bit is cleared.

If there is a carry from bit 3 to bit 4, the DC (digit carry) bit is set. Otherwise, the DC bit is cleared.

If the result of addition is zero, the Z bit is set. Otherwise, the Z bit is cleared. A sum of 0x100 is considered zero and therefore sets the Z bit.

Cycles: 1

Example: **`add w,0x099`**

This example adds the contents of data memory location 0x099 to W. For example, if the data memory location holds 0x81 and W holds 0x82, this instruction adds 0x81 to 0x82 and writes the lower eight bits of the result, 0x03, into W. It sets the C bit because of the carry out of bit 7, and clears the DC bit because there is no carry from bit 3 to bit 4. The Z bit is cleared because the result is nonzero.

ADD W,#lit8 — Add W,Literal into W

Operation: W = W + lit8

Bits affected: C, DC, Z

Opcode: `0001 110f ffff ffff`

Description: This instruction adds an 8-bit literal value (a value specified within the instruction) to the contents of W and writes the 8-bit result into W. The operands are treated as unsigned values.

If the result of addition exceeds 0xFF, the C bit is set and the lower eight bits of the result are written to W. Otherwise, the C bit is cleared.

If there is a carry from bit 3 to bit 4, the DC (digit carry) bit is set. Otherwise, the DC bit is cleared.

If the result of addition is zero, the Z bit is set. Otherwise, the Z bit is cleared. A sum of 0x100 is considered zero and therefore sets the Z bit.

Cycles: 1

Example: **`add w,#0x12`**

This example adds 0x12 to the contents of W. For example, if W holds 0x82, this instruction adds 0x12 to 0x82 and writes the lower eight bits of the result, 0x94, into W. It clears the C bit because there is no carry out of bit 7, and clears the DC bit because there is no carry from bit 3 to bit 4. The Z bit is cleared because the result is nonzero.

ADDC fr,W — Add Carry,fr,W into fr

Operation: fr = C + fr + W

Bits affected: C, DC, Z

Opcode: `0101 111f ffff ffff`

Description: This instruction adds the contents of W and the C bit to the contents of the specified data memory location and writes the 8-bit result into the same data memory location. W is left unchanged. The register contents are treated as unsigned values.

If the result of addition exceeds 0xFF, the C bit is set and the lower eight bits of the result are written to the data memory location. Otherwise, the C bit is cleared.

If there is a carry from bit 3 to bit 4, the DC (digit carry) bit is set. Otherwise, the bit is cleared.

If the result of addition is zero, the Z bit is set. Otherwise, the Z bit is cleared. A sum of 0x100 is considered zero and therefore sets the Z bit.

Cycles: 1

Example: **`addc 0x099,w`**

This example adds the contents of W and the C bit to data memory location 0x099. For example, if the data memory location holds 0x7F, W holds 0x02, and the carry bit is set, this instruction adds 0x03 to 0x7F, writes the result 0x82 into the data memory location, and clears the C and Z bits. It sets the DC bit because of the carry from bit 3 to bit 4.

ADDC W,fr — Add Carry,W,fr into W

Operation: W = C + W + fr

Bits affected: C, DC, Z

Opcode: `0101 110f ffff ffff`

Description: This instruction adds the contents of the specified data memory location and the C bit to the contents of W and writes the 8-bit result into W. The data memory location is left unchanged. The register contents are treated as unsigned values.

If the result of addition exceeds 0xFF, the C bit is set and the lower eight bits of the result are written to W. Otherwise, the C bit is cleared.

If there is a carry from bit 3 to bit 4, the DC (digit carry) bit is set. Otherwise, the DC bit is cleared.

If the result of addition is zero, the Z bit is set. Otherwise, the Z bit is cleared. A sum of 0x100 is considered zero and therefore sets the Z bit.

Cycles: 1

Example: **`addc w,0x099`**

This example adds the contents of data memory location 0x099 and the C bit to W. If the data memory location holds 0x71, W holds 0x92, and the C bit is set, this instruction adds 0x72 to 0x92 and writes the lower eight bits of the result 0x04 into W. It sets the C bit because of the carry out of bit 7, clears the DC bit because there is no carry from bit 3 to bit 4, and clears the Z bit because the result is nonzero.

AND fr,W — AND fr,W into fr

Operation: fr = fr & W

Bits affected: Z

Opcode: `0001 011f ffff ffff`

Description: This instruction performs a bitwise logical AND of the contents of the specified data memory location and W, and writes the 8-bit result into the same data memory location. W is left unchanged. If the result is zero, the Z bit is set.

Cycles: 1

Example: **`and 0x099,W`**

This example performs a bitwise logical AND of the working register W with a value stored in data memory location 0x099. The result is written back to the data memory location.

For example, suppose that the data memory location 0x099 holds the value 0x0F and W holds the value 0x13. The instruction takes the logical AND of 0x0F and 0x13 and writes the result 0x03

back to the data memory location. The result is nonzero, so the Z bit is cleared.

AND W,fr — AND W,fr into W

Operation: W = W & fr

Bits affected: Z

Opcode: `0001 010f ffff ffff`

Description: This instruction performs a bitwise logical AND of the contents of W and the specified data memory location, and writes the 8-bit result into W. The data memory location is left unchanged. If the result is zero, the Z bit is set.

Cycles: 1

Example:

```
and   w,0x099
```

This example performs a bitwise logical AND of the value stored in data memory location 0x099 with W. The result is written back to W.

For example, suppose that the data memory location 0x099 holds the value 0x0F and W holds the value 0x13. The instruction takes the logical AND of 0x0F and 0x13 and writes the result 0x03 into W. The result is nonzero, so the Z bit is cleared.

AND W,#lit8 AND W,Literal into W

Operation: W = W & lit8

Bits affected: Z

Opcode: `1110 kkkk kkkk kkkk`

Description: This instruction performs a bitwise logical AND of the contents of W and an 8-bit literal value, and writes the 8-bit result into W. If the result is zero, the Z bit is set.

Cycles: 1

Example:

```
and   w,#0x0F
```

This example performs a bitwise logical AND of W with the literal value 0x0F. The result is written back to W.

For example, suppose that W holds the value 0x50. The instruction takes the logical AND of this value with 0x0F and writes the result 0x00 into W. The result is zero, so the Z bit is set.

BREAK — Enter Break Mode

Operation: See text below

Bits affected: None

Opcode: `0000 0000 0000 0001`

Description: This instruction enters the Break mode used for software debugging. On entry into Break mode, the CPU pipeline is flushed, and program execution stops with the CPU executing a continuous series of **nop** instructions. The **`break`** instruction is used to suspend program execution at a specified point, so that the contents of registers can be examined through the debugging interface. Break mode can be exited only by reset or debugging commands issued through the ISD/ISP interface. See Chapter 1 for more information about the debugging interface.

Cycles: 1

Example: **`break`**

Enter break mode. Not useful except when using a debugger.

CALL addr13 — Call Subroutine

Operation: top-of-stack = PC15:0 + 1
PC12:0 = addr13
PC15:13 = PA2:PA0

Bits affected: None

Opcode: `110k kkkk kkkk kkkk`

Description: This instruction calls a subroutine. The full 16-bit address of the next program instruction is saved on the stack and the program counter is loaded with a new address, which causes a jump to that program address.

Bits 12:0 come from the 13-bit constant value in the instruction, and bits 15:13 come from the PA2:PA0 bits in the STATUS register.

The subroutine is terminated by a **`ret`** instruction, which restores the saved address to the program counter. Execution proceeds from the instruction following the **`call`** instruction.

Cycles: 3

Example:

```
page  0x600 ;set page bits
call  addxy ;call subroutine addxy
nop   ;addxy results
      ;available here

...

addxy: ;subroutine address label
mov   w,#0F ;subroutine begins
add   w,0x099

...

ret   ;return from subroutine
```

The **call** instruction in this example calls a subroutine called **addxy**. When the **call** instruction is executed, the address of the following instruction (the **nop** instruction) is pushed onto the stack and the program jumps to the **addxy** routine. When the **ret** instruction is executed, the 16-bit program address saved on the stack is popped and restored to the program counter, which causes the program to continue with the instruction immediately following the **call** instruction.

CLR fr — Clear fr

Operation: fr = 0

Bits affected: Z

Opcode: `0000 011f ffff ffff`

Description: This instruction clears the specified data memory location. It also sets the Z bit unconditionally.

Cycles: 1

Example:

```
clr   0x099
```

This example clears data memory location 0x099 and sets the Z bit.

CLRB fr,bit — Clear Bit in fr

Operation: fr,bit = 0

Bits affected: none

Opcode: `0100 bbbf ffff ffff`

Description: This instruction clears a bit in the specified data memory location without changing the other bits in the register. The data memory location address and the bit number (0 through 7) are the instruction operands.

Cycles: 1

Example: **`clrb 0x099,7`**

This example clears the most significant bit of data memory location 0x099.

CMP W,fr — Compare W,fr

Operation: fr - W; result is discarded, STATUS is updated

Bits affected: C, DC, Z

Opcode: `0000 010f ffff ffff`

Description: This instruction subtracts the contents of W from the contents of the specified data memory location and discards the result. The data memory location and W are left unchanged. Only the STATUS register bits are updated.

If the result of subtraction is negative (W is larger than fr), the C bit is cleared and the lower eight bits of the result are written to W. Otherwise, the C bit is set.

If there is a borrow from bit 3 to bit 4, the DC (digit carry) bit is cleared. Otherwise, the bit is set.

If the result of subtraction is zero, the Z bit is set. Otherwise, the Z bit is cleared.

Cycles: 1

Example: **`cmp w,0x099`**

This example subtracts the contents of W from data memory location 0x099. For example, if the data memory location holds 0x35 and W holds 0x06, this instruction subtracts 0x06 from 0x35. It then sets the C bit, clears the DC bit, and clears the Z bit. The contents of the data memory location and W are left unchanged.

CMP W,#lit8 Compare W,Literal

Operation: lit8 - W; result is discarded, STATUS is updated

Bits affected: C, DC, Z

Opcode: `0111 1001 kkkk kkkk`

Description: This instruction subtracts the contents of W from an 8-bit literal and discards the result. W is left unchanged. Only the STATUS register bits are updated.

If the result of subtraction is negative (W is larger than lit8), the C bit is cleared and the lower eight bits of the result are written to W. Otherwise, the C bit is set.

If there is a borrow from bit 3 to bit 4, the DC (digit carry) bit is cleared. Otherwise, the bit is set.

If the result of subtraction is zero, the Z bit is set. Otherwise, the Z bit is cleared.

Cycles: 1

Example: **`cmp w,#0x0F`**

This example subtracts the contents of W from 0x0F. For example, if W holds 0x06, this instruction subtracts 0x06 from 0x0F. It then clears the C bit, clears the DC bit, and clears the Z bit. The contents of W are left unchanged.

CSE W,fr Compare W,fr then Skip if Equal

Operation: fr - W; if result is zero, skip next instruction

Bits affected: None

Opcode: `0100 001f ffff ffff`

Description: This instruction subtracts the contents of W from the specified data memory location and discards the result. The data memory location and W are left unchanged. If the result is zero (i.e. W and the literal are equal), the following instruction is skipped. The STATUS register bits are not updated.

Cycles: 1

Example:

```
cse   w,0x1F0;compare W with 0x1F0
ret         ;if not equal, return
nop         ;else, skip to here
```

This example compares the contents of W with the contents of data memory location 0x1F0. If W and the data memory location hold the same contents, then the **`ret`** instruction is executed, otherwise the **nop** instruction is executed.

CSE W,#lit8 Compare W,Literal then Skip if Equal

Operation: lit8 - W; if result is zero, skip next instruction

Bits affected: None

Opcode: `0111 0111 kkkk kkkk`

Description: This instruction subtracts the contents of W from an 8-bit literal and discards the result. W is left unchanged. If the result is zero (i.e. W and the literal are equal), the following instruction is skipped. The STATUS register bits are not updated.

Cycles: 1

Example:

```
cse   w,#0x1B ;compare W against
                ;escape char.
jmp   normal_char;if equal, jump
                ;to normal_char
jmp   escape_char;else, jump
                ;to escape_char
```

This example compares the contents of W with the ASCII code for the escape character. If W holds 0x1B, then the **`escape_char`** routine is

called, otherwise the **`normal_char`** routine is called.

CSNE W,fr Compare W,fr then Skip if Not Equal

Operation: fr - W; if result is nonzero, skip next instruction

Bits affected: None

Opcode: `0100 000f ffff ffff`

Description: This instruction subtracts the contents of W from the specified data memory location and discards the result. The data memory location and W are left unchanged. If the result is nonzero (i.e. W and the literal are not equal), the following instruction is skipped. The STATUS register bits are not updated.

Cycles: 1

Example:

```
csne  w,0x1F0;compare W against
                    ;register 0x1F0
ret         ;if equal, return
nop         ;else, skip to here
```

This example compares the contents of W with the contents of data memory location 0x1F0. If W and the data memory location hold the same contents, then the **`nop`** instruction is executed, otherwise the **`ret`** instruction is executed.

CSNE W,#lit8 Compare W,Literal then Skip if Not Equal

Operation: lit8 - W; if result is not zero, skip next instruction

Bits affected: None

Opcode: `0111 0110 kkkk kkkk`

Description: This instruction subtracts the contents of W from an 8-bit literal and discards the result. W is left unchanged. If the result is nonzero (i.e. W and the literal are not equal), the following instruction is skipped. The STATUS register bits are not updated.

Cycles: 1

Example:

```
csne  w,#0x1B;compare W against
                  ;escape char.
jmp   escape_char;if equal, jump
                  ;to escape_char
jmp   normal_char;else, jump to
                  ;normal_char
```

This example compares the contents of W with the ASCII code for the escape character. If W holds 0x1B, then the **`escape_char`** routine is

called, otherwise the **`normal_char`** routine is called.

CWDT Clear Watchdog Timer

Operation: Clears Watchdog timer counter and prescaler counter

Bits affected: Z

Opcode: `0000 011f ffff ffff`

Description: This instruction clears the Watchdog Timer counter to zero. It also clears the Watchdog prescaler.

If the Watchdog Timer is enabled, the application software must execute this instruction periodically in order to prevent a Watchdog reset.

Cycles: 1

Example: **`cwdt`**

This example clears the Watchdog Timer counter and the Watchdog Timer prescaler.

DEC fr — Decrement fr into fr

Operation: fr = fr - 1

Bits affected: Z

Opcode: `0000 111f ffff ffff`

Description: This instruction decrements the specified register file by one.

If the data memory location holds 0x01, it is decremented to 0x00 and the Z bit is set. Otherwise, the bit is cleared.

If the data memory location holds 0x00, it is decremented to 0xFF.

Cycles: 1

Example:
```
dec   0x099
```

This example decrements data memory location 0x099.

DEC W,fr — Decrement fr into W

Operation: W = fr - 1

Bits affected: Z

Opcode: `0000 110f ffff ffff`

Description: This instruction decrements the value in the specified register file by one and moves the 8-bit result into W. The data memory location is left unchanged.

If the data memory location holds 0x01, the value moved into W is 0x00 and the Z bit is set. Otherwise, the Z bit is cleared.

Cycles: 1

Example:
```
dec   w,0x099
```

This example decrements the value in the data memory location at 0x099 and moves the result into W. For example, if the data memory location holds 0x75, the value 0x74 is loaded into W, and the Z bit is cleared. The data memory location still holds 0x75 after execution of the instruction.

DECSNZ fr Decrement fr into fr then Skip if Not Zero

Operation: fr = fr - 1; if nonzero result, then skip next instruction

Bits affected: None

Opcode: `0100 111f ffff ffff`

Description: This instruction decrements the specified register file by one and tests the new register value. If that value is not zero, the next program instruction is skipped. Otherwise, execution proceeds normally with the next instruction.

Cycles: 1 if tested condition is false, 3 if tested condition is true

Example:

```
decsnz0x18
jmp   back1
mov   0x19,w
```

The **`decsnz`** instruction decrements data memory location 0x18. If the result is zero, execution proceeds normally with the **`jmp`** instruction to **`back1`**. If the result is nonzero, the **`jmp`** instruction is skipped, and the **`mov`** instruction is executed.

DECSNZ W,fr Decrement fr into W then Skip if Not Zero

Operation: W = fr - 1; if nonzero result, then skip next instruction

Bits affected: None

Opcode: `0100 110f ffff ffff`

Description: This instruction decrements the value in the specified data memory location and moves the result to W. The data memory location is left unchanged.

If the result is zero, the next instruction in the program is skipped. Otherwise, program execution proceeds normally with the next instruction.

Cycles: 1 if tested condition is false; 2 if tested condition is true

Example:

```
decsnzw,0x099;decrement 0x099 and
            ;load result into W
ret   ;return if result is 0
nop   ;otherwise continue here
```

This example takes the contents of data memory location 0x099, decrements that value, and moves the result to W. If the result is nonzero, the **`ret`** instruction is skipped and the **`nop`** instruction is executed. If the result is zero, the **`ret`** instruction is executed.

DECSZ fr Decrement fr into fr then Skip if Zero

Operation: fr = fr - 1; if result is 0, then skip next instruction

Bits affected: None

Opcode: `0010 111f ffff ffff`

Description: This instruction decrements the specified register file by one and tests the new register value. If that value is zero, the next program instruction is skipped. Otherwise, execution proceeds normally with the next instruction.

Cycles: 1 if tested condition is false, 3 if tested condition is true

Example:

```
decsz 0x18
jmp   back1
mov   0x19,w
```

The **`decsz`** instruction decrements data memory location 0x18. If the result is nonzero, execution proceeds normally with the **`jmp`** instruction. If the result is zero, the **`jmp`** instruction is skipped and the `mov` instruction is executed.

DECSZ W,fr Decrement fr into W then Skip if Zero

Operation: W = fr - 1; if result is 0, then skip next instruction

Bits affected: None

Opcode: `0010 110f ffff ffff`

Description: This instruction decrements the value in the specified data memory location and moves the result to W. The data memory location is left unchanged.

If the result is zero, the next instruction in the program is skipped. Otherwise, program execution proceeds normally with the next instruction.

Cycles: 1 if tested condition is false; 2 if tested condition is true

Example:

```
decsz w,0x099;decrement 0x099 and
              ;load result into W
ret           ;return if result is 0
nop           ;otherwise continue
              ;from here
```

This example takes the contents of data memory location 0x099, decrements that value, and moves the result to W. If the result is zero, the **ret** instruction is skipped and the **nop** instruction is executed. If the result is nonzero, the **ret** instruction is executed.

FERASE — Erase Flash Block

Operation: See text below

Bits affected: FBUSY bit in the XCFG register

Opcode: `0000 0000 0000 0011`

Description: This instruction erases a 512-byte (256-word) block of program flash memory. The ADDRH register specifies bits 15:8 of the byte address of the block. If the block is not in program flash memory, no operation is performed. The instruction is non-blocking (i.e. other instructions may execute before the erase operation is complete).

After executing this instruction, the FBUSY bit in the XCFG register is set. When the erase operation is complete, the FBUSY bit goes clear. This instruction must not be executed if the FBUSY bit is already set from a previous **`iread`**, **`fwrite`**, or **`ferase`** instruction. Program execution out of flash memory is not possible while the FBUSY bit is set, therefore this instruction can only be executed from program RAM. For more information, see Section 3.7.

Cycles: 1

Example:

```
mov   w,#0x01;load W with 0x01
mov   addrx,w;copy W to ADDRX
mov   w,#0x83;load W with 0x83
mov   addrh,w;copy W to ADDRH
ferase      ;erase block
```

This example erases the 512-byte (256-word) block from byte address 0x18300 to 0x183FF.

The erase operation does not complete and the flash memory is not accessible until the FBUSY bit goes clear after the **`ferase`** instruction is executed. Therefore, any subsequent code that uses the flash memory must either check the state of the FBUSY bit or wait a sufficient number of delay cycles before proceeding.

FREAD — Read from Flash Memory

Operation: DATAH || DATAL = (ADDRX || ADDRH || ADDRL)

Bits affected: FBUSY bit in the XCFG register

Opcode: `0000 0000 0001 1011`

Description: This instruction transfers data from program flash memory to data memory. The 24-bit ADDRX/ADDRH/ADDRL register specifies the address of a 16-bit word in program memory which is loaded into the DATAH/DATAL register. If the address is not in program flash memory, no operation is performed. The instruction is non-blocking (i.e. other instructions may execute before the read operation is complete).

After executing this instruction, the FBUSY bit in the XCFG register is set. When the read operation is complete, the FBUSY bit goes clear. This instruction must not be executed if the FBUSY bit is already set from a previous **`fread`**, **`fwrite`**, or **`ferase`** instruction. Program execution out of flash memory is not possible while the FBUSY bit is set, therefore this instruction can only be executed from program RAM. For more information, see Section 3.7.

Cycles: 1

Example:

```
mov   w,#0x01;load W with 0x01
mov   addrx,w;copy W to ADDRX
mov   w,#0x83;load W with 0x83
mov   addrh,w;copy W to ADDRH
mov   w,#0x80;load W with 0x80
mov   addrl,w;copy W to ADDRL
fread       ;read flash memory

...         ;wait for FBUSY = 0 or
...         ;delay for longer than
...         ;flash access time

mov   w,datal;move low byte to W
mov   0x1FE,w;copy W to 0x1FE
mov   w,datah;move high byte to W
mov   0x1FF,w;copy W to 0x1FF
```

This example reads the 16-bit data stored at byte address 0x18380 in program flash memory. First, ADDRH and ADDRL are loaded with 0x018380. After the **`fread`** instruction, the DATAH/DATAL register holds the data stored in program memory at byte address 0x18380. Then, the lower byte is loaded into data memory location 0x1FE and the upper byte is loaded into data memory location 0x1FF.

FWRITE — Write into Flash Memory

Operation: (ADDRX || ADDRH || ADDRL) = DATAH || DATAL

Bits affected: FBUSY bit in the XCFG register

Opcode: `0000 0000 0001 1010`

Description: This instruction writes a word of data to program flash memory from data memory. The 24-bit ADDRX/ADDRH/ADDRL register specifies the address of a 16-bit word in program memory which is loaded with the contents of the DATAH/DATAL register. If the address is not in program flash memory, no operation is performed. The instruction is non-blocking (i.e. other instructions may execute before the write operation is complete).

After executing this instruction, the FBUSY bit in the XCFG register is set. When the write operation is complete, the FBUSY bit goes clear. This instruction must not be executed if the FBUSY bit is already set from a previous **`fread`**, **`fwrite`**, or **`ferase`** instruction. Program execution out of flash memory is not possible while the FBUSY bit is set, therefore this

instruction can only be executed from program RAM. For more information, see Section 3.7.

Cycles: 1

Example:

```
mov   w,#0x01;load W with 0x01
mov   addrx,w;copy W to ADDRX
mov   w,#0x83;load W with 0x83
mov   addrh,w;copy W to ADDRH
mov   w,#0x80;load W with 0x80
mov   addrl,w;copy W to ADDRL
mov   w,0x1FE;copy 0x1FE to W
mov   datal,w;copy W to DATAL
mov   w,0x1FF;copy 0x1FF to W
mov   datah,w;copy W to DATAH
fwrite      ;write flash memory
```

This example writes the 16-bit data held in the DATAH/DATAL register to address 0x18380 in program flash memory. First, ADDRX/ADDRH/ADDRL is loaded with 0x018380. Then, DATAH/DATAL is loaded from 0x1FE and 0x1FF. Finally, the **`fwrite`** instruction loads program flash memory from DATAH/DATAL with the data that came from registers 0x1FE and 0x1FF.

The write operation does not complete and the flash memory is not accessible until the FBUSY bit goes clear after the **`fwrite`** instruction is executed. Therefore, any subsequent code that uses the flash memory must either check the state of the FBUSY bit or wait a sufficient number of delay cycles before proceeding.

INC fr — Increment fr into fr

Operation: fr = fr + 1

Bits affected: Z

Opcode: `0010 101f ffff ffff`

Description: This instruction increments the specified register file by one.

If the data memory location holds 0xFF and is incremented to 0x00, the Z bit is set. Otherwise, the Z bit is cleared.

Cycles: 1

Example:
```
inc   0x099
```

This example increments data memory location 0x099.

INC W,fr — Increment fr into W

Operation: W = fr + 1

Bits affected: Z

Opcode: `0010 100f ffff ffff`

Description: This instruction increments the value in the specified register file by one and moves the 8-bit result into W. The data memory location is left unchanged.

If the data memory location holds 0xFF, the value moved into W is 0x00 and the Z bit is set. Otherwise, the Z bit is cleared.

Cycles: 1

Example:
```
inc   w,0x099
```

This example increments the value at data memory location 0x099 and moves the result into W. For example, if the data memory location holds 0x75, the value 0x76 is loaded into W, and the Z bit is cleared. The data memory location still holds 0x75 after execution of the instruction.

INCSNZ fr Increment fr then Skip if Not Zero

Operation: fr = fr + 1; if result is nonzero, then skip next instruction

Bits affected: None

Opcode: `0101 101f ffff ffff`

Description: This instruction increments the specified register file by one and tests the new register value. If that value is nonzero, the next program instruction is skipped. Otherwise, execution proceeds normally with the next instruction.

Cycles: 1 if tested condition is false, 2 if tested condition is true

Example:

```
incsnz0x099
jmp   back1
mov   0x017,w
```

The **`incsnz`** instruction increments data memory location 0x099. If the result is zero, execution proceeds normally with the **`jmp`** instruction to **`back1`**. If the result is nonzero, the **`jmp`** instruction is skipped and the **`mov`** instruction is executed.

INCSNZ W,fr Increment fr into W then Skip if Not Zero

Operation: W = fr + 1; if result is nonzero, then skip next instruction

Bits affected: None

Opcode: `0101 100f ffff ffff`

Description: This instruction increments the value in the specified data memory location and moves the result to W. The data memory location is left unchanged.

If the result is nonzero, the next instruction in the program is skipped. Otherwise, program execution proceeds normally with the next instruction.

Cycles: 1 if tested condition is false; 2 if tested condition is true

Example:

```
incsnzw,0x099;load 0x099 + 1 to W
ret   ;return if 0x099 + 1 is 0
nop   ;otherwise continue here
```

This example takes the contents of data memory location 0x099, increments that value, and moves

the result to W. If the result is nonzero, the **`ret`** instruction is skipped and the **`nop`** instruction is executed. If the result is zero, the **`ret`** instruction is executed.

INCSZ fr Increment fr into fr then Skip if Zero

Operation: fr = fr + 1; if result is 0, then skip next instruction

Bits affected: None

Opcode: `0011 111f ffff ffff`

Description: This instruction increments the specified data memory location by one and tests the new register value. If that value is zero, the next program instruction is skipped. Otherwise, execution proceeds normally with the next instruction.

Cycles: 1 if tested condition is false, 2 if tested condition is true

Example:

```
incsz 0x099
jmp   back1
nop
```

The **incsz** instruction increments data memory location 0x099. If the result is nonzero, execution proceeds normally with the **jmp** instruction to **back1**. If the result is zero, the **jmp** instruction is skipped and the **nop** instruction is executed.

INCSZ W,fr Increment fr into W then Skip if Zero

Operation: W = fr + 1; if result is 0, then skip next instruction

Bits affected: None

Opcode: `0011 110f ffff ffff`

Description: This instruction increments the value in the specified data memory location and moves the result to W. The data memory location is left unchanged.

If the result is zero, the next instruction in the program is skipped. Otherwise, program execution proceeds normally with the next instruction.

Cycles: 1 if tested condition is false; 2 if tested condition is true

Example:

```
incsz w,0x099;load 0x099 + 1 to W
ret   ;return if 0x099 + 1 is 0
nop   ;otherwise continue here
```

This example takes the contents of data memory location 0x099, increments that value, and moves the result to W. If the result is zero, the **`ret`**

instruction is skipped and the **nop** instruction is executed. If the result is nonzero, the **ret** instruction is executed.

INT — Software Interrupt

Operation: See text below

Bits affected: None

Opcode: `0000 0000 0000 0110`

Description: This instruction raises an interrupt if the GIE bit is set and the INT_EN bit is clear. If the GIE bit is clear or the INT_EN bit is set, no operation is performed.

Cycles: 1

Example:
```
int   ;software interrupt
```

IREAD Read External/Program Memory

Operation: DATAH || DATAL = (ADDRX || ADDRH || ADDRL)

Bits affected: None

Opcode: `0000 0000 0001 1001`

Description: This instruction transfers data from external memory, program flash memory, or program RAM to data memory. The 24-bit ADDRX/ADDRH/ADDRL register specifies the address of a 16-bit word which is loaded into the DATAH/DATAL register. The instruction is blocking (i.e. no other instructions may execute until it completes) when it is used to read program RAM or when it is executed from program flash memory to read program flash memory. The instruction is non-blocking when it is used to read external memory or when it is executed from program RAM to read program flash memory.

Cycles: 4 (blocking)/1 (non-blocking)

Example:

```
mov   w,#0x00;load W with 0x00
mov   addrx,w;copy W to ADDRX
mov   w,#0x03;load W with 0x03
mov   addrh,w;copy W to ADDRH
mov   w,#0x80;load W with 0x80
mov   addrl,w;copy W to ADDRL
iread      ;read program memory
mov   w,datal;move low byte to W
mov   0x1FE,w;copy W to 0x1FE
mov   w,datah;move high byte to W
mov   0x1FF,w;copy W to 0x1FF
```

This example reads a word stored at byte address 0x000380 in program RAM. First, ADDRX/ADDRH/ADDRL is loaded with 0x000380. After the **iread** instruction, the DATAH/DATAL register holds the word read from program RAM. Then, the lower byte of the word is copied to data memory location 0x1FE and the upper byte is copied to data memory location 0x1FF.

IREADI Read Ext./Program Memory and Increment

Operation: DATAH || DATAL = (ADDRX || ADDRH || ADDRL)
ADDRL = ADDRL + 2

Bits affected: None

Opcode: `0000 0000 0001 1101`

Description: This instruction transfers data from external memory, program flash memory, or program RAM to data memory. The 24-bit ADDRX/ADDRH/ADDRL register specifies the address of a 16-bit word which is loaded into the DATAH/DATAL register, then the address is incremented by 2. The instruction is blocking (i.e. no other instructions may execute until it completes) when it is used to read program RAM or when it is executed from program flash memory to read program flash memory. The instruction is non-blocking when it is used to read external memory or when it is executed from program RAM to read program flash memory.

Cycles: 4 (blocking)/1 (non-blocking)

Example:

```
mov   w,#0x00;load W with 0x00
mov   addrx,w;copy W to ADDRX
mov   w,#0x03;load W with 0x03
mov   addrh,w;copy W to ADDRH
mov   w,#0x80;load W with 0x80
mov   addrl,w;copy W to ADDRL
iread      ;read program memory
mov   w,datal;move low byte to W
mov   0x1FE,w;copy W to 0x1FE
mov   w,datah;move high byte to W
mov   0x1FF,w;copy W to 0x1FF
```

This example reads a word stored at byte address 0x000380 in program RAM. First, ADDRX/ADDRH/ADDRL is loaded with 0x000380. After the **iread** instruction, the DATAH/DATAL register holds the word read from program RAM. Then, the lower byte of the word is copied to data memory location 0x1FE and the upper byte is copied to data memory location 0x1FF.

IWRITE Write External/Program RAM

Operation: (ADDRX || ADDRH || ADDRL) = DATAH || DATAL

Bits affected: None

Opcode: `0000 0000 0001 1000`

Description: This instruction transfers data from data memory to external RAM or program RAM. The 24-bit ADDRX/ADDRH/ADDRL register specifies the address of a 16-bit word which is loaded into the contents of the DATAH/DATAL register. If the address is not in external RAM or program RAM, no operation is performed. The instruction is blocking (i.e. no other instructions may execute until it completes) when it is used to write program RAM. The instruction is non-blocking when it is used to write external RAM.

Cycles: 4 (blocking)/1 (non-blocking)

Example:

```
mov   w,#0x00;load W with 0x00
mov   addrx,w;copy W to ADDRX
mov   w,#0x03;load W with 0x03
mov   addrh,w;copy W to ADDRH
mov   w,#0x80;load W with 0x80
mov   addrl,w;copy W to ADDRL
mov   w,0x1FE;copy 0x1FE to W
mov   datal,w;copy W to DATAL
mov   w,0x1FF;copy 0x1FF to W
mov   datah,w;copy W to DATAH
iwrite     ;write program RAM
```

This example writes the contents of the DATAH/DATAL register to address 0x000380 in program RAM. First, the ADDRX/ADDRH/ADDRL register is loaded with 0x000380. Then, the DATAH/DATAL register is loaded from 0x1FE and 0x1FF. Finally, the **`iwrite`** instruction loads program RAM from DATAH/DATAL.

IWRITEI Write Ext./Program RAM and Increment

Operation: (ADDRX || ADDRH || ADDRL) = DATAH || DATAL
ADDRL = ADDRL + 2

Bits affected: None

Opcode: `0000 0000 0001 1100`

Description: This instruction transfers data from data memory to external RAM or program RAM. The 24-bit ADDRX/ADDRH/ADDRL register specifies the address of a 16-bit word which is loaded with the contents of the DATAH/DATAL register, then the address is incremented by 2. If the address is not in external RAM or program RAM, no operation is performed. The instruction is blocking (i.e. no other instructions may execute until it completes) when it is used to write program RAM. The instruction is non-blocking when it is used to write external RAM.

Cycles: 4 (blocking)/1 (non-blocking)

Example:

```
mov   w,#0x00;load W with 0x00
mov   addrx,w;copy W to ADDRX
mov   w,#0x03;load W with 0x03
mov   addrh,w;copy W to ADDRH
mov   w,#0x80;load W with 0x80
mov   addrl,w;copy W to ADDRL
mov   w,0x1FE;copy 0x1FE to W
mov   datal,w;copy W to DATAL
mov   w,0x1FF;copy 0x1FF to W
mov   datah,w;copy W to DATAH
iwrite     ;write program RAM
```

This example writes the contents of the DATAH/DATAL register to address 0x000380 in program RAM. First, the ADDRX/ADDRH/ADDRL register is loaded with 0x000380. Then, the DATAH/DATAL register is loaded from 0x1FE and 0x1FF. Finally, the **`iwrite`** instruction loads program RAM from DATAH/DATAL.

JMP addr13 — Jump to Address

Operation: PC12:0 = addr13
PC15:13 = PA2:PA0

Bits affected: None

Opcode: `111k kkkk kkkk kkkk`

Description: This instruction causes the program to jump to a specified address. It loads the program counter with the new address. Bits 12:0 come from the 13-bit constant value in the instruction, and bits 15:13 come from the PA2:PA0 bits in the STATUS register. The STATUS register must hold the appropriate value prior to the jump instruction.

