Chipsmall Limited consists of a professional team with an average of over 10 year of expertise in the distribution of electronic components. Based in Hongkong, we have already established firm and mutual-benefit business relationships with customers from,Europe,America and south Asia,supplying obsolete and hard-to-find components to meet their specific needs.

With the principle of “Quality Parts,Customers Priority,Honest Operation,and Considerate Service”,our business mainly focus on the distribution of electronic components. Line cards we deal with include Microchip,ALPS,ROHM,Xilinx,Pulse,ON,Everlight and Freescale. Main products comprise IC,Modules,Potentiometer,IC Socket,Relay,Connector.Our parts cover such applications as commercial,industrial, and automotives areas.

We are looking forward to setting up business relationship with you and hope to provide you with the best service and solution. Let us make a better world for our industry!

Contact us

Tel: +86-755-8981 8866 Fax: +86-755-8427 6832
Email & Skype: info@chipsmall.com Web: www.chipsmall.com
Address: A1208, Overseas Decoration Building, #122 Zhenhua RD., Futian, Shenzhen, China

ATtiny20

8-bit AVR Microcontroller with 2K Bytes In-System Programmable Flash

DATASHEET

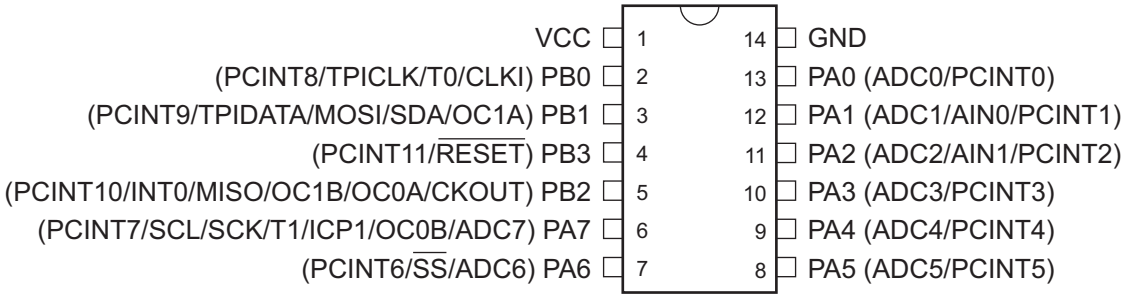
Features

- High performance, low power 8-bit AVR® microcontroller
- Advanced RISC architecture
  - 112 powerful instructions – most single clock cycle execution
  - 16 x 8 general purpose working registers
  - Fully static operation
  - Up to 12 MIPS throughput at 12MHz
- Non-volatile program and data memories
  - 2K bytes of in-system programmable flash program memory
  - 128 bytes internal SRAM
  - Flash write/erase cycles: 10,000
  - Data retention: 20 years at 85°C / 100 years at 25°C
- Peripheral features
  - One 8-bit timer/counter with two PWM channels
  - One 16-bit timer/counter with two PWM channels
  - 10-bit analog to digital converter
    - 8 single-ended channels
  - Programmable watchdog timer with separate on-chip oscillator
  - On-chip analog comparator
  - Master/slave SPI serial interface
  - Slave TWI serial interface
- Special microcontroller features
  - In-system programmable
  - External and internal interrupt sources
  - Low power idle, ADC noise reduction, stand-by and power-down modes
  - Enhanced power-on reset circuit
  - Internal calibrated oscillator
- I/O and packages
  - 14-pin SOIC/TSSOP: 12 programmable I/O lines
  - 12-ball WLCSP: 10 programmable I/O lines
  - 15-ball UFBGA: 12 programmable I/O lines
  - 20-pad VQFN: 12 programmable I/O lines
- Operating voltage:
  - 1.8 – 5.5V
- Programming voltage:
  - 5V
- Speed grade
  - 0 – 4MHz @ 1.8 – 5.5V
  - 0 – 8MHz @ 2.7 – 5.5V
  - 0 – 12MHz @ 4.5 – 5.5V
- Industrial temperature range
- Low power consumption
  - Active mode:
    - 200 µA at 1MHz and 1.8V
  - Idle mode:
    - 25µA at 1MHz and 1.8V
  - Power-down mode:
    - < 0.1µA at 1.8V

