Computational hardware

- Digital logic (CSE370)
  - Gates and flip-flops: glue logic, simple FSMs, registers
  - Two-level PLDs: FSMs, muxes, decoders
- Programmable logic devices (CSE370, CSE467)
  - Field-programmable gate arrays: FSMs, basic data-paths
  - Mapping algorithms to hardware
- Microprocessors (CSE378)
  - General-purpose computer
  - Instructions can implement complex control structures
  - Supports computations/manipulations of data in memory

Microprocessors

- Arbitrary computations
  - Arbitrary control structures
  - Arbitrary data structures
  - Specify function at high-level and use compilers and debuggers
- Microprocessors can lower hardware costs
  - If function requires too much logic when implemented with gates/FFs
    - Operations are too complex, better broken down as instructions
    - Lots of data manipulation (memory)
  - If function does not require higher performance of customized logic
    - Ever-increasing performance of processors puts more and more applications in this category
    - Minimize the amount of external logic

Microprocessor basics

- Composed of three parts
  - Data-path: data manipulation and storage
  - Control: determines sequence of actions executed in data-path and interactions to be had with environment
  - Interface: signals seen by the environment of the processor
- Instruction execution engine: fetch/execute cycle
  - Flow of control determined by modifications to program counter
  - Instruction classes:
    - Data: move, arithmetic and logical operations
    - Control: branch, loop, subroutine call
    - Interface: load, store from external memory

Microprocessor basics (cont’d)

- Can implement arbitrary state machine with auxiliary data-path
  - Control instructions implement state diagram
  - Registers and ALUs act as data storage and manipulation
  - Interaction with the environment through memory interface
  - How are individual signal wires sensed and controlled?

Microprocessor organization

- Controller
  - Inputs: from ALU (conditions), instruction read from memory
  - Outputs: select inputs for registers, ALU operations, read/write to memory
- Data-path
  - Register file to hold data
  - Arithmetic logic unit to manipulate data
  - Program counter (to implement relative jumps and increments)
- Interface
  - Data to/from memory (address and data registers in data path)
  - Read/write signals to memory (from control)

General-purpose processor

- Programmed by user
- New applications are developed routinely
- General-purpose
  - Must handle a wide ranging variety of applications
  - Interacts with environment through memory
  - All devices communicate through memory data
  - DMA operations between disk and I/O devices
  - Dual-ported memory (e.g., display screen)
  - Generally, oblivious to passage of time
Embedded processor

- Typically programmed once by manufacturer of system
  - Rarely does the user load new software
- Executes a single program (or a limited suite) with few parameters
- Task-specific
  - Can be optimized for a specific application
- Interacts with environment in many ways
  - Direct sensing and control of signal wires
  - Communication protocols to environment and other devices
  - Real-time interactions and constraints
  - Power-saving modes of operation to conserve battery power

Why embedded processors?

- High overhead in building a general-purpose system
  - Storing/loading programs
  - Operating system manages running of programs and access to data
  - Shared system resources (e.g., system bus, large memory)
- Many parts
  - Communication through shared memory/bus
  - Each I/O device often requires its own separate hardware unit
- Optimization opportunities
  - As much hardware as necessary for application
  - Cheaper, portable, lower-power systems
  - As much software as necessary for application
  - Doesn’t require a complete OS, get a lot done with a smaller processor
  - Can integrate processor, memory, and I/O devices on to a single chip

Typical general-purpose architecture

![Typical general-purpose architecture diagram]

- CPU
- Memory
- Display Interface (with dual-port video RAM)
- Disk
- Network Interface
- Sound Interface
- Standard interfaces
- System bus
- All the parts around the processor are usually required

Typical task-specific architecture

![Typical task-specific architecture diagram]

- Microcontroller (CPU+mem+…)
- ROM
- RAM
- Special I/O Device Driver
- Custom Logic

How does this change things?

- Sense and control of environment
  - Processor must be able to “read” and “write” individual wires
  - Controls I/O interfaces directly
- Measurement of time
  - Many applications require precise spacing of events in time
  - Reaction times to external stimuli may be constrained
- Communication
  - Protocols must be implemented by processor
  - Integrate I/O device or emulate in software
  - Capability of using external device when necessary

Interactions with the environment

- Basic processor only has address and data busses to memory
- Inputs are read from memory
- Outputs are written to memory
- Thus, for a processor to sense/control signal wires in the environment they must be made to appear as memory bits
  - How do we make wires look like memory?
Sensing external signals

- Map external wire to a bit in the address space of the processor
- External register or latch buffers values coming from environment
  - Map register into address space
  - Decoder selects register for reading
  - Output enable (OE) to get value on to data bus
  - Lets many registers use the same data bus

