Carambola

Embedded Linux Systeme als Microcontroller Alternative

Motivation

In bestimmten Umfeldern bietet sich der Einsatz von Microcontrollern an. Umfassen die Anforderungen z.B.:

1. Integration einfacher Elektronik
2. Geringer Stromverbrauch
3. Betriebssicherheit
4. Geringe Kosten des Einzelsystems
5. Niedriges Gewicht / Kleine Bauform

Grenzen der Microcontroller

- Komplexe Protokolle/Algorithmen
- Integration vieler verschieder Dienste
- Nebenläufigkeit von Aufgaben
- Speicherintensive Aufgaben

In diesen Aufgabenfeldern kann der Einsatz eines Embedded (Linux) Systems sinnvoll sein

Vor- und Nachteile

- ✔ Multitasking OS
- ✔ Protokollunterstützung
- ✔ Logging / Debugging
- ✔ Scriptsprachen

- ✗ Keine „harten“ Echtzeitgarantien
- ✗ Bootzeit
- ✗ Systemkomplexität / Zuverlässigkeit
- ✗ Aufwand für Crosscompiling
- ✗ Bei größeren Stückzahlen: Kosten

Positionierung der Carambola

Als Alternativen bieten sich der bekannte Raspberry PI (B) sowie das Gnublin-Board an. Die Systeme liegen preislich etwa gleich auf, setzen aber unterschiedliche Schwerpunkte:

- Raspberry PI (B): Hervorragende Grafikunterstützung (Full HD), vorbereitete Systemimages für einige Anwendungen. Aktive Community. Hohe Stromaufnahme.
- Gnublin: Gutes Angebot von vorbereiteten Hardwaremodulen, dafür Treiberunterstützung. Mittlere Stromaufnahme. Community etwas träge.
- Carambola: Sehr klein, gut dokumentiert, 2xEthernet + Wlan on Board, gute Treiberunterstützung, geringe Stromaufnahme. Aktive Community und Hersteller-unterstützung. Nachteil: OS im Flash, nicht auf SD.

Systemvergleich

	Flash	RAM	Sonstiges
Arduino (Atmel)	32 K	2 K	
Raspberry (Arm1176)	SD bis 32 G	512 M	HDMI, 2xUSB
Gnublin (Arm9)	SD bis 32 G	32 M	2xUSB
Carambola (RT3050)	8 M	32 M	Wlan b,g,n 2xEth

Erste Schritte (Software)

- Entwicklungssystem einrichten: z.B. xubuntu
- Git Repository einbinden, aktuelles OS-release auschecken (basiert auf openWRT).
- Evtl. Konfigurationsänderungen an Kernel/OS vornehmen
- OS als squashfs-image bauen
- Img in tftp-dir legen

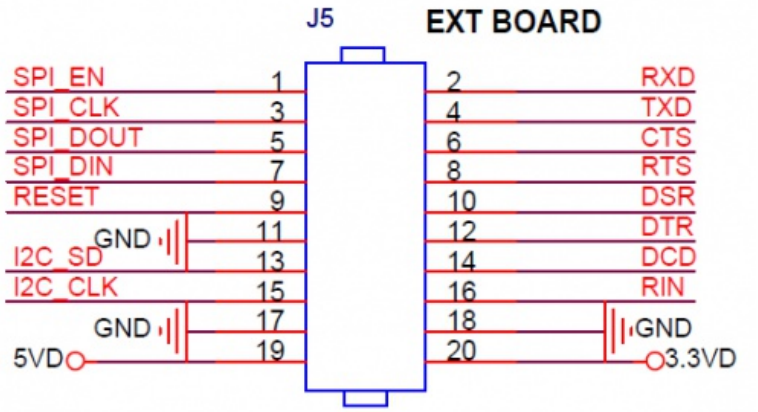
Erste Schritte (Hardware)

- Carambola auf Dev-Board stecken
- Ethernet in LAN-Buchse (neben Power)
- DEV-Board seriell an Entwicklungssystem (115200, 8N1), KEIN Nullmodem-Kabel
- Terminalprogramm an Serielle Konsole
- Netzteil an Dev-Board (9-12V, 500mA)
- „2“ Bei der Auswahl der Bootmethode (Neues Image in Flash via tftp)
- IP von Board und tftp-Server angeben, imgfile

