NixCore X1 Documentation

NIXD01001
Revision 1.3 - October 2015

NixCore Website:
http://nixcores.com

NixCore X1 Product Page:
http://nixcores.com/nixcore_x1.php

Overview

NixCores are a line of computer on module (COM) processor boards that are designed to be integrated into electronic products. The NixCore line of processors are designed to be easy to use as the primary core processor of a product, or as an add on module to enable wireless networking to an existing product.

The NixCore X1 is the initial offering from NixCore and contains a 360MHz MIPS processor, 32MB of SDRAM and 8MB of FLASH. The NixCore X1 has an integrated WiFi module and Ethernet PHY as well as up to 24 GPIO lines that can be controlled from a user application. The NixCore X1 runs a full Linux system and images are provided on the http://nixcores.com website.

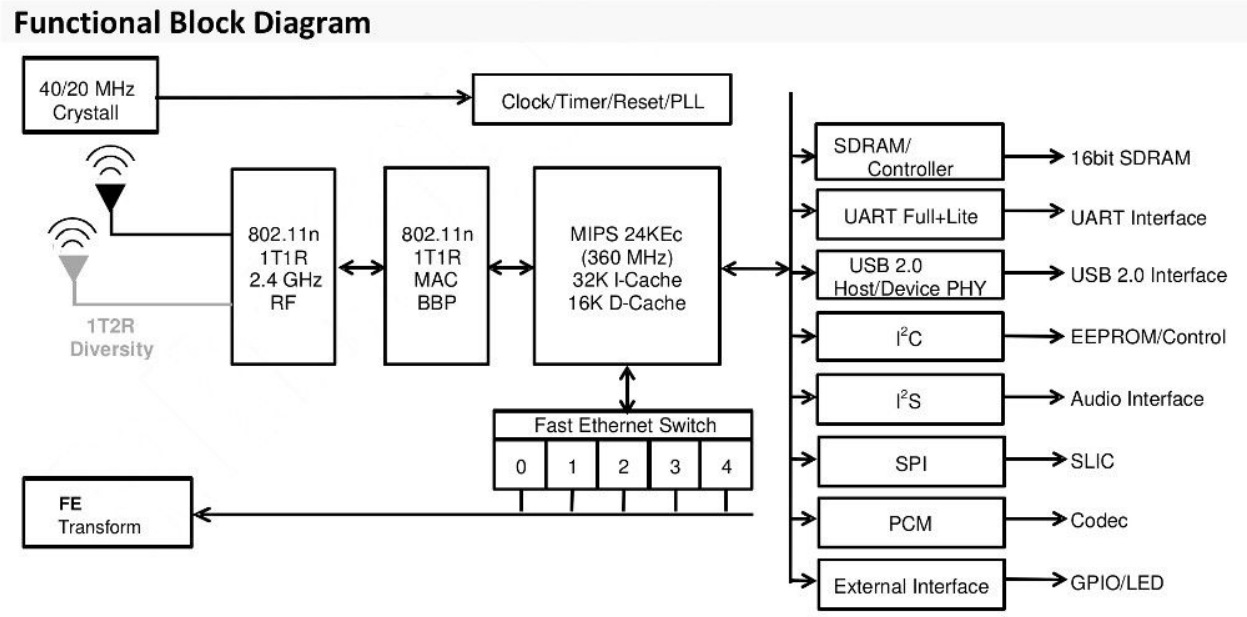
Default user: root. Password: root

Features

- 360MHz MIPS CPU
- OpenWRT Linux Firmware builds
- Linux Kernel 3.18
- 8MB Flash
- 32MB RAM
- 2x UART
- 24 GPIO
- I2C
- I2S
- SPI
- Supported by Arduino IDE on Windows or Linux
- JTAG
- 802.11g integrated WiFi
- 10/100M Ethernet PHY
- Full TCP/IP/UDP stack
- HTTP server
- SSL/TLS Support
- Documented, source code, pin outs, diagrams, PCB footprints
- Fully open source toolchain provided for both Windows and Linux

Table of Contents:

RT5350 Functional Block Diagram



Pin Description

NixCore X1 Pin Header:

Alt Function	Normal Function	Pin		Normal Function	Alt Function	
	JTAG_TRSTN/GPIO21	1	2	JTAG_TDO/GPIO17		Power
	JTAG_TCLK/GPIO20	3	4	GPIO0		Gnd
	JTAG_TMS/GPIO19	5	6	JTAD_TDI/GPIO18		GPIO
	I2C_DAT/GPIO1	7	8	I2C_CLK/GPIO2		
	3.3V	9	10	3.3V		
	ETH4_RX-	11	12	ETH4_RX+		
	ETH4_TX-	13	14	ETH4_TX+		
	GND0	15	16	GND5		
	USB D+	17	18	USB D-		
	GND1	19	20	GND4		
PCMDRX/CTS_N/GPIO13	DSR_N/GPIO13	21	22	ETH2_LED/GPIO24	BT_FREQ	
PCMFS/RTS_N/GPIO11	DTR_N/GPIO11	23	24	ETH4_LED/GPIO26	BT_ANT	
WDT_RST	SPI_CS1/GPIO27	25	26	RIN/GPIO14	PCMDTX/RXD/GPIO14	
BT_ACT	ETH0_LED/GPIO22	27	28	ETH3_LED/GPIO25	BT_WACT	
PCMCLK/TXD/GPIO12	DCD_N/GPIO12	29	30	ETH1_LED/GPIO23	BT_STAT	
	1.8V	31	32	GND3		
I2SSDI/GPIO10	RXD/GPIO10	33	34	RTS_N/GPIO7	I2SCLK/GPIO7	
I2SDO/GPIO9	CTS_N/GPIO9	35	36	TXD/GPIO8	I2SWS/GPIO8	
	GND2	37	38	MCS1_N	REFCLK0_OUT	
	RX2/GPIO16	39	40	TX2/GPIO15		