1. Pin Configurations

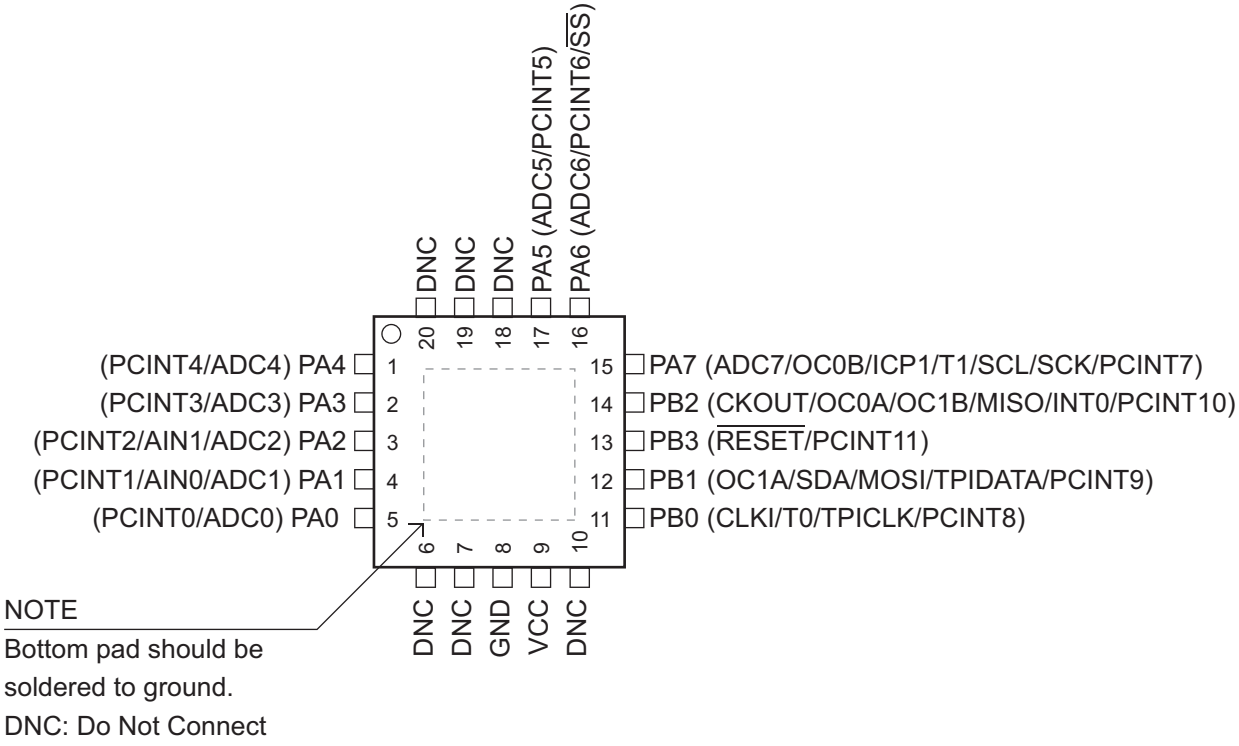
1.1 SOIC & TSSOP

Figure 1-1. SOIC/TSSOP



1.2 VQFN

Figure 1-2. VQFN



1.3 UFBGA

Figure 1-3. UFBGA

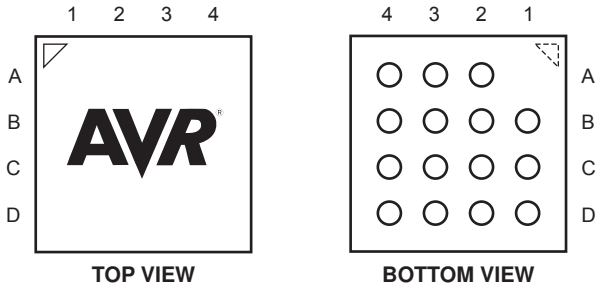


Table 1-1. UFBGA Pin Configuration

	1	2	3	4
A		PA5	PA6	PB2
B	PA4	PA7	PB1	PB3
C	PA3	PA2	PA1	PB0
D	PA0	GND	GND	VCC

1.4 Wafer Level Chip Scale Package

Figure 1-4. WLCSP

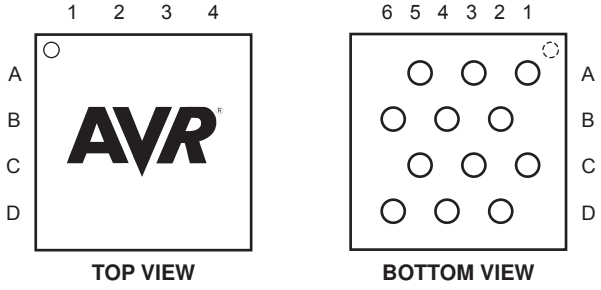


Table 1-2. WLCSP Ball Configuration

	1	2	3	4	5	6
A	PA4		PA1		PA2	
B		PA6		GND		VDD
C	PA5		PA7		PB1	
D		PB2		PB3		PB0

1.5 Pin Description

1.5.1 VCC

Supply voltage.

1.5.2 GND

Ground.

1.5.3 $\overline{\text{RESET}}$

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 and provided the reset pin has not been disabled. The minimum pulse length is given in Table 20-4 on page 170. Shorter pulses are not guaranteed to generate a reset.

The reset pin can also be used as a (weak) I/O pin.

1.5.4 Port A (PA7:PA0)

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

Port A has alternate functions as analog inputs for the ADC, analog comparator and pin change interrupt as described in “Alternate Port Functions” on page 47.

1.5.5 Port B (PB3:PB0)

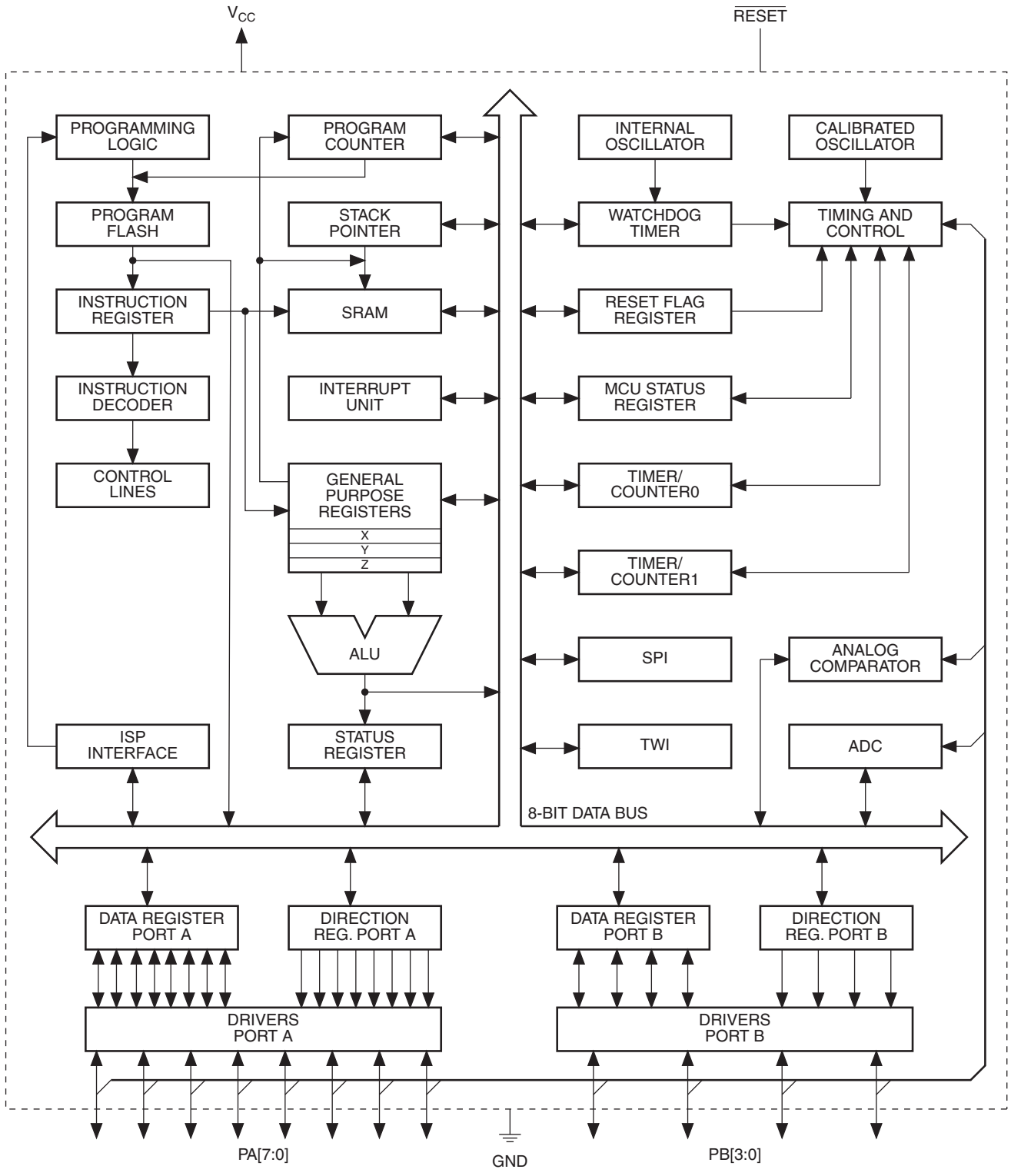
Port B is a 4-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port B output buffers have symmetrical drive characteristics with both high sink and source capability except PB3 which has the $\overline{\text{RESET}}$ capability. To use pin PB3 as an I/O pin, instead of RESET pin, program (‘0’) RSTDISBL fuse. 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 when a reset condition becomes active, even if the clock is not running.

The port also serves the functions of various special features of the ATtiny20, as listed on page 37.

2. Overview

ATtiny20 is a low-power CMOS 8-bit microcontroller based on the compact AVR enhanced RISC architecture. By executing powerful instructions in a single clock cycle, the ATtiny20 achieves throughputs approaching 1 MIPS per MHz allowing the system designer to optimize power consumption versus processing speed.

Figure 2-1. Block Diagram



The AVR core combines a rich instruction set with 16 general purpose working registers and system registers. All registers are directly connected to the Arithmetic Logic Unit (ALU), allowing two independent registers to be accessed in one single instruction executed in one clock cycle. The resulting architecture is compact and code efficient while achieving throughputs up to ten times faster than conventional CISC microcontrollers.

ATtiny20 provides the following features:

- 2K bytes of in-system programmable Flash
- 128 bytes of SRAM
- Twelve general purpose I/O lines
- 16 general purpose working registers
- An 8-bit Timer/Counter with two PWM channels
- A 16-bit Timer/Counter with two PWM channels
- Internal and external interrupts
- An eight-channel, 10-bit ADC
- A programmable Watchdog Timer with internal oscillator
- A slave two-wire interface
- A master/slave serial peripheral interface
- An internal calibrated oscillator
- Four software selectable power saving modes

The device includes the following modes for saving power:

- Idle mode: stops the CPU while allowing the timer/counter, ADC, analog comparator, SPI, TWI, and interrupt system to continue functioning
- ADC Noise Reduction mode: minimizes switching noise during ADC conversions by stopping the CPU and all I/O modules except the ADC
- Power-down mode: registers keep their contents and all chip functions are disabled until the next interrupt or hardware reset
- Standby mode: the oscillator is running while the rest of the device is sleeping, allowing very fast start-up combined with low power consumption.

