Sensor Networks

Microprocessors and algorithms
Your Joulo results

Good heating control

This Joulo started recording on Thursday 6 December, 2012 15:31 (UTC) in the Southampton, Hampshire, United Kingdom area.

Internal temperature | External temperature | Both | Good example | Bad example
--- | --- | --- | --- | ---
22°C | 21°C | 20°C | 19°C | 18°C | 17°C | 16°C | 15°C | 14°C | 13°C | 12°C | 11°C | 10°C | 9°C | 8°C | 7°C | 6°C | 5°C | 4°C | 3°C | 2°C | 1°C

Your thermostat setting is 1°C above average. Setting it to 1 degree less could save you 6% on your heating bill over the course of a year. Click the circle on the left to show your thermostat setting on your temperature log. Learn more about thermostat settings...

Your home's insulation score is 2/10 which is worse than the UK average. Click the circle on the left to show your home cooling down on your temperature log. Learn more about insulation score...
Tap the cicada to start survey
Execution unit

Functional units

Instruction control unit

Branch

Arithmetic operations

Load

Store

Instruction cache

Data cache

Fetch control

Instruction decode

Address Instructions

Operations

Prediction

OK?

Data

Data

Addr.

Addr.

Arithmetic operations

Operation results

Retirement unit

Register file

Register updates
<table>
<thead>
<tr>
<th>File Name</th>
<th>Description</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>AudioAnalyser.java</td>
<td>Android implementation v1</td>
<td>2 years ago</td>
</tr>
<tr>
<td>Baselines.java</td>
<td>added android support</td>
<td>2 years ago</td>
</tr>
<tr>
<td>Filter.java</td>
<td>fixed problem with Heterodyne in Android</td>
<td>a year ago</td>
</tr>
<tr>
<td>Goertzel.java</td>
<td>added android support</td>
<td>2 years ago</td>
</tr>
<tr>
<td>Heterodyne.java</td>
<td>fixed problem with Heterodyne in Android</td>
<td>a year ago</td>
</tr>
<tr>
<td>HighPassFilter.java</td>
<td>fixed problem with Heterodyne in Android</td>
<td>a year ago</td>
</tr>
<tr>
<td>Kalman.java</td>
<td>added android support</td>
<td>2 years ago</td>
</tr>
<tr>
<td>LowPassFilter.java</td>
<td>fixed problem with Heterodyne in Android</td>
<td>a year ago</td>
</tr>
<tr>
<td>RecorderPlugin.java</td>
<td>android wrapping of recording to file</td>
<td>4 months ago</td>
</tr>
<tr>
<td>Spectrogram.java</td>
<td>Android implementation v1</td>
<td>2 years ago</td>
</tr>
</tbody>
</table>
This is the reference design for the Arduino MEGA that comes with every installation of EAGLE.

Assembled Boards:

- 100 boards

Lead Time:

Ships on Tue Feb 21 2017 (10 week days)
AudioMoth

A low-cost open-source acoustic logger for biodiversity and environmental monitoring

Just like its namesake, AudioMoth has an extremely good sense of hearing, with the ability to listen for and capture sound from the audible frequencies, well into ultrasonic frequencies. AudioMoth is a low-cost, full-spectrum acoustic logger, based on the Silicon Labs Gecko processor range. It can record uncompressed audio to an SD card at up to 250,000 samples per second and onboard acoustic recognition algorithms allow the device to decide when and what to record.

- EFM32 Gecko processor
- Records WAV files to SD card
- Capable of recording at sample rates up 250kHz
- Capable of logging up to 125kHz
- Powered by 3 x AAA batteries
- Analog MEMS microphone
- Measures just 50 x 38 x 12 mm
- Configurable USB interface
- Onboard real time clock
Microprocessors
Intel 8086 microprocessor

Intel 8087 FPU
addq  %$1,-4(%rsi, %rax, 4)
long multiply1(long *a, long n) {
    long i;
    long acc = 1;
    for (i = 0; i < n; i += 1) {
        acc = acc * a[i];
    }
    return acc;
}
long multiply1(long *a, long n) {

    long i;
    long acc = 1;
    for (i = 0; i < n; i += 2) {
        acc = acc * a[i];
        acc = acc * a[i + 1];
    }

    return acc;
}
long multiply2(long *a, long n) {

    long i;
    long acc0, acc1 = 1;
    for (i = 0; i < n; i += 2) {
        acc0 = acc0 * a[i];
        acc1 = acc1 * a[i + 1];
    }

    return acc0 * acc1;
}
long multiply3(long *a, long n) {

    long i;
    long acc0, acc1, acc2 = 1;
    for (i = 0; i < n; i += 3) {
        acc0 = acc0 * a[i];
        acc1 = acc1 * a[i + 1];
        acc2 = acc2 * a[i + 2];
    }

    return acc0 * acc1 * acc2;
}
1. Integer arithmetic, floating point multiplication, integer and floating point division, branches

