Introduction to AVR

Atmel AVR Microcontroller

AVR Key Features

- High Performance 8-Bit MCU
- RISC Architecture
  - 32 Registers
  - 2-Address Instructions
  - Single Cycle Execution
- Low Power
- Large linear address spaces
- Efficient C Language Code Density
- On-chip in-system programmable memories

RISC Performance with CISC Code Density

Breaking Traditions

Code Density

AVR Block Diagram

<table>
<thead>
<tr>
<th>Traditional CISC</th>
<th>Traditional RISC</th>
<th>Line of Compromise</th>
<th>Code Density</th>
<th>Speed</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Traditional</td>
<td>AVR</td>
<td></td>
</tr>
</tbody>
</table>

AVR Block Diagram:

- 3-wire IN/OUT
- Flash
- 32 General Purpose Registers
- Program Counter
- EEPROM
- I/O Ports
- Interrupts
- Timer Counters
The Three AVR Families

Memory Density

- Fully Compatible
- (1KB - 128KB)
- (1KB - 8KB)
- (1KB - 2KB)

tinyAVR Products

<table>
<thead>
<tr>
<th>tiny11</th>
<th>tiny12</th>
<th>tiny15</th>
<th>tiny28</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pins</td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>Flash</td>
<td>1 KB</td>
<td>1 KB</td>
<td>1 KB</td>
</tr>
<tr>
<td>EEPROM</td>
<td>-</td>
<td>64 B</td>
<td>64 B</td>
</tr>
<tr>
<td>PWM</td>
<td>-</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ADC</td>
<td>-</td>
<td>4@10-bit</td>
<td>-</td>
</tr>
<tr>
<td>Samples</td>
<td>Now</td>
<td>Now</td>
<td>Now</td>
</tr>
<tr>
<td>Production</td>
<td>Now</td>
<td>Now</td>
<td>Now</td>
</tr>
</tbody>
</table>

AVR Products

<table>
<thead>
<tr>
<th>S1200</th>
<th>S2323</th>
<th>S2343</th>
<th>S2313</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pins</td>
<td>20</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>Flash</td>
<td>1 KB</td>
<td>2 KB</td>
<td>2 KB</td>
</tr>
<tr>
<td>SRAM</td>
<td>-</td>
<td>128 B</td>
<td>128 B</td>
</tr>
<tr>
<td>EEPROM</td>
<td>64 B</td>
<td>128 B</td>
<td>128 B</td>
</tr>
<tr>
<td>UART</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>PWM</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Samples</td>
<td>Now</td>
<td>Now</td>
<td>Now</td>
</tr>
<tr>
<td>Production</td>
<td>Now</td>
<td>Now</td>
<td>Now</td>
</tr>
</tbody>
</table>

AVR Products

<table>
<thead>
<tr>
<th>S4433</th>
<th>S8515</th>
<th>VC8534</th>
<th>S8535</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pins</td>
<td>28/32</td>
<td>40/44</td>
<td>48</td>
</tr>
<tr>
<td>Flash</td>
<td>4 KB</td>
<td>8 KB</td>
<td>8 KB</td>
</tr>
<tr>
<td>SRAM</td>
<td>128 B</td>
<td>512 B</td>
<td>256 B</td>
</tr>
<tr>
<td>EEPROM</td>
<td>256 B</td>
<td>512 B</td>
<td>512 B</td>
</tr>
<tr>
<td>UART</td>
<td>1</td>
<td>1</td>
<td>-</td>
</tr>
<tr>
<td>PWM</td>
<td>1</td>
<td>2</td>
<td>-</td>
</tr>
<tr>
<td>ADC</td>
<td>6@10-bit</td>
<td>-</td>
<td>6@10-bit</td>
</tr>
<tr>
<td>RTC</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Samples</td>
<td>Now</td>
<td>Now</td>
<td>Now</td>
</tr>
<tr>
<td>Production</td>
<td>Now</td>
<td>Now</td>
<td>Now</td>
</tr>
</tbody>
</table>
### megaAVR Products

<table>
<thead>
<tr>
<th></th>
<th>mega161</th>
<th>mega163</th>
<th>mega32</th>
<th>mega103</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pins</td>
<td>40/44</td>
<td>40/44</td>
<td>40/44</td>
<td>64</td>
</tr>
<tr>
<td>Flash</td>
<td>16 KB</td>
<td>16 KB</td>
<td>32 KB</td>
<td>128 KB</td>
</tr>
<tr>
<td>SRAM</td>
<td>1 KB</td>
<td>1 KB</td>
<td>2 KB</td>
<td>4 KB</td>
</tr>
<tr>
<td>EEPROM</td>
<td>512 B</td>
<td>512 B</td>
<td>1 KB</td>
<td>2 KB</td>
</tr>
<tr>
<td>I/O</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>TWI</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>-</td>
</tr>
<tr>
<td>PWM</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>-</td>
</tr>
<tr>
<td>ADC</td>
<td>-</td>
<td>8@10-bit</td>
<td>8@10-bit</td>
<td>8@10-bit</td>
</tr>
<tr>
<td>RTC</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>JTAG/OCD</td>
<td>-</td>
<td>Yes</td>
<td>Yes</td>
<td>-</td>
</tr>
<tr>
<td>Self Program</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>-</td>
</tr>
<tr>
<td>HW MULT</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>-</td>
</tr>
<tr>
<td>Brown Out</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>-</td>
</tr>
<tr>
<td>Samples</td>
<td>Now</td>
<td>Now</td>
<td>Now</td>
<td>Now</td>
</tr>
<tr>
<td>Production</td>
<td>Now</td>
<td>Now</td>
<td>Now</td>
<td>Now</td>
</tr>
</tbody>
</table>

