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

Features

- High Performance, Low Power Atmel AVR 8-bit Microcontroller
- Advanced RISC Architecture
  - 123 Powerful Instructions - Most Single Clock Cycle Execution
  - 32 x 8 General Purpose Working Registers
  - Fully Static Operation
- Non-volatile Program and Data Memories
  - 16Kbyte of In-system Programmable (ISP) Program Memory Flash
    - Endurance: 10,000 Write/Erase Cycles
  - 512Bytes In-system Programmable EEPROM
    - Endurance: 100,000 Write/Erase Cycles
  - 512Bytes Internal SRAM
  - Programming Lock for Self-programming Flash Program and EEPROM Data Security
  - Low Size LIN/UART Software In-system Programmable
- Peripheral Features
  - LF-RFID Reader/Writer Front End
    - Carrier Frequency f_{OSC} = 100kHz to 150kHz
    - Typical Data Rate up to 5Kbaud at 125kHz
    - Suitable for Manchester anf Biphase Modulation
    - Does not Require an External Crystal
    - Power Supply from the Car Battery or from 5V Regulated Voltage
    - Optimized for Car Immobilizer Applications
    - Tuning Capability
  - LIN 2.1 and 1.3 Controller or 8-bit UART
  - 8-bit Asynchronous Timer/Counter 0:
    - 10-bit Clock Prescaler
    - 1 Output Compare or 8-bit PWM Channel
  - 16-bit Synchronous Timer/Counter 1:
    - 10-bit Clock Prescaler
    - External Event Counter
    - 2 Output Compares Units or 16-bit PWM Channels each Driving up to 4 Output Pins
  - Master/Slave SPI Serial Interface
  - Universal Serial Interface (USI) with Start Condition Detector (Master/Slave SPI, TWI, ...)
  - 10-bit ADC:
    - 11 Single-ended Channels
    - 8 Differential ADC Channel Pairs with Programmable Gain (8x or 20x)
  - On-chip Analog Comparator with Selectable Voltage Reference
  - 100µA ±10% Current Source (LIN Node Identification)
  - On-chip Temperature Sensor
  - Programmable Watchdog Timer with Separate On-chip Oscillator

125kHz LF Reader/Writer with Integrated Atmel AVR Microcontroller

Atmel ATA5505

Preliminary

9219A–RFID–01/11

- Special Microcontroller Features
  - Dynamic Clock Switching (External/Internal RC/Watchdog Clock) for Power Control, EMC Reduction
  - DebugWIRE On-chip Debug (OCD) System
  - Hardware In-System Programmable (ISP) via SPI Port
  - External and Internal Interrupt Sources
  - Interrupt and Wake-up on Pin Change
  - Low Power Idle, ADC Noise Reduction, and Power-down Modes
  - Enhanced Power-on Reset Circuit
  - Programmable Brown-out Detection Circuit
  - Internal Calibrated RC Oscillator 8MHz
  - 4-16 MHz and 32 KHz Crystal/Ceramic Resonator Oscillators
- Operating Voltage:
  - 7 to 16V
  - 4.5 to 5.5V Internal Voltage Regulator available for Digital and Logic
- Operating Temperature: –40°C to +85°C

Applications

- Access Control Units
- Animal Identification
- Component Authetication
- Brand Protection
- Automation
- Industrial
- Waste Management
- Process Control

1. Description

The Atmel® ATA5505 is an Atmel AVR® microcontroller with LF-RFID reader/writer front end and LIN interface for low cost network applications.

The Atmel ATA5505 incorporates the energy-transfer circuit for supplying the transponder. It consists of an on-chip power supply, an oscillator, and a coil driver optimized for automotive-specific distances. It also includes all signal-processing circuits which are necessary to transform the small input signal into a microcontroller-compatible signal.

The Atmel ATA5505 integrates a low-power CMOS 8-bit microcontroller based on the Atmel AVR enhanced RISC architecture. By executing powerful instructions in a single clock cycle, the ATA5505 achieves throughputs approaching 1 MIPS per MHz, allowing the system designer to optimize power consumption versus processing speed.

The Atmel AVR 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 Atmel ATA5505 provides the following features: 16K byte of in-system programmable Flash, 512bytes EEPROM, 512bytes SRAM, 16 general purpose I/O lines, 32 general purpose working registers, one 8-bit timer/counter with compare modes, one 8-bit high speed timer/counter, a universal serial interface, a LIN controller, internal and external interrupts, an 11-channel, 10-bit ADC, a programmable watchdog timer with internal oscillator, and three software selectable power saving modes. The Idle mode stops the CPU while allowing the SRAM, timer/counter, ADC, analog comparator, and interrupt system to continue functioning. Power-down mode saves the register contents, disabling all chip functions until the next interrupt or hardware reset. To minimize switching noise during ADC conversions, ADC noise reduction mode stops the CPU and all I/O modules except ADC,

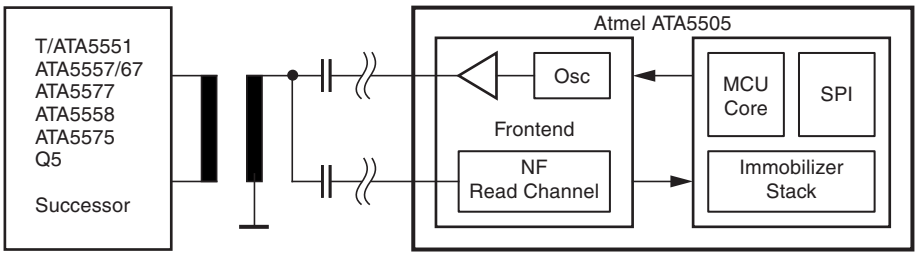
The device is manufactured using Atmel's high-density non-volatile memory technology. The on-chip ISP Flash allows the program memory to be re-programmed in-system via an SPI serial interface, by a conventional non-volatile memory programmer or by an on-chip boot code running on the Atmel AVR core. The boot program can use any interface to download the application program in the Flash memory. By combining an 8-bit RISC CPU with in-system self-programmable Flash on a monolithic chip, the Atmel ATA5505 is a powerful microcontroller providing a highly flexible and cost-effective solution to many embedded control applications.

