Atmel Mega8 Basics

• Complete, standalone computer
• Ours is a 28-pin package
• Most pins:
  – Are used for input/output
  – How they are used is configurable

```
   PDIP
   (RESET) PC6 □ 1  28 □ PC5 (ADC5/SCL)
   (RXD) PD0 □ 2  27 □ PC4 (ADC4/SDA)
   (TXD) PD1 □ 3  26 □ PC3 (ADC3)
   (INT0) PD2 □ 4  25 □ PC2 (ADC2)
   (INT1) PD3 □ 5  24 □ PC1 (ADC1)
   (XCK/T0) PD4 □ 6  23 □ PC0 (ADC0)
   VCC □ 7  22 □ GND
   GND □ 8  21 □ AREF
   (XTAL1/TOSC1) PB6 □ 9  20 □ AVCC
   (XTAL2/TOSC2) PB7 □ 10  19 □ PB5 (SCK)
   (T1) PD5 □ 11  18 □ PB4 (MISO)
   (AIN0) PD6 □ 12  17 □ PB3 (MOSI/OC2)
   (AIN1) PD7 □ 13  16 □ PB2 (SS/OC1B)
   (ICP1) PB0 □ 14  15 □ PB1 (OC1A)
```
Key Features

• Up to 16 MIPS (single cycle for most instructions)
• ~23 digital pins: configurable as inputs or outputs
• 6 channel, 10-bit analog-to-digital converter
• Serial communication support: RS232, SPI, I2C
• 3 counter/timers (2 8-bit; 1 16-bit)
• Internal/external interrupt support
• Brown-out detection
• Internal oscillator (1 MHz)
• Bootloader support
• Sleep mode
• Watchdog timer
Interrupt Sources

- External pins: state change; falling/rising edge
- Timer/counters: when counter overflows
- Communication peripherals
- Brown out
- Analog to digital conversion complete
Atmel Mega8 Basics

Power (we will use +5V)

<table>
<thead>
<tr>
<th>PDIP</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(RESET) PC6</td>
<td>1  PC5 (ADC5/SCL)</td>
</tr>
<tr>
<td>(RXD) PD0</td>
<td>2  PC4 (ADC4/SDA)</td>
</tr>
<tr>
<td>(TXD) PD1</td>
<td>3  PC3 (ADC3)</td>
</tr>
<tr>
<td>(INT0) PD2</td>
<td>4  PC2 (ADC2)</td>
</tr>
<tr>
<td>(INT1) PD3</td>
<td>5  PC1 (ADC1)</td>
</tr>
<tr>
<td>(XCK/T0) PD4</td>
<td>6  PC0 (ADC0)</td>
</tr>
<tr>
<td>VCC</td>
<td>7  VDD (5V)</td>
</tr>
<tr>
<td>GND</td>
<td>8  GND</td>
</tr>
<tr>
<td>XTAL1/TOSC1</td>
<td>9  AREF</td>
</tr>
<tr>
<td>XTAL2/TOSC2</td>
<td>10 AVCC</td>
</tr>
<tr>
<td>(T1) PD5</td>
<td>11 PB5 (SCK)</td>
</tr>
<tr>
<td>(AIN0) PD6</td>
<td>12 PB4 (MISO)</td>
</tr>
<tr>
<td>(AIN1) PD7</td>
<td>13 PB3 (MOSI/OC2)</td>
</tr>
<tr>
<td>(ICP1) PB0</td>
<td>14 PB2 (SS/OC1B)</td>
</tr>
<tr>
<td></td>
<td>15 PB1 (OC1A)</td>
</tr>
</tbody>
</table>
Atmel Mega8 Basics

Ground

(PDIP)