The device is manufactured using Atmel’s high density non-volatile memory technology. The on-chip, in-system programmable Flash allows program memory to be re-programmed in-system by a conventional, non-volatile memory programmer.

The ATtiny20 AVR is supported by a suite of program and system development tools, including macro assemblers and evaluation kits.

3. General Information

3.1 Resources

A comprehensive set of drivers, application notes, data sheets and descriptions on development tools are available for download at http://www.atmel.com/avr.

3.2 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. Please confirm with the C compiler documentation for more details.

3.3 Capacitive Touch Sensing

Atmel QTouch Library provides a simple to use solution for touch sensitive interfaces on Atmel AVR microcontrollers. The QTouch Library includes support for QTouch® and QMatrix® acquisition methods.

Touch sensing is easily added to any application by linking the QTouch Library and using the Application Programming Interface (API) of the library to define the touch channels and sensors. The application then calls the API to retrieve channel information and determine the state of the touch sensor.

The QTouch Library is free and can be downloaded from the Atmel website. For more information and details of implementation, refer to the QTouch Library User Guide – also available from the Atmel website.

3.4 Data Retention

Reliability Qualification results show that the projected data retention failure rate is much less than 1 PPM over 20 years at 85°C or 100 years at 25°C.

3.5 Disclaimer

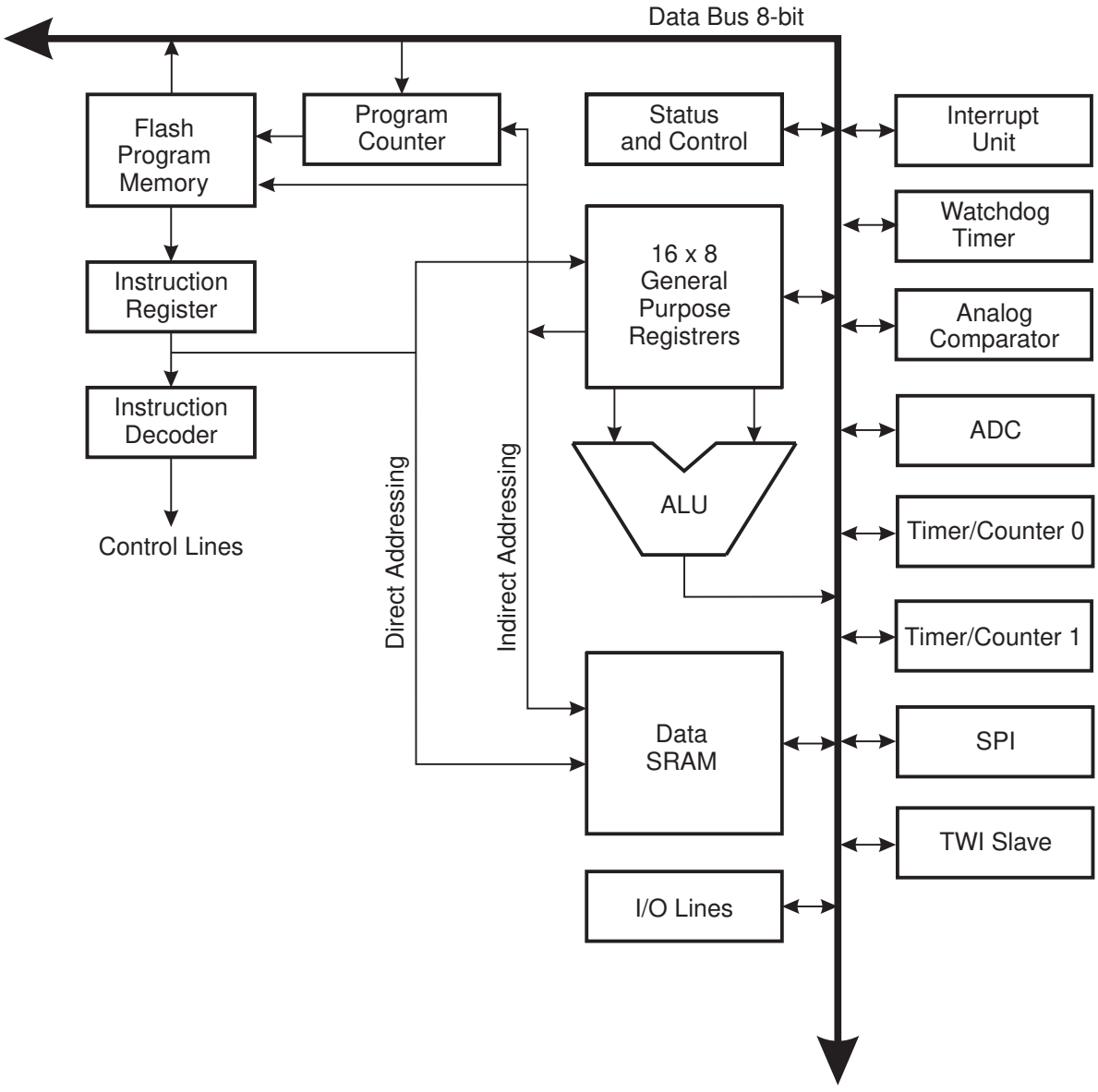
Typical values contained in this datasheet are based on simulations and characterization of other AVR microcontrollers manufactured on the same process technology.

4. CPU Core

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.

4.1 Architectural Overview

Figure 4-1. 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 16 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 16 registers can be used as three 16-bit indirect address register pointers for data space addressing – enabling efficient address calculations. 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, capable of directly addressing the whole address space. Most AVR instructions have a single 16-bit word format but 32-bit wide instructions also exist. The actual instruction set varies, as some devices only implement a part of the instruction set.

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 SRAM size and the usage of the SRAM. All user programs must initialize the SP in the Reset routine (before subroutines or interrupts are executed). The Stack Pointer (SP) is read/write accessible in the I/O space. The data SRAM can easily be accessed through the four 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 as the data space locations, 0x0000 - 0x003F.

4.2 ALU – Arithmetic Logic Unit

The high-performance AVR ALU operates in direct connection with all the 16 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 document “AVR Instruction Set” and section “Instruction Set Summary” on page 205 for a detailed description.

4.3 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. Note that the Status Register is updated after all ALU operations, as specified in document “AVR Instruction Set” and section “Instruction Set Summary” on page 205. 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.

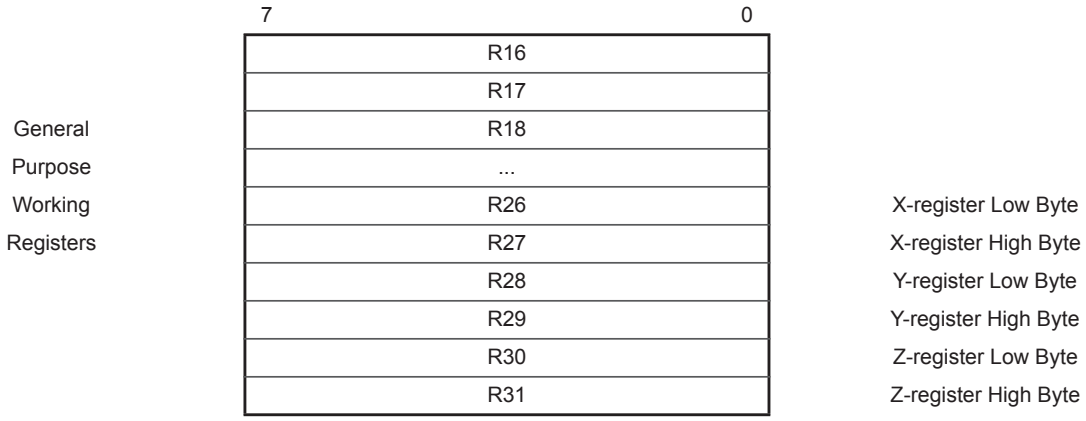
4.4 General Purpose Register File

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
- One 16-bit output operand and one 16-bit result input

Figure 4-2 below shows the structure of the 16 general purpose working registers in the CPU.

