LININO TECHNOLOGY

Virtualizing MCU peripherals

IoT dev room - FOSDEM 2015
Overview

**MPU: MicroProcessor Unit**
- Possibly SMP, large memory, few peripherals, full featured OS.
- Designed for best average performance both in hw and in sw (multiple cache levels, virtual memory, multiple threads/processes, ...): real time is difficult.

**MCU: MicroController Unit**
- Single CPU, small memory, many peripherals, special purpose OS.
- Much simpler: no cache, no VM: worst case execution times can be guaranteed.

**Conclusion**: keep protocols and data processing on the MPU, real time stuff on the MCU.
The Linino technology allows to integrate the MCU’s peripherals into the MPU as if they were standard Linux peripherals.

Currently supported MCUs:
- Atmel Atmega 32u4 (AVR core)
- Freescale KL25z (Cortex M0+)
- Nordic nRF51822 (Cortex M0)

Currently supported MPUs:
- Any MPU running Linux kernel >= 3.3
Linino OS is opensource and it is based on OpenWRT (14.07 branch) distribution. Include about 3000 package built and available. Integrated with Linino IO it is a complete linux system for IoT space.

Linino IO is based on THOS created by Alessandro Rubini. LininoIO is a software framework able to integrate MCU features inside the microprocessor environment. The source code is maintained by the Linino engineers staff and is opensource.
Hardware supported
Linino IO with Node.JS

Overview

- **Node.js** is a platform built on Chrome's JavaScript runtime for easily building fast, scalable network applications. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices.

- **IDEINO-LININO-LIB**
  [https://github.com/ideino/ideino-linino-lib](https://github.com/ideino/ideino-linino-lib)

- **Sysfs** is a feature of the Linux 2.6 kernel that allows kernel code to export information to user processes via an in-memory filesystem. The organization of the filesystem directory hierarchy is strict, and based on the internal organization of kernel data structures.
Linino IO with Node.JS
Live Demo

**BUZZ (pwm)**
```javascript
board.tone(pin, frequency, duration, [callback])
```

**BUTTON (digital input)**
```javascript
board.digitalRead(pin, [callback])
```

**OLED**
```javascript
ctx.font = '13px Pixelade';
ctx.strokeText(data.hum_temperature " C", 5, 40);
ctx.rect(1, 1, 127, 63);
ctx.stroke();
board.display('OLED', ctx, canvas);
```

**LIFX**
```javascript
var lifx = require('lifx');
lx.lightsOn();
lx.lightsColour(hue, saturation, luminance, whiteColour, fadeTime);
```

**DMESG**
```
[ 142.590000] gpiochip_add: registered GPIOs 200 to 215 on device: pca9555
[ 142.930000] sht21 0-0040: initialized
[ 143.790000] ssd1307fb 0-003c: fb0: Solomon SSD1307 framebuffer device registered, using 1024 bytes of video memory
```
Linino IO wireless demo

- MPU is emulated by a PC
- MCU is Nordic nRF51822, Cortex M0
The basic idea, a poor man’s PCI

- Let the MPU see MCU peripherals like "local" devices.
- Virtual MCU peripherals are accessed through memory-mapped registers.
- Addressing based on: bus/device/function/offset

Max. 8 busses
Max. 16 devices per bus
Max. 32 functions per device
Each function has a 4KB memory map
The basic idea, a poor man’s PCI

- Simple protocol: read/write from/to memory mapped registers.
- The MCU can interrupt the MPU (via wire or MSI-like frames).
- The MCU can in principle become "bus master".
- Unlike PCI, no separated configuration space.
- Each virtual peripheral corresponds to a MCUIO function (FN).

Max. 8 busses
Max. 16 devices per bus
Max. 32 functions per device
Each function has a 4KB memory map
The basic idea, a poor man’s PCI Function descriptor

Frames have fixed size (16 bytes) -> easier implementation on MCU

• 16 bytes is the size of a 16550 UART’s FIFO.
• If DMA is available, frame reception is easier to implement if size is fixed.
Linux implementation

The mcuio bus

- The bus concept is the foundation of Linux device management.
- Allows matching devices to drivers according to some specific criteria.
- MCUIO uses device/vendor id or device class as match criteria.
Linux implementation
A line discipline based HC

A line discipline acts as a filter between the hardware and the kernel tty layer.
Next steps

- MCU support:
  - Atmel SAM D21 (Cortex M0+)
  - STM32 (Cortex M3)
  - any other?
- Dynamic peripherals configuration
- Wireless link support
  - BLE support (IPv6)
  - 802.15.4 support
You can find LininoIO code on github:

MPU side (linux kernel): git@github.com:linino/kernel_3.3.8.git
MCU side: git@github.com:linino/bathos-mcuio.git

For any request of information, contact:

Software development mailing list
swdev@doghunter.org

Wiki
http://wiki.linino.org/doku.php

Any contribution is highly appreciated.