### megaAVR Products

<table>
<thead>
<tr>
<th></th>
<th>mega8</th>
<th>mega16</th>
<th>mega32</th>
<th>mega64</th>
<th>mega128</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pins</td>
<td>28/32</td>
<td>40/44</td>
<td>40/44</td>
<td>64</td>
<td>64</td>
</tr>
<tr>
<td>Flash</td>
<td>8 KB</td>
<td>16 KB</td>
<td>32 KB</td>
<td>64 KB</td>
<td>128 KB</td>
</tr>
<tr>
<td>SRAM</td>
<td>1 KB</td>
<td>1 KB</td>
<td>2 KB</td>
<td>4 KB</td>
<td></td>
</tr>
<tr>
<td>EEPROM</td>
<td>512 B</td>
<td>512 B</td>
<td>1 KB</td>
<td>2 KB</td>
<td>4 KB</td>
</tr>
<tr>
<td>I/O</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>TWI</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>PWM</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>ADC</td>
<td>-</td>
<td>8@10-bit</td>
<td>8@10-bit</td>
<td>8@10-bit</td>
<td></td>
</tr>
<tr>
<td>RTC</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>JTAG/OCD</td>
<td>-</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Self Program</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>HW MULT</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Brown Out</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Samples</td>
<td>Now</td>
<td>Q1/02</td>
<td>Q2/02</td>
<td>Q2/02</td>
<td>Q1/02</td>
</tr>
<tr>
<td>Production</td>
<td>Q1/02</td>
<td>Q2/02</td>
<td>Q2/02</td>
<td>Q2/02</td>
<td>Q1/02</td>
</tr>
</tbody>
</table>

### ATtiny12

- 8 pin package
- 1 KBytes ISP Flash
- 64 Bytes ISP EEPROM
- Up to 6 programmable I/O lines
- 8 Bit Timer/Counter
- 1 External Interrupt
- Interrupt and wake up on any pin change
- Analog Comparator
- Internal Accurate Oscillator with calibratable frequency

### Typical Applications, ATtiny11/12

- Secure EEPROM
- Keyless Entry for Car Alarm
- Fire Detector
- Toys
- DIP switch replacement
- Protocol Converter
- Motor Control
- Replacing External Logic
- Signal Processing Applications
- Data Logger
- Coprocessor
ATtiny28

- 28/32 pin packages
- 2 KBytes Flash
- 11 programmable I/O lines, 8 dedicated input lines and 1 dedicated output line
- 8 Bit Timer/Counter
- 2 External Interrupts
- Interrupt and wake up on any pin change
- Built-in High-current LED Driver with Programmable Modulation
- Analog Comparator
- Internal Accurate Oscillator with calibratable frequency
- Operates down to 1.8V

Typical Applications, ATtiny28

- Remote Control
- Keyboard Controller
- I/O Controller
- Communication Equipment
- Low-cost/High pin count Applications

ATmega16(L)

- 40/44 pin packages
- 16 KBytes ISP Flash, Self Programmable
- 512 Bytes ISP EEPROM
- 1 KBytes SRAM
- Full Duplex UART
- SPI – Serial Interface
- TWI – Serial Interface
- 8- and 16-bits Timer/Counters with PWM
- 2 External Interrupts
- 10-bit ADC with 8 Multiplexed Inputs
- RTC with Separate 32 kHz Oscillator
- Analog Comparator
- JTAG Interface with On-Chip Debugger

Typical Applications, ATmega16(L)

- Smart Battery
- Advanced Battery Charger
- Power Meter
- Temperature Logger
- Voltage Logger
- Tension Control
- Touch Screen Sensor
- Metering Applications
- UPS
- 3 Phase Motor Controller
- Industrial Control
- Power Management
**ATmega32(L)**

- 40/44 pin packages
- 32 KBytes ISP Flash, Self Programmable
- 1 KBytes ISP EEPROM
- 2 KBytes SRAM
- Full Duplex UART
- SPI – Serial Interface
- TWI – Serial Interface
- 8- and 16-bits Timer/Counters with PWM
- 2 External Interrupts
- 10-bit ADC with 8 Multiplexed Inputs
- RTC with Separate 32 kHz Oscillator
- Analog Comparator
- JTAG Interface with On-Chip Debugger

**Typical Applications, ATmega32(L)**

- Smart Battery
- Advanced Battery Charger
- Power Meter
- Temperature Logger
- Voltage Logger
- Tension Control
- Touch Screen Sensor
- Metering Applications
- UPS
- 3 Phase Motor Controller
- Industrial Control
- Power Management