Cycles: 3

Example:

```
snb   STATUS,0;skip if carry clear
page  0x6000;set page bits
jmp   overflo;jump to overflo
nop   ;continue here if no jump

overflo:   ;jump destination
nop   ;continue here if jump taken
...
```

This example shows one way to implement a conditional jump. The **jmp** instruction, if executed, causes a jump to **overflo**. The **snb** instruction (test bit and skip if clear) causes the **jmp** instruction to be either executed or skipped, depending on the state of the carry bit (bit 0 of the STATUS register). Because the skip instruction is immediately followed by a **page** instruction, two instructions (**page** and **jmp**) will be skipped if the carry bit is clear.

The PA2:PA0 bits of the STATUS register must hold the three high-order bits (bits 15:13) of the word address of the **overflo** entry point prior to the **jmp** instruction. This is the purpose of the **page** instruction.

LOADH addr16 Load High Data Address into DPH

Operation: DPH = addr16(15:8)

Bits affected: None

Opcode: `0111 0000 kkkk kkkk`

Description: This instruction loads the high 8 bits of a data address into DPH. A skip over a **`loadh`** instruction results in an extended skip (following instruction also skipped).

From assembly language, a 16-bit address is specified. The assembler takes the high 8 bits of the address and encodes it in the instruction. The low 8 bits are ignored.

Cycles: 1

Example:

```
MyRegisters = 0x048A
      ;define symbolic address
loadh MyRegisters
      ;load 0x02 to DPH
loadl MyRegisters
      ;load 0x45 to DPL
```

The first line of this example is a declaration that the symbolic address **`MyRegisters`** is the byte address 0x048A (word address 0x0245). The second line loads DPH with the high 8 bits of the word address for **`MyRegisters`**. The third line loads DPL with the low 8 bits of the word address for **`MyRegisters`**.

LOADL addr16 Load Low Data Address into DPL

Operation: DPL = addr16(7:0)

Bits affected: None

Opcode: `0111 0001 kkkk kkkk`

Description: This instruction loads the low 8 bits of a data address into DPH. A skip over a **`loadl`** instruction results in an extended skip (following instruction also skipped).

From assembly language, a 16-bit address is specified. The assembler takes the low 8 bits of the address and encodes it in the instruction. The high 8 bits are ignored.

Cycles: 1

Example:

```
MyRegisters = 0x048A
      ;define symbolic address
loadh MyRegisters
      ;load 0x02 to DPH
loadl MyRegisters
      ;load 0x45 to DPL
```

The first line of this example is a declaration that the symbolic address **`MyRegisters`** is the byte address 0x048A (word address 0x0245). The second line loads DPH with the high 8 bits of the word address for **`MyRegisters`**. The third line loads DPL with the low 8 bits of the word address for **`MyRegisters`**.

MOV fr,W — Move W into fr

Operation: fr = W

Bits affected: None

Opcode: `0000 001f ffff ffff`

Description: This instruction moves the contents of W into the specified data memory location. W is left unchanged.

Cycles: 1

Example:
```
mov   0x099,w;move W to 0x099
```

This example moves the contents of W into data memory location 0x099.

MOV W,fr — Move fr into W

Operation: W = fr

Bits affected: Z

Opcode: `0010 000f ffff ffff`

Description: This instruction moves the contents of the specified data memory location into W. The data memory location is left unchanged.

If the data is zero, the Z bit is set. Otherwise, the Z bit is cleared.

Cycles: 1

Example:
```
mov  w,0x099;move register to W
```

This example moves the contents of the data memory location at address 0x099 into W. The Z bit is set if the value is zero or cleared if the value is nonzero.

MOV W,#lit8 Move Literal into W

Operation: W = lit8

Bits affected: None

Opcode: `0111 1100 kkkk kkkk`

Description: This instruction loads an 8-bit literal into W.

Cycles: 1

Example:
```
mov   w,#0x75
```

This example loads the literal 0x75 into W.

MULS W,fr Signed Multiply fr,W into MULH || W

Operation: W = fr × W; low 8 bits of signed product
MULH = fr × W; high 8 bits of signed product

Bits affected: None

Opcode: `0101 010f ffff ffff`

Description: This instruction performs a signed multiply of the contents of the specified data memory location by the contents of W. The low 8 bits of the product are written to W, and the high 8 bits are written to the MULH register. The data memory location is left unchanged.

Both operands are interpreted as two's-complement numbers, and the result loaded into MULH || W is also in two's complement format.

Cycles: 1

Example:

```
muls  w,0x099;signed multiply,
      ;0x099 by W into MULH and W
```

This example multiplies the contents of the data memory location at address 0x099 by the contents of W into W. The high byte of the result is loaded into the MULH register, and the low byte is loaded into W.

If W holds 0x07 and 0x099 holds 0x06, the result loaded into MULH || W will be 0x002A (42 decimal).

If W holds 0xF9 (-7 decimal) and 0x099 holds 0x06, the result loaded into MULH || W will be 0xFFD6 (-42 decimal).

If W holds 0xF9 (-7 decimal) and 0x099 holds 0xFA (-6 decimal), the result loaded into MULH || W will be 0x002A (42 decimal).

MULS W,#lit8 Signed Multiply W,Literal into MULH || W

Operation: W = W × lit8; low 8 bits of signed product
MULH = W × lit8; high 8 bits of signed product

Bits affected: None

Opcode: `0111 0011 kkkk kkkk`

Description: This instruction performs a signed multiply of an 8-bit literal by the contents of W. The low 8 bits of the product are written to W, and the high 8 bits are written to the MULH register. The data memory location is left unchanged.

Both operands are interpreted as two's-complement numbers, and the result loaded into MULH || W is also in two's complement format.

Cycles: 1

Example: **`muls w,#0x75`**

This example multiplies the contents of W by the literal 0x75 into W. The high byte of the result is loaded into the MULH register, and the low byte is loaded into W.

If W holds 0x07 and lit8 is 0x06, the result loaded into MULH || W will be 0x002A (42 decimal).

If W holds 0xF9 (-7 decimal) and lit8 is 0x06, the result loaded into MULH || W will be 0xFFD6 (-42 decimal).

If W holds 0xF9 (-7 decimal) and lit8 is 0xFA (-6 decimal), the result loaded into MULH || W will be 0x002A (42 decimal).

MULU W,fr Unsigned Multiply W,fr into MULH || W

Operation: W = W × fr; low 8 bits of unsigned product
MULH = W × fr; high 8 bits of unsigned product

Bits affected: None

Opcode: `0101 000f ffff ffff`

Description: This instruction performs an unsigned multiply of the contents of the specified data memory location by the contents of W. The low 8 bits of the product are written to W, and the high 8 bits are written to the MULH register. The data memory location is left unchanged.

Cycles: 1

Example:

```
mulu  w,0x099;unsigned multiply,
      ;0x099 by W into W
```

This example multiplies the contents of the data memory location at address 0x099 by the contents of W into W. The high byte of the result is loaded into the MULH register, and the low byte is loaded into W.

MULU W,#lit8 Unsigned Multiply W,Literal into MULH || W

Operation: W = W × lit8; low 8 bits of unsigned product
MULH = W × lit8; high 8 bits of unsigned product

Bits affected: None

Opcode: `0111 0010 kkkk kkkk`

Description: This instruction performs an unsigned multiply of an 8-bit literal by the contents of W. The low 8 bits of the product are written to W, and the high 8 bits are written to the MULH register. The data memory location is left unchanged.

Cycles: 1

Example:
```
mulu  w,#0x75
```

This example multiplies the contents of W by the literal 0x75 into W. The high byte of the result is loaded into the MULH register, and the low byte is loaded into W.

NOP — No Operation

Operation: None

Bits affected: None

Opcode: `0000 0000 0000 0000`

Description: This instruction does nothing except to cause a one-cycle delay in program execution.

Cycles: 1

Example:

```
sb    0x05,4;set bit 4 in Port A
nop   ;no operation, 1-cycle delay
sb    0x05,6;set bit 5 in Port A
```

This example shows how a **nop** instruction can be used as a one-cycle delay between two successive read-modify-write instructions that modify the same I/O port. This delay ensures reliable results at high clock rates.

NOT fr — Complement fr into fr

Operation: fr = $\overline{fr}$

Bits affected: Z

Opcode: `0010 011f ffff ffff`

Description: This instruction complements each bit of the specified data memory location and writes the result back into the same register. If the result is zero, the Z bit is set.

Cycles: 1

Example:
```
not   0x099 ;complement 0x099
```

Suppose that data memory location 0x099 holds the value 0x1C. This instruction takes the complement of 0x1C and writes the result 0xE3 back to location 0x099. The result is nonzero, so the Z bit is cleared.

NOT W,fr — Complement fr into W

Operation: $W = \overline{fr}$

Bits affected: Z

Opcode: `0010 010f ffff ffff`

Description: This instruction loads the one's complement of the specified data memory location into W. The data memory location is left unchanged.

If the value loaded into W is zero, the Z bit is set. Otherwise, the bit is cleared.

Cycles: 1

Example:
```
mov   w,0x099
```

This example moves the one's complement of data memory location 0x099 into W. For example, if the data memory location holds 0x75, the complement of this value 0x8A is loaded into W, and the Z bit is cleared. The data memory location is left unchanged.

OR fr,W — OR fr,W into fr

Operation: fr = fr | W

Bits affected: Z

Opcode: `0001 001f ffff ffff`

Description: This instruction performs a bitwise logical OR of the contents of the specified data memory location and W, and writes the 8-bit result into the same data memory location. W is left unchanged. If the result is zero, the Z bit is set.

Cycles: 1

Example: `or 0x099,w;move fr OR W to fr`

This example performs a bitwise logical OR of W with a value stored in data memory location 0x099. The result is written back to the data memory location 0x099.

For example, suppose that the data memory location 0x099 holds the value 0x0F and W holds the value 0x13. The instruction takes the logical OR of 0x0F and 0x13 and writes the result 0x1F back to data memory location 0x099. The result is nonzero, so the Z bit is cleared.

OR W,fr — OR W,fr into W

Operation: W = W | fr

Bits affected: Z

Opcode: `0001 000f ffff ffff`

Description: This instruction performs a bitwise logical OR of the contents of W and the specified data memory location, and writes the 8-bit result into W. The data memory location is left unchanged. If the result is zero, the Z bit is set.

Cycles: 1

Example:
```
or    w,0x099;move W OR fr to W
```

This example performs a bitwise logical OR of the value stored in data memory location 0x099 with W. The result is written back to W.

For example, suppose that the data memory location 0x099 holds the value 0x0F and W holds the value 0x13. The instruction takes the logical OR of 0x0F and 0x13 and writes the result 0x1F into W. The result is nonzero, so the Z bit is cleared.

OR W,#lit8 OR W,Literal into W

Operation: W = W | lit8

Bits affected: Z

Opcode: `1101 kkkk kkkk kkkk`

Description: This instruction performs a bitwise logical OR of the contents of W and an 8-bit literal value, and writes the 8-bit result into W. If the result is zero, the Z bit is set.

Cycles: 1

Example:
```
or    w,#0x0F;set low four W bits
```

This example performs a bitwise logical OR of W with the literal value 0x0F. The result is written back to W.

For example, suppose that W holds the value 0x50. The instruction takes the logical OR of this value with 0x0F and writes the result 0x5F into W. The result is nonzero, so the Z bit is cleared.

PAGE addr16 — Load Page Bits

Operation: PA2:0 = addr(16:14)

Bits affected: None

Opcode: `0000 0000 0001 0nnn`

Description: This instruction writes a three-bit value into the PA2:0 bits of the STATUS register (bits 7:5). These bits select the program memory page for subsequent jump and subroutine call instructions.

In assembly language, the full program memory address is specified. The assembler encodes the three high-order bits of this address into the instruction opcode and ignores the fourteen low-order bits.

If a skip instruction is immediately followed by a `page` instruction and the tested condition is true, then two instructions are skipped and the operation consumes three cycles. This is useful for conditional branching to another page in which a **`page`** instruction precedes a **`jmp`** instruction. If several **`page`** instructions immediately follow a skip instruction then they are all skipped plus the

next instruction, and a cycle is consumed for each.

Cycles: 1

Example:

```
page  0x8000;load PA2:PA0 with 010
call  home1      ;jump to page 2
```

This example sets the PA2:0 bits in the STATUS register to 010. This means that the subsequent **`call`** instruction calls a subroutine that starts in the address range of byte address 0x8000 to 0x9FFE (word address 0x4000 to 0x4FFF) .

POP fr — Move Top of Stack into fr

Operation: SP = SP + 1;
fr = (SP)

Bits affected: None

Opcode: `0100 011f ffff ffff`

Description: This instruction increments the SPH/SPL register, then copies the register addressed by the SPH/SPL register to the specified data memory location. This stack is independent of the hardware stack used for subroutine calls and returns.

Cycles: 1

Example:
```
pop   0x1F0 ;pop stack to 0x1F0
```

The **pop** instruction in this example pops a byte off the stack and loads it into the data memory location at address 0x1F0.

PUSH fr — Move fr onto Top of Stack

Operation: (SP) = fr
SP = SP - 1

Bits affected: None

Opcode: `0100 010f ffff ffff`

Description: This instruction copies the specified data memory location to the register addressed by the SPH/SPL register, then decrements the SPH/SPL register. This stack is independent of the hardware stack used for subroutine calls and returns.

Cycles: 1

Example:
```
push  0x1F0 ;push 0x1F0 onto stack
```

The **push** instruction in this example pushes the contents of the data memory location at address 0x1F0 onto the stack.

PUSH #lit8 Move Literal onto Top of Stack

Operation: (SP) = lit8
SP = SP - 1

Bits affected: None

Opcode: `0111 0100 kkkk kkkk`

Description: This instruction copies an 8-bit literal to the register addressed by the SPH/SPL register, then decrements the SPH/SPL register. This stack is independent of the stack used for subroutine calls and returns.

Cycles: 1

Example:
```
push  0x#FF ;push #0xFF onto stack
```

The **push** instruction in this example pushes 0xFF onto the stack.

RET — Return from Subroutine

Operation: program counter = top-of-stack

Bits affected: None

Opcode: `0000 0000 0000 0111`

Description: This instruction causes a return from a subroutine. It pops the 16-bit value previously stored on the stack and restores that value to the program counter. This causes the program to jump to the instruction immediately following the **`call`** instruction that called the subroutine. The hardware stack used for subroutine call and return is independent from the stack used with the **`push`** and **`pop`** instructions.

It is not necessary to set the PA2:0 bits in the STATUS register to return to the correct place in the program. This is because the full 16-bit program address is restored from the stack. The **`ret`** instruction does not use (and does not affect) the PA2:0 bits.

Cycles: 3

Example:

```
page  0x0000;set page bits
calladdxy  ;call subroutine addxy
nop;addxy results available here

...

addxy:;subroutine entry point
mov   w,0x099;subroutine start
add   w,0x00F
...
ret   ;return from subroutine
```

The **call** instruction in this example calls a subroutine called **addxy**. When the **call** instruction is executed, the address of the following instruction (the **nop** instruction) is pushed onto the stack and the program jumps to the **addxy** routine. When the **ret** instruction is executed, the saved program address is popped from the stack and restored to the program counter, which causes the program to continue with the instruction immediately following the **call** instruction.

RETI #lit3 Return from Interrupt

Operation: restore CPU registers from shadow registers

Bits affected: STATUS register restored, which affects all bits

Opcode: `0000 0000 0000 1nnn`

Description: This instruction causes a return from an interrupt service routine. It restores the 16-bit program counter value that was saved when the interrupt occurred. In addition, there are three instruction options encoded by the three low-order bits of the instruction, as shown in the table below.

Bit	Function
2	Reinstate the pre-interrupt speed 1 = enable, 0 = disable
1	Store the PC+1 value in the INTVECH and INTVECL registers 1 = enable, 0 = disable
0	Add W to the T0TMR register 1 = enable, 0 = disable

Cycles: 3

Example:

```
org   0     ;set address to 0x000
...         ;ISR code goes here
reti  #0x4  ;return from interrupt
```

This is an example of an interrupt service routine. When an interrupt occurs, the CPU registers are saved into a set of shadow registers. The program then jumps to the interrupt service routine, which starts at a default address of 0x000 (software can change this address by loading a new interrupt vector into the INTVECH and INTVECL register pair). The interrupt service routine should determine the cause of the interrupt, clear the interrupt flag bit for the event that raised the interrupt, perform any required service for that event, and end with the **`reti`** instruction.

The **`reti`** instruction restores the contents of the program counter from the shadow registers. This causes the IP2022 to continue program execution from the point at which the program was interrupted.

In this example, the immediate operand specifies restoration of the pre-interrupt speed, but no modification is made to the interrupt vector or the T0TMR register.

RETW #lit8 Return from Subroutine with Literal into W

Operation: W = lit8
program counter = top-of-stack

Bits affected: None

Opcode: `0111 1000 kkkk kkkk`

Description: This instruction loads an 8-bit literal into W and causes a return from a subroutine. The literal can be used to implement lookup tables. The instruction pops the 16-bit value previously stored on the stack and restores that value to the program counter. This causes the program to jump to the instruction immediately following the **`call`** instruction that called the subroutine. The hardware stack used for subroutine call and return is independent from the stack used with the **`push`** and **`pop`** instructions.

It is not necessary to set the PA2:0 bits in the STATUS register to return to the correct place in the program. This is because the full 16-bit program address is restored from the stack. The **`ret`** instruction does not use (and does not affect) the PA2:0 bits.

Cycles: 3

Example:

```
mov   w,0x1F0;load W with index
add   pcl,w ;add index to the pc
retw  #0xFF ;if index is 0,
            ;return with 0xFF in W
retw  #0xFF ;if index is 1,
            ;return with 0xF0 in W
retw  #0xFF ;if index is 2,
            ;return with 0x0F in W
retw  #0xFF ;if index is 3,
            ;return with 0x00 in W
```

This example shows an index being read from data memory location 0x1F0 and being added to the program counter, which causes a jump into an array of **`retw`** instructions. Depending on what the index is, one of the **`retw`** instructions will be executed. Each **`retw`** instruction returns a different value in W.

RL fr Rotate fr Left through Carry into fr

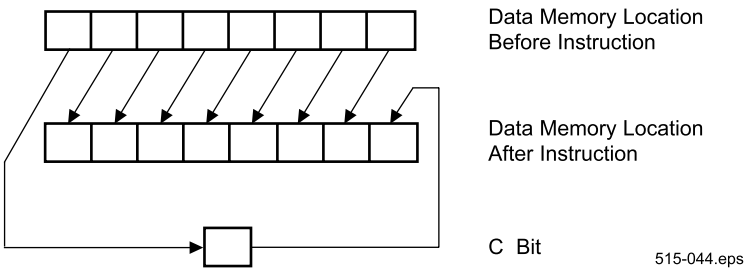
Operation: fr || C = C || fr

Bits affected: C

Opcode: `0011 011f ffff ffff`

Description: This instruction rotates the bits of the specified data memory location left through the C bit and moves the 8-bit result into the data memory location.

The bits obtained from the register are shifted left by one bit position. C is shifted into the least significant bit position and the most significant bit is shifted out into C, as shown in the diagram below.



Cycles: 1

Example: **`rl 0x099`**

This example rotates the bits of data memory location 0x099 left through the C bit. If the data memory location holds 0x14 and the C bit is set, after this instruction is executed, the data memory location will hold 0x29 and the C bit will be clear.

RL W,fr Rotate fr Left through Carry into W

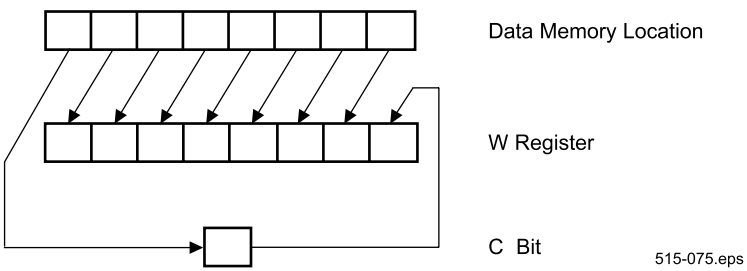
Operation: W || C = C || fr

Bits affected: C

Opcode: `0011 010f ffff ffff`

Description: This instruction rotates the bits of the specified data memory location left through the C bit and moves the 8-bit result into W. The data memory location is left unchanged.

The bits obtained from the register are shifted left by one bit position. C is shifted into the least significant bit position and the most significant bit is shifted out into C, as shown in the diagram below.



Cycles: 1

Example: `rl w,0x099`

This example rotates the bits of data memory location 0x099 left through the C bit and moves the result into W. If the data memory location holds 0x14 and the C bit is set, after this instruction is executed, W will hold 0x29 and the C bit will be clear. The data memory location will still hold 0x14 after execution of the instruction.

RR fr Rotate fr Right through Carry into fr

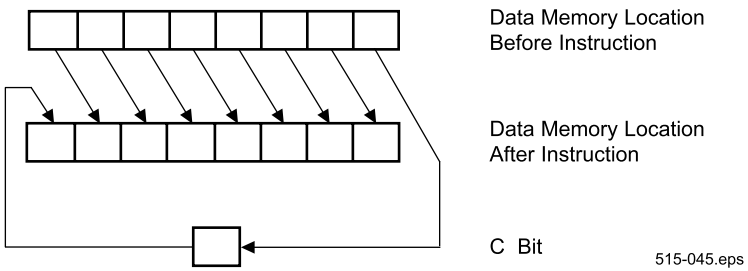
Operation: C || fr = fr || C

Bits affected: C

Opcode: `0011 001f ffff ffff`

Description: This instruction rotates the bits of the specified data memory location right through the C bit and moves the 8-bit result into the data memory location.

The bits obtained from the register are shifted right by one bit position. C is shifted into the most significant bit position and the least significant bit is shifted out into C, as shown in the diagram below.



Cycles: 1

Example: **`rr 0x099`**

This example rotates the bits of data memory location 0x099 right through the C bit. If the data memory location holds 0x12 and the C bit is set, after this instruction is executed, the data memory location will hold 0x89 and the C bit will be clear.

RR W,fr Rotate fr Right through Carry into W

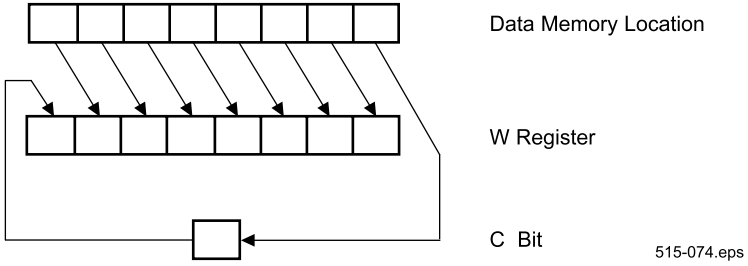
Operation: W = >> fr

Bits affected: C

Opcode: `0011 000f ffff ffff`

Description: This instruction rotates the bits of the specified data memory location right through the C bit and moves the 8-bit result into W. The data memory location is left unchanged.

The bits obtained from the register are shifted right by one bit position. C is shifted into the most significant bit position and the least significant bit is shifted out into C, as shown in the diagram below.



Cycles: 1

Example: **`mov w,0x099`**

This example rotates the bits of data memory location 0x099 right through the C bit and moves the result into W. If the data memory location holds 0x12 and the C bit is set, after this instruction is executed, W will hold 0x89 and the C bit will be clear. The data memory location will still hold 0x12 after execution of the instruction.

SB fr.bit Test Bit in fr then Skip if Set

Operation: if fr.bit = 1, skip next instruction

Bits affected: None

Opcode: `0111 bbbf ffff ffff`

Description: This instruction tests a bit in the specified data memory location. The data memory location and the bit number (0 through 7) are the instruction operands. If the bit is 1, the next instruction in the program is skipped. Otherwise, program execution proceeds normally with the next instruction.

Cycles: 1 if tested condition is false, 2 if tested condition is true

Example:

```
sb    STATUS,0 ;test carry bit
inc   0x099 ;increment if carry=0
nop
```

This example tests the carry bit (bit 0 of the STATUS register). If the bit is 1, the **`inc`** instruction is skipped and the **`nop`** instruction is executed. Otherwise, program execution proceeds normally with the **`inc`** instruction.

SETB fr,bit — Set Bit in fr

Operation: fr.bit = 1

Bits affected: None

Opcode: `0101 bbbf ffff ffff`

Description: This instruction sets a bit in the specified data memory location to 1 without changing the other bits in the register. The data memory location and the bit number (0 through 7) are the instruction operands.

Cycles: 1

Example: **`setb 0x099,7`**

This example sets the most significant bit of data memory location 0x099.

SNB fr,bit Test Bit in fr then Skip if Clear

Operation: if fr.bit = 0, skip next instruction

Bits affected: None

Opcode: `0110 bbbf ffff ffff`

Description: This instruction tests a bit in the specified data memory location. The data memory location and the bit number (0 through 7) are the instruction operands. If the bit is 0, the next instruction in the program is skipped. Otherwise, program execution proceeds normally with the next instruction.

Cycles: 1 if tested condition is false, 2 if tested condition is true

Example:

```
snb   STATUS,0;test carry bit
inc   0x099 ;increment if carry=1
nop
```

This example tests the carry bit (bit 0 of the STATUS register). If that bit is 0, the **`dec`** instruction is skipped and the **`nop`** instruction is executed. Otherwise, program execution proceeds normally with the **`inc`** instruction.

SPEED #lit8 — Change CPU Speed

Operation: SPDREG = lit8

Bits affected: None

Opcode: `0000 0001 nnnn nnnn`

Description: This instruction writes an 8-bit value into the SPDREG register. This register controls the power-down options, system clock source, and clock divisor used to generate the CPU core clock from the system clock. The format of the SPDREG register is shown below.

7 6	5 4	3 0
PWRD1:0	CLK1:0	CDIV3:0

PWRD1:0—controls whether the PLL clock multiplier and OSC oscillator are running.
CLK1:0—selects the system clock source.
CDIV3:0—selects the clock divisor used to generate the system clock.

For a more detailed description of these fields, see Section 2.1.1.

Cycles: 1 instruction cycle (as opposed to clock cycles)

Example:

```
nop   ;assume divisor is 4, so
      ;instruction takes 4 cycles
speed div8  ;change divisor to 8,
      ;instruction takes 4 cycles
nop   ;instruction takes 8 cycles
speed div1  ;change divisor to 1,
      ;instruction takes 8 cycles
nop   ;instruction takes 1 cycle
```

In this example, **div1** and **div8** are assumed to be constants defined with appropriate bit settings for the SPDREG register encoding.

If the clock divisor prior to the first **speed** instruction is 4, the first **nop** and **speed** instructions each take 4 clock cycles.

The first **speed** instruction changes the clock divisor to 8, so the second **nop** and **speed** instructions each take 8 clock cycles.

The second **speed** instruction changes the clock divisor to 1, so the third **nop** instruction takes 1 clock cycle.

SUB fr,W — Subtract W from fr into fr

Operation: fr = fr - W

Bits affected: C, DC, Z

Opcode: `0000 101f ffff ffff`

Description: This instruction subtracts the contents of W from the contents of the specified data memory location and writes the 8-bit result into the same data memory location. W is left unchanged. The register contents are treated as unsigned values.

If the result of subtraction is negative (W is larger than fr), the C bit is cleared and the lower eight bits of the result are written to the data memory location. Otherwise, the C bit is set.

If there is a borrow from bit 3 to bit 4, the DC (digit carry) bit is cleared. Otherwise, the bit is set.

If the result of subtraction is zero, the Z bit is set. Otherwise, the bit is cleared.

Cycles: 1

Example 1:

```
sub   0x099,w
```

This example subtracts the contents of W from data memory location 0x099. For example, if the data memory location holds 0x35 and W holds 0x06, this instruction subtracts 0x06 from 0x35 and writes the result 0x2F into the data memory location. It also sets the C bit, clears the DC bit, and clears the Z bit.

Example 2:

```
mov   w,0x095;load W from 0x095
sub   0x097,w;subtract low bytes
            ;C = 0 for borrow out
mov   w,0x096;load W from 0x096
subc  0x098,w;subtract high bytes
```

This example performs 16-bit subtraction of data memory locations 0x095 (low byte) and 0x096 (high byte) from data memory locations 0x097 (low byte) and 0x098 (high byte).

The **sub** instruction subtracts the contents of 0x095 from 0x097 and clears the C bit if a borrow occurs out of bit 7, or sets the C bit otherwise. The **subc** instruction subtracts the contents of 0x096 from 0x098 with borrow-in using the C bit.

SUB W,fr — Subtract W from fr into W

Operation: W = fr - W

Bits affected: C, DC, Z

Opcode: `0000 100f ffff ffff`

Description: This instruction subtracts the contents of W from the contents of the specified data memory location and writes the 8-bit result into W. The data memory location is left unchanged. The register contents are treated as unsigned values.

If the result of subtraction is negative (W is larger than fr), the C bit is cleared and the lower eight bits of the result are written to W. Otherwise, the C bit is set.

If there is a borrow from bit 3 to bit 4, the DC (digit carry) bit is cleared. Otherwise, the bit is set.

If the result of subtraction is zero, the Z bit is set. Otherwise, the Z bit is cleared.

Cycles: 1

Example: **`sub w,0x099`**

This example subtracts the contents of W from data memory location 0x099 and moves the result into W. For example, if the data memory location holds 0x35 and W holds 0x06, this instruction subtracts 0x06 from 0x35 and writes the result 0x2F into W. It also sets the C bit, clears the DC bit, and clears the Z bit. The data memory location is left unchanged.

SUB W,#lit8 Subtract W from Literal into W

Operation: W = lit8 - W

Bits affected: C, DC, Z

Opcode: `0111 1010 kkkk kkkk`

Description: This instruction subtracts the contents of W from an 8-bit literal and writes the 8-bit result into W. The register contents are treated as unsigned values.

If the result of subtraction is negative (W is larger than lit8), the C bit is cleared and the lower eight bits of the result are written to W. Otherwise, the C bit is set.

If there is a borrow from bit 3 to bit 4, the DC (digit carry) bit is cleared. Otherwise, the bit is set.

If the result of subtraction is zero, the Z bit is set. Otherwise, the bit is cleared.

Cycles: 1

Example: `sub w,#0xFF`

This example subtracts the contents of W from 0xFF. For example, if W holds 0x06, this instruction subtracts 0x06 from 0xFF and writes the result 0xF9 into W. It also sets the C bit, clears the DC bit, and clears the Z bit.

SUBC fr,W Subtract $\overline{\text{Carry}}$,W from fr into fr

Operation: fr = fr - $\overline{C}$ - W

Bits affected: C, DC, Z

Opcode: `0100 101f ffff ffff`

Description: This instruction subtracts the contents of W and the complement of the C bit (which indicates borrow) from the contents of the specified data memory location and writes the 8-bit result into the same data memory location. W is left unchanged. The register contents are treated as unsigned values.

If the result of subtraction is negative (W + $\overline{C}$ is larger than fr), the C bit is cleared and the lower eight bits of the result are written to the data memory location. Otherwise, the C bit is set.

If there is a borrow from bit 3 to bit 4, the DC (digit carry) bit is cleared. Otherwise, the bit is set.

If the result of subtraction is zero, the Z bit is set. Otherwise, the bit is cleared.

Cycles: 1

Example 1:

```
subc  0x099,w
```

This example subtracts the contents of W and the complement of the C bit from data memory location 0x099. For example, if the data memory location holds 0x35, W holds 0x06, and the C bit is clear, this instruction subtracts 0x07 from 0x35 and writes the result 0x2E into the data memory location. It also sets the C bit, clears the DC bit, and clears the Z bit.

Example 2:

```
mov   w,0x095;load W from 0x095
sub   0x097,w;subtract low bytes
            ;C = 0 for borrow out
mov   w,0x096;load W from 0x096
subc  0x098,w;subtract high bytes
```

This example performs 16-bit subtraction of data memory locations 0x095 (low byte) and 0x096 (high byte) from data memory locations 0x097 (low byte) and 0x098 (high byte).

The **sub** instruction subtracts the contents of 0x095 from 0x097 and clears the C bit if a borrow occurs out of bit 7, or sets the C bit otherwise. The **subc** instruction subtracts the contents of 0x096 from 0x097 with borrow-in using the C bit.

SUBC W,fr Subtract $\overline{\text{Carry}}$,W from fr into W

Operation: $W = fr - \overline{C} - W$

Bits affected: C, DC, Z

Opcode: `0100 100f ffff ffff`

Description: This instruction subtracts the contents of W and the complement of the C bit (which indicates borrow) from the contents of the specified data memory location and writes the 8-bit result into W. The data memory location is left unchanged. The register contents are treated as unsigned values.

If the result of subtraction is negative (W is larger than fr), the C bit is cleared and the lower eight bits of the result are written to W. Otherwise, the C bit is set.

If there is a borrow from bit 3 to bit 4, the DC (digit carry) bit is cleared. Otherwise, the bit is set.

If the result of subtraction is zero, the Z bit is set. Otherwise, the Z bit is cleared.

Cycles: 1

Example: **`subc w,0x099`**

This example subtracts the contents of W from data memory location 0x099 and moves the result into W. For example, if the data memory location holds 0x35, W holds 0x06, and the C bit is clear, this instruction subtracts 0x07 from 0x35 and writes the result 0x2E into W. It also sets the C bit, clears the DC bit, and clears the Z bit. The data memory location is left unchanged.

SWAP fr Swap High,Low Nibbles of fr into fr

Operation: fr = fr3:0 || fr7:4

Bits affected: None

Opcode: `0011 101f ffff ffff`

Description: This instruction exchanges the high-order and low-order nibbles (4-bit fields) of the specified data memory location.

Cycles: 1

Example: **`swap 0x099`**

This example swaps the high-order and low-order nibbles of data memory location 0x099. For example, if the memory location holds 0xA5, after executing this instruction, it will hold 0x5A.

SWAP W,fr Swap High,Low Nibbles of fr into W

Operation: W = fr3:0 || fr7:4

Bits affected: None

Opcode: `0011 100f ffff ffff`

Description: This instruction exchanges the high-order and low-order nibbles (4-bit fields) of the value in the specified data memory location and moves the result to W. The data memory location is left unchanged.

Cycles: 1

Example:

```
swap  W,0x099
```

This example swaps the high-order and low-order nibbles of the value in data memory location 0x099 and moves the result into W. For example, if the data memory location holds 0xA5, after executing this instruction, W will hold 0x5A.