Figure 4-2. **AVR** CPU General Purpose Working Registers



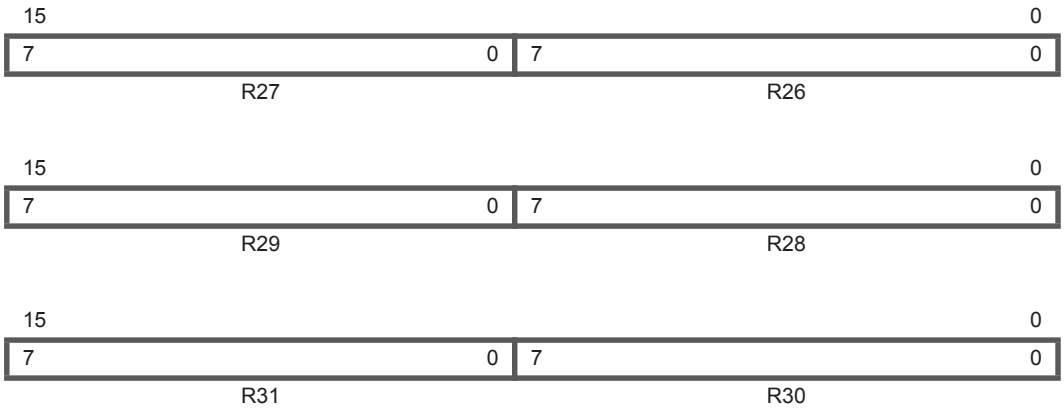
Note: A typical implementation of the AVR register file includes 32 general purpose registers but ATtiny20 implements only 16 registers. For reasons of compatibility the registers are numbered R16:R31 and not R0:R15.

Most of the instructions operating on the Register File have direct access to all registers, and most of them are single cycle instructions.

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

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 Figure 4-3.

Figure 4-3. The X-, Y-, and Z-registers



In different addressing modes these address registers function as automatic increment and automatic decrement (see document “AVR Instruction Set” and section “Instruction Set Summary” on page 205 for details).

4.5 Stack Pointer

The stack is mainly used for storing temporary data, local variables and return addresses after interrupts and subroutine calls. The Stack Pointer registers (SPH and SPL) always point to the top of the stack. Note that the stack grows from higher memory locations to lower memory locations. This means that the PUSH instructions decreases and the POP instruction increases the stack pointer value.

The stack pointer points to the area of data memory where subroutine and interrupt stacks are located. This stack space must be defined by the program before any subroutine calls are executed or interrupts are enabled.

The pointer is decremented by one when data is put on the stack with the PUSH instruction, and incremented by one when data is fetched with the POP instruction. It is decremented by two when the return address is put on the stack by a subroutine call or a jump to an interrupt service routine, and incremented by two when data is fetched by a return from subroutine (the RET instruction) or a return from interrupt service routine (the RETI instruction).

The AVR stack pointer is typically implemented as two 8-bit registers in the I/O register file. The width of the stack pointer and the number of bits implemented is device dependent. In some AVR devices all data memory can be addressed using SPL, only. In this case, the SPH register is not implemented.

The stack pointer must be set to point above the I/O register areas, the minimum value being the lowest address of SRAM. See Figure 5-1 on page 15.

4.6 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.

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

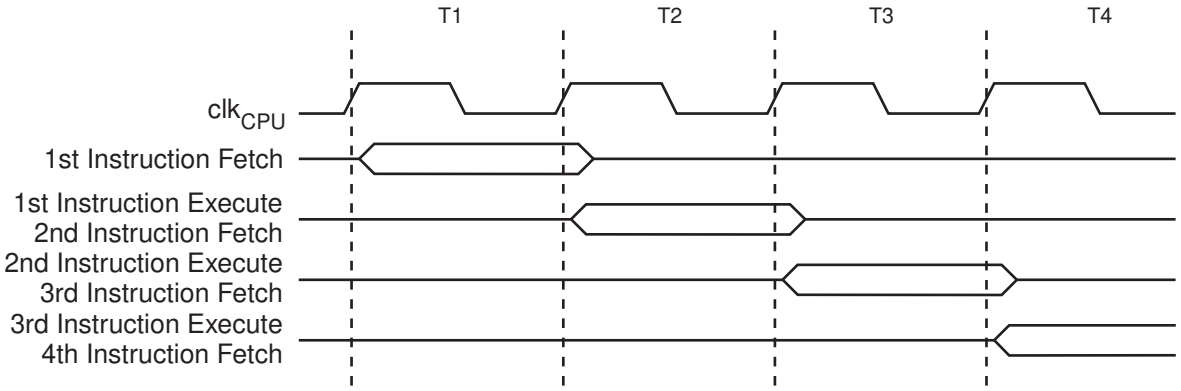
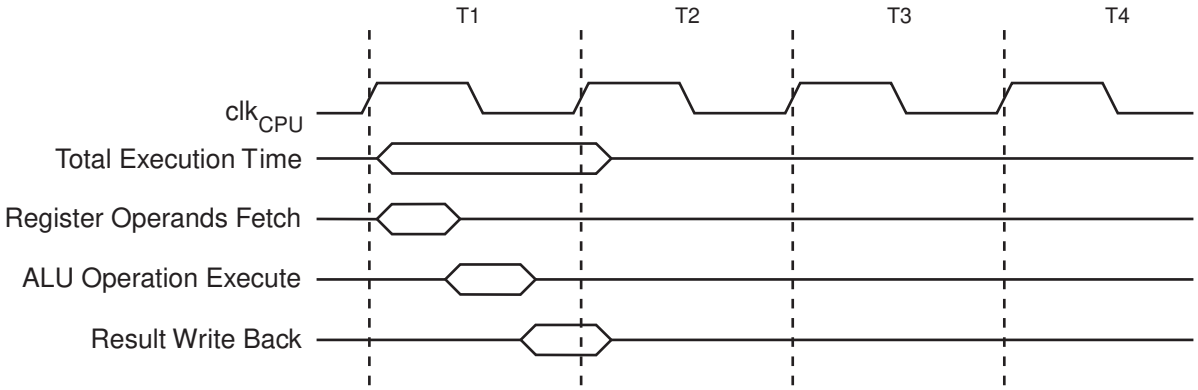


Figure 4-4 shows the parallel instruction fetches and instruction executions enabled by the Harvard architecture and the fast access Register File concept. This is the basic pipelining concept to obtain up to 1 MIPS per MHz with the corresponding unique results for functions per cost, functions per clocks, and functions per power-unit.

Figure 4-5 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 4-5. Single Cycle ALU Operation



4.7 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.

The lowest addresses in the program memory space are by default defined as the Reset and Interrupt Vectors. The complete list of vectors is shown in “Interrupts” on page 36. The list also determines the priority levels of the different interrupts. The lower the address the higher is the priority level. RESET has the highest priority, and next is INT0 – the External Interrupt Request 0.

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 also 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.

Note that 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.

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

Assembly Code Example

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

Note: See “Code Examples” on page 7.

4.7.1 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.