**ATmega128(L)**

- 64 pin package (48 I/O, 16 Special Function)
- 128 KBytes ISP Flash, Self Programmable
- 4 KBytes SRAM
- 4 KBytes ISP EEPROM
- Dual Full Duplex UARTs
- SPI – Serial Interface
- TWI – Serial Interface
- 8- and 16-bits Timer/Counters with PWM
- 2 External Interrupts
- 10-bit ADC with 8 Multiplexed Inputs
- RTC with Separate 32 kHz Oscillator
- Analog Comparator
- JTAG Interface with On-Chip Debugger

**Typical Applications, ATmega128(L)**

- Analog Telephone
- Blood Oxymeter
- Sensor Applications
- Automobile Applications
- Paper Feeder
- Telecom Applications
- 3-phase Motor Control
- GPS
- Industrial Control
AVR for ASIC

- RTL soft-core enables a variety of processes
- RTL for Standard I/O modules
- AVR ASIC ICE available with reference designs
- On-Chip Debug Support
- Testing using scan chains
- AVR ASIC handbook

A C Code Example

The following example illustrates how the AVR benefits in terms of:

- Code Size
- Throughput
- Power Consumption

A Small C Function

```c
/* Return the maximum value of a table of 16 integers */
int max(int *array)
{
    char a;
    int maximum=-32768;
    for (a=0;a<16;a++)
        if (array[a]>maximum)
            maximum=array[a];
    return (maximum);
}
```

AVR Assembly output

Code Size: 46 Bytes, Execution time: 335 cycles
**C51 Assembly Output**

Initial code snippet:

```
MOV     R6, maximum
MOV     R7, maximum + 01H
```

Later code snippet:

```
RET
```

Code Size: 112 Bytes, Execution time: 9384 cycles

**HC11 Assembly output**

```
; 7. for (a=0;a<16;a++)
TSX
LDD     #-32768
STD     1, X
CLRF (?a_maxnum+2)& (0+127)
movlw 128
movwf (?a_maxnum+3)& (0+127)
; MAX_MAIN.C: 8: {
; MAX_MAIN.C: 9: if (array[a]>maximum)
bcf 3, 5
movf (?a_maxnum+1)& (0+127), w
addwf (?a_maxnum+1)& (0+127), w
addwf ?a_maxnum& (0+127), w
movwf 4
movf 0, w
movwf btemp
incf 4
movf 0, w
movwf btemp+1
movf (?a_maxnum+3)& (0+127), w
xorlw 128
movwf btemp+2
movf btemp+1, w
xorlw 128
subwf btemp+2, w
btfss 3, 2
goto u15
movf btemp, w
```

Code Size: 57 Bytes, Execution time: 5244 cycles

**PIC16C74 Assembly output**

```
bcf 3, 5
movwf ?a_maxnum& (0+127)
clrf (?a_maxnum+2)& (0+127)
movlw 128
movwf (?a_maxnum+3)& (0+127)
; MAX_MAIN.C: 7: for (a=0;a<16;a++)
clrf (?a_maxnum+1)& (0+127)
l2
; MAX_MAIN.C: 8: {
; MAX_MAIN.C: 9: if (array[a]>maximum)
bcf 3, 5
movf (?a_maxnum+1)& (0+127), w
addwf (?a_maxnum+1)& (0+127), w
addwf ?a_maxnum& (0+127), w
movwf 4
movf 0, w
movwf btemp
incf 4
movf 0, w
movwf (?a_maxnum+2)& (0+127)
bcf 3, 5
movf (?a_maxnum+1)& (0+127), w
addwf (?a_maxnum+1)& (0+127), w
addwf ?a_maxnum& (0+127), w
movwf 4
movf 0, w
movwf (?a_maxnum+2)& (0+127)
incf (?a_maxnum+1)& (0+127)
movlw 16
subwf (?a_maxnum+1)& (0+127), w
btfss 3, 0
goto l5
bcf 3, 5
movf (?a_maxnum+3)& (0+127), w
movwf btemp+1
movf (?a_maxnum+2)& (0+127), w
movwf btemp
```

Code Size: 87 Bytes, Execution time: 2492 cycles

**Performance Comparison**

- **AT90S8515** @ 8 MHz
- **80C51** @ 24 MHz
- **68HC11A8** @ 12 MHz
- **PIC16C74** @ 20 MHz

<table>
<thead>
<tr>
<th>Processor</th>
<th>Code Size (Bytes)</th>
<th>Function Execution Time (us)</th>
<th>Current Consumption (mA)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AT90S8515</td>
<td>46 (1)</td>
<td>42 (1)</td>
<td>11 (1)</td>
</tr>
<tr>
<td>80C51</td>
<td>112 (2.4)</td>
<td>391 (9)</td>
<td>16 (1.5)</td>
</tr>
<tr>
<td>68HC11A8</td>
<td>57 (1.2)</td>
<td>437 (10)</td>
<td>27 (2.5)</td>
</tr>
<tr>
<td>PIC16C74</td>
<td>87 (1.9)</td>
<td>125 (3)</td>
<td>13.5 (1.2)</td>
</tr>
</tbody>
</table>

Normalized figures given in parentheses
Some conclusions on this case

- The C51 would have to run at 224 MHz to match the 8 MHz AVR.
- The HC11 is quite code efficient, but delivers only one 16th of the processing power at more than twice the current consumption.
- The PIC is a fast microcontroller, but the AVR delivers more than 3.5 times higher throughput per mW.