2. Integer arithmetic, floating point addition, integer and floating point multiplication

3. Load, address computation

4. Load, address computation

5. Store

6. Integer arithmetic

7. Integer arithmetic, branches

8. Store address computation
<table>
<thead>
<tr>
<th>Operation</th>
<th>Integer</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>latency</td>
<td>issue</td>
<td>capacity</td>
</tr>
<tr>
<td>Addition</td>
<td>1</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>Multiplication</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Division</td>
<td>3-30</td>
<td>3-30</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Operation</th>
<th>Floating Point</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>latency</td>
<td>issue</td>
<td>capacity</td>
</tr>
<tr>
<td>Addition</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Multiplication</td>
<td>5</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Division</td>
<td>3-15</td>
<td>3-15</td>
<td>1</td>
</tr>
<tr>
<td>Bound</td>
<td>Integer</td>
<td></td>
<td>Floating Point</td>
</tr>
<tr>
<td>-------------</td>
<td>---------</td>
<td>-------</td>
<td>----------------</td>
</tr>
<tr>
<td></td>
<td>+</td>
<td>×</td>
<td></td>
</tr>
<tr>
<td>Latency</td>
<td>1.0</td>
<td>3.0</td>
<td>3.0</td>
</tr>
<tr>
<td>Throughput</td>
<td>0.5</td>
<td>1.0</td>
<td>1.0</td>
</tr>
</tbody>
</table>
long multiply3(long *a, long n) {

    long i;
    long acc0, acc1, acc2 = 1;
    for (i = 0; i < n; i += 3) {
        acc0 = acc0 * a[i];
        acc1 = acc1 * a[i + 1];
        acc2 = acc2 * a[i + 2];
    }

    return acc0 * acc1 * acc2;
}
add $r0, $r2, $r3

add $r0, $r2, $r3 lsl #2

add $r0, $r2, $r3 lsl $r4

add $r0, $r2, #1024
- Register, optionally with shift operation applied.
- Shift value can be either be:
  - 5 bit unsigned integer
  - Specified in bottom byte of another register.

*Immediate value*
- 8 bit number
- Can be rotated right through an even number of positions.
- Assembler will calculate rotate for you from constant.
add r0, r2, r3
add r0, r2, r3 lsl #2
add r0, r2, r3 lsl r4
add r0, r2, #1024
Microcontrollers
# Product Families