Controlling external signals

- Map external wire to a bit in the address space of the processor
- Connect output of memory-mapped register to environment
  - Map register into address space
  - Decoder selects register for writing (holds value indefinitely)
  - Input enable (EN) to take value from data bus
  - Lets many registers use the same data bus

Time and instruction execution

- Keep track of detailed timing of each instruction's execution
  - Highly dependent on code
  - Hard to use compilers
  - Not enough control over code generation
  - Interactions with caches/instruction-buffers
- Loops to implement delays
  - Keep track of time in counters
  - Keeps processor busy counting and not doing other useful things
- Timer
  - Take differences between measurements at different points in code
  - Keeps running even if processor is idle to save power
  - An independent "co-processor" to main processor

Time measurement via parallel timers

- Separate and parallel counting unit(s)
  - Co-processor to microprocessor
  - Does not require microprocessor intervention
  - May be a simple counter or a more featured real-time clock
  - Alarms can be set to generate interrupts
- More interesting timer units
  - Self reloading timers for regular interrupts
  - Pre-scaling for measuring larger times
  - Started by external events

Input/output events

- Input capture
  - Record time when input event occurred
  - Can be used in later handling of event
- Output compare
  - Set output event to happen at a point in the future
  - Reactive outputs
    - e.g., set output to happen a pre-defined time after some input
    - Processor can go on to do other things in the meantime

System bus based communication

- Extend address/data bus outside of chip
- Use specialized devices to implement communication protocol
- Map devices and their registers to memory locations
- Read/write data to receive/send buffers in shared memory or device
- Poll registers for status of communication
- Wait for interrupt from device on interesting events
  - Send completed
  - Receive occurred
Support for communication protocols

- Built-in device drivers
  - For common communication protocols
    - e.g., RS232, IrDA, USB, Bluetooth, etc.
  - Serial-line protocols most common as they require fewer pins
- Serial-line controller
  - Special registers in memory space for interaction
  - May use timer unit(s) to generate timing events
    - For spacing of bits on signal wire
    - For sampling rate
- Increase level of integration
  - No external devices
  - May further eliminate need for shared memory or system bus

Microcontrollers

- Embedded processor with much more integrated on same chip
  - Processor core + co-processors + memory
  - ROM for program memory, RAM for data memory, special registers to interface to outside world
  - Parallel I/O ports to sense and control wires
  - Timer units to measure time in various ways
  - Communication subsystems to permit direct links to other devices

Microcontrollers (cont’d)

- Other features not usually found in general-purpose CPUs
  - Expanded interrupt handling capabilities
    - Multiple interrupts with priority and selective enable/disable
    - Automatic saving of context before handling interrupt
  - More instructions for bit manipulations
    - Support operations on bits (signal wires) rather than just words
  - Integrated memory and support functions for cheaper system cost
    - Built-in EEPROM, Flash, and/or RAM
    - DRAM controller to handle refresh
    - Page-mode support for faster block transfers

The AVR Microcontroller Family

- Features

<table>
<thead>
<tr>
<th>Features</th>
<th>Flash</th>
<th>SRAM</th>
<th>EEPROM</th>
<th>UART</th>
<th>PWM</th>
<th>ADC</th>
<th>RTC</th>
<th>JTAG/OCD</th>
<th>TPS</th>
<th>Self Program</th>
<th>HW MULT</th>
<th>Brown Out</th>
</tr>
</thead>
<tbody>
<tr>
<td>51200</td>
<td>1 KB</td>
<td>128 B</td>
<td>64 B</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Yes</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>52020</td>
<td>1 KB</td>
<td>128 B</td>
<td>64 B</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Yes</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>52020</td>
<td>1 KB</td>
<td>128 B</td>
<td>64 B</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Yes</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>53133</td>
<td>1 KB</td>
<td>128 B</td>
<td>64 B</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Yes</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

The AVR Microcontroller Family

- Memory Density

<table>
<thead>
<tr>
<th>Memory Density</th>
<th>Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1 KB - 2KB)</td>
<td>16KB - 128KB</td>
</tr>
</tbody>
</table>

