ATmega328PB

AVR® Microcontroller with Core Independent Peripherals and PicoPower® Technology

Introduction

The picoPower® ATmega328PB is a low-power CMOS 8-bit microcontroller based on the AVR® enhanced RISC architecture. By executing powerful instructions in a single clock cycle, the ATmega328PB achieves throughputs close to 1 MIPS per MHz. This empowers system designers to optimize the device for power consumption versus processing speed.

Features

High Performance, Low-Power AVR® 8-bit Microcontroller Family

- Advanced RISC Architecture
  - 131 Powerful Instructions
  - Most Single Clock Cycle Execution
  - 32 x 8 General Purpose Working Registers
  - Fully Static Operation
  - Up to 20 MIPS Throughput at 20 MHz
  - On-Chip 2-Cycle Multiplier
- High Endurance Nonvolatile Memory Segments
  - 32 KB of In-System Self-Programmable Flash program memory
  - 1 KB EEPROM
  - 2 KB Internal SRAM
  - Write/Erase Cycles: 10,000 Flash/100,000 EEPROM
  - Optional Boot Code Section with Independent Lock Bits
    - In-System Programming by On-chip Boot Program
    - True Read-While-Write Operation
  - Programming Lock for Software Security
- Peripheral Features
  - Peripheral Touch Controller (PTC)
    - Capacitive Touch Buttons, Sliders, and Wheels
    - 24 Self-Cap Channels and 144 Mutual Cap Channels
  - Two 8-bit Timer/Counters with Separate Prescaler and Compare Mode
  - Three 16-bit Timer/Counters with Separate Prescaler, Compare Mode, and Capture Mode
  - Real-Time Counter with Separate Oscillator
  - Ten PWM Channels
  - 8-channel 10-bit ADC

  - Two Programmable Serial USARTs
  - Two Master/Slave SPI Serial Interfaces
  - Two Byte-Oriented Two-Wire Serial Interfaces (Philips I^2C Compatible)
  - Programmable Watchdog Timer with Separate On-chip Oscillator
  - On-Chip Analog Comparator
  - Interrupt and Wake-Up on Pin Change
- Special Microcontroller Features
  - Power-On Reset and Programmable Brown-Out Detection
  - Internal 8 MHz Calibrated Oscillator
  - External and Internal Interrupt Sources
  - Six Sleep Modes: Idle, ADC Noise Reduction, Power-save, Power-down, Standby, and Extended Standby
  - Clock Failure Detection Mechanism and Switch to Internal 8 MHz RC Oscillator in case of Failure
  - Individual Serial Number to Represent a Unique ID
- I/O and Packages
  - 27 Programmable I/O Lines
  - 32-pin TQFP and 32-pin QFN /MLF
- Operating Voltage:
  - 1.8 - 5.5V
- Temperature Range:
  - -40°C to 105°C
- Speed Grade:
  - 0 - 4 MHz @ 1.8 - 5.5V
  - 0 - 10 MHz @ 2.7 - 5.5V
  - 0 - 20 MHz @ 4.5 - 5.5V
- Power Consumption at 1 MHz, 1.8V, 25°C
  - Active Mode: 0.24 mA
  - Power-Down Mode: 0.2 µA
  - Power-Save Mode: 1.3 µA (Including 32 kHz RTC)

Table of Contents

1. Description

The ATmega328PB is a low-power CMOS 8-bit microcontroller based on the AVR enhanced RISC architecture. By executing powerful instructions in a single clock cycle, the ATmega328PB achieves throughputs close to 1 MIPS per MHz. This empowers system designer to optimize the device for power consumption versus processing speed.

The core combines a rich instruction set with 32 general purpose working registers. All the 32 registers are directly connected to the Arithmetic Logic Unit (ALU), allowing two independent registers to be accessed in a single instruction executed in one clock cycle. The resulting architecture is more code efficient while achieving throughputs up to ten times faster than conventional CISC microcontrollers.

The ATmega328PB provides the following features: 32 KB of In-System Programmable Flash with Read-While-Write capabilities, 1 KB EEPROM, 2 KB SRAM, 27 general purpose I/O lines, 32 general purpose working registers, five flexible Timer/Counters with compare modes, internal and external interrupts, two serial programmable USART, two byte-oriented two-wire Serial Interface (I^2C), two SPI serial ports, an 8-channel 10-bit ADC in TQFP and QFN/MLF package, a programmable Watchdog Timer with internal Oscillator, Clock failure detection mechanism, and six software selectable power saving modes. The Idle mode stops the CPU while allowing the SRAM, Timer/Counters, USART, two-wire Serial Interface, SPI port, and interrupt system to continue functioning. PTC with enabling up to 24 self-cap and 144 mutual-cap sensors. The Power-Down mode saves the register contents but freezes the Oscillator, disabling all other chip functions until the next interrupt or hardware reset. In Power-Save mode, the asynchronous timer continues to run, allowing the user to maintain a timer base while the rest of the device is sleeping. Also ability to run PTC in Power-Save mode/wake-up on touch and Dynamic ON/OFF of PTC analog and digital portion. The ADC Noise Reduction mode stops the CPU and all I/O modules except asynchronous timer, PTC, and ADC to minimize switching noise during ADC conversions. In Standby mode, the crystal/resonator Oscillator is running while the rest of the device is sleeping. This allows very fast start-up combined with low power consumption.

The device is manufactured using high-density non-volatile memory technology. The On-chip ISP Flash allows the program memory to be reprogrammed In-System through an SPI serial interface, by a conventional nonvolatile memory programmer or by an On-chip Boot program running on the AVR core. The Boot program can use any interface to download the application program in the Application Flash memory. Software in the Boot Flash section will continue to run while the Application Flash section is updated, providing true Read-While-Write operation. By combining an 8-bit RISC CPU with In-System Self-Programmable Flash on a monolithic chip, the ATmega328PB is a powerful microcontroller that provides a highly flexible and cost-effective solution to many embedded control applications.

The ATmega328PB is supported by a full suite of program and system development tools including C Compilers, Macro Assemblers, Program Debugger/Simulators, In-Circuit Emulators, and Evaluation kits.

2. Configuration Summary

Features	ATmega328PB
Pin count	32
Flash (KB)	32
SRAM (KB)	2
EEPROM (KB)	1
General Purpose I/O pins	27
SPI	2
TWI (I^2C)	2
USART	2
ADC	10-bit 15 ksps
ADC channels	8
AC propagation delay	400 ns (Typical)
8-bit Timer/Counters	2
16-bit Timer/Counters	3
PWM channels	10
PTC	Available
Clock Failure Detector (CFD)	Available
Output Compare Modulator (OCM1C2)	Available

3. Ordering Information

Speed [MHz]	Power Supply [V]	Ordering Code[2]	Package[1]	Operational Range
20	1.8 - 5.5	ATmega328PB-AU ATmega328PB-AUR[3] ATmega328PB-MU ATmega328PB-MUR[3]	32A 32A 32MS1 32MS1	Industrial (-40°C to 85°C)
		ATmega328PB-AN ATmega328PB-ANR[3] ATmega328PB-MN ATmega328PB-MNR[3]	32A 32A 32MS1 32MS1	Industrial (-40°C to 105°C)

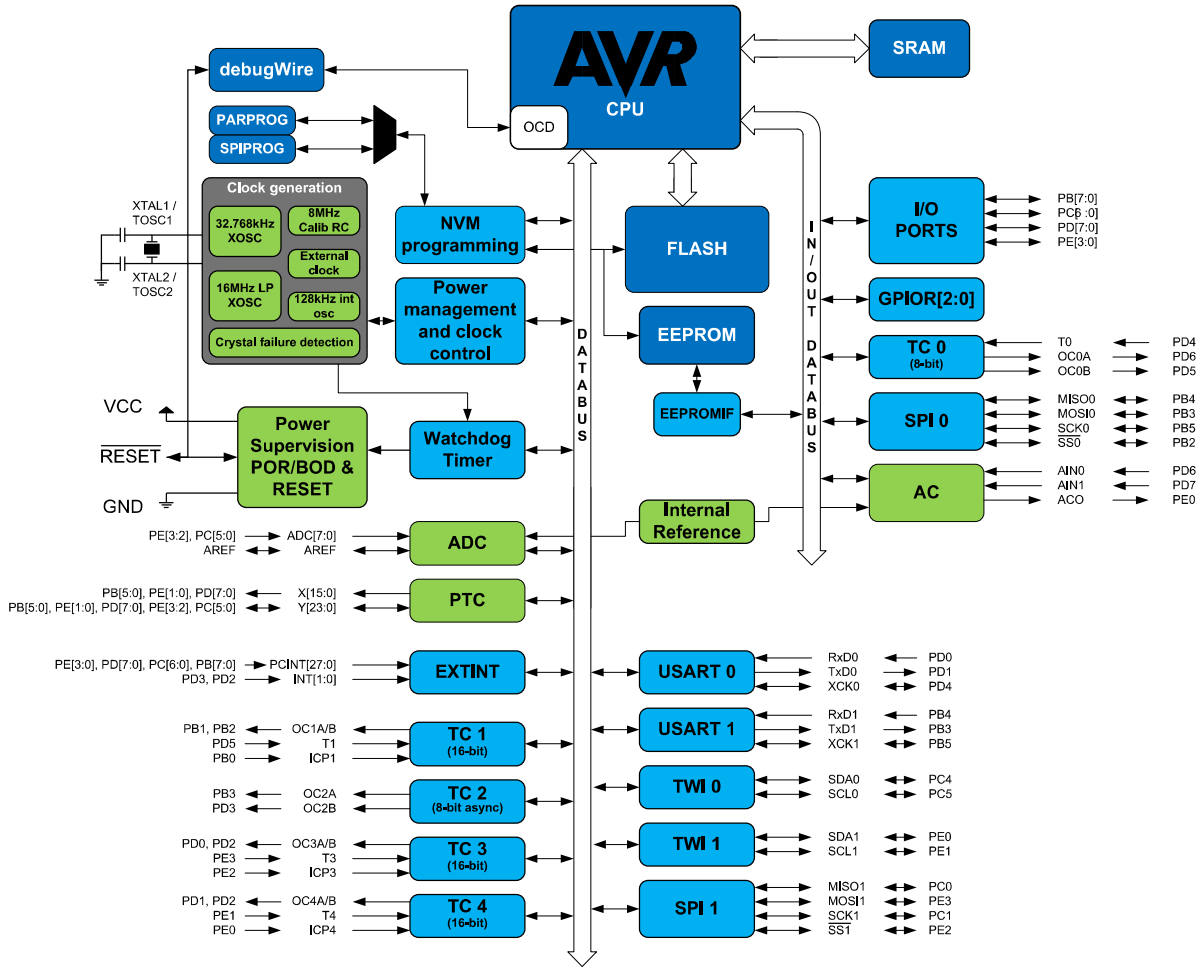
Note:

1. This device can also be supplied in wafer form. Contact your local Microchip sales office for detailed ordering information and minimum quantities.
2. Pb-free packaging complies to the European Directive for Restriction of Hazardous Substances (RoHS directive). Also Halide free and fully Green.
3. Tape & Reel.

