Atmel AVR Microcontroller

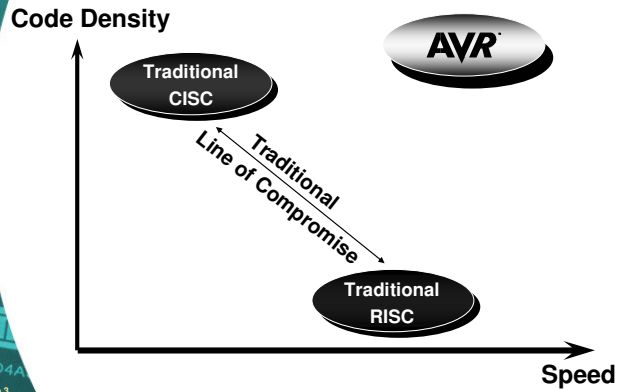
Introduction to AVR

AVR Key Features

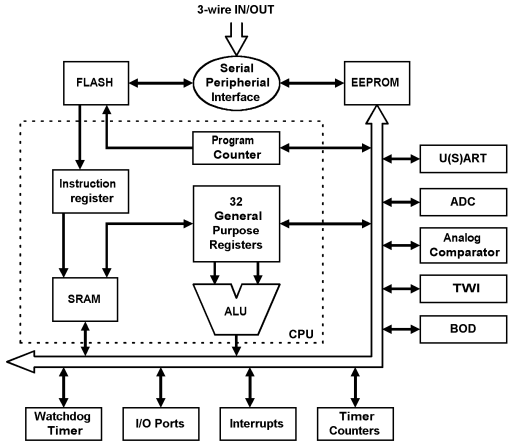
- High Performance 8-Bit MCU
- RISC Architecture
  - 32 Registers
  - 2-Address Instructions
  - Single Cycle Execution
- Low Power
- Large linear address spaces
- Efficient C Language Code Density
- On-chip in-system programmable memories

RISC Performance with CISC Code Density

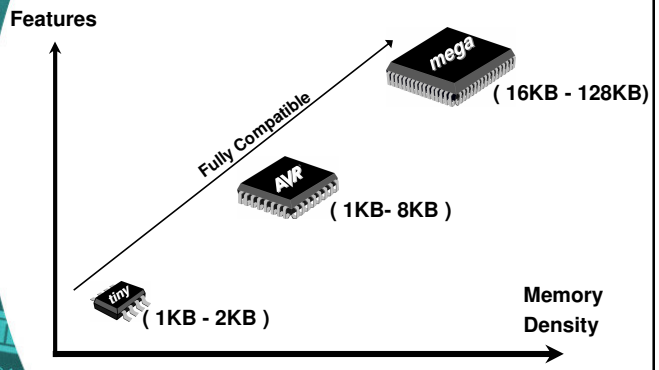
Breaking Traditions



AVR Block Diagram



The Three AVR Families



CSE466-Page 5

tinyAVR Products

	tiny11	tiny12	tiny15	tiny28
Pins	8	8	8	28/32
Flash	1 KB	1 KB	1 KB	2 KB
EEPROM	-	64 B	64 B	-
PWMs	-	-	1	1
ADC	-	-	4@10-bit	-
Samples	Now	Now	Now	Now
Production	Now	Now	Now	Now

CSE466-Page 6

AVR Products

	S1200	S2323	S2343	S2313
Pins	20	8	8	20
Flash	1 KB	2 KB	2 KB	2 KB
SRAM	-	128 B	128 B	128 B
EEPROM	64 B	128 B	128 B	128 B
UART	-	-	-	1
PWM	-	-	-	1
Samples	Now	Now	Now	Now
Production	Now	Now	Now	Now

CSE466-Page 7

AVR Products

	S4433	S8515	VC8534	S8535
Pins	28/32	40/44	48	40/44
Flash	4 KB	8 KB	8 KB	8 KB
SRAM	128 B	512 B	256 B	512 B
EEPROM	256 B	512 B	512 B	512 B
UART	1	1	-	1
PWM	1	2	-	2
ADC	6@10-bit	-	6@10-bit	8@10-bit
RTC	-	-	-	Yes
Samples	Now	Now	Now	Now
Production	Now	Now	Now	Now