4.8 Register Description

4.8.1 CCP – Configuration Change Protection Register

Bit	7	6	5	4	3	2	1	0	
0x3C	CCP[7:0]								CCP
Read/Write	W	W	W	W	W	W	W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bits 7:0 – CCP[7:0]: Configuration Change Protection**

In order to change the contents of a protected I/O register the CCP register must first be written with the correct signature. After CCP is written the protected I/O registers may be written to during the next four CPU instruction cycles. All interrupts are ignored during these cycles. After these cycles interrupts are automatically handled again by the CPU, and any pending interrupts will be executed according to their priority.

When the protected I/O register signature is written, CCP0 will read as one as long as the protected feature is enabled, while CCP[7:1] will always read as zero.

Table 4-1 shows the signatures that are recognized.

Table 4-1. Signatures Recognized by the Configuration Change Protection Register

Signature	Group	Description
0xD8	IOREG: CLKMSR, CLKPSR, WDTCSR(1), MCUCR(2)	Protected I/O register

Notes: 1. Only WDE and WDP[3:0] bits are protected in WDTCSR.
2. Only BODS bit is protected in MCUCR.

4.8.2 SPH and SPL — Stack Pointer Registers

Initial Value	0	0	0	0	0	0	0	0	
Read/Write	R	R	R	R	R	R	R	R	
Bit	15	14	13	12	11	10	9	8	
0x3E	–	–	–	–	–	–	–	–	SPH
0x3D	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0	SPL
Bit	7	6	5	4	3	2	1	0	
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	RAMEND	RAMEND	RAMEND	RAMEND	RAMEND	RAMEND	RAMEND	RAMEND	

- **Bits 7:0 – SP[7:0]: Stack Pointer**

The Stack Pointer register points to the top of the stack, which is implemented growing from higher memory locations to lower memory locations. Hence, a stack PUSH command decreases the stack pointer.

The stack space in the data SRAM must be defined by the program before any subroutine calls are executed or interrupts are enabled.

In ATtiny20, the SPH register has not been implemented.

4.8.3 SREG – Status Register

Bit	7	6	5	4	3	2	1	0	
0x3F	I	T	H	S	V	N	Z	C	SREG
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	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 document “AVR Instruction Set” and “Instruction Set Summary” on page 205.

- **Bit 6 – T: Bit 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 is useful in BCD arithmetic. See document “AVR Instruction Set” and section “Instruction Set Summary” on page 205 for detailed information.

- **Bit 4 – S: Sign Bit, 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 document “AVR Instruction Set” and section “Instruction Set Summary” on page 205 for detailed information.

- **Bit 3 – V: Two’s Complement Overflow Flag**

The Two’s Complement Overflow Flag V supports two’s complement arithmetic. See document “AVR Instruction Set” and section “Instruction Set Summary” on page 205 for detailed information.

- **Bit 2 – N: Negative Flag**

The Negative Flag N indicates a negative result in an arithmetic or logic operation. See document “AVR Instruction Set” and section “Instruction Set Summary” on page 205 for detailed information.

- **Bit 1 – Z: Zero Flag**

The Zero Flag Z indicates a zero result in an arithmetic or logic operation. See document “AVR Instruction Set” and section “Instruction Set Summary” on page 205 for detailed information.

- **Bit 0 – C: Carry Flag**

The Carry Flag C indicates a carry in an arithmetic or logic operation. See document “AVR Instruction Set” and section “Instruction Set Summary” on page 205 for detailed information.

5. Memories

This section describes the different memories in the ATtiny20. The device has two main memory areas, the program memory space and the data memory space.

5.1 In-System Re-programmable Flash Program Memory

The ATtiny20 contains 2K byte 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 1024 x 16.

The Flash memory has an endurance of at least 10,000 write/erase cycles. The ATtiny20 Program Counter (PC) is 10 bits wide, thus capable of addressing the 1024 program memory locations, starting at 0x000. “Memory Programming” on page 159 contains a detailed description on Flash data serial downloading.

Constant tables can be allocated within the entire address space of program memory. Since program memory can not be accessed directly, it has been mapped to the data memory. The mapped program memory begins at byte address 0x4000 in data memory (see Figure 5-1 on page 15). Although programs are executed starting from address 0x000 in program memory it must be addressed starting from 0x4000 when accessed via the data memory.

Internal write operations to Flash program memory have been disabled and program memory therefore appears to firmware as read-only. Flash memory can still be written to externally but internal write operations to the program memory area will not be successful.

Timing diagrams of instruction fetch and execution are presented in “Instruction Execution Timing” on page 11.

5.2 Data Memory

Data memory locations include the I/O memory, the internal SRAM memory, the non-volatile memory lock bits, and the Flash memory. See Figure 5-1 for an illustration on how the ATtiny20 memory space is organized.

Figure 5-1. Data Memory Map (Byte Addressing)

I/O SPACE	0x0000 ... 0x003F
SRAM DATA MEMORY	0x0040 ... 0x00BF
(reserved)	0x00C0 ... 0x3EFF
NVM LOCK BITS	0x3F00 ... 0x3F01
(reserved)	0x3F02 ... 0x3F3F
CONFIGURATION BITS	0x3F40 ... 0x3F41
(reserved)	0x3F42 ... 0x3F7F
CALIBRATION BITS	0x3F80 ... 0x3F81
(reserved)	0x3F82 ... 0x3FBF
DEVICE ID BITS	0x3FC0 ... 0x3FC3
(reserved)	0x3FC4 ... 0x3FFF
FLASH PROGRAM MEMORY	0x4000 ... 0x47FF
(reserved)	0x4800 ... 0xFFFF

The first 64 locations are reserved for I/O memory, while the following 128 data memory locations (from 0x0040 to 0x00BF) address the internal data SRAM.

The non-volatile memory lock bits and all the Flash memory sections are mapped to the data memory space. These locations appear as read-only for device firmware.

The four different addressing modes for data memory are direct, indirect, indirect with pre-decrement, and indirect with post-increment. In the register file, registers R26 to R31 function as pointer registers for indirect addressing.

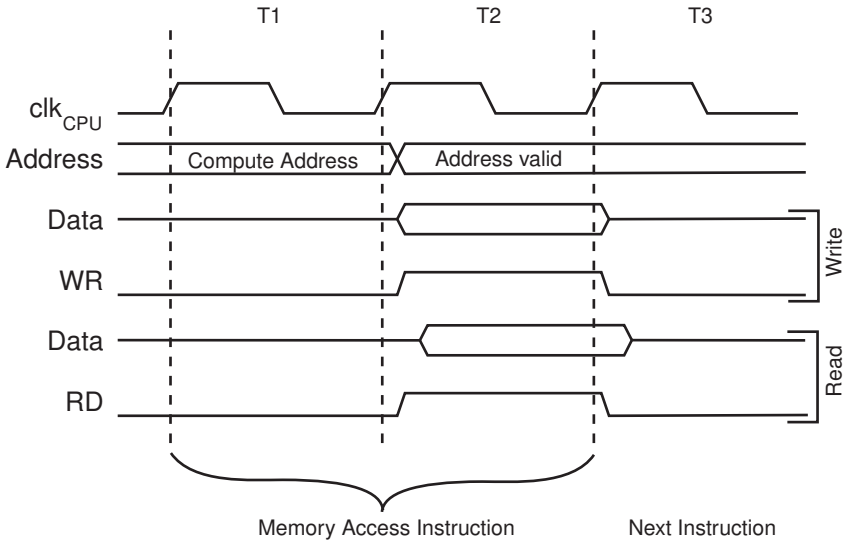
The IN and OUT instructions can access all 64 locations of I/O memory. Direct addressing using the LDS and STS instructions reaches the 128 locations between 0x0040 and 0x00BF.

The indirect addressing reaches the entire data memory space. When using indirect addressing modes with automatic pre-decrement and post-increment, the address registers X, Y, and Z are decremented or incremented.