Package Type	
32A	32-lead, Thin (1.0 mm) Plastic Quad Flat Package (TQFP)
32MS1	32-pad, 5.0x5.0x0.9 mm body, Lead Pitch 0.50 mm, Very-thin Fine pitch, Quad Flat No Lead Package (VQFN)

4. Block Diagram

Figure 4-1. Block Diagram



5. Pin Configurations

Figure 5-1. 32 TQFP Pinout ATmega328PB

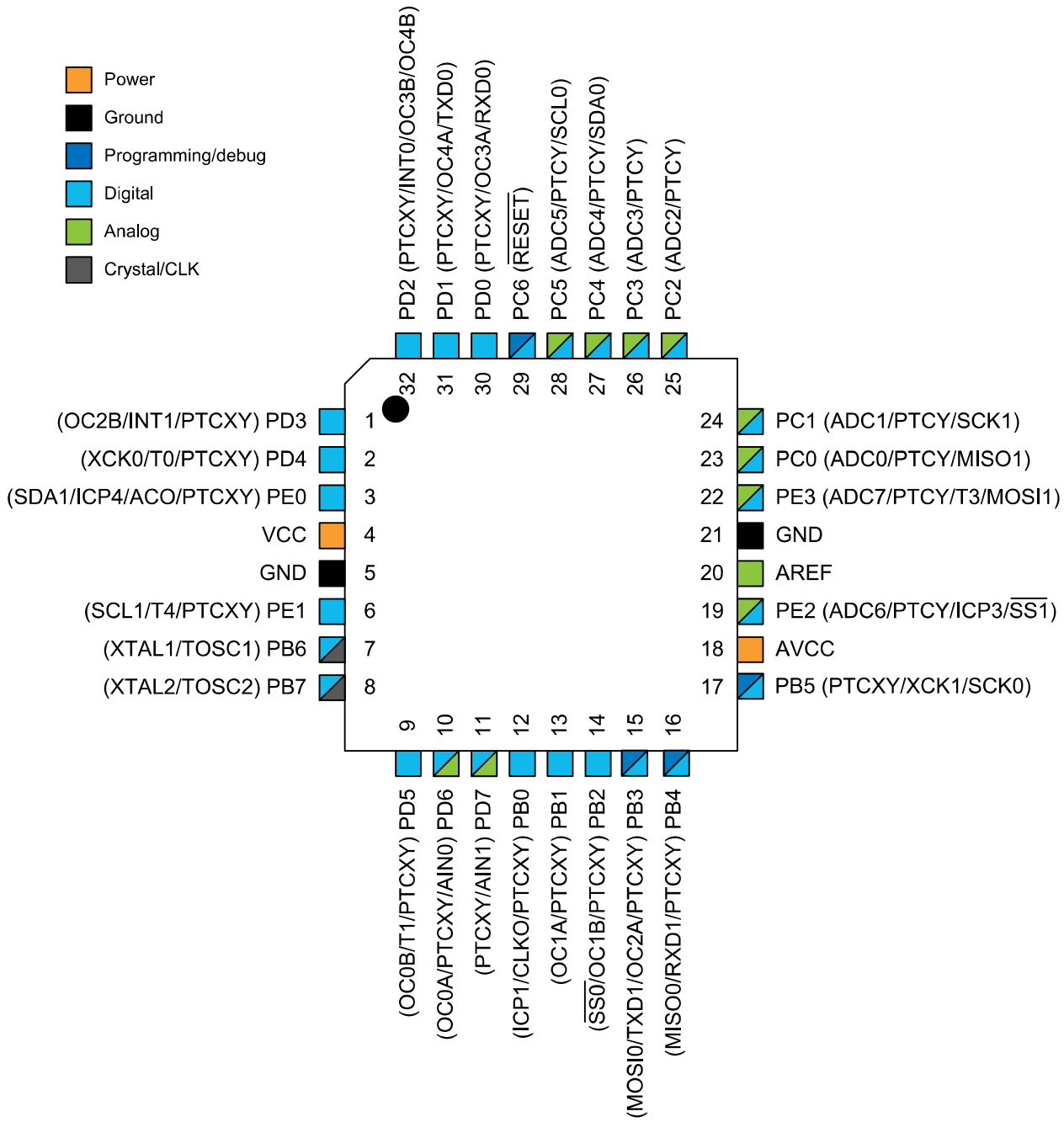
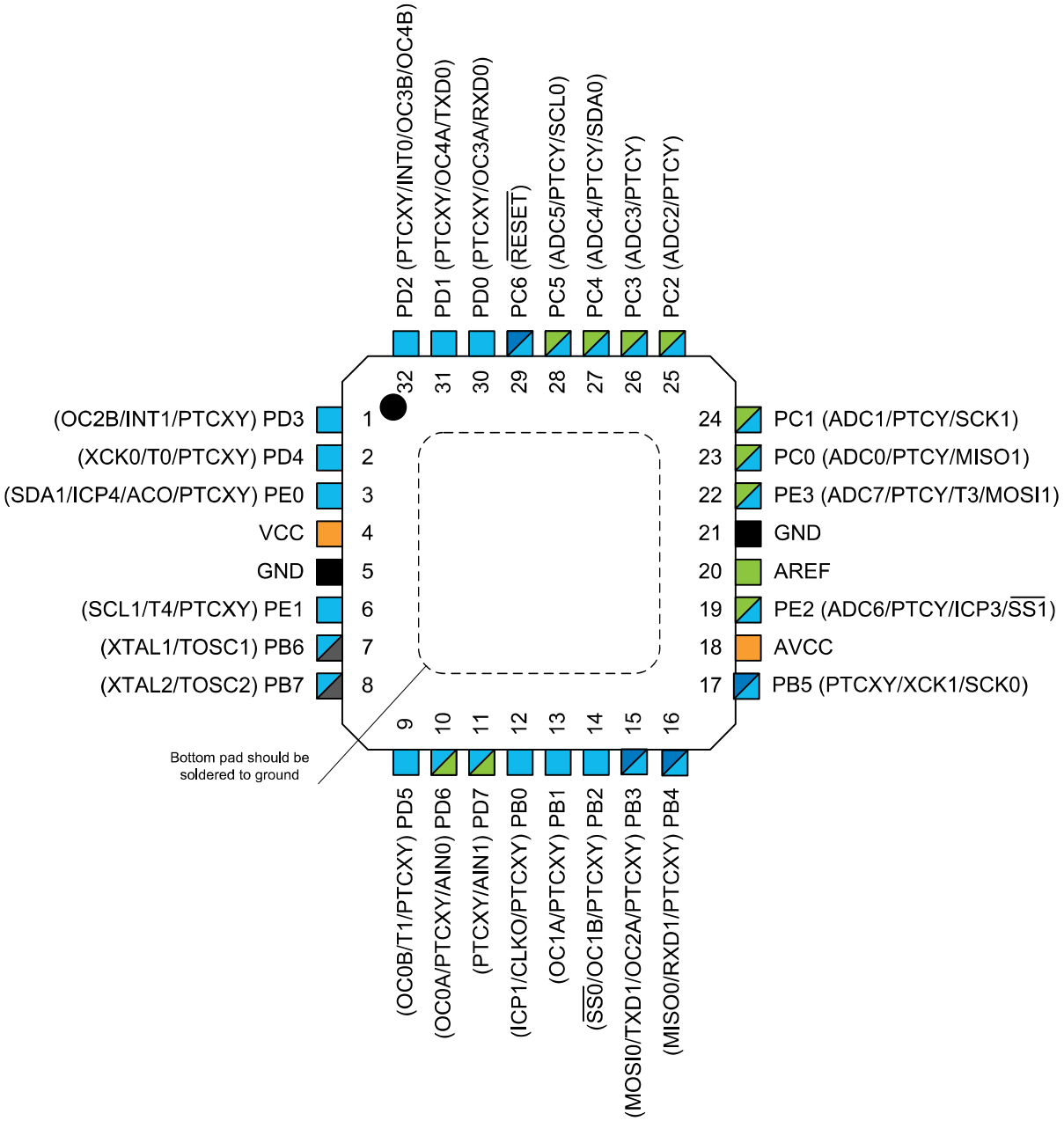


Figure 5-2. 32 VQFN Pinout ATmega328PB



5.1 Pin Descriptions

5.1.1 VCC

Digital supply voltage pin.

5.1.2 GND

Ground.

5.1.3 Port B (PB[7:0]) XTAL1/XTAL2/TOSC1/TOSC2

Port B is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each pin). The Port B output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port B pins that are externally pulled low will source current if the pull-up resistors are activated. The Port B pins are tri-stated during a reset condition even if the clock is not running.

Depending on the clock selection fuse settings, PB6 can be used as input to the inverting Oscillator amplifier and input to the internal clock operating circuit.

Depending on the clock selection fuse settings, PB7 can be used as output from the inverting Oscillator amplifier.

If the Internal Calibrated RC Oscillator is used as chip clock source, PB[7:6] is used as TOSC[2:1] input for the Asynchronous Timer/Counter2 if the AS2 bit in ASSR is set.

5.1.4 Port C (PC[5:0])

Port C is a 7-bit bi-directional I/O port with internal pull-up resistors (selected for each pin). The PC[5:0] output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port C pins that are externally pulled low will source current if the pull-up resistors are activated. The Port C pins are tri-stated during a reset condition even if the clock is not running.

5.1.5 PC6/$\overline{\text{RESET}}$

If the RSTDISBL Fuse is programmed, PC6 is used as an I/O pin. Note that the electrical characteristics of PC6 differ from those of the other pins of Port C.

If the RSTDISBL Fuse is unprogrammed, PC6 is used as a Reset input. A low level on this pin for longer than the minimum pulse length will generate a Reset, even if the clock is not running. Shorter pulses are not guaranteed to generate a Reset.

The various special features of Port C are elaborated in the *Alternate Functions of Port C* section.

5.1.6 Port D (PD[7:0])

Port D is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each pin). The Port D output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port D pins that are externally pulled low will source current if the pull-up resistors are activated. The Port D pins are tri-stated during a reset condition even if the clock is not running.