CSE466-Page 8

megaAVR Products

	mega161	mega163	mega32	mega103
Pins	40/44	40/44	40/44	64
Flash	16 KB	16 KB	32 KB	128 KB
SRAM	1 KB	1 KB	2 KB	4 KB
EEPROM	512 B	512 B	1 KB	2 KB
U(S)ART	2	1	1	1
TWI	1	1	1	-
PWM	4	4	4	4
ADC	-	8@10-bit	8@10-bit	8@10-bit
RTC	Yes	Yes	Yes	Yes
JTAG/OCD	-	-	Yes	-
Self Program	Yes	Yes	Yes	-
HW MULT	Yes	Yes	Yes	-
Brown Out	Yes	Yes	Yes	-
Samples	Now	Now	Now	Now
Production	Now	Now	Now	Now

megaAVR Products

	mega8	mega16	mega32	mega64	mega128
Pins	28/32	40/44	40/44	64	64
Flash	8 KB	16 KB	32 KB	64 KB	128 KB
SRAM	1 KB	1 KB	2 KB	4 KB	4 KB
EEPROM	512 B	512 B	1 KB	2 KB	4 KB
U(S)ART	1	1	1	2	2
TWI	1	1	1	1	1
PWM	3	4	4	8	8
ADC	8@10-bit	8@10-bit	8@10-bit	8@10-bit	8@10-bit
RTC	Yes	Yes	Yes	Yes	Yes
JTAG/OCD	-	Yes	Yes	Yes	Yes
Self Program	Yes	Yes	Yes	Yes	Yes
HW MULT	Yes	Yes	Yes	Yes	Yes
Brown Out	Yes	Yes	Yes	Yes	Yes
Samples	Now	Q1/02	Q2/02	Q2/02	Now
Production	Q1/02	Q2/02	Q2/02	Q2/02	Q1/02

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 USART
- 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 USARTs
- 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 varity 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

```
/* 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

```
; 7.    for (a=0;a<16;a++)
        LDI     R18,LOW(0)
        LDI     R19,128
        CLR     R22
?0001:
        CPI     R22,LOW(16)
        BRCC    ?0000
; 8.      {
; 9.        if (array[a]>maximum)
        MOV     R30,R22
        CLR     R31
        LSL     R30
        ROL     R31
        ADD     R30,R16
        ADC     R31,R17
        LDD     R20,Z+0
        LDD     R21,Z+1
        CP      R18,R20
        CPC     R19,R21
        BRGE    ?0005
; 10.           maximum=array[a];
        MOV     R18,R20
        MOV     R19,R21
?0005:
        INC     R22
        RJMP    ?0001
?0000:
; 11.     }
; 12.   return (maximum);
        MOV     R16,R18
        MOV     R17,R19
; 13.   }
        RET
```

Code Size: 46 Bytes, Execution time: 335 cycles

C51 Assembly Output