TEST fr — Test fr for Zero

Operation: if fr = 0, Z = 1
else Z = 0

Bits affected: Z

Opcode: `0010 001f ffff ffff`

Description: This instruction moves the contents of the specified data memory location into the same register. There is no net effect except to set or clear the Z bit. If the register holds zero, the bit is set. Otherwise, the bit is cleared. If the **`test`** instruction is performed on the T0TMR register, the Timer 0 prescaler is initialized to zero. If the prescaler is about to expire causing Timer 0 to increment and the **`test`** instruction is executed, Timer 0 will not increment.

Cycles: 1

Example:

```
test  0x099      ;test 0x099
sb    STATUS,2   ;skip if Z=1
inc   w          ;increment W
nop
```

This example tests the contents of data memory location 0x01B. The **`test`** instruction sets or clears the Z bit based on the contents of the data memory location. The **`sb`** instruction tests the Z bit and skips to the **`nop`** instruction if the Z bit is set. The **`inc`** instruction is executed only if the data memory location is nonzero.

XOR fr,W — XOR fr,W into fr

Operation: fr = fr ^ W

Bits affected: Z

Opcode: `0001 101f ffff ffff`

Description: This instruction performs a bitwise exclusive OR of the contents of the specified data memory location and W, and writes the 8-bit result into the same data memory location. W is left unchanged. If the result is zero, the Z bit is set.

Cycles: 1

Example:
```
xor  0x099,w ;move fr XOR W to fr
```

This example performs a bitwise logical XOR of W with a value stored in data memory location 0x099. The result is written back to the data memory location 0x099.

For example, suppose that the data memory location 0x099 is holds the value 0x0F and W holds the value 0x13. The instruction takes the logical XOR of 0x0F and 0x13 and writes the

result 0x1C back to the data memory location. The result is nonzero, so the Z bit is cleared.

XOR W,fr — XOR W,fr into W

Operation: W = W ^ fr

Bits affected: Z

Opcode: `0001 100f ffff ffff`

Description: This instruction performs a bitwise exclusive OR of the contents of W and the specified data memory location, and writes the 8-bit result into W. The data memory location is left unchanged. If the result is 0x00, the Z bit is set.

Cycles: 1

Example:
```
xor  w,0x099 ;move W XOR fr to W
```

This example performs a bitwise logical XOR of the value stored in data memory location 0x099 with W. The result is written back to W.

For example, suppose that the data memory location 0x099 holds the value 0x0F and W holds the value 0x13. The instruction takes the logical XOR of 0x0F and 0x13 and writes the result 0x1C into W. The result is nonzero, so the Z bit is cleared.

XOR W,#lit8 XOR W,Literal into W

Operation: W = W ^ lit8

Bits affected: Z

Opcode: `0111 1111 kkkk kkkk`

Description: This instruction performs a bitwise exclusive OR of the contents of W and an 8-bit literal value, and writes the 8-bit result into W. If the result is 0x00, the Z bit is set.

Cycles: 1

Example:
```
xor   w,0x#0F ;complement W3:0
```

This example performs a bitwise logical XOR of W with the literal value 0x0F. The result is written back to W.

For example, suppose that W holds the value 0x51. The instruction takes the logical XOR of this value with 0x0F and writes the result 0x5E into W. The result is nonzero, so the Z bit is cleared.

4.0 Peripherals

The IP2022 provides an array of on-chip peripherals needed to support a broad range of embedded Internet applications:

- Watchdog Timer
- Real-Time Timer
- 2 Multifunction Timers with Compare and Capture Registers
- 2 Serializer/Deserializer (SERDES) Channels
- 8-Channel, 10-Bit A/D Converter
- Analog Comparator
- Parallel Slave Peripheral Interface
- Linear Feedback Shift Register

All of the peripherals except the Watchdog Timer and the Real-Time Timer use alternate functions of the I/O port pins to interface with external signals.

4.1 I/O Ports

The IP2022 contains one 4-bit I/O port (Port A) and six 8-bit I/O ports (Port B through Port G). The four Port A pins have 24 mA current drive capability. All the ports have symmetrical drive. Inputs are 5V-tolerant CMOS levels. Outputs can use the same 2.3–2.7V power supply used for the CPU core and peripheral logic, or they can use a higher voltage (up to 3.6V). The IOVdd

pins are provided for those applications in which a separate power supply is used for the I/O port pin output drivers. Port G has a separate GVdd pin which can be used to run the Port G output drivers at a voltage different from that used for the other ports, however Port G must run from a 2.3–2.7V power supply.

Each port has separate input, output, and configuration registers, which are memory mapped. The numbers in the pin names correspond to the bit positions in these registers. These registers allow each port bit to be individually configured as a general-purpose input or output under software control. Unused pins should be configured as outputs, to prevent them from floating. Port B has three additional registers for supporting external interrupts (see Section 4.1.1).

In addition, each port pin has an alternate function used to support the on-chip hardware peripherals, as listed in Table 4-1. Port A and Port B support the multi-function timers Timer 1 and Timer 2. Port B, Port C, and Port D support the Parallel Slave Peripheral (PSP) and external memory functions. Port E and Port F support the serializer/deserializer (SERDES) units. Port G supports the analog to digital converter (ADC) and the analog comparator. Before enabling a hardware peripheral, configure the port pins for input or output as required by the peripheral.

Table 4-1 I/O Port Pin Alternate Functions

Name	Pin	Alternate Function
RA0	5	High Power Output, Timer 1 Capture 1 Input (T1CPI1 pin)
RA1	6	High Power Output, Timer 1 Capture 2 Input (T1CPI2 pin)
RA2	7	High Power Output, Timer 1 Clock Input (T1CLK pin)
RA3	8	High Power Output, Timer 1 Output (T1OUT pin)
RB0	13	External Interrupt, Timer 2 Capture 1 Input (T2CPI1 pin)
RB1	14	External Interrupt, Timer 2 Capture 2 Input (T2CPI2 pin)
RB2	15	External Interrupt, Timer 2 Clock Input (T2CLK pin)
RB3	16	External Interrupt, Timer 2 Output (T2OUT pin)
RB4	17	External Interrupt
RB5	18	External Interrupt, Parallel Slave Peripheral $\overline{\text{HOLD}}$ output
RB6	19	External Interrupt, Parallel Slave Peripheral R/$\overline{\text{W}}$ input
RB7	20	External Interrupt, Parallel Slave Peripheral $\overline{\text{CS}}$ input
RC0	21	Parallel Slave Peripheral Data
RC1	22	Parallel Slave Peripheral Data
RC2	23	Parallel Slave Peripheral Data
RC3	24	Parallel Slave Peripheral Data
RC4	24	Parallel Slave Peripheral Data
RC5	26	Parallel Slave Peripheral Data
RC6	27	Parallel Slave Peripheral Data
RC7	28	Parallel Slave Peripheral Data
RD0	29	Parallel Slave Peripheral Data
RD1	30	Parallel Slave Peripheral Data
RD2	35	Parallel Slave Peripheral Data
RD3	36	Parallel Slave Peripheral Data

Table 4-1 I/O Port Pin Alternate Functions (continued)

Name	Pin	Alternate Function
RD4	37	Parallel Slave Peripheral Data
RD5	38	Parallel Slave Peripheral Data
RD6	39	Parallel Slave Peripheral Data
RD7	40	Parallel Slave Peripheral Data
RE0	41	Serial 1 Clock (S1CLK pin)
RE1	42	Serial 1 Receive Plus-Side Data (S1RXP pin)
RE2	43	Serial 1 Receive Minus-Side Data (S1RXM pin)
RE3	44	Serial 1 Receive Data (S1RXD pin)
RE4	45	Serial 1 Transmit Plus-Side Pre-Emphasis Data/Output Enable (S1TXPE/S1OE pin)
RE5	46	High Power Output, Serial 1 Transmit Plus-Side Data (S1TXP pin)
RE6	47	High Power Output, Serial 1 Transmit Minus-Side Data (S1TXM pin)
RE7	48	Serial 1 S1 Transmit Minus-Side Pre-Emphasis Data (S1TXME pin)
RF0	49	Serial 2 Transmit Plus-Side Pre-Emphasis Data/Output Enable (S2TXPE/S2OE pin)
RF1	50	High Power Output, Serial 2 Transmit Plus-Side Data (S2TXP pin)
RF2	51	High Power Output, Serial 2 Transmit Minus-Side Data (S2TXM pin)
RF3	52	Serial 2 Transmit Minus-Side Pre-Emphasis Data (S2TXME pin)
RF4	57	Serial 2 Clock (S2CLK pin)

Table 4-1 I/O Port Pin Alternate Functions (continued)

Name	Pin	Alternate Function
RF5	58	Serial 2 Receive Plus-Side Data (S2RXP pin)
RF6	59	Serial 2 Receive Minus-Side Data (S2RXM pin)
RF7	60	Serial 2 Receive Data (S2RXD pin)
RG0	61	ADC0 Input, Comparator Output
RG1	62	ADC1 Input, Comparator – Input
RG2	63	ADC2 Input, Comparator + Input
RG3	64	ADC3 Input, ADC reference Input
RG4	66	ADC4 Input, SERDES1 Squelch – Input
RG5	67	ADC5 Input, SERDES1 Squelch + Input
RG6	68	ADC6 Input, SERDES2 Squelch – Input
RG7	69	ADC7 Input, SERDES2 Squelch + Input

Figure 4-1 shows the internal hardware structure and configuration registers for each pin of a port.

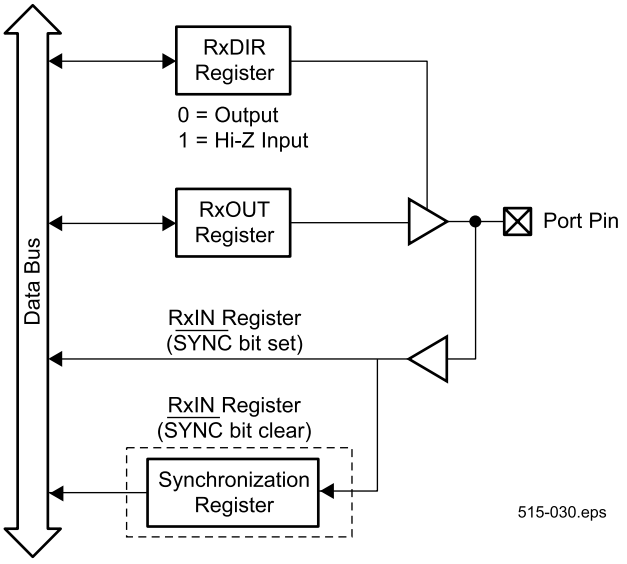


Figure 4-1 Port Pin Block Diagram

4.1.1 Port B Interrupts

Any of the 8 Port B pins can be configured as an external interrupt input. Logic on these inputs can be programmed to sense rising or falling edges. When an edge is detected, the interrupt flag for the port pin is set.

The recommended initialization sequence is:

1. Configure the port pins used for interrupts as inputs by programming the RBDIR register.
2. Select the desired edge for triggering the interrupt by programming the INTED register. This may set interrupt flags. Be sure all enabled interrupt pins are driven to valid logic levels, not floating.
3. Clear the interrupt flags in the INTF register.
4. Enable the interrupt input(s) by setting the corresponding bit(s) in the INTE register.
5. Set the GIE bit.

Figure 4-2 shows the Port B interrupt logic. Port B has three registers for supporting external interrupts, the INTED (Section 4.1.6), INTF (Section 4.1.7), and INTE (Section 4.1.8) registers. The INTED register controls the logic which selects the edge sensitivity (i.e. rising or falling edge) of the Port B pins. When an edge of the selected type occurs, the corresponding flag in the INTF register is set, whether or not the interrupt is enabled. The interrupt signal passed to the system interrupt logic is the OR function of the AND of each interrupt flag in the INTF register with its corresponding enable bit in the INTE register.

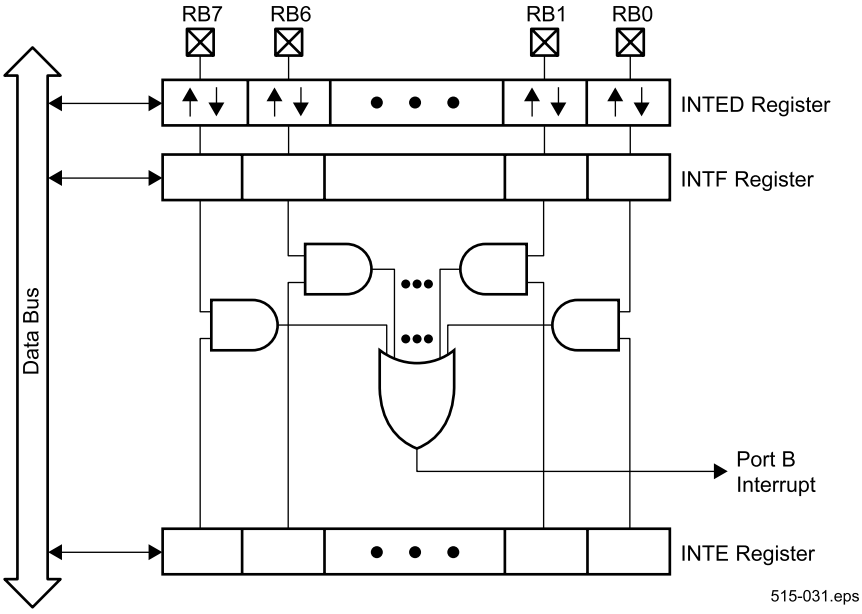


Figure 4-2 Port B Interrupt Logic

4.1.2 Reading and Writing the Ports

The port registers are memory-mapped into the data memory address space between 0x020 and 0x03A. In addition, Port B has three extra registers located between 0x017 and 0x019 which are used to support external interrupt inputs.

When two successive read-modify-write instructions read and write the same I/O port with a very high clock rate, the "write" part of one instruction might not occur soon enough before the "read"

part of the very next instruction, resulting in the second instruction getting "old" data. To ensure predictable results, avoid using two successive read-modify-write instructions that access the same register if the clock rate is high, or insert 2 **nop** instructions between successive read-modify-write instructions (if the $\overline{\text{SYNC}}$ bit in the FUSE1 register is clear, 3 **nop** instructions are required). When reading from the RxOUT register rather than the RxIN register, the CPU reads data from a register instead of the port pins. In this case, the **nop** instructions are not required.

4.1.3 RxIN Register

The RxIN registers are virtual registers that provide read-only access to the physical I/O pins. Reading these registers returns the states on the pins, which may be driven either by the IP2022 or an external device. If the $\overline{\text{SYNC}}$ bit in the FUSE1 register is clear, the states are read from a synchronization register. If an application reads data from a device running asynchronously to the IP2022, the $\overline{\text{SYNC}}$ bit should be cleared to avoid the occurrence of metastable states (i.e. corrupt data caused by an input which fails to meet the setup time before the sampling clock edge, which theoretically could interfere with the operation of the CPU).

4.1.4 RxOUT Register

The RxOUT registers are data output buffer registers. The data in these registers is driven on any I/O pins that are configured as

outputs. On reads, the RxOUT registers return the data previously written to the data output buffer registers, which might not correspond to the states actually present on pins configured as inputs or pins forced to another state by an external device.

4.1.5 RxDIR Register

The RxDIR registers select the direction of the port pins. For each output port pin, clear the corresponding RxDIR bit. For each input port pin, set the corresponding RxDIR bit. Unused pins that are left open-circuit should be configured as outputs, to keep them from floating.

For example, to configure Port A pins RA3 and RA2 as outputs and RA1 and RA0 as inputs, the following code could be used:

```
mov     w,#0x03     ;load W with the value 03h
                    ;(bits 3:2 low, and bits 1:0
                    ;high)
mov     0x022,w     ;write 03h to RADIR
                    ;register
```

The second move instruction in this example writes the RADIR register, located at address 0x022. Because Port A has only four I/O pins, only the four least significant bits of this register are used.

To drive the RA1 pin low and the RA0 pin high, the following code then could be executed:

```
mov     w,#0x01     ;load W with the value 01h
                    ;(bits 3:1 low, and bit 0
                    ;high)
mov     0x021,w     ;write 01h to RAOUT
                    ;register
```

The second move instruction shown above writes the RAOUT register, located at address 0x021. When reading the Port A pins through the RAIN register (0x020), the upper four bits always read as zero.

When a write is performed to the RxOUT register of a port pin that has been configured as an input, the write is performed but it has no immediate effect on the pin. If later that pin is configured as an output, the pin will be driven with the data that had been previously written to the RxOUT register.

4.1.6 INTED Register

The INTED register consists of 8 edge detection bits that correspond to the 8 pins of Port B. A set bit in the INTED register makes the corresponding port pin trigger on falling edges, while a clear bit makes the pin trigger on rising edges.

4.1.7 INTF Register

The INTF register consists of 8 interrupt flags that correspond to the 8 pins of Port B. If the trigger condition for a Port B pin occurs,

the corresponding bit in the INTF register is set. The bit is set even if the port pin is not enabled as a source of interrupts.

The interrupt service routine (ISR) can check this register to determine the source of an external interrupt. If a Port B pin enabled for generating interrupts has a set bit in the INTF register, software must clear the bit prior to exiting to prevent repeated calls to the ISR.

The Port B interrupt logic is asynchronous (e.g. functions without a clock in clock-stop mode). A side effect is that there is a 2-cycle delay between the instruction that clears a INTF bit and the bit being cleared. This means that software must clear the bit at least 2 cycles before executing a return from interrupt (**`reti`**) instruction.

4.1.8 INTE Register

The INTE register consists of 8 interrupt enable bits that correspond to the 8 pins of Port B. A Port B pin is enabled as a source of interrupts by setting the corresponding bit in the INTE register. The pin is disabled as an interrupt source by clearing the corresponding INTE bit.

4.1.9 Port Configuration Upon Power-Up

On power-up, all the port control registers (RxDIR) are initialized to 0xFF. Therefore, each port pin is configured as a high-impedance input. This prevents any false signalling to external

components which could occur if the ports were allowed to assume a random configuration at power-up.

4.2 Timer 0

Timer 0 is an 8-bit timer intended to generate periodic interrupts for ipModule software that requires being called at a constant rate, such as UART and DTMF functions. This use is supported in the instruction set by an option for the **`reti`** instruction which adds the W register to the T0TMR register when returning from an interrupt. For more information about the use of this option, see the *Virtual Peripheral Guidelines* document. Figure 4-3 shows the Timer 0 logic.

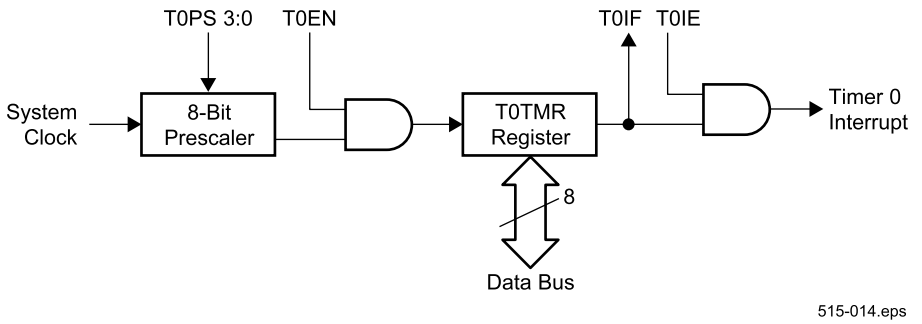


Figure 4-3 Timer 0 Block Diagram

Timer 0 is readable and writable as the T0TMR register. The control and status register for Timer 0 is the T0CFG register, as described in Section 4.2.1. Timer 0 should be enabled before its interrupt is enabled.

4.2.1 T0CFG Register

7	6 3	2	1	0
T0EN	T0PS3:0	Reserved	T0IE	T0IF

Name	Description
T0EN	Enables Timer 0 0 = Timer 0 disabled 1 = Timer 0 enabled
T0PS3:0	Specifies Timer 0 prescaler divisor 0000 = 1 0001 = 2 0010 = 4 0011 = 8 0100 = 16 0101 = 32 0110 = 64 0111 = 128 1000 = 256 1001 to 1111 = Reserved
T0IE	Timer 0 interrupt enable bit 0 = Timer 0 interrupt disabled 1 = Timer 0 interrupt enabled

Name	Description
T0IF	Timer 0 interrupt flag 0 = No timer overflow has occurred since this bit was last cleared 1 = Timer overflow has occurred

4.3 Real-Time Timer

The Real-Time Timer is an 8-bit timer intended to provide a periodic system wake-up interrupt. Unlike the other peripherals (except the Watchdog Timer and port B interrupts), the Real-Time Timer continues to function when the system clock is disabled. For those applications which spend much of their time with the OSC clock oscillator turned off to conserve power, there are only three available mechanisms for a wake-up: reset from the Watchdog Timer, interrupt from a Port B input, and interrupt from the Real-Time Timer. By using an interrupt rather than reset, more of the CPU state is preserved and some reset procedures such as initializing the port direction registers can be skipped.

Figure 4-4 shows the Real-Time Timer logic.

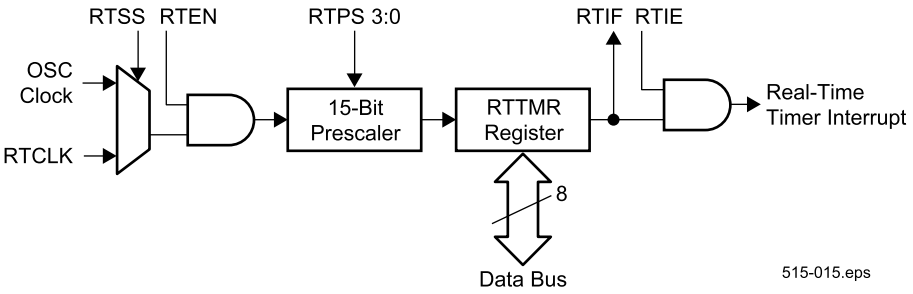


Figure 4-4 Real-Time Timer Block Diagram

The real-time timer is readable and writable as the RTTMR register. The control and status register for the real-time timer is the RTCFG register, as described in Section 4.3.1.

The RTEOS bit in the XCFG register selects the sampling mode for the external input. If the RTEOS bit is set, the external input is over-sampled with the system clock. The CPU can always read the value in the RTTMR register, however, the system clock must be at least twice the frequency of the external input. If the system clock source is changed to RTCLK or turned off, then the RTEOS bit must be clear for the Real-Time Timer to function.

If the RTEOS bit is clear then the external input directly clocks the Real-Time Timer (i.e. RTCLK is not oversampled). The Real-Time Timer will always function whether the clock input is synchronous or asynchronous. However, the CPU cannot reliably read the

value in the RTTMR register unless the RTCLK clock is synchronous to the system clock.

If the value in the RTTMR register does not need to be used by the CPU (i.e. only the interrupt flag is of interest) then the RTEOS bit can be clear (i.e. RTCLK not oversampled) which allows the Real-Time Timer to function for any configuration of the system clock.

If the value in the RTTMR register needs to be used by the CPU, but the Real-Time Timer is not required to function when the system clock is set to RTCLK or turned off, then the RTEOS bit should be set to ensure the CPU can reliably read the RTTMR register.

If the value in the RTTMR register needs to be used by the CPU and the Real-Time Timer is required to function when the system clock is set to RTCLK or off, then software must change the RTEOS bit when changing the system clock source. To read the RTTMR register when the system clock is not synchronous to the RTCLK, the RTEOS bit must be clear to ensure reliable operation. Before the system clock is changed to RTCLK or turned off, the RTEOS bit must be set (i.e. RTCLK not oversampled) for the Real-Time Timer to continue to function.

The Real-Time Timer should be enabled before its interrupt is enabled.

4.3.1 RTCFG Register

7	6 3	2	1	0
RTEN	RTPS3:0	RTSS	RTIE	RTIF

Name	Description
RTEN	Real-Time Timer enable bit 0 = Real-Time Timer disabled 1 = Real-Time Timer enabled
RTPS3:0	Real-Time Timer prescaler divisor 0000 = 1 0001 = 2 0010 = 4 0011 = 8 0100 = 16 0101 = 32
	0110 = 64 0111 = 128 1000 = 256 1001 = 512 1010 = 1024 1011 = 2048 1100 = 4096 1101 = 8192 1110 = 16384 1111 = 32768

Name	Description
RTSS	Real-Time Timer clock source select (see Figure 2-12) 0 = pre-PLL clock 1 = external RTCLK
RTIE	Real-Time Timer interrupt enable bit 0 = Real-Time Timer interrupt disabled 1 = Real-Time Timer interrupt enabled
RTIF	Real-Time Timer interrupt flag 0 = No timer overflow has occurred since this bit was last cleared 1 = Timer overflow has occurred. This bit goes high two cycles after the actual overflow occurs.

4.4 Multi-Function Timers (T1 and T2)

The IP2022 contains two independent 16-bit multi-function timers, called T1 and T2. These versatile, programmable timers reduce the software burden on the CPU in real-time control applications such as PWM generation, motor control, triac control, variable-brightness display control, sine-wave generation, and data acquisition.

Each timer consists of a 16-bit counter register supported by a dedicated 16-bit capture register and two 16-bit compare registers. The second compare register can also serve as capture register. Each timer may use up to four external pins: TxCPI1 (Capture Input), TxCPI2 (Capture Input), TxCLK (Clock Input), TxOUT (Output). These pins are multiplexed with general-purpose

I/O port pins. The port direction register has priority over the timer configuration, so the port direction register must be programmed appropriately for each of these four signals if their associated timer functions are used. Each timer should be enabled before its interrupt is enabled.

Figure 4-5 is a block diagram showing the registers and I/O pins of one timer. Each timer is based on a 16-bit counter/timer driven by a 15-bit prescaler. The input of the prescaler can be either the system clock or an external clock signal which is internally synchronized to the system clock. The counter cannot be directly written by software, but it may be cleared by writing to the TxRST bit in the TxCTRL register.

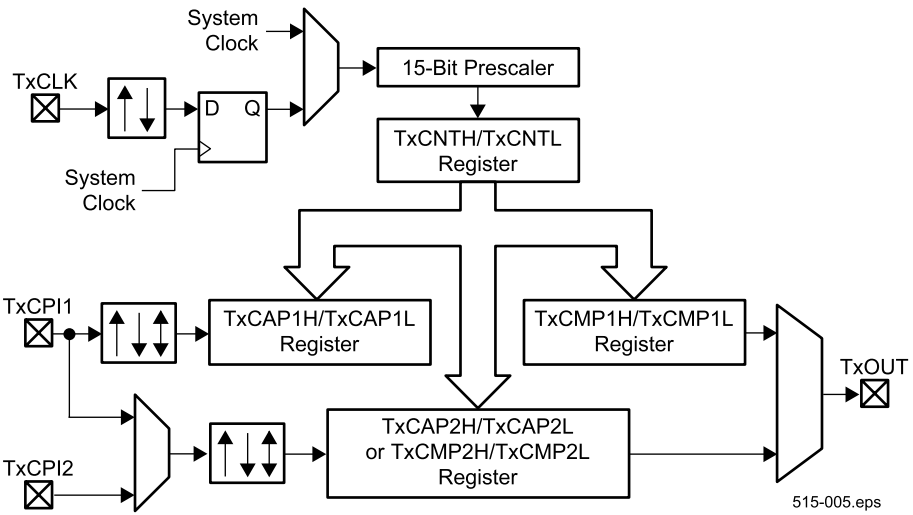


Figure 4-5 Multi-Function Timer Block Diagram

4.4.1 Timers T1, T2 Operating Modes

Each timer can be configured to operate in one of the following modes:

- Pulse-Width Modulation (PWM)
- Timer
- Capture/Compare

PWM Mode

In PWM Mode, the timer can generate a pulse-width modulated signal on its output pin, TxOUT. The period of the PWM cycle (high + low) is specified by the value in the TxCAP2H/TxCAP2L register. The high time of the pulse is specified by the value in the TxCMP1H/TxCMP1L register.

PWM mode can be used to generate an external clock signal that is synchronous to the IP2022 system clock. For example, by loading TxCMP1H/TxCMP1L with 1 and TxCAP2H/TxCAP2L with 2, a symmetric 50-MHz external clock can be generated from a 100 MHz system clock. In some applications, this can eliminate crystals or oscillators required to produce clock signals for other components in the system.

The 16-bit counter/timer counts upward. After reaching the value stored in the TxCMP1H/TxCMP1L register minus one, at the next clock edge the TxOUT pin is driven low. The counter/timer is unaffected by this event and continues to increment. After reaching the value stored in the TxCAP2H/TxCAP2L register minus one, at the next clock edge the timer is cleared. When the

counter is cleared, the TxOUT output is driven high, *unless* the TxCMP1H/TxCMP1L register is clear, in which case the TxOUT pin is driven low.

There are two special cases. When the TxCMP1H/TxCMP1L register is clear, the TxOUT pin is driven with a continuous low, corresponding to a duty-cycle of 0%. When the value in the TxCMP1H/TxCMP1L register is equal to the value in the TxCAP2H/TxCAP2L register, the TxOUT output is driven with a continuous high, corresponding to a duty-cycle of 100%.

The behavior of the timers when the value in the TxCMP1H/TxCMP1L register are greater than the value in the TxCAP2H/TxCAP2L register is undefined.

The timer is glitch-free no matter when the TxCMP1H/TxCMP1L register or the TxCMP2H/TxCMP2L register are changed relative to the value of the internal counter/timer. The new duty cycle or period values do not take effect until the current PWM cycle is completed (the counter/timer is reset).

Interrupts, if enabled through the TxCFG1 register, can be generated whenever the timer output is set or cleared. If the TxCMP1H/TxCMP1L register is clear, or if the value in the TxCMP1H/TxCMP1L register is equal to the value in the TxCAP2H/TxCAP2L register, an interrupt is generated each time the counter/timer is reset to zero.

In PWM mode, the Capture 1 input remains active (if enabled by the CPI1EN bit in the TxCFG1 register) and, when triggered, captures the current counter/timer value into the TxCAP1 register.

The multifunction timers can be configured to interrupt on a Capture 1 event and reset the counter/timer on the event. For PWM operation without Capture 1, software must disable the Capture 1 input by clearing the CPI1EN bit in the TxCFG1 register.

Timer Mode

This is not a separate timer mode (from the hardware point of view), but is a conceptual mode for programmers. It is the PWM mode, except that software disables the timer output by clearing the OEN bit in the TxCFG register.

Capture/Compare Mode

In Capture/Compare mode, one or both of the timer capture inputs (TxCPI1 and TxCPI2) may be used. Their pin functions must be enabled in the TxCFG1 register. Each capture input can be programmed in the TxCFG2 register to trigger on a rising edge, falling edge, or both rising and falling edges.

When a trigger event occurs on either capture pin, the current value of the counter/timer is captured into the TxCAP1H/TxCAP1L register or the TxCAP2H/TxCAP2L register for that input pin.

The counter/timers can also be configured to reset on a TxCP1 input event, in which case the value of the counter/timer before it was reset is captured in the TxCAP1H/TxCAP1L register and the counter/timer is reset to zero. This mode is useful for measuring the frequency (or width) of external signals. By using both capture inputs and configuring them for opposite edges, the duty cycle of a signal can also be measured. To avoid wasting I/O port pins in

this configuration, the CPI2CPI1 bit in the TxCFG1 register is provided to internally tie the TxCPI1 and TxCPI2 inputs together, which frees the TxCPI2 pin to be used as a general-purpose I/O port pin.

An interrupt can be generated for any capture event and for counter/timer overflows.

This mode also features an output-compare function. The TxCMP1H/TCMP1L register is constantly compared against the internal counter/timer. When the counter/timer reaches the value of the TxCMP1H/TxCMP1L register minus one, at the next counter clock the TxOUT output is toggled. The TxOUT output, if enabled via the OEN bit, can be driven high or low by writing to the TOUTSET and TOUTCLR bits in the TxCFG2 register. An interrupt can be enabled for this event.

4.4.2 T1 and T2 Timer Pin Assignments

The following table lists the I/O port pins associated with the Timer T1 and Timer T2 I/O functions.

Table 4-2 Timer T1/T2 Pin Assignments

I/O Pin	Timer T1/T2 Function
RA0	Timer T1 Capture 1 Input
RA1	Timer T1 Capture 2 Input
RA2	Timer T1 External Event Clock Source
RA3	Timer T1 Output
RB0	Timer T2 Capture 1 Input
RB1	Timer T2 Capture 2 Input
RB2	Timer T2 External Event Clock Source
RB3	Timer T2 Output

4.4.3 TxCNTH/TxCNTL Register

The TxCNTH/TxCNTL register indicates the value of the counter/timer and increments synchronously with the rising edge of the system clock. This register is read-only. The timer counter may be cleared by writing to the TxRST bit in the TCTRL register.

Reading the TxCNTL register returns the least-significant 8 bits of the internal TxCNT counter and causes the most-significant 8 bits of the counter to be latched into the TxCNTH register. This allows software to read the TxCNTH register later and still be assured of atomicity.

4.4.4 TxCAP1H/TxCAP1L Register

The TxCAP1H/TxCAP1L register captures the value of the counter/timer when the TxCPI1 input is triggered. This register is read-only.

Reading the TxCAP1L register returns the least-significant 8 bits of an internal capture register and causes the most-significant 8-bits of the register to be latched into the TxCAP1H register. This allows software to read the TxCAP1H register later and still be assured of atomicity.

4.4.5 TxCMP1H/TxCMP1L Register

In Capture/Compare mode, the TxOUT output pin is toggled (if enabled by the OEN bit in the TxCFG1 register) when the counter/timer increments to the value in the TxCMP1 register. In this mode, the value written to the TxCMP1 register takes effect immediately.

Writing to the TxCMP1L register causes the value to be stored in the TxCMP1L register with no other effect. Writing to the TxCMP1H register causes an internal compare register to be loaded with a 16-bit value in which the low 8 bits come from the TxCMP1L register and high 8 bits come from the value being written to the TxCMP1H register. Software should write the TxCMP1L register before writing the TxCMP1H register, because writing to the TxCMP1H register is used as an indication that a new compare value has been written. Writing to the TxCMP1H register is required for the new compare value to take effect. In

PWM mode, the 16-bit number latched into the internal compare register by writing to the TxCMP1H register does not take effect until the end of the current PWM cycle.

Reading the TxCMP1H or TxCMP1L registers returns the previously written value whether or not the value stored in these registers has been transferred to the internal compare register by writing to the TxCMP1H register.

4.4.6 TxCAP2H/TxCAP2L or TxCMP2H/TxCMP2L Register

This register may be called the TxCAP2H/TxCAP2L register or the TxCMP2H/TxCMP2L register.

