WISCORE

NET-Start!
User's Guide

Revision: 2.4a
Mar, 2004

Copyright

Trademarks

Patents

Disclaimer

Publisher

WISCORE Inc.
6F, No. 180, Sec. 2, Duenhua S. Rd.,
Taipei, Taiwan 106, R.O.C.
Tel: +886(0)2 8732 8380
Fax: +886(0)2 8732 8381
http://www.wiscore.com

Contents

1. Overview

1.1. Introduction

NET-Start!™ is a compact networking startup kit for embedded system developers and students to evaluate ARM™ technologies and embedded Linux solutions. The system is designed based on a 32-bit ARM7TDMI™ network processor - S3C4510™. S3C4510™ is a high performance and low cost solution for network applications. It has been widely used in a variety of network equipments. The central processor core of the Samsung Electronics' S3C4510 is ARM7TDMI, which is a low power 32-bit RISC macro-cell incorporating Thumb™ 16-bit compressed instruction set. With Thumb™ feature, the system cost can be saved by reducing the required memory size and achieving 32-bit system performance from 16-bit memories.

In addition, the S3C4510 processor features a configurable unified 8-Kbyte cache/SRAM, an I^2C serial interface, two UARTs, two timers, 18 programmable I/O ports, and a 10/100BaseT Ethernet controller. These on chip features also significantly reduce the total system cost of networking devices.

The NET-Start! contains all the hardware and software development tools needed to build your network applications. It contains complete schematics, GNU development tools, uClinux kernel source codes, and the source code of application programs. By integrating the stability and open source advantages of Linux with Samsung Electronics' cost-effective and high performance microprocessor, you can start your networking applications development faster.

1.2. Board Supporting Package Contents

The NET-Start! Board Supporting Package is a complete ARM embedded Linux development platform. It includes all the components required to evaluate the ARM based system, TCP/IP networking, and embedded Linux technologies. The kit contains the following components:

- The NET-Start! Evaluation Board.
- The NET-Start! Board Software and Documentation CD-ROM.
- A 9-pin null-modem RS-232C serial cable.
- A twisted-pair UTP Ethernet cable.
- A 12V wall-mounted power adapter.

1.3. Hardware Specifications

The NET-Start! evaluation board contains the following components:

- Samsung Electronics' S3C4510B network processor running at 50MHz
- 2MB Flash ROM (1M x 16bit)
- 16MB SDRAM (2 x 4M x 16bit)
- 9-pin D-sub RS-232C serial console port
- 9-pin D-sub full RS-232C electricity serial port
- RJ-45 10/100 Base-T Ethernet interface
- 36 macro cells high performance CPLD
- Real time clock with charger control
- Watchdog with power failure detection
- Two programmable output LEDs
- One 7-Segment LED display
- Six user input DIP switches
- Two general purpose push buttons
- Multi-ICE connector
- 16 x 2 / 20 x 2 dot matrix LCD module connector (optional)

1.4. Software Specifications

- GNU tool chain, including ARM cross compiler, Linker, Assembler, and Utilities
- uClinux kernel source and image
- Library (uClibc)
- System & Network utilities (busybox)
- Web Server (thttpd)
- FTP client
- DHCP client (dhcpcd)

1.5. Trademarks

S3C4510 is a registered trademark of Samsung Electronics Co., Ltd.

Other trademarks are the properties of their respective owners.

2. Getting started with NET-Start!

2.1. Connection

NET-Start! is designed for evaluation and experimentation in conjunction with ARM-Linux technologies. Figure 2-1 shows the connection of the NET-Start! software development environment.

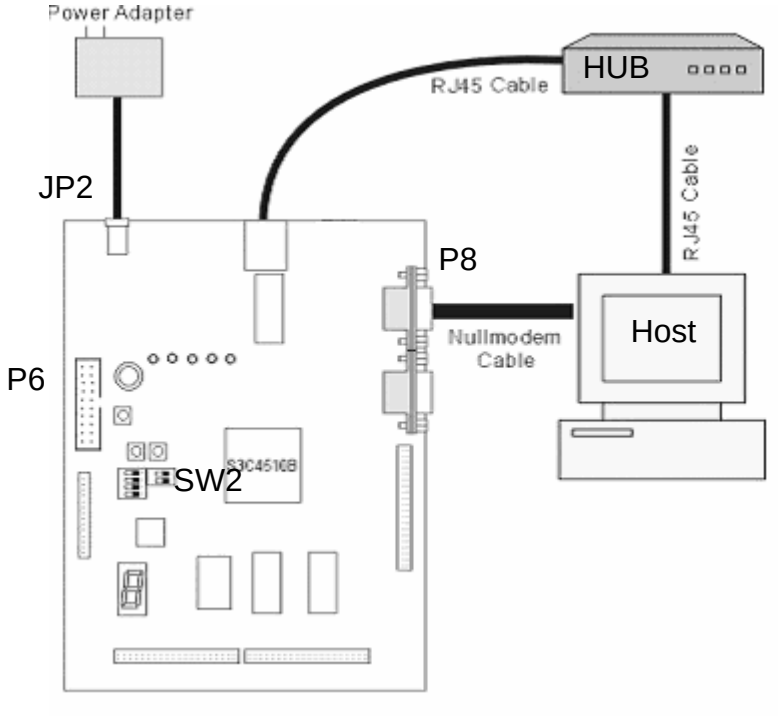


Figure 2-1 NET-Start! setup configuration

2.2. Configuring and Powering Up

2.2.1. Working with NET-Start! for the first time

Please refer to Figure 2-1 for the connector location first, and then follow the steps below:

Step 1:

Configure a host computer (PC) as a terminal (i.e. in minicom or HyperTerm) with the properties set as 19200 bauds, 8 bits of data, no parity, one stop bit, and none flow control. Connect the supplied RS232 cable between a serial port of the host computer and connector (P8) of the NET-Start! evaluation board.

Here we only introduce two commonly used tools: minicom (under Linux) and HyperTerm (under MS-Windows).

Configuring **HyperTerm**:

1. Launch HyperTerm, a 'New Connection' window will be popped up.
2. Enter a name, and Press 'OK'
3. Select a COM port.
4. Set BPS to '19200', Bit to '8', Parity to 'None', Stop-bit to '1', and Flow-Control to 'None'
5. Save the setting.
6. exit

Configuring **Minicom**:

1. Switch to root (the superuser)
2. Type 'minicom -s'
3. Move the cursor bar to 'Serial port setup' and press 'ENTER'
4. Press 'A' to type your COM port device. (ex. “/dev/ttyS0”)
5. Press 'E' to change the communication parameters
6. Press 'F' and 'Q' to select '19200 bps' and '8N1', respectively
7. Press 'ENTER' to accept the setting
8. Press 'F' to set Hardware Flow Control to 'No'
9. Press 'ENTER' to accept the setting
10. Save setup as df1
11. Exit

Step 2:

Connect the supplied power adapter to connector JP2 on the board.

Step 3:

Set the turn on mode by setting switch 2 of SW2.

On: Boot up into the bootstrap loader command mode

Off: Boot up into the pre-compiled uClinux Kernel

Step 4:

Turn on the power adapter.

Step 5:

In the terminal of the host computer, the following prompt will be shown:

The bootstrap loader mode: 'RUN>'

The uClinux Kernel mode: '#'

2.2.2. Connect NET-Start! to network

Besides the basic configuration described in the previous section, NET-Start! evaluation board can be connected to the network by plugging the RJ-45 Ethernet cable into port JP1. If you want to connect the host computer with the board directly, you can use a twisted-pair UTP Ethernet cable to do this.

2.2.3. Connect NET-Start! with an Multi-ICE / JTAG based Emulator

To connect the NET-Start! evaluation board with the ICE emulator, please plug the JTAG cable into connector P6 before turning on the power adapter. For details about the configuration, please refer to the user's guide of the ICE emulator.

