Linux as an Embedded Operating System

Linux is a popular, UNIX compatible, open source operating system that was designed originally for desktop computers. In a first big success wave Linux conquered the servers in the enterprises. Whether as an operating system for a file server or on the Internet Access gateway, Linux offers the necessary features and the absolutely necessary stability for the unattended continuous operation without reboots. Now Linux is taking the embedded system market by storm.

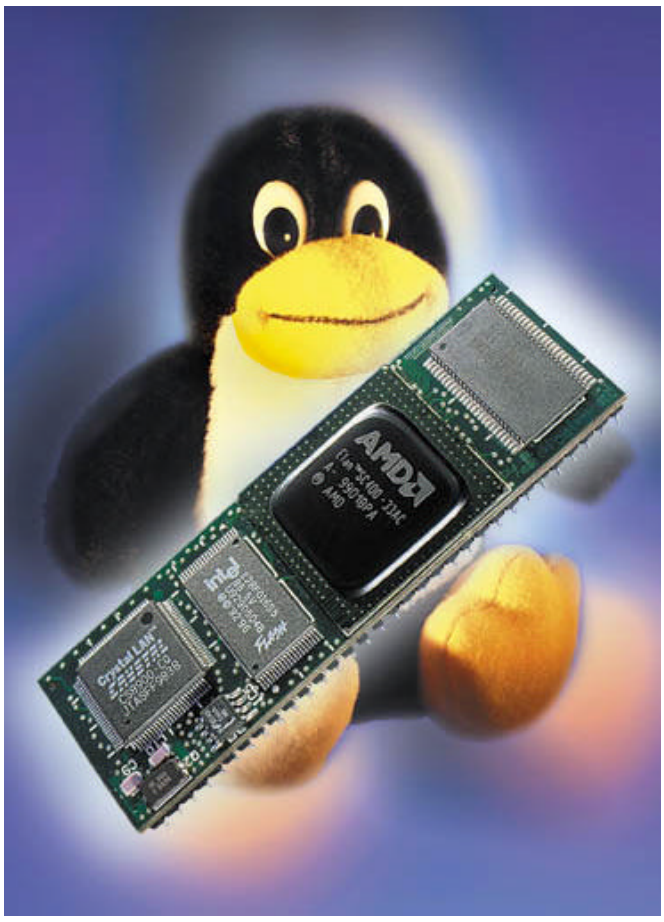

Figure 1: The DIL/NetPC with 64-pin IC Pinout

Introduction

Linux is a modular operating system that is available in the source code. Runtimes of Linux are royalty free. Linux is documented outstandingly and offers a robust multitasking operation as well as an advanced programming interface (API). Extensive professional relief broadcasts it numerous independent companies and a worldwide developer municipality that advances the operating system and helps

about the Internet with tips and pieces of advice mutually with almost every problem. Linux supports almost all 32-bit processor architectures as well as meanwhile also some low-cost 32-bit microcontrollers and offers device drivers for any piece of hardware. Linux implies extensive functions for networking tasks as the TCP/IP- records of the Internet. Through these features Linux is suitable for applications in the automatic control area and in the field of measurement and control. Particularly as a so-called "embedded operating system" within networked automation components as for example a distributed industrial operation it can play all advantages. In order to support the circulation of Linux as an embedded operating system, in first days of 2000 the "Embedded Linux Consortium (ELC)" in the USA and the "Emblix Consortium" in Japan were setting up to work. Both institutions have important finance averages through their numerous industrial members and want Linux through purposeful press job, fair appearances and over a strong Internet-presence to be the leading embedded operating system in the future.

Also the real-time ability, necessary for many embedded applications, now subsequently receives Linux. Several companies and institutes pursue with different approaches the destination to allow the commitment of Linux in time-critical applications. One of these evolutions named RTLinux settles directly about the hardware as real-time kernels below the standard Linux and lets the standard Linux run as an own task. Through that standard Linux does not become indeed real time capable. It is, however, possible, to let a task run with guaranteed response times about the real-time kernel. For another approach the entire one became Linux kernel re-works around the desired real-time ability to receive.