In PWM mode, this register determines the period of the PWM signal. In this mode, this register is both readable and writeable. However, on writes the value is not applied until the end of the current PWM cycle.

Writing to the TxCAP2L register causes the value to be stored in the TxCAP2L register with no other effect. Writing to the TxCAP2H register causes an internal compare register to be loaded with a 16-bit value in which the low 8 bits come from the TxCAP2L register and the high 8 bits come from the value being written to the TxCAP2H register. Software should write the TxCAP2L register before writing the TxCAP2H register, because writing to the TxCAP2H register is used as an indication that a new compare value has been written. Writing to the TxCAP2H register is required for the new compare value to take effect. In PWM mode, the 16-bit number latched into the internal compare register by

writing to the TxCAP2H register does not take effect until the end of the current PWM cycle.

Reading the TxCAP2H or TxCAP2L registers returns the previously written value regardless of whether the value stored in these registers has been transferred to the internal compare register by writing to the TxCAP2H register.

In Capture/Compare mode, this register captures the value of the counter/timer when the TxCPI2 input is triggered. In this mode, this register is read-only.

Reading the TxCAP2L register returns the least-significant 8 bits of an internal capture register and causes the most-significant 8-bits to be latched into the TxCAP2H register. This allows software to read the TxCAP2H register later and still be assured of atomicity.

4.4.7 TxCFG1H/TxCFG1L Register

15	14	13	12	11	10	9	8
OFIE	CAP2IE CMP2IE	CAP1IE	CMP1IE	OFIF	CAP2IF CMP2IF	CAP1IF	CMP1IF

7	6	5	4	3	2	1	0
MODE	OEN	ECLKEN	CPI2EN	CPI1EN	ECLKEDG	CAP1RST	TMREN

Name	Description
OFIE	Timer overflow interrupt enable bit 0 = Overflow interrupt disabled 1 = Overflow interrupt enabled
CAP2IE CMP2IE	PWM mode: Compare 2 interrupt enable bit Capture/Compare mode: Capture 2 interrupt enable bit 0 = Capture/Compare 2 interrupt disabled 1 = Capture/Compare 2 interrupt enabled
CAP1IE	Capture 1 interrupt enable bit 0 = Capture 1 interrupt disabled 1 = Capture 1 interrupt enabled
CMP1IE	Compare 1 interrupt enable bit 0 = Compare 1 interrupt disabled 1 = Compare 1 interrupt enabled

Name	Description
OFIF	Timer overflow interrupt flag 0 = No timer overflow has occurred since this bit was last cleared 1 = Timer overflow has occurred
CAP2IF CMP2IF	PWM mode: Compare 2 interrupt flag (i.e. timer value matched TxCMP2 value) Capture/Compare mode: Capture 2 flag (i.e. TxCPI2 input triggered) 0 = No capture/compare 2 event has occurred since this bit was last cleared 1 = Capture/compare 2 event has occurred
CAP1IF	Capture 1 interrupt flag 0 = No capture 1 event has occurred since this bit was last cleared 1 = Capture 1 event has occurred
CMP1IF	Compare 1 interrupt flag 0 = No compare 1 event has occurred since this bit was last cleared 1 = Compare 1 event has occurred
MODE	Timer mode select 0 = PWM/timer mode 1 = Capture/compare mode
OEN	TxOUT enable bit 0 = TxOUT disabled. Port pin available for general-purpose I/O. 1 = TxOUT enabled. Port pin must be configured for output in corresponding RxDIR register bit.

Name	Description
ECLKEN	TxCLK enable bit 0 = TxCLK disabled. Port pin available for general-purpose I/O. 1 = TxCLK enabled as clock source for timer. Enabling this bit does not make any other restrictions on the use of the TxCLK port pin for general-purpose I/O.
CPI2EN	TxCPI2 enable bit 0 = System clock enabled as clock source for timer. TxCPI2 port pin available for general-purpose I/O. 1 = TxCLK enabled as clock source for timer. Enabling this bit does not make any other restrictions on the use of the port pin for general-purpose I/O.
CPI1EN	TxCPI1 enable bit 0 = Capture 1 input disabled. TxCPI1 port pin available for general-purpose I/O. 1 = TxCPI1 enabled as capture 1 input. Enabling this bit does not make any other restrictions on the use of the port pin for general-purpose I/O.
ECLKEDG	TxCLK edge sensitivity select. (This bit is ignored if the ECLKEN bit is clear.) 0 = TxCLK increments timer on rising edge 1 = TxCLK increments timer on falling edge
CAP1RST	Reset timer on capture 1 event enable bit 0 = Timer value unchanged by occurrence of a capture 1 event 1 = Timer value cleared by occurrence of a capture 1 event
TMREN	Timer enable bit 0 = Timer disabled. Timer clock source shut off to reduce power consumption. 1 = Timer enabled

4.4.8 TxCFG2H/TxCFG2L Register

15	14	13	12	11			8
0	0	0	0	PS3:0			

7	6	5	4	3	2	1	0
Reserved	TOUTSET	TOUTCLR	CPI2CPI1	CPI2EDG1:0		CPI1EDG1:0	

Name	Description
PS3:0	Timer prescaler divisor 0000 = 1 0001 = 2 0010 = 4 0011 = 8 0100 = 16 0101 = 32 0110 = 64 0111 = 128 1000 = 256 1001 = 512 1010 = 1024 1011 = 2048 1100 = 4096 1101 = 8192 1110 = 16384 1111 = 32768

Name	Description
TOUTSET	Override bit to set the TxOUT output. This bit always reads as zero. 0 = Writing 0 to this bit has no effect 1 = Writing 1 to this bit forces the TxOUT signal high
TOUTCLR	Override bit to clear the TxOUT output. This bit always reads as zero. 0 = Writing 0 to this bit has no effect 1 = Writing 1 to this bit forces the TxOUT signal low
CPI2CPI1	Internally connect the TxCPI2 input to the TxCPI1 input. This makes the TxCPI2 port pin available for general-purpose I/O. 0 = No internal connection between TxCPI1 and TxCPI2 1 = TxCPI1 and TxCPI2 internally connected
CPI2EDG1:0	TxCPI2 edge sensitivity select 00 = Falling edge on TXCPI2 recognized as capture 2 event 01 = Rising edge on TXCPI2 recognized as capture 2 event 10 = Any falling or rising edge on TXCPI2 recognized as capture 2 event 11 = Any falling or rising edge on TXCPI2 recognized as capture 2 event

Name	Description
CPI1EDG1:0	TxCPI1 edge sensitivity select 00 = Falling edge on TXCPI1 recognized as capture 1 event 01 = Rising edge on TXCPI1 recognized as capture 1 event 10 = Any falling or rising edge on TXCPI1 recognized as capture 1 event 11 = Any falling or rising edge on TXCPI1 recognized as capture 1 event

4.4.9 TCTRL Register

7	6	5	4	3	2	1	0
0	0	T2IE	T1IE	0	0	T2RST	T1RST

Name	Description
T2IE	Timer 2 interrupt enable 0 = Timer 2 interrupt disabled 1 = Timer 2 interrupt enabled
T1IE	Timer 1 interrupt enable 0 = Timer 1 interrupt disabled 1 = Timer 1 interrupt enabled

Name	Description
T2RST	Timer 2 reset bit. This bit always reads as zero. 0 = Writing 0 to this bit has no effect. 1 = Writing 1 to this bit clears Timer 2.
T1RST	Timer 1 reset bit. This bit always reads as zero. 0 = Writing 0 to this bit has no effect. 1 = Writing 1 to this bit clears Timer 1.

4.5 Watchdog Timer

A Watchdog Timer is available for recovering from unexpected system hangups. When the Watchdog Timer is enabled, software must periodically clear the timer by executing a **`cwdt`** instruction. Otherwise, the timer will overflow, which resets the IP2022 and sets the WD bit in the STATUS register. Any other source of reset clears the WD bit, so software can use this bit to identify a reset caused by the Watchdog Timer. The Watchdog Timer is shown in Figure 4-6.

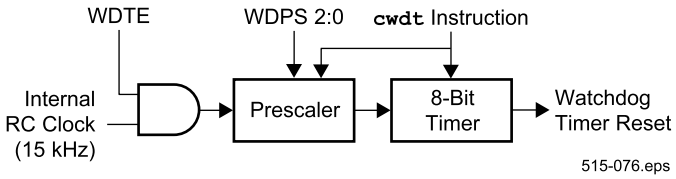


Figure 4-6 Watchdog Timer

The Watchdog Timer is enabled by setting the WDTE bit in the FUSE1 register. The time period between reset or clearing the

timer and timer overflow is controlled by the WDPS2:0 bits in the FUSE1 register, as shown in Table 4-3.

Table 4-3 Watchdog Timer Period

WDPS2:0 (FUSE1 register)	Period (ms)*
000	20
001	40
010	80
011	160
100	320
101	640
110	1280
111	2560

* Time periods are approximate

The Watchdog Timer register is not visible to software. The only feature of the Watchdog Timer visible to software is the WD bit in the STATUS register.

4.6 Serializer/Deserializer (SERDES)

There are two SERDES units in the IP2022, which support a variety of serial communication protocols, including GPSI, SPI, UART, USB, and 10Base-T Ethernet. By performing data serialization/deserialization in hardware, the CPU bandwidth needed to support serial communication is greatly reduced, especially at high baud rates. Providing two units allows easy

implementation of protocol conversion or bridging functions, such as a USB to 10Base-T Ethernet bridge.

Each SERDES unit uses up to 8 digital signals: SxCLK, SxRXD, SxRXM, SxRXP, SxTXM, SxTXME, SxTXP, and SxTXPE/SxOE. The signals for SERDES1 are multiplexed with the Port E pins, and the signals for SERDES2 are multiplexed with the Port F pins. The port direction bits must be set appropriately for each pin that is used. The SxOE signal is multiplexed with the SxTXPE signal. Port pin usage is shown in Table 4-4. Not all signals are used in all protocol modes (see Table 4-7 for details).

Table 4-4 SERDES Digital Port Pin Usage

SERDES1 Signal	Port Pin	SERDES2 Signal	Port Pin	Description
S1CLK	RE0	S2CLK	RF0	Serial clock
S1RXP	RE1	S2RXP	RF1	Plus-side differential input
S1RXM	RE2	S2RXM	RF2	Minus-side differential input
S1RXD	RE3	S2RXD	RF3	Serial data
S1TXPE S1OE	RE4	S2TXPE S1OE	RF4	Plus-side differential output with pre-emphasis, or output enable for external transceiver
S1TXP	RE5	S2TXP	RF5	Plus-side differential output
S1TXM	RE6	S2TXM	RF6	Minus-side differential output
S1TXME	RE7	S2TXME	RF7	Minus-side differential output with pre-emphasis

In 10Base-T mode, there are two analog inputs: Rx+ and Rx-. The mapping of these signals to port pins is shown in Table 4-5.

Table 4-5 SERDES Analog Port Pin Usage

SERDES1 Signal	Port Pin	SERDES2 Signal	Port Pin	Description
S1Rx+	RG5	S2Rx+	RG7	Plus-side differential input
S1Rx-	RG4	S2Rx-	RG6	Minus-side differential input

Figure 4-7 shows the clock/data separation and EOP detection logic of a SERDES unit. In USB mode, the SxRXD input carries the data received from an external transceiver. The SxRXP and SxRXM pins correspond to the differential inputs of the USB bus. Providing both inputs allows sensing of an End-of-Packet (EOP) condition. The SxRXD input is also used for interfaces with single-ended input (i.e. GPSI, SPI, and UART modes), in which case the clock/data separation circuit is bypassed. For 10Base-T Ethernet, a differential line receiver is provided.

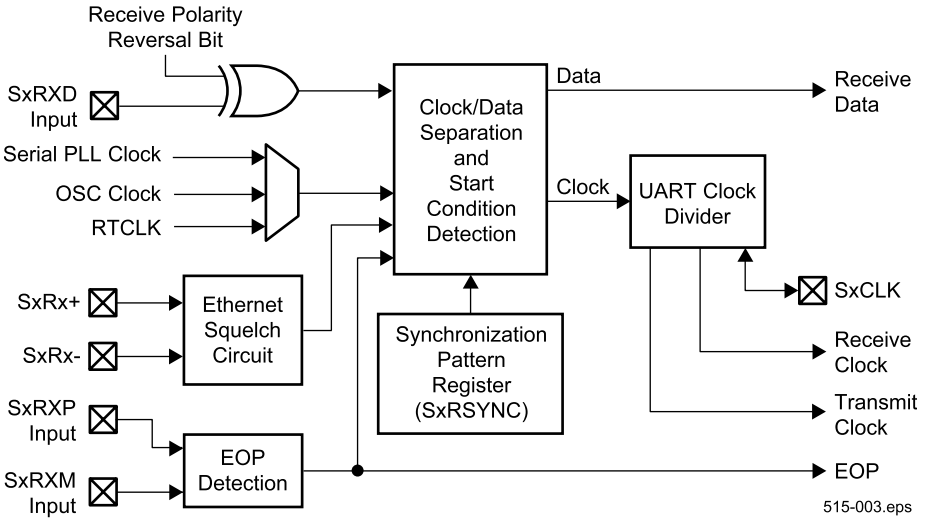


Figure 4-7 Clock/Data Separation and EOP Detection

The SxRXP and SxRXM pins should not both be grounded or left floating, otherwise a false EOP condition may be detected. For those interfaces which sense an EOP condition (i.e. any mode except UART), at least one of these inputs should be driven high to prevent spurious EOP events.

The synchronization pattern register (SxRSYNC) is used for USB and 10Base-T protocols for detecting bit patterns that signal the start of a frame. For USB, this register is loaded with 00000001, while for 10Base-T, it is 10101011 (also called the SFD, start of frame delimiter). The incoming data stream, after passing through

the polarity inversion logic (which can be turned on or off under software control) is compared to the synchronization pattern. Once a match is found, an internal counter is set to zero and data is shifted into a shift register. The synchronization matching operation is then disabled until an EOP condition is detected, because the synchronization pattern potentially could be embedded in the data stream as valid data.

The clock/data separation circuit performs Manchester decoding and NRZI decoding. In addition, bit unstuffing is performed after the NRZI decoding for the USB bus. This means every bit after a series of six consecutive ones is dropped.

For 10Base-T Ethernet operation, each SERDES is equipped with a squelch circuit for discriminating between noise, link pulses, and data. Link pulses are sent periodically to keep the channel open when no data is being transmitted. The squelch circuit handles link pulse detection, link pulse polarity detection, carrier sense, and EOP detection.

For UART operation, two internal divide-by-16 circuits are used. Based on the clock source (either internal or external), the receive section and the transmit section use two divided-by-16 clocks that potentially can be out of phase. This is due to the nature of the UART bus transfers. The receive logic, based on the 16x bit clock (the clock source chosen by user), will sample the incoming data for an falling edge. Once the edge is detected, the receive logic counts 8 clock cycles and samples the number of bits specified in the SxRCNT register using the bit clock (which is obtained by dividing the clock source by 16).

Figure 4-8 shows the receive data paths. Software prepares a SERDES unit to receive data by programming the receive shift count register (SxRCNT) and the clock select bits in the SxMODE register appropriately for the selected protocol. The SxRCNT register is copied to an internal counter, and when that number of bits of data has been received, the received data is loaded into the SxRBUF register.

In 10Base-T, GPSI, or USB mode, when an EOP is detected the SxRCNT register is loaded with the number of bits actually received, the EOP bit of the SxINTF register is set, and the data bits are loaded into the SxRBUF register. The RXBF bit in the SxINTE register can be set to enable an interrupt on this event.

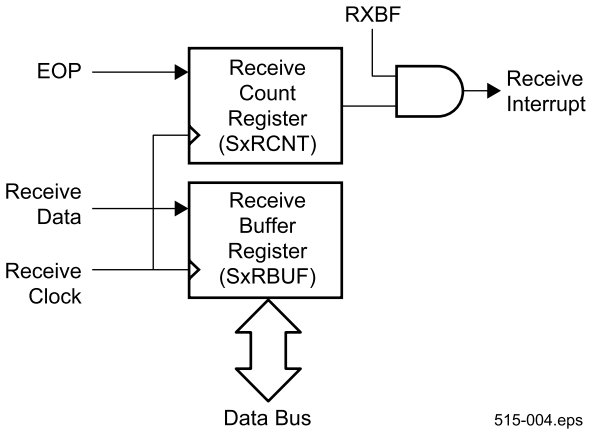


Figure 4-8 Receive Data Paths

Figure 4-9 shows the transmit data paths. The SxTXP and SxTXM pins correspond to the differential outputs of the USB or Ethernet bus. Other serial protocols require only one output pin, which is SxTXP by default.

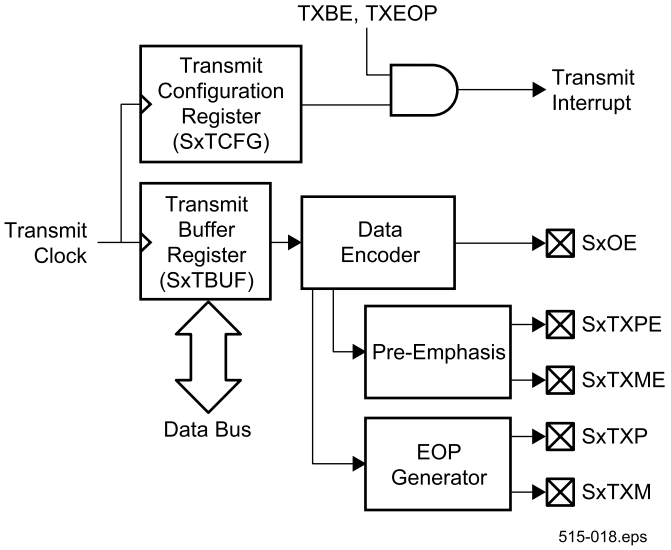


Figure 4-9 Transmit Data Paths

The SxTXP and SxTXM pins have high current outputs for driving Ethernet magnetics directly without the use of transceivers.

When the clock select register is programmed with the value for 10Base-T, the transmit pre-emphasis requirement enables the SxTXPE and SxTXME outputs, which have a 50ns-delayed

version of the transmit output that is resistively combined outside the chip before driving the magnetics.

The data encode block performs polarity inversion, if necessary, then in 10Base-T mode it performs Manchester encoding. In USB bus mode, it performs bit stuffing and then NRZI encoding. Bit stuffing means that after six consecutive ones, a zero bit is inserted. The active low SxOE pin is used to enable the USB transceiver for transmission. Otherwise, this pin is held high. For 10Base-T, the output pins of the serializer are driven low when not transmitting. The encode block is bypassed for all other protocols.

For transmitting, software must specify the number of bits to transmit and load the data in the SxTBUF register. This data is then transferred to an internal register, from which it is serially shifted out to the transmit logic. The TXBE bit in the INTE register can be set to enable an interrupt when the data has been transferred from the SxTBUF register. When there is a transmit buffer underrun event (i.e. all of the data has been shifted out from the internal register, but the SxTBUF register has not been reloaded), an EOP condition is generated on the SxTXP and SxTXM outputs after an internal counter decrements to zero. The TXEOP bit in the SxINTE register can be set to enable an interrupt when an underrun event occurs. For protocols other than USB and Ethernet, the EOP generator is bypassed.

4.6.1 Protocol Mode

Table 4-6 shows the features which are enabled for each protocol, as controlled by the PRS3:0 bits in the SxMODE register. These features affect which registers and register fields are used, for example the SxRSYNC register is only used in the USB and 10Base-T modes.

Table 4-6 Protocol Features

PRS3:0	Mode	Encoding Method	Differential or Single-Ended?	Synchronization Register Enabled?	EOP Generation/ Detection?	Bit Stuffing/ Unstuffing?	Pre-Emphasis Outputs Enabled?
0000	Disabled	None	None	No	No	No	No
0001	10Base-T	Manchester	Differen-tial	Yes	Yes	No	Yes
0010	USB Bus	NRZI	Differen-tial	Yes	Yes	Yes	No
0011	UART	None	Single-Ended	No	No	No	No
0101	SPI	None	Single-Ended	No	Yes	No	No
0110	GPSI	None	Single-Ended	No	Yes	No	No

The protocol mode also affects the signal usage, as shown in Table 4-7. The clock frequencies required for 10Base-T Ethernet and USB protocols are shown in Table 4-8.

Table 4-7 Signal Usage

Mode	SxRXD	SxRXP	SxRXM	SxTXM	SxTXP	SxTPE	SxTME	SxCLK
Disabled	Not Used	Not Used	Not Used	Not Used	Not Used	Not Used	Not Used	Not Used
10Base-T	Input	Input	Input	Output	Output	Output	Not Used	Input
USB Bus	Input	Not Used	Not Used	Output	Output	Output	Output	Not Used
UART	Input	Not Used	Not Used	Not Used	Output	Not Used	Not Used	Not Used
SPI	Input	Note 1	Not Used	Not Used	Output	Not Used	Not Used	Note 2
GPSI	Input	Input	Not Used	Input/ Output	Output	Output	Input	Note 2

Note 1. Not used in master mode, input in slave mode.

Note 2. Output in master mode, input in slave mode.

Table 4-8 Required Clock Frequencies From PLL

Protocol	Receive	Transmit
USB 1.1 High Speed	48 MHz	12 MHz
USB 1.1 Low Speed	6 MHz	1.5 MHz
10Base-T Ethernet	80 MHz	20 MHz

4.6.2 SxMODE Register

7	6	5	4	3	2	1	0
PRS3:0				SUBM1:0		CLKS1:0	

Name	Description
PRS3:0	Protocol select (see Table 4-6). All other encodings are reserved. 0000 = Disabled 0001 = 10Base-T 0010 = USB Bus 0011 = UART 0101 = SPI 0110 = GPSI

Name	Description
SUBM1:0	Submode select USB mode: 01 = Low-speed USB interface 10 = High-speed USB interface SPI mode: 00 = Positive clock polarity, receive on rising edge, transmit on falling edge 01 = Positive clock polarity, receive on falling edge, transmit on rising edge 10 = Negative clock polarity, receive on falling edge, transmit on rising edge 11 = Negative clock polarity, receive on rising edge, transmit on falling edge GPSI mode: 00 = Receive on rising edge, transmit on falling edge 01 = Receive on falling edge, transmit on falling edge 10 = Receive on rising edge, transmit on rising edge 11 = Receive on falling edge, transmit on rising edge
CLKS1:0	Clock source select 00 = Clock disabled 01 = Reserved 10 = OSC clock oscillator 11 = PLL clock multiplier

4.6.3 SxRSYNC Register

7	2	1	0
SYNCPAT7:2		SQUELCHEN	DRIBBITEN

Name	Description
SYNCPAT7:2	Synchronization pattern, bits 7:2 (USB mode only)
SQUELCHEN	USB mode: synchronization pattern, bit 1 10Base-T mode: 0 = Squelch disabled 1 = Squelch enabled
DRIBBITEN	USB mode: synchronization pattern, bit 0 10Base-T mode: 0 = Hardware handles dribble bit 1 = Software is responsible for handling dribble bit

4.6.4 SxSYNCMASK Register

10Base-T mode:

7	6 3	2	1	0
Reserved	PREAMCNT3:0	Reserved	CONTPAIR	Reserved

USB mode:

7 0
MASK7:0

Name	Description
PREAMCNT3:0	Preamble pair count (10Base-T mode only). All other encodings are reserved. 0000 = 24 pairs 0001 = 20 pairs 0010 = 16 pairs 0011 = 12 pairs 0100 = 8 pairs 0101 = 4 pairs
MASK7:0	Mask bits for SxRSYNC (USB mode only) 0 = Ignore corresponding bit in SxRSYNC 1 = Use corresponding bit in search pattern for synchronization byte

4.6.5 SxRBUFH/SxRBUFL Register

16-bit register pair for unloading received data. The RXBF bit in the SxINTF register indicates when new data has been loaded into this register. If the corresponding bit in the SxINTE register is set, an interrupt is generated.

4.6.6 SxRCFG Register

7	6	5	4 … 0
MASSEL	SYNCDETEN	RPOREV	RXSCNT4:0

Name	Description
MASSEL	10Base-T mode: 0 = Normal polarity detected 1 = Reverse polarity detected GPSI or SPI mode: 0 = Slave mode 1 = Master mode
SYNCDETEN	Synchronization byte detection enable (USB mode only) 0 = Synchronization byte detection enabled 1 = Synchronization byte detection disabled
RPOREV	Receive data polarity reversal select 0 = Data polarity uninverted 1 = Data polarity inverted
RXSCNT4:0	Receive shift count, specifies number of bits to receive

4.6.7 SxRCNT Register

7	6	5	4	0
BITORDER	RxCRSCTRL1:0	RXACNT4:0		

Name	Description
BITORDER	Bit order for transmit and receive 0 = LSB first 1 = MSB first
RxCRSCTRL1:0	Carrier sense status and interrupt control 0x = No interrupt. Software can poll the RXXCRS bit in the SxINTF register for carrier status. 10 = Interrupt on carrier detection 11 = Interrupt on loss of carrier
RXACNT4:0	Receive shift count, actual number of bits received (read-only). Exceptions occur during the last transfer: RXACNT = 0 if bit count is less than 8 RXACNT = 8 if bit count is greater than or equal to 8, but less than 16 RXACNT = 16 if bit count is greater than or equal to 16 and the RXSCNT4:0 field in the SxRCFG register is 16

4.6.8 SxTBUFH/SxTBUFL Register

16-bit register pair for loading transmit data. The TXBE bit in the SxINTF register indicates when the data has been transmitted and the register is ready to be loaded with new data. If the corresponding bit in the SxINTE register is set, an interrupt is generated.

4.6.9 SxTCFG Register

7	6	5	4 … 0
GLOBEN	LPBACK	TPOREV	TXSCNT4:0

Name	Description
GLOBEN	Global enable bit 0 = Disable SERDES output 1 = Enable SERDES output
LPBACK	Loopback enable bit 0 = Normal operation 1 = Output is driven into input
TPOREV	Transmit data polarity reversal select 0 = Data polarity uninverted 1 = Data polarity inverted
TXSCNT4:0	Transmit shift count, specifies number of bits to transmit

4.6.10 SxINTF Register

7	6	5	4	3	2	1	0
RXERROR	RXEOP	SYND	TXBE	TXEOP	SXLINKPULSE	RXBF	RXXCRS

Name	Description
RXERROR	Receive error interrupt flag 10Base-T mode: Manchester encoding data phase error USB mode: bit unstuffing error (1111111 received) 0 = Receive error has not been detected since this bit was last cleared 1 = Receive error has been detected
RXEOP	End-of-Packet detection interrupt flag 10Base-T and USB modes: end-of-packet detected GPSI mode: RxEN deasserted 0 = End-of-Packet has not been detected since this bit was last cleared 1 = End-of-Packet has been detected
SYND	Synchronization pattern detection interrupt flag (10Base-T and USB modes only) 0 = Synchronization pattern has not been detected since this bit was last cleared 1 = Synchronization pattern has been detected

Name	Description
TXBE	Transmit buffer empty interrupt flag 0 = Transmit buffer has not been empty since this bit was last cleared 1 = Transmit buffer has been empty
TXEOP	Transmit underrun. This bit is set when the previous data in the transmit buffer register (SxTXBUF) has been transmitted and no new data has been loaded in the register. In USB and 10Base-T modes, this causes an EOP condition to be generated. 0 = Transmit underrun has not occurred since this bit was last cleared 1 = Transmit underrun has occurred
SXLINKPULSE	10Base-T mode: set after a link pulse of 75 to 250 ns duration is detected. 0 = No link pulse has been detected since this bit was last cleared 1 = Link pulse detected USB mode: 0 = SERDES is transmitting 1 = SERDES is not transmitting
RXBF	Receive buffer full interrupt flag 0 = Receive buffer has not been full since this bit was last cleared 1 = Receive buffer has been full

Name	Description
RXXCRS	In 10Base-T mode, set if signal energy is detected but no link pulse is detected 0 = No signal energy without link pulse has been detected since this bit was last cleared 1 = Signal energy without a link pulse has been detected

4.6.11 SxINTE Register

The SxINTE register has the same format as the SxINTF register. For each condition indicated by a flag in the SxINTF register, setting the corresponding bit in the SxINTE register enables the interrupt for that condition. Figure 4-10 shows the interrupt logic for the two SERDES units.

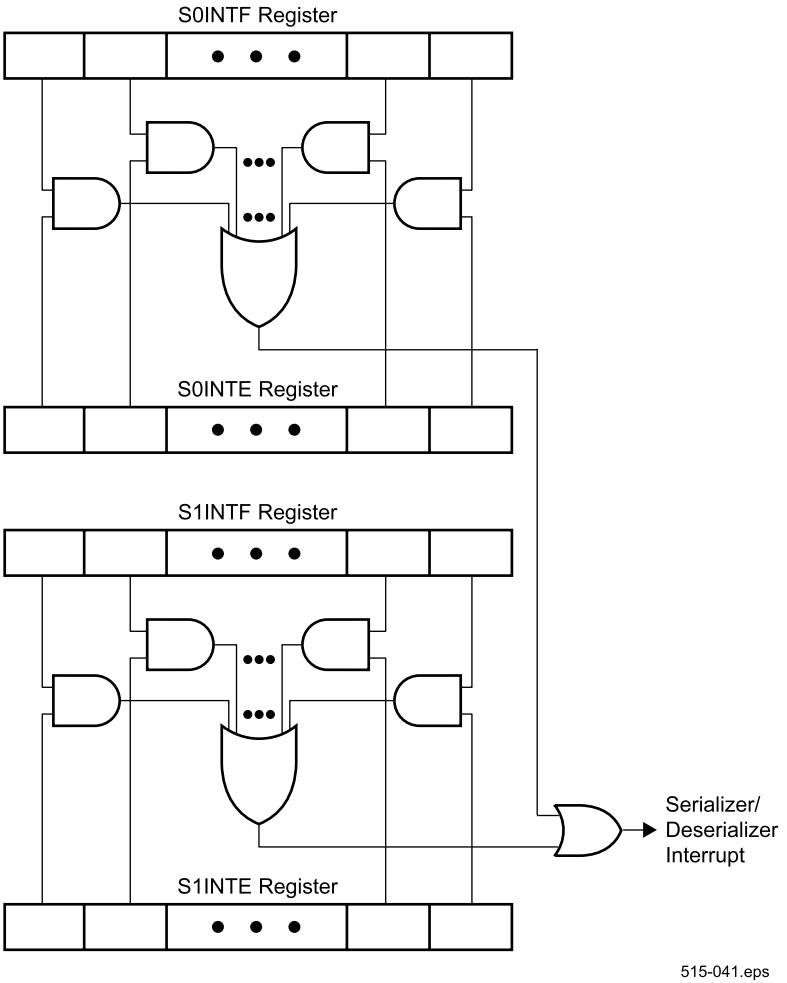


Figure 4-10 SERDES Interrupt Logic

4.6.12 SERDES Protocol-Specific Considerations

UART Interface

To set up a SERDES unit for UART mode, select UART mode in the PRS3:0 bits of the SxMODE register. This causes the data to be clocked in after a valid start bit is detected. Make sure that the polarity selected by the RPOREV bit in the SxRCFG register and the TPOREV bit in the SxTCFG register match the polarity provided by the RS-232 transceiver. (Most of them are inverted.) Make sure the bit order is compatible with the data format (RS-232 uses LSB-first bit order). The receiver uses 16X oversampling, so select a clock divisor that is 16 times the desired baud rate.

To operate in UART mode, depending on the application, either transmit or receive can be performed first. In both cases, the configuration register needs to be programmed with a bit count that is appropriate for the format. The bit count depends on the number of data bits, stop bits, and parity bits. The start bit is included in the bit count. The receiver does not check for the presence of stop bits. To detect framing errors caused by missing stop bits, increase the receiver's bit count (i.e. the RXSCNT field in the SxRCFG register) and test the trailing bit(s) in software.

SPI Protocol

To set up a SERDES unit for the SPI protocol, set up the clock with opposing phases for transmit and receive by programming the SUBM1:0 field of the SxMODE register. If in slave mode, specify an external clock. If in master mode, specify an internal clock. In slave mode, software must check if the designated slave select

line is activated before responding. In master mode, software must set the designated slave select pin to the active level before enabling the SERDES unit.

To operate in SPI mode, once the transmit and receive bit counts in the configuration registers are programmed with non-zero values, the SERDES unit begins shifting operations on the programmed clock edges. Caution must be exercised to program them quickly to avoid losing any data.

USB 1.1 Protocol

To set up a SERDES unit for USB mode, the received data output of the USB transceiver should be connected to SxRXD. The VP and VM pins of the transceiver are connected to the SxRXP and SxRXM pins to allow detection of the EOP condition. Figure 4-11 shows the connections required between an external USB transceiver and the IP2022.

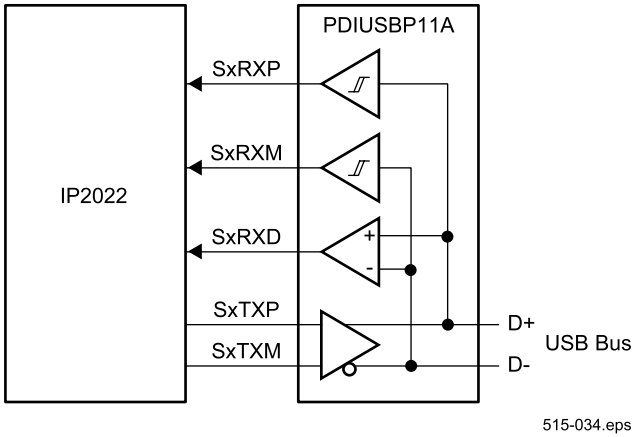


Figure 4-11 USB Interface Example

The SxMODE register must be programmed with values for a recovered clock, and the PLL clock multiplier must be programmed to generate the appropriate frequency. For example, it can be programmed at 48 MHz for full speed with a divisor of zero (=1). A divisor of 8 will make it suitable for low-speed operation. The synchronization pattern must be programmed into the SxRSYNC register to trigger an interrupt when a packet is received.

To operate in USB mode, software must perform the following functions:

- CRC generation and checking.
- Detecting reset of the device function, which is indicated by 10 milliseconds of a single-ended zero (SE0) condition on the bus.
- Detecting the suspend state, which is indicated by more than 3 milliseconds of idle. Software must make sure that the suspend current of 500 µA will be drawn after 10 milliseconds of bus inactivity.
- Formation of the USB packet by putting the sync, pid, and data into the transmit register and setting the proper count.