<table>
<thead>
<tr>
<th>Family</th>
<th>Core</th>
<th>Speed (MHz)</th>
<th>Flash Memory (kB)</th>
<th>RAM</th>
<th>USB</th>
<th>LCD</th>
<th>Communications</th>
<th>Packages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zero Gecko</td>
<td>ARM Cortex-M0+</td>
<td>24</td>
<td>4, 8, 16, 32</td>
<td>2, 4</td>
<td>No</td>
<td>No</td>
<td>I2C, I2S, SPI, UART, USART</td>
<td>QFN24, QFN32, QFP48</td>
</tr>
<tr>
<td>Happy Gecko</td>
<td>ARM Cortex-M0+</td>
<td>25</td>
<td>32, 64</td>
<td>4, 8</td>
<td>Yes</td>
<td>No</td>
<td>I2C, I2S, SPI, UART, USART</td>
<td>CSP36, QFN24, QFN32, QFP48</td>
</tr>
<tr>
<td>Tiny Gecko</td>
<td>ARM Cortex-M3</td>
<td>32</td>
<td>4, 8, 16, 32</td>
<td>2, 4</td>
<td>No</td>
<td>Yes</td>
<td>I2C, I2S, SPI, UART, USART</td>
<td>BGA48, QFN24, QFN32, QFN64, QFP48, QFP64</td>
</tr>
<tr>
<td>Gecko</td>
<td>ARM Cortex-M3</td>
<td>32</td>
<td>16, 32, 64, 128</td>
<td>8, 16</td>
<td>No</td>
<td>Yes</td>
<td>I2C, SPI, UART, USART</td>
<td>BGA112, QFN32, QFN64, QFP100, QFP48, QFP64</td>
</tr>
<tr>
<td>Jade Gecko</td>
<td>ARM Cortex-M3</td>
<td>40</td>
<td>128, 256</td>
<td>32</td>
<td>No</td>
<td>No</td>
<td>I2C, I2S, SPI, UART, USART</td>
<td>QFN32, QFN48</td>
</tr>
<tr>
<td>Leopard Gecko</td>
<td>ARM Cortex-M3</td>
<td>48</td>
<td>64, 128, 256</td>
<td>32</td>
<td>Yes</td>
<td>Yes</td>
<td>I2C, I2S, SPI, UART, USART</td>
<td>BGA112, BGA120, CSP81, QFN64, QFP100, QFP64</td>
</tr>
<tr>
<td>Giant Gecko</td>
<td>ARM Cortex-M3</td>
<td>48</td>
<td>512, 1024</td>
<td>128</td>
<td>Yes</td>
<td>Yes</td>
<td>I2C, I2S, SPI, UART, USART</td>
<td>BGA112, BGA120, QFN64, QFP100, QFP64</td>
</tr>
<tr>
<td>Pearl Gecko</td>
<td>ARM Cortex-M4</td>
<td>40</td>
<td>128, 256</td>
<td>32</td>
<td>No</td>
<td>No</td>
<td>I2C, I2S, SPI, UART, USART</td>
<td>QFN32, QFN48</td>
</tr>
<tr>
<td>Wonder Gecko</td>
<td>ARM Cortex-M4</td>
<td>48</td>
<td>64, 128, 256</td>
<td>32</td>
<td>Yes</td>
<td>Yes</td>
<td>I2C, I2S, SPI, UART, USART</td>
<td>BGA112, BGA120, CSP81, QFN64, QFP100, QFP64</td>
</tr>
</tbody>
</table>
ATtiny5
32 B
512 B
12 MHz

ATtiny45
256 B
4 KB
20 MHz

AT90usb162
512 B
16 KB
16 MHz

ATmega328
2 KB
32 KB
20 MHz
2. Overview

The AT90USB82/162 is a low-power CMOS 8-bit microcontroller based on the AVR enhanced RISC architecture. By executing powerful instructions in a single clock cycle, the AT90USB82/162 achieves throughputs approaching 1 MIPS per MHz allowing the system designer to optimize power consumption versus processing speed.

2.1 Block Diagram

Figure 2-1. Block Diagram
<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>R0</td>
<td>0x00</td>
</tr>
<tr>
<td>R1</td>
<td>0x01</td>
</tr>
<tr>
<td>R2</td>
<td>0x02</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>R13</td>
<td>0x0D</td>
</tr>
<tr>
<td>R14</td>
<td>0x0E</td>
</tr>
<tr>
<td>R15</td>
<td>0x0F</td>
</tr>
<tr>
<td>R16</td>
<td>0x10</td>
</tr>
<tr>
<td>R17</td>
<td>0x11</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>R26</td>
<td>0x1A</td>
</tr>
<tr>
<td>R27</td>
<td>0x1B</td>
</tr>
<tr>
<td>R28</td>
<td>0x1C</td>
</tr>
<tr>
<td>R29</td>
<td>0x1D</td>
</tr>
<tr>
<td>R30</td>
<td>0x1E</td>
</tr>
<tr>
<td>R31</td>
<td>0x1F</td>
</tr>
</tbody>
</table>
### Data Memory

<table>
<thead>
<tr>
<th>Type</th>
<th>Address Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>32 Registers</td>
<td>$0000 - $001F</td>
</tr>
<tr>
<td>64 I/O Registers</td>
<td>$0020 - $005F</td>
</tr>
<tr>
<td>160 Ext I/O Registers</td>
<td>$0060 - $00FF</td>
</tr>
<tr>
<td></td>
<td>$0100</td>
</tr>
</tbody>
</table>

**Internal S RAM**

- **Size**: (512 x 8)
- **Address**: $2FF
#include <avr/io.h>

int main(void) {

    DDRD |= (1<<4);

    while (1) {

        if (PIND & (1<<7)) {
            PORTD &= ~(1<<4);
        } else {
            PORTD |= (1<<4);
        }

    }

    return 0;
}
sbi DDRD, 0x04
loop:
sbis PIND, 0x07
rjmp button_up
cbi PORTD, 4
rjmp loop
button_up:
sbi PORTD, 4
rjmp loop