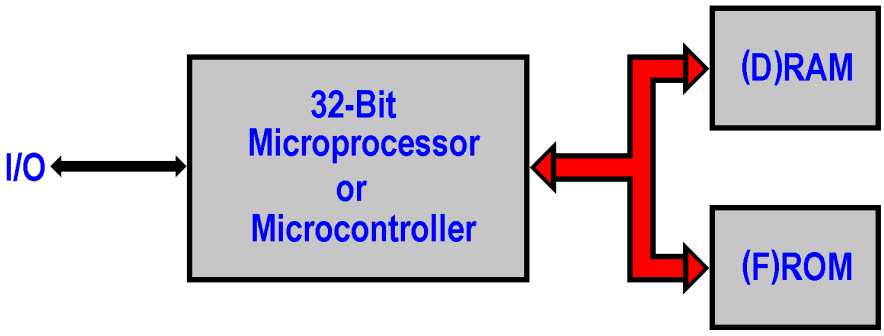

Figure 2: Block Diagram for a minimum Linux Hardware

Linux can be used onto platforms with minimum resources. Some small embedded devices need already three integrated circuits (ICs) for running a embedded Linux: 1. a 32-bit microcontroller or microprocessor with or without MMU (Memory Management Unit), 2. a FLASH- memory chip for the operating system and the application-specific programs as well as 3. a (D)RAM as main memories and RAM- disk for the root filesystem. Figure 2 shows the block diagram for a small Linux hardware. As an example of a typical small embedded system for embedded Linux the DIL/NetPC is supposed to serve here [1]. This miniature module with a 10BASE-T Ethernet interface brings industrial automation, medical instruments and measurement and control systems to Ethernet-based networks. As a mechanical base, the DIL/NetPC is using the JEDEC standard format of a 64-pin "Dual-In-Line (DIL)“ integrated circuit case.

The DIL/NetPC - A Example for a minimum Linux Hardware

The DIL/NetPC central functional unit is formed by a 32-bit microcontroller SC410 of AMD. This circuit offers in a 292-pin BGA (Ball Grid Array) case a AM486SX processor core with 33, 66 or 99 (100) MHz and all functional units of a typical PC/AT personal computer architecture. In addition to that the SC410 contains one 16C450/16C550 compatible serial port with IrDA interface and some general purpose parallel I/O lines.

The DIL/NetPC offers two memory devices: one 8 MByte DRAM as working memory and for the Linux root filesystem on a RAM disk and one 2 MByte FLASH for storing the embedded operating system together with the application programs. The DIL/NetPC 10BASE-T Ethernet interface is build with a CS8900 LAN controller. The chip also contains the necessary Ethernet packet buffer memory.

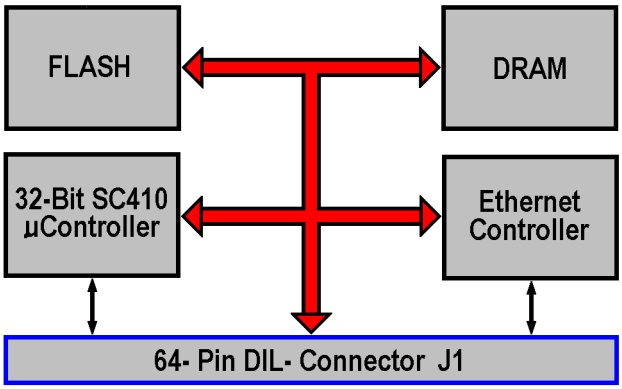

Figure 3: DIL/NetPC Block Diagram

In order to mount all the components of the DIL/NetPC to a board with only 82 by 28 mm, a 8-layer printed circuit board is used. The bottom of this multilayer printed circuit board offers a 64-pin connector in the format of a DIL-IC for mechanical integration to existing systems. At the 64 pin's, the DIL/NetPC offers 20 parallel I/O (PIO) signals, a complete serial PC-based COM1 interface with all handshake signals, the 10BASE-T Ethernet interface and a 8-bit I/O- extension bus with four programmable chip select signals and five interrupt input lines. Thus the accessibility to existing systems might not represent any great problem. In the simplest case the combination over COM1 occurs, at more pretentious solutions about that one I/O bus. This bus can for example A/D- and/or D/A to access transformers or other controllers.

Also an accessibility of LCD panels (text- and/or small graphic module versions) is possible without additional logic.