3. Hardware Description

3.1. System Block Diagram

NET-Start! is a complete ARM based networking platform that includes a minimal set of core facilities. It is a powerful and flexible evaluation and experimentation tool in conjunction with ARM-Linux technologies. Figure 3-1 shows the layout of the NET-Start! evaluation board.

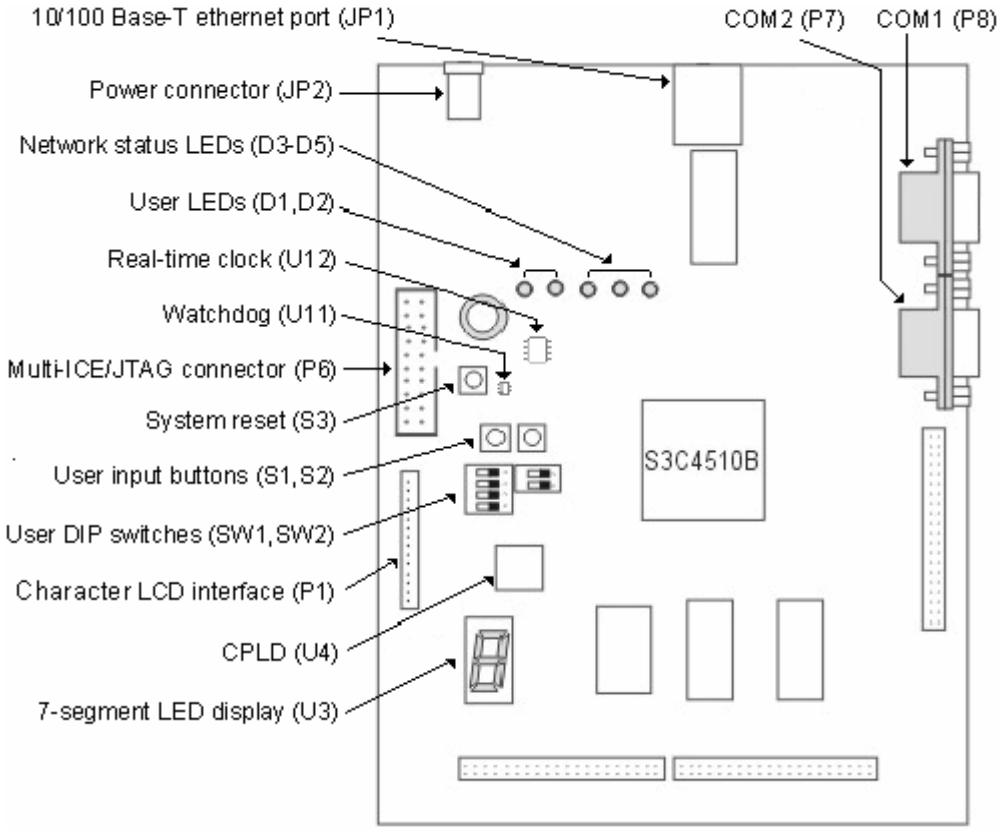


Figure 3-1 Layout of the NET-Start!

Figure 3-2 shows the block diagram of the NET-Start! evaluation board. The major components are described in the following sections.

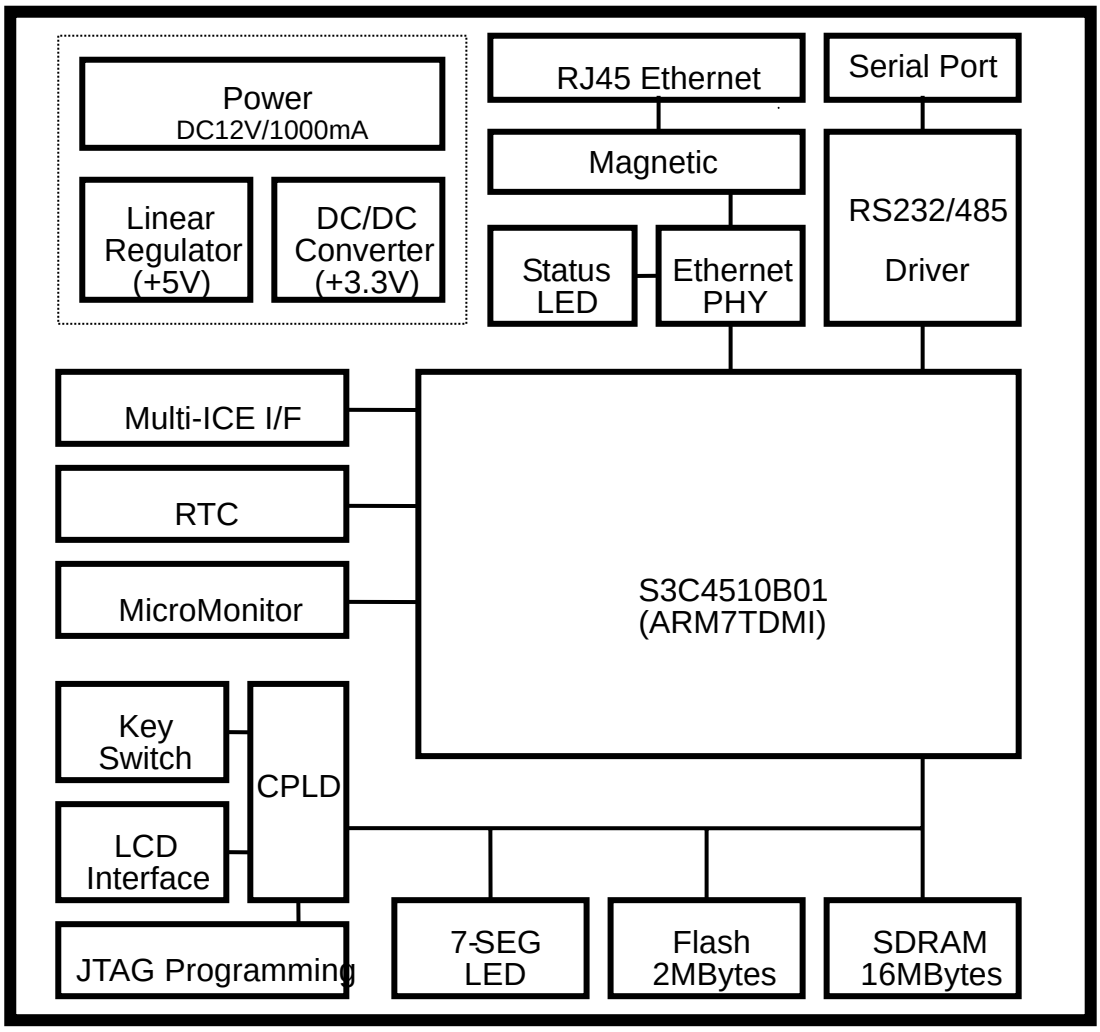


Figure 3-2 NET-Start! block diagram

3.2. The Samsung S3C4510 microprocessor

The S3C4510 is a square, 208-pin Quad Flat Pack (QFP), embedded network processor manufactured by Samsung Electronics Co., Ltd. It is an ARM7TDMI microprocessor incorporating a number of on-chip functions listed below:

- 8KB unified cache/SRAM
- 10/100Mbps Ethernet controller
- Memory controller providing byte / half-word / word external bus support for ROM/SRAM, Flash ROM, SDRAM, DRAM, and external input/output
- Two programmable 32-bit timers
- One Interrupt controller
- Two DMA controller channels
- High-level Data Link Control (HDLC) supported
- Two UARTs
- 18 programmable input/output bit ports
- Master I^2C serial interface

3.3. Flash ROM

The NET-Start! evaluation board includes 2M bytes Flash ROM, which contains bootstrap loader, pre-compiled Linux kernel, Ethernet MAC addresses, and an initial ram disk image. The remaining space is also available for your application programs. The Flash ROM is implemented as a single half-word size (16-bit) device and is mapped to ROM BANK0 of the Samsung S3C4510 microprocessor.