(RESET) PC6 1 28  PC5 (ADC5/SCL)
(RXD) PD0 2 27  PC4 (ADC4/SDA)
(TXD) PD1 3 26  PC3 (ADC3)
(INT0) PD2 4 25  PC2 (ADC2)
(INT1) PD3 5 24  PC1 (ADC1)
(XCK/T0) PD4 6 22  PC0 (ADC0)
VCC 7 21  AREF
GND 8 20  AVCC
(XTAL1/TOSC1) PB6 9 19  PB5 (SCK)
(XTAL2/TOSC2) PB7 10 18  PB4 (MISO)
(T1) PD5 11 17  PB3 (MOSI/OC2)
(AIN0) PD6 12 16  PB2 (SS/OC1B)
(AIN1) PD7 13 15  PB1 (OC1A)
(ICP1) PB0 14

Andrew H. Fagg: Embedded Systems: Atmel Basics
Atmel Mega8 Basics

Reset

• Bring low to reset the processor
• In general, we will tie this pin to high through a pull-up resistor (10K ohm)
Atmel Mega8 Basics

PORT B

<table>
<thead>
<tr>
<th>Pin</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PC6</td>
<td>(RESET)</td>
</tr>
<tr>
<td>PD0</td>
<td>(RXD)</td>
</tr>
<tr>
<td>PD1</td>
<td>(TXD)</td>
</tr>
<tr>
<td>PD2</td>
<td>(INT0)</td>
</tr>
<tr>
<td>PD3</td>
<td>(INT1)</td>
</tr>
<tr>
<td>PD4</td>
<td>(XCK/T0)</td>
</tr>
<tr>
<td>VCC</td>
<td></td>
</tr>
<tr>
<td>GND</td>
<td></td>
</tr>
<tr>
<td>PB0</td>
<td>(XTAL1/TOSC1)</td>
</tr>
<tr>
<td>PB1</td>
<td>(XTAL2/TOSC2)</td>
</tr>
<tr>
<td>PB2</td>
<td>(T1)</td>
</tr>
<tr>
<td>PB3</td>
<td>(AIN0)</td>
</tr>
<tr>
<td>PB4</td>
<td>(AIN1)</td>
</tr>
<tr>
<td>PB5</td>
<td>(ICP)</td>
</tr>
<tr>
<td>PC0</td>
<td></td>
</tr>
<tr>
<td>PC1</td>
<td></td>
</tr>
<tr>
<td>PC2</td>
<td></td>
</tr>
<tr>
<td>PC3</td>
<td></td>
</tr>
<tr>
<td>PC4</td>
<td></td>
</tr>
<tr>
<td>PC5</td>
<td></td>
</tr>
</tbody>
</table>

Andrew H. Fagg: Embedded Systems: Atmel Basics
Atmel Mega8 Basics

PORT C

(RESET) PC6 □ 1
(RXD) PB0 □ 2
(TXD) PD1 □ 3
(INT0) PD2 □ 4
(INT1) PD3 □ 5
(XCK/T0) PD4 □ 6
VCC □ 7
GND □ 8
(XTAL1/TOSC1) PB6 □ 9
(XTAL2/TOSC2) PB7 □ 10
(T1) PD5 □ 11
(AIN0) PD6 □ 12
(AIN1) PD7 □ 13
(ICP1) PB0 □ 14

28 □ PC5 (ADC5/SCL)
27 □ PC4 (ADC4/SDA)
26 □ PC3 (ADC3)
25 □ PC2 (ADC2)
24 □ PC1 (ADC1)
23 □ PC0 (ADC0)
22 □ GND
21 □ AREF
20 □ AVCC
19 □ PB5 (SCK)
18 □ PB4 (MISO)
17 □ PB3 (MOSI/OC2)
16 □ PB2 (SS/OC1B)
15 □ PB1 (OC1A)
Atmel Mega8 Basics

PORT D
(all 8 bits are available)
A First Circuit
Common Special-Purpose Registers

- Program counter
- Status register
- Instruction register
- Stack pointer
- Peripheral control is all done through registers
Atmel Mega8

8-bit data bus
- Primary mechanism for data exchange
Atmel Mega8

32 general purpose registers
- 8 bits wide
- 3 pairs of registers can be combined to give us 16 bit registers
Atmel Mega8

Special purpose registers

• Control of the internals of the processor
Atmel Mega8

Random Access Memory (RAM)
- 1 KByte in size
-Globals, heap and stack are stored here
Atmel Mega8