Example for a Embedded Linux Configuration

Fundamentally an embedded Linux consists of three basic modules: 1. the bootloader, 2. the actual Linux kernel and 3. the root filesystem. There can be

these three base elements in the form of only one binary image (Figure 4.a) or also as separate files (Figure 4.b). Even the possibility consists in this case then, that embedded Linux can be start direct from a DOS command line.

```
@echo off
ECHO Start SSV Embedded Linux for DILNetPC.
loadlin zimage console=ttys0,115200 initrd=rimage.gz
ECHO SSV Embedded Linux stopped or failed !
```

Listing 1: Starting embedded Linux with a DOS command line

For the DIL/NetPC, a compact embedded Linux hardware platform, the manufacturer supplies the three files LOADLIN.EXE as bootloader, ZIMAGE as kernel and the root filesystem as a file named RIMAGE.GZ. The Linux kernel is stored as compressed binary code which unpacks itself automatically direct after the system starts. An example is shown in listing 1, as kernel and file system are to be started by DOS command line. The command line parameters are especially important. Here the kernel (ZIMAGE) is informed by the command line parameters that a COM1-based serial console (console=ttyS0, 115200) with the transmission speed of 115 kbps is used for console I/O. In order to get into contact with the DIL/NetPC embedded Linux, a serial ASCII terminal or a PC with a terminal emulation program can be used.

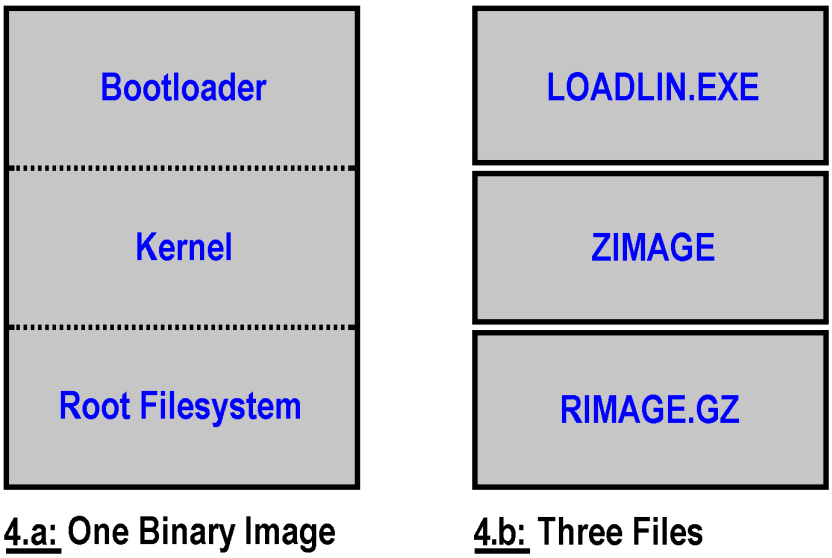

Figure 4: Basic Parts of a Embedded Linux

In the DIL/NetPC a 2 MByte FLASH memory chip an embedded Linux with TCP/IP support, Telnet, FTP and Web server as firmware is stored. Within the framework of the Linux porting process only the components and drivers which are required by the hardware was moved to the kernel and the root filesystem. This kernel was combined together with the root filesystem and a modified bootloader into one binary file (Binary Image - see Figure 4.a). Table 1 shows the size of the basic parts for the DIL/NetPC embedded Linux**.**

Within the booting process the DIL/NetPC Linux kernel is responsible for setting up all the normal hardware configuration of the SC410 microcontroller and peripherals, things such as internal and external chip selects, DRAM controller, internal counter/timer, interrupt controller etc. After that, the Linux kernel loads all the necessary drivers for the serial console, Ethernet controller and so on.

Component/Module	**Size**
Bootloader	10.819 Bytes
Kernel with TCP/IP Support	418.978 Bytes
Root Filesystem incl. Telnet, FTP and Web Server	1.098.921 Bytes
	1.528.718 Bytes

Table 1: Example of the DIL/NetPC Linux Configuration.

After the hardware setup the embedded Linux kernel builds a RAM disk within the DIL/NetPC DRAM chip. Then the kernel decompresses the root filesystem and forms the binary image in FLASH memory to the new empty RAM disk. This provides a significant performance advantage. The access to DRAM is faster than typical FLASH memory. After the file system is complete the kernel causes the so-called "systeminit process". For that process the program `init` is started in the RAM disk subdirectory `/sbin`. This program activates further programs under circumstances in accordance to the content the configuration file `/etc/inittab`.

Networking with Linux

Linux offers a strong TCP/IP protocol suite [2] with BSD socket interface. The TCP/IP protocol suite started as a research project for the Department of Defence (DoD) in the USA as a way of sending data so that a single failure does not cripple the communication between two points. When a network connection either becomes congested or is made inactive, data for that link are routed in a different way. This network grew into what is known as the Internet. The TCP/IP protocol stack manages the assembling of data into packets. The packet are transmitted over the Internet and received at the destination where the packets are reassembled to their original state.