The Atmel ATA5505 AVR is supported with a full suite of program and system development tools including: C compilers, macro assemblers, program debuggers/simulators, in-circuit emulators, and evaluation kits.

1.1 System Block Diagram

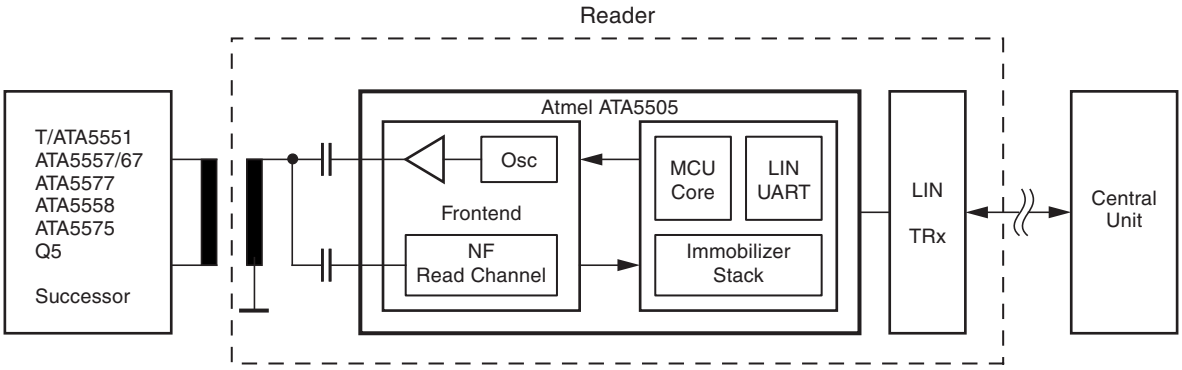
The system block diagram for a stand alone reader application is shown in Figure 1-1. No additional microcontroller is required..

Figure 1-1. System Block Diagram - Stand Alone Reader



A system block diagram for a reader station application with a LIN transceiver IC is shown in Figure 1-2. The reader is then communicating with a central unit.

Figure 1-2. System Block Diagram - Reader with LIN Connection



All Atmel® LF-RFID tag ICs can be read by the ATA5505: e5530, TK5530, e5551, T5551, TK5551, e5552, TK5552, e5554, T5555, Q5, T5557, ATA5567, ATA5558, T5561, TK5561, ATA5577, ATA5575.

1.2 Block Diagram

In Figure 1-3 on page 5 the basic architecture of the Atmel ATA5505 consisting of the Atmel AVR and the front end is shown. The Atmel AVR is a low-power CMOS 8-bit microcontroller with RISC architecture. The Atmel AVR includes several features. The main features are in-system programmable Flash, EEPROM, SRAM, Ports A and B as general purpose I/O lines, a timer and a LIN controller. The details of the front end are shown in Figure 1-4 on page 5.