5.2.1 Data Memory Access Times

This section describes the general access timing concepts for internal memory access. The internal data SRAM access is performed in two clk_{CPU} cycles as described in Figure 5-2.

Figure 5-2. On-chip Data SRAM Access Cycles



5.3 I/O Memory

The I/O space definition of the ATtiny20 is shown in “Register Summary” on page 203.

All ATtiny20 I/Os and peripherals are placed in the I/O space. All I/O locations may be accessed using the LD and ST instructions, enabling data transfer between the 16 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. See document “AVR Instruction Set” and section “Instruction Set Summary” on page 205 for more details. When using the I/O specific commands IN and OUT, the I/O addresses 0x00 - 0x3F must 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.

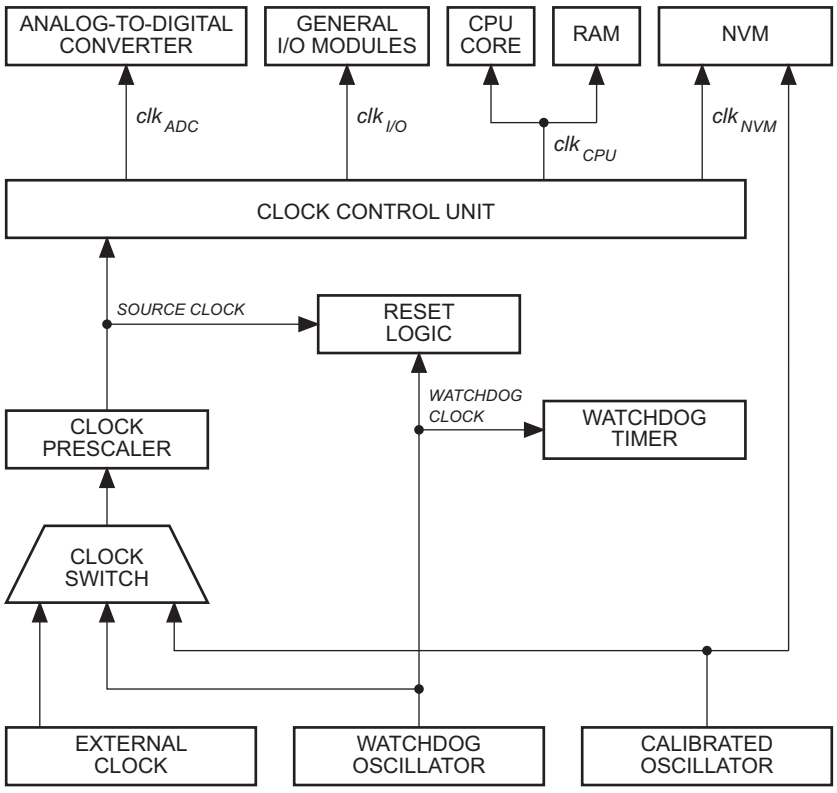
Some of the status flags are cleared by writing a logical one to them. Note that CBI and SBI instructions will only operate on the specified bit, and can therefore be used on registers containing such status flags. The CBI and SBI instructions work on registers in the address range 0x00 to 0x1F, only.

The I/O and Peripherals Control Registers are explained in later sections.

6. Clock System

Figure 6-1 presents the principal clock systems and their distribution in ATtiny20. All of 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 and power reduction register bits, as described in “Power Management and Sleep Modes” on page 23. The clock systems is detailed below.

Figure 6-1. Clock Distribution



6.1 Clock Subsystems

The clock subsystems are detailed in the sections below.

6.1.1 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 System Registers and the SRAM data memory. Halting the CPU clock inhibits the core from performing general operations and calculations.

6.1.2 I/O Clock – $clk_{I/O}$

The I/O clock is used by the majority of the I/O modules, like Timer/Counter. The I/O clock is also used by the External Interrupt module, but note that some external interrupts are detected by asynchronous logic, allowing such interrupts to be detected even if the I/O clock is halted.

6.1.3 NVM clock - clk_{NVM}

The NVM clock controls operation of the Non-Volatile Memory Controller. The NVM clock is usually active simultaneously with the CPU clock.

6.1.4 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.

6.2 Clock Sources

All synchronous clock signals are derived from the main clock. The device has three alternative sources for the main clock, as follows:

- Calibrated Internal 8 MHz Oscillator (see page 18)
- External Clock (see page 18)
- Internal 128 kHz Oscillator (see page 18)

See Table 6-3 on page 21 on how to select and change the active clock source.

6.2.1 Calibrated Internal 8 MHz Oscillator

The calibrated internal oscillator provides an approximately 8 MHz clock signal. Though voltage and temperature dependent, this clock can be very accurately calibrated by the user. See Table 20-2 on page 169, and "Internal Oscillator Speed" on page 200 for more details.

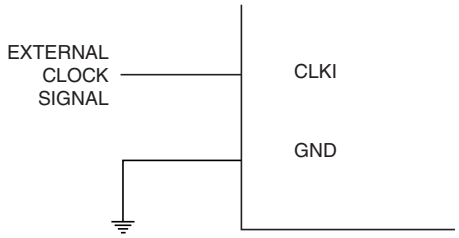
This clock may be selected as the main clock by setting the Clock Main Select bits CLKMS[1:0] in CLKMSR to 0b00. Once enabled, the oscillator will operate with no external components. During reset, hardware loads the calibration byte into the OSCCAL register and thereby automatically calibrates the oscillator. The accuracy of this calibration is shown as Factory calibration in Table 20-2 on page 169.

When this oscillator is used as the main clock, the watchdog oscillator will still be used for the watchdog timer and reset time-out. For more information on the pre-programmed calibration value, see section "Calibration Section" on page 162.

6.2.2 External Clock

To use the device with an external clock source, CLKI should be driven as shown in Figure 6-2. The external clock is selected as the main clock by setting CLKMS[1:0] bits in CLKMSR to 0b10.

Figure 6-2. External Clock Drive Configuration



When applying an external clock, it is required to avoid sudden changes in the applied clock frequency to ensure stable operation of the MCU. A variation in frequency of more than 2% from one clock cycle to the next can lead to unpredictable behavior. It is required to ensure that the MCU is kept in reset during such changes in the clock frequency.

6.2.3 Internal 128 kHz Oscillator

The internal 128 kHz oscillator is a low power oscillator providing a clock of 128 kHz. The frequency depends on supply voltage, temperature and batch variations. This clock may be select as the main clock by setting the CLKMS[1:0] bits in CLKMSR to 0b01.

6.2.4 Switching Clock Source

The main clock source can be switched at run-time using the “CLKMSR – Clock Main Settings Register” on page 20. When switching between any clock sources, the clock system ensures that no glitch occurs in the main clock.

6.2.5 Default Clock Source

The calibrated internal 8 MHz oscillator is always selected as main clock when the device is powered up or has been reset. The synchronous system clock is the main clock divided by 8, controlled by the System Clock Prescaler. The Clock Prescaler Select Bits can be written later to change the system clock frequency. See “System Clock Prescaler”.

6.3 System Clock Prescaler

The system clock is derived from the main clock via the System Clock Prescaler. The system clock can be divided by setting the “CLKPSR – Clock Prescale Register” on page 21. The system clock prescaler can be used to decrease power consumption at times when requirements for processing power is low or to bring the system clock within limits of maximum frequency. The prescaler can be used with all main clock source options, and it will affect the clock frequency of the CPU and all synchronous peripherals.

The System Clock Prescaler can be used to implement run-time changes of the internal clock frequency while still ensuring stable operation.