Over the Linux BSD socket interface (API= Application Programming Interface) the application programmer can access direct to the TCP and UDP protocols [3] within the DIL/NetPC TCP/IP stack for own applications.

Above the BSD socket interface numerous TCP/IP networking applications are to be found. Within the DIL/NetPC embedded Linux configuration, there are the Telnet, FTP and Web server as the most important TCP/IP applications.

Telnet is a protocol used to implement a remote login facility on virtually any host computer from a remote terminal. The idea is that a terminal creates a session on a remote server anywhere in a network or internetwork (i.e. the Internet). Because terminals and hosts can vary in terms of the functionality provided, the Telnet protocol was designed to enable the host and terminal to negotiate additional

options to augment the facilities offered to the user. For using the Telnet protocol, the host needs a Telnet server and the terminal needs a Telnet client software. The DIL/NetPC Linux Telnet server offers a complete Ethernet-based remote login facility for any Telnet client for example the standard telnet client within a Microsoft Windows operating system. Figure 5 shows a Windows 98 Telnet client connected to the DIL/NetPC Telnet server.

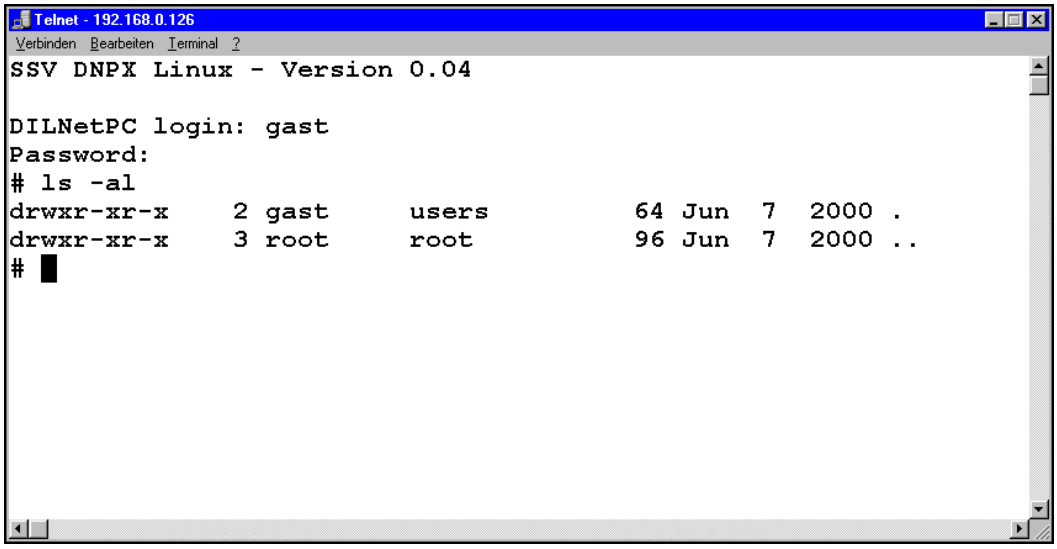

Figure 5: Telnet Session on the DIL/NetPC

FTP (File Transfer Protocol) provides a common approach of transferring files between clients and servers. The DIL/NetPC Linux FTP server is used for Ethernet-based file transfers to and from the RAM disk. The FTP client on the other end can be running on a different operating system. Most desktop operating systems includes a FTP client. Microsoft, for example, includes a command line FTP client with Windows 95, 98, NT and 2000 operating system. This client can be used for transferring files between a desktop PC and the DIL/NetPC.

The Web server within the DIL/NetPC embedded Linux configuration offers a embedded home page with a graphical user interface (GUI) for any device or system, which contains a DIL/NetPC. This GUI is then accessible from a PC or any other computer with a standard Web browser over a Ethernet-based network and over the Internet for Web-based device monitoring.

Software Development for Embedded Linux

Most Linux software development tasks for embedded systems are done in a cross development environment. This environment consists of a host and a target. Typical the host is a standard desktop PC with a Linux operating system. This systems forms the development system. The target is a embedded system like the DIL/NetPC. Figure 6 shows the block diagram of this cross software development environemt.

Most desktop PCs runs under the Microsoft Windows operating systems. In this case Linux should simply be installed as a second operating system to the PC hard

disk. Together with this Linux distributions comes the GNU tool chain with compilers, assembler, linker and debuggers. Programs are written on the development system. After each compiler/linker run, the binary output is downloaded to the target for testing. Some parts of a embedded system application can be developed natively. In this case the development system and the embedded system needs the same CPU type (i.e. the AMD or Intel x86 architecture).