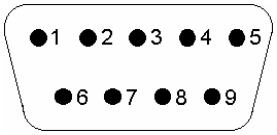
3.4. Synchronous DRAM

The evaluation board contains 16M bytes SDRAM for storing program stack and variables. Two SDRAM chips are used together to provide 32-bit data accesses. They are mapped to the DRAM BANK0 of the Samsung S3C4510 microprocessor.

Though the S3C4510 microprocessor has 8K bytes unified cache/SRAM, it was configured as cache memory to increase the system performance. To use all or part of this memory as a zero wait state internal SRAM, you have to reconfigure the System Manager register explicitly.

3.5. Serial ports

Two UARTs (Universal Asynchronous Receiver / Transmitter) provide simple communications to PC and attached modem through serial ports. In conjunction with the programmable GPIO pins from the S3C4510 microprocessor, the port COM2 also can be **simulated** as a full RS-232C electricity DTE interface to the modem. The pin definition of the serial port connector is shown below.



Pin	Name	Description	COM1/P8	COM2/P7
1	DCD	Data Carrier Detect	NC	Connected
2	RXD	Receive Data	Connected	Connected
3	TXD	Transmit Data	Connected	Connected
4	DTR	Data Terminal Ready	Connected	Connected
5	GND	Ground	Connected	Connected
6	DSR	Data Set Ready	Connected	Connected
7	RTS	Request To Send	NC	Connected
8	CTS	Clear To Send	NC	Connected
9	RI	Ring Indicator	NC	Connected

Figure 3-3 Pin definition of the RS-232C serial port connector (P8, P7)

3.6. Switches and user input buttons

The six DIP switches and two buttons are independent and further divided into two groups. The four switches of SW1 and the buttons S1 and S2 are connected to the External I/O Bank 2 of the S3C4510 microprocessor. The other two switches within SW2 are directly connected to the GPIO pins of the S3C4510 microprocessor. Figure 3-4 shows the circuit of the DIP switches and buttons. Selecting the ON position of DIP switch will pull the corresponding input to LOW. The software program can read its status from the External I/O bank 2 (SW1 & buttons) or GPIO [2:3] (SW2).

As shown in Figure 3-4, the SW1 DIP switches and buttons S1 and S2 are connected to the S3C4510 microprocessor through a CPLD (Complex Programmable Logic Device). The CPLD is used to route the input buttons and switches at the External I/O bank 2 and to support the interface of the character based LCD module at the External I/O bank 0. (See Section 3.10.)

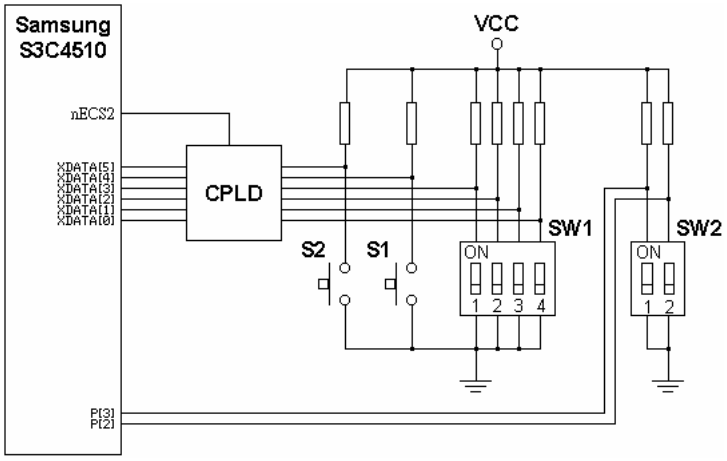


Figure 3-4 Schematic of DIP switches and user input buttons

3.7. LEDs

There are five LEDs on the NET-Start! evaluation board. Three of them, D3 to D5, are the network status indicators from the DM9161 Ethernet PHY. The other two are user controllable and are connected to the GPIO [16:17] of the S3C4510 network processor. Figure 3-5 lists the description of these five LEDs. The circuit of the user controllable LEDs is shown in Figure 3-6.

LED	Name	Description
D1	TOUT0	User controllable LED 0
D2	TOUT1	User controllable LED 1
D3	FDX/COLLED	Full Duplex / Collision
D4	SPEED	100Mbps
D5	LINKACT	Link Active

Figure 3-5 Meaning of LEDs

3.8. 7-Segment LED

The onboard 7-segment LED provides a user-friendly status output display. It is mapped to the External I/O Bank 1 of the S3C4510 microprocessor. The assignment of the display segments is shown in Figure 3-6.

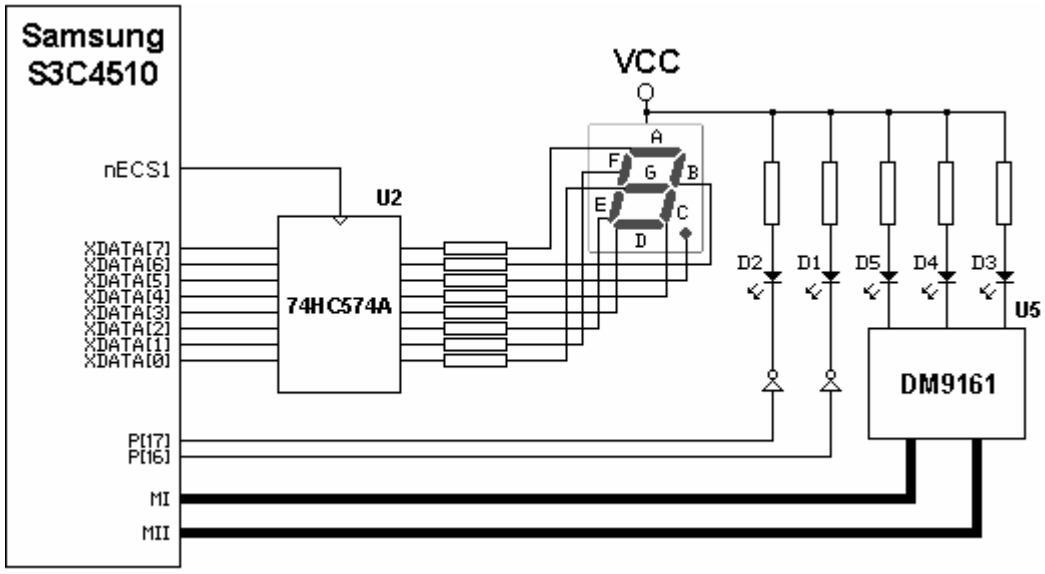


Figure 3-6 Schematic of output LEDs

3.9. General purpose I/O

The S3C4510 microprocessor has 18 programmable I/O ports. You can configure each I/O port to input mode, output mode, or special function mode by making appropriate settings to the IOPMOD and IOPCON registers. NET-Start! preserves some of the GPIO pins for controlling onboard peripherals such as watchdog timer. Some of those pins are also used to provide auxiliary RS-232C control signals to interface with the attached modem. Figure 3-7 shows the assignment of the GPIO pins on the NET-Start! evaluation board.

GPIO	Name	Description
0	GPIO_0	General purpose I/O 0
1	GPIO_1	General purpose I/O 1
2	GPIO_IN0	User input switch 1 within SW2
3	GPIO_IN1	User input switch 2 within SW2 [Note]
4	GPIO_RESERVED	Not Available
5	GPIO_WATCHDOG_CLK	Watchdog timer clock control
6	GPIO_RESERVED	Not Available
7	GPIO_UART1_EN	RS232 enable
8	GPIO_8 / GPIO_EXT_IRQ0	General purpose I/O 8 / External IRQ0
9	GPIO_9 / GPIO_EXT_IRQ1	General purpose I/O 9 / External IRQ1
10	GPIO_CTS	RS232 Clear To Send
11	GPIO_RI	RS232 Ringing Indicator
12	GPIO_12 / GPIO_EXT_DRQ0	General purpose I/O 12 / External DMA request 0
13	GPIO_RTS	RS232 Request To Send
14	GPIO_14 / GPIO_EXT_DACK0	General purpose I/O 14 / External DMA acknowledge 0
15	GPIO_CD	RS232 Carrier Detect
16	GPIO_LED0	User programmable LED D1
17	GPIO_LED1	User programmable LED D2

Figure 3-7 NET-Start! GPIO assignment

Note: The GPIO_IN1 is used by the bootstrap loader. Selecting the ON position will force the bootstrap loader to run in the diagnostic mode. Otherwise, the embedded Linux will be booted up after system power on.