```
; FUNCTION _max (BEGIN)
;---- Variable 'array' assigned to Register 'R1/R2/R3' ----
        ; SOURCE LINE # 4
        ; SOURCE LINE # 5
        ; SOURCE LINE # 7
        MOV     maximum,#08BH
        MOV     maximum+01H,#09H
        ; SOURCE LINE # 9
;---- Variable 'a' assigned to Register 'R5' ----
        CLR     A
        MOV     R5,A
?C0001:
        ; SOURCE LINE # 10
        ; SOURCE LINE # 11
        MOV     A,R5
        MOV     R7,A
        RLC     A
        SUBB    A,ACC
        MOV     R6,A
        MOV     A,R7
        ADD     A,ACC
        MOV     R7,A
        MOV     A,R6
        RLC     A
        MOV     DPL,R7
        MOV     DPH,A
        LCALL   ?C?ILDOPTR
        MOV     R7,A
        MOV     R4,B
        SETB    C
        SUBB    A,maximum+01H
        MOV     A,maximum
        XRL     A,#080H
        MOV     R0,A
        MOV     A,R6
        XRL     A,#080H
        SUBB    A,R0
        JC      ?C0003
        ; SOURCE LINE # 12
        MOV     maximum,R6
        MOV     maximum+01H,R7
                                        ; SOURCE LINE # 13
?C0003:
        INC     R5
        CJNE    R5,#0100,?C0001
?C0002:
                                        ; SOURCE LINE # 14
        MOV     R6,maximum
        MOV     R7,maximum+01H
                                        ; SOURCE LINE # 15
?C0005:
        RET
        ; FUNCTION _max (END)
?C?ILDOPTR:
        CJNE    R3,#0x01,0x195
        MOV     A,0x82
        ADD     A,R1
        MOV     0x82,A
        MOV     A,0x83
        ADDC    A,R2
        MOV     0x83,A
        MOVX    A,@DPTR
        MOV     0x07,A
        INC     DPTR
        MOVX    A,@DPTR
        RET
        JNC     0x1A0
        MOV     A,R1
        ADD     A,0x82
        MOV     R0,A
        MOV     0x07,@R0
        INC     R0
        MOV     A,@R0
        RET
        CJNE    R3,#0xFE,0xAD
        MOV     A,R1
        ADD     A,0X82
        MOV     R0,A
        MOV     A,@R0
        MOV     0x07,A
        INC     R0
        RET
        MOV     A,0X83
        ADD     A,R2
        MOV     0X83,A
        MOV     A,R1
        MOVC    A,@A+DPTR
        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
        CLR     0,X
?0001:
        LDAA    0,X
        CMPA    #16
        BHS     ?0000
; 8.      {
; 9.        if (array[a]>maximum)
        PSHY
        TAB
        CLRA
        LSLD
        TSX
        ADDD    0,X
        XGDX
        LDD     0,X
        TSX
        STD     5,X
        INS
        INS
        CPD     3,X
        BLE     ?0005
; 10.                 maximum=array[a];
        STD     3,X
?0005:
        TSX
        INC     0,X
        BRA     ?0001
?0000:
; 11.     }
; 12.     return (maximum);
        LDD     1,X
; 13.   }
        PULX
        PULX
        INS
        PULY
        RTS
```

Code Size: 57 Bytes, Execution time: 5244 cycles

PIC16C74 Assembly output

```
        bcf     3,5
        movwf   ?a_maxnum& (0+127)
;MAX_MAIN.C: 4: char a;
;MAX_MAIN.C: 5: int maximum=-32768;
        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)
12
;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
        subwf   (?a_maxnum+2)& (0+127),w
u15
        btfsc   3,0
        goto    15
;MAX_MAIN.C: 10: maximum=array[a];
        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    4
        movf    0,w
        movwf   (?a_maxnum+3)& (0+127)
15
;MAX_MAIN.C: 11: }
        bcf     3,5
        incf    (?a_maxnum+1)& (0+127)
        movlw   16
        subwf   (?a_maxnum+1)& (0+127),w
        btfss   3,0
        goto    12
;MAX_MAIN.C: 12: return (maximum);
        bcf     3,5
        movf    (?a_maxnum+3)& (0+127),w
        movwf   btemp+1
        movf    (?a_maxnum+2)& (0+127),w
        movwf   btemp
        return
```

Code Size: 87 Bytes, Execution time: 2492 cycles

Performance Comparison

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

	Code Size [Bytes]	Function Execution Time [uS]	Current Consumption [mA]
AT90S8515	46 (1)	42 (1)	11 (1)
80C51	112 (2.4)	391 (9)	16 (1.5)
68HC11	57 (1.2)	437 (10)	27 (2.5)
PIC16C74	87 (1.9)	125 (3)	13.5 (1.2)

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:

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

Generated code:

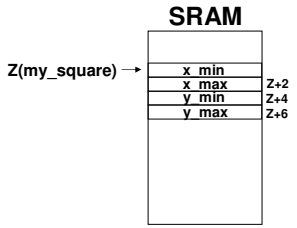
```
LD      R16,-X
ST      Z+,R16
```


C-Like Addressing Modes (2)

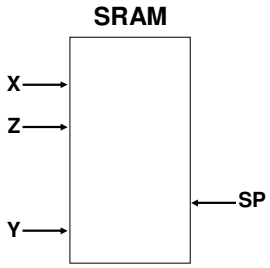
Indirect with Displacement:

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