Figure 1-3. Block Diagram

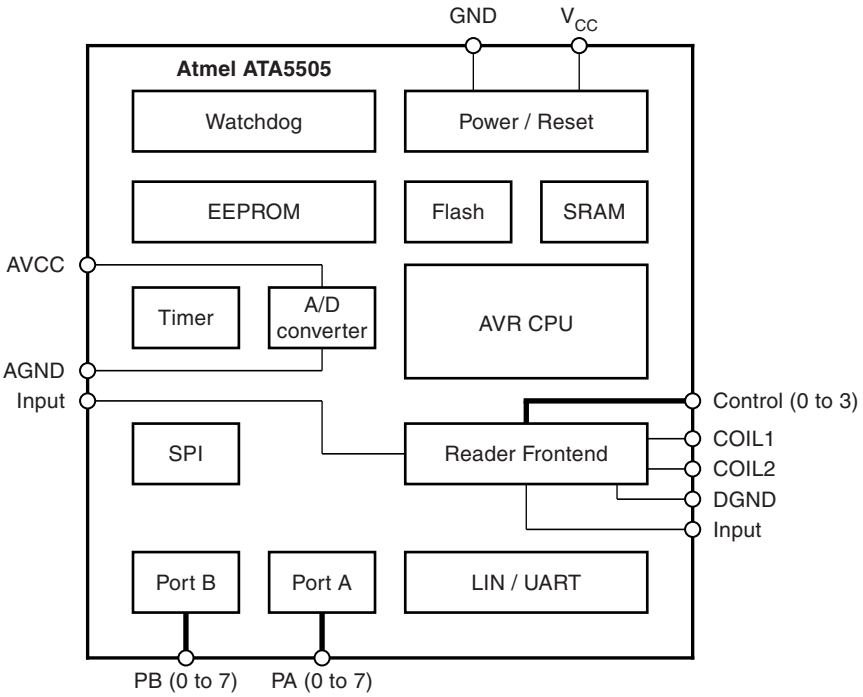
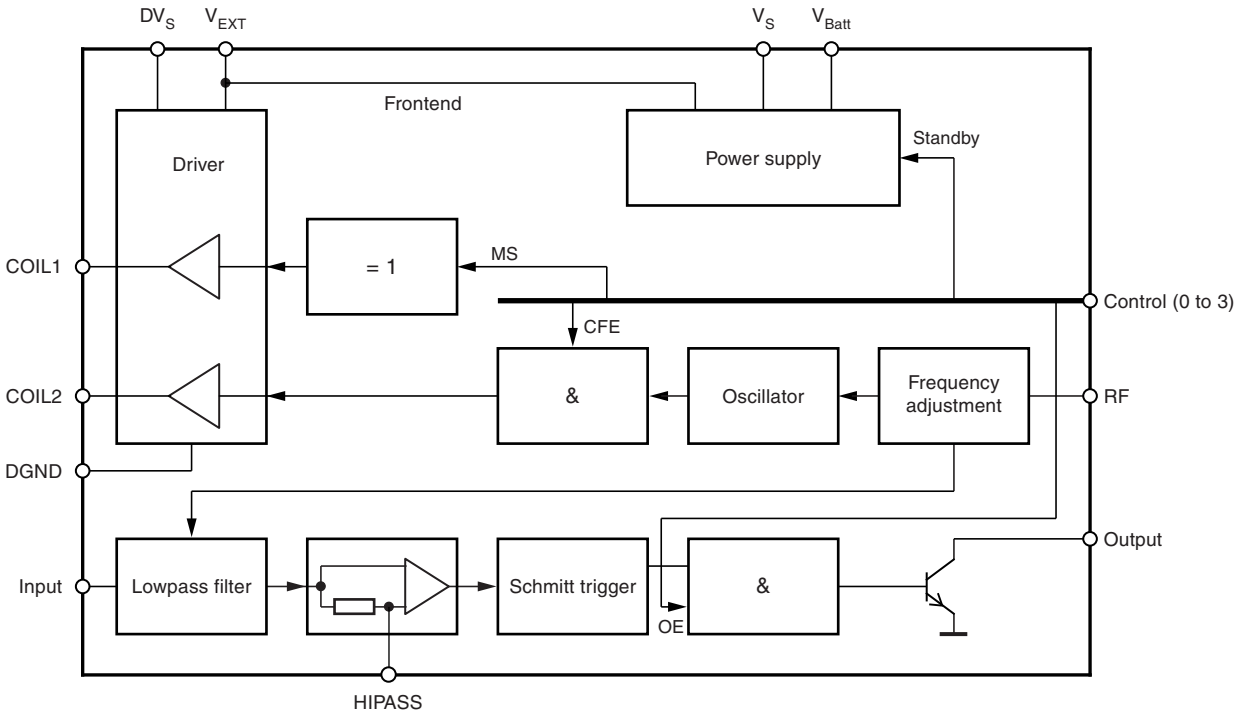


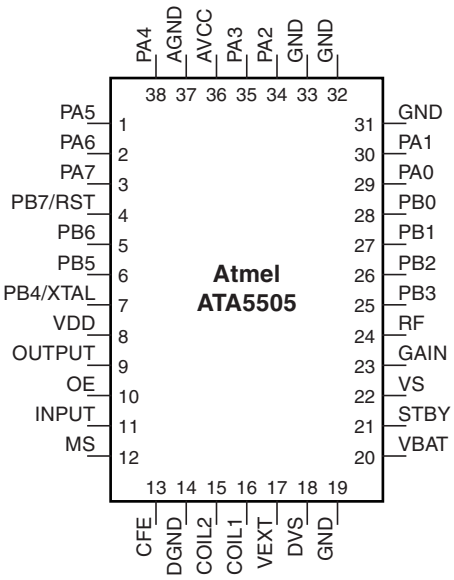
Figure 1-4 shows the front end in more detail. The circuit block depicted consists of the receiver, an internal oscillator and driver for the coil, and a power supply unit.

Figure 1-4. Reader Front End Block Diagram



1.3 Pin Configuration

Figure 1-5. Pinout Atmel® ATA5505 - QFN38 5 mm by 7 mm



1.4 Pin Description

Table 1-1. Pin Description

Pin	Symbol	Function
1	PA5 PCINT5 ADC5 T1 USCK SCL SCK	GPIO Port A – Pin 5 PCINT5 (Pin Change Interrupt 5) ADC5 (ADC Input Channel 5) T1 (Timer/Counter1 Clock Input) USCK (Three-wire Mode USI *Alternate* Clock Input) SCL (Two-wire Mode USI *Alternate* Clock Input) SCK (SPI Master Clock)
2	PA6 PCINT6 ADC6 AIN0 SS	GPIO Port A – Pin 6 PCINT6 (Pin Change Interrupt 6) ADC6 (ADC Input Channel 6) AIN0 (Analog Comparator Negative Input) SS (SPI Slave Select Input)
3	PA7 PCINT7 ADC7 AIN1 XREF AREF	GPIO Port A – Pin 7 PCINT7 (Pin Change Interrupt 7) ADC7 (ADC Input Channel 7) AIN1 (Analog Comparator Positive Input) XREF (Internal Voltage Reference Output) AREF (External Voltage Reference Input)
4	PB7 PCINT15 ADC10 OC1BX RESET dW	GPIO Port B – Pin 7 PCINT15 (Pin Change Interrupt 15) ADC10 (ADC Input Channel 10) OC1BX (Output Compare and PWM Output B-X for Timer/Counter1) RESET (Reset input pin) dW (debugWIRE I/O)