3.10. Character based LCD interface

NET-Start! supports the interface of the character based LCD module at the External I/O bank 0. Figure 3-8 shows the pin definition of the character based LCD connector.

Pin	Name
1	VCC (+5V)
2	GND
3	Register
4	Read / -Write
5	Enable
6	Data0
7	Data1
8	Data2
9	Data3
10	Data4
11	Data5
12	Data6
13	Data7
14	GND

Figure 3-8 Pin definition of the LCD connector (P1)

3.11. Real-time clock

The DS1672 real-time clock (RTC) is controlled by software programs for the calculation of time of day, week, month, and year. A precision temperature compensated reference and comparator circuit monitors the status of the input voltage. When a power failure condition occurs, the RTC will be automatically switched to the backup power input, a super-capacitor (C12), which is used to keep ticking during power loss.

3.12. Reset circuit and watchdog

The architecture of the reset circuit on the NET-Start! evaluation board is shown in Figure 3-9.

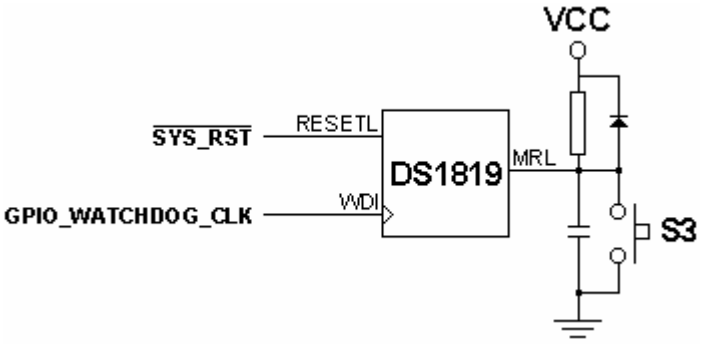


Figure 3-9 NET-Start! reset circuit

As shown in Figure 3-9, the DS1819 watchdog controls the "RESET" signal of the S3C4510 network processor. It detects the out-of-tolerance power supply conditions and reset the whole system. It also provides a “Watchdog” function to detect whether the user program is stuck in an infinite loop due to some unpredictable situations. On power up, the watchdog is disabled. Once the watchdog is started, the user program must drive the GPIO [5] pin with one period clock in order not to reset the system. Besides, a button (S3) is provided to reset the system manually.

3.13. Ethernet

The S3C4510 microprocessor has an Ethernet controller that operates at 10/100M-bits per second in half-duplex or full-duplex mode. In half-duplex mode, the controller supports the IEEE 802.3 CSMA/CD protocol. In full-duplex mode, it supports the IEEE 802.3 MAC Control Layer including the Pause operation for the flow control.

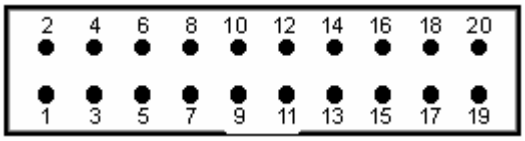
The MAC layer of the Ethernet controller supports both the Media Independent Interface (MII) and the Buffered DMA Interface (BDI). The MII conforms to the ISO/IEC 802.3 standards for a media independent layer that separates the physical layer issues from the MAC layer.

The physical layer entity (PHY) DM9161 is used to perform all of the decoding / encoding on incoming and outgoing data. The Manchester coding is used for 10Base-T, 4B/5B for 100Base-X, and 8B/6T for 100Base-T4. The MII provides the raw data it receives, starting with preamble and ending with CRC. When the raw data is available for transmission, the MII also starts with the preamble and ends with CRC. The MAC layer also generates jam data and transmits it to the PHY.

For more information about the Ethernet controller of the S3C4510 microprocessor, you may consult the *Samsung S3C4510 32-BIT RISC Micro Controller Embedded Network Controller User’s Manual.*

3.14. JTAG

The 20-pin connector (P6) is connected to the JTAG interface of the S3C4510 microprocessor. The pin definition is compatible with the ARM Multi-ICE interface and as such supports several ICE Breaker compatible modules. Figure 3-10 shows the pin definition of the JTAG connector.



Pin	Name	Description
1	VTref	Target Reference Voltage
2	Vsupply	Supply Voltage
3	NTRST	Test System Reset
4,6,8,10,12,14,16,18,20	GND	Ground
5	TDI	Test Data Serial In
7	TMS	Test Mode Select
9	TCK	Test Clock
11	RTCK	Return Test Clock
13	TDO	Test Data Serial Out
15	NRESET	Target System Reset
17,19	NC	Not Connected

Figure 3-10 Pin definition of the JTAG connector (P6)

3.15. DC/DC converter

The Samsung S3C4510 microprocessor and most of the onboard peripherals are operated at DC 3.3V. The input voltage, 12V DC, is converted to 3.3V DC by the on-board switching power converter. This DC/DC converter is efficient; thus, overheat is not a concern.

3.16. Power supply

NET-Start! is powered through an external DC12V / 1000 mA wall-mounted power adapter. This adapter is plugged into the jack connector JP2. Please make sure the level of input voltage is within the acceptable range; otherwise, **permanent damage might occur**.

4. Software Description

4.1. Introduction

The software of bootable device consists of the following components:

Bootstrap loader -- The first program to be executed which boots up the system, initializes the system peripherals, and provides a console feature to fully control the hardware devices over the serial port.

uClinux kernel -- The operating system.

Initial ramdisk -- The file system containing the utilities and system configurations.

Utilities -- A program set providing the interface between user and the operating system.

NET-Start! can be booted directly into the uClinux environment. The boot sequence of this mode is described as follows:

1. Boot up with bootstrap loader.
2. Initialize the system and peripherals.
3. Launch uClinux kernel
4. Read initial ramdisk
5. Mount the initial ramdisk as root
6. Execute ***init*** and launch shell

4.2. Bootstrap Loader

The bootstrap loader is a hardware diagnostic program. It boots up the system, initializes the system peripherals, and provides a console feature to fully control the hardware devices over the serial port. By default, the bootstrap loader will load and execute Linux kernel after initializing the system peripherals. If the switch 2 of SW2 is selected at the ON position during hardware reset (or power-on), the serial console of the bootstrap loader will be activated, and files can be uploaded to or downloaded from PC via the XMODEM protocol. The memory manipulation operations, including fill(), copy(), dump(), peek(), and poke(), and the program execution commands are available for low-level hardware controls. The bootstrap loader commands are listed in Figure 4-1.

```
HELP – Help
UNIT - Set access unit
DUMP - Memory dump
COPY - Memory copy
FILL - Fill memory
POKE - Poke memory
PEEK - Peek memory
TX - Send XMODEM file
RX - Receive XMODEM file
GO - Execute binary
INFO - Print system information
SWITCH - Switch mode
```

Figure 4-1 Commands of bootstrap loader

4.3. uClinux

Linux is a Unix clone written from scratch by Linus Torvalds with assistance from a loosely knit team of hackers across the Internet. It has all the features you would expect in a modern Unix machine, including fully-functioned multitasking, virtual memory, shared libraries, demand loading, proper memory management and TCP/IP networking. In addition, it aims on POSIX compliance. Programs, which are compliant to POSIX standard, have greater portability to other POSIX machines.