```
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

```
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 *

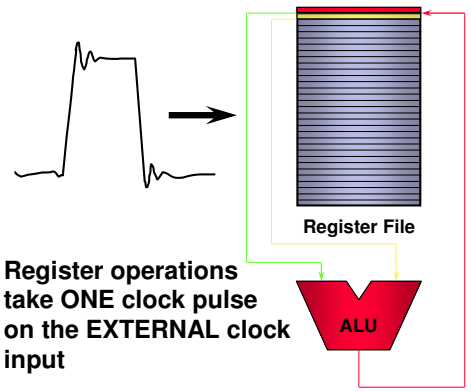
*** Works on the 4 Uppermost Register Pairs**

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**

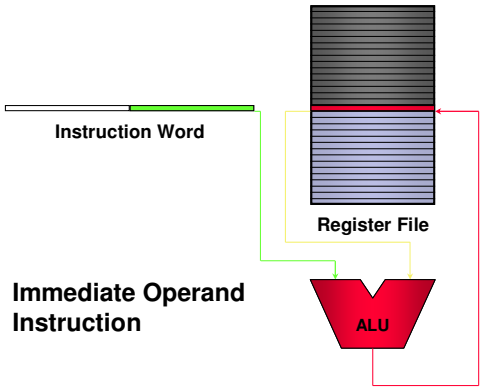
* **Works on Registers R16 - R31**
** **Works on the 4 Uppermost Register**

Direct Register - ALU Connection



Register operations take ONE clock pulse on the EXTERNAL clock input

Executing “subi r16,k”



Immediate Operand Instruction

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**

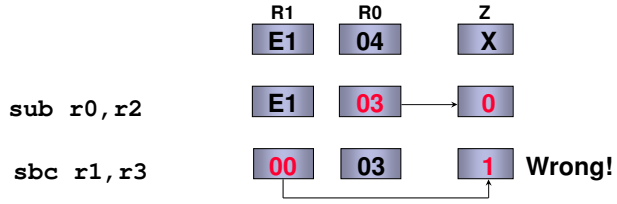
16/32-Bit Data Support

- 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

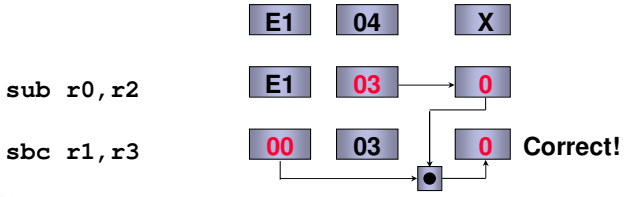
R1:R0 - R3:R2 ($E104 - $E101)



Subtract Two 16-Bit Values

With Zero Flag Propagation

R1:R0 - R3:R2 ($E104 - $E101)



Compare Two 32-Bit Values

- Example: Compare R3:R2:R1:R0 and R7:R6:R5:R4

```
cp  r0,r4
cpc r1,r5
cpc r2,r6
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

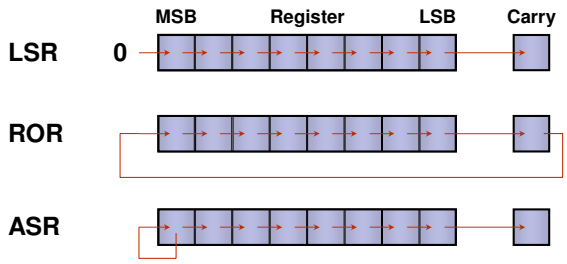
“AND”	Logical AND Two Registers
“ANDI”	Logical AND Immediate and Register *
“OR”	Logical OR Two Registers
“ORI”	Logical OR Immediate and Register *
“EOR”	Logical XOR Two Registers

* 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



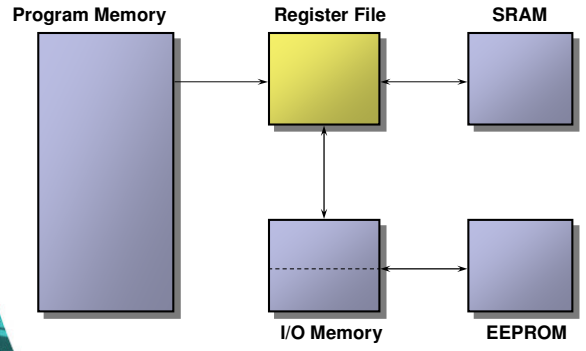
Data Transfer Instruction Types

	# of Cycles
Data SRAM <–> Register File	(2)
Program Memory –> Register File	(1/3)
I/O Memory <–> Register File	(1)
Register File <–> Register File	(1)