Table 1-1. Pin Description

Pin	Symbol	Function
5	PB6 PCINT14 ADC9 OC1AX INT0	GPIO Port B – Pin 6 PCINT14 (Pin Change Interrupt 14) ADC9 (ADC Input Channel 9) OC1AX (Output Compare and PWM Output A-X for Timer/Counter 1) INT0 (External Interrupt0 Input)
6	PB5 PCINT13 ADC8 OC1BW XTAL2 CLKO	GPIO Port B – Pin 5 PCINT13 (Pin Change Interrupt 13) ADC8 (ADC Input Channel 8) OC1BW (Output Compare and PWM Output B-W for Timer/Counter 1) XTAL2 (Chip Clock Oscillator Pin 2) CLKO (System Clock Output)
7	PB4 PCINT12 OC1AW XTAL1 CLKI	GPIO Port B – Pin 4 PCINT12 (Pin Change Interrupt 12) OC1AW (Output Compare and PWM Output A-W for Timer/Counter 1) XTAL1 (Chip clock Oscillator pin 1) CLKI (External Clock Input)
8	VDD	Atmel® AVR® Supply Voltage
9	OUTPUT	Reader Data Output
10	OE	Reader Station Data Output Enable
11	INPUT	Reader Station Data Input
12	MS	Reader Station Mode Select
13	CFE	Reader Station: Carrier Frequency Enable
14	DGND	Digital Ground (Driver Ground)
15	COIL 2	Reader Coil Driver 2
16	COIL 1	Reader Coil Driver 1
17	VEXT	Reader External Power Supply
18	DVS	Reader Driver Supply Voltage
19	GND	Ground
20	VBATT	Battery Voltage for Reader
21	STANDBY	Front End Standby Input
22	VS	Internal Power Supply (5V) for Reader
23	HIPASS	HIPASS DC Decoupling (Gain)
24	RF	Frequency Adjustment
25	PB3 PCINT11 OC1BV	GPIO Port B - Pin 3 PCINT11 (Pin Change Interrupt 11) OC1BV (Output Compare and PWM Output B-V for Timer/Counter 1)
26	PB2 PCINT10 OC1AV USCK SCL	GPIO Port B – Pin 2 PCINT10 (Pin Change Interrupt 10) OC1AV (Output Compare and PWM Output A-V for Timer/Counter 1) USCK (Three-wire Mode USI *Default* Clock Input) SCL (Two-wire Mode USI *Default* Clock Input)
27	PB1 PCINT9 OC1BU DO	GPIO Port B – Pin 1 PCINT9 (Pin Change Interrupt 9) OC1BU (Output Compare and PWM Output B-U for Timer/Counter 1) DO (Three-wire Mode USI *Default* Data Output)

Table 1-1. Pin Description

Pin	Symbol	Function
28	PB0 PCINT8 OC1AU DI SDA	GPIO Port B – Pin 0 PCINT8 (Pin Change Interrupt 8) OC1AU (Output Compare and PWM Output A-U for Timer/Counter 1) DI (Three-wire Mode USI *Default* Data Input) SDA (Two-wire Mode USI *Default* Data Input / Output)
29	PA0 PCINT0 ADC0 RXD RXLIN	GPIO Port A – Pin 0 PCINT0 (Pin Change Interrupt 0) ADC0 (ADC Input Channel 0) RXD (UART Receive Pin) RXLIN (LIN Receive Pin)
30	PA1 PCINT1 ADC1 TXD TXLIN	GPIO Port A – Pin 1 PCINT1 (Pin Change Interrupt 1) ADC1 (ADC Input Channel 1) TXD (UART Transmit Pin) TXLIN (LIN Transmit Pin)
31	GND	Ground
32	GND	Ground
33	GND	Ground
34	PA2 PCINT2 ADC2 OCA0A DO MISO	GPIO Port A – Pin 2 PCINT2 (Pin Change Interrupt 2) ADC2 (ADC Input Channel 2) OC0A (Output Compare and PWM Output A for Timer/Counter 0) DO (Three-wire Mode USI *Alternate* Data Output) MISO (SPI Master Input/Slave Output)
35	PA3 PCINT3 ADC3 ISRC INT1	GPIO Port A – Pin 3 PCINT3 (Pin Change Interrupt 3) ADC3 (ADC Input Channel 3) ISRC (Current Source Pin) INT1 (External Interrupt1 Input)
36	AVCC	Analog Supply Voltage
37	AGND	Analog Ground
38	PA4 PCINT4 ADC4 ICP1 DI SDA MOSI	GPIO Port A – Pin 4 PCINT4 (Pin Change Interrupt 4) ADC4 (ADC Input Channel 4) ICP1 (Timer/Counter 1 Input Capture Trigger) DI (Three-wire Mode USI *Alternate* Data Input) SDA (Two-wire Mode USI *Alternate* Data Input/Output) MOSI (SPI Master Output/Slave Input)

1.4.1 Port A (PA7..PA0)

Port A is an 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 also serves the functions of various special features of the microcontroller.

1.4.2 Port B (PB7..PB0)

Port B is an 8-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. As inputs, Port B pins that are pulled low externally 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. Port B also serves the functions of various special features of the microcontroller.

1.4.3 RESET

Device reset

1.4.4 XTAL1 and XTAL2

XTAL1 and XTAL2 are input and output, respectively, of an inverting amplifier which can be configured for use as an On-chip oscillator. Either a quartz crystal or a ceramic resonator may be used.

1.5 Atmel AVR Description

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

The Atmel AVR core combines a rich instruction set with 32 general purpose working registers. All 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 microcontroller provides the following features: 16K byte of in-system programmable Flash, 512bytes EEPROM, 512bytes SRAM, 16 general purpose I/O lines, 32 general purpose working registers, one 8-bit timer/counter with compare modes, one 8-bit high speed timer/counter, a universal serial interface, a LIN controller, internal and external interrupts, an 11-channel, 10-bit ADC, a programmable watchdog timer with internal oscillator, and three software selectable power saving modes. The Idle mode stops the CPU while allowing the SRAM, timer/counter, ADC, analog comparator, and interrupt system to continue functioning. Power-down mode saves the register contents, disabling all chip functions until the next interrupt or hardware reset. To minimize switching noise during ADC conversions, ADC noise reduction mode stops the CPU and all I/O modules except ADC.