10Base-T Ethernet Protocol

To set up a SERDES unit for 10Base-T Ethernet, the input data from a differential line receiver or on-board comparator is connected to the SxRXD input. The signals designated Tx+, Tx-, TxD+, and TxD- correspond to the SxTXP, SxTXM, SxTXPE, and SxTXME pins of the corresponding serializers/deserializers. These pins are connected to an RJ45 jack through a transformer with terminations.

The SxMODE register must be programmed for a recovered clock, and the PLL clock multiplier must be programmed for an appropriate speed. For example, it can be programmed to be 80 MHz for 8× oversampling. The received data stream is used, together with the clock recovery circuit, to recover the original transmit clock and data.

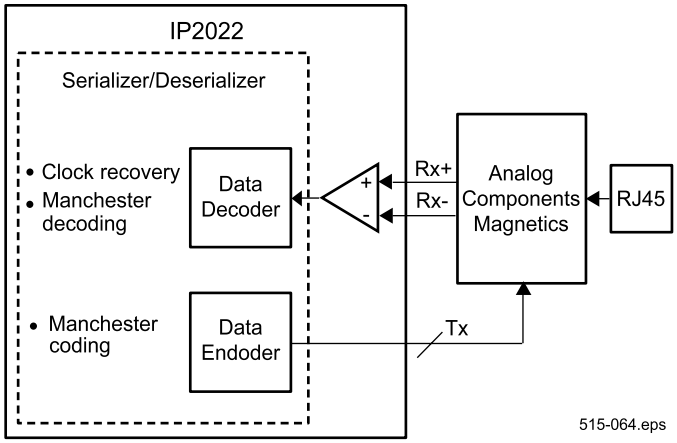


Figure 4-12 Ethernet Interface Example

The SxRXP and SxRXM signals must not be allowed to float, even if they are not used. These signals must not be driven low simultaneously.

Software must perform the following functions:

- Polarity detection and reversal.
- Carrier sense.
- Jabber detection.
- Link integrity test and link pulse generation.
- Random back off in case of collision.
- When a collision is detected, sending a 32-bit jam sequence. Collisions can be detected by either receiving an RXXCRS interrupt or by setting the bit count to 7 and then received a RXBF interrupt while transmitting.
- Formation of Ethernet packet by putting the preamble, sfd, destination address, source address, length/type, MAC client data into the transmit buffer. Frame check computation also must be done in software.
- MAC layer functions.

GPSI Interface

GPSI is a general-purpose, point-to-point, full-duplex serial bus protocol. Only two devices are allowed to exist on a bus. The GPSI master device is responsible for maintaining bus timing by driving two continuously running clocks, TxClk and RxClk. The device that does not drive the clocks is the slave device. The TxEn and TxD signals are synchronized to the TxClk clock. The RxD and RxEn signals are synchronized to the RxClk clock. The mapping of GPSI signals to IP2022 signals is shown in Table 4-9.

Table 4-9 GPSI Interface Signal Usage

GPSI Signal Name	SERDES Signal Name	Direc-tion	Description
TxClk	SxTXM	I/O	Transmit clock
TxD	SxTXP	Output	Transmit data
TxEn	SxTXPE	Output	Transmit data valid
RxClk	SxCLK	I/O	Receive clock
RxD	SxRXD	Input	Receive data
RxEn	SxRXP	Input	Receive data valid
COLLSION	GPIO	I/O	Indicates a collision at PHY layer (handled by software)
TxBUSY	SxTxME	I/O	Indicates a data transfer in progress (handled by soft-ware)

COLLISION is connected to a general-purpose port pin, and TxBUSY is connected to the SxTXME to provide additional flow control capabilities for the software device driver. The COLLISION signal is used to indicate that a PHY device has detected a collision condition. This signal is only useful when the SERDES is connected to a PHY device or acting as a PHY device.

The TxBUSY signal is used by a GPSI device to indicate that the device is currently busy, and that another device should not attempt to start a data transfer. COLLISION and TxBUSY are asynchronous to both TxClk and RxClk. TxBUSY can be

configured as either an input or an output depending which device is slower and has a need to stall incoming data.

4.7 Analog to Digital Converter (ADC)

The on-chip A/D converter has the following features:

- 10-bit ADC, ½ LSB accuracy
- 8 input channels
- 48 kHz maximum sampling rate
- One-shot conversion.
- Optional external reference voltage
- Vmax = AVdd (max 2.7V)
- Result returned in the ADCH and ADCL registers

Figure 4-13 shows the A/D converter circuitry. The ADC input pins use alternate functions of the Port G pins. The result of an ADC sample is the analog value measured on the selected pin. To correctly read an external voltage, the pin being sampled must be configured as an input in the port direction register (i.e. the RGDIR register). If the pin is configured as an output, then the result will indicate the voltage level being driven by the output buffer.

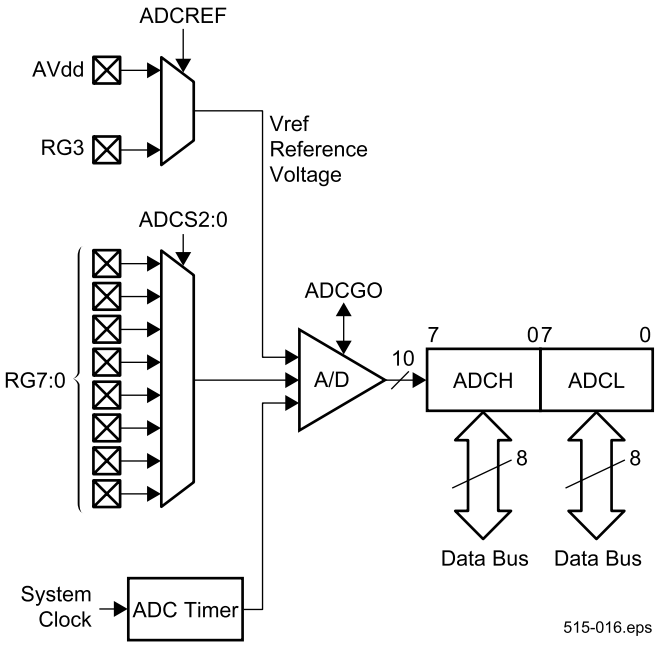


Figure 4-13 Analog-to-Digital Converter Block Diagram

The RG1 and RG2 port pins are also used as the analog comparator input pins. The result of sampling the RG1 or RG2 pins will be correct whether or not the comparator is operating. The RG0 pin is also used as the comparator output pin. If the comparator is enabled, then sampling the RG0 pin will indicate the voltage level being driven by the comparator. The RG3 pin is multiplexed with the external reference voltage.

4.7.1 ADC Reference Voltage

The reference voltage (Vref) can come from either the RG3 port pin or from the AVdd supply voltage. If AVdd is used, the RG3 port pin may be used as a channel of analog input or as a general-purpose port pin.

Vref defines a voltage level which reads as one increment of resolution below the full-scale voltage. The full-scale voltage reads as 0x3FF, so the Vref voltage reads as 0x3FE and the A/D converter resolution is 10 bits. Table 4-10 shows the values reported at the upper and lower limits of the ADC input voltage range.

Table 4-10 ADC Values

Vin Voltage	ADC Value
0V	0x000
Vref/0x3FE	0x001
Vref	0x3FE
Vref + (Vref/0x3FE)	0x3FF

4.7.2 ADC Result Justification

The 10 bits of the ADC value can be mapped to the 16 bits of the ADCH/ADCL register pair in three different ways, as shown in Table 4-10. In this table, the numbers in the cells represent bit positions in the 10-bit ADC value, Z represents zero (as opposed to bit position 0), and -9 represents the inversion of bit position 9.

Table 4-11 Justification of the ADC Value

Mode	ADCH Register Bits								ADCL Register Bits							
	7	**6**	**5**	**4**	**3**	**2**	**1**	**0**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Left Justified	9	8	7	6	5	4	3	2	1	0	Z	Z	Z	Z	Z	Z
Right Justified	Z	Z	Z	Z	Z	Z	9	8	7	6	5	4	3	2	1	0
Signed	-9	-9	-9	-9	-9	-9	-9	8	7	6	5	4	3	2	1	0

4.7.3 Using the A/D Converter

The following sequence is recommended:

1. Set the ADCTMR register to the correct value for the system clock speed.
2. Load the ADCCFG register to specify the channel and set the ADCGO bit. Setting the ADCGO bit enables and resets the ADC timer.
3. After a period of time (12 timer overflows = 20.8 µs) the conversion will complete, the ADCGO bit will be cleared, and the ADC timer will be disabled.
4. A timer-based interrupt service routine can detect or assume the ADCGO bit has been cleared and read the ADC value.
5. Another load to the ADCCFG register can then be used to start another conversion.

4.7.4 ADCTMR Register

The ADCTMR register is used to specify the number of system clock cycles required for a delay of 1736 ns, which is used to provide the 576 kHz (48 kHz × 12) clock period reference clock for the A/D converter.

At a system clock frequency of 100 MHz, the timer register should be set to 174 (100 MHz/0.576 MHz). The minimum value that may be loaded into the ADCTMR register is 2, so the system clock must be at least 24 times the ADC sampling frequency for the ADC to function.

4.7.5 ADCCFG Register

7	6 5	4	3	2 0
ADCREF	ADCJST	Reserved	ADCGO	ADCS2:0

Name	Description
ADCREF	A/D converter reference voltage select 0 = AVdd is the reference voltage 1 = RG3 port pin is used to receive an external reference voltage
ADCJST	A/D converter result justification mode select (see Table 4-11) 00 = Right justified 01 = Signed 10 = Left justified 11 = Reserved
ADCGO	A/D converter GO/DONE bit 0 = When the last conversion has completed, this bit reads as 0. Clearing this bit terminates any conversion in progress, in which case the data must be ignored. 1 = Write 1 to begin a new conversion. While the conversion is in progress, this bit reads as 1.

Name	Description
ADCS2:0	A/D converter input channel select 000 = Port pin RG0 001 = Port pin RG1 010 = Port pin RG2 011 = Port pin RG3 100 = Port pin RG4 101 = Port pin RG5 110 = Port pin RG6 111 = Port pin RG7

4.8 Comparator

The IP2022 has an on-chip analog comparator which uses alternate functions of the RG0, RG1, and RG2 port pins. The RG1 and RG2 pins are the comparator negative and positive inputs, respectively, while the RG0 pin is the comparator output pin. To use the comparator, software must program the port direction register (RGDIR) so that RG1 and RG2 are inputs. RG0 may be set up as a comparator output pin.

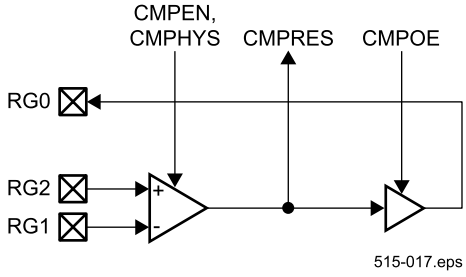


Figure 4-14 Analog Comparator Block Diagram

The comparator enable bits are cleared on reset, which disables the comparator. To avoid drawing additional current during power-down mode, the comparator should be disabled before entering power-down mode. A 50 mV hysteresis is applied between the inputs, when the CMPHYS bit is set in the CMPCFG register.

4.8.1 CMPCFG Register

7	6	5	4	3	1	0
CMPEN	CMPOE	CMPHYS	CMPMOD	Reserved		CMPRES

Name	Description
CMPEN	Comparator enable bit 0 = Comparator disabled 1 = Comparator enabled
CMPOE	Comparator output enable bit 0 = Comparator output disabled. 1 = Comparator output enabled on port pin RG0.
CMPHYS	Comparator hysteresis enable bit 0 = Hysteresis disabled 1 = Hysteresis enabled
CMPMOD	Comparator mode bit 0 = Normal mode 1 = Squelch or comparator mode for Ethernet
CMPRES	Comparator result (read-only) 0 = RG2 voltage > RG1 1 = RG1 voltage > RG2

4.9 Linear Feedback Shift Register

Four linear feedback shift register (LFSR) units provide hardware support for the computation-intensive inner loops of algorithms commonly used in data communications, such as:

- Cyclic Redundancy Check (CRC)
- Data Scrambling
- Data Whitening
- Encryption/Decryption
- Hashing

The LFSR units implement a programmable architecture, which can be adapted for algorithms used by the Bluetooth, Ethernet, Homeplug, HomePNA, HomeRF, IEEE 802.11, and USB communication protocols. Table 4-12 shows the LFSR configurations used to support these protocols. Figure 4-15 is a block diagram of the LFSR architecture.

Table 4-12 LFSR Configurations for Various Protocols

Proto-col	Subfunc-tion	D0 In	Feed-back	D Out
USB	CRC16	Din^D15	Din^D15	
	CRC5	Din^D4	Din^D4	
Ether-net	CRC32	Din^D31	Din^D31	
	Scrambler	Din^D17^D22		Din^D17^D22
	Descram-bler	Din		Din^D17^D22

Table 4-12 LFSR Configurations for Various Protocols (continued)

Proto-col	Subfunc-tion	D0 In	Feed-back	D Out
Home-Plug	CRC8	Din^D7	Din^D7	
	Scrambler	D6^D3		Din^D6^D3
Home-PNA	CRC8	Din^D7	Din^D7	
	CRC16	Din^D15	Din^D15	
	Scrambler	D17^D22		Din^D17^D22
802.11	CRC32 (FCS)	Din^D31	Din^D31	
	CRC16 (HEC)	Din^D15	Din^D15	
	Data Whitening	D3^D6		Din^D3^D6
	CRC16 (CRC)	Din^D15	Din^D15	
	Scrambler	Din^D3^D6		Din^D3^D6
	Descram-bler	Din		Din^D3^D6
Home-RF	CRC	Din^D31	Din^D31	
	Scrambler	Din		Din^D3^D8
	Descram-bler	Din		Din^D3^D8

Table 4-12 LFSR Configurations for Various Protocols (continued)

Proto-col	Subfunc-tion	D0 In	Feed-back	D Out
Blue-tooth	FEC	Din^D4	Din^D4	
	HEC	Din^D7	Din^D7	
	CRC	Din^D15	Din^D15	
	Data Whitening	D6	D6	Din^D6
	Encryption	Din^D8^D12^D20^D25		D24
		Din^D12^D16^D24^D31		D24
		Din^D4^D24^D28^D33		D32
		Din^D4^D28^D36^D39		D32

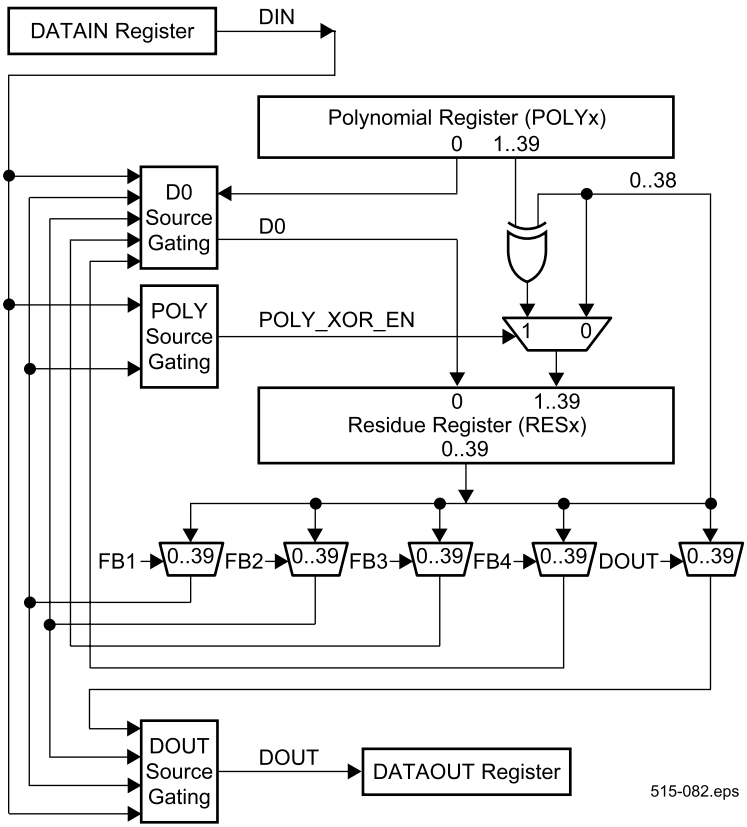


Figure 4-15 LFSR Block Diagram

The 40-bit residue register and its surrounding circuits are the computational core of an LFSR unit. On every clock cycle, 39 output bits from the register are available at the input for performing a shift operation or a polynomial add/subtract-and-shift operation. Four 40-bit multiplexers at the output of the residue

register allow selecting up to four terms of the register for feedback into the input (D0), polynomial operation control (POLY_XOR_EN), and output (DOUT) bit streams. A fifth multiplexer is only used for generating the output bit stream.

The polynomial and residue registers are mapped as five 8-bit registers. The mapping of the residue register is controlled by the ML_OUT bit of the LFSRCFG3 register, as shown in Figure 4-16.

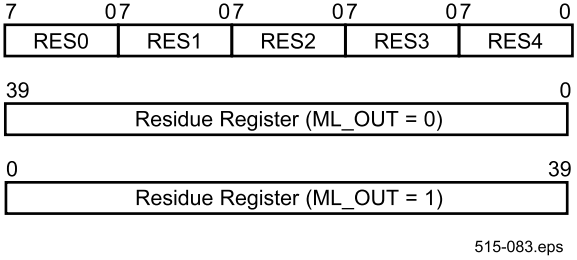


Figure 4-16 Mapping of the Residue Register

Input data is shifted out of the 16-bit DATAIN register, which can be programmed to provide the data LSB-first or MSB-first. Output data is shifted LSB-first into the 16-bit DATAOUT register.

A 32-bit RESCMP register (not shown) can be used to compare the result in the residue register against an expected value. When ML_OUT is set, residue register bits 0:31 are compared against RESCMP bits 0:31. When ML_OUT is clear, residue register bits 39:8 are compared to RESCMP bits 0:31, respectively. If there are bits in the residue register which do not participate in the programmed LFSR operation, be sure that the corresponding bits

in the RESCMP register are initialized to the same values as these non-participating bits.

Each LFSR unit has three configuration registers (LFSRCFG1, LFSRCFG2, and LFSRCFG3) for various control and status bits. The HL_TRIGGER bit in the LFSRCFG3 register controls whether operation of the LFSR unit is triggered by a write to the high byte or the low byte of the DATAIN register (i.e. DATAINH or DATAINL). The operation then proceeds for some number of cycles programmed in the SHIFT_COUNT3:0 field of the LFSRCFG1 register. Completion of the operation is indicated when the DONE bit in the LFSRCFG1 register is set. (Alternatively, software can wait an appropriate number of cycles before reading the result.)

An autoloading option is available for each LFSR unit to load the DATAIN register automatically when the SxRBUF register of the corresponding SERDES unit is loaded. LFSR0 and LFSR2 are paired with SERDES1, and LFSR1 and LFSR3 are paired with SERDES2.

Three registers in data memory are used to access the LFSR register banks:

Table 4-13 LFSR Registers in Data Memory

Address	Name	Description
0x23	LFSRH	High data byte
0x27	LFSRL	Low data byte
0x2B	LFSRA	Address register

The LFSRA register is loaded to point to a specific LFSR unit and a register pair within the unit. The LFSRA register has the following format:

7 4	3 0
UNIT3:0	INDEX3:0

Only 0, 1, 2, and 3 are valid as the UNIT. The valid encodings for the INDEX are shown in Table 4-14.

Table 4-14 LFSRA Register INDEX Encoding

INDEX3:0	High Byte (LFSRH)	Low Byte (LFSRL)
0x0	DATAINH	DATAINL
0x1	DATAOUTH	DATAOUTL
0x2	FB2	FB1
0x3	LFSRCFG2	RES4
0x4	RES3	RES2
0x5	RES1	RES0
0x6	FB4	FB3
0x7	LFSRCFG3	DOUT
0x8	LFSRCFG1	POLY4
0x9	POLY3	POLY2
0xA	POLY1	POLY0
0xB	RESCMP3	RESCMP2
0xC	RESCMP1	RESCMP0

The LFSR registers do not support consecutive read-modify-write operations. For example, the following instruction sequence loads unpredictable values:

```
clrb lfsrh,7
clrb lfsrh,4
```

4.9.1 LFSRCFG1 Register

7	6	5	4	3 0
Reserved	SET_RES	DONE	CMP_RES	SHIFT_COUNT3:0

Name	Description
SET_RES	Set all bits in residue register (write only) 0 = No effect (this bit always reads as 0) 1 = Set all bits
DONE	LFSR operation complete (read only) 0 = LFSR unit busy 1 = Last operation has completed
CMP_RES	Residue register comparison bit (read only) 0 = Last LFSR operation result did not match contents of RESCMP register 1 = Last result matched RESCMP contents
SHIFT_COUNT3:0	LFSR operation shift count. Load this field with N for an operation of N+1 shifts.

4.9.2 LFSRCFG2 Register

7	6	5	4	3	2	1	0
DOUT_DOUT_EN	DIN_DOUT_EN	FB1_DOUT_EN	FB2_DOUT_EN	DIN_D0_EN	FB1_D0_EN	FB2_D0_EN	DATA_IN_POLYXOR_EN

Name	Description
DOUT_DOUT_EN	Enable DOUT multiplexer output in source gating for DOUT node 0 = DOUT multiplexer output is not used 1 = DOUT multiplexer output is XORed with other enabled inputs
DIN_DOUT_EN	Enable DIN signal in source gating for DOUT node 0 = DIN is not used 1 = DIN is XORed with other enabled inputs

Name	Description
FB1_DOUT_EN	Enable FB1 signal in source gating for DOUT node 0 = FB1 is not used 1 = FB1 is XORed with other enabled inputs
FB2_DOUT_EN	Enable FB2 signal in source gating for DOUT node 0 = FB2 is not used 1 = FB2 is XORed with other enabled inputs
DIN_D0_EN	Enable DIN signal in source gating for D0 node 0 = DIN is not used 1 = DIN is XORed with other enabled inputs
FB1_D0_EN	Enable FB1 signal in source gating for D0 node 0 = FB1 is not used 1 = FB1 is XORed with other enabled inputs
FB2_D0_EN	Enable FB2 signal in source gating for D0 node 0 = FB2 is not used 1 = FB2 is XORed with other enabled inputs

Name	Description
FB1_DOUT_EN	Enable FB1 signal in source gating for DOUT node 0 = FB1 is not used 1 = FB1 is XORed with other enabled inputs
FB2_DOUT_EN	Enable FB2 signal in source gating for DOUT node 0 = FB2 is not used 1 = FB2 is XORed with other enabled inputs
DIN_D0_EN	Enable DIN signal in source gating for D0 node 0 = DIN is not used 1 = DIN is XORed with other enabled inputs
FB1_D0_EN	Enable FB1 signal in source gating for D0 node 0 = FB1 is not used 1 = FB1 is XORed with other enabled inputs
FB2_D0_EN	Enable FB2 signal in source gating for D0 node 0 = FB2 is not used 1 = FB2 is XORed with other enabled inputs

Name	Description
DATA_IN_POLYXOR_EN	Enable DIN signal in source gating for POLY_XOR_EN node 0 = DIN is not used 1 = DIN is XORed with other enabled inputs

4.9.3 LFSRCFG3 Register

7	6	5	4	3	2	1	0
Reserved		AUTOLOAD_EN	ML_OUT	ML_IN	HL_TRIGGER	FB3_D0_EN	FB4_D0_EN

Name	Description
AUTOLOAD_EN	Enable autoloading DATAIN register when SxRBUF register of corresponding SERDES unit is loaded 0 = Autoload disabled 1 = Autoload enabled

Name	Description
ML_OUT	Specify shift direction of the residue register, mapping of the residue register to the RESx registers, and mapping of the residue register to the RESCMP comparator 0 = Data is shifted into MSB and out of LSB 1 = Data is shifted into LSB and out of MSB
ML_IN	Specify shift direction of DATAIN register to DIN node 0 = Data is shifted LSB-first to DIN 1 = Data is shifted MSB-first to DIN
HL_TRIGGER	Specify trigger condition for starting LFSR operation 0 = Trigger on loading DATAINL (low byte) 1 = Trigger on loading DATAINH (high byte)
FB3_D0_EN	Enable FB3 signal in source gating for D0 node 0 = FB3 is not used 1 = FB3 is XORed with other enabled inputs
FB4_D0_EN	Enable FB4 signal in source gating for D0 node 0 = FB4 is not used 1 = FB4 is XORed with other enabled inputs

4.9.4 DATAIN Register

The 8-bit DATAINH and DATAINL registers together comprise the 16-bit DATAIN register. For LFSR0 and LFSR2, the AUTOLOAD_EN bit in the LFSRCFG3 register can be used to enable automatic loading from SERDES1. For LFSR1 and LFSR3, the AUTOLOAD_EN bit in the LFSRCFG3 register can be

used to enable automatic loading from SERDES2. The HL_TRIGGER bit in the LFSRCFG3 register controls whether loading the DATAINH or DATAINL register triggers the start of the LFSR operation. The ML_IN bit in the LFSRCFG3 register controls whether data is shifted MSB-first or LSB-first from the DATAIN register to the DIN node.

4.9.5 DATAOUT Register

The 8-bit DATAOUTH and DATAOUTL registers together comprise the 16-bit DATAOUT register. Data shifted out of the residue register is shifted LSB-first into the DATAOUT register.

4.9.6 DOUT Register

The DOUT register controls a 40:1 multiplexer on the residue register outputs. It selects a term which can be used in the source gating for the DOUT bit stream.

4.9.7 FBx Registers

The four FBx registers control four 40:1 multiplexers on the residue register outputs. They select feedback terms which can be used in the source gating for the D0, POLY_XOR_EN, and DOUT bit streams.

4.9.8 POLYx Registers

The five POLYx registers hold the 40-bit polynomial used in the LFSR operation.

4.9.9 RESx Registers

The five RESx registers hold the 40-bit residue used in the LFSR operation. The ML_OUT bit controls the mapping of the residue register to the RESx registers, as shown in Figure 4-16. The residue register can be initialized to all ones by setting the SET_RES bit in the LFSRCFG1 register.

4.9.10 RESCMPx Registers

The four RESCMPx registers hold a 32-bit value for comparison with the contents of the residue register. After an LFSR operation is completed, the CMP_RES bit in the LFSRCFG1 register indicates whether the result of the operation matched the 32-bit value. When the ML_OUT bit in the LFSRCFG3 register is clear, bits 39:8 of the residue register are compared against bits 0:31 of the RESCMP register. When ML_OUT is set, bits 0:31 of the residue register are compared against bits 0:31 of RESCMP.

4.10 Parallel Slave Peripheral

The Parallel Slave Peripheral allows the IP2022 to operate as an 8- or 16-bit slave to an external device, much like a memory chip. Alternate functions of Port C and Port D are used for transferring data, and alternate functions of Port B are used for control signals. Figure 4-17 shows the connections between an external master and the Parallel Slave Peripheral interface.

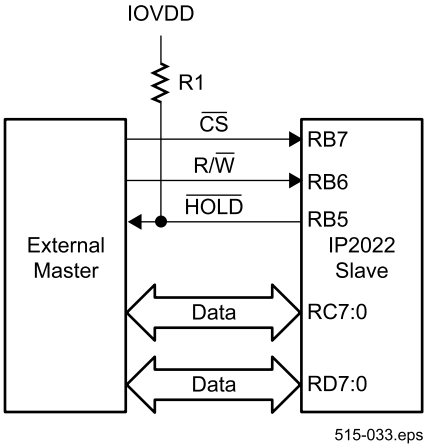


Figure 4-17 Parallel Slave Peripheral Interface

To read or write through the Parallel Slave Peripheral interface, the external master asserts the chip select ($\overline{CS}$) signal low. This signal is an alternate function of port pin RB7. The direction of transfer is indicated by the R/$\overline{W}$ signal, which is an alternate function of port pin RB6. When the R/$\overline{W}$ signal is high, the master is reading from the slave. When the R/$\overline{W}$ signal is low, the master

is writing to the slave. Optionally, a $\overline{\text{HOLD}}$ signal may be enabled as an alternate function of port pin RB5. Assertion of $\overline{\text{HOLD}}$ indicates to the external master that the Parallel Slave Peripheral interface is not ready to allow the data transfer to complete. The $\overline{\text{HOLD}}$ signal should have an external pullup resistor (R1 = 10K Ω is recommended). The $\overline{\text{CS}}$ signal must not be allowed to float.

Internally, an interrupt is generated when $\overline{\text{CS}}$ is asserted. Software then reads the Port C, Port D, or both if the data transfer is a write from the external master. If the data transfer is a read, software writes the data to the port or ports. Finally, software releases the $\overline{\text{HOLD}}$ signal (if enabled) by writing to the PSPRDY bit in the PSPCFG register.

The Parallel Slave Peripheral does not generate interrupts by itself. Software is required to enable port pin RB7 (the $\overline{\text{CS}}$ input) as a falling-edge interrupt input for the Parallel Slave Peripheral to function. The $\overline{\text{CS}}$ signal must go high, then back low, for each data transfer. RB6 (the R/$\overline{\text{W}}$ input) must also be configured as an input. The setting in the RBDIR register for RB5 (the $\overline{\text{HOLD}}$ output) is overridden by the programming of the Parallel Slave Peripheral.

The PSPCFG register is used to enable the Parallel Slave Peripheral, select which ports are used for data transfer, enable the $\overline{\text{HOLD}}$ output, and release the $\overline{\text{HOLD}}$ output when the data transfer is ready to complete.

4.10.1 PSPCFG Register

7	6	5	4	3 … 0
PSPEN2	PSPEN1	PSPHEN	PSPRDY	Reserved

Name	Description
PSPEN2	Port D enable bit 0 = Port D is available for general-purpose I/O 1 = Port D is configured for the Parallel Slave Peripheral interface
PSPEN1	Port C enable bit 0 = Port C is available for general-purpose I/O 1 = Port C is configured for the Parallel Slave Peripheral interface (the Parallel Slave Peripheral will immediately override the RCDIR register)
PSPHEN	$\overline{\text{HOLD}}$ output enable bit 0 = $\overline{\text{HOLD}}$ output disabled. Port pin RB5 available for general-purpose I/O. 1 = $\overline{\text{HOLD}}$ output enabled on port pin RB5 (the Parallel Slave Peripheral will immediately override bit 5 of the RBDIR register)
PSPRDY	Ready bit 0 = This bit always reads as zero 1 = Write 1 to release $\overline{\text{HOLD}}$ when the IP2022 is ready to allow the data transfer to complete

4.11 External Memory Interface

Port C and Port D can also be used for a parallel interface for up to 128K bytes of external memory, as shown in Figure 4-18. Port C implements the high address bits, and Port D is multiplexed between data and the low address bits. A level-triggered 8-bit latch (TI part number SN74AC573 or equivalent) is required for demultiplexing.

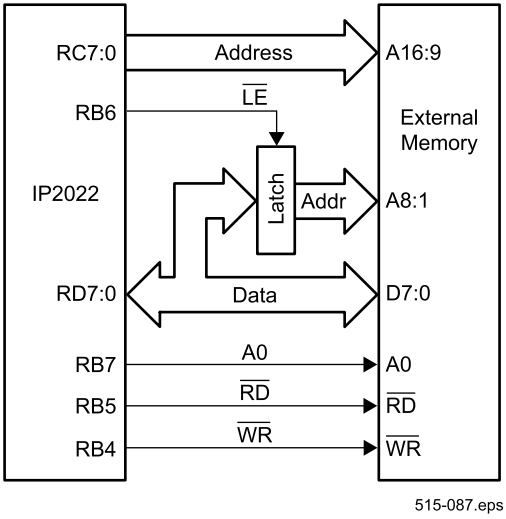


Figure 4-18 External Memory Interface

External memory is accessed as 16-bit words at word-aligned byte addresses 0x1000000 to 0x103FFFE, as shown in Figure 4-19. External memory can only be accessed through the current ADDRX/ADDRH/ADDRL pointer using the **`iread`**/**`ireadi`** and

`iwrite`/**`iwritei`** instructions. Programs cannot execute directly out of external memory, and commands on the ISD/ISP interfaces cannot directly access external memory. Like data memory, however, external memory can be accessed over the ISD/ISP interface by executing instructions which move data between memory and the W register.

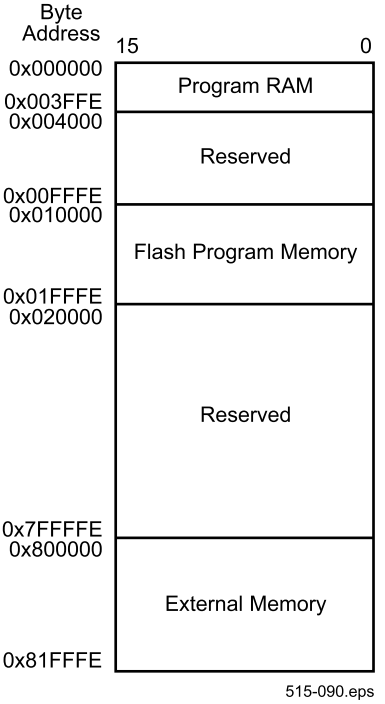


Figure 4-19 External Memory Map

Software is responsible for inserting a one-instruction delay between changing the address (i.e. the contents of the ADDRSEL, ADDRX, ADDRH, or ADDRL registers) and executing the **`iread`**/**`ireadi`** or **`iwrite`**/**`iwritei`** instruction, if required by the timing of the external latch. Table 4-15 shows the timing specifications which the register must meet for operation without delay insertion.

Table 4-15 External Latch Timing Specifications

Parameter	Value (ns)
Minimum LE pulse width	7
Setup time before LE falling edge	4
Hold time after LE falling edge	2
Input to output delay, transparent mode	15

For zero wait-state access, the external memory must meet the access time specification shown in Table 4-16. Slower memories can be accommodated by programming wait states in the EMCFG register. Software is responsible for allowing the memory cycle to complete before reading the DATAH/DATAL registers.

Table 4-16 SRAM Access Time Specification

CPU Core Clock Frequency (MHz)	SRAM Access Time (ns)
80	35
100	25
120	25
150	12/15/20*

* Depends on minimum $\overline{WR}$ pulse width specification.

A read cycle to external memory has the timing shown in Figure 4-20. Write cycle timing is shown in Figure 4-21. All external memory cycles are 16-bit transfers, with the low byte (A0 = 0) followed by the high byte (A0 = 1).