Linux was first developed for 386/486-based PCs. These days, it has been ported to ARMs, DEC Alphas, SUN Sparcs, M68000 machines, MIPS, SH, PowerPC, and other processors. These processors are all equipped with Memory Management Unit (MMU) to support virtual memory feature of Linux. However, the MMU requires lots of silicon area in the embedded processor and increases the system cost. uClinux is a port of Linux to systems running on embedded processors without MMU.

4.4. Initial Ramdisk

The initial ramdisk contains the preloaded modules, configuration files, and programs. And it will be mounted as the root directory. To work properly, the initial ramdisk should include a minimum set of files to support the system.

4.5. Utilities – Shell & BusyBox

The initial ramdisk contains a msh (Minix Shell) and a lot of utility programs (i.e. ps, ls, mount, etc.) provided by **busybox**. The utility programs are listed in the following list.

```
cat – concatenate files and print on the standard output
chmod – change file access permissions
cp – copy files and directories
date – print or set the system date and time
df – report filesystem disk space usage
echo – display a line of text
kill – terminate a process
ln – make links between files
ls – list directory contents
mkdir – make directories
mknod – make block or character special files
mount – mount a file system
mv – move (rename) files
ping – send ICMP ECHO_REQUEST packets to network hosts
ps – report process status
pwd – print name of current/working directory
rm – remove files or directories
```

```
rmdir – remove empty directories                                .
sleep – delay for a specified amount of time                    .
umount – unmount file systems
uname – print system information
vi – a programmers text editor
```

Besides, a DHCP client and a simple HTTP server are included. By running the DHCP client, the NET-Start! could get the dynamic IP address and network configurations from the DHCP server. Otherwise, you have to configure the network manually as follows:

The following command asks the DHCP server for the network settings,

```
# dhcpcd &
# ifconfig
eth0     Link encap:Ethernet  HWaddr 00:90:A0:01:30:0A
         inet addr:192.168.0.43  Bcast:192.168.0.255  Mask:255.255.255.0
         UP BROADCAST NOTRAILERS RUNNING  MTU:1500  Metric:1
         RX packets:%Lu errors:0 dropped:0 overruns:0 frame:0
         TX packets:%Lu errors:0 dropped:0 overruns:0 carrier:0
         collisions:0
         RXbytes:%Lu(%Lu.0(null)b)(null)Xbytes:%Lu(%Lu.0(null)b)(null)
```

while the following three lines configure the network manually.

```
# ifconfig eth0 192.168.0.250 broadcast 192.168.0.255 netmask 255.255.255.0
# route add -net 192.168.0.0 netmask 255.255.255.0 eth0

# route add default gw 192.168.0.2 (adapted to your local gateway IP)(Note)

  Note:
  If you don't have any gateway in your environment, please just omit it.
  Then you could only have a local LAN environment.
```

5. Setting up Development Environment

5.1. Tool chain installation

GCC tool chain is used as NET-Start!'s development tool. GCC tool chain includes gcc compiler, assembler, linker, and other related utilities. Besides the tool chain, developers also need to build the C library for the application program to link with. NET-Start! adopts a well-known small size library, uClibc. This tool kit is included within the CD-ROM associated with the evaluation board. Please install the tool chain on the host computer by the following steps:

1 The tool chain will be installed in /usr/local/gnu-2.95.3. You have to log in the host machine as 'root' and add /usr/local/gnu-2.95.3/bin to the searching path. Type the following commands to set up the tool chain.

```
(Host)# rpm -i armtools-2.95.3-5.i386.rpm
(Host)# export PATH=/usr/local/gnu-2.95.3/bin:$PATH
```

To make the path effective permanently, you may add the above 'export' command into any of the login script files, for example, the file $HOME/.bash_profile.

2 Installing uClibc

The uClibc will be installed in /usr/local/uClibc-0.9.5 . Make a symbol link from /usr/local/uClibc-0.9.5/linux-2.4.x to uClibc under your home directory.

```
(Host)# rpm -i uClibc-0.9.5-1.i386.rpm
(Host)# rm –f ~/uClibc
(Host)# ln –s /usr/local/uClibc-0.9.5/linux-2.4.x ~/uClibc
```

Now, you have built a complete development environment.

3 Rebuilding uClibc

This step is not needed, if you have finished step 2. The kernel source must be built before uClibc is built. Please refer to Section 6.5 to see how this can be done.

```
(Host)# tar zxvf uClibc-snapshot.tar.gz
(Host)# gzip -cd uClibc-20020118.patch.gz | patch -p0
(Host)# cd uClibc
(Host)# make
```

Once you have compiled the uClibc package, the C startup file (crt0.o), C library (libc.a), math library (libm.a), and other selected libraries can be found in the ‘uClibc/libc’ directory.

5.2. Makefile

To utilize the tool chain and link uClibc on your program, you could write 'makefiles' to build your programs. For your convenience, we provide a makefile, App.mk, to help release the burden that will otherwise be put on the developer when compiling applications on the target machine. App.mk, together with other application programs, can be found by extracting the file “NETStart-Apps.tar.gz” as follows:

```
(Host)# tar zxvf NETStart-Apps.tar.gz
(Host)# cd Apps/common
(Host)# ls
```

The file “NET-Start-Apps.tar.gz” can be found in the directory ‘src/’ of the associated CD.
The following subsections will describe how to utilize App.mk for your application.

5.2.1. Single Target

When compiling a single file for NET-Start!, the easiest way is to write a makefile, with the file name “Makefile”, which includes App.mk as follows:

```
OBJECTS = <object file>
TARGET = <executable file name>
include <PATH of App.mak>/App.mk
```

where <object file> has the same file name with the source file, except that the suffix ".c" is replaced by ".o".

For instance, the Hello World example contains the following Makefile,

```
OBJECTS = hello.o
TARGET = hello
include ../common/App.mk
```

where the path of App.mk is '../common/App.mk'.

After the Makefile has been prepared, all you need to do is to type the 'make' command. Then, an executable file "hello" will be generated.

5.2.2. Multiple Targets and more complex rules

When writing a Makefile with complex rules and/or multiple targets, you can refer to the following steps:

1. Include App.mk at the '-bottom-' of the Makefile by 'include ../common/App.mk'
2. Use $WIS_LINK instead of linking the object files

For instance, the Makefile will help you compile and generate two targets prog1 and prog2 from prog1.c and prog2.c.

```
CFLAGS = -g

all: prog1 prog2

prog1: prog1.o
$(WIS_LINK)


prog2: prog2.o
$(WIS_LINK)

prog1.o: prog1.c


prog2.o: prog2.c

include ../common/App.mk
```

5.2.3. Compiling host objects by App.mk

App.mk also allows users to compile binary codes for the host machine directly. To do this, add 'HOST_COMPILE=1' after the make command, e.g. 'make HOST_COMPILE=1'. The same rule of the makefile will be used to build binary codes for the host machine.

5.3. Download/Upload File between Host and Evaluation board

Before uploading (target to host) or downloading (host to target) your program (or any other data file), your NET-Start! evaluation board must be turned on, connected to the host through the serial port (assume on COM1/ttyS0), and booted into uClinux. The next two sections will explain the procedures using two popular terminal programs: HyperTerm on Windows and

Minicom on Linux.

5.3.1. Download

Follow the steps listed below:

Step 1:

(On NET-Start!) Change the directory to the destination location.

Step 2:

Host sends the file by the terminal connected to NET-Start!.

HyperTerm:

1. Select 'Send->SendFile'
2. Enter the file name or choose it by pressing 'Browse'.
3. Select 'Zmodem' as a transmitting protocol.
4. Press 'Send'.

Minicom:

1. Press 'Ctrl+A->S'.
2. Select 'Zmodem' as a transmitting protocol.
3. Use cursor to move up/down in the file select menu.
4. Press 'SPACE' key twice to change the directory whenever needed.
5. Press 'ENTER' to send the selected file.

Step 3:

Wait till the transmission is finished.