6.3.1 Switching Prescaler Setting

When switching between prescaler settings, the system clock prescaler ensures that no glitch occurs in the system clock and that no intermediate frequency is higher than neither the clock frequency corresponding the previous setting, nor the clock frequency corresponding to the new setting.

The ripple counter that implements the prescaler runs at the frequency of the main clock, which may be faster than the CPU's clock frequency. Hence, it is not possible to determine the state of the prescaler - even if it were readable, and the exact time it takes to switch from one clock division to another cannot be exactly predicted.

From the time the CLKPS values are written, it takes between T1 + T2 and T1 + 2*T2 before the new clock frequency is active. In this interval, two active clock edges are produced. Here, T1 is the previous clock period, and T2 is the period corresponding to the new prescaler setting.

6.4 Starting

6.4.1 Starting from Reset

The internal reset is immediately asserted when a reset source goes active. The internal reset is kept asserted until the reset source is released and the start-up sequence is completed. The start-up sequence includes three steps, as follows.

1. The first step after the reset source has been released consists of the device counting the reset start-up time. The purpose of this reset start-up time is to ensure that supply voltage has reached sufficient levels. The reset start-up time is counted using the internal 128 kHz oscillator. See Table 6-1 for details of reset start-up time.
   Note that the actual supply voltage is not monitored by the start-up logic. The device will count until the reset start-up time has elapsed even if the device has reached sufficient supply voltage levels earlier.
2. The second step is to count the oscillator start-up time, which ensures that the calibrated internal oscillator has reached a stable state before it is used by the other parts of the system. The calibrated internal oscillator needs to oscillate for a minimum number of cycles before it can be considered stable. See Table 6-1 for details of the oscillator start-up time.
3. The last step before releasing the internal reset is to load the calibration and the configuration values from the Non-Volatile Memory to configure the device properly. The configuration time is listed in Table 6-1.

Table 6-1. Start-up Times when Using the Internal Calibrated Oscillator

Reset	Oscillator	Configuration	Total start-up time
64 ms	6 cycles	21 cycles	64 ms + 6 oscillator cycles + 21 system clock cycles (1)(2)

Notes: 1. After powering up the device or after a reset the system clock is automatically set to calibrated internal 8 MHz oscillator, divided by 8
2. When the Brown-out Detection is enabled, the reset start-up time is 128 ms after powering up the device.

6.4.2 Starting from Power-Down Mode

When waking up from Power-Down sleep mode, the supply voltage is assumed to be at a sufficient level and only the oscillator start-up time is counted to ensure the stable operation of the oscillator. The oscillator start-up time is counted on the selected main clock, and the start-up time depends on the clock selected. See Table 6-2 for details.

Table 6-2. Start-up Time from Power-Down Sleep Mode

Oscillator start-up time	Total start-up time
6 cycles	6 oscillator cycles (1)(2)

Notes: 1. The start-up time is measured in main clock oscillator cycles.
2. When using software BOD disable, the wake-up time from sleep mode will be approximately 60 µs.

6.4.3 Starting from Idle / ADC Noise Reduction / Standby Mode

When waking up from Idle, ADC Noise Reduction or Standby Mode, the oscillator is already running and no oscillator start-up time is introduced.

6.5 Register Description

6.5.1 CLKMSR – Clock Main Settings Register

Bit	7	6	5	4	3	2	1	0	
0x37	–	–	–	–	–	–	CLKMS1	CLKMS0	CLKMSR
Read/Write	R	R	R	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bits 7:2 – Res: Reserved Bits**

These bits are reserved and will always read as zero.

- **Bits 1:0 – CLKMS[1:0]: Clock Main Select Bits**

These bits select the main clock source of the system. The bits can be written at run-time to switch the source of the main clock. The clock system ensures glitch free switching of the main clock source.

The main clock alternatives are shown in Table 6-3.

Table 6-3. Selection of Main Clock

CLKM1	CLKM0	Main Clock Source
0	0	Calibrated Internal 8 MHz Oscillator
0	1	Internal 128 kHz Oscillator (WDT Oscillator)
1	0	External clock
1	1	Reserved

To avoid unintentional switching of main clock source, a protected change sequence must be followed to change the CLKMS bits, as follows:

1. Write the signature for change enable of protected I/O register to register CCP
2. Within four instruction cycles, write the CLKMS bits with the desired value

6.5.2 CLKPSR – Clock Prescale Register

Bit	7	6	5	4	3	2	1	0	
0x36	–	–	–	–	CLKPS3	CLKPS2	CLKPS1	CLKPS0	CLKPSR
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	1	1	

- **Bits 7:4 – Res: Reserved Bits**

These bits are reserved and will always read as zero.

- **Bits 3:0 – CLKPS[3:0]: Clock Prescaler Select Bits 3 - 0**

These bits define the division factor between the selected clock source and the internal system clock. These bits can be written at run-time to vary the clock frequency and suit the application requirements. As the prescaler divides the master clock input to the MCU, the speed of all synchronous peripherals is reduced accordingly. The division factors are given in Table 6-4.

Table 6-4. Clock Prescaler Select

CLKPS3	CLKPS2	CLKPS1	CLKPS0	Clock Division Factor
0	0	0	0	1
0	0	0	1	2
0	0	1	0	4
0	0	1	1	8 (default)
0	1	0	0	16
0	1	0	1	32
0	1	1	0	64
0	1	1	1	128
1	0	0	0	256
1	0	0	1	Reserved

CLKPS3	CLKPS2	CLKPS1	CLKPS0	Clock Division Factor
1	0	1	0	Reserved
1	0	1	1	Reserved
1	1	0	0	Reserved
1	1	0	1	Reserved
1	1	1	0	Reserved
1	1	1	1	Reserved

To avoid unintentional changes of clock frequency, a protected change sequence must be followed to change the CLKPS bits:

1. Write the signature for change enable of protected I/O register to register CCP
2. Within four instruction cycles, write the desired value to CLKPS bits

At start-up, the CLKPS bits will be reset to 0b0011 to select the clock division factor of 8. The application software must ensure that a sufficient division factor is chosen if the selected clock source has a higher frequency than the maximum frequency of the device at the present operating conditions.

6.5.3 OSCCAL – Oscillator Calibration Register

Bit	7	6	5	4	3	2	1	0	
0x39	CAL7	CAL6	CAL5	CAL4	CAL3	CAL2	CAL1	CAL0	OSCCAL
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bits 7:0 – CAL[7:0]: Oscillator Calibration Value**

The oscillator calibration register is used to trim the calibrated internal oscillator and remove process variations from the oscillator frequency. A pre-programmed calibration value is automatically written to this register during chip reset, giving the factory calibrated frequency as specified in Table 20-2, “Calibration Accuracy of Internal RC Oscillator,” on page 169.

The application software can write this register to change the oscillator frequency. The oscillator can be calibrated to frequencies as specified in Table 20-2, “Calibration Accuracy of Internal RC Oscillator,” on page 169. Calibration outside the range given is not guaranteed.

The CAL[7:0] bits are used to tune the frequency of the oscillator. A setting of 0x00 gives the lowest frequency, and a setting of 0xFF gives the highest frequency.

7. Power Management and Sleep Modes

The high performance and industry leading code efficiency makes the AVR microcontrollers an ideal choice for low power applications. In addition, sleep modes enable the application to shut down unused modules in the MCU, thereby saving power. The AVR provides various sleep modes allowing the user to tailor the power consumption to the application's requirements.

7.1 Sleep Modes

Figure 6-1 on page 17 presents the different clock systems and their distribution in ATtiny20. The figure is helpful in selecting an appropriate sleep mode. Table 7-1 shows the different sleep modes and their wake up sources.