The device is manufactured using Atmel®'s high- density non-volatile memory technology. The on-chip ISP Flash allows the program memory to be re-programmed In-system through via an SPI serial interface, by a conventional non-volatile memory programmer or by an on-chip boot code running on the Atmel AVR® core. The boot program can use any interface to download the application program in the Flash memory. By combining an 8-bit RISC CPU with In-system self-programmable Flash on a monolithic chip, the Atmel ATA5505 is a powerful microcontroller that provides a highly flexible and cost- effective solution to many embedded control applications.

The embedded microcontroller is supported with a full suite of program and system development tools including: C compilers, macro assemblers, program debuggers/simulators, in-circuit emulators, and evaluation kits.

1.6 Resources

A comprehensive set of development tools, application notes and datasheets are available for download at http://www.atmel.com/avr.

1.7 About Code Examples

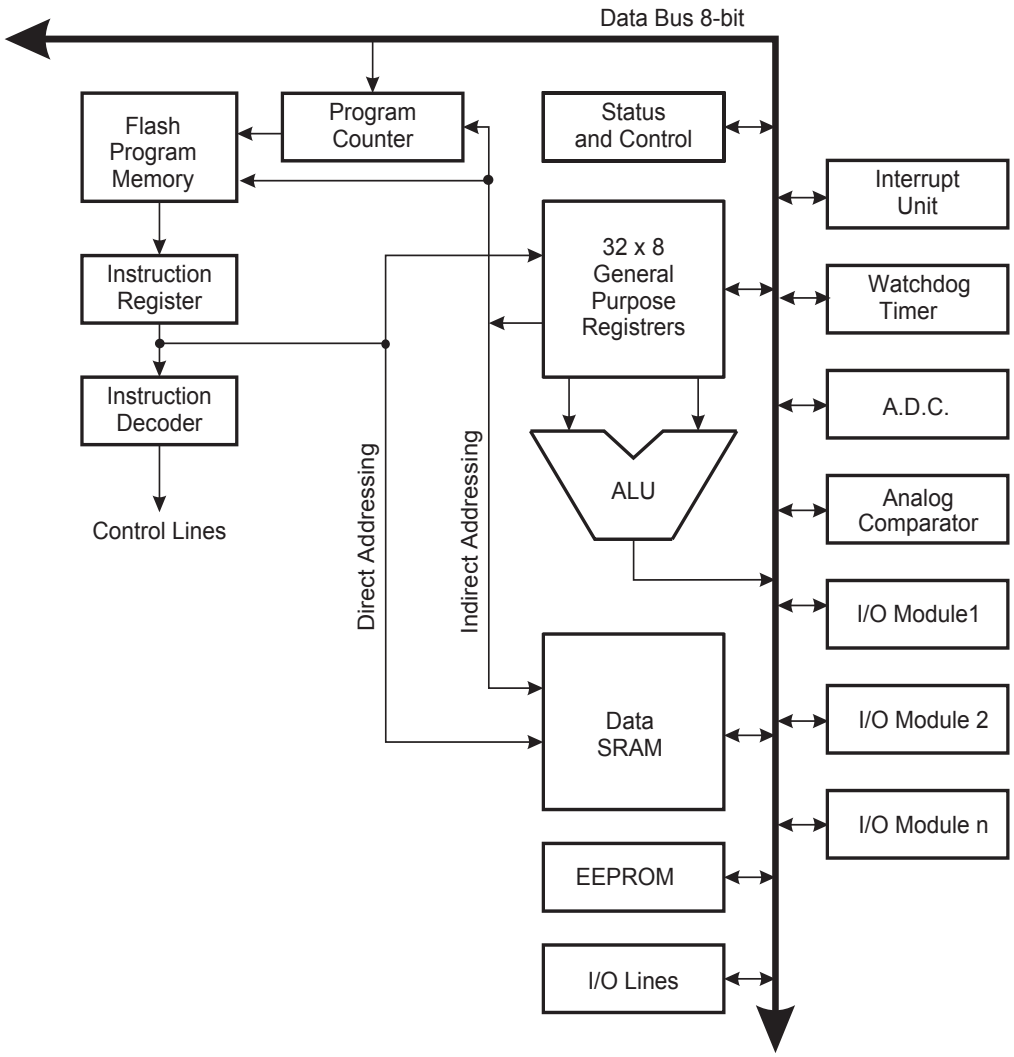
This documentation contains simple code examples that briefly show how to use various parts of the device. These code examples assume that the part-specific header file is included before compilation. Be aware that not all C compiler vendors include bit definitions in the header files and interrupt handling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

2. AVR CPU Core

2.1 Overview

This section discusses the Atmel® 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 2-1. Block Diagram of the Atmel AVR Architecture



In order to maximize performance and parallelism, the Atmel 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 the these address pointers can also be used as an address pointer for look up 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 Atmel® AVR® instructions have a single 16-bit word format. Every Program memory address contains a 16- or 32-bit instruction.

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 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 five different addressing modes supported in the Atmel AVR architecture.

The memory spaces in the Atmel 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.

2.2 ALU – Arithmetic Logic Unit

The high-performance Atmel 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 the “Instruction Set” section for a detailed description.

2.3 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 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.

2.3.1 SREG – Atmel AVR Status Register

The Atmel® AVR® Status Register – SREG – is defined as:

Bit	7	6	5	4	3	2	1	0	
	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 instruction set reference.

- **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 the “Instruction Set Description” 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 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 arithmetics. 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.