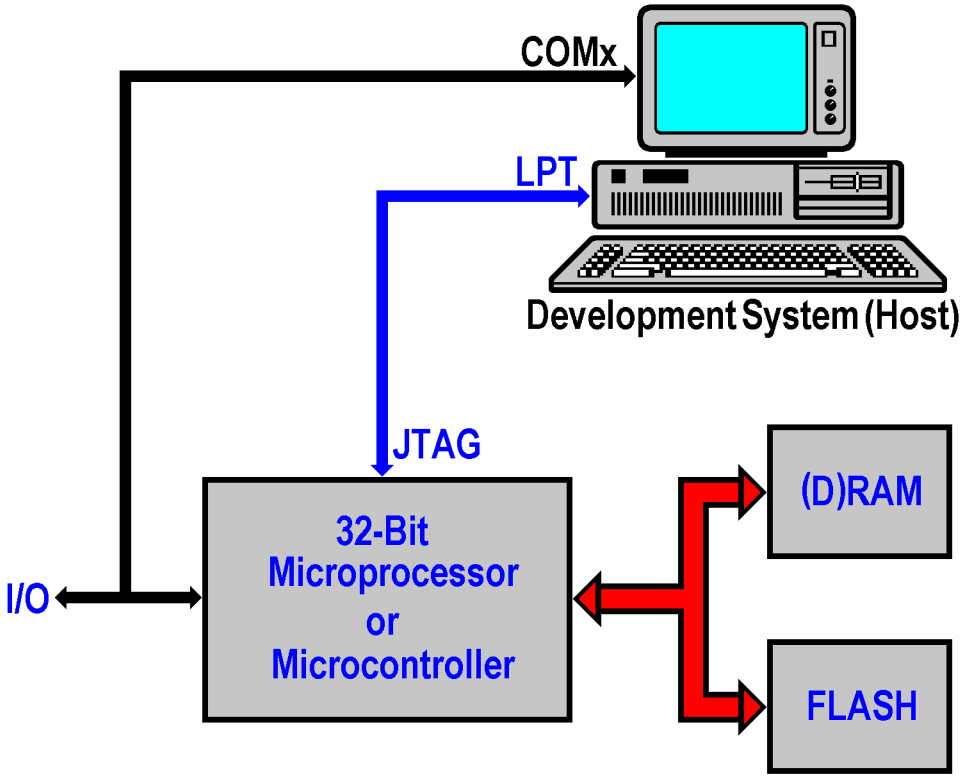

Figure 6: Software Development Environment

For downloading to the target and debugging the embedded system application on the target, the development environment needs a link between host and target. Figure 6 shows two links: one serial link over RS232 for the debugger and second link between the PC parallel port (LPT) and the JTAG interface of the 32-bit microprocessor or microcontroller. Over this interface, the PC can write a binary image direct into the embedded system FLASH memory.

An enormous assistance during the software development forms an integrated programming environment (IDE). Such a tool allows the use of editor, compiler, linker and debugger from a homogeneous surface. Modern IDEs combine frequently also debugger and editor, so that for example a breakpoint for the debugger over the editor directly set into the C source code. An IDE manages all files and libraries of a project in a hierarchical list. Today several very strong integrated programming environments for Linux are available. Typical representatives are the open source project Kdevelop as well as Sniff++ and Metrowerks Codewarrior [4]. Most Linux IDEs are delivered, however, without compilers and so forth. This IDEs works with the GNU tool chain.

Appendix 1: Pinout 64-pin Connector DNP/1486-3V (1. Part)