What made the AVR do better?

- Excellent support for 16-bit arithmetic operations.
- A lot of registers which eliminate move to and from SRAM.
- Single Cycle execution.

High Level Languages

- Increased importance for Microcontrollers
  - Time to market
  - Simplified maintenance
  - Portability
  - Learning time
  - Reusability
  - Libraries
- Potential drawbacks
  - Increased codesize
  - Decreased speed

AVR Influenced by IAR

- Architecture and Instruction Set co-designed with IAR systems through several iterations:
  - Compiler development project initiated before architecture and instruction set frozen.
  - Compiler expert’s advice implemented in hardware.
  - Potential HLL bottlenecks identified and removed.
C-like Addressing Modes (1)

Auto Increment/Decrement:

C Source:

```c
unsigned char *var1, *var2;
*var1++ = *--var2;
```

Generated code:

```assembly
LD R16,-X
ST Z+,R16
```

C-like Addressing Modes (2)

Indirect with Displacement:

- Efficient for accessing arrays and structs
- Autos placed on Software Stack

```
SRAM
<table>
<thead>
<tr>
<th>x_min</th>
<th>x_max</th>
<th>y_min</th>
<th>y_max</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z+2</td>
<td>Z+4</td>
<td>Z+6</td>
<td></td>
</tr>
</tbody>
</table>

Struct square

```c
struct square {
    int x_min;
    int x_max;
    int y_min;
    int y_max;
} my_square;
```

Four Memory Pointers

- Code efficient memory to memory copy
- Pointer reloading is minimized
- Separate stacks for return addresses and local variables

16 and 32 bit support

- Carry instructions
  - Addition, Subtraction and Comparison
  - Register with register or immediate
  - Zero flag propagation

```assembly
SUB   R16,R24  SUBI   R16,1
SBC   R17,R25  SBCI   R17,0
```

All branches can be made based on last result

- Direct 16 bit instructions
  - Addition and Subtraction of small immediates
  - Pointer arithmetics
Instruction Set General Features

- Most Instructions are 16 Bits Wide
- Most Instructions are Executed in One Clock Cycle

Instruction Classes

- Arithmetic/Logic Instructions
- Data Transfer Instructions
- Program Control Instructions
- Bit Set/Test Instructions

Arithmetic/Logical Instructions

- Add / Increment
- Subtract / Decrement
- Compare
- Multiply
- Logical (AND, OR, XOR)
- Shift / Rotate

Add Instructions

- “ADD” Add Two Registers
- “ADC” Add Two Registers and Carry
- “INC” Increment a Register
- “ADIW” Add Immediate to Word *

* Works on the 4 Uppermost Register Pairs
Subtract Instructions

```
“SUB” Subtract Two Registers
“SBC” Subtract with Carry Two Registers
“SUBI” Subtract Immediate from Register*
“SBCI” Subtract with Carry Immediate from Register*
“DEC” Decrement Register
“SBIW “ Subtract Immediate From Word**
```

* Works on Registers R16 - R31
** Works on the 4 Uppermost Register

Direct Register - ALU Connection

Register operations take ONE clock pulse on the EXTERNAL clock input

Executing “subi r16,k”

Compare Instructions