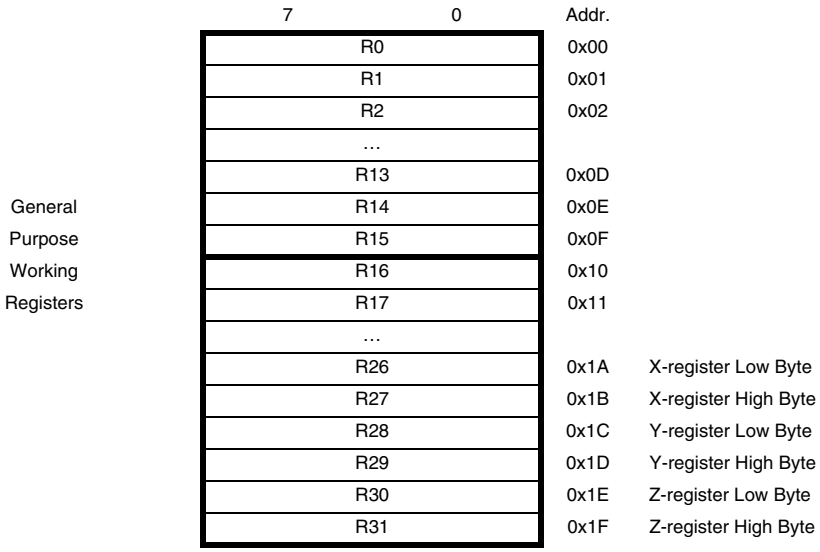
2.4 General Purpose Register File

The Register File is optimized for the Atmel® 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 2-2 shows the structure of the 32 general purpose working registers in the CPU.

Figure 2-2. Atmel 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 Figure 2-2, 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.

2.4.1 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 Figure 2-3 on page 15.

Figure 2-3. 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).

2.5 Stack Pointer

The Stack is mainly used for storing temporary data, for storing local variables and for storing return addresses after interrupts and subroutine calls. The Stack Pointer Register always points to the top of the Stack. Note that the Stack is implemented as growing from higher memory locations to lower memory locations. This implies that a Stack PUSH command decreases the Stack Pointer.

The Stack Pointer points to the data SRAM Stack area where the Subroutine and Interrupt Stacks are located. This Stack space in the data SRAM must be defined by the program before any subroutine calls are executed or interrupts are enabled. The Stack Pointer must be set to point above 0x60. The Stack Pointer is decremented by one when data is pushed onto the Stack with the PUSH instruction, and it is decremented by two when the return address is pushed onto the Stack with subroutine call or interrupt. The Stack Pointer is incremented by one when data is popped from the Stack with the POP instruction, and it is incremented by two when data is popped from the Stack with return from subroutine RET or return from interrupt RETI.

The Atmel® 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 Atmel AVR architecture is so small that only SPL is needed. In this case, the SPH Register will not be present

2.5.1 SPH and SPL – Stack Pointer Register

Bit	15	14	13	12	11	10	9	8	
	SP15	**SP14**	**SP13**	**SP12**	**SP11**	**SP10**	**SP9**	**SP8**	**SPH**
	SP7	**SP6**	**SP5**	**SP4**	**SP3**	**SP2**	**SP1**	**SP0**	**SPL**
	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	
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	ISRAM end (See Table 3-1 on page 19)								

2.6 Instruction Execution Timing

This section describes the general access timing concepts for instruction execution. The Atmel® 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 2-4 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 2-4. The Parallel Instruction Fetches and Instruction Executions

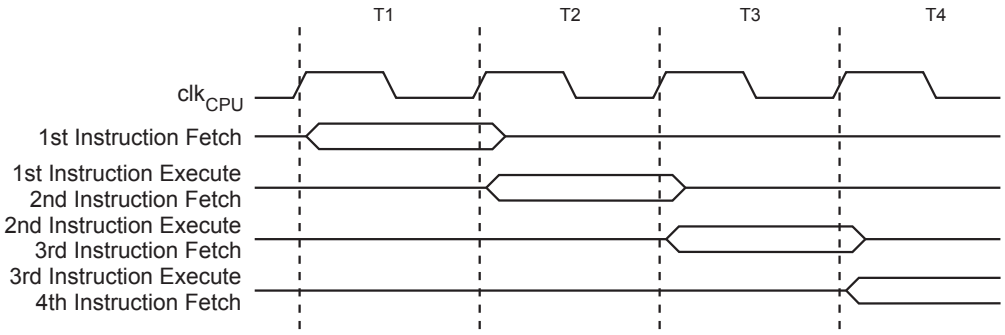
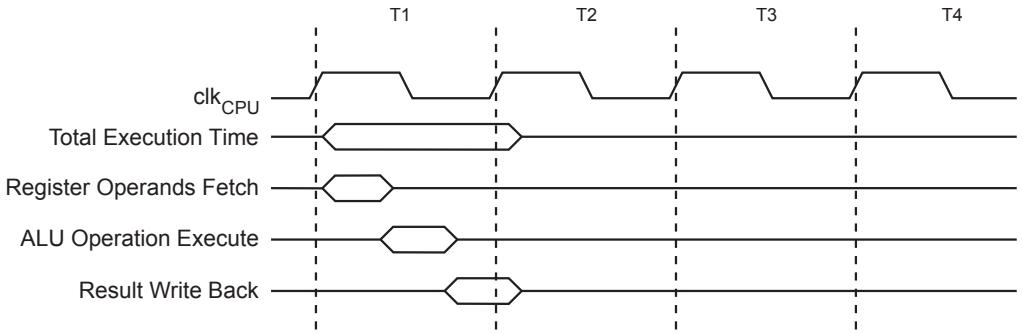


Figure 2-5 shows the internal timing concept for the Register File. In a single clock cycle an ALU operation using two register operands is executed, and the result is stored back to the destination register.