Pin Listing:

Pin X-#	Pin Name	Function	GPIO	Alt Function
1	JTAG_TRSTN/GPIO21	JTAG TRST		
2	JTAG_TDO/GPIO17	JTAG TDO		
3	JTAG_TCLK/GPIO20	JTAG CLK		
4	GPIO0		GPIO0	
5	JTAG_TMS/GPIO19	JTAG TMS	GPIO19	
6	JTAG_TDI/GPIO18	JTAG TDI	GPIO18	
7	I2C_DAT/GPIO1	I2C Data	GPIO1	
8	I2C_CLK/GPIO2	I2C Clock	GPIO2	

9	3.3V	3.3v		
10	3.3V	3.3v		
11	ETH4_RX-	Ethernet 4 RX Negative		
12	ETH4_RX+	Ethernet 4 RX Positive		
13	ETH4_TX-	Ethernet 4 TX Negative		
14	ETH4_TX+	Ethernet 4 TX Positive		
15	GND0	Ground		
16	GND5	Ground		
17	USB D+	USB Data Positive		
18	USB D-	USB Data Negative		
19	GND1	Ground		
20	GND4	Ground		
21	DSR_N/GPIO13	UART1 DSR	GPIO13	PCMDRX/CTS_N
22	ETH2_LED/GPIO24	Ethernet 2 LED	GPIO24	BT_FREQ
23	DTR_N/GPIO11	UART1 DTR	GPIO11	PCMFS/RTS_N
24	ETH4_LED/GPIO26	Ethernet 4 LED	GPIO26	BT_ANT
25	SPI_CS1/GPIO27	SPI CS1	GPIO27	WDT_RST
26	RIN/GPIO14	UART1 RIN	GPIO14	PCMDTX/RXD
27	ETH0_LED/GPIO22	Ethernet 0 LED	GPIO22	BT_ACT
28	ETH3_LED/GPIO25	Ethernet 3 LED	GPIO25	BT_WACT
29	DCD_N/GPIO12	UART1 DCD	GPIO12	PCMCLK/TXD
30	ETH1_LED/GPIO23	Ethernet 1 LED	GPIO23	BT_STAT
31	1.8V	Ethernet 1.8v		
32	GND3	Ground		
33	RXD/GPIO10	UART1 RX	GPIO10	I2SSDI
34	RTS_N/GPIO7	UART1 RTS	GPIO7	I2SCLK
35	CTS_N/GPIO9	UART1 CTS	GPIO9	I2SDO
36	TXD/GPIO8	UART1 TX	GPIO8	I2SWS
37	GND2	Ground		

38	MCS1_N			REFCLK0_OUT
39	RX2/GPIO16	UART2 RX	GPIO16	
40	TX2/GPIO15	UART2 TX	GPIO15	

Reference RT5350 Datasheet "DSRT5350_V1.0" section "1.3 Pin Sharing Scheme" for function and register settings.

Maximum Ratings and Operating Conditions

Absolute Maximum Ratings:
Supply Voltage ……..3.6 V
Vcc to Vcc Decouple…… –0.3 to +0.3 V
Input, Output or I/O Voltage…… GND –0.3 V to Vcc+0.3 V
(Pins are NOT 5V tolerant, exceeding the I/O Voltage can result in damage to the processor)

Thermal Information:
Thermal characteristics without external heat sink in still air conditions ……36.4 °C /W

Operating Conditions:
Temperature Range…….-10 to 55 °C
Core Supply Voltage……1.2 V +/- 5%
I/O Supply Voltage ……3.3 V +/- 10%

Logic Levels and I/O Current:
Logic High …… 2.0 V
Logic Low …….. 0.8 V
High Level Output Current ……... 18 mA
Low Level Output Current ……... 10 mA