```
“CP” Compare Two Registers
“CPC” Compare with Carry Two Registers
“CPI” Compare Register and Immediate*
“CPSE” Compare Two Registers and Skip Next Instruction if Equal
```

* Works on Registers R16 - R31
16/32-Bit Data Support

- The AVR Supports Code and Time Efficient 16 and 32-bit Arithmetic Through the Following:
  - Single Cycle Execution
  - A Register File That Holds Several 16/32-Bit Values
  - Carry and Zero Flag Propagation on Subtract and Compare

Subtract Two 16-Bit Values

**Without Zero Flag Propagation**
R1:R0 - R3:R2 ($E104 - $E101)

<table>
<thead>
<tr>
<th>R1</th>
<th>R0</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1</td>
<td>04</td>
<td>X</td>
</tr>
</tbody>
</table>

\[ \text{sub r0,r2} \]
\[ \text{sbc r1,r3} \]

Wrong!

**With Zero Flag Propagation**
R1:R0 - R3:R2 ($E104 - $E101)

\[ \text{sub r0,r2} \]
\[ \text{sbc r1,r3} \]

Correct!

Compare Two 32-Bit Values

- Example: Compare R3:R2:R1:R0 and R7:R6:R5:R4
  - \[ \text{cp r0,r4} \]
  - \[ \text{cpc r1,r5} \]
  - \[ \text{cpc r2,r6} \]
  - \[ \text{cpc r3,r7} \]

- At the end, the Status Register Indicates Equal, Higher, Lower, Greater (signed), Less Than (signed).
- Besides, it takes only 4 instructions and 4 clock cycles to do it...
Multiply instructions

“MUL” 8x8 -> 16 (UxU)
“MULS” 8x8 -> 16 (SxS)*
“MULSU” 8x8 -> 16 (SxU)**
“FMUL” 1.7x1.7 -> 1.15 (UxU)**
“FMULS” 1.7x1.7 -> 1.15 (SxS)**
“FMULSU” 1.7x1.7 -> 1.15 (SxU)**

* Works on Registers R16 - R31
** Works on Registers R16-R23

The result is present in R1:R0
All multiplication instructions are 2 cycles
Only available in Enhanced Core

Logical Instructions

“AND” Logical AND Two Registers
“ANDI” Logical AND Immediate and Register *
“OR” Logical OR Two Registers
“ORI” Logical OR Immediate and Register *
“EOR” Logical XOR Two Registers

* Works on Registers R16 - R31

Shift / Rotate Instructions

“LSL” Logical Shift Left
“LSR” Logical Shift Right
“ROL” Rotate Left Through Carry
“ROR” Rotate Right Through Carry
“ASR” Arithmetic Shift Right

Shift / Rotate Operations

<table>
<thead>
<tr>
<th>Shift/Rotation</th>
<th>MSB</th>
<th>Register</th>
<th>LSB</th>
<th>Carry</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>LSR</strong></td>
<td>0</td>
<td></td>
<td>0</td>
<td></td>
</tr>
<tr>
<td><strong>ROR</strong></td>
<td>0</td>
<td></td>
<td>0</td>
<td></td>
</tr>
<tr>
<td><strong>ASR</strong></td>
<td>0</td>
<td></td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
Data Transfer Instruction Types

<table>
<thead>
<tr>
<th>Instruction Type</th>
<th># of Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data SRAM &lt;−&gt; Register File</td>
<td>(2)</td>
</tr>
<tr>
<td>Program Memory −&gt; Register File</td>
<td>(1/3)</td>
</tr>
<tr>
<td>I/O Memory &lt;−&gt; Register File</td>
<td>(1)</td>
</tr>
<tr>
<td>Register File &lt;−&gt; Register File</td>
<td>(1)</td>
</tr>
</tbody>
</table>

The Five Memory Areas

- General Purpose Register File: 32 Bytes
- Flash Program Memory: ≤ 8 MB
- SRAM Data Memory: ≤ 16 MB
- I/O Memory: ≤ 64 (224) Bytes
- EEPROM Data Memory: ≤ 16 MB

Memory Interconnections

AVR Register File

- R0
- R1
- R2
- R3
- R26
- R27
- R28
- R29
- R30
- R31

- X Pointer
- Y Pointer
- Z Pointer
Data SRAM → Register File

- "LD Rd,<PTR>" Load Indirect
- "LD Rd,<PTR>+" Load Indirect with Post-Increment
- "LD Rd,-<PTR>" Load Indirect with Pre-Decrement
- "LDD Rd,<PTR>+q" Load Indirect with Displacement (0-63)*
- "LDS Rd,<ADDR>" Load Direct from SRAM (16-bit ADDR)**

* PTR = X, Y or Z
** Parts with SRAM > 64k bytes, memory page selected using the RAMPZ register

Data SRAM ← Register File

- "ST <PTR>,Rd" Store Indirect
- "ST <PTR>+,Rd" Store Indirect with Post-Increment
- "ST -<PTR>,Rd" Store Indirect with Pre-Decrement
- "STD <PTR>+q,Rd" Store Indirect with Displacement (0-63)*
- "STS <ADDR>,Rd" Store Direct from SRAM (16-bit ADDR)**

* PTR = X, Y or Z
** Parts with SRAM > 64k bytes, memory page selected using the RAMPZ register

Data Transfer RF <-> SRAM Stack

- "PUSH" PUSH a register on the stack
- "POP" POP a register from the stack