Figure 2-5. Single Cycle ALU Operation



2.7 Reset and Interrupt Handling

The Atmel 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 Section 7. “Interrupts” on page 63. 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.

2.7.1 Interrupt behavior

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 Atmel® 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. The following example shows how this can be used to avoid interrupts during the timed EEPROM write sequence.

Assembly Code Example

```
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

```
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) */
```

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

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

C Code Example

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

2.7.2 Interrupt Response Time

The interrupt execution response for all the enabled Atmel® 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.

3. Atmel AVR Memories

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

Table 3-1. Memory Mapping.

Memory		Mnemonic	Atmel AVR
Flash	Size	Flash size	16 K bytes
	Start Address	-	0x0000
	End Address	Flash end	0x3FFF(1) 0x1FFF(2)
32 Registers	Size	-	32 bytes
	Start Address	-	0x0000
	End Address	-	0x001F
I/O Registers	Size	-	64 bytes
	Start Address	-	0x0020
	End Address	-	0x005F
Ext I/O Registers	Size	-	160 bytes
	Start Address	-	0x0060
	End Address	-	0x00FF
Internal SRAM	Size	ISRAM size	512 bytes
	Start Address	ISRAM start	0x0100
	End Address	ISRAM end	0x02FF
EEPROM	Size	E2 size	512 bytes
	Start Address	-	0x0000
	End Address	E2 end	0x01FF

Notes: 1. Byte address.
2. Word (16-bit) address.

3.1 In-System Re-programmable Flash Program Memory

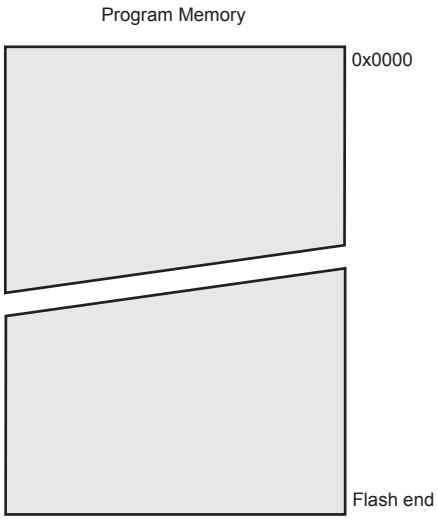
The Atmel AVR contains On-chip In-System Reprogrammable Flash memory for program storage (see “Flash size” in Table 3-1 on page 19). Since all Atmel AVR instructions are 16 or 32 bits wide, the Flash is organized as 16 bits wide. Atmel AVR does not have separate Boot Loader and Application Program sections, and the SPM instruction can be executed from the entire Flash. See SELFPRGEN description in Section 21.2.1 “Store Program Memory Control and Status Register – SPMCSR” on page 230 for more details.

The Flash memory has an endurance of at least 10,000 write/erase cycles in automotive range. The Atmel AVR Program Counter (PC) address the program memory locations. Section 22. “Memory Programming” on page 236 contains a detailed description on Flash data serial downloading using the SPI pins.

Constant tables can be allocated within the entire Program memory address space (see the LPM – Load Program memory instruction description).

Timing diagrams for instruction fetch and execution are presented in Section 2.6 “Instruction Execution Timing” on page 16.

Figure 3-1. Program Memory Map



3.2 SRAM Data Memory

Figure 3-2 shows how the Atmel® AVR® SRAM Memory is organized.

The Atmel AVR 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 in SRAM, only the ST/STS/STD and LD/LDS/LDD instructions can be used.

The 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 locations address the internal data SRAM (see “ISRAM size” in Table 3-1 on page 19).

The five different addressing modes for the Data memory cover: Direct, Indirect with Displacement, Indirect, Indirect with Pre-decrement, and Indirect with Post-increment. In the Register File, registers R26 to R31 feature the indirect addressing pointer registers.

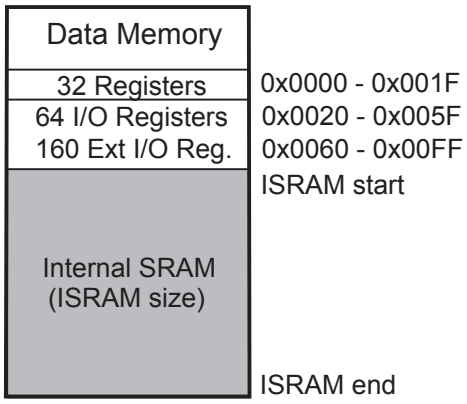
The direct addressing reaches the entire data space.

The Indirect with Displacement mode reaches 63 address locations from the base address given by the Y- or Z-register.

When using register indirect addressing modes with automatic pre-decrement and post-increment, the address registers X, Y, and Z are decremented or incremented.

The 32 general purpose working registers, 64 I/O Registers, 160 Extended I/O Registers and the internal data SRAM in the Atmel AVR are all accessible through all these addressing modes. The Register File is described in “General Purpose Register File” on page 14.

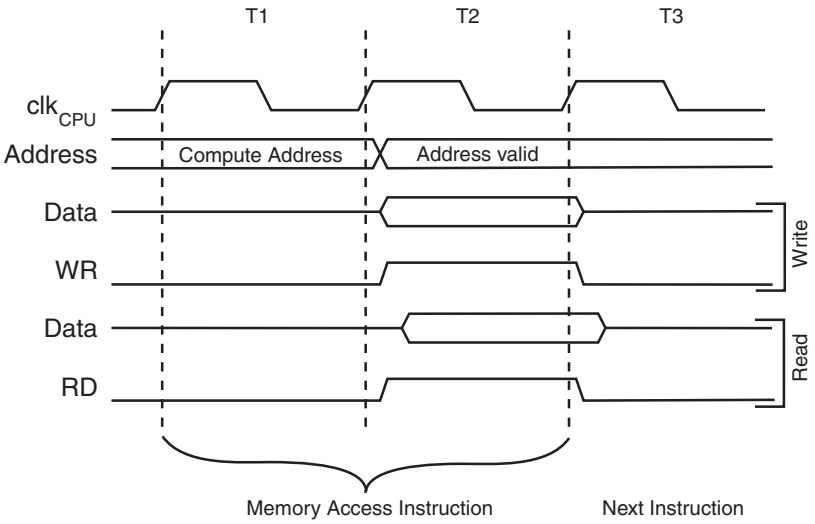
Figure 3-2. Data Memory Map



3.2.1 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 3-3.

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



3.3 EEPROM Data Memory

The Atmel® AVR® contains EEPROM memory (see “E2 size” in Table 3-1 on page 19). It is organized as a separate data space, in which single bytes can be read and written. The EEPROM has an endurance of at least 100,000 write/erase cycles in automotive range. 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.

Section 22. “Memory Programming” on page 236 contains a detailed description on EEPROM programming in SPI or Parallel Programming mode.

3.3.1 EEPROM Read/Write Access

The EEPROM Access Registers are accessible in the I/O space.

The write access times for the EEPROM are given in Table 3-2. 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 minimum for the clock frequency used. See “Preventing EEPROM Corruption” on page 24 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 “Atomic Byte Programming” on page 22 and “Split Byte Programming” on page 22 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.

3.3.2 Atomic Byte Programming

Using Atomic Byte Programming is the simplest mode. When writing a byte to the EEPROM, the user must write the address into the EEARL Register and data into EEDR Register. If the EEPMn bits are zero, writing EEPE (within four cycles after EEMPE is written) will trigger the erase/write operation. Both the erase and write cycle are done in one operation and the total programming time is given in Table 1. The EEPE bit remains set until the erase and write operations are completed. While the device is busy with programming, it is not possible to do any other EEPROM operations.

3.3.3 Split Byte Programming

It is possible to split the erase and write cycle in two different operations. This may be useful if the system requires short access time for some limited period of time (typically if the power supply voltage falls). In order to take advantage of this method, it is required that the locations to be written have been erased before the write operation. But since the erase and write operations are split, it is possible to do the erase operations when the system allows doing time-critical operations (typically after Power-up).

3.3.4 Erase

To erase a byte, the address must be written to EEAR. If the EEPMn bits are 0b01, writing the EEPE (within four cycles after EEMPE is written) will trigger the erase operation only (programming time is given in Table 1). The EEPE bit remains set until the erase operation completes. While the device is busy programming, it is not possible to do any other EEPROM operations.

3.3.5 Write

To write a location, the user must write the address into EEAR and the data into EEDR. If the EEPMn bits are 0b10, writing the EEPE (within four cycles after EEMPE is written) will trigger the write operation only (programming time is given in Table 1). The EEPE bit remains set until the write operation completes. If the location to be written has not been erased before write, the data that is stored must be considered as lost. While the device is busy with programming, it is not possible to do any other EEPROM operations.

The calibrated Oscillator is used to time the EEPROM accesses. Make sure the Oscillator frequency is within the requirements described in “OSCCAL – Oscillator Calibration Register” on page 41.

The following code examples show one assembly and one C function for erase, write, or atomic write of 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.

Assembly Code Example