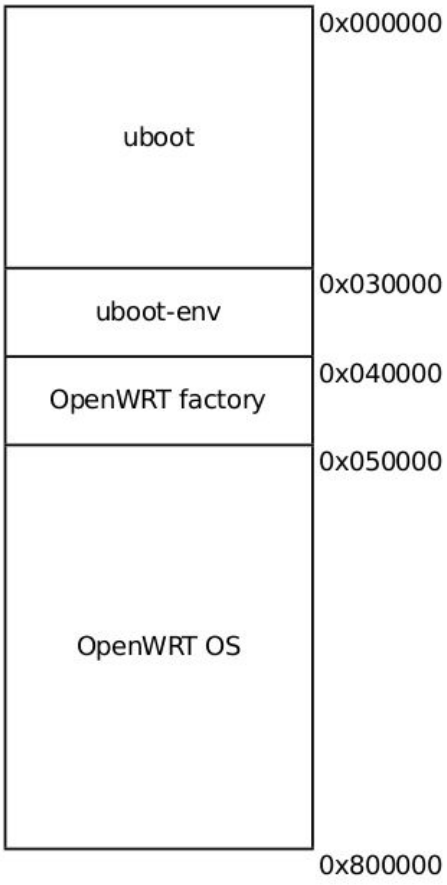
Operating Power @ 3.3v
WiFi Enabled Typical …… 1.0 W
WiFi Disabled Typical …… 0.74W

NixCore X1 Memory & FLASH

NixCore X1 Memory Map:

Reference RT5350 Datasheet “DSRT5350_V1.0” section “3.2 Memory Map Summary” for register settings and addresses

FLASH Map of NixCore X1:



Processor

The primary processor of the NixCore X1 is a Ralink RT5350 360 MHz MIPS24KEc SOC manufactured by Mediatek, part number RT5350F. The RT5350 includes all peripheral hardware and processor core. The NixCore X1 pairs the RT5350 with 8MB of S25FL064K/M25P80 compatible FLASH and 32MB of EtronTech EM63A165TS-6G DRAM.

RT5350 information:

- https://wikidevi.com/wiki/Ralink_RT5350

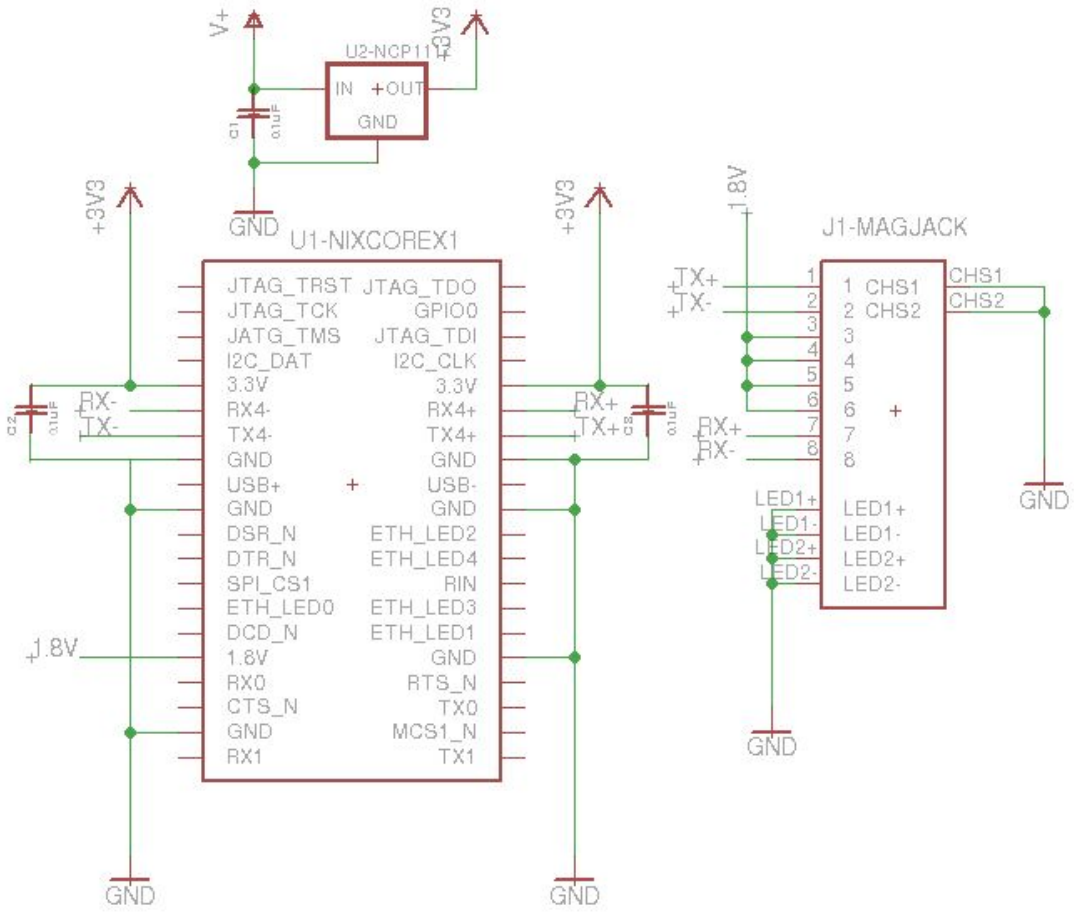
8MB FLASH information:

- http://www.spansion.com/Support/Datasheets/S25FL064K_00.pdf
- http://www.micron.com/parts/nor-flash/serial-nor-flash/m25p80-vmc6g

32MB RAM information:

- http://www.etron.com/manager/uploads/EM63A165TS_v1.4.pdf

Connection Diagram



Toolchain

Compiler on GNU Linux:

Software is developed on GNU Linux using GCC as the compiler and the uClibc C Library. A toolchain can be built using the .config file for OpenWRT (https://openwrt.org/), or using Buildroot (http://buildroot.uclibc.org/)

Setup OpenWRT Compiler:

We are working to get NixCore support integrated into the main branch of OpenWRT. There are a number of steps we need to go through to get approval from the main OpenWRT developers.