Data Transfer Program Memory → RF

- "LDI" Load a Register with an Immediate Value (1 Cycle)*
- "(E)LPM" Transfer a byte from Program Memory@Z to R0 (3 Cycles)
- "(E)LPM Rd,Z" Transfer a byte from Program Memory@Z to Rd (3 Cycles)**
- "(E)LPM Rd,Z+" As above but with post-increment of the Z pointer**
- "SPM" Write to the Program Memory**

* Works on R16 - R31
** Enhanced Core
ELPM instructions use RAMPZ Register to reach >64KBytes
The (E)LPM Instructions

- Z Pointer
- MSB
- LSB
- Selects Program Memory Address
- Selects Low/High Byte
- The byte pointed to by Z is transferred to R0/Rd
- The ELPM instructions extend the Z pointer with RAMPZ

SPM Instruction

- Can be used to write program memory and to read and write fuse bits.
- Operation is dependent on value in I/O register. See datasheet for details.
- To update the program memory:
  - Write the data to a page buffer
  - Z points into Flash
  - R1:R0 contains data
  - Transfer the buffer to the Flash

Register File <--> I/O Memory Transfer

- "OUT" Transfer a Byte from RF to I/O
- "IN" Transfer a Byte from I/O to RF

Register File --> Register File

- "MOV" Copy a Register to another Register
- "MOVW" Copy a Register pair to another Register pair. Aligned.
Flow Control

- Unconditional Jumps
- Conditional Branches
- Subroutine Call and Returns

Unconditional Jump Instructions

“RJMP” Relative Jump *
“JMP” Absolute Jump **
“IJMP” Indirect Jump to Program Address Z***

* Reaches ±2K instructions from current program location. Reaches all locations for devices up to 8KBytes (wrapping)
** 4-Byte Instruction
*** An EIJMP instruction that uses the EIND register exists to extend the IJMP reach for devices with 256KBytes program memory or more.

Instruction Fetch/Execution

Instructions are Sequential

Fetch Execute

T1 T2 T3

INSTR n+1 INSTR n+2 INSTR n+3

One Instruction is a Jump

Fetch Execute

T1 T2 T3

INSTR m+1 INSTR m INSTR m

INSTR m

Reject
Conditional Branches (Flag Set)

“BREQ” Branch if Equal
“BRSH” Branch if Same or Higher
“BRGE” Branch if Greater or Equal (Signed)
“BRHS” Branch if Half Carry Set
“BRCS” Branch if Carry Set
“BRMI” Branch if Minus
“BRVS” Branch if Overflow Flag Set
“BRTS” Branch if T Flag Set
“BRIE” Branch if Interrupt Enabled

The Status Register - SREG

- **I** Interrupt Enable
- **T** T Flag
- **H** Half Carry
- **S** Signed Flag
- **V** Overflow Flag
- **N** Negative Flag
- **Z** Zero Flag
- **C** Carry Flag

Branch on SREG Settings

- **BRID** Branches if Bit Clear
- **BRTC** Branches if Bit Set
- **BRHC** Branch if Half Carry Set
- **BRGE** Branch if Greater
- **BRVC** Branch if Signed Overflow
- **BRPL** Branch if Positive
- **BRNE** Branch if Not Equal
- **BRSH, BRCC** Branch if Interrupt Enabled

Subroutine Call and Return

- **RCALL** Relative Subroutine Call *
- **CALL** Absolute Subroutine Call **
- **ICALL** Indirect Subroutine Call to Routine @Z***
- **RET** Return from Subroutine
- **RETI** Return from Interrupt Routine

* Reaches ± 2K instructions from current program location. Reaches all locations for devices up to 8KBytes (wrapping)
** 4-Byte Instruction
*** An EICALL instruction that use the EIND register exists to extend the ICALL reach for devices with 256KBytes program memory or more.
Bit Set and Bit Test Instructions

- **“SBR”** Set Bit(s) in Register *
- **“SBI”** Set Bit in I/O Register **
- **“SBRS”** Skip if Bit in Register Set
- **“SBIS”** Skip if Bit in I/O Register Set **

* Works on Registers R16 - R31
** Works on I/O Addresses $00 - $1F

Bit Clear and Bit Test Instructions

- **“CBR”** Clear Bit(s) in Register *
- **“CBI”** Clear Bit in I/O Register **
- **“SBRC”** Skip if Bit in Register Clear
- **“SBIC”** Skip if Bit in I/O Register Clear **

* Works on Registers R16 - R31
** Works on I/O Addresses $00 - $1F

Interrupt System Features

- Short Response Time
  – 4 Clock Cycles + RJMP to interrupt handler
- Automatic Interrupt Flag Clearing
- Automatic Disable of Other Interrupts
  Inside the Interrupt Routine
- Status flag must be preserved manually

Interrupt Vector Example