5.1.7 Port E (PE[3:0])

Port E is a 4-bit bi-directional I/O port with internal pull-up resistors (selected for each pin). The Port E output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port E pins that are externally pulled low will source current if the pull-up resistors are activated. The Port E pins are tri-stated during a reset condition even if the clock is not running.

5.1.8 AV_{CC}

AV_{CC} is the supply voltage pin for the A/D Converter, PC[3:0], and PE[3:2]. It should be externally connected to V_{CC}, even if the ADC is not used. If the ADC is used, it should be connected to V_{CC} through a low-pass filter. Note that PC[6:4] use digital supply voltage, V_{CC}.

5.1.9 AREF

AREF is the analog reference pin for the A/D Converter.

5.1.10 ADC[7:6]

In the TQFP and VFQFN package, ADC[7:6] serve as analog inputs to the A/D converter. These pins are powered by the analog supply and serve as 10-bit ADC channels.

6. I/O Multiplexing

Each pin is by default controlled by the PORT as a general purpose I/O and alternatively, it can be assigned to one of the peripheral functions.

The following table describes the peripheral signals multiplexed to the PORT I/O pins.

Table 6-1. PORT Function Multiplexing

No	PAD	EXTINT	PCINT	ADC/AC	PTC X	PTC Y	OSC	T/C	USART	I2C	SPI
1	PD[3]	INT1	PCINT19		X3	Y11		OC2B			
2	PD[4]		PCINT20		X4	Y12		T0	XCK0		
3	PE[0]		PCINT24	ACO	X8	Y16		ICP4		SDA1	
4	VCC										
5	GND										
6	PE[1]		PCINT25		X9	Y17		T4		SCL1	
7	PB[6]		PCINT6				XTAL1/TOSC1				
8	PB[7]		PCINT7				XTAL2/TOSC2				
9	PD[5]		PCINT21		X5	Y13		OC0B / T1			
10	PD[6]		PCINT22	AIN0	X6	Y14		OC0A			
11	PD[7]		PCINT23	AIN1	X7	Y15					
12	PB[0]		PCINT0		X10	Y18	CLKO	ICP1			
13	PB[1]		PCINT1		X11	Y19		OC1A			
14	PB[2]		PCINT2		X12	Y20		OC1B			$\overline{SS0}$
15	PB[3]		PCINT3		X13	Y21		OC2A	TXD1		MOSI0
16	PB[4]		PCINT4		X14	Y22			RXD1		MISO0
17	PB[5]		PCINT5		X15	Y23			XCK1		SCK0
18	AVCC										
19	PE[2]		PCINT26	ADC6		Y6		ICP3			$\overline{SS1}$
20	AREF										
21	GND										
22	PE[3]		PCINT27	ADC7		Y7		T3			MOSI1
23	PC[0]		PCINT8	ADC0		Y0					MISO1
24	PC[1]		PCINT9	ADC1		Y1					SCK1
25	PC[2]		PCINT10	ADC2		Y2					
26	PC[3]		PCINT11	ADC3		Y3					
27	PC[4]		PCINT12	ADC4		Y4				SDA0	
28	PC[5]		PCINT13	ADC5		Y5				SCL0	
29	PC[6]/$\overline{RESET}$		PCINT14								
30	PD[0]		PCINT16		X0	Y8		OC3A	RXD0		

No	PAD	EXTINT	PCINT	ADC/AC	PTC X	PTC Y	OSC	T/C	USART	I2C	SPI
31	PD[1]		PCINT17		X1	Y9		OC4A	TXD0		
32	PD[2]	INT0	PCINT18		X2	Y10		OC3B / OC4B			

7. Resources

A comprehensive set of development tools, application notes, and datasheets are available for download on http://www.microchip.com/design-centers/8-bit/microchip-avr-mcus.

8. About Code Examples

This documentation contains simple code examples that briefly show how to use various parts of the device. These code examples assume that the part specific header file is included before compilation. Be aware that not all C compiler vendors include bit definitions in the header files and interrupt handling in C is compiler dependent. Confirm with the C compiler documentation for more details.

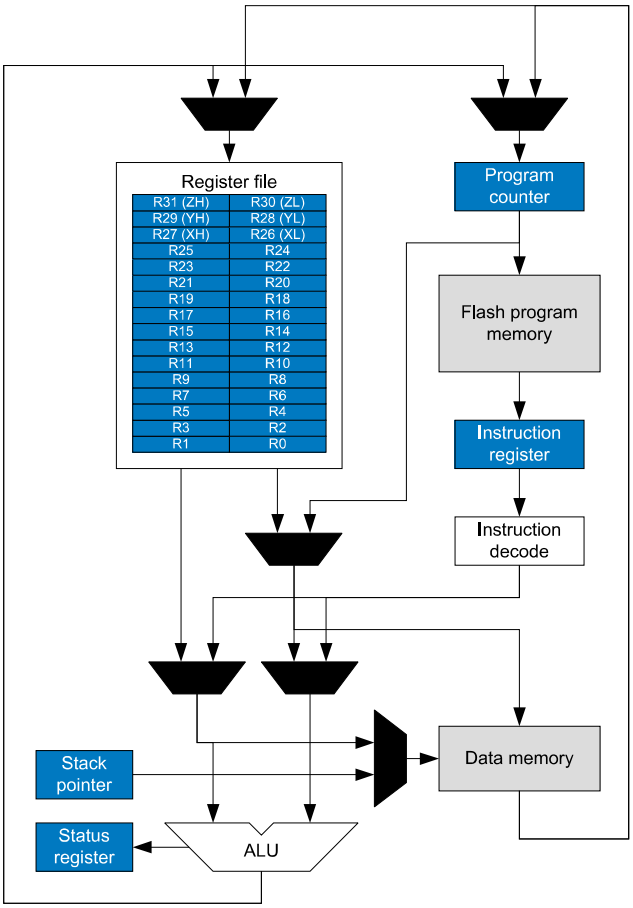
For I/O Registers located in extended I/O map, “IN”, “OUT”, “SBIS”, “SBIC”, “CBI”, and “SBI” instructions must be replaced with instructions that allow access to extended I/O. Typically “LDS” and “STS” combined with “SBRS”, “SBRC”, “SBR”, and “CBR”.

9. AVR CPU Core

9.1 Overview

This section discusses the AVR core architecture in general. The main function of the CPU core is to ensure correct program execution. The CPU must, therefore, be able to access memories, perform calculations, control peripherals, and handle interrupts.

Figure 9-1. Block Diagram of the AVR Architecture



In order to maximize performance and parallelism, the AVR uses a Harvard architecture – with separate memories and buses for program and data. Instructions in the program memory are executed with a single level pipelining. While one instruction is being executed, the next instruction is pre-fetched from the program memory. This concept enables instructions to be executed in every clock cycle. The program memory is In-System Reprogrammable Flash memory.

The fast-access register file contains 32 x 8-bit general purpose working registers with a single clock cycle access time. This allows single-cycle Arithmetic Logic Unit (ALU) operation. In a typical ALU operation, two operands are output from the register file, the operation is executed, and the result is stored back in the register file – in one clock cycle.

Six of the 32 registers can be used as three 16-bit indirect address register pointers for data space addressing – enabling efficient address calculations. One of these address pointers can be used as an

address pointer for lookup tables in Flash program memory. These added function registers are the 16-bit X-, Y-, and Z-register, described later in this section.

The ALU supports arithmetic and logic operations between registers or between a constant and a register. Single register operations can also be executed in the ALU. After an arithmetic operation, the Status register is updated to reflect information about the result of the operation.

Program flow is provided by conditional and unconditional jump and call instructions, able to directly address the whole address space. Most AVR instructions have a single 16-bit word format. Every program memory address contains a 16- or 32-bit instruction.

Program Flash memory space is divided into two sections, the Boot Program section and the Application Program section. Both sections have dedicated Lock bits for write and read/write protection. The SPM instruction that writes into the Application Flash memory section must reside in the Boot Program section.

During interrupts and subroutine calls, the return address Program Counter (PC) is stored on the Stack. The Stack is effectively allocated in the general data SRAM, and consequently, the Stack size is only limited by the total SRAM size and the usage of the SRAM. All user programs must initialize the Stack Pointer (SP) in the Reset routine (before subroutines or interrupts are executed). The SP is read/write accessible in the I/O space. The data SRAM can easily be accessed through the five different addressing modes supported in the AVR architecture.

The memory spaces in the AVR architecture are all linear and regular memory maps.

A flexible interrupt module has its control registers in the I/O space with an additional global interrupt enable bit in the Status register. All interrupts have a separate interrupt vector in the interrupt vector table. The interrupts have priority in accordance with their interrupt vector position. The lower the interrupt vector address, the higher the priority.

The I/O memory space contains 64 addresses for CPU peripheral functions as Control registers, SPI, and other I/O functions. The I/O memory can be accessed directly, or as the data space locations following those of the register file, 0x20 - 0x5F. In addition, this device has extended I/O space from 0x60 - 0xFF in SRAM where only the ST/STS/STD and LD/LDS/LDD instructions can be used.

9.2 ALU – Arithmetic Logic Unit

The high-performance AVR ALU operates in direct connection with all the 32 general purpose working registers. Within a single clock cycle, arithmetic operations between general purpose registers or between a register and an immediate are executed. The ALU operations are divided into three main categories – arithmetic, logical, and bit-functions. Some implementations of the architecture also provide a powerful multiplier supporting both signed/unsigned multiplication and fractional format. See *Instruction Set Summary* section for a detailed description.

Related Links

Instruction Set Summary

9.3 Status Register

The Status register contains information about the result of the most recently executed arithmetic instruction. This information can be used for altering program flow in order to perform conditional operations. The Status register is updated after all ALU operations, as specified in the instruction set reference. This will in many cases remove the need for using the dedicated compare instructions, resulting in faster and more compact code.

The Status register is not automatically stored when entering an interrupt routine and restored when returning from an interrupt. This must be handled by software.

9.3.1 Status Register

Name: SREG
Offset: 0x5F
Reset: 0x00
Property: When addressing as I/O Register: address offset is 0x3F

When addressing I/O registers as data space using LD and ST instructions, the provided offset must be used. When using the I/O specific commands IN and OUT, the offset is reduced by 0x20, resulting in an I/O address offset within 0x00 - 0x3F.

The device is a complex microcontroller with more peripheral units than can be supported within the 64 locations reserved in Opcode for the IN and OUT instructions. For the extended I/O space from 0x60 in SRAM, only the ST/STS/STD and LD/LDS/LDD instructions can be used.