Until we get full trunk support in OpenWRT we are providing all files needed to get a OpenWRT build for NixCore X1.

We have provided a ZIP file on the NixCore X1 product page with all libraries, targets, files and configurations need to get a NixCore OpenWRT image built. We have also provided a 'setup.sh' bash script which will copy all files to the correct place within a local copy of the OpenWRT branch.

To define where the OpewnWRT branch is there is a single variable within the setup file that needs to be modified for any system.

Modify the OPENWRT_DIR variable at the top of the script to point to the location of the OpenWRT branch. By default the script assumes that the source is in a directory named "openwrt" one level up from where it is executed from.

Custom location example:
OPENWRT_DIR="/home/USER/openwrt_branch_XXX"

OpenWRT with Existing .config

Compile OpenWRT using the selected .config file on the NixCore X1 product page. Compile OpwnWRT on your machine "make -j X" where X is the number of concurrent threads you want to run. The toolchain for MIPS should be located under the "[YOUR_OPEWNWRT_DIR]/staging_dir/toolchain-mipsel_24kec+dsp_gcc-X.X-linaro_uClibc-X.X.XX.X/bin/" (Where Xs are replaced with the GCC and uClibc version numbers). Binaries should be prefixed with "mipsel-openwrt-linux-uclibc-".

OpenWRT .config Options:

If you can not use a .config file provided by NixCore, manual selection of the following settings will build a toolchain compatible with the NixCore X1

Target System: Ralink RT288x/RT3xxx
Subtarget: RT3x5x/RT5350 based boards)
Target Profile: NixcoreX1

Using Buildroot:

Compile buildroot using the selected .config file in the NixCore X1 product page. Compile buildroot on your machine (make). The toolchain for MIPS should be located under the "[YOUR_BUIDLROOT_DIR]/output/host/usr/bin/". Binaries should be prefixed with "mipsel-buildroot-linux-uclibc-"

Buildroot .config options:

If you can not use a .config file provided by NixCore, manual selection of the following settings will build a toolchain compatible with the NixCore X1

Architecture: MIPS Little Endian
Binary: ELF
Architecture Variant: mips 32r2
Use Soft-float: Yes [*]
Kernel Headers: 3.18
C Library: uClibc
Enable C++ Support: Yes [*]

Compiler on Windows:

Software is developed on Windows using GCC and the uClibc C Library. GCC and associated applications are provided as native Windows binaries from Mentor Graphics as part of their Sourcery CodeBench Lite product. Download the IA32 Windows Installer from the following link:

https://sourcery.mentor.com/GNUToolchain/release3068?cmpid=7108

Install the IA32 Windows Installer with all default options. Ensure that the option for adding the location of the tools to the PATH is enabled. Once installed the GCC toolchain should be located in C:\XXXXXXX\XXXXXXX\bin\ and typing "gcc-mips-XXXX.exe -v" in a command window should show version XX.XX

Root File System and Library Linking Overview:

The Root File System (RFS) is a directory that holds all of the files for the embedded system but it is on the host ("build") machine. The RFS directory should include usr/, etc/, bin/, etc directories. These directories contain header files and binary builds of libraries that will be used on the target embedded system. Since they are designed to execute on the embedded system the binaries are built with the cross compiler and will only work on the target architecture. When building new target applications it is required to link to libraries that are matched with the same architecture. This means that if you are building a MIPS application, you must point gcc to the location of any MIPS binaries, not the binaries for your "build" machine (most likely x86/amd64).

Example: The rxsrvr.bin application for transferring data across a serial link requires the Z Library for CRC. While the host ("build") machine (for example a desktop) has a libz.a binary in /usr/lib/ it is compiled for the host architecture, not NixCore X1 MIPS. To correctly build the rxsrvr.bin application we need to have a MIPS binary for libz and point the MIPS gcc compiler to that location.

Building RFS and Libraries for NixCore X1:

There are a number of ways to build an RFS for the NixCore X1, they are listed here from easiest to most difficult:

- Build OpenWRT for NixCore X1
- Build Buildroot for MIPS 32r2
- Build manually

OpenWRT Build:

OpenWRT is the default image for the NixCore X1 and as such any libraries built from an official config file will be available to the user application. Config files for OpenWRT 15.05 are available on the NixCore X1 product page. Steps to build the OpenWRT image include:

1. Download the OpenWRT 15.05 branch: *git clone git://git.openwrt.org/15.05/openwrt.git*
2. Download a .config file for NixCore X1: *http://nixcores.com/nixcore_x1.php#downloads*
3. Copy the .config to the root directory of the OpenWRT branch
4. run "make menuconfig" to select additional libraries you require
5. Save the .config file from menuconfig
6. Run "make -j X" where X is the number of concurrent threads you want to run

This sequence will build OpenWRT and generate a RFS in the OpenWRT folder under "staging_dir/target-mipsel_24kec+dsp_uClibc-0.X.XX.X/" where 0.X.XX.X will match the version

of uClibc (as of this document writing the version is 0.9.33.2). Ensure that any new libraries created by the build are copied to the same RFS directory on the target device.