Table 7-1. Active Clock Domains and Wake-up Sources in Different Sleep Modes

	Active Clock Domains				Oscillators	Wake-up Sources				
Sleep Mode	clk_{CPU}	clk_{NVM}	clk_{IO}	clk_{ADC}	Main Clock Source Enabled	INT0 and Pin Change	Watchdog Interrupt	TWI Slave	ADC	Other I/O
Idle			X	X	X	X	X	X	X	X
ADC Noise Reduction				X	X	X(1)	X	X(2)	X	
Standby					X	X(1)	X	X(2)		
Power-down						X(1)	X	X(2)		

Notes: 1. For INT0, only level interrupt.
2. Only TWI address match interrupt.

To enter any of the four sleep modes, the SE bits in MCUCR must be written to logic one and a SLEEP instruction must be executed. The SM[2:0] bits in the MCUCR register select which sleep mode (Idle, ADC Noise Reduction, Standby or Power-down) will be activated by the SLEEP instruction. See Table 7-2 for a summary.

If an enabled interrupt occurs while the MCU is in a sleep mode, the MCU wakes up. The MCU is then halted for four cycles in addition to the start-up time, executes the interrupt routine, and resumes execution from the instruction following SLEEP. The contents of the Register File and SRAM are unaltered when the device wakes up from sleep. If a reset occurs during sleep mode, the MCU wakes up and executes from the Reset Vector.

Note that if a level triggered interrupt is used for wake-up the changed level must be held for some time to wake up the MCU (and for the MCU to enter the interrupt service routine). See "External Interrupts" on page 37 for details.

7.1.1 Idle Mode

When bits SM[2:0] are written to 000, the SLEEP instruction makes the MCU enter Idle mode, stopping the CPU but allowing the analog comparator, ADC, timer/counters, watchdog, TWI, SPI and the interrupt system to continue operating. This sleep mode basically halts clk_{CPU} and clk_{NVM}, while allowing the other clocks to run.

Idle mode enables the MCU to wake up from external triggered interrupts as well as internal ones like the timer overflow. If wake-up from the analog comparator interrupt is not required, the analog comparator can be powered down by setting the ACD bit in "ACSRA – Analog Comparator Control and Status Register" on page 106. This will reduce power consumption in idle mode. If the ADC is enabled, a conversion starts automatically when this mode is entered.

7.1.2 ADC Noise Reduction Mode

When bits SM[2:0] are written to 001, the SLEEP instruction makes the MCU enter ADC Noise Reduction mode, stopping the CPU but allowing the ADC, the external interrupts, TWI and the watchdog to continue operating (if enabled). This sleep mode halts $clk_{I/O}$, clk_{CPU}, and clk_{NVM}, while allowing the other clocks to run.

This mode improves the noise environment for the ADC, enabling higher resolution measurements. If the ADC is enabled, a conversion starts automatically when this mode is entered.

7.1.3 Power-down Mode

When bits SM[2:0] are written to 010, the SLEEP instruction makes the MCU enter Power-down mode. In this mode, the oscillator is stopped, while the external interrupts, TWI and the watchdog continue operating (if enabled). Only a watchdog reset, an external level interrupt on INT0, a pin change interrupt, or a TWI slave interrupt can wake up the MCU. This sleep mode halts all generated clocks, allowing operation of asynchronous modules only.

7.1.4 Standby Mode

When bits SM[2:0] are written to 100, the SLEEP instruction makes the MCU enter Standby mode. This mode is identical to Power-down with the exception that the oscillator is kept running. This reduces wake-up time, because the oscillator is already running and doesn't need to be started up.

7.2 Software BOD Disable

When the Brown-out Detector (BOD) is enabled by BODLEVEL fuses (see Table 19-5 on page 161), the BOD is actively monitoring the supply voltage during a sleep period. In some devices it is possible to save power by disabling the BOD by software in Power-Down and Stand-By sleep modes. The sleep mode power consumption will then be at the same level as when BOD is globally disabled by fuses.

If BOD is disabled by software, the BOD function is turned off immediately after entering the sleep mode. Upon wake-up from sleep, BOD is automatically enabled again. This ensures safe operation in case the V_{CC} level has dropped during the sleep period.

When the BOD has been disabled, the wake-up time from sleep mode will be approximately 60µs to ensure that the BOD is working correctly before the MCU continues executing code.

BOD disable is controlled by the BODS (BOD Sleep) bit of MCU Control Register, see "MCUCR – MCU Control Register" on page 26. Writing this bit to one turns off BOD in Power-Down and Stand-By, while writing a zero keeps the BOD active. The default setting is zero, i.e. BOD active.

Writing to the BODS bit is controlled by a timed sequence, see "MCUCR – MCU Control Register" on page 26.

7.3 Power Reduction Register

The Power Reduction Register (PRR), see "PRR – Power Reduction Register" on page 27, provides a method to reduce power consumption by stopping the clock to individual peripherals. When the clock for a peripheral is stopped then:

- The current state of the peripheral is frozen.
- The associated registers can not be read or written.
- Resources used by the peripheral will remain occupied.

The peripheral should in most cases be disabled before stopping the clock. Clearing the PRR bit wakes up the peripheral and puts it in the same state as before shutdown.

Peripheral shutdown can be used in Idle mode and Active mode to significantly reduce the overall power consumption. See "Supply Current of I/O Modules" on page 174 for examples. In all other sleep modes, the clock is already stopped.

7.4 Minimizing Power Consumption

There are several issues to consider when trying to minimize the power consumption in an AVR Core controlled system. In general, sleep modes should be used as much as possible, and the sleep mode should be selected so that as few as possible of the device's functions are operating. All functions not needed should be disabled. In particular, the following modules may need special consideration when trying to achieve the lowest possible power consumption.

7.4.1 Analog Comparator

When entering Idle mode, the analog comparator should be disabled if not used. In the power-down mode, the analog comparator is automatically disabled. See "Analog Comparator" on page 105 for further details.

7.4.2 Analog to Digital Converter

If enabled, the ADC will be enabled in all sleep modes. To save power, the ADC should be disabled before entering any sleep mode. When the ADC is turned off and on again, the next conversion will be an extended conversion. See "Analog to Digital Converter" on page 109 for details on ADC operation.

7.4.3 Watchdog Timer

If the Watchdog Timer is not needed in the application, this module should be turned off. If the Watchdog Timer is enabled, it will be enabled in all sleep modes, and hence, always consume power. In the deeper sleep modes, this will contribute significantly to the total current consumption. Refer to "Watchdog Timer" on page 31 for details on how to configure the Watchdog Timer.

7.4.4 Brown-out Detector

If the Brown-out Detector is not needed in the application, this module should be turned off. If the Brown-out Detector is enabled by the BODLEVEL Fuses, it will be enabled in all sleep modes, and hence, always consume power. In the deeper sleep modes, this will contribute significantly to the total current consumption. See "Brown-out Detection" on page 30 and "Software BOD Disable" on page 24 for details on how to configure the Brown-out Detector.

7.4.5 Port Pins

When entering a sleep mode, all port pins should be configured to use minimum power. The most important thing is then to ensure that no pins drive resistive loads. In sleep modes where the I/O clock ($clk_{I/O}$) is stopped, the input buffers of the device will be disabled. This ensures that no power is consumed by the input logic when not needed. In some cases, the input logic is needed for detecting wake-up conditions, and it will then be enabled. Refer to the section "Digital Input Enable and Sleep Modes" on page 46 for details on which pins are enabled. If the input buffer is enabled and the input signal is left floating or has an analog signal level close to $V_{CC}/2$, the input buffer will use excessive power.

For analog input pins, the digital input buffer should be disabled at all times. An analog signal level close to $V_{CC}/2$ on an input pin can cause significant current even in active mode. Digital input buffers can be disabled by writing to the Digital Input Disable Register (DIDR0). Refer to "DIDR0 – Digital Input Disable Register 0" on page 108 for details.