Flash (EEPROM)
- Program storage
- 8 KByte in size
- 16 bit words
Atmel Mega8

EEPROM

- Permanent data storage
Atmel Mega8

Arithmetic Logical Unit
- Data inputs from registers
- Control inputs not shown (derived from instruction decoder)
Processors in the Atmel Family

- Memory/program size
- Different numbers and types of I/O pins
- Custom support for other communication protocols (e.g., CANbus)
Instruction Fetch/Execution Cycle

- While one instruction is being executed, the next is already being fetched from memory
- In many cases: each step happens on a single clock cycle

From Atmel Mega8 spec
Instruction Execution Cycle

Address the registers and wait for the values to become available
Perform the operation dictated by the instruction
Instruction Execution Cycle

Result stored in destination register
Status register state changed
Atmel Mega8

Control the pins through the I/O modules

- At the heart, these are registers … that are implemented using D flip-flops!

Andrew H. Fagg: Embedded Systems: Atmel Basics
I/O Pin Implementation

Single bit of PORT B

PUD: PULLUP DISABLE
SLEEP: SLEEP CONTROL
clk\_VO: I/O CLOCK
WDx: WRITE DDRx
RDx: READ DDRx
WPx: WRITE PORTx
RRx: READ PORTx REGISTER
RPx: READ PORTx PIN
I/O Pin Implementation

The physical pin

PINx

PORTx

DDRx

WDx

RDx

WPx

RX

RPx

clk

PUD

SLEEP

SLEEP CONTROL

clk

WRITE DDRx

READ DDRx

WRITE PORTx

READ PORTx REGISTER

READ PORTx PIN
I/O Pin Implementation

DDRB

- Defines whether this is an input or an output
I/O Pin Implementation

PORTB
- Defines the value that is written out to the pin (if it is an output)
Tristate buffer
• When this pin is an output pin, it allows the PORTB flip-flop to drive the pin.
I/O Pin Implementation

Input tri-state buffer
I/O Pin Implementation
I/O Pin Implementation

$$\text{DDRB} = 0;$$
I/O Pin Implementation

DDRB = 0;

• “0” is written to the data bus
I/O Pin Implementation

DDRB = 0;

• “0” is written to the data bus
• This is input to the DDRB register
I/O Pin Implementation

$$\text{DDRB} = 0;$$

- “0” is written to the data bus
- This is input to the DDRB register
- WDB is clocked from high to low

- PUD: PULLUP DISABLE
- SLEEP: SLEEP CONTROL
- clkVDD: I/O CLOCK
- WDx: WRITE DDRx
- RDx: READ DDRx
- WPx: WRITE PORTx
- RRx: READ PORTx REGISTER
- RPx: READ PORTx PIN
I/O Pin Implementation

DDRB = 0;

- “0” is written to the data bus
- This is input to the DDRB register
- WDB is clocked from high to low
- “0” is stored by the flip-flop
I/O Pin Implementation

DDRB = 0;

- "0" is written to the data bus
- This is input to the DDRB register
- WDB is clocked from high to low
- "0" is stored by flip-flop
- Which turns off the tri-state buffer

-> this is an input pin
I/O Pin Implementation

DDRB = 1;

• “1” is written to the data bus

PUD: PULLUP DISABLE
SLEEP: SLEEP CONTROL
clk\_vo: I/O CLOCK
WDx: WRITE DDRx
RDx: READ DDRx
WPx: WRITE PORTx
RRx: READ PORTx REGISTER
RPx: READ PORTx PIN
I/O Pin Implementation

DDRB = 1;

• “1” is written to the data bus
• This is input to the DDRB register
• WDB is clocked from high to low
• “1” is stored by flip-flop
• Which turns on the tri-state buffer

-> this is an output pin
I/O Pin Implementation

PORTB = 1;

1

DDRx

PORTx

PINx

PUD: PULLUP DISABLE
SLEEP: SLEEP CONTROL
clkVO: I/O CLOCK
WDx: WRITE DDRx
RDx: READ DDRx
WPx: WRITE PORTx
RRx: READ PORTx REGISTER
RPx: READ PORTx PIN
I/O Pin Implementation