Link against the binaries on the build machine by adding an include path to GCC:
-I [PATH_TO_OPENWRT]/staging_dir/target-mipsel_24kec+dsp_uClibc-0.X.XX.X/

Buildroot Build:

Buildroot is actually the basis for the OpenWRT build system, and can generate toolchains and an RFS for embedded systems. Since Buildroot supports the MIPS 32r2 LE processor of the NixCore X1 is is possible to build an RFS from Buildroot. Steps to build the Buildroot RFS include:

1. Download Buildroot: http://buildroot.uclibc.org/download.html
2. Download a Buildroot .config files for NixCore X1: http://nixcores.com/nixcore_x1.php#downloads
    a. Optional: Follow the "**Compiler on GNU Linux"** section to generate a custom X1 compatible Buildroot config for mips32r2
3. Run 'make menuconfig' to select additional libraries you require
4. Ensure that at least one "Filesystem image" options is selected
5. The number of concurrent threads to execute is located in the "Build options" menu
6. Save the .config file upon exiting
7. Execute "make"

This sequence will build Buildroot and generate a RFS in the Buildroot folder under "output/target/". Ensure that any new libraries created by the build are copied to the same RFS directory on the target device.

Link against the binaries on the build machine by adding an include path to GCC:
-I [PATH_TO_BUILDROOT]/output/target/

Manual RFS Build:

As with any embedded system as long as library and binaries are built for the target architecture any compiler can be used. This means that given

GPIO Pins

GPIO pins are accessed from the Linux system using the SYSFS interface (https://www.kernel.org/doc/Documentation/filesystems/sysfs.txt). GPIO pins are controlled by

virtual “files” in a virtual file system. These files are read from and written to in exactly the same method as any normal file, including from C using FILE * pointers and from the shell using “echo >” syntax.

GPIO Data location: /sys/class/gpio/

Each exported GPIO pin is listed as a folder in the sysfs folder. Under each folder are a number of files, each related to a function of the GPIO pin

GPIO Folder structure:

- active_low - Binay value for if the output logic is active low.
- edge - String for edge detection, default “none”
- value - Binary value for the output logic level of the GPIO, “1” or “0”
- direction - String either “in” or “out” representing the input or output function of the pin

An excellent overview of how to control GPIO functions via sysfs is provided is the Kernel document “GPIO Sysfs Interface for Userspace” found at:
https://www.kernel.org/doc/Documentation/gpio/sysfs.txt

Pin Direction:

GPIO pins in the NixCore X1 can be configured to be either inputs or outputs. Inputs read in binary data from the line and outputs drive a logic signal on a line. By default all GPIO lines in the system are inputs. To change direction of a GPIO pin the string “in” or “out” must be written to the direction virtual file within the GPIO pin folder.

Example from shell script, make GPIO pin 27 an output:
root@NixCoreX1:/# echo "out" > /sys/class/gpio/gpio27/direction

Pin Control:

The value of an output pin can be controlled by writing a single character ‘1’ or ‘0’ to the ‘value’ virtual file in the GPIO pin folder. Values can be determined by reading the same virtual file ASCII value.

Example from shell script, assert logic high to GPIO pin 27:
root@NixCoreX1:/# echo "1" > /sys/class/gpio/gpio27/value

Pins Available:

On the stock firmware, by default there are 8 GPIO pins automatically available for use. GPIOs enabled on firmware by default include:

- GPIO0
- GPIO17-GPIO21
- GPIO26-GPIO27

The RT5350 SoC shares GPIO functions with some communication functions such as UART and SPI. A total of 24 GPIO pins can be enabled from the system however enabling some pins as GPIO will disable some communication.

The following pins can be enabled using the “export” sysfs function. The GPIO pin numbers are grouped with the communication bus that they use.

GPIO Pins	Communication Bus
GPIO 2 & 3	I^2C Bus
GPIO 7-14	UART1, PCM, I^2S
GPIO 15 & 16	UART2 - Serial console
GPIO 22-25	Software SPI

Exporting Additional Pins:

The pins listed above require an ‘export’ step to make them available to userspace applications. Exporting is done by writing a GPIO value to the “export” file in the GPIO virtual file system. This can be done with an application or a shell script.

Example, export GPIO 25 to userspace:

- Ensure that the SPI driver has not claimed GPIO 25 (/dev/spidev0.1 should not exist)
    - Disable SPI driver: rmmod spi_gpio_custom
- root@NixCoreX1:/# echo "25" > /sys/class/gpio/export
- A new gpio25 folder will be visible

Unexporting Pins:

Linux provides the ability to “unexport” a GPIO pin and disassociate it from the GPIO driver. This allows the pins to be used for other functions such as communication hardware.