Pin	Name	Gruppe	Funktion
1	PA0	PIO	Parallel I/O, Port A, Bit 0
2	PA1	PIO	Parallel I/O, Port A, Bit 1
3	PA2	PIO	Parallel I/O, Port A, Bit 2
4	PA3	PIO	Parallel I/O, Port A, Bit 3
5	PA4	PIO	Parallel I/O, Port A, Bit 4
6	PA5	PIO	Parallel I/O, Port A, Bit 5
7	PA6	PIO	Parallel I/O, Port A, Bit 6
8	PA7	PIO	Parallel I/O, Port A, Bit 7
9	PB0	PIO	Parallel I/O, Port B, Bit 0
10	PB1	PIO	Parallel I/O, Port B, Bit 1
11	PB2	PIO	Parallel I/O, Port B, Bit 2
12	PB3	PIO	Parallel I/O, Port B, Bit 3
13	PB4	PIO	Parallel I/O, Port B, Bit 4
14	PB5	PIO	Parallel I/O, Port B, Bit 5
15	PB6	PIO	Parallel I/O, Port B, Bit 6
16	PB7	PIO	Parallel I/O, Port B, Bit 7
17	PC0	PIO	Parallel I/O, Port C, Bit 0
18	PC1	PIO	Parallel I/O, Port C, Bit 1
19	PC2	PIO	Parallel I/O, Port C, Bit 2
20	PC3	PIO	Parallel I/O, Port C, Bit 3
21	RXD	SIO	COM1 Serial Port, RXD Pin
22	TXD	SIO	COM1 Serial Port, TXD Pin
23	CTS	SIO	COM1 Serial Port, CTS Pin
24	RTS	SIO	COM1 Serial Port, RTS Pin
25	DCD	SIO	COM1 Serial Port, DCD Pin
26	DSR	SIO	COM1 Serial Port, DSR Pin
27	DTR	SIO	COM1 Serial Port, DTR Pin
28	RI	SIO	COM1 Serial Port, RI Pin
29	RESIN	RESET	RESET Input
30	TX+	LAN	10BASE-T Ethernet Interface, TX+ Pin
31	TX-	LAN	10BASE-T Ethernet Interface, TX- Pin
32	GND	----	Ground

Table 2a: Pinout Pin 1 to 32

Appendix 1: Pinout 64-pin Connector DNP/1486-3V (2. Part)

Pin	Name	Gruppe	Funktion
33	RX+	LAN	10BASE-T Ethernet Interface, RX+ Pin
34	RX-	LAN	10BASE-T Ethernet Interface, RX- Pin
35	RESOUT	RESET	RESET Output
36	VBAT	PSP	SC410 Real Time Clock Battery Input
37	CLKOUT	PSP	Clock Output (Default 1.8432 MHz)
38	IRTXD	PSP	SC410 IrDA TXD Pin
39	IRRXD	PSP	SC410 IrDA RXD Pin
40	INT5	PSP	Programmable Interrupt Input 5
41	INT4	PSP	Programmable Interrupt Input 4
42	INT3	PSP	Programmable Interrupt Input 3
43	INT2	PSP	Programmable Interrupt Input 2
44	INT1	PSP	Programmable Interrupt Input 1
45	CS4	PSP	Programmable Chip Select Output 4
46	CS3	PSP	Programmable Chip Select Output 3
47	CS2	PSP	Programmable Chip Select Output 2
48	CS1	PSP	Programmable Chip Select Output 1
49	IOCHRDY	PSP	I/O Channel Ready
50	IOR	PSP	I/O Read Signal, I/O Expansion Bus
51	IOW	PSP	I/O Write Signal, I/O Expansion Bus
52	SA3	PSP	I/O Expansion Bus, Address Bit 3
53	SA2	PSP	I/O Expansion Bus, Address Bit 2
54	SA1	PSP	I/O Expansion Bus, Address Bit 1
55	SA0	PSP	I/O Expansion Bus, Address Bit 0
56	SD7	PSP	I/O Expansion Bus, Data Bit 7
57	SD6	PSP	I/O Expansion Bus, Data Bit 6
58	SD5	PSP	I/O Expansion Bus, Data Bit 5
59	SD4	PSP	I/O Expansion Bus, Data Bit 4
60	SD3	PSP	I/O Expansion Bus, Data Bit 3
61	SD2	PSP	I/O Expansion Bus, Data Bit 2
62	SD1	PSP	I/O Expansion Bus, Data Bit 1
63	SD0	PSP	I/O Expansion Bus, Data Bit 0
64	VCC	----	3.3 Volt Power Input

Table 2b: Pinout Pin 33 to 64

Literature

[1] Web Site of SSV Embedded Systems. www.ssv-embedded.de
[2] Robin Burk a.o.: TCP/IP Blueprints. SAMS PUBLISHING 1997. Page 157 ff.
[3] Fischer/Müller: Netwerkprogrammierung unter Linux. Carl Hanser 1999.
[4] Web Site of Metrowerks. www.metrowerks.com

Contact

SSV Embedded Systems
Heisterbergallee 72
D-30453 Hannover
Germany
Tel. +49-(0)511-40000-0
Fax. +49-(0)511-40000-40
Email: sales@ist1.de
Web: www.ssv-embedded.de

Notes to this Document (Emblinx3.Doc)

Revision	Date		Name
1.00	31.10.2000	First Version based on a Call for Paper.	KDW