PORTB = 1;

- “1” is written to the data bus
- This is input to the PORTB register
I/O Pin Implementation

PORTB = 1;

- "1" is written to the data bus
- This is input to the PORTB register
- WPB is clocked from high to low
- "1" is stored by flip-flop
I/O Pin Implementation

PORTB = 1;

- “1” is written to the data bus
- This is input to the PORTB register
- WPB is clocked from high to low
- “1” is stored by flip-flop
- Which provides a “1” to the tri-state buffer

-> output a “1”
I/O Pin Implementation

PORTB = 0;

• “0” is written to the data bus

---

PUD: PULLUP DISABLE
SLEEP: SLEEP CONTROL
clkVoid: I/O CLOCK
WDx: WRITE DDRx
RDx: READ DDRx
WPx: WRITE PORTx
R Rx: READ PORTx REGISTER
RPx: READ PORTx PIN
PORTB = 0;

- “0” is written to the data bus
- This is input to the PORTB register
- WPB is clocked from high to low
- “0” is stored by flip-flop
- Which provides a “0” to the tri-state buffer

→ output a “0”
I/O Pin Implementation

foo = PORTB;
foo = PORTB;

- RPB is set high
I/O Pin Implementation

foo = PORTB;

- RPB is clocked from high to low
- “0” is written to the data bus
I/O Pin Implementation

$\text{DDRB} = 0$;

- “0” is written to the data bus
- This is input to the DDRB register
- WDB is clocked from high to low
- “0” is stored by flip-flop
- Which turns off the tri-state buffer

$\rightarrow$ this is an input pin
I/O Pin Implementation

foo = PINB;

PUD: PULLUP DISABLE
SLEEP: SLEEP CONTROL
clk\_VO: I/O CLOCK
WDx: WRITE DDRx
RDx: READ DDRx
WPx: WRITE PORTx
RRx: READ PORTx REGISTER
RPx: READ PORTx PIN
foo = PINB;

- RPB is set high
I/O Pin Implementation

```c
foo = PINB;
```

- RPB is clocked from high to low
- The pin state is copied to the data bus
Bit Manipulation

PORTB is a register
• Controls the value that is output by the set of port B pins
• But – all of the pins are controlled by this single register (which is 8 bits wide)

• In code, we need to be able to manipulate the pins individually
Bit-Wise Operators

If A and B are bytes, what does this code mean?

\[ C = A \& B; \]

The corresponding bits of A and B are ANDed together
Bit-Wise Operators

If A and B are bytes, what does this code mean?

\[ C = A \& B; \]
Bit-Wise Operators

\[ \begin{array}{c}
0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 \\
\end{array} \quad A \\
\begin{array}{c}
1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\
\end{array} \quad B \\
\begin{array}{c}
? \\
C = A \& B \\
\end{array} \]
Bit-Wise Operators

0 1 0 1 1 1 1 0  A
1 0 0 1 1 0 1 1  B

C = A & B
Bit-Wise Operators

\[
\begin{align*}
0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & \quad & A \\
1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & \quad & B \\
\hline
0 & & & & & & & & \quad & C = A \& B
\end{align*}
\]
Bit-Wise Operators

0 1 0 1 1 1 1 0  \quad A

1 0 0 1 1 0 1 1  \quad B

1 0  \quad C = A \& B
Bit-Wise Operators

\[ \begin{array}{c}
0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 \\
1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\
\hline
0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\
\end{array} \]

\[ A = \begin{array}{c}
0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 \\
1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\
\hline
0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\
\end{array} \]
Bit-Wise Operators

Other Operators:

• OR:  |
• XOR: ^
Bit Manipulation

Given a byte A, how do we set bit 2 (counting from 0) of A to 1?
Bit Manipulation

Given a byte A, how do we set bit 2 (counting from 0) of A to 1?

\[ A = A \mid 4; \]
Bit Manipulation

Given a byte $A$, how do we set bit 2 (counting from 0) of $A$ to 0?
Bit Manipulation

Given a byte $A$, how do we set bit 2 (counting from 0) of $A$ to 1?

$$A = A \& 0xFFB;$$

$$A \&= \sim 4;$$
A First Program

Flash the LEDs at a regular interval

• How do we do this?
A First Program

How do we flash the LED at a regular interval?