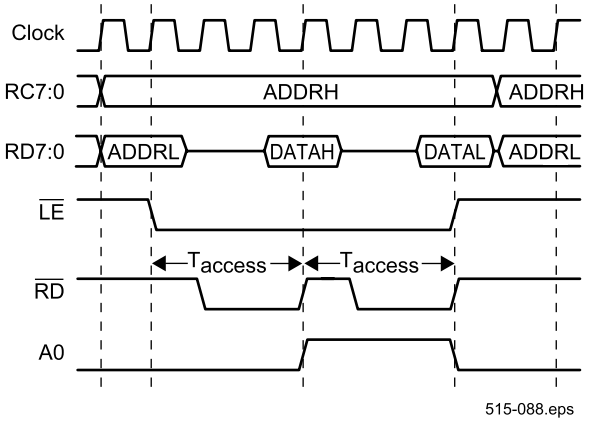


Figure 4-20 Read Cycle

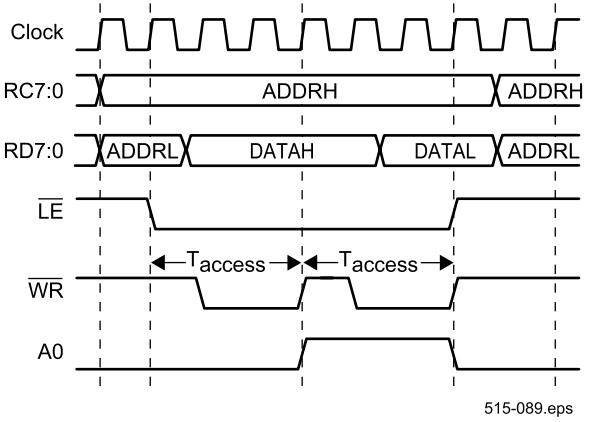


Figure 4-21 Write Cycle

4.11.1 EMCFG Register

7	6	5 3	2 0
EMEN	EMBRT	EMWRT2:0	EMRDT2:0

Name	Description
EMEN	Enable external memory interface 0 = Port C and Port D available for general-purpose I/O 1 = Port C and Port D used for external memory interface

Name	Description
EMBRT	Enable bus release wait state 0 = No wait state 1 = One wait state added between a read cycle followed by a write cycle
EMWRT2:0	$\overline{WR}$ pulse width, in CPU core clock cycles 000 = 1 001 = 2 010 = 3 011 = 4 100 = 5 101 = 6 110 = 7 111 = 8
EMRDT2:0	$\overline{RD}$ pulse width, in CPU core clock cycles 000 = 1 001 = 2 010 = 3 011 = 4 100 = 5 101 = 6 110 = 7 111 = 8

5.0 In-System Debugging

The IP2022 provides on-chip hardware for interface to debugging tools. This eliminates any need for special “bond-out” chips for software development, which may not share the exact electrical characteristics of the target chip used for high-volume production. By providing a debugging facility that works with the target chip installed in the actual system being developed, the risk of minor incompatibilities turning into major production delays is eliminated.

The in-system debugging (ISD) hardware provides the following capabilities:

- Basic control of processor in a target system such as reset and reading device ID.
- Visibility of all chip registers and the ability to modify their contents.
- Ability to set and clear breakpoints.
- Single-step, run, and stop operations.

An industry-standard SPI bus is used for in-system debugging. The SPI bus is also used for in-system programming (ISP), as described in Chapter 6. The dedicated SPI bus is not available for any other uses. If an application requires an SPI bus for another purpose, the two serializer/deserializer (SERDES) units are available for implementing SPI, or other popular serial interface standards.

5.1 Debugging Modes

When using a debugger, the IP2022 has two modes:

- *Run mode*—free-running program execution
- *Break mode*—CPU stopped

In Break mode, the instruction pipeline is flushed and NOP instructions are executed continuously without incrementing the PC, which has the effect of stalling program execution. There are three conditions which cause the IP2022 to enter Break mode:

- Issuing an INITIALIZE or BREAK command through the ISD/ISP interface.
- Triggering a hardware breakpoint set up using the SET_BKPT command.
- Executing a **`break`** instruction (an IP2022 instruction, not to be confused with the BREAK command on the ISD/ISP interface).

If Break mode is entered by triggering the hardware breakpoint or executing the **`break`** instruction, the CPU is stalled with the PC pointing to the instruction that triggered entry into Break mode. When triggered by a hardware breakpoint, the instruction at the breakpoint address is not executed before entering Break mode. Unlike interrupts, no registers are saved or restored on entry or exit from Break mode.

The CPU core clock remains unchanged on entry into Break mode, except in the following cases:

- An INITIALIZE or BREAK command was issued through the ISD/ISP interface, and the system clock is off.
- An INITIALIZE or BREAK command was issued through the ISD/ISP interface, and the system clock is derived from RT-CLK (rather than the OSC clock).
- An INITIALIZE or BREAK command was issued through the ISD/ISP interface, and the CPU core clock is off.

In these cases, the CPU core clock is changed to the OSC clock (i.e. PLL bypass) with a clock divisor of 1. This change is not reflected in the SPDREG register, which will continue to indicate the speed before entry into Break mode.

If a flash memory read, write, or erase operation is in progress, the operation will continue in Break mode. The debugger must poll the FBUSY bit to ensure the operation is complete before making any change to the CPU core speed. If the debugger changes the CPU core speed before exiting Break mode (by issuing a STEP or RUN command), the speed must be restored to its value on entry into Break mode.

If the CPU core clock is off when Break mode is entered, the CPU will execute one instruction before entering Break mode. There is no mechanism through the ISD/ISP interface for returning to Run mode with the CPU core clock shut off, because the STEP and RUN commands cannot be issued unless the CPU core clock is running.

The clock is suspended to the following peripherals in Break mode:

- Real-Time Timer
- Timer 0
- Timer 1
- Timer 2
- ADC Timer
- Watchdog Timer (only if the ISD/ISP interface is open)

The timers used for flash memory read, write, and erase operations continue to function in Break mode. If the Watchdog Timer is enabled, it will be running after the ISD/ISP interface is opened until the DEBUG_RESET command is issued. If the Watchdog Timer overflows during this interval, the start-up sequence should be repeated.

5.2 ISD/ISP Interface Protocol

The ISD/ISP interface protocol consists of transmitting a command byte followed by a 16-bit operand word while simultaneously receiving a command acknowledge byte (for the previous command) followed by a 16-bit return value, as shown in Figure 5-1. Command and command acknowledge formats have a fixed length of three bytes. There are two types of commands: commands that require a response such as the IREAD command (read from program memory, not to be confused with the **`iread`** instruction), and commands that do not require any data to be returned. The SPI bus requires that the same number of bits are

transmitted and received, so commands that do not return data return zero.

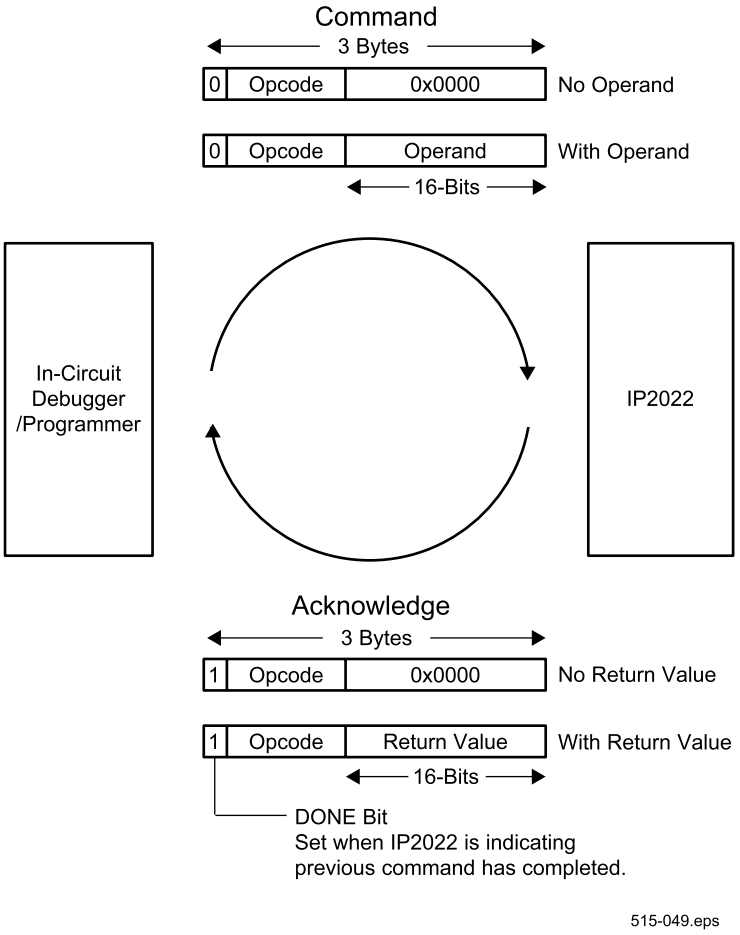


Figure 5-1 Command/Acknowledge Formats

Because the IP2022 has to send something back simultaneously with receiving each command, it echoes the previous command and sends it back with its MSB (bit 7 of the command byte) set to signal completion of the previous command. The very first command acknowledgement returns 0x800000 because there is no previous command, i.e. the acknowledgment has the DONE bit set and a return value that is all zero.

Commands that do not require any data to be returned generate command acknowledgements with zero in the return value. However, most command acknowledgements have a return value that consists of the STATUS and W registers concatenated together, with the STATUS register occupying the high byte of the return value and the W register occupying the low byte. Because there is no ISD/ISP command for reading the IP2022 data memory, data memory must be read using instructions executed on the IP2022. By using CPU instructions to read data memory into the W register, the data becomes visible on the ISD/ISP interface in the command acknowledgement return value.

A few ISD/ISP commands are acknowledged with the ISP_STATUS register as the return value. The format of this register is shown in Section 5.2.1.

5.2.1 ISP_STATUS Register

15 … 6	5	4	3	2	1	0
Reserved	FBUSY	Rsrv.	IBREAK	HBREAK	SBREAK	RUN

Name	Description
FBUSY	Flash memory status flag bit 0 = Flash memory is available for reading and writing 1 = Flash memory is busy with previous operation
IBREAK	CPU Break mode from ISD/ISP BREAK command flag bit 0 = No ISD/ISP BREAK command has been issued since the CPU was in Run mode 1 = CPU is in Break mode caused by BREAK command
HBREAK	CPU Break mode from hardware breakpoint flag bit 0 = No hardware breakpoint has been encountered since the CPU was in Run mode 1 = CPU is in Break mode caused by hardware break-point (i.e. PC matches last SET_BKPT command)
SBREAK	CPU Break mode from **`break`** instruction flag bit 0 = No break instruction has been executed since the CPU was in Run mode 1 = CPU is in Break mode caused by executing **`break`** instruction
RUN	CPU Run mode status flag bit 0 = CPU is not in Run mode 1 = CPU is in Run mode

5.3 ISD/ISP Commands

The ISD/ISP interface has two states, open and closed. Before any commands are accepted, the interface must be opened with the OPEN command. When the interface is closed, the IP2022 does not respond to any other commands. The CLOSE command can be used during testing to make sure that the debugger/programmer is not interfering with the operation of the target system. OPEN and CLOSE commands are also used to connect and disconnect the debugger/programmer while the target system is running. The ISD/ISP connector may only be attached or removed while the ISD/ISP interface is closed, to ensure that no spurious commands are issued to the IP2022.

After being opened, a blank device must be initialized with the INITIALIZE command. This selects various modes normally specified in the FUSE0 register, including critical operating modes such as clock source selection and the prescaler and postscaler clock divisors. The OSC clock must be running before using the debugging or programming features, however the SPI clock (TSCK) is the only clock required to be operational before issuing the INITIALIZE command. The operand word issued with the INITIALIZE command has the same format as the FUSE0 register. The operand word is loaded into a temporary register that controls the IP2022 until it is reset, as shown in Figure 5-2.

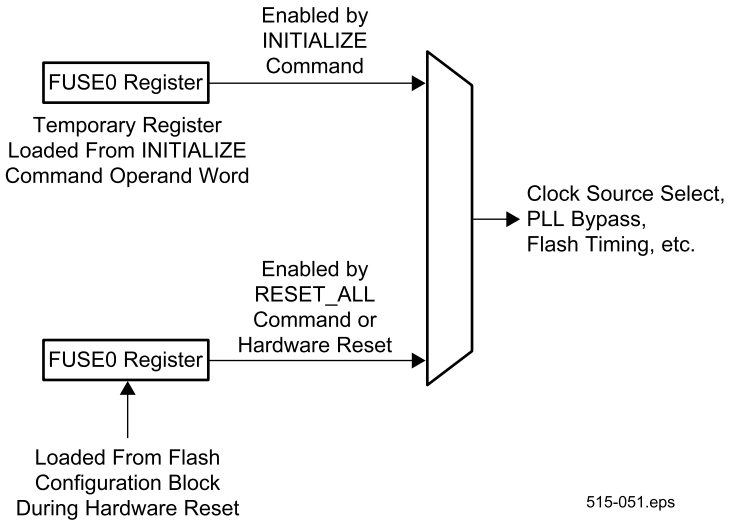


Figure 5-2 FUSE0 Register Selection

A typical sequence of commands is shown below:

1. *OPEN*—Begin ISD/ISP communication.
2. *INITIALIZE*—Loads and enables a temporary copy of the FUSE0 register.
3. *DEBUG_RESET*—This does not affect the clock selection circuit; acts as BREAK command implicitly.
4. *BULK_ERASE*—Erase all flash memory bits, optionally including the configuration block.
5. *ADDR_HI*—Load high word of address pointer.
6. *ADDR_LO*—Load low word of address pointer.

7. *FWRITE*—Write flash memory and increment address pointer. A series of FWRITE commands can be used to write all of flash memory, one location at a time.
8. *RESET_ALL*—Resets the chip. After executing this command, the FUSE0 register initialized from the configuration block will be used. The ISD/ISP interface is closed and the IP2022 begins execution at the reset vector.
9. *OPEN*—If there are further ISD/ISP commands and RESET_ALL was used earlier, an OPEN command is required before issuing any more commands.

The ISD/ISP command set is listed in Table 5-1. A command from the debugger/programmer is sent with an MSB of zero (which corresponds to the position of the DONE bit in the command response). The MSB is followed by seven opcode bits and an additional 16 bits that may hold an operand. The command acknowledgement consists of the DONE bit followed by the opcode of the command being acknowledged and an additional 16 bits that may hold a return value.

Table 5-1 ISD/ISP Command Set

Command Name	Opcode (1.), Operand (2.), Return Value (3.)	Description
ADDR_HI	1. `0001000` 2. Address pointer (high word) 3. Address pointer (high word)	Run/Break mode: Set high word of address pointer.
ADDR_LO	1. `0001001` 2. Address pointer (low word) 3. Address pointer (low word)	Run/Break mode: Set low word of address pointer.
BREAK	1. `0011001` 2. None 3. STATUS || W	Run/Break mode: Trigger CPU debug interrupt. The BREAK command suspends the Watchdog Timer. The BREAK command must be used to ensure that the CPU is stopped prior to programming the flash memory.

Table 5-1 ISD/ISP Command Set (continued)

Command Name	Opcode (1.), Operand (2.), Return Value (3.)	Description
BULK_ERASE	1. `0010011` 2. None 3. STATUS || W	Break mode: Erase all program RAM and program flash memory bits. If bit 16 of the address pointer (bit 0 of the operand to the ADDR_HI command) is 1, code-protect mode is off (i.e. $\overline{CP}$ = 1), and all of the locations in program RAM and program flash memory are 0xFFFF, then the configuration block is also erased. The operation takes about 200 milliseconds to complete.
CLOSE	1. `0111110` 2. None 3. ISP_STATUS	Run/Break mode: Close communication. The ISD/ISP interface must be closed before disconnecting the interface cable.

Table 5-1 ISD/ISP Command Set (continued)

Command Name	Opcode (1.), Operand (2.), Return Value (3.)	Description
DEBUG_RESET	1. 0000111 2. None 3. STATUS || W	Run/Break mode: Operates like the RESET_ALL command, except: • Does not affect the clock selection. It acts as a BREAK command if the CPU is running. Use this command after the INITIALIZE command to reset the CPU which may have been affected by clock glitches during the clock switch. • Does not close the ISD/ISP interface. • Does not cause the processor to enter Run mode automatically after completion, so it can be followed immediately by the STEP or RUN commands.
DEVICE_ID	1. 0000001 2. None 3. ID	Run/Break mode: Read device ID. For the IP2022, ID = 0x2022.
DEVICE_VER	1. 0000010 2. None 3. Version	Run/Break mode: Read device version.

Table 5-1 ISD/ISP Command Set (continued)

Command Name	Opcode (1.), Operand (2.), Return Value (3.)	Description
FCFG	1. 0010101 2. 0x00 || FCFG 3. 0x00 || FCFG	Run/Break mode: Load the FCFG register. Command is unaffected by code-protect mode (i.e. $\overline{CP}$ bit = 0).
FERASE	1. 0010010 2. None 3. STATUS || W	Break mode: Erase flash block that includes location addressed by address pointer. Command is acknowledged but no operation occurs in code-protect mode (i.e. $\overline{CP}$ bit = 0). Command is unaffected by flash write-protect mode (i.e. $\overline{FWP}$ bit = 0).
INITIALIZE	1. 0000101 2. FUSE0 register 3. FUSE0 register	Break mode: Load temporary FUSE0 register, and enable selection of this register for device control. This command is used to override configuration block programming of device default parameters such as clock source, type, etc. Always use the BREAK and DEBUG_RESET or RESET_ALL commands after executing an INITIALIZE command.

Table 5-1 ISD/ISP Command Set (continued)

Command Name	Opcode (1.), Operand (2.), Return Value (3.)	Description
INST_LO	1. 0100000 2. CPU Instruction 3. STATUS || W	Break mode: Instruction specified by the opcode is executed.
IREAD	1. 0010000 2. None 3. Data	Break mode: Read word from program flash memory or program RAM and increment the low byte of the address pointer. The high byte of the address is not affected when the low byte rolls over to 0. In code-protect mode (i.e. $\overline{CP}$ bit = 0), only the configuration block (i.e. FUSE0, FUSE1, and TRIM0 bits) is available for reading. All other flash memory bits read as zero.
IWRITE	1. 0010001 2. Data 3. None	Break mode: Write word to program flash memory or program RAM and increment address pointer. The high byte of the address is not affected when the low byte rolls over to 0. Command is acknowledged but no operation occurs in code-protect mode (i.e. $\overline{CP}$ bit = 0).

Table 5-1 ISD/ISP Command Set (continued)

Command Name	Opcode (1.), Operand (2.), Return Value (3.)	Description
NOP	1. `0000000` 2. None 3. STATUS || W	Run/Break mode: No operation. May be used to read response to an earlier command when there is no new command to be issued. Also, may be used to check for "chip live" verification.
OPEN	1. `0111111` 2. None 3. ISP_STATUS	Run/Break mode: Open communication performed after connecting to an external device.
RESET_ALL	1. `0000110` 2. None 3. STATUS || W	Run/Break mode: Resets the processor to a reset state; however the surrounding external logic is not affected, i.e., not reset. It is equivalent to power-on reset operation, and processor will begin execution at the reset vector. After this command, the processor is placed in Run mode. This command also performs an implicit CLOSE command.
RUN	1. `0011110` 2. None 3. STATUS || W	Break mode: Start or resume program execution.

Table 5-1 ISD/ISP Command Set (continued)

Command Name	Opcode (1.), Operand (2.), Return Value (3.)	Description
SET_BKPT	1. 0011000 2. Address 3. None	Break mode: Sets the breakpoint register to the address specified by the operand.
STATUS	1. 0000100 2. None 3. ISP_STATUS	Run/Break mode: Read the ISP/ISD status bits.
STEP	1. 0011101 2. None 3. STATUS || W	Break mode: Execute one instruction, then return to Break mode.

5.3.1 Data Transfer

An SPI bus connection between an SPI master and an SPI slave is like two shift registers connected in a circular configuration, as shown in Figure 5-3. Data shifted out of the most-significant bit of the SPI master shift register is driven on the TSI signal, which is shifted into the least-significant bit of the SPI slave shift register. Data shifted out of the most-significant bit of the SPI slave shift register is driven on the TSO signal, which is shifted into the least-significant bit of the SPI master shift register.

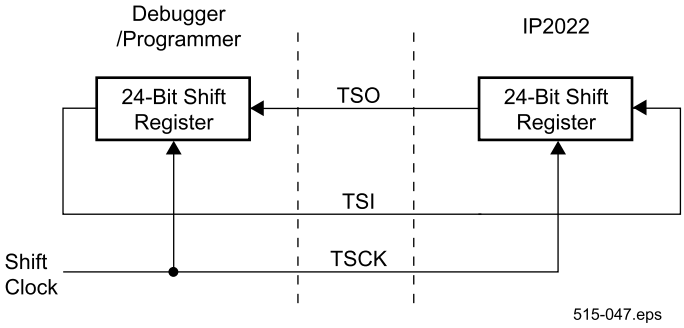


Figure 5-3 SPI Serial Data Transmission

Data is simultaneously transmitted and received on every transfer, as shown in Figure 5-4. The TSCK clock signal of the SPI bus synchronizes the shifting and sampling of the data on the two serial data lines, both of which occur on the rising edge of TSCK.

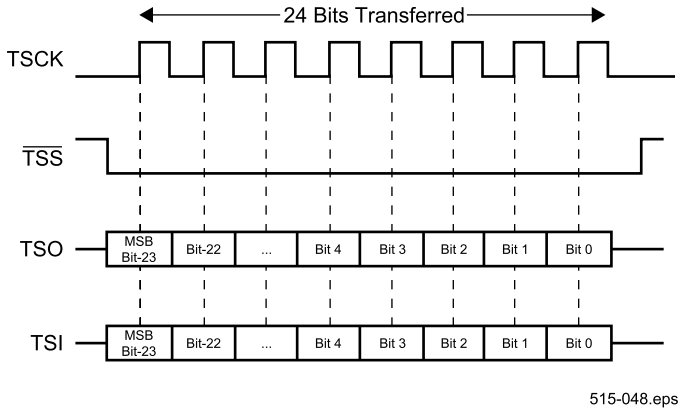


Figure 5-4 SPI Bus Timing Diagram

The ISD/ISP interface could be driven by an SPI bus master that sends three bytes for each command. The SPI slave select signal $\overline{\text{TSS}}$ must not be negated between each byte transfer, i.e. it must stay asserted for the whole command including the operand word. The $\overline{\text{TSS}}$ signal must be driven low before the command, stay low between the command byte and operand word, and must be released after the operand word. In other words, $\overline{\text{TSS}}$ falling indicates the start of a frame, and $\overline{\text{TSS}}$ rising indicates the end of a frame. The $\overline{\text{TSS}}$ slave select signal allows individual device selection, which could be used for the verification stage after gang programming multiple parts connected to the same programmer.

5.4 Host Connection to ISD/ISP Interface

The ISD/ISP interface could be connected to an inexpensive and already available interface, such as the parallel port of a PC driven under software control, as shown in Figure 5-5. This would not require any external hardware except a cable, but it will be slower than an interface that implements data serialization in hardware.

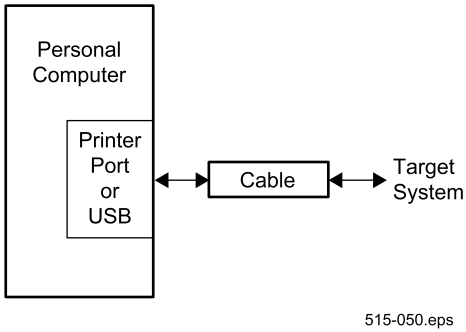


Figure 5-5 Simple ISD/ISP Interface

Using the printer or USB port interface to communicate with a Serial Debug Unit (SDU) with shift registers for data serialization, as shown in Figure 5-6, provides about an order of magnitude performance improvement over using programmed I/O on the printer port.

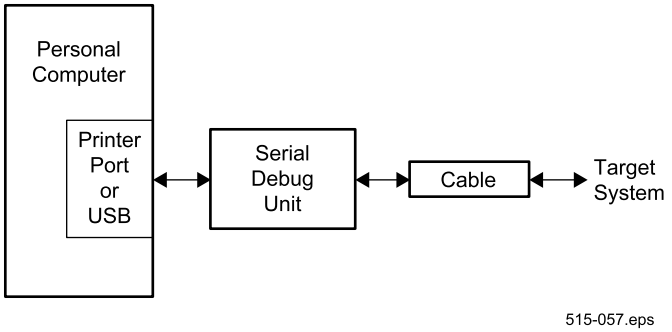


Figure 5-6 Serial Debug Unit

After sending a command, the debugger/programmer has to check that the command has completed before sending another command. There are several ways to do this:

- Allow sufficient time to elapse before sending another command.
- Issue a NOP command, and check the DONE bit in the command acknowledge frame.
- Poll the TSO signal, waiting for it to be sampled high. When the processor completes the requested command, the command byte with its most-significant bit (i.e. the DONE bit) set is loaded into the SPI shift register, which causes the TSO signal to be driven high. To make the state of the DONE bit visible on the TSO signal, the select line $\overline{\text{TSS}}$ must be asserted.

5.5 ISD/ISP Interface

There are options in the implementation of an SPI bus. The options chosen for the ISD/ISP interface are:

- *SPI Bus Slave*—The IP2022 is an SPI bus slave. The device programmer or debugging tool is the SPI bus master.
- *Clock Polarity and Phase Selection*—The clock polarity and phase options are chosen so that the clock signal is low when it is not active, and negative edges of the clock are used to output serial data.
- *Data Transfer Length*—Data transfers are always 24 bits in length.

The SPI bus consists of four signal lines: TSI, TSO, TSCK, and $\overline{\text{TSS}}$. In addition, the ISD/ISP interface includes a reset signal ($\overline{\text{TRST}}$), power connections (DVdd, IOVdd), and an oscillator clock input, as shown in Figure 5-7.

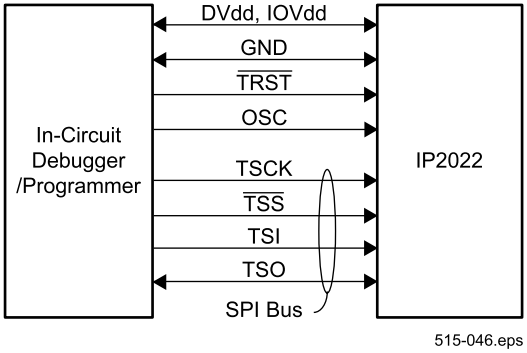


Figure 5-7 ISD/ISP Interface Signals

These signals are described below:

- *DVdd, IOVdd, GND (Power and Ground)*—If the debugger/programmer draws power from the target system, then:
  - The debugger/programmer must function on any voltage in the range from 2.3 to 3.6V. The current requirements of the debugger/programmer must be published by the vendor.
  - The debugger/programmer may use the IOVdd pin as the logic supply rail when driving the target microcontroller, therefore the specification that IOVdd will not exceed 3.6V.
  - There is no disadvantage in a debugger/programmer being able to operate with an IOVdd voltage greater 3.6V as long as it is capable of running with an IOVdd voltage down to 2.3V and does not violate the maximum logic high voltage of 3.6V.
  - Current drawn from the target circuit should be minimal to minimize the influence of the debugger on the target system.

  If the debugger/programmer is capable of supplying power to the circuit under test, then the user must be provided with the ability to turn off this feature so that it does not interfere with a circuit under test that is self-powered and provides IOVdd to the ISD/ISP interface. Additionally, the debugger/programmer must not be damaged when this featured is enabled and the target system provides IOVdd to the ISD/ISP interface. If the debugger/programmer is capable of supplying an adjustable regulated voltage to the IOVdd pin, then there is no require-

ment that the voltage range supplied be limited to 2.3 to 3.6V. The voltage and current capabilities of the IOVdd supply from the debugger/programmer must be published by the vendor.

- *$\overline{\text{TRST}}$ (Target Reset)*—Initializes the IP2022 into a known state. The debugger/programmer may provide a 100-ms system reset signal ($\overline{\text{TRST}}$) to the target system. If supported, the $\overline{\text{TRST}}$ output must be an open-collector driver to accommodate other sources of reset in the target system. The minimum source requirement for this driver is 6 mA.

  The circuit under test may use the $\overline{\text{TRST}}$ signal to reset the entire system, to reset only the IP2022, or it may ignore the $\overline{\text{TRST}}$ signal.

  Driving the $\overline{\text{TRST}}$ signal low does not guarantee that the IP2022 has been reset. The debugger/programmer must issue a DEBUG_RESET or RESET_ALL command to ensure that the IP2022 is reset.

  The debugger/programmer should not detect or be reset by the $\overline{\text{TRST}}$ signal being driven low by the target system. There is no requirement that the IP2022 is connected to the $\overline{\text{TRST}}$ signal, so the debugger/programmer cannot assume that the IP2022 has been reset if the target system pulls the $\overline{\text{TRST}}$ pin low.
- *OSC (Target Clock Oscillator)*—If the debugger/programmer is capable of supplying an OSC clock for the target system, then it must be configurable so that the clock can be disabled to prevent it from interfering with the target system (i.e. the OSC clock output is placed in a high-impedance state).

- *TSCK (Target Serial Clock)*—Clock input used to synchronize data transfer through the TSI and TSO signals.
- *$\overline{TSS}$ (Target Slave Select)*—Active-low select signal which, when asserted, enables the IP2022 to communicate on the SPI bus. This signal must be asserted before data transfers and must stay low for the duration of the transaction. The transaction is defined as an 8-bit transfer, and with the clock phase and polarity options chosen provides the flexibility of keeping TSS asserted or negated between transactions.
- *TSI (Target Serial Input)*—Sampled on the rising edge of TSCK. Unidirectional input signal to the processor driven by the SPI master.
- *TSO (Target Serial Output)*—Driven after the falling edge of TSCK. Unidirectional output signal from the IP2022 driven only when the processor SPI interface is selected by asserting TSS. Placed in high-impedance mode if the IP2022 is not selected.

5.5.1 Recommended Connector Pin Assignments

The recommended connector layout for the ISD/ISP interface is shown in Figure 5-8. The recommended connector is male 10-pin connector with 100-mil pin spacing, whose pin assignments are listed in Table 5-2. The connector is keyed to prevent backward insertion.

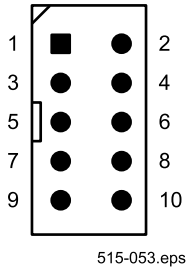


Figure 5-8 Recommended ISD/ISP Connector (top view)

Table 5-2 Connector Pin Assignments

Pin	Name	Description
1	KEY	Key (not a signal)
2	$\overline{\text{TSS}}$	Target Slave Select
3	GND	Ground
4	TSCK	Target Data Clock
5	OSC	Target Clock Oscillator
6	Reserved	Reserved
7	$\overline{\text{TRST}}$	Target Reset
8	TSI	Target Serial Input
9	Vdd	Power
10	TSO	Target Serial Output

5.6 ISD/ISP Electrical Specifications

The signal levels for the ISD/ISP interface are listed in Table 5-3.

Table 5-3 ISD/ISP Electrical Specifications

Symbol	Parameter	Min	Typ	Max	Units
$\overline{TSS}$ V_{OH}	Output high voltage (I = 0 mA)	2.3		3.6	V
$\overline{TSS}$ V_{OH}	Output high voltage (I = 4 mA)	2.3		3.6	V
$\overline{TSS}$ V_{OL}	Output low voltage (I = 0 mA)	0		0.5	V
$\overline{TSS}$ V_{OL}	Output low voltage (I = 4 mA)	0		0.5	V
TSCK V_{OH}	Output high voltage (I = 0 mA)	2.3		3.6	V
TSCK V_{OH}	Output high voltage (I = 4 mA)	2.3		3.6	V
TSCK V_{OL}	Output low voltage (I = 0 mA)	0		0.5	V
TSCK V_{OL}	Output low voltage (I = 4 mA)	0		0.5	V
TSI V_{OH}	Output high voltage (I = 0 mA)	2.3		3.6	V
TSI V_{OH}	Output high voltage (I = 4 mA)	2.3		3.6	V
TSI V_{OL}	Output low voltage (I = 0 mA)	0		0.5	V
TSI V_{OL}	Output low voltage (I = 4 mA)	0		0.5	V
TSO V_{IH}	Input high voltage	1.2		-	V
TSO V_{IL}	Input low voltage	-		0.6	
OSC V_{OH}	Output high voltage (I = 0 mA)	2.3		3.6	
OSC V_{OH}	Output high voltage (I = 1 mA)	2.3		3.6	
OSC V_{OL}	Output low voltage (I = 0 mA)	-0.3		0.5	
OSC V_{OL}	Output low voltage (I = 1 mA)	-0.3		0.5	

Table 5-3 ISD/ISP Electrical Specifications (continued)

Symbol	Parameter	Min	Typ	Max	Units
$\overline{TRST}$ V_{OL}	Output low voltage (I = 0 mA)	-0.3		0.5	
$\overline{TRST}$ V_{OL}	Output low voltage (I = 6 mA)	-0.3		0.5	
$\overline{TRST}$ T_L	Reset duration (low time)	90	100	110	ms

The timing parameters are shown in Figure 5-9. The timing requirements for the ISD/ISP interface are listed in Table 5-4.