5.3.2. Upload

Sometimes users like to save the output of a program in a file. Besides, some log files may be generated by the application. Thus, You may want to upload the file from the evaluation board into your host machine. To do this, you may use the zmodem utility on the NET-Start! evaluation board. The command to send out a file is 'sz <file>'. You may issue this command under the command line prompt.

6. Programmers Reference

6.1. Memory Map

The S3C4510 microprocessor has a total of 64M bytes memory space. It is divided into memory banks. Each memory bank can be located on any address by setting appropriate memory control register (ROMCON). The data bus width of each bank can be configured by setting the data bus width register (EXTDBWTH).

The NET-Start! evaluation board contains two types of memories. The first one is the 2M bytes Flash ROM, which is used as a boot ROM located at Bank0. The data bus width of Bank0 is configured to 16 bits by hardware pins (i.e. B0SIZE [1:0] = '10'). The other one is the 16M bytes onboard SDRAM available for users. The bootstrap loader enables the access of SDRAM by setting SYSCFG [31]. The S3C4510 microprocessor also has 8K bytes SRAM, it is configured in unified (Instruction/Data) cache mode by the bootstrap loader to increase the system performance.

On system power up, the bootstrap loader begins execution from address 0, and then reconfigures the memory map as listed in Figure 6-1. If the switch 2 of SW2 is set at the OFF position, the bootstrap loader will boot into the built-in Linux kernel. Otherwise, the command-line diagnostic program will be executed. While the diagnostic program is running, the memory usage is listed as below in Figure 6-2.

Address range	Size	Description
0x00000000 to 0x01000000	16MB	32-bit SDRAM, using DRAMCON0
0x01800000 to 0x01A00000	2MB	16-bit Flash bank, using ROMCON0
0x03FE0000 to 0x03FE1FFF	8KB	32-bit internal SRAM, used as cache
0x03FF0000 to 0x03FFFFFF	64KB	S3C4510 control registers
0x03600000 to 0x03603FFF	16KB	External I/O Bank0, used for character based LCD
0x03604000 to 0x03607FFF	16KB	External I/O Bank0, used for 7-segment
0x03608000 to 0x0360BFFF	16KB	External I/O Bank0, used for DIP-Switch

Figure 6-1 NET-Start! memory map

Address range	Description
0x00000000 to 0x0000003F	Exception vector table and address constants
0x00000040 to 0x00007FFF	Read-write data space for bootloader
0x01800000 to 0x0180FFEF	Program and read only data space for bootloader
0x0180FFF0 to 0x0180FFFF	Reserved for Ethernet MAC address
0x01810000 to 0x018FFFFF	NET-Start! Linux kernel image
0x01900000 to 0x0199FFFF	Initial ramdisk image for NET-Start! Linux kernel
0x019a0000 to 0x019FFFFF	Unused Flash memory

Figure 6-2 Memory Usage

6.2. Programming limitations on uClinux

One characteristic of uClinux is that it does not employ MMU - Memory Management Unit (mainly due to the reason that the underlying hardware platform does not support MMU at all.) This is quite different from most Linux environments. Some restrictions are thus put on the programs running on uClinux. Fortunately, these restrictions appear only when you are developing system software or when your program requires a large stack size. Most applications will not be affected at all.

6.2.1. Stack

Since uClinux does not support MMU, no virtual memory is available. Even more, there is no paging. The physical memory space is shared among all processes. To run programs under such an environment, each process will be assigned a pre-determined fixed size space within the memory space during the program load time. This is called the 'flat' mode. One consequence of this is that the stack size must be known before the program is launched. On NET-Start!, the default stack size is 8K bytes. Users may alter this value by defining a variable 'FLTFLAGS' within the makefile. For instance, the following example illustrates how to set the stack size to 4k for the Hello World example:

```
TARGET = hello
OBJECTS = hello.o
FLTFLAGS = '-s 4096'
include App.mk
```

6.2.2. Fork v.s. vfork

Normally, under Linux, a process invokes fork() to create a child process. The parent and the child process will share almost everything at the beginning, such as all data and file descriptors, etc. Once any of the two processes tries to change a data value (i.e. write to memory), the MMU will create a private copy of the memory page where the changed data exists in the writing process. The other process still views its data unaltered. This method is called ‘Copy-on-Write’. By using this method, Linux provides an efficient fork() system call.

uClinux runs on systems without MMU support. Using fork() is inhibited for the uClinux programs. The major reason is that the fork() system call relies on the Copy-on-Write mechanism, which requires the support of MMU. To solve this problem, uClinux implements vfork() instead of fork(). In the vfork() process model, child process shares the same memory space and stack with the parent process. Using this function would lead to the following restrictions:

1. Avoid modifying the global variables or static variables in the child process. This is

because all these changes will not only affect the child process, but also reflect on the parent process.

2. Do not use ‘return’ from the function where vfork() was called, use _exit() instead.

For the semantics of fork() and vfork(), please refer to the Linux on-line manual page.

6.2.3. Other system calls

Some system calls, such as shared memory, are not implemented in the current version yet.

6.3. Developing Application Program

Developing an application on NET-Start! is as easy as that on your Linux box. All you have to do is to set up your cross-compiling environment first. Then, follow the procedure below, and you will be able to make your program work.

Step 1: Develop your programs

Step 2: Write a makefile (App.mk must be included in this file.)

Step 3: Build the binary code

Step 4: Download the executable file

Step 5: Execute the program

6.3.1. Example: Hello World

The following example shows how to write a "Hello World" program. You can find this example in the directory Apps/hello/.

Step 1: Write hello.c as the following program:

```
#include <stdio.h>
int main() {
                  printf("Hello World\n");
}
```

Step 2: Write a Makefile

```
TARGET = hello
OBJECTS = hello.o
include ../common/App.mk
```

For the usage of App.mk and the variables TARGET and OBJECTS, please refer to Section 5.2.

Step 3: Type 'make'. A program named 'hello' will be produced.

```
(Host)# cd Apps/hello
(Host)# make
[ -- Compile hello.c to hello.o -- ]
```

```
arm-elf-gcc -I/home/cyc/uClibc/include -msoft-float -mcpu=arm7tdmi
-fomit-frame-pointer -fsigned-char -Os -Wall -DEMBED -D_uclinux_ -o
hello.o -c hello.c
[ -- Link hello.o into hello1 -- ]
arm-elf-ld -L/usr/local/gnu-2.95.3/lib/gcc-lib/arm-elf/2.95.3
-L/home/cyc/uClibc/lib -L/usr/local/gnu-2.95.3/arm-elf/lib -elf2flt -o
hello1 /home/cyc/uClibc/lib/crt0.o
/usr/local/gnu-2.95.3/lib/gcc-lib/arm-elf/2.95.3/crtbegin.o hello.o
/usr/local/gnu-2.95.3/lib/gcc-lib/arm-elf/2.95.3/crtend.o -lc -lgcc -lc
2> hello1.err_ld || cat hello1.err_ld
Done.
(Host)# ls
hello  hello.c  hello.elf  hello.err_ld  hello.gdb  hello.o  Makefile
```

To clean the generated files, just type 'make clean'.

Step 4: Download the program.

Please refer to Section 5.3 for details about download. After the download is completed, you have to change its mode by 'chmod +x hello', where hello is a downloaded executable file.

On the console of NET-Start! evaluation board:

```
# chmod +x hello
# ls -l hello
-rwxr-xr-x   1 root    root       10328 Mar 13  2002 hello
#
```

Step 5: Execute the program by typing 'hello'

Type './hello' to execute the program under the directory where you download the program. The 'Hello World' message will be printed on the evaluation board console.

```
# ./hello
Hello World
#
```

6.3.2. Example: The 7-segment LED demo program

This program is located at Apps/demo. After building the executable code ‘demo’, you may download it onto the NET-Start! evaluation board by the methods introduced in Section 5.3.1. Change the permission and execute the program as follows:

```
# chmod +x ./demo
# ./demo
Welcome to use NET-Start! for developing Linux applications.

I'll enter an infinite loop for reading DIP switches and
displaying the corresponding hexadecimal digit on the LED.

Press any onboard button to abort the program...
```

On execution, the program ‘demo’ will count down to 0 on the 7-segment LED first. Then, the corresponding hexadecimal number of the state of the user input switches SW1 will be

displayed on the LED until any onboard button (S1, S2) is pressed.