Bit	7	6	5	4	3	2	1	0
	I	T	H	S	V	N	Z	C
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit 7 – I Global Interrupt Enable
The global interrupt enable bit must be set for the interrupts to be enabled. The individual interrupt enable control is then performed in separate control registers. If the Global Interrupt Enable register is cleared, none of the interrupts are enabled independent of the individual interrupt enable settings. The I-bit is cleared by hardware after an interrupt has occurred, and is set by the RETI instruction to enable subsequent interrupts. The I-bit can also be set and cleared by the application with the SEI and CLI instructions, as described in the instruction set reference.

Bit 6 – T Copy Storage
The bit copy instructions BLD (Bit LoaD) and BST (Bit STore) use the T-bit as source or destination for the operated bit. A bit from a register in the register file can be copied into T by the BST instruction, and a bit in T can be copied into a bit in a register in the register file by the BLD instruction.

Bit 5 – H Half Carry Flag
The half carry flag H indicates a half carry in some arithmetic operations. Half carry flag is useful in BCD arithmetic. See the *Instruction Set Description* for detailed information.

Bit 4 – S Sign Flag, S = N ⊕ V

The S-bit is always an exclusive or between the negative flag N and the two's complement overflow flag V. See the *Instruction Set Description* for detailed information.

Bit 3 – V Two's Complement Overflow Flag
The two's complement overflow flag V supports two's complement arithmetic. See the *Instruction Set Description* for detailed information.

Bit 2 – N Negative Flag
The negative flag N indicates a negative result in an arithmetic or logic operation. See the *Instruction Set Description* for detailed information.

Bit 1 – Z Zero Flag
The zero flag Z indicates a zero result in an arithmetic or logic operation. See the *Instruction Set Description* for detailed information.

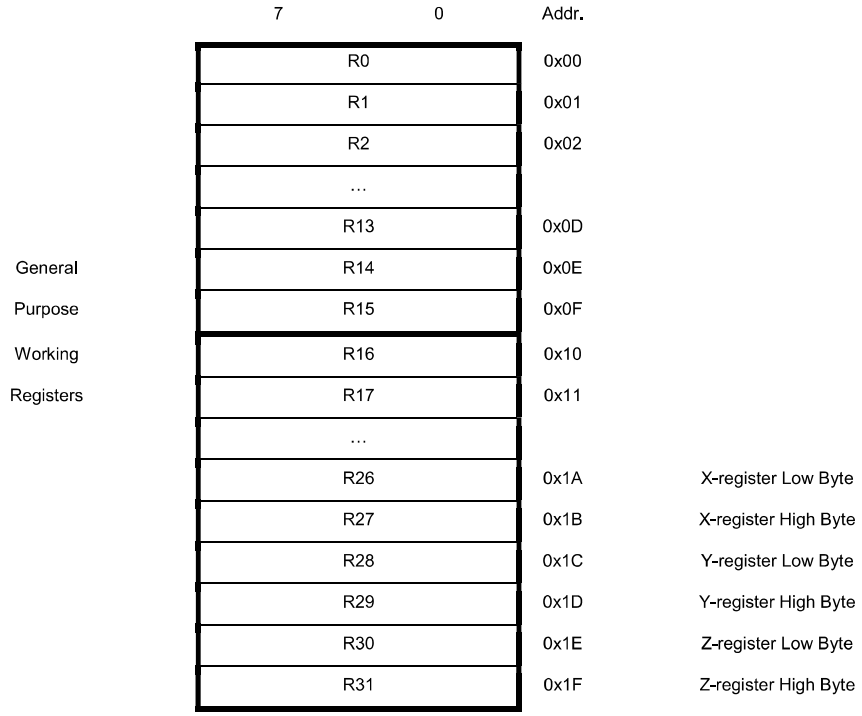
Bit 0 – C Carry Flag
The carry flag C indicates a carry in an arithmetic or logic operation. See the *Instruction Set Description* for detailed information.

9.4 General Purpose Register File

The register file is optimized for the AVR Enhanced RISC instruction set. In order to achieve the required performance and flexibility, the following input/output schemes are supported by the register file:

- One 8-bit output operand and one 8-bit result input
- Two 8-bit output operands and one 8-bit result input
- Two 8-bit output operands and one 16-bit result input
- One 16-bit output operand and one 16-bit result input

Figure 9-2. AVR CPU General Purpose Working Registers



Most of the instructions operating on the register file have direct access to all registers, and most of them are single cycle instructions. As shown in the figure, each register is also assigned a data memory address, mapping them directly into the first 32 locations of the user data space. Although not being physically implemented as SRAM locations, this memory organization provides great flexibility in access of the registers, as the X-, Y-, and Z-pointer registers can be set to index any register in the file.

9.4.1 The X-register, Y-register, and Z-register

The registers R26...R31 have some added functions to their general purpose usage. These registers are 16-bit address pointers for indirect addressing of the data space. The three indirect address registers X, Y, and Z are defined as described in the figure.

Figure 9-3. The X-, Y-, and Z-registers

In the different addressing modes, these address registers have functions as fixed displacement, automatic increment, and automatic decrement (see the instruction set reference for details).

Related Links

Instruction Set Summary

9.5 Stack Pointer

The Stack is mainly used for storing temporary data, local variables, and return addresses after interrupts and subroutine calls. The Stack is implemented as growing from higher to lower memory locations. The Stack Pointer register always points to the top of the Stack.

The Stack Pointer points to the data SRAM Stack area where the Subroutine and Interrupt Stacks are located. A Stack PUSH command will decrease the Stack Pointer. The Stack in the data SRAM must be defined by the program before any subroutine calls are executed or interrupts are enabled. Initial Stack Pointer value equals the last address of the internal SRAM and the Stack Pointer must be set to point above start of the SRAM. See the table for Stack Pointer details.

Table 9-1. Stack Pointer Instructions

Instruction	Stack Pointer	Description
PUSH	Decremented by 1	Data is pushed onto the stack
CALL ICALL RCALL	Decremented by 2	Return address is pushed onto the stack with a subroutine call or interrupt
POP	Incremented by 1	Data is popped from the stack
RET RETI	Incremented by 2	Return address is popped from the stack with return from subroutine or return from interrupt

The AVR Stack Pointer is implemented as two 8-bit registers in the I/O space. The number of bits actually used is implementation dependent. Note that the data space in some implementations of the AVR architecture is so small that only SPL is needed. In this case, the SPH register will not be present.

9.5.1 Stack Pointer Register Low and High byte

Name: SPL and SPH
Offset: 0x5D
Reset: 0x4FF
Property: When addressing I/O Registers as data space the offset address is 0x3D

The SPL and SPH register pair represents the 16-bit value, SP. The low byte [7:0] (suffix L) is accessible at the original offset. The high byte [15:8] (suffix H) can be accessed at offset + 0x01. For more details on reading and writing 16-bit registers, refer to *Accessing 16-bit Timer/Counter Registers*.

When using the I/O specific commands IN and OUT, the I/O addresses 0x00 - 0x3F must be used. When addressing I/O registers as data space using LD and ST instructions, 0x20 must be added to these offset addresses. The device is a complex microcontroller with more peripheral units than can be supported within the 64 locations reserved in Opcode for the IN and OUT instructions. For the extended I/O space from 0x60 in SRAM, only the ST/STS/STD and LD/LDS/LDD instructions can be used.

Bit	15	14	13	12	11	10	9	8
					SP11	SP10	SP9	SP8
Access	R	R	R	R	RW	RW	RW	RW
Reset	0	0	0	0	0	1	0	0

Bit	7	6	5	4	3	2	1	0
	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0
Access	RW	RW	RW	RW	RW	RW	RW	RW
Reset	1	1	1	1	1	1	1	1

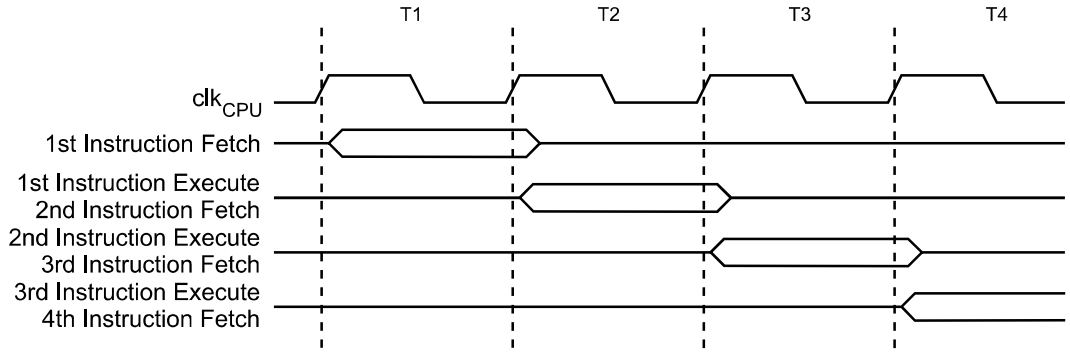
Bits 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 – SP Stack Pointer Register
SPL and SPH are combined into SP.

Related Links
Accessing 16-bit Timer/Counter Registers

9.6 Instruction Execution Timing

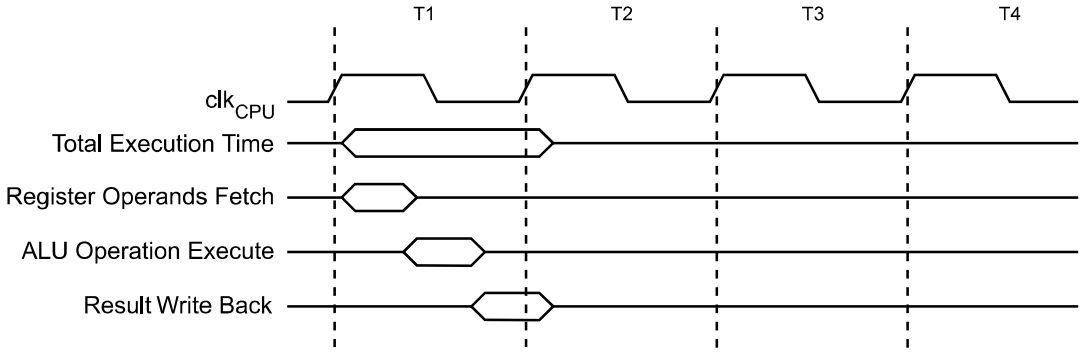
This section describes the general access timing concepts for instruction execution. The AVR CPU is driven by the CPU clock clk_{CPU}, directly generated from the selected clock source for the chip. No internal clock division is used. The figure below shows the parallel instruction fetches and instruction executions enabled by the Harvard architecture and the fast-access register file concept. This is the basic pipelining concept to obtain up to 1 MIPS per MHz with the corresponding unique results for functions per cost, functions per clocks, and functions per power unit.