- The AVR Microcontroller Family

<table>
<thead>
<tr>
<th>Features</th>
<th>Flash</th>
<th>SRAM</th>
<th>EEPROM</th>
<th>UART</th>
<th>PWM</th>
<th>ADC</th>
<th>RTC</th>
<th>JTAG/OCD</th>
<th>TPS</th>
<th>Self Program</th>
<th>HW MULT</th>
<th>Brown Out</th>
</tr>
</thead>
<tbody>
<tr>
<td>51200</td>
<td>1 KB</td>
<td>128 B</td>
<td>64 B</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Yes</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>52020</td>
<td>1 KB</td>
<td>128 B</td>
<td>64 B</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Yes</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>52020</td>
<td>1 KB</td>
<td>128 B</td>
<td>64 B</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Yes</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>53133</td>
<td>1 KB</td>
<td>128 B</td>
<td>64 B</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Yes</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
Microcontroller we will be using

- Atmel AVR Microcontroller (ATmega16) – 16 MIPS at 16 MHz
  - 8-bit microcontroller (8-bit data, 16-bit instructions) – RISC Architecture
  - 128 instructions (most single-cycle – except 2-cycle multiply)
  - 512 byte scratchpad RAM
  - 40 pins
  - Internal and external interrupts
  - Memory
    - Instruction (16KB Flash memory – read-while-write)
    - Flash ROM (16K byte EEPROM)
    - data (16K byte SRAM)
  - Timer/counters
    - 2 8-bit and 1 16-bit timer/counters with compare modes and prescalers
    - Realtime clock (32.768 kHz) with separate oscillator
  - Serial communication interfaces
    - JTAG boundary-scan interface for programming/debugging
    - Programmable USART (universal synchronous/asynchronous receiver transmitter)
  - Two-wire serial interface (can emulate different communication protocols)
  - SPI serial port (serial peripheral interface)

Peripheral features
- Four channels with support for pulse-width modulation
- Analog-digital converter (8-channel, 10-bit)
- Up to 32 general-purpose I/O pins (with interrupt support)
- Six power saving modes

Why did we pick the ATmega16

- Modern microcontroller
- Easy to use C compiler
- Better performance/power than competitors
  - Microchip PIC
  - Motorola 68HC11
  - Intel 80C51
- Excellent support for 16-bit arithmetic operations
- A lot of registers that eliminate moves to and from SRAM
- Single cycle execution of most instructions
- Used in the UC Berkeley sensor mote (2nd half of qtr)
Operations on Register/Immediate Values

- One cycle register/immediate instructions

Instruction Timing

The Five Memory Areas

- General Purpose Register File = 32 B
- Flash Program Memory = 8 KB (≤ 8 MB)
- SRAM Data Memory = 1 KB (≤ 16 MB)
- I/O Memory = 64 B (≤ 64 B)
- EEPROM Data Memory = 512 B (≤ 16 MB)

Memory Map

Data SRAM → Register File (RF)

- "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)*

* PTR = X, Y or Z

Data SRAM ← Register File (RF)

- "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) *

* PTR = X, Y or Z
Data Transfer Program Memory -> RF

- "LDI"  Load a register with an immediate value (1 Cycle) *
- "LPM"  Transfer a byte from program Memory @ Z to R0 (3 Cycles)
- "LPM Rd,Z"  Transfer a byte from program Memory @ Z to Rd (3 Cycles)
- "LPM Rd,Z+" As above but with post-increment of the Z pointer

* Works on R16 - R31

Register File -> Register File

- "OUT"  Transfer a byte from RF to I/O
- "IN"  Transfer a byte from I/O to RF
- "MOV"  Copy a register to another register
- "MOVW"  Copy a register pair to another register pair. Aligned.

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

\[ \text{Struct square} \]
\[
\begin{align*}
\text{int } x_{\text{min}}; \\
\text{int } x_{\text{max}}; \\
\text{int } y_{\text{min}}; \\
\text{int } y_{\text{max}}; \\
\text{my_square;} 
\end{align*}
\]

SRAM

Branch on SREG Settings

\[ \text{BRID} \]
\[
\begin{align*}
\text{BRIE} & \text{ } \text{T} \\
\text{BRTS} & \text{ } \text{T} \\
\text{BRHS} & \text{ } \text{H} \\
\text{BRLT} & \text{ } \text{L} \\
\text{BRVC} & \text{ } \text{V} \\
\text{BRMI} & \text{ } \text{M} \\
\text{BREQ} & \text{ } \text{E} \\
\end{align*}
\]

\[ \text{Branches if Bit Clear} \]
\[ \text{Branches if Bit Set} \]
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++)
   LDD R20,2+0
   LDI R18,LOW(0)
   CP R18,R20
   CLI R19,128
   BREQ 70005
   BCF R22,LOW(16)

; 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
           MOV R18,R20
           MOV R19,R21
           ?0005:
           INC R22
           RJMP ?0001

; 10.    }
; 11.      return (maximum);
           MOV R16,R18
           MOV R17,R19
; 12.    }
           RET