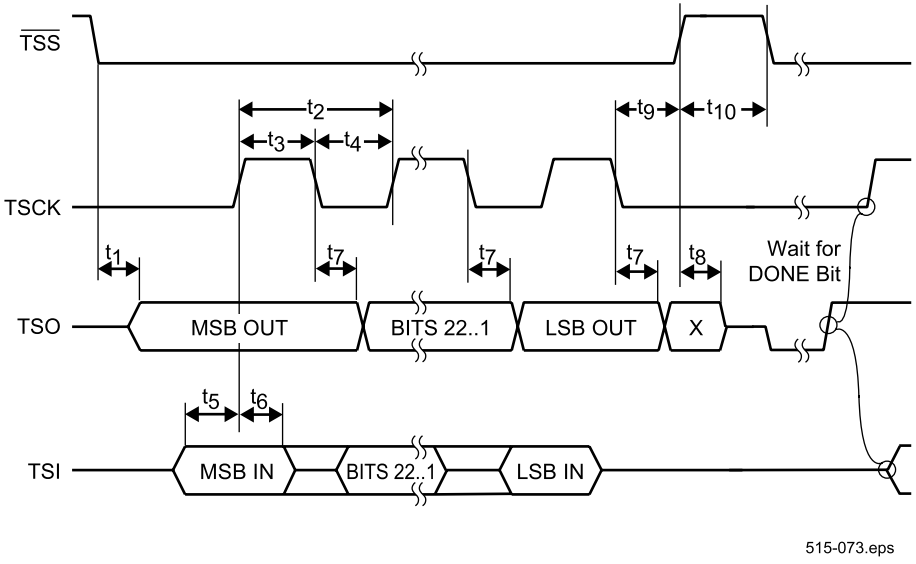


Figure 5-9 ISD/ISP Timing Diagram

Table 5-4 ISD/ISP Timing Specifications

Symbol	Parameter	Min	Typ	Max	Units
t_1	$\overline{TSS}$ select asserted to TSO driven			5	ns
t_2	Clock period	50			ns
t_3	Clock high time	20			ns
t_4	Clock low time	20			ns
t_5	TSI data setup time before TCLK positive edge	5			ns
t_6	TSI data hold time after TCLK positive edge	5			ns
t_7	TSO hold time after negative edge of TCLK	5			ns
t_8	$\overline{TSS}$ negated to TSO undriven			5	ns
t_9	Last falling edge of TSCK to $\overline{TSS}$ negated	20			ns
t_{10}	Negation time between Multiple Transfer Cycles	100			ns

6.0 In-System Programming

The IP2022 provides a dedicated serial interface for in-system programming (ISP) of the flash program memory and configuration block. ISP allows designers to incorporate a small connector which can be used to interface to a device programmer for programming or reprogramming the IP2022 after it has been soldered to a circuit board. This feature has several uses, including:

- Rapid reprogramming of prototype hardware during several iterations of software development.
- Repair of assembled boards that have been programmed with buggy software.
- Customization of a product for multiple applications that share the same hardware design, to reduce inventory costs.
- Customization of a product for local requirements, such as the language used for the user interface, the format of telephone numbers, and the format used to represent calendar dates.
- Enhancement of a product to accommodate modifications or extensions to rapidly changing communication protocols.
- Assignment of individual serial numbers or encryption keys to manufactured units.

6.1 Flash Programming

An address pointer is used to specify the memory location to be read or written. This pointer is not mapped into any address space readable through the ISD/ISP interface, but it can be loaded using the ADDR_HI and ADDR_LO commands. After the pointer has been loaded, only a single FWRITE command (which automatically increments the address pointer) needs to be executed for each location that is written. (The FWRITE command on the ISD/ISP interface should not be confused with the **`fwrite`** instruction of the IP2022 instruction set; they perform similar functions, but operate differently.) The ADDR_HI and ADDR_LO commands provide a 32-bit address for accessing program memory and the configuration block through the ISD/ISP interface, as shown in Figure 6-1.

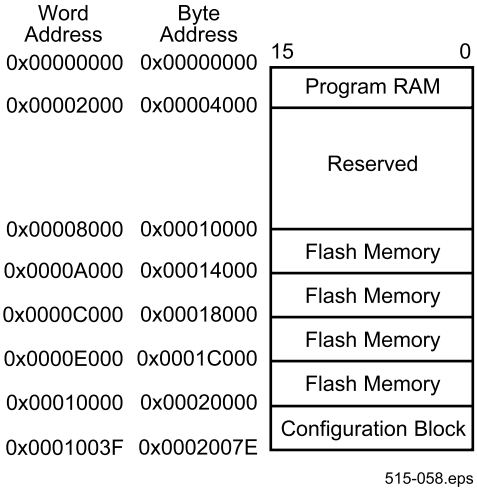


Figure 6-1 ISD/ISP Address Space

To read data memory through the ISD/ISP interface, an INST command is used to execute a **`mov`** instruction for moving the contents of a register or data memory location into the W register. A second INST command (which could execute a **`nop`** instruction, if no other instruction would be appropriate) then would return the data in the command acknowledge reply.

To write data memory, an INST command executing a **`mov w,#lit8`** instruction would load the data into the W register, after which it can be moved to any register or data memory location by issuing a second INST command.

Before programming the flash memory through the ISD/ISP interface, the OSC clock must be running and the CPU must be stopped (i.e. break mode). The debugger/programmer can read the FUSE0 and FREQ registers in the configuration block to determine the flash timing parameters, which are then loaded in the flash configuration (FCFG) register. If the CPU clock is not running or the settings in the FUSE0 or FREQ registers are incorrect, the IP2022 may fail to program.

Programming each word takes about 30–40 microseconds. The SPI bus transferring three-byte commands at 1 MHz takes a minimum of 24 clock cycles per command, or 24 microseconds. Therefore, the speed of programming at this clock rate is limited by the flash write speed, not the speed of the SPI bus.

After programming, the contents of the flash memory should be verified using IREAD commands (again, not to be confused with the **`iread`** instruction), which automatically increment the address pointer. To read the same flash location multiple times, the address pointer must be reinitialized using ADDR_HI and ADDR_LO commands between IREAD commands.

For gang programming, multiple processor chips could be connected as shown in Figure 6-2. The programmer must verify that all chips are done for a given program command before proceeding by checking their command acknowledge signal (i.e. the DONE bit). This configuration also allows the programmer to individually verify the programming of each chip.

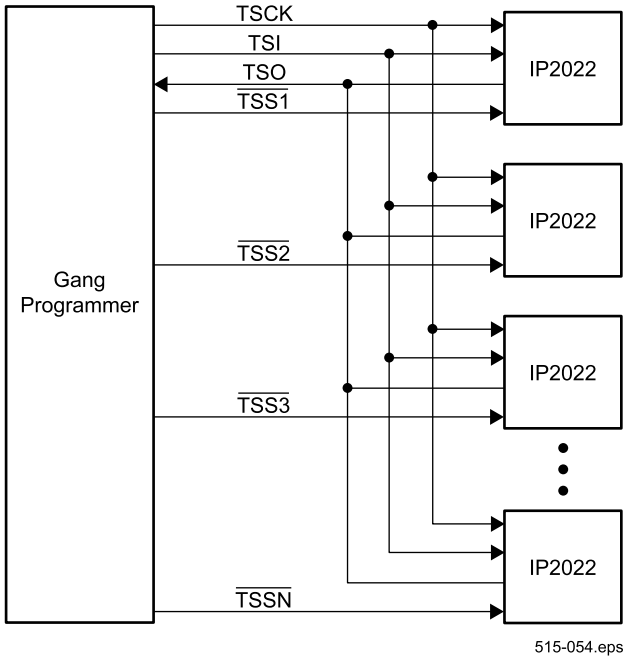


Figure 6-2 Gang Programming

6.2 Configuration Block

The configuration block holds flash memory bits outside of the program and data memories, but accessible through the ISD/ISP interface. The programmer must read and save the contents of the configuration block before any bulk erase (BULK_ERASE

command) of the IP2022. The configuration block must then be restored after erasure.

If code protection is enabled (i.e. the CP bit in the FUSE1 register is clear), bulk erase twice with the address pointer set to 0x00000000, then bulk erase once with the address pointer set to 0x00010000 to erase the configuration block.

The FUSE0, FUSE1, and TRIM0 registers hold bits that affect the hardware operation of the IP2022. All other bits in the configuration block have functions established by software convention, which may affect the operation of the debugger/programmer and software development tools.

Table 6-1 Configuration Block

Address	Words	Name	Description
0x00010000	1	FUSE0	FUSE0 register
0x00010001	1	FUSE1	FUSE1 register
0x00010002-0x00010003	2	-	Reserved
0x00010004	1	TRIM0	TRIM0 register
0x00010005-0x0001001D	9	-	Reserved
0x0001001E-0x0001001F	2	FREQ	OSC1 input frequency during device programming

Table 6-1 Configuration Block (continued)

Address	Words	Name	Description
0x00010020-0x00010027	8	VCOMPANY	Company name
0x00010028-0x0001002F	8	VPRODUCT	Product name
0x00010030-0x00010031	2	VVERSION	Software version
0x00010032-0x00010033	2	VSOFTDATE	Software date
0x00010034-0x00010035	2	VPROGDATE	Programming date
0x00010036-0x0001003F	10	-	Reserved

6.2.1 FUSE0 Register

15	14	13 12	11 9	8 6	5 3	2 0
$\overline{\text{XTAL}}$	$\overline{\text{RTCLK}}$	POUT1:0	PIN2:0	Reserved	WUDP2:0	WUDX2:0

Name	Description
$\overline{\text{XTAL}}$	OSC2 crystal drive output. Output must be disabled if an external clock signal is driven on OSC1 input. 0 = Enabled 1 = Disabled
$\overline{\text{RTCLK}}$	RTCLK2 crystal drive output. Output must be disabled if an external clock signal is driven on RTCLK1 input. 0 = Enabled 1 = Disabled
POUT1:0	Specifies PLL clock multiplier postscaler divisor. 00 = 1 01 = 2 10 = 3 11 = 4

Name	Description
PIN2:0	Specifies PLL clock multiplier prescaler divisor. 000 = 1 001 = 2 010 = 3 011 = 4 100 = 5 101 = 6 110 = 7 111 = 8
WUDP2:0	Specifies suspend time for system clock during PLL startup (after a **speed** instruction clears the $\overline{\text{PLL}}$ bit in the SPDREG register). 000 = 128 µs 001 = 192 µs 010 = 320 µs 011 = 576 µs 100 = 1.088 ms 101 = 2.112 ms 110 = 4.160 ms 111 = 8.256 ms

Name	Description
WUDX2:0	Specifies suspend time for system clock during OSC startup. 000 = 320 µs 001 = 1.088 ms 010 = 4.160 ms 011 = 8.256 ms 100 = 16.448 ms 101 = 65.600 ms 110 = 524.352 ms 111 = 1048.64 ms

6.2.2 FUSE1 Register

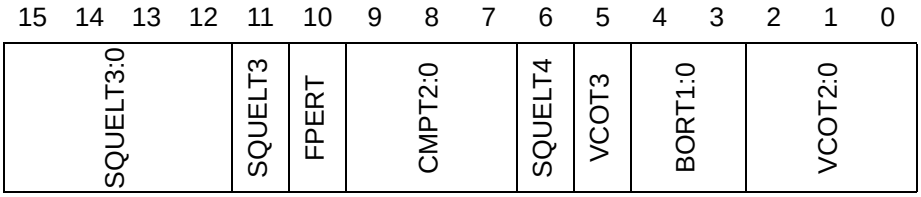
15	14	13 … 7	6 … 4	3	2 … 0
$\overline{CP}$	$\overline{SYNC}$	Reserved	BOR2:0	WDTE	WDIV2:0

Name	Description
$\overline{CP}$	Clear to enable code protection. Once cleared, this bit cannot be set until the entire device is erased. When code protection is enabled, program memory reads as all 0 to an external device programmer. This bit does not affect access to program flash memory made by software, using the **`iread`** instruction. In-system debugging is not available when code protection is enabled. Code protection does not protect the configuration block against reading. 0 = Enabled 1 = Disabled
$\overline{SYNC}$	Set to read directly from the port pins through the RxIN register, clear to read through a synchronization register. This bit should be clear if any external devices that can be read from I/O port pins are running asynchronously to the CPU core clock. 0 = Enabled 1 = Disabled

Name	Description
BOR2:0	Specifies brown-out voltage level. If AVdd goes below this level, the IP2022 is reset. 000 = 2.40V 001 = 2.35V 010 = 2.30V 011 = 2.25V 100 = 2.20V 101 = 2.10V 110 = 2.00V 111 = Disabled, no brown-out reset can occur.
WDTE	Enables Watchdog Timer. 0 = Disabled 1 = Enabled

Name	Description
WDPS2:0	Specifies the Watchdog Timer prescaler divisor. This controls the time period before the Watchdog Timer expires. If the Watchdog Timer is enabled, software must clear the Watchdog Timer periodically within this time period to prevent a reset of the IP2022 from occurring. The **`cwdt`** instruction or a reset from the $\overline{\text{RST}}$ pin clears both the Watchdog Timer and its prescaler. 000 = 1 (~20 ms) 001 = 2 (~40 ms) 010 = 4 (~80 ms) 011 = 8 (~160 ms) 100 = 16 (~320 ms) 101 = 32 (~640 ms) 110 = 64 (~1280 ms) 111 = 128 (~2560 ms)

6.2.3 TRIM0 Register



Name	Description
SQUELT5:0	SERDES squelch trim bits
FPERT	Controls flash block pulse erase time, for both self-programming **`ferase`** and the FERASE command from the ISD/ISP interface 0 = 20 ms 1 = 10 ms
CMPT2:0	Comparator offset trim bits
VCOT3:0	PLL VCO trim bits
BORT2:0	Brown-out detector bits

6.2.4 FREQ Register

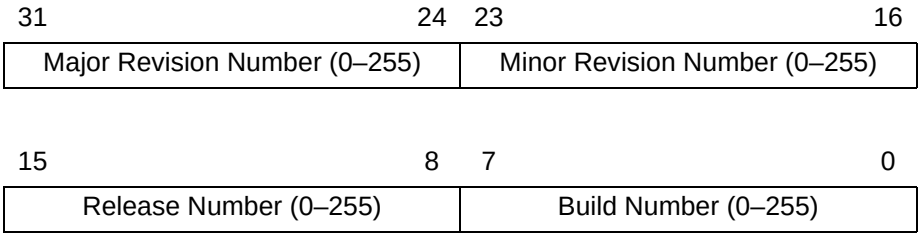
This register holds a 32-bit unsigned integer which represents the OSC1 input frequency in Hz. Debugger/programmers can use this number to calculate the values loaded into the flash configuration register (FCFG) for device programming. The frequency is specified from software using the FREQ assembler directive.

6.2.5 VCOMPANY and VPRODUCT Registers

Each of these registers contains 16 bytes for company and product identifiers. The identifiers are ASCII strings, beginning with rightmost byte at the lowest address. Identifiers shorter than 16 bytes should be padded to 16 bytes with space characters.

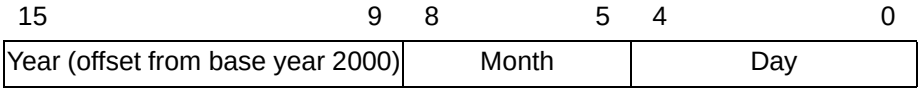
6.2.6 VVERSION Register

This register holds the software version number, using the encoding shown below.



6.2.7 VSOFTDATE and VPROGDATE Registers

These registers hold the software build date and the device programming date, respectively, using the encoding shown below.



A.0 Similarities with SX-Series Devices

This appendix summarizes the significant differences between the operation of SX-Series devices and the IP2022.

Instruction Format—The 12-bit instruction formats on the SX-Series devices have been extended to 16-bit formats on the IP2022, as shown below. (A, B, C, D, and E represent opcode bits.)

Instruction Type	SX-Series Instruction	IP2022 Instruction
Register Operation	00AB CDEf ffff	00AB CDEf ffff ffff
Bit Operation	01AB bbbf ffff	10AB bbbf ffff ffff
8-bit Literal Operation	1ABC kkkk kkkk	0111 1ABC kkkk kkkk

Most SX-Series instructions have an equivalent IP2022 instruction in the 16-bit format. An important feature of the 16-bit format is the extension of the “fr” field from five bits to nine bits.

Register Banks—The register banking scheme used in the SX-Series devices has been discarded in favor of a uniform address space. Register banking can be simulated by using the IPH/IPL pointer register to emulate the FSR register.

Page Size—The SX-Series devices had a 9-bit field for specifying a jump destination within a 512-instruction page, and an 8-bit field

for specifying a call destination within the first half of a 512-instruction page. The IP2022 uses 13-bit fields for each, resulting in a page size of 8192 instructions.

Call/Return Stack—The depth of the hardware call/return stack has been increased from 8 levels on the SX-Series devices to 16 levels on the IP2022. The top of the stack is mapped into data memory, so software can extend the stack depth for those applications which require more deeply nested subroutines.

Carry/Borrow—The SX-Series devices had a fuse bit to distinguish between **add** and **addc** (add-with-carry) instructions. The IP2022 implements separate **add** and **addc** instructions, as well as the corresponding **sub** and **subc** instructions.

Interrupt Processing—The SX-Series devices disabled interrupts during the execution of the interrupt service routine (ISR). The IP2022 allows re-enabling interrupts in the ISR, and it provides a 2-level stack for saving critical registers during nested interrupts.

STATUS Register Bits—The power-down bit (PD) and time-out bit (TO) have been renamed the brown-out bit (BO) and watchdog bit (WD), respectively. Functionally, the WD bit differs from the TO bit in that the **cwdt** instruction does not automatically clear the WD bit.

Low-Power Support—The SX-Series devices had a sleep mode which only could be exited by a device reset. The IP2022 does not have a sleep mode, but it supports clock throttling and clock-stop modes. The IP2022 can exit from these modes in response to an external interrupt or an on-chip interrupt from the real-time timer.

RTCC External Clock—The RTCC on the SX-Series devices allowed an external clock source. The equivalent function on the IP2022 does not have this option, however the real-timer timer can accept an external clock for those applications which require this functionality.

I/O Ports—The SX-Series devices provided options for on-chip pullup resistors, Schmitt trigger inputs, and TTL logic levels. The IP2022 only allows CMOS logic levels and does not offer pullup or Schmitt trigger options. The SX-Series had a bit for specifying whether a read from a port would read data sampled from the port pins or the data register used to drive output port pins. On the IP2022, both of these options are always available by reading either the RxIN or RxOUT registers, respectively. All of the port registers are mapped into the data memory of the IP2022, which eliminates the need for the MODE register used on the SX-Series devices.

B.0

Pin Assignments

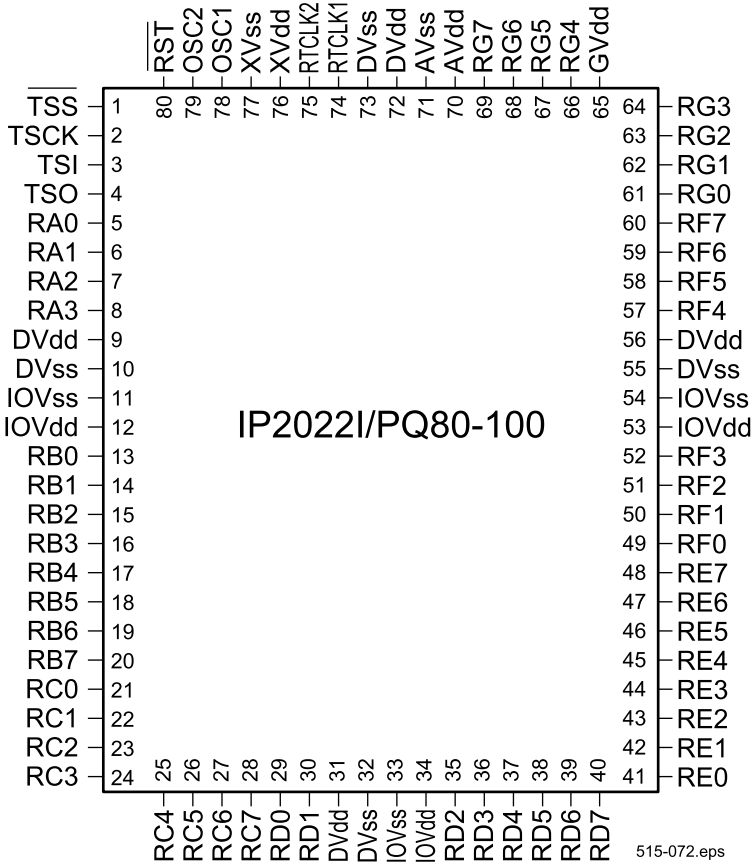


Figure B-1 Pin Assignments (top view)

B.1 Signal Descriptions

I = Digital Input, AI = Analog Input, O/DO = Digital Output, HiZ = High Impedance, P = Power, PLP = On-Chip Pullup, ST = Schmitt Trigger

Table B-1 Signal Descriptions

Name	Pin	Type	Sink @ 3.3V IOVDD	Source @ 3.3V IOVDD	Description
AVDD	70	P			Analog Supply
AVSS	71	P			Analog Ground
DVDD	9, 31, 56, 72	P			Logic Supply
DVSS	10, 32, 55, 73	P			Logic Ground
GVDD	65	P			I/O Port G supply
IOVDD	12, 34, 53	P			I/O Supply (except Port G)
IOVSS	11, 33, 54	P			I/O Ground (all ports)
XVDD	76	P			PLL Supply
XVSS	77	P			PLL Ground
OSC1	78	I			Clock/Crystal/Resonator Input
OSC2	79	O			Crystal/Resonator Output
$\overline{\text{RST}}$	80	I/ST/PLP			Reset Input
RTCLK1	74	I			Real-Time Clock/Crystal Input

Table B-1 Signal Descriptions (continued)

Name	Pin	Type	Sink @ 3.3V IOVDD	Source @ 3.3V IOVDD	Description
RTCLK2	75	O			Real-Time Crystal Output
TSCK	2	I/ST/PLP			Target SPI Clock
TSI	3	I/ST/PLP			Target SPI Serial Data Input
TSO	4	O/HiZ			Target SPI Serial Data output
$\overline{\text{TSS}}$	1	I/ST/PLP			Target SPI Slave Select
RA0	5	I/O	24 mA	24 mA	I/O Port, High Power Output, Timer 1 Capture 1 Input
RA1	6	I/O	24 mA	24 mA	I/O Port, High Power Output, Timer 1 Capture 2 Input
RA2	7	I/O	24 mA	24 mA	I/O Port, High Power Output, Timer 1 Clock Input
RA3	8	I/O	24 mA	24 mA	I/O Port, High Power Output, Timer 1 Output
RB0	13	I/O	8 mA	8 mA	I/O Port, External Interrupt, Timer 2 Capture 1 Input
RB1	14	I/O	8 mA	8 mA	I/O Port, External Interrupt, Timer 2 Capture 2 Input
RB2	15	I/O	8 mA	8 mA	I/O Port, External Interrupt, Timer 2 Clock Input
RB3	16	I/O	8 mA	8 mA	I/O Port, External Interrupt, Timer 2 Output
RB4	17	I/O	8 mA	8 mA	I/O Port, External Interrupt

Table B-1 Signal Descriptions (continued)

Name	Pin	Type	Sink @ 3.3V IOVDD	Source @ 3.3V IOVDD	Description
RB5	18	I/O	8 mA	8 mA	I/O Port, External Interrupt, Parallel Slave Peripheral $\overline{\text{HOLD}}$
RB6	19	I/O	8 mA	8 mA	I/O Port, External Interrupt, Parallel Slave Peripheral R/W
RB7	20	I/O	8 mA	8 mA	I/O Port, External Interrupt, Parallel Slave Peripheral $\overline{\text{CS}}$
RC0	21	I/O	4 mA	4 mA	I/O Port, Parallel Slave Peripheral Data
RC1	22	I/O	4 mA	4 mA	I/O Port, Parallel Slave Peripheral Data
RC2	23	I/O	4 mA	4 mA	I/O Port, Parallel Slave Peripheral Data
RC3	24	I/O	4 mA	4 mA	I/O Port, Parallel Slave Peripheral Data
RC4	25	I/O	4 mA	4 mA	I/O Port, Parallel Slave Peripheral Data
RC5	26	I/O	4 mA	4 mA	I/O Port, Parallel Slave Peripheral Data
RC6	27	I/O	4 mA	4 mA	I/O Port, Parallel Slave Peripheral Data
RC7	28	I/O	4 mA	4 mA	I/O Port, Parallel Slave Peripheral Data

Table B-1 Signal Descriptions (continued)

Name	Pin	Type	Sink @ 3.3V IOVDD	Source @ 3.3V IOVDD	Description
RD0	29	I/O	4 mA	4 mA	I/O Port, Parallel Slave Peripheral Data
RD1	30	I/O	4 mA	4 mA	I/O Port, Parallel Slave Peripheral Data
RD2	35	I/O	4 mA	4 mA	I/O Port, Parallel Slave Peripheral Data
RD3	36	I/O	4 mA	4 mA	I/O Port, Parallel Slave Peripheral Data
RD4	37	I/O	4 mA	4 mA	I/O Port, Parallel Slave Peripheral Data
RD5	38	I/O	4 mA	4 mA	I/O Port, Parallel Slave Peripheral Data
RD6	39	I/O	4 mA	4 mA	I/O Port, Parallel Slave Peripheral Data
RD7	40	I/O	4 mA	4 mA	I/O Port, Parallel Slave Peripheral Data
RE0	41	I/O	8 mA	8 mA	I/O Port, Serial 1 CLK
RE1	42	I/O	8 mA	8 mA	I/O Port, Serial 1 RXP
RE2	43	I/O	8 mA	8 mA	I/O Port, Serial 1 RXM
RE3	44	I/O	8 mA	8 mA	I/O Port, Serial 1 RXD
RE4	45	I/O	8 mA	8 mA	I/O Port, Serial 1 TXPE
RE5	46	I/O	24 mA	24 mA	I/O Port, High Power Output, Serial 1 TXP

Table B-1 Signal Descriptions (continued)

Name	Pin	Type	Sink @ 3.3V IOVDD	Source @ 3.3V IOVDD	Description
RE6	47	I/O	24 mA	24 mA	I/O Port, High Power Output, Serial 1 TXM
RE7	48	I/O	8 mA	8 mA	I/O Port, Serial 1 TXME
RF0	49	I/O	8 mA	8 mA	I/O Port, Serial 2 TXPE
RF1	50	I/O	24 mA	24 mA	I/O Port, High Power Output, Serial 2 TXP
RF2	51	I/O	24 mA	24 mA	I/O Port, High Power Output, Serial 2 TXM
RF3	52	I/O	8 mA	8 mA	I/O Port, Serial 2 TXME
RF4	57	I/O	8 mA	8 mA	I/O Port, Serial 2 CLK
RF5	58	I/O	8 mA	8 mA	I/O Port, Serial 2 RXP
RF6	59	I/O	8 mA	8 mA	I/O Port, Serial 2 RXM
RF7	60	I/O	8 mA	8 mA	I/O Port, Serial 2 RXD

Table B-1 Signal Descriptions (continued)

Name	Pin	Type	Sink @ 3.3V IOVDD	Source @ 3.3V IOVDD	Description
RG0	61	AI/DO	4 mA*	4 mA*	Output Port, ADC0 Input, Comparator Output
RG1	62	AI/DO	4 mA*	4 mA*	Output Port, ADC1 Input, Comparator – Input
RG2	63	AI/DO	4 mA*	4 mA*	Output Port, ADC2 Input, Comparator + Input
RG3	64	AI/DO	4 mA*	4 mA*	Output Port, ADC3 Input, ADC reference Input
RG4	66	AI/DO	4 mA*	4 mA*	Output Port, ADC4 Input, SERDES1 Squelch – Input
RG5	67	AI/DO	4 mA*	4 mA*	Output Port, ADC5 Input, SERDES1 Squelch + Input
RG6	68	AI/DO	4 mA*	4 mA*	Output Port, ADC6 Input, SERDES2 Squelch – Input
RG7	69	AI/DO	4 mA*	4 mA*	Output Port, ADC7 Input, SERDES2 Squelch + Input

* GVDD = 2.5V

C.0 Register Quick Reference

This appendix presents a register map and register bit definitions for the special-purpose registers in data memory. The bit definitions are only shown for those registers that have dedicated bits or fields within the register. The configuration block registers (i.e. fuse registers) are described in Section 6.2.

C.1 Registers (sorted by address)

Table C-1 shows the addresses and reset values of all special-purpose registers in data memory, sorted by their address.

Table C-1 Register Addresses and Reset State

Address	Name	Description	Reset Value
0x001	Reserved	Reserved	Reserved
0x002	ADDRSEL	Selector for current external/program memory pointer	00000000
0x003	ADDRX	External/program memory pointer (bits 23:16)	00000000
0x004	IPH	Indirect Pointer (high byte)	00000000
0x005	IPL	Indirect Pointer (low byte)	00000000
0x006	SPH	Stack Pointer (high byte)	00000000
0x007	SPL	Stack Pointer (low byte)	00000000
0x008	PCH	Current PC bits 15:8 (read-only)	11111111
0x009	PCL	Virtual register for direct PC modification	11110000
0x00A	W	W register	00000000
0x00B	STATUS	STATUS register	On POR or RST Reset: 11100000 On Brown-out Reset: 11101000 On WDT Overflow: 11110000
0x00C	DPH	Data Pointer (high byte)	00000000
0x00D	DPL	Data Pointer (low byte)	00000000
0x00E	SPDREG	Current speed (read-only)	10010011

Table C-1 Register Addresses and Reset State (continued)

Address	Name	Description	Reset Value
0x00F	MULH	Multiply result (high byte)	00000000
0x010	ADDRH	External/program memory pointer (bits 15:8)	00000000
0x011	ADDRL	External/program memory pointer (bits 7:0)	00000000
0x012	DATAH	External/program memory data (high byte)	00000000
0x013	DATAL	External/program memory data (low byte)	00000000
0x014	INTVECH	Interrupt vector (high byte)	00000000
0x015	INTVECL	Interrupt vector (low byte)	00000000
0x016	INTSPD	Interrupt speed register	00000000
0x017	INTF	Port B interrupt flags	Undefined
0x018	INTE	Port B interrupt enable bits	00000000
0x019	INTED	Port B interrupt edge select bits	00000000
0x001	FCFG	Flash configuration register	00000000
0x01B	TCTRL	Timer 1/2 common control register	00000000
0x01C	XCFG	Extended configuration	00000001
0x01D	EMCFG	External memory configuration register	00000000
0x01E	IPCH	Interrupt return address (high byte)	00000000
0x01F	IPCL	Interrupt return address (low byte)	00000000
0x020	RAIN	Data on Port A pins (read-only, upper four bits read as 0000)	N/A

Table C-1 Register Addresses and Reset State (continued)

Address	Name	Description	Reset Value
0x021	RAOUT	Port A output latch	00000000
0x022	RADIR	Port A direction register	11111111
0x023	LFSRH	LFSR data register (high byte)	00000000
0x024	RBIN	Data on Port B pins (read-only)	N/A
0x025	RBOUT	Port B output latch	00000000
0x026	RBDIR	Port B direction register	11111111
0x027	LFSRL	LFSR data register (low byte)	00000000
0x028	RCIN	Data on Port C pins (read-only)	N/A
0x029	RCOUT	Port C output latch	00000000
0x02A	RCDIR	Port C direction register	11111111
0x02B	LFSRA	LFSR address register	00000000
0x02C	RDIN	Data on Port D pins (read-only)	N/A
0x02D	RDOUT	Port D output latch	00000000
0x02E	RDDIR	Port D direction register	11111111
0x02F	Reserved	Reserved	Reserved
0x030	REIN	Data on Port E pins (read-only)	N/A
0x031	REOUT	Port E output latch	00000000
0x032	REDIR	Port E direction register	11111111
0x033	Reserved	Reserved	Reserved
0x034	RFIN	Data on Port F pins (read-only)	N/A
0x035	RFOUT	Port F output latch	00000000
0x036	RFDIR	Port F direction register	11111111
0x037	Reserved	Reserved	Reserved
0x038	Reserved	Reserved	Reserved

Table C-1 Register Addresses and Reset State (continued)

Address	Name	Description	Reset Value
0x039	RGOUT	Port G output latch	00000000
0x03A	RGDIR	Port G direction register	11111111
0x03B	Reserved	Reserved	Reserved
0x03C	Reserved	Reserved	Reserved
0x03D	Reserved	Reserved	Reserved
0x03E	Reserved	Reserved	Reserved
0x03F	Reserved	Reserved	Reserved
0x040	RTTMR	Real-time timer value	00000000
0x041	RTCFG	Real-time timer configuration register	00000000
0x042	T0TMR	Timer 0 value	00000000
0x043	T0CFG	Timer 0 configuration register	00000000
0x044	T1CNTH	Timer 1 counter register (high byte, read-only)	00000000
0x045	T1CNTL	Timer 1 counter register (low byte, read-only)	00000000
0x046	T1CAP1H	Timer 1 Capture 1 register (high byte, read-only)	00000000
0x047	T1CAP1L	Timer 1 Capture 1 register (low byte, read-only)	00000000
0x048	T1CAP2H or T1CMP2H	Timer 1 Capture 2 (high byte) Timer 1 Compare 2 (high byte)	00000000

Table C-1 Register Addresses and Reset State (continued)

Address	Name	Description	Reset Value
0x049	T1CAP2L or T1CMP2L	Timer 1 Capture 2 (low byte) Timer 1 Compare 2 (low byte)	00000000
0x04A	T1CMP1H	Timer 1 Compare 1 register (high byte)	00000000
0x04B	T1CMP1L	Timer 1 Compare 1 register (low byte)	00000000
0x04C	T1CFG1H	Timer 1 configuration register 1 (high byte)	00000000
0x04D	T1CFG1L	Timer 1 configuration register 1 (low byte)	00000000
0x04E	T1CFG2H	Timer 1 configuration register 2 (high byte)	00000000
0x04F	T1CFG2L	Timer 1 configuration register 2 (low byte)	00000000
0x050	ADCH	ADC value (high byte)	00000000
0x051	ADCL	ADC value (low byte)	00000000
0x052	ADCCFG	ADC configuration register	00000000
0x053	ADCTMR	ADC timer register	00000000
0x054	T2CNTH	Timer 2 counter register (high byte, read-only)	00000000
0x055	T2CNTL	Timer 2 counter register (low byte, read-only)	00000000

Table C-1 Register Addresses and Reset State (continued)

Address	Name	Description	Reset Value
0x056	T2CAP1H	Timer 2 Capture 1 register (high byte, read-only)	00000000
0x057	T2CAP1L	Timer 2 Capture 1 register (low byte, read-only)	00000000
0x058	T2CAP2H or T2CMP2H	Timer 2 Capture 2 (high byte) Timer 2 Compare 2 (high byte)	00000000
0x059	T2CAP2L or T2CMP2L	Timer 2 Capture 2 (low byte) Timer 2 Compare 2 (low byte)	00000000
0x05A	T2CMP1H	Timer 2 Compare 1 register (high byte)	00000000
0x05B	T2CMP1L	Timer 2 Compare 1 register (low byte)	00000000
0x05C	T2CFG1H	Timer 2 configuration register 1 (high byte)	00000000
0x05D	T2CFG1L	Timer 2 configuration register 1 (low byte)	00000000
0x05E	T2CFG2H	Timer 2 configuration register 2 (high byte)	00000000
0x05F	T2CFG2L	Timer 2 configuration register 2 (low byte)	00000000
0x060	S1TMRH	SERDES 1 clock timer register (high byte)	00000000

Table C-1 Register Addresses and Reset State (continued)

Address	Name	Description	Reset Value
0x061	S1TMRL	SERDES 1 clock timer register (low byte)	00000000
0x062	S1TBUFH	SERDES 1 transmit buffer (high byte)	Undefined
0x063	S1TBUFL	SERDES 1 transmit buffer (low byte)	Undefined
0x064	S1TCFG	SERDES 1 transmit configuration	00000000
0x065	S1RCNT	SERDES 1 received bit count (actual) (read-only)	00000000
0x066	S1RBUFH	SERDES 1 receive buffer (high byte)	Undefined
0x067	S1RBUFL	SERDES 1 receive buffer (low byte)	Undefined
0x068	S1RCFG	SERDES 1 receive configuration	00000000
0x069	S1RSYNC	SERDES 1 receive bit sync pattern	00000000
0x06A	S1INTF	SERDES 1 status/interrupt flags	00000000
0x06B	S1INTE	SERDES 1 interrupt enable bits	00000000
0x06C	S1MODE	SERDES 1 serial mode/clock select register	00000000
0x06D	S1SMASK	SERDES 1 receive sync mask	00000000
0x06E	PSPCFG	Parallel slave peripheral configuration register	00000000
0x06F	CMPCFG	Comparator configuration register	0000000X
0x070	S2TMRH	SERDES 2 clock timer register (high byte)	00000000
0x071	S2TMRL	SERDES 2 clock timer register (low byte)	00000000

Table C-1 Register Addresses and Reset State (continued)

Address	Name	Description	Reset Value
0x072	S2TBUFH	SERDES 2 transmit buffer (high byte)	Undefined
0x073	S2TBUFL	SERDES 2 transmit buffer (low byte)	Undefined
0x074	S2TCFG	SERDES 2 transmit configuration	00000000
0x075	S2RCNT	SERDES 2 received bit count (actual) (read-only)	00000000
0x076	S2RBUFH	SERDES 2 receive buffer (high byte)	Undefined
0x077	S2RBUFL	SERDES 2 receive buffer (low byte)	Undefined
0x078	S2RCFG	SERDES 2 receive configuration	00000000
0x079	S2RSYNC	SERDES 2 receive bit sync pattern	00000000
0x07A	S2INTF	SERDES 2 status/interrupt flags	00000000
0x07B	S2INTE	SERDES 2 interrupt enable bits	00000000
0x07C	S2MODE	SERDES 2 serial mode/clock select register	00000000
0x07D	S2SMASK	SERDES 2 receive sync mask	00000000
0x07E	CALLH	Top of call stack (high byte)	11111111
0x07F	CALLL	Top of call stack (low byte)	11111111

C.2 Registers (sorted alphabetically)

Table C-2 shows the addresses and reset values of all special-purpose registers in data memory, sorted alphabetically by name.