Unexporting is the same as exporting however the GPIO value is written to the “unexport” virtual file rather than the “export” file.

Example, unexport GPIO 25:

- root@NixCoreX1:/# echo "25" > /sys/class/gpio/unexport

NOTE: Automatically exported GPIOs (0,17-21,26,27) can NOT be unexported as they are associated with the GPIO hardware in the Device Tree. For advanced users, an explanation of Device Tree can be found at http://www.devicetree.org/Device_Tree_Usage and the NixCore X1 dts file in the OpwnWRT directory (target/linux/ramips/dts/NIXCORE-X1.dts)

UART

The Nixcore X1 processor board is equipped with a single Full-Featured UART and a single UART-lite, both of which generate 3.3v level digital signals. These two serial UARTs can support almost all common serial speeds up to 345600 and have been tested up to 115200 bps. The serial hardware can be access from the Linux system by accessing device file /dev/ttyS0 (Full-featured UART) and /dev/ttyS1 (UART-lite).

Full-featured UART:

- 8-pins (RIN, /DSR, /DCD, /DTR, RXD, /CTS, TXD, /RTS)
- 3.3v logic level inputs/outputs
- Exported to userspace by default
- Fully supported by Linux 3.18 kernel

UART-lite:

- 2-pins (RXD, TXD)
- 3.3v logic level inputs/outputs
- Exported to userspace by default
- Used for serial console by default
- Used for early Kernel messages
- Fully supported by Linux 3.18 kernel
- Default uboot serial console
- Uboot support for Kermit serial firmware update

OpenWRT serial documentation: http://wiki.openwrt.org/doc/hardware/port.serial

Linux serial programming HOW-TO using termios:
http://tldp.org/HOWTO/Serial-Programming-HOWTO/

I2C

The RT5350 SoC has a single I2C PHY hardware which is supported in the Linux 3.18 kernel. The I2C hardware is enabled by default in firmware builds and available in userspace as /dev/i2c-0. The I2C hardware fully supports the i2c-dev interface specification and can be accessed with standard file reads and writes. Since I2C is an addressed bus, ioctrl support exists for changing the slave number as well as other parameters.

Detailed information with examples is provided in the Linux Kernel i2c-dev documentation: https://www.kernel.org/doc/Documentation/i2c/dev-interface

SPI

SPI functions are implemented via software based GPIO SPI Linux Kernel driver. The RT5350 SOC has a single hardware based SPI hardware physical driver however it is used for the FLASH interface. The hardware based chip select 1 (CS1) line is routed to the NixCore X1 header if you would like to use it to control hardware.

By default SPI is implemented on GPIO lines 22 through 25 of the NixCore X1. The software based SPI is routed to pins pin27, pin30, pin22, and pin28, on the NixCore X1 header. The pinout is as follows:

SPI Function	GPIO Pin	X1 Header Pin
CLK	gpio22	X1-27
MOSI	gpio23	X1-30
MISO	gpio24	X1-22
CS	gpio25	X1-28

Enabling the SPI driver:

The SPI driver is not installed by default on stock firmware is must be enabled via a system command. This command can be entered manually or added to the NixCore startup script to be run at boot time.

To enable the spi device the spi-gpio-custom driver must be installed with the correct parameters. The following is the command to install the driver with the default gpio pins running at 100KHz.

root@NixCoreX1:/# insmod spi-gpio-custom bus0=1,22,23,24,0,100000,25

This will create a new device node at /dev/spi1.0

Disabling the SPI driver:

The gpio-spi-custom driver can be removed from the system with the following command: “rmmod spi_gpio_custom”. Once the driver is removed the pins are released to the system and can be used for other functions such as GPIO.

SPI Usage:

The SPI hardware is exposed to user space applications as /dev/spi1.0 and can be written to and read from in the same manner as data files in Linux.

Example: Send a byte of data out on the SPI bus:

```
FILE * fp;
fp = fopen("/dev/spi1.0","w");
if(fp)
{
   fwrite('A',1,1,fp);
   fclose(fp);
}else{
   printf("Can't open hardware\n");
}
```

More SPI Information:

More information about the SPI GPIO driver can be found on the author’s site:
https://randomcoderdude.wordpress.com/2013/08/15/spi-over-gpio-in-openwrt/

More information on the spidev interface can be found in the Kernel Documentation:
https://www.kernel.org/doc/Documentation/spi/spidev

WiFi

Wifi access is provided by the OpenWRT system. For a full discussion of the WiFi subsystem of OpenWRT, please see “Wireless Configuration” of the OpenWRT manual at http://wiki.openwrt.org/doc/uci/wireless

NixCore has developed scripts to manipulate the OpenWRT WiFi system from the system console. Bash scripts have been provided to both connect to an existing wireless network as well as generate a new wireless access point. These script can be called from user applications as well as web based CGI scripts.