Figure 9-4. The Parallel Instruction Fetches and Instruction Executions



The following figure shows the internal timing concept for the register file. In a single clock cycle, an ALU operation using two register operands is executed and the result is stored back to the destination register.

Figure 9-5. Single Cycle ALU Operation



9.7 Reset and Interrupt Handling

The AVR provides several different interrupt sources. These interrupts and the separate Reset vector each have a separate program vector in the program memory space. All interrupts are assigned individual enable bits, which must be written logic one together with the global interrupt enable bit in the Status register in order to enable the interrupt. Depending on the program counter value, interrupts may be automatically disabled when Boot Lock bits BLB02 or BLB12 are programmed. This feature improves software security.

The lowest addresses in the program memory space are by default defined as the Reset and interrupt vectors. They have determined priority levels: The lower the address the higher is the priority level. RESET has the highest priority, and next is INT0 – the External Interrupt Request 0. The interrupt vectors can be moved to the start of the boot Flash section by setting the IVSEL bit in the MCU Control Register (MCUCR). The Reset vector can be moved to the start of the boot Flash section by programming the BOOTRST Fuse.

When an interrupt occurs, the global interrupt enable I-bit is cleared and all interrupts are disabled. The user software can write logic one to the I-bit to enable nested interrupts. All enabled interrupts can then interrupt the current interrupt routine. The I-bit is automatically set when a return from interrupt instruction – RETI – is executed.

There are basically two types of interrupts:

The first type is triggered by an event that sets the interrupt flag. For these interrupts, the program counter is vectored to the actual interrupt vector in order to execute the interrupt handling routine, and

hardware clears the corresponding interrupt flag. Interrupt flags can be cleared by writing a logic one to the flag bit position(s) to be cleared. If an interrupt condition occurs while the corresponding interrupt enable bit is cleared, the interrupt flag will be set and remembered until the interrupt is enabled, or the flag is cleared by software. Similarly, if one or more interrupt conditions occur while the global interrupt enable bit is cleared, the corresponding interrupt flag(s) will be set and remembered until the global interrupt enable bit is set and will then be executed by order of priority.

The second type of interrupts will trigger as long as the interrupt condition is present. These interrupts do not necessarily have interrupt flags. If the interrupt condition disappears before the interrupt is enabled, the interrupt will not be triggered. When the AVR exits from an interrupt, it will always return to the main program and execute one more instruction before any pending interrupt is served.

The Status register is not automatically stored when entering an interrupt routine, nor restored when returning from an interrupt routine. This must be handled by software.

When using the CLI instruction to disable interrupts, the interrupts will be immediately disabled. No interrupt will be executed after the CLI instruction, even if it occurs simultaneously with the CLI instruction. The following example shows how this can be used to avoid interrupts during the timed EEPROM write sequence.

Assembly Code Example(1)

```
in r16, SREG ; store SREG value
cli ; disable interrupts during timed sequence
sbi EECR, EEMPE ; start EEPROM write
sbi EECR, EEPE
out SREG, r16 ; restore SREG value (I-bit)
```

C Code Example(1)

```
char cSREG;
cSREG = SREG; /* store SREG value */
/* disable interrupts during timed sequence */
_CLI();
EECR |= (1<<EEMPE); /* start EEPROM write */
EECR |= (1<<EEPE);
SREG = cSREG; /* restore SREG value (I-bit) */
```

1. Refer to *About Code Examples*.

When using the SEI instruction to enable interrupts, the instruction following SEI will be executed before any pending interrupts, as shown in this example.

Assembly Code Example(1)

```
sei ; set Global Interrupt Enable
sleep ; enter sleep, waiting for interrupt
; note: will enter sleep before any pending interrupt(s)
```

C Code Example(1)

```
__enable_interrupt(); /* set Global Interrupt Enable */
__sleep(); /* enter sleep, waiting for interrupt */
/* note: will enter sleep before any pending interrupt(s) */
```

1. Refer to *About Code Examples*.

Related Links

Memory Programming

Boot Loader Support – Read-While-Write Self-Programming

About Code Examples

9.7.1 Interrupt Response Time

The interrupt execution response for all the enabled AVR interrupts is four clock cycles minimum. After four clock cycles, the program vector address for the actual interrupt handling routine is executed. During this four clock cycle period, the program counter is pushed onto the stack. The vector is normally a jump to the interrupt routine, and this jump takes three clock cycles. If an interrupt occurs during execution of a multi-cycle instruction, this instruction is completed before the interrupt is served. If an interrupt occurs when the MCU is in sleep mode, the interrupt execution response time is increased by four clock cycles. This increase comes in addition to the start-up time from the selected sleep mode. A return from an interrupt handling routine takes four clock cycles. During these four clock cycles, the program counter (two bytes) is popped back from the Stack, the Stack Pointer is incremented by two, and the I-bit in SREG is set.

10. AVR Memories

10.1 Overview

This section describes the different memory types in the device. The AVR architecture has two main memory spaces, the Data Memory and the Program Memory space. In addition, the device features an EEPROM Memory for data storage. All memory spaces are linear and regular.

10.2 In-System Reprogrammable Flash Program Memory

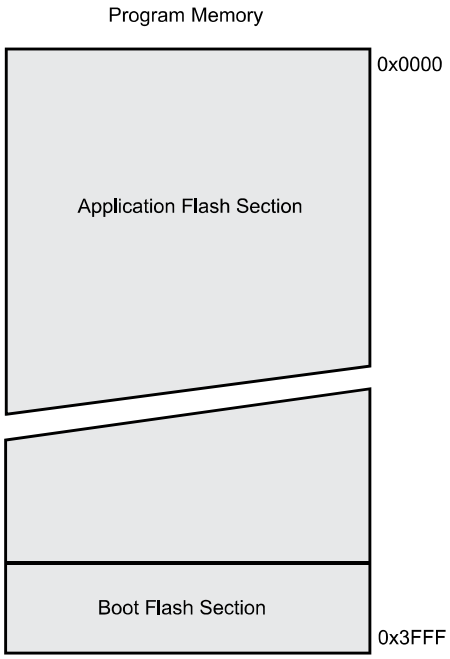
The ATmega328PB contains 32 Kbytes on-chip in-system reprogrammable Flash memory for program storage. Since all AVR instructions are 16 or 32 bits wide, the Flash is organized as 16K x 16.

The ATmega328PB Program Counter (PC) is 14 bits wide, thus addressing the 16K program memory locations. The operation of the Boot Program section and associated Boot Lock bits for software protection are described in detail in *Boot Loader Support – Read-While-Write Self-Programming*. Refer to *Memory Programming* for the description of Flash data serial downloading using the SPI pins.

Constant tables can be allocated within the entire program memory address space, using the Load Program Memory (LPM) instruction.

Timing diagrams for instruction fetch and execution are presented in *Instruction Execution Timing*.

Figure 10-1. Program Memory Map ATmega328PB



Related Links

BTLDR - Boot Loader Support – Read-While-Write Self-Programming
MEMPROG - Memory Programming
Instruction Execution Timing

10.3 SRAM Data Memory

The following figure shows how the device SRAM memory is organized.

The device is a complex microcontroller with more peripheral units than can be supported within the 64 locations reserved in the Opcode for the IN and OUT instructions. For the extended I/O space from 0x60 - 0xFF in SRAM, only the ST/STS/STD and LD/LDS/LDD instructions can be used.

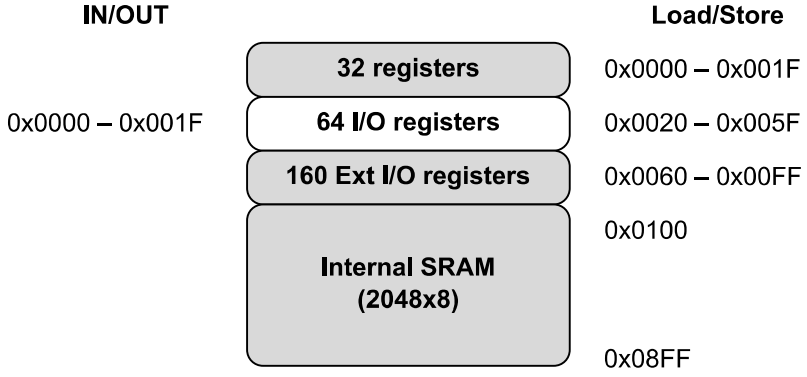
The lower 2303 data memory locations address both the register file, the I/O memory, extended I/O memory, and the internal data SRAM. The first 32 locations address the register file, the next 64 location the standard I/O memory, then 160 locations of extended I/O memory, and the next 2K locations address the internal data SRAM.

The five different addressing modes for the data memory cover:

- Direct
  - The direct addressing reaches the entire data space.
- Indirect with Displacement
  - The indirect with displacement mode reaches 63 address locations from the base address given by the Y- or Z-register.
- Indirect
  - In the register file, registers R26 to R31 feature the indirect addressing pointer registers.
- Indirect with Pre-decrement
  - The address registers X, Y, and Z are decremented.
- Indirect with Post-increment
  - The address registers X, Y, and Z are incremented.

The 32 general purpose working registers, 64 I/O registers, 160 extended I/O registers, and the 2 K bytes of internal data SRAM in the device are all accessible through all these addressing modes.

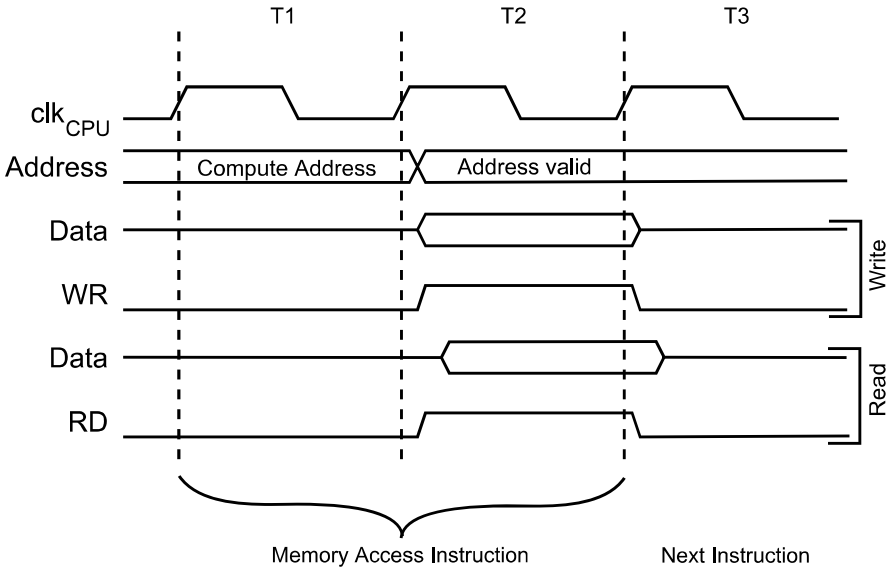
Figure 10-2. Data Memory Map with 2048 Byte Internal Data SRAM



10.3.1 Data Memory Access Times

The internal data SRAM access is performed in two clk_{CPU} cycles as described in the following Figure.