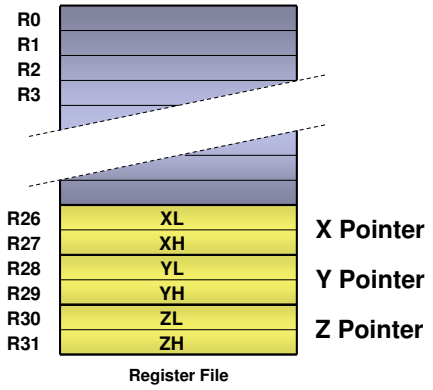
The Five Memory Areas

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



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

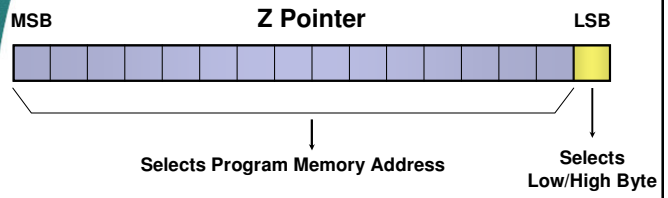
“LDI”	Load a Register with an Immediate Value (1 Cycle) *
“(E)LPM”	Transfer a byte from Program Memory@Z to R0 (3 Cycles)
“(E)LPM Rd,Z”	Transfer a byte from Program Memory@Z to Rd (3 Cycles)**
“(E)LPM Rd,Z+”	As above but with post-increment of the Z pointer**
“SPM”	Write to the Program Memory**

* **Works on R16 - R31**

** **Enhanced Core**

ELPM instructions use RAMPZ Register to reach >64KBytes

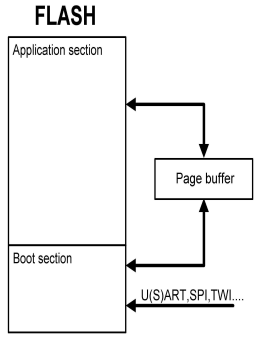
The (E)LPM Instructions



- **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

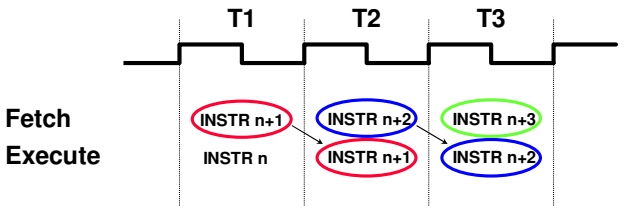
Unconditional Jump Instructions

"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 use the EIND register exists to extend the IJMP reach for devices with 256KBytes program memory or more.

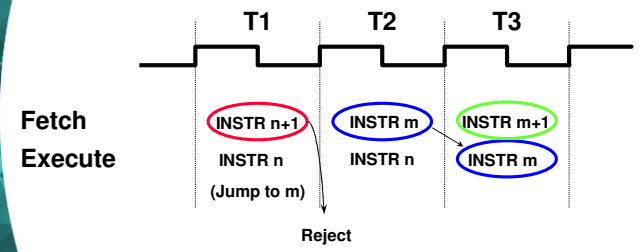
Instruction Fetch/Execution

Instructions are Sequential



Instruction Fetch/Execution

One Instruction is a Jump



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

	7	
Interrupt Enable	**I**	**Enables Global Interrupts when Set**
T Flag	**T**	**Source and Destination for BLD and BST**
Half Carry	**H**	**Set if an operation has half carry**
Signed Flag	**S**	**Used for Signed Tests**
Overflow Flag	**V**	**Set if Signed Overflow**
Negative Flag	**N**	**Set if a Result is Negative**
Zero Flag	**Z**	**Set if a Result is Zero**
Carry Flag	**C**	**Set if an operation has Carry**
	0	

Branch on SREG Settings

Branches if Bit Clear	7	**Branches if Bit Set**
BRID	**I**	**BRIE**
BRTC	**T**	**BRTS**
BRHC	**H**	**BRHS**
BRGE	**S**	**BRLT**
BRVC	**V**	**BRVS**
BRPL	**N**	**BRMI**
BRNE	**Z**	**BREQ**
BRSH, BRCC	**C**	**BRCS, BRLO**
	0	