```
$0000  rjmp  RESET  ;Reset Vector
$0001  rjmp  IRQ0  ;IRQ0 Vector
$0002  rjmp  IRQ1  ;IRQ1 Vector

IRQ0:  ;IRQ0 Routine
  $0003  <opcode>
  $0004  <opcode>
  ...
  $xxxx  <opcode>
  $xxxx  reti  ;Ret. from Int.
```
Executing an Interrupt

Interrupt in the Main Program

```
$0123 mov r16, r0  
$0124 add r20, r0  
$0125 adc r21, r1  
Interrupt $0126 lsl r0  
$0127 clc
```

- “lsl r0” is completed
- $0127 (return address) is stored on the stack
- The Interrupt Flag is Cleared
- The I-bit in SREG is Cleared
- The Program Jumps to the IRQ Vector

4 Clock Cycles

Executing an Interrupt

The Interrupt Vector and Routine

```
$0000 RJMP RESET  
$0001 RJMP IRQ0  
$0002 RJMP IRQ1  
IRQ0:
$0003 in r0, PINB  
$0004 out PORTD, r0  
Return $0005 reti
```

- The I-bit in SREG is Set
- $0127 is read from the return stack
- Program execution resumes from $0127

4 Clock Cycles

Sleep Modes

- Idle Mode
- Power Down Mode
- Power Save Mode (Parts with RTC)
- ADC Noise Reduction Mode
- Stand-by Mode
- Extended Stand-by

Idle Mode

- CPU is stopped
- XTAL oscillator continues to operate
- All peripherals continue to operate
- Reset, or any enabled interrupt can wake up the MCU
Power Down Mode

- CPU is stopped
- XTAL oscillator is stopped
- Most peripherals are stopped
- TWI Address Watch continue to operate
- The Watchdog can be enabled
- Reset, TWI Address Watch interrupt and external level interrupt can wake up the MCU
- Typical power consumption: 100 nA

* WDT Not Enabled

Power Save Mode

- Applies to devices with Asynchronous 32 kHz Timer (RTC)
- Similar to Power Down Mode, but Timer Oscillator is kept running
- RTC Interrupt wakes up the device
- Typical Power Consumption: 5mA @ 5V

ADC Noise Reduction Mode

- CPU is Stopped
- ADC, RTC Timer/Counter, TWI Address Watch and External Interrupt Operate
- The Watchdog can be Enabled.
- ADC Conversion Complete, Reset, TWI Address Match Interrupt, RTC Interrupt and External Interrupt Can Wake up the MCU
- Improves Noise environment for ADC
- ADC conversion starts automatically when the sleep mode is entered

Stand-by Mode

- Similar to Power Down Mode, but the Oscillator is kept running
- Wakes up the device in 6 clock cycles only
Extended Stand-by Mode

- Similar to Power Save Mode, but the Oscillator is kept running
- Wakes up the device in 6 clock cycles only

Wake-Up from Sleep Mode

- **RESET:** The MCU Starts Execution from the Reset Vector
- **INTERRUPT:** The MCU Enters the Interrupt Routine, Runs it and Resumes Execution from the Instruction following “SLEEP”.

I/O Ports General Features

- Push-Pull Drivers
- High Current Drive (sinks up to 40 mA)
- Pin-wise Controlled Pull-Up Resistors
- Pin-wise Controlled Data Direction
- Fully Synchronized Inputs
- Three Control/Status Bits per Bit/Pin
- Real Read-Modify-Write

3 Control/Status Bits per Pin

- **DDx** Data Direction Control Bit
- **PORTx** Output Data or Pull-Up Control Bit
- **PINx** Pin Level Bit
**Default Configuration**

- **Direction:** INPUT
- **Pull-Up:** OFF

**Switch On Pull-Up**

- **Direction:** INPUT
- **Pull-Up:** ON

**Port is Output**

- **Direction:** OUTPUT
- **Pull-Up:** OFF

**General T/C Features**

- Various Clock Prescaling Options
- Can Run at Undivided XTAL Frequency (High Resolution)
- Can be Set to Any Value at Any Time
- Can be Clocked Externally by Signals with Transition Periods down to XTAL/2
- Can be Clocked Externally on both Rising and Falling Edge
- The features vary from device to device, see datasheets for details
8 Bit Timer/Counter

- Prescaler
- Overflow Interrupt
- Output Compare Function with Interrupt
- Real Time Counter with 32 kHz oscillator
- PWM

16 Bit Timer/Counter

- Prescaler
- Overflow Interrupt
- Output Compare Function with Interrupt
- Input Capture with Interrupt and Noise Canceler
- PWM

8 Bit Timer Block Diagram

16 Bit T/C Block Diagram
Output Compare Features

- Compare match can control an external pin (Rise, Fall or Toggle) even if the Interrupt is disabled.
- As an option, the timer can be automatically cleared when a compare match occurs.

Input Capture Features

- Capture Can Trigger on Rising or Falling Edge (Optional)
- The Input Capture Noise Canceler will not trigger the Capture until 4 Subsequent samples of the same value are seen.
  The Noise Canceler is Optional

PWM Features

- Selectable 8, 9 or 10-Bit Resolution.
- Frequency @ 10 MHz (8-bit): 19 KHz
- Centered Pulses
- Glitch-Free Pulse Width Change
- Selectable Polarity

PWM Operation

- Diagram showing PWM waveform with compare values, timer values, and PWM outputs.
Advanced Timer Features

- Available in selected parts
- Internal PLL for high speed operation
- Dual speed PWM
- Variable Top Value
  - High Frequency
  - High Resolution
  - Variable Frequency

Analog Comparator Features

- Comparator has its own Interrupt on Output Transitions
- Interrupt has selectable trigger on Rise, Fall or Toggle
- The Comparator Output can be Connected to the Input Capture of Timer/Counter1
  - Enables Pulse-Width Measurement of Analog Signals
  - Enables Easy Implementation of Dual Slope ADC

ADC Features

- Successive approximation w/ sample & hold
- Up to 8 single ended channels and up to 7 differential channels
- 10-Bit resolution
- Gain Stage
- Configurable conversion time
- Accuracy:
  - down to 65 ms conversion time: 10bits ± 0.5LSB
  - down to 12 ms conversion time: 8bits ± 0.5LSB
- Free-run and single conversion modes
- Interrupt on conversion complete
- CPU Turn-Off Noise Reduction

A/D converter

- ADC Control and Status Register (ADCSR)
- ADC data register (ADCH, ADCL)
- Successive approximation logic
- 10-bit DAC
- ADC multiplexer select (ADMUX)
- 8-Channel Mux
Watchdog

- Clocked from Internal 1 MHz RC Oscillator
- Time-Out Adjustable 16 - 2048 ms.