6.4. Porting Applications

An advantage of using Linux as your software platform is that you also get a lot of handy applications. Tons of open source applications are already available on the Internet. All you have to do is to retrieve the source code and build it according to your needs. To accomplish this, several porting issues should be considered.

6.4.1. uClinux restrictions

As mentioned in Section 6.2, users have to check whether the uClinux programming restrictions are satisfied. If necessary, users have to modify the code accordingly. One convention used during porting codes to the embedded systems is to use a C macro EMBED to distinguish the codes for the embedded version. The provided App.mk includes -DEMBED in CFLAGS by default when compiling the target code. For example, the following code will use vfork() instead of fork() automatically.

```
#ifndef EMBED
fork();
#else
vfork();
#endif
```

6.4.2. Adapting the makefile

Although Wiscore has prepared the one-step-to-reach makefile 'App.mk', it may not be that straightforward to just include it in your own Makefile. This is because there often already exists a Makefile while you are porting an application. You need to rearrange the makefile to include App.mk, to combine the flags together, and to remove some unused configurations that will possibly result in rule conflicting.

6.4.3. Example: The web-server (thttpd) program

The web-server program thttpd is a small and efficient HTTP Server. We have ported a stable version (2.20c) onto our environment (uClinux). The executable binary file has also been put in the NET-Start! evaluation board. You may directly run thttpd under directory /usr/sbin/thttpd within the evaluation board. (Of course, users may by themselves build the binary code from the source under directory Apps/thttpd-2.20c/.)

6.4.3.1. Execute thttpd

The home page of thttpd is at http://www.acme.com/software/thttpd/thttpd.html. Users can find the usage of thttpd and all relative information in this website. As an example, users may run

thttpd with the following command:

```
# thttpd -d <www_root> -c '**.cgi' &
```

This will start up thttpd with <www_root> as its WWW document root directory. The CGI program can be executed if it has the extension name '.cgi', and its default port will be 80.

Note: This porting always shows the log message on the standard error.

6.4.3.2. Use CGI

CGI is also supported by thttpd. However, CGI is not the subject of this document. We will not discuss the detail here. Users may find a lot of CGI documents on Internet or in books. To illustrate the use of CGI, users may follow the example below to test the CGI on NET-Start!

.

Note: We recommend you edit the script under Linux environment.

1. Type the following shell script as a file, called env.cgi.

```
#!/bin/sh
echo "Content-type: text/plain"
echo
echo "This is a CGI program, written in shell script"
date
echo
echo "Available Shell Variables:"
echo
set
```

2. Download and put it under the root directory.
3. Change the mode to 755 (chmod 755 env.cgi)
4. Connect the evaluation board to the network (using a RJ-45 twisted-pair line)

Three programs 'ifconfig', 'route', and 'dhcpcd' in NET-Start! can be used to initialize the network. Users can get the usage of this utility from the manual page on the host Linux box. (If there is a DHCP server on your LAN, running 'dhcpcd' will set up all things.) Assume the IP address is set as 192.168.0.2 here.

5. Launch the HTTP server:

```
thttpd -d / -c '**.cgi' &
```

6. Now from a browser of your host machine, type the URL

http://192.168.0.2/env.cgi

On the browser, you will see the result as below:

```
This is a CGI program, written in shell script
Thu Jan 01 00:01:24 ??? 1970

Available Shell Variables:



PS2=
PS1=
IFS=

HOME=/
SHELL=/bin/sh
CGI_PATTERN=**.cgi
AUTH_TYPE=Basic
HTTP_HOST=192.168.0.2
HTTP_ACCEPT_ENCODING=gzip, deflate
HTTP_ACCEPT=*/*
HTTP_USER_AGENT=Mozilla/4.0 (compatible; MSIE 5.0; Windows 98; DigExt)
HTTP_REFERER=http://192.168.0.2/
REMOTE_ADDR=192.168.0.1
SCRIPT_NAME=/env.cgi
REQUEST_METHOD=GET
SERVER_PORT=80
SERVER_PROTOCOL=HTTP/1.1
GATEWAY_INTERFACE=CGI/1.1
SERVER_NAME=(none)
SERVER_SOFTWARE=thttpd/2.20c 21nov01
PATH=/usr/local/bin:/usr/ucb:/bin:/usr/bin
```

6.5. Building Kernel

6.5.1. Configuring kernel option for NET-Start!

The S3C4510 microprocessor doesn't have MMU built-in. To take the stability and the advantage of open source from Linux world, WISCORE has ported uClinux to Samsung S3C4510 network processor. The original source tar-ball of uClinux for ARM platforms and WISCORE patch are provided from attached CD-ROM . Untar the source tar-ball and apply the WISCORE patch for NET-Start! as follows:

```
(Host)# tar zxvf uClinux-2.4.tgz
(Host)# patch -p0 < uClinux-2.4.x-wiscore.patch
(Host)# cd uClinux-2.4.x
```

The default configuration options for the NET-Start! evaluation board is provided in the file 'arch/armnommu/def-config/wiscore', you may simply press ENTER to pass each configuration query.

```
(Host)# make wiscore_config
(Host)# make oldconfig
```

The detail descriptions on the configuration options can be found in the Linux sub-directory 'Documentation/Configure.help'. Those options contributed by WISCORE are described below:

```
*
* System Type
*
WISCORE Linux port (CONFIG_WISCORE) [Y/n/?]
Target (FA510, FA520, S3C4510, S3C2510, TMX320DSC2x) [S3C4510]
Set flash/sdram size and base addr (CONFIG_SET_MEM_PARAM) [Y/n/?]
(S)DRAM Base Address (DRAM_BASE) [00000000]
(S)DRAM Size  (DRAM_SIZE) [01000000]
FLASH Base Address  (FLASH_MEM_BASE) [01800000]
FLASH Size  (FLASH_SIZE) [00200000]
```

The above options are used to select the code specific to the S3C4510 microprocessor. There are 16M bytes SDRAM and 2M bytes Flash ROM on the NET-Start! evaluation board. (Note: the options Target, RAM size, and FLASH ROM size under the WISCORE Linux port option are fixed for NET-Start!.)

```
*
* General setup
*
Networking support (CONFIG_NET) [Y/n/?]
```

The most exciting feature of Samsung S3C4510 network processor is its built-in Ethernet controller. To enable the networking support in the Linux kernel, this option should be selected.

```
Enable ZFLAT support (CONFIG_BINFMT_ZFLAT) [Y/n/?]
```

To load and execute programs dynamically on uClinux platforms, the executables have to be converted into Flat binary format before execution. The Flat binary format has less overhead compared with other executable formats. It is suitable for embedded applications where memory is a scarce resource. The Linux Flat binary handler supports the compressed executables (ZFLAT). That is, the execution file can be stored in compressed form, then de-compressed into memory on program loading.

```
*
* Networking options
*
TCP/IP networking (CONFIG_INET) [Y/n/?]
...

*
* Network device support
*
Network device support (CONFIG_NETDEVICES) [Y/n/?]
...
```

```
*
* Ethernet (10 or 100Mbit)
*
Ethernet (10 or 100Mbit) (CONFIG_NET_ETHERNET) [Y/n/?]
  Wiscore DM9161 PHY attached (CONFIG_PHY_DM9161) [Y/n/?]
  Wiscore MAC address support (CONFIG_S3C4510_ETHERNET) [Y/n/?]
  WIscore RTL8019AS ethernet driver support (CONFIG_NETSTART_RTL8019AS)
[Y/n/?]
```