Connecting to an existing network:

The *wifi_connect.sh* script is provided to connect a NixCore X1 to an existing Wifi network. The encryption supported is WEP,WPA,WPA2,and WPA Enterprise.

wifi_connect.sh SSID_NAME [ENCRYPTION_TYPE] [ENCRYPTION_KEY]

- SSID_NAME - Required. Name of the base station to connect to
- ENCRYPTION_TYPE - Optional. Mode value for encryption as per the OpwnWRT WPA Modes table: http://wiki.openwrt.org/doc/uci/wireless#wpa_modes
- ENCRYPTION_KEY - Optional. Key value for encryption

If ENCRPYTION_TYPE is not passed, the encryption value will be “none”. If ENCRYPTION_KEY is not passed the key value will be “none”.

Example, connect to an access point named “TestAP” with WPA2 encryption and key “ABC123”
wifi_connect.sh TestAP psk2 ABC123

Example, connect to an open access point PublicWifi
wifi_connect.sh PublicWifi

Creating a new Access Point:

The wifi_make_ap.sh script is provided to create new Access Point (AP). The encryption supported is WEP, WPA, and WPA2.

wifi_make_ap.sh SSID_NAME [ENCRYPTION_TYPE] [ENCRYPTION_KEY] [CHANNEL]

- SSID_NAME - Required. Name of the AP

- ENCRYPTION_TYPE - Optional. Mode value for encryption as per the OpwnWRT WPA Modes table: http://wiki.openwrt.org/doc/uci/wireless#wpa_modes
- ENCRYPTION_KEY - Optional. Key value for encryption
- CHANNEL - Optional. Channel to use for AP

If ENCRPYTION_TYPE is not passed, the encryption value will be “none”. If ENCRYPTION_KEY is not passed the key value will be “none”. If CHANNEL is not passed the default will be channel 6.

Example, make an access point named “TestAP” with WPA2 encryption and key “ABC123”
wifi_make_ap.sh TestAP psk2 ABC123

Example, make an open access point PublicWifi
wifi_connect.sh PublicWifi

Wireless hardware control:

Wireless networking in Linux/OpenWRT is broken into two sections, Wifi connection information, and Wifi network information. A network associated with Wifi hardware is created in OpenWRT as wlan0. This network can be connected to any hardware and the hardware can be in any mode.

The wifi connection is how the hardware communicates with other hardware. The wifi connection on the NixCore X1 can either be in station mode where it is a client of an access point, or it can be in AP mode in which it is an access point itself. Regardless of the type of wifi connection, wlan0 will always be the network that data is sent and received on.

Manual control of wifi connection:

The wifi connection can be started and stopped manually with the “wifi” command provided by OpenWRT. Running “wifi down” will turn off the wifi hardware on the chip. “wifi on” will turn on the wifi hardware on the chip.

NOTE: While the wifi command controls the wireless hardware, when the wireless hardware stops the wlan0 stops as well.

Manual control of wifi network:

While it is unadvisable to control the settings of the network directly, it is possible to keep the wifi connection alive while stopping the wlan0 network. Since the wlan0 network is a standard network interface all ifconfig commands are available.

Advanced control:

The Wifi connection and network are documented in the OpenWRT “Wireless Configuration” wiki page at http://wiki.openwrt.org/doc/uci/wireless. NixCore X1 is configured with two different network profiles for wireless, depending on what mode the wifi connection is in; ‘apwan’ for access point mode and ‘wwan’ for station mode. These modes can be found in /etc/config/network. ‘apwan’ is static with address 192.168.2.1 and dnsmasq DHCP server enabled. ‘wwan’ is a dhcp client. DHCP server settings are found in /etc/onfig/dhcp

Ethernet

The RT5350 SoC contains hardware for 4 Ethernet PHY drivers as well as an integrated switch for routing between the Ethernet drivers. The NixCore X1 exposes Ethernet hardware #4 to the header connector.

Connection

The four connections for Ethernet are RX +/- as well as TX +/-. The RX/TX signals can not be connected directly to an Ethernet RJ-45 jack and require Ethernet magnetics to decouple the board from the Ethernet network. Many products are available with integrated decoupling magnetics:

Magnetics and RJ45 Jacks:

- MagJack - http://belfuse.com/ethernet/magjack-connector-modules/
- Sparkfun RJ45 - https://www.sparkfun.com/products/8534
- Pulse LAN Magnetics - http://www.pulseelectronics.com/products/lan

Interface:

The RT5350 network hardware is fully supported by the Linux kernel and Ethernet #4 is available to userspace applications as eth0. A network “lan” is created and a virtual interface eth0.1 is created to attach to the network. Interface eth0.1 should be used as the interface to the Ethernet hardware on the SoC. By default eth0.1 is enabled as a DHCP client for a network. It is possible to enable eth0.1 to have a static IP address or enable eth0.1 to be a DHCP server for a network.