Hello World

An Pin 10 des Devboards hängt eine LED, die über einen Vorwiderstand gegen Pin 11 geschaltet ist. In der Dokumentation findet sich, dass Pin 10 mit GPIO13, und Pin 11 mit Masse verbunden ist. Um die LED an- und wieder auszuschalten kann man folgendermassen vorgehen:

- cd /sys/class/gpio
- echo 13 > export
- cd gpio13
- echo out > direction
- echo 1 > value
- echo 0 > value



JTAG_GPIO_MODE description

Pin Name	JTAG_GPIO_MODE=0	JTAG_GPIO_MODE=1
JTAG_TRST_N	JTAG_TRST_N	GPIO21
JTAG_TCLK	JTAG_TCLK	GPIO20
JTAG_TMS	JTAG_TMS	GPIO19
JTAG_TDI	JTAG_TDI	GPIO18
JTAG_TDO	JTAG_TDO	GPIO17

UARTL_GPIO_MODE description

Pin Name	UARTL_GPIO_MODE=0	UARTL_GPIO_MODE=1
RXD2	RXD2	GPIO16
TXD2	TXD2	GPIO15

UARTF_SHARE_MODE description

Pin Name	3'b000 UARTF	3'b001 PCM, UARTF	3'b010 PCM, I2S	3'b011 I2S UARTF	3'b100 PCM, GPIO	3'b101 GPIO, UARTF	3'b110 GPIO I2S	3'b111 GPIO
RIN	RIN	PCMDTX	PCMDTX	RXD	PCMDTX	GPIO14	GPIO14	GPIO14
DSR_N	DSR_N	PCMDRX	PCMDRX	CTS_N	PCMDRX	GPIO13	GPIO13	GPIO13
DCD_N	DCD_N	PCMCLK	PCMCLK	TXD	PCMCLK	GPIO12	GPIO12	GPIO12
DTR_N	DTR_N	PCMFS	PCMFS	RTS_N	PCMFS	GPIO11	GPIO11	GPIO11
RXD	RXD	RXD	REFCLK	REFCLK	REFCLK	RXD	REFCLK	GPIO10
CTS_N	CTS_N	CTS_N	I2SSD	I2SSD	GPIO9	CTS_N	I2SSD	GPIO9
TXD	TXD	TXD	I2SWS	I2SWS	GPIO8	TXD	I2SWS	GPIO8
RTS_N	RTS_N	RTS_N	I2SCLK	I2SCLK	GPIO7	RTS_N	I2SCLK	GPIO7

SPI_GPIO_MODE description

Pin Name	SPI_GPIO_MODE=0	SPI_GPIO_MODE=1
SPI_DIN	SPI_DIN	GPIO6
SPI_DOUT	SPI_DOUT	GPIO5
SPI_CLK	SPI_CLK	GPIO4
SPI_EN	SPI_EN	GPIO3

I2C_GPIO_MODE description

Pin Name	I2C_gpio_mode=0	I2C_gpio_mode=1
I2C_SCLK	I2C_SCLK	GPIO2
I2C_SD	I2C_SD	GPIO1

I2C Basic

- I2C (TWI) Tools installieren via make menuconfig, utilities, i2ctools
- Dann i2cdetect 0 aufrufen
- Nach Anschluss eines i2c Devices findet i2cdetect die Adresse auf dem Bus.
- Einfache I2C Implementierung unter Python (Subset SMBus), leider Installation nicht OK, daher keine Demo.

Zusammenfassung

- Einsatzzweck genau evaluieren, danach Plattform aussuchen
- Entwicklungssystem berücksichten Compilerläufe auf dem Embedded System sind unpraktikabel
- Bussysteme nutzen wo immer möglich, da oft fertige Libraries vorhanden
- Implementierungsaufwand gegen Hardwareaufwand / Betriebskosten abwägen

Vielen Dank!

http://www.raspberrypi.org/

http://gnublin.embedded-projects.net/

http://8devices.com/carambola

http://openwrt.org/

http://arduino.cc/

http://jeelabs.org/

http://www.atmel.com/

http://www.microcontroller.net/