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
$0126 lsl   r0        <- Interrupt
$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
$0005 reti  <- Return
```

- 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”.

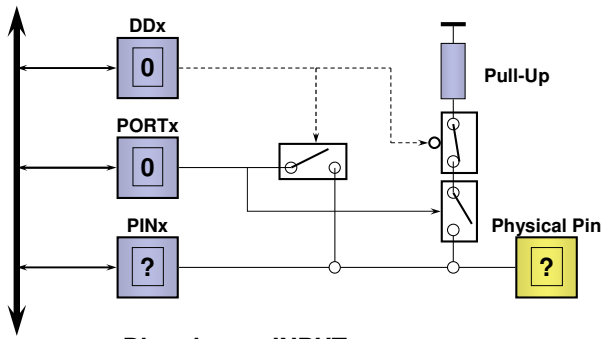
I/O Ports General Features

- 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

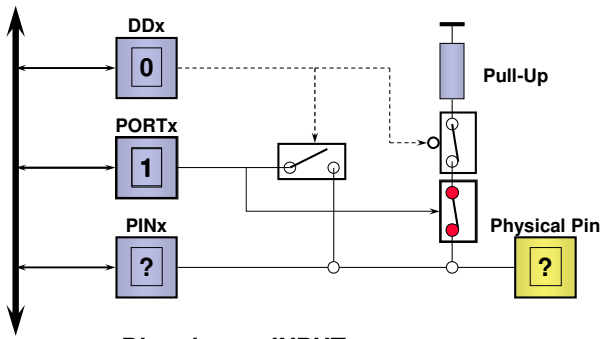
Default Configuration



Direction: INPUT
Pull-Up: OFF

CSE466-Page 89

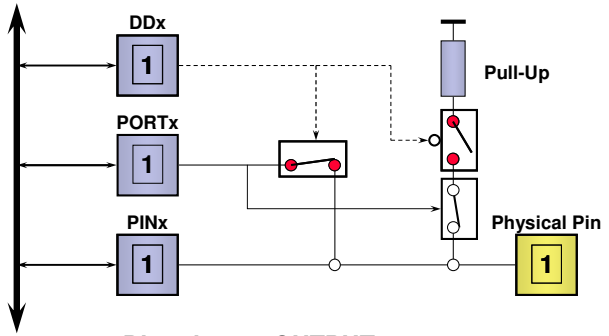
Switch On Pull-Up



Direction: INPUT
Pull-Up: ON

CSE466-Page 90

Port is Output



Direction: OUTPUT
Pull-Up: OFF

CSE466-Page 91

General T/C Features

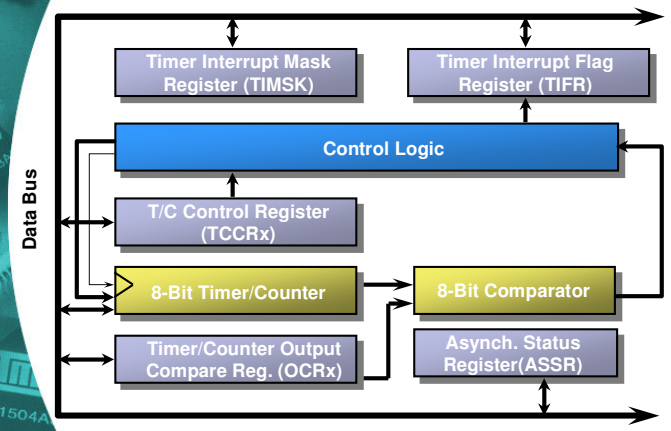
- Various Clock Prescaling Options
- Can Run at Undivided XTAL Frequency (High Resolution)
- Can be Set to Any Value at Any Time
- Can be Clocked Externally by Signals with Transition Periods down to XTAL/2
- Can be Clocked Externally on both Rising and Falling Edge
- The features vary from device to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

CSE466-Page 92

8 Bit Timer/Counter

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

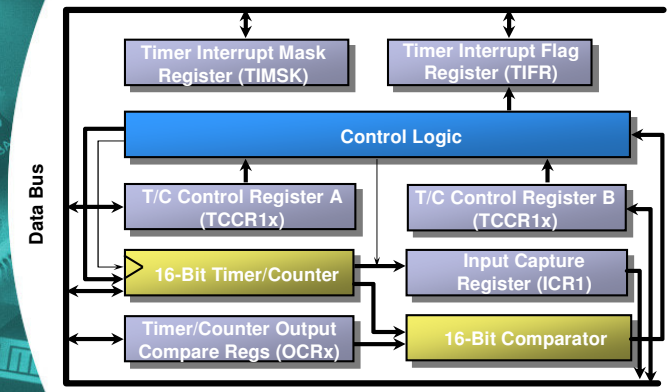
8 Bit Timer Block Diagram



16 Bit Timer/Counter

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

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

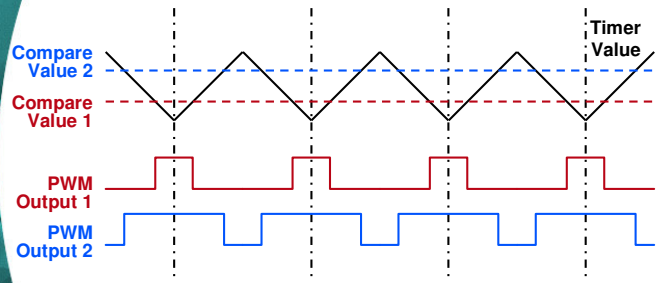
- Capture Can Trigger on Rising or Falling Edge (Optional)
- The Input Capture Noise Cancler will not trigger the Capture until 4 Subsequent samples of the same value are seen.