Code Size: 46 Bytes, Execution time: 335 cycles

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

I/O Port Configurations

- 3 Control/Status Bits per Pin
  - DDX Data Direction Control Bit
  - PORTx Output Data or Pull-Up Control Bit
  - PINx Pin Level Bit

<table>
<thead>
<tr>
<th>DDX</th>
<th>PORTx</th>
<th>PINx (pin SFR)</th>
<th>I/O</th>
<th>Pullup</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>X</td>
<td>Input</td>
<td>No</td>
<td>Tri-state (Hi-Z)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>Input</td>
<td>Yes</td>
<td>Pin will source current if not pulled low</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>Input</td>
<td>No</td>
<td>Tri-state (Hi-Z)</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>X</td>
<td>Output</td>
<td>No</td>
<td>Output Low (Sink)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>X</td>
<td>Output</td>
<td>No</td>
<td>Output High (Source)</td>
</tr>
</tbody>
</table>

Port is Input (Default Configuration)
Port is Open-Collector Input (Switch On Pull-Up)

Port is Output

Sample Code from Lab 1

```
.include "C:\Program Files\Atmel\AVR\Tools\avrAssemble\Appnotes\m16def.inc"
.cseg
ldi r16, 0xff
out DDRB, r16
ldi r16, 0x00
out PORTB, r16
loop:
    jmp loop
```

Alternate Port Functions

- Generalizing I/O ports so that they can be used by other I/O devices

I/O Port Registers

Full I/O Register Map (pg. 329)
Instruction Classes (pg. 331)

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

Arithmetic/Logical Instructions

- Add Instructions
  - "ADD" Add Two Registers
  - "ADC" Add Two Registers and Carry
  - "INC" Increment a Register
  - "ADWI" Add Immediate to Word *
- 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
  - "SBWI" Subtract Immediate From Word**
- Compare Instructions
  - "CP" Compare Two Registers
  - "CPCT" Compare with Carry Two Registers
  - "CPI" Compare Register and Immediate*
  - "CPSE" Compare Two Registers and Skip Next Instruction if Equal

16-bit 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

Subtracting Two 16-Bit Values

- R1:R0 – R3:R2 (e.g., $E104 – $E101 )
  - Without zero-flag propagation
  - With zero-flag propagation

Comparing 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

- After last instruction, status register indicates equal, higher, lower, greater (signed), or less than (signed)

Arithmetic/Logical Instructions (cont’d)

- Multiply instructions
  - "MUL" 8x8 -> 16 (UxU)
  - "MULS" 8x8 -> 16 (SxS) *
  - "MULSU" 8x8 -> 16 (SxU)**

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

- 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

The result is present in R1:R0. All multiplication instructions are 2 cycles.
Arithmetic/Logical Instructions (cont’d)

- 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

Data Transfer Instruction Types

<table>
<thead>
<tr>
<th>Instruction Type</th>
<th>No 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>

Data Transfer RF <-> SRAM Stack

- “PUSH” PUSH a register on the stack
  - Decrement stack pointer by 1
  - Decrement by 2 when a return address is pushed on the stack
- “POP” POP a register from the stack
  - Increment stack pointer by 1
  - Incremented by 2 when a return address is popped off on return

Stack grows from higher to lower memory locations

Unconditional Jump Instructions

- “RJMP” Relative Jump *
- “JMP” Absolute Jump **

* Reaches ±2K instructions from current program location.
  Reaches all locations for devices up to 8KBytes (wrapping)
** 4-Byte Instruction

Flow Control

- Unconditional Jumps
- Conditional Branches
- Subroutine Call and Returns

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
**Subroutine Call and Return**

- "RCALL" Relative Subroutine Call *
- "CALL" Absolute Subroutine Call **
- "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)

---

**Bit Set/Clear 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 **
- "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

---

**Programming the ATmega16**

- Traditional in-system programming
  - In-system programmable FLASH, EEPROM, and lock bits
  - Programmable at all frequencies
  - Programmable at all VCCs above 2.7V
  - Only four pins + ground required
  - Requires adapter device to control programming pins
- Self programming
  - The AVR reprograms itself without any external components
  - Re-programmable through any communication interface
  - Does not have to be removed from board
  - Uses existing communication ports
  - Critical functions still operating
  - device is running during programming

---

**AVR JTAG Interface**

- Complies to IEEE std 1149.1 (JTAG)
- Boundary-scan for efficient PCB test
  - Standard for interconnection test
  - All I/O pins controllable and observable from tester
- On-chip debugging in production

---

**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

---

**JTAG In-Circuit Emulator**

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