Figure 10-3. On-chip Data SRAM Access Cycles



10.4 EEPROM Data Memory

The ATmega328PB contains 1 KB of data EEPROM memory. It is organized as a separate data space, in which single bytes can be read and written. The access between the EEPROM and the CPU is described in the following, specifying the EEPROM Address registers, the EEPROM Data register, and the EEPROM Control register.

See the related links for a detailed description on EEPROM Programming in SPI or Parallel Programming mode.

Related Links

MEMPROG - Memory Programming

10.4.1 EEPROM Read/Write Access

The EEPROM access registers are accessible in the I/O space.

The write access time for the EEPROM is given in Table 10-2. A self-timing function, however, lets the user software detect when the next byte can be written. If the user code contains instructions that write the EEPROM, some precautions must be taken. In heavily filtered power supplies, V_{CC} is likely to rise or fall slowly on power-up/down. This causes the device for some period of time to run at a voltage lower than specified as a minimum for the clock frequency used. Refer to Preventing EEPROM Corruption for details on how to avoid problems in these situations.

In order to prevent unintentional EEPROM writes, a specific write procedure must be followed. Refer to the description of the EEPROM Control register for details on this.

When the EEPROM is read, the CPU is halted for four clock cycles before the next instruction is executed. When the EEPROM is written, the CPU is halted for two clock cycles before the next instruction is executed.

10.4.2 Preventing EEPROM Corruption

During periods of low $V_{CC,}$ the EEPROM data can be corrupted because the supply voltage is too low for the CPU and the EEPROM to operate properly. These issues are the same as for board level systems using EEPROM, and the same design solutions should be applied.

An EEPROM data corruption can be caused by two situations when the voltage is too low. First, a regular write sequence to the EEPROM requires a minimum voltage to operate correctly. Secondly, the CPU itself can execute instructions incorrectly, if the supply voltage is too low.

EEPROM data corruption can easily be avoided by following this design recommendation:

Keep the AVR $\overline{\text{RESET}}$ active (low) during periods of insufficient power supply voltage. This can be done by enabling the internal Brown-out Detector (BOD). If the detection level of the internal BOD does not match the needed detection level, an external low V_{CC} reset Protection circuit can be used. If a reset occurs while a write operation is in progress, the write operation will be completed provided that the power supply voltage is sufficient.

10.5 I/O Memory

The I/O space definition of the device is shown in the *Register Summary*.

All device I/Os and peripherals are placed in the I/O space. All I/O locations may be accessed by the LD/LDS/LDD and ST/STS/STD instructions, transferring data between the 32 general purpose working registers and the I/O space. I/O registers within the address range 0x00-0x1F are directly bit-accessible using the SBI and CBI instructions. In these registers, the value of single bits can be checked by using the SBIS and SBIC instructions.

When using the I/O specific commands IN and OUT, the I/O addresses 0x00-0x3F must be used. When addressing I/O registers as data space using LD and ST instructions, 0x20 must be added to these addresses. The device is a complex microcontroller with more peripheral units than can be supported within the 64 locations reserved in Opcode for the IN and OUT instructions. For the extended I/O space from 0x60..0xFF in SRAM, only the ST/STS/STD and LD/LDS/LDD instructions can be used.

For compatibility with future devices, reserved bits should be written to zero if accessed. Reserved I/O memory addresses should never be written.

Some of the status flags are cleared by writing a '1' to them; this is described in the flag descriptions. Note that, unlike most other AVRs, the CBI and SBI instructions will only operate on the specified bit, and can, therefore, be used on registers containing such status flags. The CBI and SBI instructions work with registers 0x00-0x1F only.

The I/O and peripherals control registers are explained in later sections.

Related Links

MEMPROG - Memory Programming

Register Summary

Instruction Set Summary

10.5.1 General Purpose I/O Registers

The device contains three general purpose I/O registers; General purpose I/O register 0/1/2 (GPIOR 0/1/2). These registers can be used for storing any information, and they are particularly useful for storing global variables and status flags. General purpose I/O registers within the address range 0x00 - 0x1F are directly bit-accessible using the SBI, CBI, SBIS, and SBIC instructions.

10.6 Register Description

10.6.1 Accessing 16-Bit Registers

The AVR data bus is 8-bits wide, so accessing 16-bit registers requires atomic operations. These registers must be byte-accessed using two read or write operations. 16-bit registers are connected to the 8-bit bus and a temporary register using a 16-bit bus.

For a write operation, the high byte of the 16-bit register must be written before the low byte. The high byte is then written into the temporary register. When the low byte of the 16-bit register is written, the temporary register is copied into the high byte of the 16-bit register in the same clock cycle.

For a read operation, the low byte of the 16-bit register must be read before the high byte. When the low byte register is read by the CPU, the high byte of the 16-bit register is copied into the temporary register in the same clock cycle as the low byte is read. When the high byte is read, it is then read from the temporary register.

This ensures that the low and high bytes of 16-bit registers are always accessed simultaneously when reading or writing the register.

Interrupts can corrupt the timed sequence if an interrupt is triggered and accesses the same 16-bit register during an atomic 16-bit read/write operation. To prevent this, interrupts can be disabled when writing or reading 16-bit registers.

The temporary registers can be read and written directly from user software.

Note: For more information, refer to section Accessing 16-bit Timer/Counter registers in chapter 16-bit Timer/Counter1 with PWM.

Related Links

Accessing 16-bit Timer/Counter Registers

About Code Examples

10.6.2 EEPROM Address Register Low and High Byte

Name: EEARL and EEARH
Offset: 0x41 [ID-000004d0]
Reset: 0xXX
Property: When addressing as I/O Register: address offset is 0x21

The EEARL and EEARH register pair represents the 16-bit value, EEAR. The low byte [7:0] (suffix L) is accessible at the original offset. The high byte [15:8] (suffix H) can be accessed at offset + 0x01. For more details on reading and writing 16-bit registers, refer to accessing 16-bit registers in the section above.

When addressing I/O registers as data space using LD and ST instructions, the provided offset must be used. When using the I/O specific commands IN and OUT, the offset is reduced by 0x20, resulting in an I/O address offset within 0x00 - 0x3F.

The device is a complex microcontroller with more peripheral units than can be supported within the 64 locations reserved in Opcode for the IN and OUT instructions. For the extended I/O space from 0x60 in SRAM, only the ST/STS/STD and LD/LDS/LDD instructions can be used.

Bit	15	14	13	12	11	10	9	8
							EEAR[9:8]	
Access							R/W	R/W
Reset							x	x

Bit	7	6	5	4	3	2	1	0
	EEAR[7:0]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	x	x	x	x	x	x	x	x

Bits 9:0 – EEAR[9:0] EEPROM Address
The EEPROM Address Registers, EEARH and EEARL, specify the EEPROM address in the 1 KB EEPROM space. The EEPROM data bytes are addressed linearly between 0 and 1023. The initial value of EEAR is undefined. A proper value must be written before the EEPROM may be accessed.

10.6.3 EEPROM Data Register

Name: EEDR
Offset: 0x40 [ID-000004d0]
Reset: 0x00
Property: When addressing as I/O Register: address offset is 0x20

When addressing I/O registers as data space using LD and ST instructions, the provided offset must be used. When using the I/O specific commands IN and OUT, the offset is reduced by 0x20, resulting in an I/O address offset within 0x00 - 0x3F.

The device is a complex microcontroller with more peripheral units than can be supported within the 64 locations reserved in Opcode for the IN and OUT instructions. For the extended I/O space from 0x60 in SRAM, only the ST/STS/STD and LD/LDS/LDD instructions can be used.

Bit	7	6	5	4	3	2	1	0
	EEDR[7:0]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bits 7:0 – EEDR[7:0] EEPROM Data
For the EEPROM write operation, the EEDR register contains the data to be written to the EEPROM in the address given by the EEAR register. For the EEPROM read operation, the EEDR contains the data read out from the EEPROM at the address given by EEAR.

10.6.4 EEPROM Control Register

Name: EECR
Offset: 0x3F [ID-000004d0]
Reset: 0x00
Property: When addressing as I/O register: address offset is 0x1F

Bit	7	6	5	4	3	2	1	0
			EEPM[1:0]		EERIE	EEMPE	EEPE	EERE
Access			R/W	R/W	R/W	R/W	R/W	R/W
Reset			x	x	0	0	x	0

Bits 5:4 – EEPM[1:0] EEPROM Programming Mode Bits
The EEPROM Programming mode bit setting defines which programming action will be triggered when writing EEPE. It is possible to program data in one atomic operation (erase the old value and program the new value) or to split the erase and write operations into two different operations. The programming times for the different modes are shown in the table below. While EEPE is set, any write to EEPMn will be ignored. During reset, the EEPMn bits will be reset to 0b00 unless the EEPROM is busy programming.

Table 10-1. EEPROM Mode Bits

EEPM[1:0]	Typ. Programming Time	Operation
00	3.4ms	Erase and Write in one operation (Atomic Operation)
01	1.8ms	Erase Only
10	1.8ms	Write Only
11	-	Reserved for future use

Bit 3 – EERIE EEPROM Ready Interrupt Enable
Writing EERIE to '1' enables the EEPROM ready interrupt if the I bit in SREG is set. Writing EERIE to zero disables the interrupt. The EEPROM ready interrupt generates a constant interrupt when EEPE is cleared. The interrupt will not be generated during EEPROM write or SPM.

Bit 2 – EEMPE EEPROM Master Write Enable
The EEMPE bit determines whether writing EEPE to '1' causes the EEPROM to be written.
When EEMPE is '1', setting EEPE within four clock cycles will write data to the EEPROM at the selected address.

If EEMPE is zero, setting EEPE will have no effect. When EEMPE has been written to '1' by software, hardware clears the bit to zero after four clock cycles. See the description of the EEPE bit for an EEPROM write procedure.