  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



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

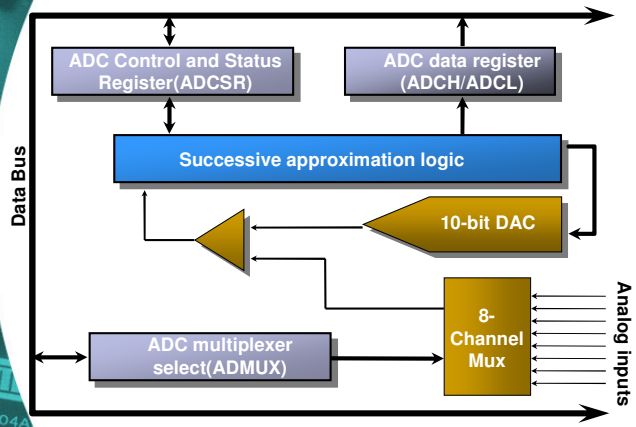
Analog Comparator Features

- 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



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

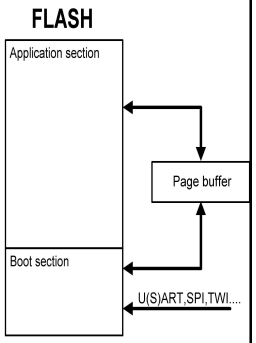
- In-System Programmable FLASH, EEPROM, Fuses and Lock Bits
- Programmable at all Frequencies
- Programmable at all VCCs above 2.7V
- Only four pins + Ground required

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

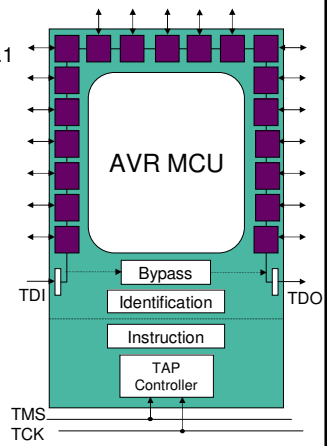
- Each memory partition has independent lock bit protection
  - Protect entire FLASH memory
  - Protect Boot Block while updating Application partition
  - Protect Application partition while updating Boot Block
  - Allow software update in entire FLASH address space where Boot Block is not used

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

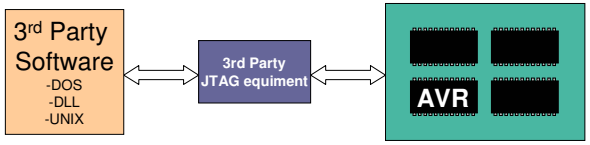
- Complies to IEEE std 1149.1
  - (JTAG)
- Boundary-Scan for Efficient PCB Test
- On-Chip Debugging in Production Silicon
- ISP Programming of both Flash and EEPROM during JTAG production test



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

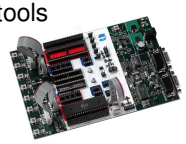
JTAG In System Programming



- 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 provides 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
  - Datasheets
  - Application Notes
  - FAQ
- Unofficial AVR websites
  www.avrfreaks.net
  www.avr-forum.com
- Support mail
  avr@atmel.com