These options select the hardware components used in the network sub-system. On the NET-Start! evaluation board, the network sub-system is composed of the physical layer entity (PHY) DM9161 and the internal Ethernet controller of the S3C4510 microprocessor. Selecting these options enables the network driver support for the NET-Start! evaluation board.

```
*
* Block devices
*
RAM disk support (CONFIG_BLK_DEV_RAM) [Y/n/?]
  Default RAM disk size (CONFIG_BLK_DEV_RAM_SIZE) [1024]
  Initial RAM disk (initrd) support (CONFIG_BLK_DEV_INITRD) [Y/n/?]
```

The storage devices that are usually adopted in an embedded system include RAMs, ROMs, and Flash ROM; while the removable disks and hard disks are not always used due to system cost. Thus, we have installed an initial ramdisk, which contains a minimum set of devices and utilities for the user's runtime environment.

```
*
* Filesystems
*
Minix fs support (CONFIG_MINIX_FS) [Y/n/?]
...
/proc filesystem support (CONFIG_PROC_FS) [Y/n/?]
```

NET-Start! has 16M bytes onboard SDRAM, which are used for storing the code and data of the kernel, drivers, the user processes, and the ramdisks. The space left will be used for the stack and heap of the running tasks. The MINIX filesystem has been used to minimize the amount of memory used by Linux kernel.

```
*
* Character devices
*
Standard/generic serial support (CONFIG_ARCH_SERIAL) [Y/n/?]
  Echo console messages (CONFIG_SERIAL_ECHO) [Y/n/?]
```

From the kernel point of view, the serial port is the only one interface that the developers can input their commands and show messages during the program execution. The kernel modules will also output all the warning messages to the console. These options select whether you want to include the driver for the standard serial ports. You should enter YES to direct the stdin / stdout to the serial ports.

6.5.2. Building uClinux

The default built target of uClinux is an ELF format file, called ‘linux’. This file must be transformed into the binary format file ‘linux.bin’ first, which will be programed into the onboard Flash ROM.

Follow the instructions below to build the Linux kernel image.

```
(Host)# make dep; make clean
(Host)# make linux.bin
```

6.6. Building an initial ramdisk image

The initial ramdisk is usually manipulated through the mechanism of the loopback device. You will need to do this as root. Please follow the commands listed below to build your own initial ramdisk.

```
(Host)# mkdir initrd
(Host)# cp ($CDROM_PATH)/bin/kernel_2.4/initrd-2.4.x.gz ./
(Host)# gzip -d initrd-2.4.x.gz
(Host)# mount initrd-2.4.x initrd -o loop
(Host)# cd initrd

[add/delete files into this directory]

(Host)# cd ..
(Host)# umount initrd
(Host)# gzip -c -9 initrd-2.4.x > initrd-2.4.x.gz
```

6.7. Download kernel and initial ramdisk image to Flash ROM

Once you have compiled the uClinux kernel successfully, you should program the kernel image to the onboard Flash ROM at address 0x01810000. To download the image onto NET-Start!, you have to enter the diagnostic mode of the bootstrap loader. The following steps show how to download the kernel image and program the image into Flash ROM.

Download kernel image:

1. Type the following command under the bootstrap loader mode.

```
RUN>rx 0x10000
```

2. Send the file, 'linux.bin', from the host via XMODEM before the next ‘RUN>’ prompt is shown.
3. After the transmission is done, the ‘RUN>’ prompt will appear again. Then type the following command to write the downloaded image into Flash ROM.

```
RUN>copy 0x10000 0x100000 0x1810000
```

Download initial ramdisk image:

1. Type the following command.

```
RUN>rx 0x10000
```

2. Send the file, 'initrd-2.4.x.gz', from the host via XMODEM before the next ‘RUN>’ prompt is shown.
3. After the transmission is done, type the following command to write the initial ramdisk.

```
RUN>copy 0x10000 0xb0000 0x1900000
```

```
And it will show as follows:
```

```
RUN>
```

shows the screen snapshot of the NET-Start! from hardware reset. In order to boot the Linux kernel directly, the switch 2 of SW2 must be selected at the OFF position.

```
Linux version 2.4.19-uc1 (pack@linux.icore) (gcc version 2.95.3 20010315
(release)) #1 週五 1月 16 15:04:48 CST 2004
Processor: Samsung S3C4510B revision 6
Architecture: SAMSUNG
On node 0 totalpages: 4096
zone(0): 0 pages.
zone(1): 4096 pages.
zone(2): 0 pages.
Kernel command line:
Calibrating delay loop... 24.88 BogoMIPS
Memory: 16MB = 16MB total
Memory: 14240KB available (691K code, 164K data, 40K init)

Dentry cache hash table entries: 2048 (order: 2, 16384 bytes)
Inode cache hash table entries: 1024 (order: 1, 8192 bytes)
Mount-cache hash table entries: 512 (order: 0, 4096 bytes)
Buffer-cache hash table entries: 1024 (order: 0, 4096 bytes)
Page-cache hash table entries: 4096 (order: 2, 16384 bytes)
POSIX conformance testing by UNIFIX
Linux NET4.0 for Linux 2.4
```

```
Based upon Swansea University Computer Society NET3.039
Initializing RT netlink socket
Samsung S3C4510 Ethernet driver version 0.1 (2002-02-20)
<mac@os.nctu.edu.tw>
ne.c:v1.10 9/23/94 Donald Becker (becker@scyld.com)
Last modified Nov 1, 2000 by Paul Gortmaker

eth1 MAC: 00 0d 21 00 07 31
eth0: 00:0d:21:00:07:30
Samsung S3C4510 Serial driver version 0.9 (2001-12-27) with no serial options
enabled
ttyS00 at 0x3ffe000 (irq = 7) is a S3C4510B
ttyS01 at 0x3ffd000 (irq = 5) is a S3C4510B
Starting kswapd
RAMDISK driver initialized: 16 RAM disks of 1024K size 1024 blocksize
NET4: Linux TCP/IP 1.0 for NET4.0
IP Protocols: ICMP, UDP, TCP
```

```
IP: routing cache hash table of 512 buckets, 4Kbytes
TCP: Hash tables configured (established 1024 bind 1024)
RAMDISK: Compressed image found at block 0
VFS: Mounted root (minix filesystem).
Freeing init memory: 40K
Bummer, can't write to log on /dev/tty5!
console=/dev/ttyS0
init started:  BusyBox v0.60.5 (2003.06.25-03:03+0000) multi-call binary


BusyBox v0.60.5 (2003.06.25-03:03+0000) Built-in shell (msh)
Enter 'help' for a list of built-in commands.

# busybox
BusyBox v0.60.5 (2003.06.25-03:03+0000) multi-call binary

Usage: busybox [function] [arguments]...
   or: [function] [arguments]...

BusyBox is a multi-call binary that combines many common Unix
utilities into a single executable.  Most people will create a
link to busybox for each function they wish to use, and BusyBox
        will act like whatever it was invoked as.

Currently defined functions:
[, busybox, cat, chmod, chroot, clear, cp, date, dd, df, du, echo,
env, free, gunzip, gzip, ifconfig, init, kill, killall, linuxrc,
ln, ls, makedevs, mkdir, mkfs.minix, mknod, mount, msh, mv, ping,
ps, pwd, rm, rmdir, route, sed, sh, sleep, stty, sync, tar, telnet,
    test, tftp, umount, vi, watchdog, wget, which, zcat

# df
Filesystem         1k-blocks     Used Available Use% Mounted on
rootfs                 1009      940       69  93% /
/dev/root              1009      940       69  93% /
# free
             total       used       free     shared    buffers
  Mem:       15304       3360      11944          0         12
```

Figure 6-3 NET-Start! screen snapshot

7. FAQ

Please visit the website http://www.wiscore.com for more information.

8. Bug Report

If you find any bug on NET-Start! evaluation board or the associated software, please describe the problem as detailed as possible and e-mail it to bug-report@wiscore.com. Thank you!

9. Appendix A Schematics