Static configuration:

Settings for IP assignment for eth0.1 are located in the /etc/config/network file. The section “lan” is what controls the IP address assignment. By default the “option proto” setting will be “dhcp” however it can be changed to “static” for a fixed IP assignment. After changing “option proto” to “static” one must set an IP address and netmask for the interface. These options are “option ipaddr” and “option netmask”. A correct static IP interface section looks as follows:

```
config interface 'lan'
        option ifname 'eth0.1'
        option proto 'static'
        option ipaddr '192.168.2.100'
        option netmask '255.255.255.0'
```

DHCP Server:

To enable a DHCP server on eth0.1 requires the interface to be set at a static IP address. How to set a static IP is listed in this document under “Static configuration”.

Once eth0.1 has been configured for static IP address the DHCP server needs to be attached to the “lan” network. This configuration is done in the /etc/config/dhcp file on the NixCore. Within the dhcp file there is a “config dhcp ‘lan’ “ section. This section is disabled by default since the Ethernet interface is defined to be a DHCP client. The option to disable the section is “option ignore ‘1’ “, remove this line or place a # symbol in front of it to comment it out. A full DHCP configuration is below:

```
config dhcp 'lan'
        option interface 'lan'
        option start '100'
        option limit '150'
        option leasetime '12h'
        option dhcpv6 'server'
        option ra 'server'
        #option ignore '1'
```

Controlling the network:

If a user changes any information in a /etc/config/ file related to the network, the new configuration must be reloaded into the networking subsystem. OpenWRT provides two command to reload information into the network using the init.d startup scripts; ‘reload’ and ‘restart’.

The reload command tells the networking system to re-read the /etc/config/network file and updated the interface. The restart command stops the networking and dhcp server and re-starts them as if they were started in a boot up. Reload can apply some changes such as IP address or netmask while restart will include changes to the network or DHCP settings.

More Information:

The networking subsystem of the NixCore X1 is an unmodified OpenWRT system. More information can be found on the OpenWRT “Network Configuration” wiki page at http://wiki.openwrt.org/doc/uci/network

Init Scripts

OpenWRT supports a modified init.d startup script boot environment. Startup/shutdown scripts are located in /etc/init.d/ however these scripts are not executed at boot by default, when a developer adds a new script it must be “enabled” by the system. This provides the ability of “enabling” and “disabling” scripts without moving files or dealing with permissions

Enable/Disable an init script:

All scripts are located in /etc/init.d/ and have support for an ‘enable’ and ‘disable’ command similar to ‘start’ and ‘stop’.

```
Enable:
root@NixCoreX1:/# /etc/init.d/SCRIPTNAME enable
Disable:
root@NixCoreX1:/# /etc/init.d/SCRIPTNAME disenable
```

More information about init scripts can be found on the “Init Scripts” OpenWRT wiki page at http://wiki.openwrt.org/doc/techref/initscripts

Automatic Startup

The NixCore X1 has a special init script added to the stock image which allows for commands to be executed at boot time. The script starts after firewall and networking scripts complete and before the remainder of application layer programs such as uhttpd and dnsmasq.

Commands can be added to the start and stop sections of /etc/init.d/nixcore

Installing new OpenWRT Packages

The OpenWRT package manager is opkg, itself a fork of ipkg closely resembling APT/dpkg for Debian Linux. Additional software can be installed from the OpenWRT Chaos Calmer (15.05) package repository. Packages can be installed in two ways, from the internet and from local files.

Internet install

A functional internet connection is required for downloading packages directly from the OpenWRT servers.

Chaos Calmer 15.05 packages can be viewed in a web browser from the following URL:

https://downloads.openwrt.org/chaos_calmer/15.05/ramips/rt305x/packages/

The opkg information files for Chaos Calmer 15.05 is stored on the OpenWRT server, downloading these files allows opkg to have a list of the latest software available. The package folders enabled for opkg is located in /etc/opkg/distfeeds.conf, this lists all folders opkg should search for ipk files. Updating the local package list is is done by running update from a command line as follows:

opkg update

After the Packages.gz and Packages.sig files are downloaded for each enabled package folder opkg will have a list of all packages on the server. Installing a package from the server is done by running the following:

opkg install [PACKAGENAME]

Where PACKAGENAME is the short name of the package. The short name of the package is the string before the underscore '_' in the ipkg file name. As an example, to install the nano editor from the nano_2.4.1-1_ramips_24kec.ipk file, the command would be

opkg install nano

More information about opkg can be found on the OpenWRT opkg wiki page at:

http://wiki.openwrt.org/doc/techref/opkg

Local Install

OpenWRT ipk packages can be installed from a local source without internet access by passing the location of the ipk file to the opkg install command as follows:

opkg install ../mnt/usbdrive/somepackage.ipk

Since opkg does not have a list of current packages if there is a required dependency that is not installed in the system issues can arise. This method is suggested for advanced users only.

Contact Information

Sales, comments, errata	NixCore Admin	admin@nixcores.com
Engineering Support	Andrew Gaylo	drew@nixcores.com

Revisions

1.0	9/28/2015	Initial revision.
1.1	9/30/2015	Added information on how to install additional OpenWRT files to support NixCore X1 target.
1.2	10/5/2015	Added default user and password information for device
1.3	10/12/2015	Added information about opkg. Added note about 5V tolerance in electrical section. Added power to operating conditions