Bit 1 – EEPE EEPROM Write Enable
The EEPROM write enable signal EEPE is the write strobe to the EEPROM. When address and data are correctly set up, the EEPE bit must be written to '1' to write the value into the EEPROM. The EEMPE bit must be written to '1' before EEPE is written to '1', otherwise, no EEPROM write takes place. The following procedure should be followed when writing the EEPROM (the order of steps 3 and 4 is not essential):

1. Wait until EEPE becomes zero.
2. Wait until SPMEN in SPMCSR becomes zero.
3. Write new EEPROM address to EEAR (optional).
4. Write new EEPROM data to EEDR (optional).
5. Write a '1' to the EEMPE bit while writing a zero to EEPE in EECR.
6. Within four clock cycles after setting EEMPE, write a '1' to EEPE.

The EEPROM cannot be programmed during a CPU write to the Flash memory. The software must check that the Flash programming is completed before initiating a new EEPROM write. Step 2 is only relevant if the software contains a Boot Loader allowing the CPU to program the Flash. If the Flash is never being updated by the CPU, step 2 can be omitted.

⚠ CAUTION

An interrupt between step 5 and step 6 will make the write cycle fail, since the EEPROM Master Write Enable will time-out. If an interrupt routine accessing the EEPROM is interrupting another EEPROM access, the EEAR or EEDR register will be modified, causing the interrupted EEPROM access to fail. It is recommended to have the global interrupt flag cleared during all the steps to avoid these problems.

When the write access time has elapsed, the EEPE bit is cleared by hardware. The user software can poll this bit and wait for a zero before writing the next byte. When EEPE has been set, the CPU is halted for two cycles before the next instruction is executed.

Bit 0 – EERE EEPROM Read Enable
The EEPROM read enable signal EERE is the read strobe to the EEPROM. When the correct address is set up in the EEAR register, the EERE bit must be written to a '1' to trigger the EEPROM read. The EEPROM read access takes one instruction, and the requested data is available immediately. When the EEPROM is read, the CPU is halted for four cycles before the next instruction is executed.

The user should poll the EEPE bit before starting the read operation. If a write operation is in progress, it is neither possible to read the EEPROM, nor to change the EEAR register.

The calibrated oscillator is used to time the EEPROM accesses. See the following table for typical programming times for EEPROM access from the CPU.

Table 10-2. EEPROM Programming Time

Symbol	Number of Calibrated RC Oscillator Cycles	Typ. Programming Time
EEPROM write (from CPU)	26,368	3.3ms

The following code examples show one assembly and one C function for writing to the EEPROM. The examples assume that interrupts are controlled (e.g. by disabling interrupts globally) so that no interrupts will occur during execution of these functions. The examples also assume that no Flash Boot Loader is present in the software. If such code is present, the EEPROM write function must also wait for any ongoing SPM command to finish.

Assembly Code Example(1)

```
EEPROM_write:
      ; Wait for completion of previous write
      sbic     EECR,EEPE
      rjmp     EEPROM_write
```

```
      ; Set up address (r18:r17) in address register
      out     EEARH, r18
      out     EEARL, r17
      ; Write data (r16) to Data Register
      out     EEDR,r16
      ; Write logical one to EEMPE
      sbi     EECR,EEMPE
      ; Start eeprom write by setting EEPE
      sbi     EECR,EEPE
      ret
```

C Code Example(1)

```
void EEPROM_write(unsigned int uiAddress, unsigned char ucData)
{
    /* Wait for completion of previous write */
    while(EECR & (1<<EEPE))
    ;
    /* Set up address and Data Registers */
    EEAR = uiAddress;
    EEDR = ucData;
    /* Write logical one to EEMPE */
    EECR |= (1<<EEMPE);
    /* Start eeprom write by setting EEPE */
    EECR |= (1<<EEPE);
}
```

Note: (1) Refer to *About Code Examples*

The following code examples show assembly and C functions for reading the EEPROM. The examples assume that interrupts are controlled so that no interrupts will occur during execution of these functions.

Assembly Code Example(1)

```
EEPROM_read:
   ; Wait for completion of previous write
   sbic     EECR,EEPE
   rjmp     EEPROM_read
   ; Set up address (r18:r17) in address register
   out     EEARH, r18
   out     EEARL, r17
   ; Start eeprom read by writing EERE
   sbi     EECR,EERE
   ; Read data from Data Register
   in      r16,EEDR
   ret
```

C Code Example(1)

```
unsigned char EEPROM_read(unsigned int uiAddress)
{
    /* Wait for completion of previous write */
    while(EECR & (1<<EEPE))
    ;
    /* Set up address register */
    EEAR = uiAddress;
    /* Start eeprom read by writing EERE */
    EECR |= (1<<EERE);
    /* Return data from Data Register */
    return EEDR;
}
```

1. Refer to *About Code Examples*.

10.6.5 GPIOR2 – General Purpose I/O Register 2

Name: GPIOR2
Offset: 0x4B [ID-000004d0]
Reset: 0x00
Property: When addressing as I/O Register: address offset is 0x2B

When addressing I/O registers as data space using LD and ST instructions, the provided offset must be used. When using the I/O specific commands IN and OUT, the offset is reduced by 0x20, resulting in an I/O address offset within 0x00 - 0x3F.

The device is a complex microcontroller with more peripheral units than can be supported within the 64 locations reserved in Opcode for the IN and OUT instructions. For the extended I/O space from 0x60 in SRAM, only the ST/STS/STD and LD/LDS/LDD instructions can be used.

Bit	7	6	5	4	3	2	1	0
	GPIOR2[7:0]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bits 7:0 – GPIOR2[7:0] General Purpose I/O

10.6.6 GPIOR1 – General Purpose I/O Register 1

Name: GPIOR1
Offset: 0x4A [ID-000004d0]
Reset: 0x00
Property: When addressing as I/O Register: address offset is 0x2A

When addressing I/O registers as data space using LD and ST instructions, the provided offset must be used. When using the I/O specific commands IN and OUT, the offset is reduced by 0x20, resulting in an I/O address offset within 0x00 - 0x3F.

The device is a complex microcontroller with more peripheral units than can be supported within the 64 locations reserved in Opcode for the IN and OUT instructions. For the extended I/O space from 0x60 in SRAM, only the ST/STS/STD and LD/LDS/LDD instructions can be used.

Bit	7	6	5	4	3	2	1	0
	GPIOR1[7:0]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bits 7:0 – GPIOR1[7:0] General Purpose I/O

10.6.7 GPIOR0 – General Purpose I/O Register 0

Name: GPIOR0
Offset: 0x3E [ID-000004d0]
Reset: 0x00
Property: When addressing as I/O Register: address offset is 0x1E

When addressing I/O registers as data space using LD and ST instructions, the provided offset must be used. When using the I/O specific commands IN and OUT, the offset is reduced by 0x20, resulting in an I/O address offset within 0x00 - 0x3F.

The device is a complex microcontroller with more peripheral units than can be supported within the 64 locations reserved in Opcode for the IN and OUT instructions. For the extended I/O space from 0x60 in SRAM, only the ST/STS/STD and LD/LDS/LDD instructions can be used.

Bit	7	6	5	4	3	2	1	0
	GPIOR0[7:0]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bits 7:0 – GPIOR0[7:0] General Purpose I/O

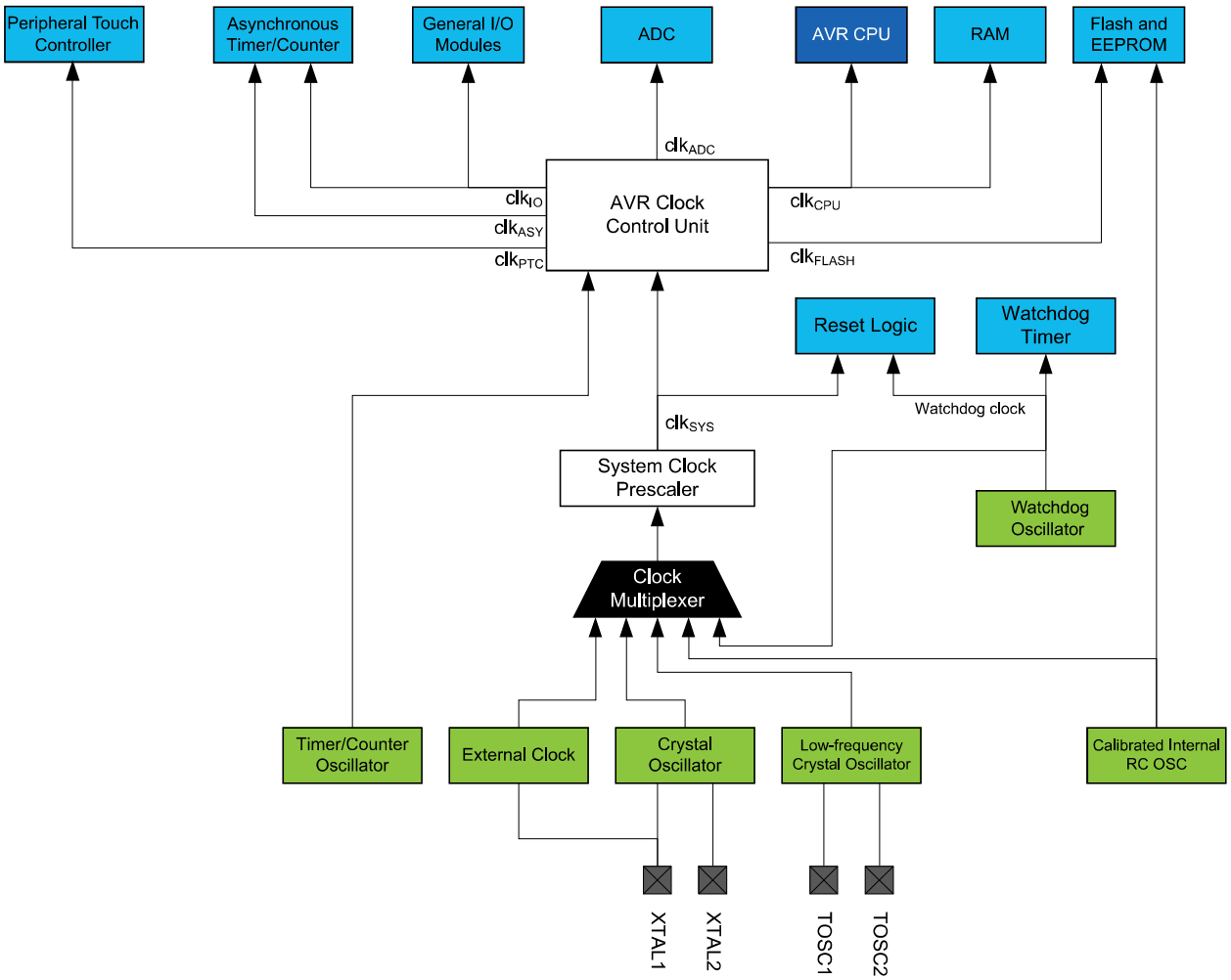
11. System Clock and Clock Options

11.1 Clock Systems and Their Distribution

The following figure illustrates the principal clock systems in the device and their distribution. All the clocks need not be active at a given time. In order to reduce power consumption, the clocks to modules not being used can be halted by using different sleep modes. The clock systems are described in the following sections.