- We toggle the state of PB0
I/O Pin Implementation

Single bit of PORT B
A First Program

```c
main() {
    DDRB = 0x1; // Pin 0 to output

    while(1) {
        PORTB = PORTB ^ 0x1; // XOR bit 0 with 1
        delay_ms(500); // Pause for 500 msec
    }
}
```
A Second Program

main() {
    DDRB = 0x3; // Set all port B pins as outputs

    while(1) {
        PORTB = PORTB ^ 0x1; // XOR bit 0 with 1
        delay_ms(500); // Pause for 500 msec
        PORTB = PORTB ^ 0x2; // XOR bit 1 with 1
        delay_ms(250);
        PORTB = PORTB ^ 0x2; // XOR bit 1 with 1
        delay_ms(250);
    }
}

What does this program do?
A Second Program

```c
main() {
    DDRB = 0xFF;   // Set all port B pins as outputs

    while(1) {
        PORTB = PORTB ^ 0x1;   // XOR bit 0 with 1
        delay_ms(500);         // Pause for 500 msec
        PORTB = PORTB ^ 0x2;   // XOR bit 1 with 1
        delay_ms(250);
        PORTB = PORTB ^ 0x2;   // XOR bit 1 with 1
        delay_ms(250);
    }
}
```

**Flashes LED on PB1 at 1 Hz**

**on PB0: 0.5 Hz**
More Bit Masking

• Suppose we have a 3-bit number (so values 0 ... 7)

• Suppose we want to set the state of B3, B4, and B5 with this number (B3 is the least significant bit)

• How do we express this in code?
Bit Masking

```c
main() {
    DDRB = 0x38;   // Set pins B3, B4, B5 as outputs

    uint8_t val;  // A short is 8-bits wide

    val = command_to_robot;   // A value between 0 and 7

    PORTB = (PORTB & ~0x38)      // Set the current B3-B5 to 0s
      | ((val & 0x7)<<3);      // OR with new values (shifted
      // to fit within B3-B5
}
```
Reading the Digital State of Pins

Given: we want to read the state of PB6 and PB7 and obtain a value of 0 … 3

• How do we configure the port?
• How do we read the pins?
• How do we translate their values into an integer of 0 .. 3?
Reading the Digital State of Pins

main() {
    DDRB = 0x38;   // Set pins B3, B4, B5 as outputs
    // All others are inputs (suppose we care
    // about bits B6 and B7 only (so a 2-bit
    // number)

    :
    :

    unsigned short val, outval;  // A short is 8-bits wide

    val = PINB;

    outval = (val & 0xC0) >> 6;
}

Port-Related Registers

The set of C-accessible register for controlling digital I/O:

<table>
<thead>
<tr>
<th></th>
<th>Directional control</th>
<th>Writing</th>
<th>Reading</th>
</tr>
</thead>
<tbody>
<tr>
<td>Port B</td>
<td>DDRB</td>
<td>PORTB</td>
<td>PINB</td>
</tr>
<tr>
<td>Port C</td>
<td>DDRC</td>
<td>PORTC</td>
<td>PINC</td>
</tr>
<tr>
<td>Port D</td>
<td>DDRD</td>
<td>PORTD</td>
<td>PIND</td>
</tr>
</tbody>
</table>
A Note About the C/Atmel Book

The book uses C syntax that looks like this:
PORTA.0 = 0;  // Set bit 0 to 0

This syntax is not available with our C compiler. Instead, you will need to use:
PORTA &= 0xFE;

or
PORTA &= ~1;

or
PORTA = PORTA & ~1;
Putting It All Together

• Program development:
  – On your own laptop
  – We will use a C “crosscompiler” (avr-gcc and other tools) to generate code on your laptop for the mega8 processor

• Program download:
  – We will use “in circuit programming”: you will be able to program the chip without removing it from your circuit