Table C-2 Register Addresses and Reset State

Address	Name	Description	Reset Value
0x052	ADCCFG	ADC configuration register	00000000
0x050	ADCH	ADC value (high byte)	00000000
0x051	ADCL	ADC value (low byte)	00000000
0x053	ADCTMR	ADC timer register	00000000
0x010	ADDRH	External/program memory pointer (high byte)	00000000
0x011	ADDRL	External/program memory pointer (low byte)	00000000
0x002	ADDRSEL	Selector for current external/program memory pointer	00000000
0x003	ADDRX	External/program memory pointer (bits 23:16)	00000000
0x07E	CALLH	Top of call stack (high byte)	11111111
0x07F	CALLL	Top of call stack (low byte)	11111111
0x06F	CMPCFG	Comparator configuration register	0000000X
0x012	DATAH	Program memory data (high byte)	00000000
0x013	DATAL	Program memory data (low byte)	00000000
0x00C	DPH	Data Pointer (high byte)	00000000
0x00D	DPL	Data Pointer (low byte)	00000000

Table C-2 Register Addresses and Reset State (continued)

Address	Name	Description	Reset Value
0x01D	EMCFG	External memory configuration register	00000000
0x01A	FCFG	Flash configuration register	00000000
0x018	INTE	Port B interrupt enable bits	00000000
0x019	INTED	Port B interrupt edge select bits	00000000
0x017	INTF	Port B interrupt flags	Undefined
0x016	INTSPD	Interrupt speed register	00000000
0x014	INTVECH	Interrupt vector (high byte)	00000000
0x015	INTVECL	Interrupt vector (low byte)	00000000
0x01E	IPCH	Interrupt return address (high byte)	00000000
0x01F	IPCL	Interrupt return address (low byte)	00000000
0x004	IPH	Indirect Pointer (high byte)	00000000
0x005	IPL	Indirect Pointer (low byte)	00000000
0x02B	LFSRA	LFSR address register	00000000
0x023	LFSRH	LFSR data register (high byte)	00000000
0x027	LFSRL	LFSR data register (low byte)	00000000
0x00F	MULH	Multiply result (high byte)	00000000
0x008	PCH	Current PC bits 15:8 (read-only)	11111111
0x009	PCL	Virtual register for direct PC modification	11110000
0x06E	PSPCFG	Parallel slave peripheral configuration register	00000000
0x022	RADIR	Port A direction register	11111111

Table C-2 Register Addresses and Reset State (continued)

Address	Name	Description	Reset Value
0x020	RAIN	Data on Port A pins (read-only, upper four bits read as 0000)	N/A
0x021	RAOUT	Port A output latch	00000000
0x026	RBDIR	Port B direction register	11111111
0x024	RBIN	Data on Port B pins (read-only)	N/A
0x025	RBOUT	Port B output latch	00000000
0x02A	RCDIR	Port C direction register	11111111
0x028	RCIN	Data on Port C pins (read-only)	N/A
0x029	RCOUT	Port C output latch	00000000
0x02E	RDDIR	Port D direction register	11111111
0x02C	RDIN	Data on Port D pins (read-only)	N/A
0x02D	RDOUT	Port D output latch	00000000
0x032	REDIR	Port E direction register	11111111
0x030	REIN	Data on Port E pins (read-only)	N/A
0x031	REOUT	Port E output latch	00000000
0x036	RFDIR	Port F direction register	11111111
0x034	RFIN	Data on Port F pins (read-only)	N/A
0x035	RFOUT	Port F output latch	00000000
0x03A	RGDIR	Port G direction register	11111111
0x039	RGOUT	Port G output latch	00000000
0x041	RTCFG	Real-time timer configuration register	00000000
0x040	RTTMR	Real-time timer value	00000000
0x06B	S1INTE	SERDES 1 interrupt enable bits	00000000
0x06A	S1INTF	SERDES 1 status/interrupt flags	00000000

Table C-2 Register Addresses and Reset State (continued)

Address	Name	Description	Reset Value
0x06C	S1MODE	SERDES 1 serial mode/clock select register	00000000
0x066	S1RBUFH	SERDES 1 receive buffer (high byte)	Undefined
0x0067	S1RBUFL	SERDES 1 receive buffer (low byte)	Undefined
0x068	S1RCFG	SERDES 1 receive configuration	00000000
0x065	S1RCNT	SERDES 1 received bit count (actual) (read-only)	00000000
0x069	S1RSYNC	SERDES 1 receive bit sync pattern	00000000
0x06D	S1SMASK	SERDES 1 receive sync mask	00000000
0x062	S1TBUFH	SERDES 1 transmit buffer (high byte)	Undefined
0x063	S1TBUFL	SERDES 1 transmit buffer (low byte)	Undefined
0x064	S1TCFG	SERDES 1 transmit configuration	00000000
0x060	S1TMRH	SERDES 1 clock timer register (high byte)	00000000
0x061	S1TMRL	SERDES 1 clock timer register (low byte)	00000000
0x07B	S2INTE	SERDES 2 interrupt enable bits	00000000
0x07A	S2INTF	SERDES 2 status/interrupt flags	00000000
0x07C	S2MODE	SERDES 2 serial mode/clock select register	00000000
0x076	S2RBUFH	SERDES 2 receive buffer (high byte)	Undefined
0x077	S2RBUFL	SERDES 2 receive buffer (low byte)	Undefined
0x078	S2RCFG	SERDES 2 receive configuration	00000000

Table C-2 Register Addresses and Reset State (continued)

Address	Name	Description	Reset Value
0x075	S2RCNT	SERDES 2 received bit count (actual) (read-only)	00000000
0x079	S2RSYNC	SERDES 2 receive bit sync pattern	00000000
0x07D	S2SMASK	SERDES 2 receive sync mask	00000000
0x072	S2TBUFH	SERDES 2 transmit buffer (high byte)	Undefined
0x073	S2TBUFL	SERDES 2 transmit buffer (low byte)	Undefined
0x074	S2TCFG	SERDES 2 transmit configuration	00000000
0x070	S2TMRH	SERDES 2 clock timer register (high byte)	00000000
0x071	S2TMRL	SERDES 2 clock timer register (low byte)	00000000
0x00E	SPDREG	Current speed (read-only)	10010011
0x006	SPH	Stack Pointer (high byte)	00000000
0x007	SPL	Stack Pointer (low byte)	00000000
0x00B	STATUS	STATUS register	On POR or RST Reset: 11100000 On Brown-out Reset: 11101000 On WDT Overflow: 11110000

Table C-2 Register Addresses and Reset State (continued)

Address	Name	Description	Reset Value
0x043	T0CFG	Timer 0 configuration register	00000000
0x042	T0TMR	Timer 0 value	00000000
0x046	T1CAP1H	Timer 1 Capture 1 register (high byte, read-only)	00000000
0x047	T1CAP1L	Timer 1 Capture 1 register (low byte, read-only)	00000000
0x048	T1CAP2H T1CMP2H	Timer 1 Capture 2 (high byte) Timer 1 Compare 2 (high byte)	00000000
0x049	T1CAP2L T1CMP2L	Timer 1 Capture 2 (low byte) Timer 1 Compare 2 (low byte)	00000000
0x04C	T1CFG1H	Timer 1 configuration register 1 (high byte)	00000000
0x04D	T1CFG1L	Timer 1 configuration register 1 (low byte)	00000000
0x04E	T1CFG2H	Timer 1 configuration register 2 (high byte)	00000000
0x04F	T1CFG2L	Timer 1 configuration register 2 (low byte)	00000000
0x04A	T1CMP1H	Timer 1 Compare 1 register (high byte)	00000000
0x04B	T1CMP1L	Timer 1 Compare 1 register (low byte)	00000000
0x044	T1CNTH	Timer 1 counter register (high byte, read-only)	00000000

Table C-2 Register Addresses and Reset State (continued)

Address	Name	Description	Reset Value
0x045	T1CNTL	Timer 1 counter register (low byte, read-only)	00000000
0x056	T2CAP1H	Timer 2 Capture 1 register (high byte, read-only)	00000000
0x057	T2CAP1L	Timer 2 Capture 1 register (low byte, read-only)	00000000
0x058	T2CAP2H T2CMP2H	Timer 2 Capture 2 (high byte) Timer 2 Compare 2 (high byte)	00000000
0x059	T2CAP2L T2CMP2L	Timer 2 Capture 2 (low byte) Timer 2 Compare 2 (low byte)	00000000
0x05C	T2CFG1H	Timer 2 configuration register 1 (high byte)	00000000
0x05D	T2CFG1L	Timer 2 configuration register 1 (low byte)	00000000
0x05E	T2CFG2H	Timer 2 configuration register 2 (high byte)	00000000

Table C-2 Register Addresses and Reset State (continued)

Address	Name	Description	Reset Value
0x05F	T2CFG2L	Timer 2 configuration register 2 (low byte)	00000000
0x05A	T2CMP1H	Timer 2 Compare 1 register (high byte)	00000000
0x05B	T2CMP1L	Timer 2 Compare 1 register (low byte)	00000000
0x054	T2CNTH	Timer 2 counter register (high byte, read-only)	00000000
0x055	T2CNTL	Timer 2 counter register (low byte, read-only)	00000000
0x01B	TCTRL	Timer 1/2 common control register	00000000
0x00A	W	W register	00000000
0x01C	XCFG	Extended configuration	00000001

C.3 Register Bit Definitions

For those registers which have special functions assigned to bits or fields within the register, the definition of those bits and fields is described below. The registers are presented alphabetically.

C.3.1 ADCCFG Register (A/D Converter Configuration)

7	6 5	4	3	2 0
ADCREF	ADCJST	Reserved	ADCGO	ADCS2:0

Name	Description
ADCREF	A/D converter reference voltage select 0 = AVdd is the reference voltage 1 = RG3 port pin is used to receive an external reference voltage
ADCJST	A/D converter result justification mode select (see Table 4-11) 00 = Right justified 01 = Signed 10 = Left justified 11 = Reserved
ADCGO	A/D converter GO/DONE bit 0 = When the last conversion has completed, this bit reads as 0. 1 = Write 1 to begin a new conversion. While the conversion is in progress, this bit reads as 1.

Name	Description
ADCS2:0	A/D converter input channel select 000 = Port pin RG0 001 = Port pin RG1 010 = Port pin RG2 011 = Port pin RG3 100 = Port pin RG4 101 = Port pin RG5 110 = Port pin RG6 111 = Port pin RG7

C.3.2 CMPCFG Register (Comparator Configuration)

7	6	5	4 ... 1	0
CMPEN	CMPOE	CMPHYS	Reserved	CMPRES

Name	Description
CMPEN	Comparator enable bit 0 = Comparator disabled 1 = Comparator enabled
CMPOE	Comparator output enable bit 0 = Comparator output disabled. 1 = Comparator output enabled on port pin RG0.

Name	Description
CMPHYS	Comparator hysteresis enable bit 0 = Hysteresis disabled 1 = Hysteresis enabled
CMPRES	Comparator result (read-only) 0 = RG2 voltage > RG1 1 = RG1 voltage > RG2

C.3.3 EMCFG Register

7	6	5 3	2 0
EMEN	EMBRT	EMWRT2:0	EMRDT2:0

Name	Description
EMEN	Enable external memory interface 0 = Port C and Port D available for general-purpose I/O 1 = Port C and Port D used for external memory interface
EMBRT	Enable bus release wait state 0 = No wait state 1 = One wait state added between a read cycle followed by a write cycle

Name	Description
EMWRT2:0	$\overline{WR}$ pulse width, in CPU core clock cycles 000 = 1 001 = 2 010 = 3 011 = 4 100 = 5 101 = 6 110 = 7 111 = 8
EMRDT2:0	$\overline{RD}$ pulse width, in CPU core clock cycles 000 = 1 001 = 2 010 = 3 011 = 4 100 = 5 101 = 6 110 = 7 111 = 8

C.3.4 FCFG Register (Flash Configuration)

7	6	5	4	3	2	1	0
FRDTS1:0		FRDTC1:0		FWRT3:0			

Name	Description
FRDTS1:0	The CPU core clock while executing from flash program memory must not exceed 30 MHz, otherwise unreliable operation may result. The IP2022 automatically increases the number of system clock cycles for each CPU core clock cycle when executing from flash, but it is the responsibility of software to load the FRDTS1:0 bits appropriately for the operating frequency, as shown below. The actual speed will be the slower of the speed indicated in the SPDREG register and the speed specified by the FRDTS1:0 bits. The previous CPU core clock divisor is reinstated when jumping back to program RAM from program flash memory. 00 = 1 cycle 01 = 2 cycles 10 = 3 cycles 11 = 4 cycles

Name	Description
FRDTC1:0	The number of CPU core cycles for reading the flash memory using an **`iread`** instruction must be specified to prevent the flash memory access time from being exceeded. Because the CPU core is subject to changes in speed, the value programmed in these bits should be appropriate for the fastest speed that might be used (typically, the faster of the main line code and the interrupt service routine). The FRDTC1:0 bits specify the number of CPU core clock cycles required for read access. 00 = 1 cycle 01 = 2 cycles 10 = 3 cycles 11 = 4 cycles

Name	Description
FWRT3:0	The flash memory erase and write timing is derived from the CPU core clock through a programmable divider. The FWRT3:0 bits specify the divisor. The time base must be 1 to 2 microseconds. Below 1 microsecond, the flash memory will be underprogrammed, and data retention is not guaranteed. Above 2 microseconds, the flash memory will be overprogrammed, and reliability is not guaranteed. 0000 = 2 0001 = 3 0010 = 4 0011 = 6 0100 = 8 0101 = 12 0110 = 16 0111 = 24 1000 = 32 1001 = 48 1010 = 64 1011 = 96 1100 = 128 1101 = 192 1110 = 256 1111 = 384

C.3.5 INTSPD Register

7	6	5	4	3	0
PWRD1:0		CLK1:0		CDIV3:0	

Name	Description
PWRD1	Controls PLL clock multiplier operation. If the PLL is not required, power consumption can be reduced by disabling it. 0 = PLL clock multiplier enabled 1 = PLL clock multiplier disabled
PWRD0	Controls OSC oscillator operation. If the oscillator is not required, power consumption can be reduced by disabling it. 0 = OSC oscillator enabled 1 = OSC oscillator disabled
CLK1:0	Selects the system clock source. 00 = PLL clock multiplier 01 = OSC oscillator/external clock on OSC1 input 10 = RTCLK oscillator/external clock on RTCLK1 input 11 = System clock disabled (off)

Name	Description
CDIV3:0	Selects the system clock divisor. 0000 = 1 0001 = 2 0010 = 3 0011 = 4 0100 = 5 0101 = 6 0110 = 8 0111 = 10 1000 = 12 1001 = 16 1010 = 24 1011 = 32 1100 = 48 1101 = 64 1110 = 128 1111 = System clock disabled (off)

C.3.6 LFSRA Register (LFSR Address)

7 4	3 0
UNIT3:0	INDEX3:0

Name	Description
UNIT3:0	LFSR unit (only 0, 1, 2, and 3 are valid)
INDEX3:0	LFSR register index (see Table 4-13)

C.3.7 PSPCFG Register (Parallel Slave Peripheral Config.)

7	6	5	4	3 … 0
PSPEN2	PSPEN1	PSPHEN	PSPRDY	Reserved

Name	Description
PSPEN2	Port D enable bit 0 = Port D is available for general-purpose I/O 1 = Port D is configured for the Parallel Slave Peripheral interface
PSPEN1	Port C enable bit 0 = Port C is available for general-purpose I/O 1 = Port C is configured for the Parallel Slave Peripheral interface
PSPHEN	$\overline{\text{HOLD}}$ output enable bit 0 = $\overline{\text{HOLD}}$ output disabled. Port pin RB5 available for general-purpose I/O. 1 = $\overline{\text{HOLD}}$ output enabled on port pin RB5.
PSPRDY	Ready bit 0 = This bit always reads as zero. 1 = Write 1 to release $\overline{\text{HOLD}}$ when the IP2022 is ready to allow the data transfer to complete.

C.3.8 RTCFG Register (Real-Time Timer Configuration)

7	6 — 3	2	1	0
RTEN	RTPS3:0	RTSS	RTIE	RTIF

Name	Description
RTEN	Real-Time Timer enable bit 0 = Real-Time Timer disabled 1 = Real-Time Timer enabled
RTPS3:0	Real-Time Timer prescaler divisor 0000 = 1 0001 = 2 0010 = 4 0011 = 8 0100 = 16 0101 = 32
	0110 = 64 0111 = 128 1000 = 256 1001 = 512 1010 = 1024 1011 = 2048 1100 = 4096 1101 = 8192 1110 = 16384 1111 = 32768

Name	Description
RTSS	Real-Time Timer clock source select 0 = pre-PLL clock 1 = external RTCLK
RTIE	Real-Time Timer interrupt enable bit 0 = Real-Time Timer interrupt disabled 1 = Real-Time Timer interrupt enabled
RTIF	Real-Time Timer interrupt flag 0 = No timer overflow has occurred since this bit was last cleared 1 = Timer overflow has occurred. This bit goes high two cycles after the actual overflow occurs.

C.3.9 SxINTF Register

7	6	5	4	3	2	1	0
RXERROR	EOP	SYND	TXBE	TXEOP	SXLINKPULSE	RXBF	RXBUSY

Name	Description
RXERROR	Receive error in preamble (Manchester encoding only) 0 = Receive error has not been detected since this bit was last cleared 1 = Double zero has been detected in preamble (not affected by double zero in data)
EOP	End-of-Packet detection interrupt flag (USB and 10Base-T modes only) 0 = End-of-Packet has not been detected since this bit was last cleared 1 = End-of-Packet has been detected
SYND	Synchronization pattern detection interrupt flag (USB and 10Base-T modes only) 0 = Synchronization pattern has not been detected since this bit was last cleared 1 = Synchronization pattern has been detected
TXBE	Transmit buffer empty interrupt flag 0 = Transmit buffer has not been empty since this bit was last cleared 1 = Transmit buffer has been empty

Name	Description
TXEOP	Transmit underrun. This bit is set when the previous data in the transmit buffer register (SxTXBUF) has been transmitted and no new data has been loaded in the register. In USB and 10Base-T modes, this causes an EOP condition to be generated. 0 = Transmit underrun has not occurred since this bit was last cleared 1 = Transmit underrun has occurred
SXLINKPULSE	Set after a link pulse of 75 to 250 ns duration is detected. 0 = No link pulse has been detected since this bit was last cleared 1 = Link pulse detected
RXBF	Receive buffer full interrupt flag 0 = Receive buffer has not been full since this bit was last cleared 1 = Receive buffer has been full
RXBUSY	Receive buffer busy interrupt flag. This bit can be used to check whether data is being received before entering a low-power mode. 0 = No new data has been received since this bit was last cleared 1 = New data has been (or is being) received

C.3.10 SxMODE Register

7	6	5	4	3	2	1	0
PRS3:0				SUBM1:0		CLKS1:0	

Name	Description
PRS3:0	Protocol select. Unassigned encodings are reserved. 0000 = Disabled 0001 = 10Base-T 0010 = USB Bus 0011 = UART 0100 = I^2C 0101 = SPI/Microwire
SUBM1:0	Submode select (in USB mode): 01 = Low-speed USB interface 10 = High-speed USB interface Submode select (in Microwire and SPI modes): 00 = Receive on falling edge, transmit on rising edge 01 = Receive on rising edge, transmit on rising edge 10 = Receive on falling edge, transmit on falling edge 11 = Receive on rising edge, transmit on falling edge

Name	Description
CLKS1:0	Clock source select 00 = Clock disabled 01 = Reserved 10 = OSC clock oscillator 11 = PLL clock multiplier

C.3.11 SxRCFG Register

7	6	5	4	0
Reserved	$\overline{\text{SYNCMP}}$	RPOREV	RXSCNT4:0	

Name	Description
$\overline{\text{SYNCMP}}$	Synchronization pattern detection disable bit 0 = Synchronization pattern detection enabled 1 = Synchronization pattern detection disabled
RPOREV	Receive data polarity reversal select 0 = Data polarity uninverted 1 = Data polarity inverted
RXSCNT4:0	Receive shift count, specifies number of bits to receive

C.3.12 SxRCNT Register

7	6	5	4	0
Reserved	RxCRSCTRL1:0		RXACNT4:0	

Name	Description
RxCRSCTRL1:0	Carrier sense status and interrupt control 0x = No interrupt. Software can poll the RXXCRS bit in the SxINTF register for carrier status. 10 = Intterupt on carrier detection 11 = Interrupt on loss of carrier
RXACNT4:0	Receive shift count, actual number of bits received (read-only)

C.3.13 SxTCFG Register

7	6	5	4	0
SEREN	Reserved	TPOREV	TXSCNT4:0	

Name	Description
SEREN	SERDES enable bit 0 = SERDES disabled 1 = SERDES enabled

Name	Description
TPOREV	Transmit data polarity reversal select 0 = Data polarity uninverted 1 = Data polarity inverted
TXSCNT4:0	Transmit shift count, specifies number of bits to transmit

C.3.14 SPDREG Register

7	6	5	4	3	0
$\overline{\text{PLL}}$	$\overline{\text{OSC}}$	CLK1:0		CDIV3:0	

Name	Description
$\overline{\text{PLL}}$	Controls PLL clock multiplier operation. If the PLL is not required, power consumption can be reduced by disabling it. 0 = PLL clock multiplier enabled 1 = PLL clock multiplier disabled
$\overline{\text{OSC}}$	Controls OSC oscillator operation. If the oscillator is not required, power consumption can be reduced by disabling it. 0 = OSC oscillator enabled 1 = OSC oscillator disabled
CLK1:0	Selects the system clock source. 00 = PLL clock multiplier 01 = OSC oscillator/external clock on OSC1 input 10 = RTCLK oscillator/external clock on RTCLK1 input 11 = System clock disabled (off)

Name	Description
CDIV3:0	Selects the system clock divisor. 0000 = 1 0001 = 2 0010 = 3 0011 = 4 0100 = 5 0101 = 6 0110 = 8 0111 = 10 1000 = 12 1001 = 16 1010 = 24 1011 = 32 1100 = 48 1101 = 64 1110 = 128 1111 = System clock disabled (off)

C.3.15 STATUS Register

7 5	4	3	2	1	0
PA2:0	WD	BO	Z	DC	C

Name	Description
PA2:0	Program memory page select bits. Used to extend the 13-bit address encoded in jump and call instructions. Modified using the **page** instruction.
WD	Watchdog time-out bit. Set at reset, if reset was triggered by Watchdog Timer overflow, otherwise cleared. 0 = Last reset was not caused by a Watchdog Timer overflow. 1 = Last reset was caused by a Watchdog Timer overflow.
BO	Brown-out reset bit. Set at reset, if reset was triggered by brown-out voltage level detection. 0 = Last reset was not caused by the brown-out voltage detector sensing a low-voltage condition. 1 = Last reset was caused by the brown-out voltage detector sensing a low-voltage condition.
Z	Zero bit. Affected by most logical, arithmetic, and data movement instructions. Set if the result was zero, otherwise cleared. 0 = Result of last ALU operation was non-zero. 1 = Result of last ALU operation was zero.

Name	Description
DC	Digit Carry bit. After addition, set if carry from bit 3 occurred, otherwise cleared. After subtraction, cleared if borrow from bit 3 occurred, otherwise set. 0 = Last addition did not generate carry out of bit 3, or last subtraction generated borrow out of bit 3. 1 = Last addition generated carry out of bit 3, or last subtraction did not generate borrow out of bit 3.
C	Carry bit. After addition, set if carry from bit 7 of the result occurred, otherwise cleared. After subtraction, cleared if borrow from bit 7 of the result occurred, otherwise set. After rotate (**`rr`** or **`rl`**) instructions, loaded with the LSB or MSB of the operand, respectively. 0 = Last addition did not generate carry out of bit 7, last subtraction generated borrow out of bit 7, or last rotate loaded a 0. 1 = Last addition generated carry out of bit 7, last subtraction did not generate borrow out of bit 7, or last rotate loaded a 1.

C.3.16 T0CFG Register (Timer 0 Configuration)

7	6 3	2	1	0
T0EN	T0PS3:0	Reserved	T0IE	T0IF

Name	Description
T0EN	Enables Timer 0 0 = Timer 0 disabled 1 = Timer 0 enabled
T0PS3:0	Specifies Timer 0 prescaler divisor 0000 = 1 0001 = 2 0010 = 4 0011 = 8 0100 = 16 0101 = 32 0110 = 64 0111 = 128 1000 = 256 1001 to 1111 = Reserved
T0IE	Timer 0 interrupt enable bit 0 = Timer 0 interrupt disabled 1 = Timer 0 interrupt enabled

Name	Description
T0IF	Timer 0 interrupt flag 0 = No timer overflow has occurred since this bit was last cleared 1 = Timer overflow has occurred

C.3.17 TxCFG1H/TxCFG1L Register

15	14	13	12	11	10	9	8
OFIE	CAP2IE CMP2IE	CAP1IE	CMP1IE	OFIF	CAP2IF CMP2IF	CAP1IF	CMP1IF

7	6	5	4	3	2	1	0
MODE	OEN	ECLKEN	CPI2EN	CPI1EN	ECLKEDG	CAP1RST	TMREN

Name	Description
OFIE	Timer overflow interrupt enable bit 0 = Overflow interrupt disabled 1 = Overflow interrupt enabled
CAP2IE CMP2IE	PWM mode: Compare 2 interrupt enable bit Capture/Compare mode: Capture 2 interrupt enable bit 0 = Capture/Compare 2 interrupt disabled 1 = Capture/Compare 2 interrupt enabled

Name	Description
CAP1IE	Capture 1 interrupt enable bit 0 = Capture 1 interrupt disabled 1 = Capture 1 interrupt enabled
CMP1IE	Compare 1 interrupt enable bit 0 = Compare 1 interrupt disabled 1 = Compare 1 interrupt enabled
OFIF	Timer overflow interrupt flag 0 = No timer overflow has occurred since this bit was last cleared 1 = Timer overflow has occurred
CAP2IF CMP2IF	PWM mode: Compare 2 interrupt flag (i.e. timer value matched TxCMP2 value) Capture/Compare mode: Capture 2 flag (i.e. TxCPI2 input triggered) 0 = No capture/compare 2 event has occurred since this bit was last cleared 1 = Capture/compare 2 event has occurred
CAP1IF	Capture 1 interrupt flag 0 = No capture 1 event has occurred since this bit was last cleared 1 = Capture 1 event has occurred
CMP1IF	Compare 1 interrupt flag 0 = No compare 1 event has occurred since this bit was last cleared 1 = Compare 1 event has occurred

Name	Description
MODE	Timer mode select 0 = PWM/timer mode 1 = Capture/compare mode
OEN	TxOUT enable bit 0 = TxOUT disabled. Port pin available for general-purpose I/O. 1 = TxOUT enabled. Port pin must be configured for output in corresponding RxDIR register bit.
ECLKEN	TxCLK enable bit 0 = TxCLK disabled. Port pin available for general-purpose I/O. 1 = TxCLK enabled as clock source for timer. Enabling this bit does not make any other restrictions on the use of the TxCLK port pin for general-purpose I/O.
CPI2EN	TxCPI2 enable bit 0 = System clock enabled as clock source for timer. TxCPI2 port pin available for general-purpose I/O. 1 = TxCLK enabled as clock source for timer. Enabling this bit does not make any other restrictions on the use of the port pin for general-purpose I/O.
CPI1EN	TxCPI1 enable bit 0 = Capture 1 input disabled. TxCPI1 port pin available for general-purpose I/O. 1 = TxCPI1 enabled as capture 1 input. Enabling this bit does not make any other restrictions on the use of the port pin for general-purpose I/O.
ECLKEDG	TxCLK edge sensitivity select. (This bit is ignored if the ECLKEN bit is clear.) 0 = TxCLK increments timer on rising edge 1 = TxCLK increments timer on falling edge

Name	Description
CAP1RST	Reset timer on capture 1 event enable bit 0 = Timer value unchanged by occurrence of a capture 1 event 1 = Timer value cleared by occurrence of a capture 1 event
TMREN	Timer enable bit 0 = Timer disabled. Timer clock source shut off to reduce power consumption. 1 = Timer enabled

C.3.18 TxCFG2H/TxCFG2L Register

15	14	13	12	11	8
0	0	0	0	PS3:0	

7	6	5	4	3	2	1	0
Reserved	TOUTSET	TOUTCLR	CPI2CPI1	CPI2EDG1:0		CPI1EDG1:0	

Name	Description
PS3:0	Timer prescaler divisor 0000 = 1 0001 = 2 0010 = 4 0011 = 8 0100 = 16 0101 = 32

Name	Description
	0110 = 64 0111 = 128 1000 = 256 1001 = 512 1010 = 1024 1011 = 2048 1100 = 4096 1101 = 8192 1110 = 16384 1111 = 32768
TOUTSET	Override bit to set the TxOUT output. This bit always reads as zero. 0 = Writing 0 to this bit has no effect 1 = Writing 1 to this bit forces the TxOUT signal high
TOUTCLR	Override bit to clear the TxOUT output. This bit always reads as zero. 0 = Writing 0 to this bit has no effect 1 = Writing 1 to this bit forces the TxOUT signal low
CPI2CPI1	Internally connect the TxCPI2 input to the TxCPI1 input. This makes the TxCPI2 port pin available for general-purpose I/O. 0 = No internal connection between TxCPI1 and TxCPI2 1 = TxCPI1 and TxCPI2 internally connected

Name	Description
CPI2EDG1:0	TxCPI2 edge sensitivity select 00 = Falling edge on TXCPI2 recognized as capture 2 event 01 = Rising edge on TXCPI2 recognized as capture 2 event 10 = Any falling or rising edge on TXCPI2 recognized as capture 2 event 11 = Any falling or rising edge on TXCPI2 recognized as capture 2 event
CPI1EDG1:0	TxCPI1 edge sensitivity select 00 = Falling edge on TXCPI1 recognized as capture 1 event 01 = Rising edge on TXCPI1 recognized as capture 1 event 10 = Any falling or rising edge on TXCPI1 recognized as capture 1 event 11 = Any falling or rising edge on TXCPI1 recognized as capture 1 event

C.3.19 TCTRL Register

7	6	5	4	3	2	1	0
0	0	T2IE	T1IE	0	0	T2RST	T1RST

Name	Description
T2IE	Timer 2 interrupt enable 0 = Timer 2 interrupt disabled 1 = Timer 2 interrupt enabled
T1IE	Timer 1 interrupt enable 0 = Timer 1 interrupt disabled 1 = Timer 1 interrupt enabled
T2RST	Timer 2 reset bit. This bit always reads as zero. 0 = Writing 0 to this bit has no effect. 1 = Writing 1 to this bit clears Timer 2.
T1RST	Timer 1 reset bit. This bit always reads as zero. 0 = Writing 0 to this bit has no effect. 1 = Writing 1 to this bit clears Timer 1.

C.3.20 XCFG Register

7	6	5	4	3	2	1	0
GIE	$\overline{\text{FWP}}$	RTEOS	$\overline{\text{RTOSC\_EN}}$	INT_EN	Reserved		FBUSY

Name	Description
GIE	Global interrupt enable bit. For more information about interrupt processing, see Section 2.5. 0 = Interrupts disabled 1 = Interrupts enabled
$\overline{\text{FWP}}$	Flash write protect bit. This bit only affects operation of the self-programming instructions, not programming through the ISD/ISP interface. For more information about programming the flash memory, see Section 3.7. 0 = Writes to flash memory disabled 1 = Writes to flash memory enabled
RTEOS	Real-time timer oversampling bit. For more information about the real-time timer, see Section 4.3. 0 = Oversampling disabled 1 = Oversampling enabled
$\overline{\text{RTOSC\_EN}}$	RTCLK oscillator enable bit 0 = RTCLK oscillator is operational 1 = RTCLK oscillator turned off

Name	Description
INT_EN	**`int`** instruction interrupt enable bit 0 = **`int`** instructions only increment the PC, like **`nop`** 1 = **`int`** instructions cause interrupts
FBUSY	Flash memory busy bit (read-only). For more information about programming the flash memory, see Section 3.7. 0 = Flash memory is idle 1 = CPU is fetching an instruction out of flash memory or processing an **`iread`**, **`fwrite`**, or **`ferase`** instruction