- Watchdog Timer Reset is done by executing the "WDR" instruction

UART Features

- Full Duplex
- 8 or 9 Data Bits
- Framing Error Detection
- False Start Bit Detection
- Noise Canceling
- High BAUD Rates at low clock frequencies
  E.g. 115,200 Baud at 1.8432 MHz
- Can run at Practically any Baud Rate
- Three Interrupts with Separate Vectors

USART Features

- Synchronous high-speed mode
- Possibility for 5, 6, 7, 8 or 9 bit transmission
- Parity check / generation
- One or two stop-bits
- Extra FIFO buffer
- Overrun situation will not occur before start-condition detected

TWI – Two Wire Interface

- Two wire serial communication protocol
- Fully I2C compliant
  - I2C Fast Mode support (up to 400kbit/S)
  - Wake from Power Down Sleep mode on address recognition, both own slave address and General Call.
  - Both Master and Slave implementation
Internal Brown-Out Detection

- Flexible BOD levels
  - BOD levels dependent on device operating range
- Extremely fast detection (24 us or 7 us)
- Low power consumption: 25 uA typical.
- BOD is optional (BODEN fuse)
- BOD status bit (MCUSR) after reset

Internal Calibrated RC oscillator

- Internal high accuracy RC oscillator
- Calibrated within 2% accuracy
- Calibrated during Atmel’s production test
- Frequency offset programmed into the AVR’s signature bytes

Traditional In-System Programming

- In-System Programmable FLASH, EEPROM, Fuses and Lock Bits
- Programmable at all Frequencies
- Programmable at all VCCs above 2.7V
- Only four pins + Ground required

Redefining ISP - Self Programming

- The AVR reprograms itself without any external components
- In application re-programmable through any communication interface
  - Use the existing communication port
- Critical functions still operating
  - Device is still running during programming
Self Programming

- Dual memory areas
  - Application section
  - Boot section (optional)
- Read data from
  - Any communication interface
  - Application section
  - Boot section
- Write it to a page buffer
- Transfer the buffer to the Flash page in Application or Boot section

Self Programming Features

- Flexible Boot Block Sizes (256 B - 2 KB)
  - Allows efficient encryption and CRC code
- Small sector (block) sizes, allows efficient data variables modifications
  - 128/256 Bytes sector size
- Extremely Fast Programming Time
  - 10 ms for One Sector
- Program Over the Full Temperature and Voltage Ranges

Self Programming Security

- Each memory partition has independent lock bit protection
  - Protect entire FLASH memory
  - Protect Boot Block while updating Application partition
  - Protect Application partition while updating Boot Block
  - Allow software update in entire FLASH address space where Boot Block is not used

Benefits of Self Programming

- ISP of the MCU is completely autonomous
  - No external device(s) required to reprogram the AVR microcontroller in-system
  - Application code can be updated or modified based on external conditions
- Sectored Flash provides unmatched flexibility
  - Small sections of application code can be changed instead of having to do a full update
  - Unused sectors provide additional data storage memory – perfect for calibration data
AVR JTAG Interface

- Complies to IEEE std 1149.1
  - (JTAG)
- Boundary-Scan for Efficient PCB Test
- On-Chip Debugging in Production Silicon
- ISP Programming of both Flash and EEPROM during JTAG production test

Boundary Scan

- Boundary-Scan
  - Standard for interconnection test
  - All IO pins controllable and observable from tester
- Enables interconnection testing of PCB
- Supported by all major third party vendors
- BSDL files available from Atmel

JTAG In System Programming

- The JTAG interface can be used to program the Flash and EEPROM
- Save time and production cost
  - No additional programming stage
  - Programming time independent of system clock

Development Tools

- Complete suite of development tools available
- ANSI compliant C Compilers
- Macro-Assemblers
- Linkers/Librarians
- Debuggers/Simulators
- RTOS
- In-Circuit Emulators
- Evaluation boards
- Programmers
- Design notes, Application notes and Reference designs
AVR Studio

- Integrated Development Environment for AVR
- Front end for the AVR Simulator and Emulators
- C and Assembly source level debugging
- Supports third party compilers
- Maintains Project information
- Freely available from www.atmel.com

JTAG ICE

- Controlled by AVR Studio
- Real-Time Emulation in actual silicon
  - Debug the real device at the target board
  - Talks directly to the device through
    - a 4-pin JTAG interface
- Supports
  - Program and Data breakpoints
  - Full execution control
  - Full I/O-view and Watches

AVR C-compilers

- Third parties provide a full scale of C-compilers for AVR
- Low cost to high end compilers
- Good support and coverage of AVR devices from Third party providers

GNU AVR-GCC

- The GNU ANSI C compiler for AVR
- Freeware C compiler (and assembler)
- Supports all AVRs, including tiny devices
- AVR Studio text editor supports integration of AVR-GCC
  - Compile and run the code from
    AVR Studio
    www.avrfreaks.net/AVRGCC
### Full range of development tools

- **Evaluation tools**
  - STK500 and AVR Studio
  - Total Cost $79

- **Low cost tools**
  - STK500 and AVR Studio
  - ICE200/JTAGICE
  - Imagecraft/CodeVision/GNU
  - Total cost < $500

- **High Performance tools**
  - STK500 and AVR Studio
  - ICE30/ICE10/ICEPRO
  - IAR C/C++
  - Total Cost ~ $7100

### AVR websites and mail

- **ATMEL website** [www.atmel.com](http://www.atmel.com)
  - Datasheets
  - Application Notes
  - FAQ

- **Unofficial AVR websites**
  - [www.avrfreaks.net](http://www.avrfreaks.net)
  - [www.avr-forum.com](http://www.avr-forum.com)

- **Support mail**
  - avr@atmel.com