```
EEPROM_write:
  ; Wait for completion of previous write
  sbic   EECR,EEPE
  rjmp   EEPROM_write
  ; Set Programming mode
  ldi    r16, (0<<EEPM1)|(0<<EEPM0)
  out    EECR, r16
  ; 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

```
void EEPROM_write(unsigned char ucAddress, unsigned char ucData)
{
  /* Wait for completion of previous write */
  while(EECR & (1<<EEPE))
    ;
  /* Set Programming mode */
  EECR = (0<<EEPM1)|(0<<EEPM0);
  /* Set up address and data registers */
  EEAR = ucAddress;
  EEDR = ucData;
  /* Write logical one to EEMPE */
  EECR |= (1<<EEMPE);
  /* Start eeprom write by setting EEPE */
  EECR |= (1<<EEPE);
}
```

The next 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

```
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

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

3.3.6 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 Atmel® AVR® 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.

3.4 I/O Memory

The I/O space definition of the Atmel AVR is shown in Section 26. “Register Summary” on page 282.

All Atmel AVR 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. Refer to the instruction set section for more details. 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 Atmel AVR is a complex microcontroller with more peripheral units than can be supported within the 64 location 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 logical one to them. Note that, unlike most other Atmel 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 to 0x1F only.

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

3.4.1 General Purpose I/O Registers

The Atmel AVR contains three General Purpose I/O Registers. These registers can be used for storing any information, and they are particularly useful for storing global variables and Status Flags.

The General Purpose I/O Registers within the address range 0x00 - 0x1F are directly bit-accessible using the SBI, CBI, SBIS, and SBIC instructions.

3.5 Register Description

3.5.1 EEARH and EEARL – EEPROM Address Register

Bit	7	6	5	4	3	2	1	0	
	-	-	-	-	-	-	-	EEAR8	EEARH
	EEAR7	EEAR6	EEAR5	EEAR4	EEAR3	EEAR2	EEAR1	EEAR0	EEARL
Bit	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R	R	R	R	R/W	
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	X	
Initial Value	X	X	X	X	X	X	X	X	