The system clock frequency refers to the frequency generated from the system clock prescaler. All clock outputs from the AVR clock control unit runs in the same frequency.

Figure 11-1. Clock Distribution



Related Links

Power Management and Sleep Modes

11.1.1 CPU Clock – clk_{CPU}

The CPU clock is routed to parts of the system concerned with operation of the AVR core. Examples of such modules are the general purpose register file, the Status register, and the data memory holding the

stack pointer. Halting the CPU clock inhibits the core from performing general operations and calculations.

11.1.2 I/O Clock – $clk_{I/O}$

The I/O clock is used by the majority of the I/O modules, like Timer/Counters, SPI, and USART. The I/O clock is also used by the External Interrupt module, but the start condition detection in the USI module is carried out asynchronously when $clk_{I/O}$ is halted, TWI address recognition in all sleep modes.

Note: If a level triggered interrupt is used for wake-up from power-down, the required level must be held long enough for the MCU to complete the wake-up to trigger the level interrupt. If the level disappears before the end of the start-up time, the MCU will still wake up, but no interrupt will be generated. The start-up time is defined by the SUT and CKSEL fuses.

11.1.3 PTC Clock - clk_{PTC}

The PTC is provided with a dedicated clock domain. This allows halting the CPU and I/O clocks in order to reduce noise and power due to digital circuitry.

11.1.4 Flash Clock – clk_{FLASH}

The Flash clock controls operation of the Flash interface. The Flash clock is usually active simultaneously with the CPU clock.

11.1.5 Asynchronous Timer Clock – clk_{ASY}

The asynchronous timer clock allows asynchronous Timer/Counters to be clocked directly from an external clock or an external 32 kHz clock crystal. The dedicated clock domain allows using this Timer/Counter as a real-time counter even when the device is in sleep mode.

11.1.6 ADC Clock – clk_{ADC}

The ADC is provided with a dedicated clock domain. This allows halting the CPU and I/O clocks in order to reduce noise generated by digital circuitry. This gives more accurate ADC conversion results.

11.2 Clock Sources

The device has the following clock source options, selectable by Flash fuse bits as shown below. The clock from the selected source is input to the AVR clock generator and routed to the appropriate modules.

Table 11-1. Device Clocking Options Select

Device Clocking Option	CKSEL[3:0]
Low-Power Crystal Oscillator	1111 - 1000
Low Frequency Crystal Oscillator	0101 - 0100
Internal 128 kHz RC Oscillator	0011
Calibrated Internal RC Oscillator	0010
External Clock	0000
Reserved	0001

Note: For all fuses, '1' means unprogrammed while '0' means programmed.

11.2.1 Default Clock Source

The device is shipped with internal RC oscillator at 8.0 MHz and with the fuse CKDIV8 programmed, resulting in 1.0 MHz system clock. The start-up time is set to maximum, and the time-out period is enabled: CKSEL=0010, SUT=10, CKDIV8=0. This default setting ensures that all users can make their desired clock source setting using any available programming interface.

11.2.2 Clock Start-Up Sequence

Any clock source needs a sufficient V_{CC} to start oscillating and a minimum number of oscillating cycles before it can be considered stable.

To ensure sufficient V_{CC}, the device issues an internal Reset with a time-out delay (t_{TOUT}) after the device Reset is released by all other Reset sources. See the Related Links for a description of the start conditions for the internal Reset. The delay (t_{TOUT}) is timed from the Watchdog oscillator and the number of cycles in the delay is set by the SUTx and CKSELx fuse bits. The selectable delays are shown in the table below. The frequency of the Watchdog oscillator is voltage dependent.

Table 11-2. Number of Watchdog Oscillator Cycles

Typ. Time-out (V_{CC} = 5.0V)	Typ. Time-out (V_{CC} = 3.0V)
0ms	0ms
4 ms	4.3 ms
65 ms	69 ms

Main purpose of the delay is to keep the device in Reset until it is supplied with minimum V_{CC}. The delay will not monitor the actual voltage, so it is required to select a delay longer than the V_{CC} rise time. If this is not possible, an internal or external Brown-out Detection (BOD) circuit should be used. A BOD circuit will ensure sufficient V_{CC} before it releases the reset, and the time out delay can be disabled. Disabling the time-out delay without utilizing a BOD circuit is not recommended.

The oscillator is required to oscillate for a minimum number of cycles before the clock is considered stable. An internal ripple counter monitors the oscillator output clock, and keeps the internal Reset active for a given number of clock cycles. The Reset is then released and the device will start to execute. The recommended oscillator start-up time is dependent on the clock type, and varies from six cycles for an externally applied clock to 32K cycles for a low frequency crystal.

The start-up sequence for the clock includes both the time-out delay and the start-up time when the device starts up from Reset. When starting up from Power-save or Power-down mode, V_{CC} is assumed to be at a sufficient level and only the start-up time is included.

Related Links

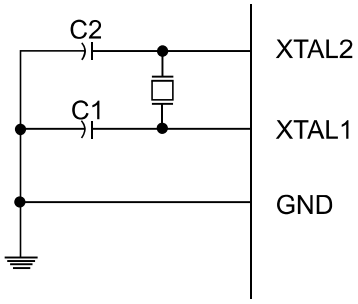
System Control and Reset

11.2.3 Clock Source Connections

Pins XTAL1 and XTAL2 are input and output, respectively, of an inverting amplifier that can be configured for use as an on-chip oscillator, as shown in the figure below. Either a quartz crystal or a ceramic resonator may be used.

C1 and C2 should always be equal for both crystals and resonators. The optimal value of the capacitors depends on the crystal or resonator in use, the amount of stray capacitance, and the electromagnetic noise of the environment. Some initial guidelines for choosing capacitors for use with crystals are given in the next table. For ceramic resonators, the capacitor values given by the manufacturer should be used.

Figure 11-2. Crystal Oscillator Connections



11.3 Low-Power Crystal Oscillator

This Crystal Oscillator is a low-power oscillator, with reduced voltage swing on the XTAL2 output. It gives the lowest power consumption, but is not capable of driving other clock inputs, and may be more susceptible to noise in noisy environments.

The crystal should be connected as described in Clock Source Connections. When selecting crystals, load capacitance must be taken into consideration. The capacitance (C_e+C_i) needed at each TOSC pin can be calculated by using:

$$C_e + C_i = 2C_L - C_s$$

where:

- C_e - is optional external capacitors. (= C_1, C_2 as shown in the schematics.)
- C_i - is the pin capacitance in the following table.
- C_L - is the load capacitance specified by the crystal vendor.
- C_S - is the total stray capacitance for one XTAL pin.

Table 11-3. Internal Capacitance of Low-Power Oscillator

32 kHz Osc. Type	Internal Pad Capacitance (XTAL1)	Internal Pad Capacitance (XTAL2)
C_i of system oscillator (XTAL pins)	18 pF	8 pF

The Low-power Oscillator can operate in three different modes, each optimized for a specific frequency range. The operating mode is selected by the fuses CKSEL[3:1], as shown in the following table:

Table 11-4. Low-Power Crystal Oscillator Operating Modes(1)

Frequency Range [MHz]	CKSEL[3:1](2)	Absolute Limits for Total Capacitance of C1 and C2 [pF](4)
0.4 - 0.9	100(3)	–
0.9 - 3.0	101	12 - 22
3.0 - 8.0	110	12 - 22
8.0 - 16.0	111	12 - 22

Note:

1. These are the recommended CKSEL settings for the different frequency ranges.

2. This option should not be used with crystals, only with ceramic resonators.
3. If the crystal frequency exceeds the specification of the device (depends on V_{CC}), the CKDIV8 Fuse can be programmed in order to divide the internal frequency by 8. It must be ensured that the resulting divided clock meets the frequency specification of the device.
4. When selecting the external capacitor value, the stray capacitance from the PCB and device should be deducted. The total load (Ce+Ci+Cs) on XTAL pins must not exceed 22 pF.

The CKSEL0 Fuse together with the SUT[1:0] Fuses select the start-up times, as shown in the following table:

Table 11-5. Start-Up Times for the Low-Power Crystal Oscillator Clock Selection

Oscillator Source/Power Conditions	Start-Up Time from Power-Down and Power-Save	Additional Delay from Reset (V_{CC} = 5.0V)	CKSEL0	SUT[1:0]
Ceramic resonator, fast rising power	258 CK	19CK + 4 ms(1)	0	00
Ceramic resonator, slowly rising power	258 CK	19CK + 65 ms(1)	0	01
Ceramic resonator, BOD enabled	1K CK	19CK(2)	0	10
Ceramic resonator, fast rising power	1K CK	19CK + 4 ms(2)	0	11
Ceramic resonator, slowly rising power	1K CK	19CK + 65 ms(2)	1	00
Crystal Oscillator, BOD enabled	16K CK	19CK	1	01
Crystal Oscillator, fast rising power	16K CK	19CK + 4 ms	1	10
Crystal Oscillator, slowly rising power	16K CK	19CK + 65 ms	1	11

Note:

1. These options should only be used when not operating close to the maximum frequency of the device, and only if frequency stability at start-up is not important for the application. These options are not suitable for crystals.
2. These options are intended for use with ceramic resonators and will ensure frequency stability at start-up. They can also be used with crystals when not operating close to the maximum frequency of the device and if frequency stability at start-up is not important for the application.

11.4 Low Frequency Crystal Oscillator

The Low Frequency Crystal Oscillator is optimized for use with a 32.768 kHz watch crystal. When selecting crystals, load capacitance and crystal's Equivalent Series Resistance (ESR) must be taken into consideration. Both values are specified by the crystal vendor. The oscillator is optimized for very low power consumption, and thus when selecting crystals, consider the Maximum ESR Recommendations: