

# **uNetSerial Chip**

## **Technical Specification**

---

**Version Number:** 0.7  
**Date:** January 08, 2006  
**Contact:** [www.nchip.com](http://www.nchip.com)  
**Document Number:** 001-0003

Preliminary - Subject to change

© 2003-2006 nChip all rights reserved

## Table of Contents

<b>0.4</b>	<b>GENERAL INFORMATION .....</b>	<b>4</b>
0.4	SCOPE .....	4
0.4	DEFINITIONS .....	4
0.4	APPLICABLE DOCUMENTS .....	4
0.4	REVISION HISTORY .....	4
<b>2.0</b>	<b>UNETSERIAL OVERVIEW .....</b>	<b>5</b>
2.1	OVERVIEW .....	5
2.2	SUPPORTED FEATURE LIST .....	5
2.3	NON-SUPPORTED FEATURE LIST .....	5
2.4	UNETSERIAL SYSTEM BLOCK DIAGRAMS .....	6
2.4.1	<i>uNetSerial Block Diagram</i> .....	6
2.4.2	<i>Typical Usage Diagram</i> .....	6
<b>3.0</b>	<b>UNETSERIAL PIN CONFIGURATION .....</b>	<b>7</b>
3.1	UNETSERIAL CHIP PIN CONFIGURATION .....	7
3.2	UNETSERIAL MODULE PIN CONFIGURATION .....	8
3.3	UNETSERIAL CHIP I/O LIST .....	9
<b>4.0</b>	<b>EMULATION PLATFORM .....</b>	<b>10</b>
<b>5.0</b>	<b>INTERNET COMMAND PROCESSOR .....</b>	<b>11</b>
5.1	OVERVIEW .....	11
5.2	IR COMMAND PROCESSOR USAGE .....	11
5.2.1	<i>Command-Line</i> .....	11
5.2.2	<i>Operating Modes</i> .....	11
5.2.3	<i>Streaming Sockets</i> .....	13
5.3	IR COMMANDS .....	14
5.3.1	PPP COMMANDS .....	14
5.3.1.1	IRC .....	14
5.3.1.2	IRD .....	14
5.3.2	DNS COMMANDS .....	14
5.3.2.1	IRN .....	14
5.3.3	ICMP COMMANDS .....	15
5.3.3.1	IRP .....	15
5.3.4	UDP COMMANDS .....	15
5.3.4.1	IRUB .....	15
5.3.4.2	IRUP .....	15
5.3.4.3	IRUG .....	16
5.3.4.4	IRUV .....	16
5.3.5	TCP COMMANDS .....	16
5.3.5.1	IRT .....	16
5.3.5.2	IRR .....	17
5.3.5.3	IRX .....	17
5.3.6	MISC COMMANDS .....	17
5.3.6.1	IRO .....	17
5.3.6.2	IRM .....	17
5.3.7	EEPROM COMMANDS .....	18
5.3.7.1	IRES .....	18
5.3.7.2	IREL .....	18
5.3.7.3	IREI .....	18
5.3.7.4	IRE<address> .....	19
5.3.7.5	EEPROM Memory Map .....	19
5.3.8	S-REGISTER COMMANDS .....	20

5.3.8.1 S-REGISTER Map .....	20
5.3.8.2 Lower S-Registers.....	21
5.3.8.2.1 TCP0_STATUS.....	22
5.3.8.2.2 TCP1_STATUS.....	22
5.3.8.2.3 UDP_STATUS .....	22
5.3.8.2.4 FIRMWARE_VER.....	22
5.3.8.2.5 BOOT_VER .....	23
5.3.8.2.6 OUR_IPADDR.....	23
5.3.8.2.7 PRI_DNS.....	23
5.3.8.2.8 SEC_DNS.....	23
5.3.8.2.9 MODEM_BAUD.....	23
5.3.8.2.10 CONSOLE_BAUD.....	24
5.3.8.2.11 GPIO_DIR.....	24
5.3.8.2.12 GPIO_STATE.....	25
5.3.8.3 Upper S-Registers.....	25
5.3.8.3.1 UNET_CONFIG.....	25
5.3.8.3.2 ESC_CHAR.....	26
5.3.8.3.3 ESC_TIMEOUT .....	26
5.3.8.3.4 TCP_STREAM_TICK.....	27
5.3.8.3.5 UDP_STREAM_TICK.....	27
5.3.8.3.6 DNS_TIMEOUT .....	27
5.3.8.3.7 SERIAL_CONFIG.....	29
5.3.8.3.8 MODEM_BAUD_HEX .....	29
5.3.8.3.9 CONSOLE_BAUD_HEX .....	30
5.3.8.3.10 PPP_TIMEOUT .....	30
5.3.8.3.11 PPP_RX_ACCM.....	30
5.3.8.3.12 UDP_FLAGS.....	31
5.3.8.3.13 TCP_CONNECT_TIME .....	31
5.3.8.3.14 TCP_RETRANS_TIME .....	31
5.3.8.3.15 IP_TTL .....	32
5.3.8.3.16 IP_TOS.....	32
5.3.8.3.17 OUR_IP_ADDR_HEX.....	32
5.3.8.3.18 PEER_IP_ADDR_HEX .....	33
5.3.8.3.19 PRI_DNS_ADDR_HEX.....	33
5.3.8.3.20 SEC_DNS_ADDR_HEX .....	34
5.3.8.3.21 IRCMD_STATE.....	34
5.3.8.3.21 PPP_FLAGS.....	35
5.3.8.3.22 LCP_STATE .....	35
5.3.8.3.23 PAP_STATE .....	36
5.3.8.3.24 PPP_TX_MRU .....	36
<b>6.0 BOOTLOADER OPERATION .....</b>	<b>37</b>
<b>7.0 CORRUPT EEPROM RECOVERY .....</b>	<b>38</b>
<b>8.0 HARDWARE INTEGRATION .....</b>	<b>39</b>
<b>9.0 SAMPLE IR COMMAND SEQUENCES.....</b>	<b>40</b>
9.1 SAMPLE HTTP CLIENT.....	40
9.2 SAMPLE SMTP CLIENT .....	42
9.3 SAMPLE UDP TRANSACTION.....	44
9.4 SAMPLE FTP CLIENT.....	45
<b>10.0 SCHEMATICS .....</b>	<b>48</b>
10.1 UNETSERIAL MODULE SCHEMATIC .....	49
10.2 UNETSERIAL DEVELOPMENT BOARD SCHEMATIC.....	49
10.2 UNETSERIAL DEVELOPMENT BOARD SCHEMATIC.....	50
10.3 UNETSERIAL MODEM EVALUATION SCHEMATIC.....	51

## **GENERAL INFORMATION**

### **0.4 SCOPE**

This document contains an overview of the uNetSerial Chip. It defines the architecture of the chip, block diagrams, theory of operation, I/O's, timing information, features, command set and examples.

### **0.4 DEFINITIONS**

TBD	To Be Defined

### **0.4 APPLICABLE DOCUMENTS**

### **0.4 REVISION HISTORY**

Rev.	Date	Comments
0.7	01/08/06	Added FTP example, fixed bugs in S-Register descriptions, Added EEPROM recovery Chapter.
0.5	01/20/04	Cleaned up a few errors in the S-Register descriptions.
0.4	12/08/03	Added Schematics in section 9. Added uNetSerial module pin out and package diagram. Changed some of the pins on the uNetSerial chip (low impact pins). Changed the chip i/o list to show the uNetSerial module pins as well as the uNetSerial chip pins.
0.3	11/27/03	Section 5.3.5.1 IRT, no colon in-between socket and name. Figure 3 was corrupt. Added Debug Mode Bit to the UNET_CONFIG S-Register. RI signal document change in pin list.
0.2	11/22/03	Changed S-Registers to reflect GPIO and moved Upper S-Registers added bootloader section. IRC command fix, spelling and grammar fixes.
0.1	11/03/03	Initial Specification

## **2.0 uNetSERIAL OVERVIEW**

### **2.1 OVERVIEW**

The uNetSerial chip is an inexpensive, updateable TCP/IP coprocessor that connects between a host processor or microcontroller and the Internet. The uNetserial chip can connect to the Internet via dialup modems or wireless phone infrastructure (supports AMPS, CDMA, CDPD, GPRS, GSM, IDEN, PIAFS, TDMA and other protocols). uNetSerial works with GPRS modems and phones from Siemens, Wavecom, Sony, Ericsson, Nokia and others. uNetSerial's processor can negotiate a PPP (Point to Point Protocol, the standard way to connect by modem or wireless phone) connection using the standard PAP or script authentication methods. It will negotiate name-server addresses and supports full name resolution. Internet protocols and configuration parameters are stored in the uNetSerial chips FLASH and EEPROM memory and all parameters and code are upgradable.

uNetSerial connects to a device's processor via a serial connection and uNetSerial's powerful yet simple IR command Interface which enables Internet Protocols like PPP, DNS, ICMP, UDP, TCP, SNMP, POP and HTTP with 'Simple Text Strings'. These IR commands offer powerful Internet functionality including the innovative Streaming Socket Technology that allows multiple \*TCP connections to be effectively managed over a single serial port. The IR command Interface is flexible and efficient, does not limit usage, almost any Internet protocol can be implemented. It is as simple as writing a few simple commands and letting uNetSerial take care of all the complex Internet protocols.

uNetSerial and its related products are one of the simplest ways to get a device talking on the Internet. Please go to the website [www.nchip.com](http://www.nchip.com) for up to date info on innovative Internet products.

### **2.2 SUPPORTED FEATURE LIST**

The following features are supported by uNetSerial

1. Robust PPP with PAP authentication and DNS server address negotiation.
2. ICMP echo request and echo reply.
3. Automatic DNS name resolution.
4. Fully configurable (TCP/IP parameters and timeouts, Port Speed and much more) parameters that can be saved in non-volatile memory.
5. UDP Send and Receive.
6. Two TCP streaming socket connections.
7. Generic I/O pin control.
8. Two full flow-controlled serial ports with auto baud rate detection.
9. FLASH upgrade-able firmware.
10. Minimal external components required. (7.3728Mhz crystal, 4 Capacitors, 1 resistor and a 3.3 or 5volt regulator).
11. Interfaces with GPRS phones, Dialup Modems or other PPP serial devices from 2400bps to 230.5Kbps

### **2.3 NON-SUPPORTED FEATURE LIST**

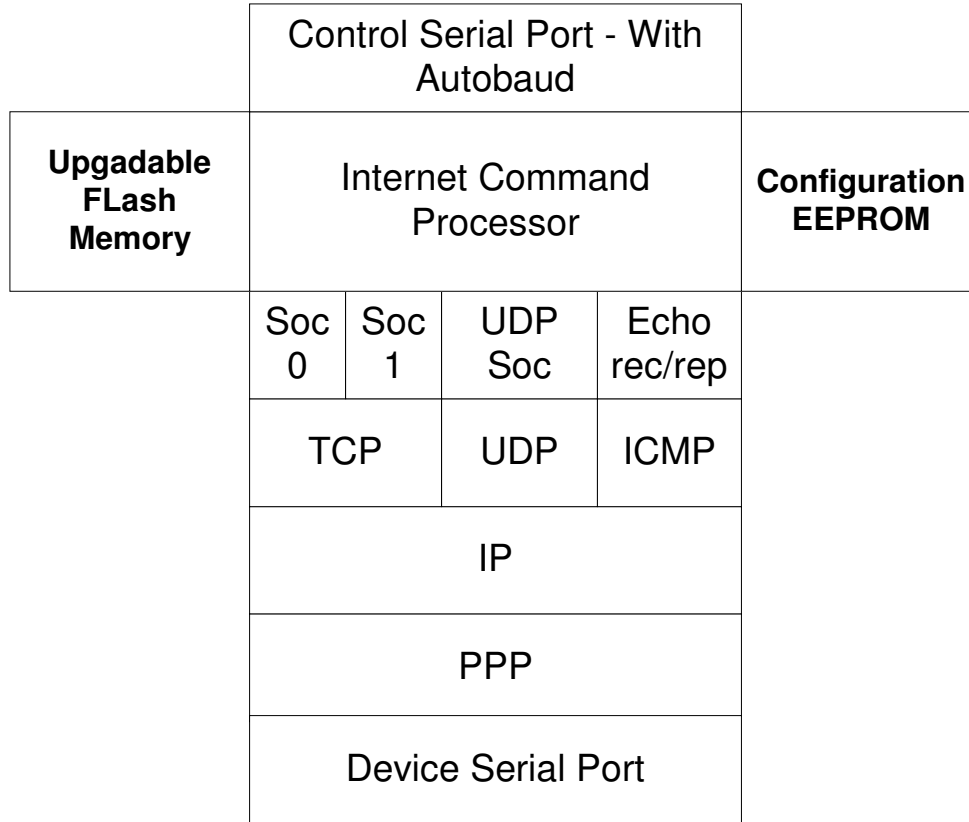
The following features are *not* currently supported by uNetSerial but are under development for a later version.

1. CHAP authentication for PPP
2. ADC support.
3. Programmable Scripting Language.

## 2.4 uNetSerial SYSTEM BLOCK DIAGRAMS

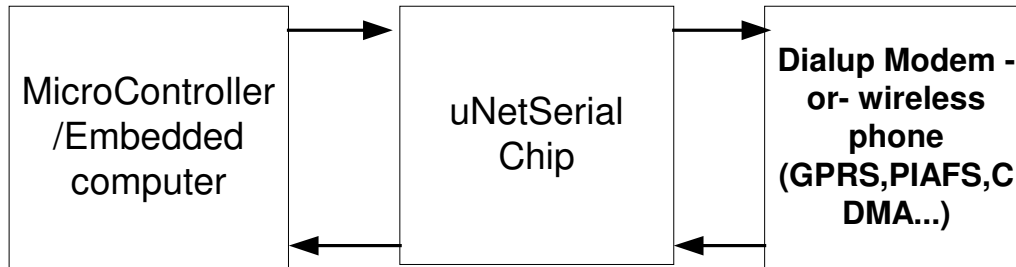
### 2.4.1 uNetSerial Block Diagram

The following block diagram depicts uNetSerial's main features.



*Figure 1) uNetSerial Block Diagram*

### 2.4.2 Typical Usage Diagram



*Figure 2) uNetSerial Usage Diagram*

### 3.0 uNetSERIAL PIN CONFIGURATION

The uNetSerial product comes in two forms, a chip and a module. The operation of the chip and module are the same and they share a related pin configuration.

#### 3.1 uNetSERIAL CHIP PIN CONFIGURATION

Figure 3 shows the uNetserial chip Pin Configuration. An example hardware schematic containing the chip is shown in section 7.

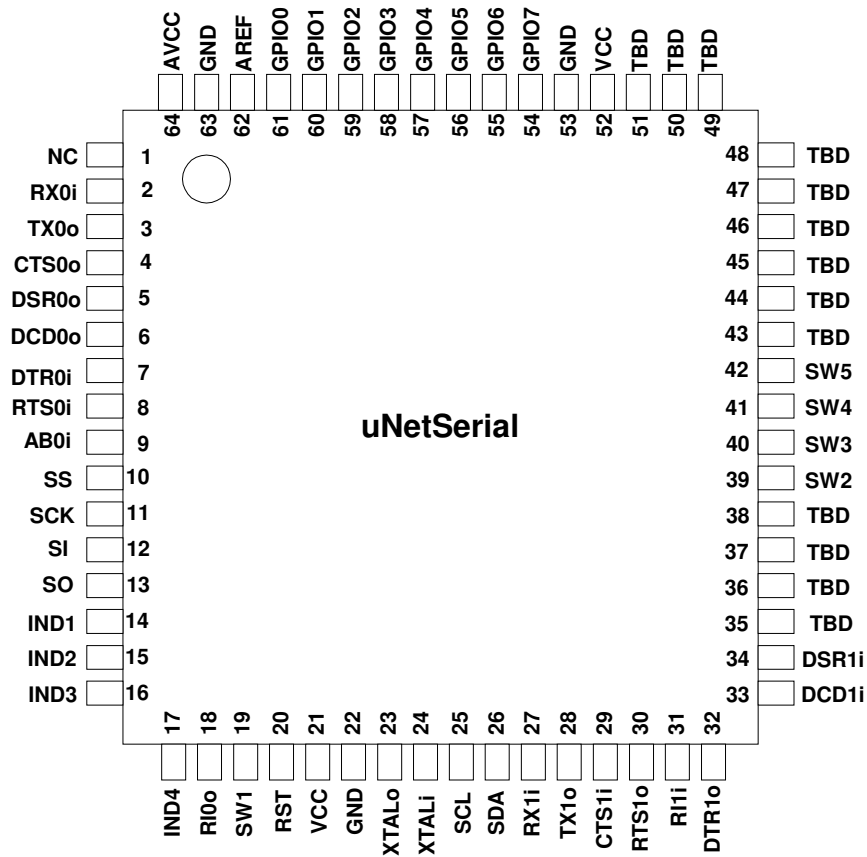


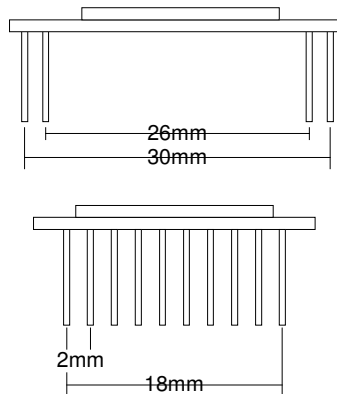
Figure 3) uNetSerial Pinout Diagram

### 3.2 UNETSERIAL MODULE PIN CONFIGURATION

Figure 4 shows the uNetserial module pin configuration. The uNetSerial module is a PCB module that contains an uNetserial Chip reset and power filter circuits and oscillator. The module pins out into standard 2mm headers. The uNetSerial Module package Diagram is shown in Figure 5.

VCC	①	②	GND	GPIO0	②①	②②	GPIO1
TX0o	③	④	RX0i	GPIO2	②③	②④	GPIO3
CTS0o	⑤	⑥	DSR0o	GPIO4	②⑤	②⑥	GPIO5
DCD0o	⑦	⑧	DTR0i	GPIO6	②⑦	②⑧	GPIO7
RTS0i	⑨	⑩	RI0o	SW5	②⑨	③⑩	SW4
SS	⑪	⑫	SCK	SW3	③①	③②	SW2
SI	⑬	⑭	SO	RX1i	③③	③④	TX1o
IND1	⑮	⑯	IND2	CTS1i	③⑤	③⑥	RTS1o
IND3	⑰	⑱	IND4	RI1i	③⑦	③⑧	DTR1o
SW1	⑲	⑳	RST	DCD1i	③⑨	④⑩	DSR1i

**Figure 4) uNetSerial Pinout Diagram**



**Figure 5) uNetSerial Module Package Diagram**



**3.3 UNETSERIAL CHIP I/O LIST**

Chip Pin	Module Pin	Signal	B/I/O	Description
1		NC	I	No Connect / Do not connect
2	4	RX0i	I	Data In – Serial Port 0
3	3	TX0o	O	Data Out – Serial Port 0
4	5	CTS0o	O	Clear To Send – Serial Port 0
5	6	DSR0o	O	Data Set Ready – Serial Port 0
6	7	DCD0o	O	Carrier Detect – Serial Port 0
7	8	DTR0I	I	Data Terminal Ready – Serial Port 0
8	8	RTS0I	I	Request To Send – Serial Port 0
9		AB0I	I	Auto Baud Port 0 – Must connect to pin 2 (RX0I) for autobaud to work. This is done in the module.
10	11	SS	I	Reserved – Do Not Connect – for future SPI interface
11	12	SCK	I	Reserved – Do Not Connect – for future SPI interface
12	13	SI	I	Reserved – Do Not Connect – for future SPI interface
13	14	SO	O	Reserved – Do Not Connect – for future SPI interface
14	15	IND1	O	Indicator 1
15	16	IND2	O	Indicator 2
16	17	IND3	O	Indicator 3
17	18	IND4	O	Indicator 4
18	10	RI0o	O	RI pass-through from modem port –or- Data Available (DAV)
19	19	SW1	I	Switch/interrupt 1
20	20	RST	I	Chip Reset, a low level pulse will cause a chip reset.
21	1	VCC	I	Supply Voltage
22	2	GND	I	System Ground
23		XTALo	O	Output from the inverting Oscillator amplifier.
24		XTALi	I	Input to the inverting Oscillator amplifier.
25		SCL	O	Reserved – Don Not Connect – for future use.
26		SDA	B	Reserved – Don Not Connect – for future use.
27	33	RX1i	I	Data In – Serial Port 1
28	34	TX1o	O	Data Out – Serial Port 1
29	35	CTS1i	I	Clear To Send – Serial Port 1
30	36	RTS1o	O	Request To Send – Serial Port 1
31	37	RI1i	O	Ring Signal – Serial Port 1
32	38	DTR1o	O	Data Terminal Ready – Serial Port 1
33	39	DCD1i	I	Carrier Detect – Serial Port 1
34	40	DSR1i	I	Data Set Ready – Serial Port 1
35-37 43-48		TBD	X	Reserved for future use, Do not connect
42	29	SW5	I	Switch5/Force BootLoader
39-41	30-32	SW2-4	I	Switch/Interrupt 2-4
52		VCC	I	
53		GND	GND	
54-61	21-28	GPIO	B	General purpose I/O
62		AREF	I	Analog Reference (Future support)
63		GND	GND	System Ground
64		AVCC	I	

**Table 1) uNetSerial I/O List****4.0 EMULATION PLATFORM**

Available is a Microsoft Windows program (Windows 2K or Windows XP) that emulates much of the uNetSerial firmware. It provides extra debugging information and is very valuable for debugging network connection issues and developing communications scenarios for use later on an embedded platform

Typically the emulation program will be named 'unetser.exe' though in certain packages it could be named something else. This program is should be run from the command line. 'unetser.exe' has two optional parameters associated with it. If specified both parameters must be specified. The first parameters is the windows COM port that the emulation platform should use for its modem connection and the second parameters is the speed that the COM port should run at. If no parameters are specified, the emulation platform will default to COM 1 at 19200bps. To exit the emulator at any time, press the <ctrl>-c key pair.

The emulator will also write a log file named unetser.txt. To get valid output into the log file, make sure you exit the emulator using the <ctrl-c> command sequence while in the emulator, this will make sure the full log file contents are flushed to the log file. This log file can be used to easily submit bugs or develop and record command sequences.

## 5.0 INTERNET COMMAND PROCESSOR

### 5.1 OVERVIEW

This section describes the usage of the IR Internet Command Processor. The Internet Command Processor provides Internet functionality for serial connected devices. It can be used in conjunction with dialup and wireless phones and modems, to enable Internet and modem functionality over a simple serial port. This section describes a IR command processor command set to provide flexible Internet functionality. The IR command processor also allows complete configuration of the uNetSerial device of which all Internet parameters, serial port configurations can be configured for any application.

### 5.2 IR COMMAND PROCESSOR USAGE

#### 5.2.1 Command-Line

The IR command interface provides specific Internet functionality. The IR commands follow similar guidelines as the standard modem AT commands.

- When using IR commands, start every command line (except the +++command) with the IR code characters.
- The commands following the IR prefix can be uppercase or lowercase or a combination of both.
- Commands must be terminated with a carriage-return character. This is ASCII 13 (0x0D).

Each command line to the IR command processor follows the format below:

**<IR><Command>{Argument}**

Note: Information above in “angle” brackets <> must be included as part of the command line, while information in “curly” braces { } may or may not be necessary as part of the command line.

- <IR>** – This is the attention code.
- <Command>** – A command consists of one letter.
- {Argument}** – Optional information specific to the Command.
- {=n}** – Used in some instances to qualify an Argument specific to the Command.

#### 5.2.2 Operating Modes

A standard modem will always be in one of two modes, the command mode and the data mode. In the **command mode**, characters received from the serial port are interpreted as commands and must follow the guidelines above. When using a modem in conjunction with the IR command processor, the IR command device (uNetSerial) will interpret both AT modem commands and IR commands. In the **data mode**, the transmitting modem received characters from the serial port, converts the data to analog signals then transmit them over a telephone line.

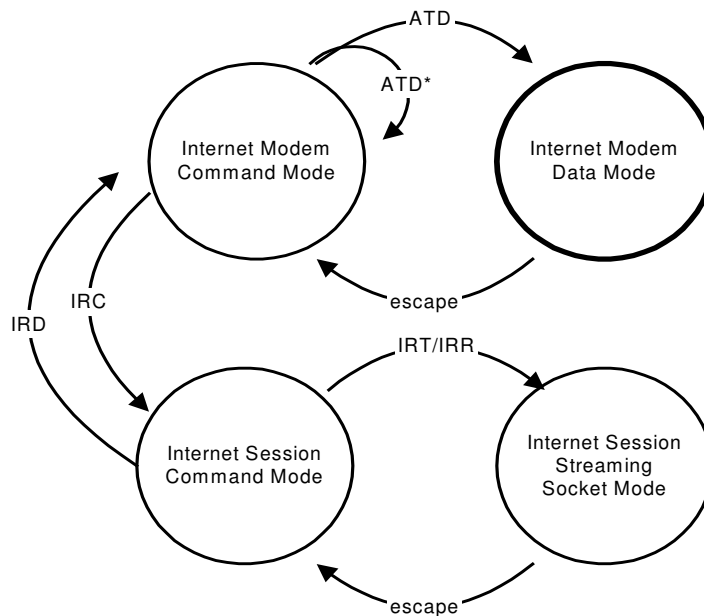
A modem transitions from the command mode to the data mode after a successful ATD command. To transition from the data mode to the command mode, an escape sequence of characters must be sent to the modem. This sequence is typically “+++”, however, modifying the modem’s S Registers can change it.

Some modems’ AT command sets supports the TIES (Timing Independent Escape Sequence) which require

an AT to follow the “+++” escape characters. Since TIES can present some problems when escape sequences are embedded in the datastream, the IR command processor use a special variable sampling algorithm that determines if the “+++” is an escape sequence. In cases where data can contain the TIES sequence, the AT command escape sequence can be turned off and the DTR line can be used as a substitute or the IRM command can escape out of modem data mode if the modem is setup for DTR pin escape.

By clearing the INBAND ESCAPE bit in IR S-register 6 escaping from data mode to Internet command mode with the “+++” sequence is disabled and you must use the DTR pin to cause the escape action. IRO will turn off the IR command processor functionality and the IM device will work like a standard modem without the IR command processor commands available. The only way to clear this condition is to reset the modem via the reset line or by power-cycling the IR command processor.

The IR command processor extends the standard modes and adds a set of ‘Internet Session Modes’, the Internet Session Command Mode and the Internet Session Streaming Socket Mode. A state diagram showing all of the IM modes is shown in figure 1.



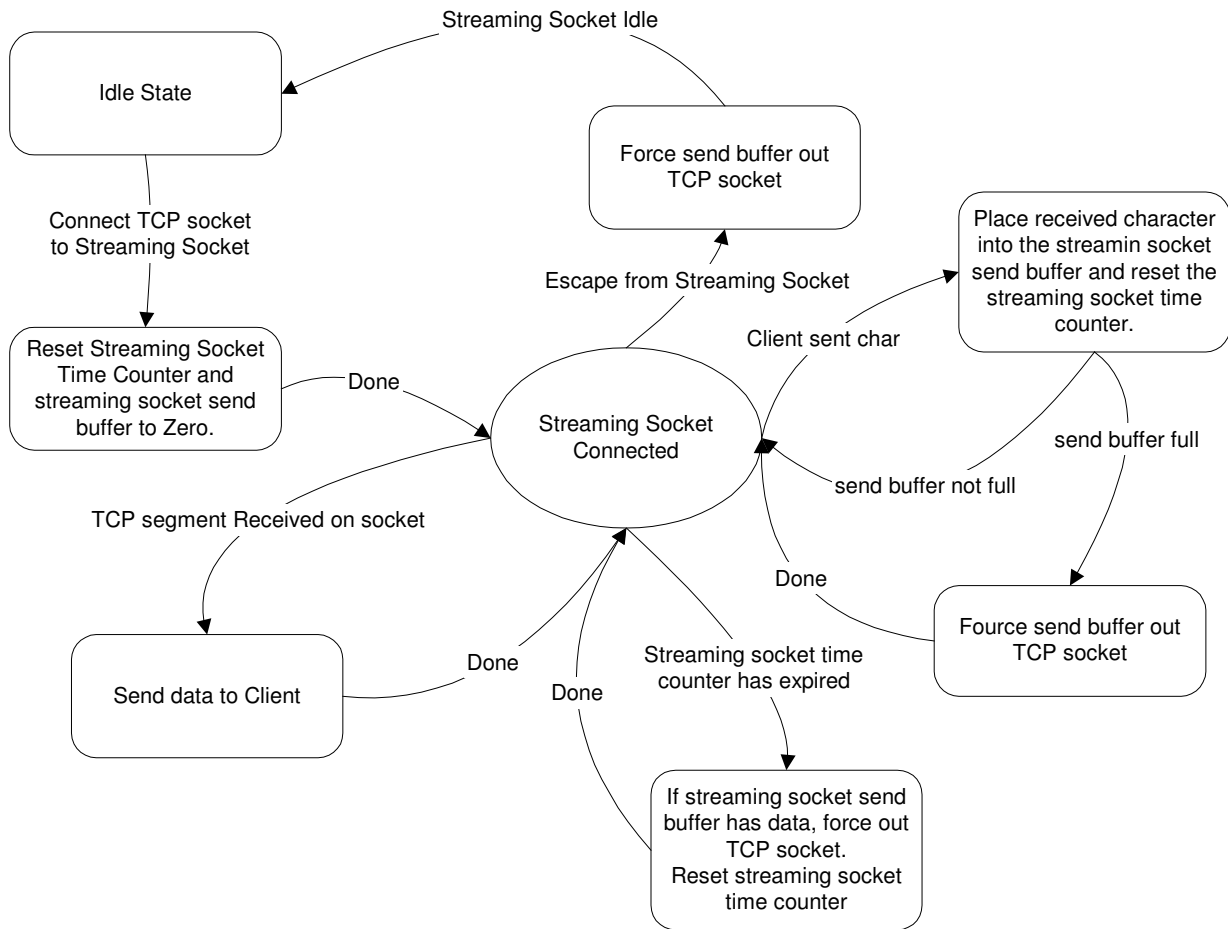
**Figure 4) uNetSerial State Diagram**

When the IM is not in the Internet Session Modes it behaves exactly like a standard modem, and can be in either the command or data mode. When in the Internet Session Modes only the IR commands are interpreted by the IM. Standard AT commands are not supported in this mode. In the Internet Session modes the IM receives characters from the serial port, converts the data into Internet protocol packets, then converts them to analog signals and transmits these signals over the telephone line.

The IM transitions to the Internet Session Command mode after a successful IRC command. It transitions from the Internet Session Command mode to the Internet Session Streaming Socket mode after a successful IRT command. To transition from the Internet Session Streaming Socket mode back to the Internet Session Command mode and escape sequence of characters must be sent to the IM. This is handled exactly as it is done in a standard modem transitioning from data mode to command mode. The IM will then transition from the Internet Session Command mode to the Internet Modem Command mode after a successful IRD command.

### 5.2.3 Streaming Sockets

Streaming Sockets are a simple yet advanced way to manage multiple TCP connections over a serial connection.



**Figure 5) Streaming Socket State Machine**

### 5.3 IR COMMANDS

This communications protocol is intended to be similar in execution to a standard modem AT commands. All commands are prefixed with the characters IR. Sending IR alone will always return OK or 0 if the IR command processor is ready to receive commands. All commands return a result code indicating the result of the command.

#### 5.3.1 PPP COMMANDS

This section describes a set of commands to controls the PPP operation of the uNetSerial Chip.

##### 5.3.1.1 IRC

Usage:

**IRC [name],[password]**

IRC will try to raise a PPP connection. Name and or Password are optional. If the PPP peer(peer server) requests PAP the device will used the passed name and password for authentication.

Returns:

CONNECT[CR/LF]	- if PPP has been raised successfully.
FAIL PPP[CR/LF]	- PPP timed out or otherwise failed to negotiate a connection
AUTH FAIL PPP[CR/LF]	- if PAP authentication failed.

##### 5.3.1.2 IRD

Usage:

**IRD**

IRD disables the PPP connection and return to command mode. This will also toggle the modem DTR pin to drop the modem/phone carrier and force it into command mode.

Returns:

OK[CR/LF]	- Always Returns OK
-----------	---------------------

#### 5.3.2 DNS COMMANDS

This section describes a set of commands that resolves hostnames into IP addresses via the Domain Name System protocol.

##### 5.3.2.1 IRN

Usage:

**IRN <name>**

This command will try to resolve a name into an IP address using the Domain Name System protocol.

Returns:

IP address[CR/LF]	- if name was correctly resolved
FAIL DNS[CR/LF]	- if DNS resolution has failed.

### 5.3.3 ICMP COMMANDS

This section describes a set of commands that generate ICMP packets.

#### 5.3.3.1 IRP

Usage:

**IRP <hostname/IP>**

Sends a ping (ICMP Echo Request) and waits for a reply.

Returns:

OK [xxx]ms[CR/LF]	- if an ping response was returned.
FAIL[CR/LF]	- if there was no response.

### 5.3.4 UDP COMMANDS

This section describes a set of commands that generate and receive UDP packets. Currently only one UDP port can be enabled for read.

#### 5.3.4.1 IRUB

Binds a UDP port to UPD for receive. (enables UDP receive on a port)

Usage:

**IRUB <port>**

Returns:

ERROR[CR/LF]	- passed command is invalid format.
OK[CR/LF]	- source port has been successfully set for receive.

#### 5.3.4.2 IRUP

Sends a UDP packet to IP:destport from ourip:sourceport. All parameters are optional. If no parameters are set the packet will be sent to the Ipaddress and Port of the last received UDP packet (or zeros if there were none). Source port, if not specified will be zero or the port set by the last IRU command.

On successful input of the IRUP command the IR command processor will return OK. Input will then be queued to send in the packet until there is no input for the UDP\_STREAM\_TICK timeout (see S-Register 0xe) or the maximum number of allowable characters have been inputted. When the packet has been sent the IR command processor returns OK.

Usage:

**IRUP [ip/name]:[destport]:[sourceport]**

Returns:

DNS FAIL[CR/LF]	- if name could not be resolved to IP
ERROR[CR/LF]	- passed command is invalid format
OK[CR/LF]	- ready to accept packet to send, then again when packet has been sent.

### 5.3.4.3 IRUG

Get a UDP packet that was received on the port set by the last IRUB command. By default the IRG command will return 'NO DATA' if there are no UDP packets waiting, or 3 lines followed by the packet data.

Usage:

**IRUG**

Returns:

NO DATA

- if there is no data waiting on the UDP socket

-or-

IP address of peer that sent packet[CR/LF]

- followed by the following if there is a UDP packet waiting.

Port of peer that send the packet[CR/LF]

Length of packet[CR/LF]

[Packet data]

### 5.3.4.4 IRUV

Stops the device from listing on the previously specified port (IRU)

Usage:

**IRUV**

Returns:

OK[CR/LF]

- Always

## 5.3.5 TCP COMMANDS

This section describes a set of commands that generate and receive TCP packets. Currently two TCP sockets (connections) can be in use at the same time.

### 5.3.5.1 IRT

Try's to connect a TCP socket to a remote host at a specified port. If command returns OK the socket is connected in streaming socket mode and any data sent or received on the socket will be sent to/from console serial port. You may go back to command mode by sending +++ or toggling the DTR pin. Going back to command mode will not close the socket and socket can be reconnected or dropped by the commands IRR and IRX respectively.

Usage:

**IRT<socket><name/ip>:<destport>**

Returns:

FAIL DNS[CR/LF]

- if name could not be resolved to IP

FAIL TCP[CR/LF]

- Could not connect to socket.

ERROR[CR/LF]

- Error in passed parameter format.



### 5.3.5.2 IRR

Reconnect to a specified TCP socket from command mode.

Usage:

**IRR <socket>**

Returns:

OK [CR/LF]

- if the socket is reconnected

FAIL SOCKET DOWN[CR/LF]

- Socket is down, cannot reconnect

### 5.3.5.3 IRX

Closes a specified TCP socket. Always returns OK

Usage:

**IRX <socket>**

Returns:

OK [CR/LF]

- Always returns OK

## 5.3.6 MISC COMMANDS

This section describes commands that control aspects of the uNetSerial Chip not related to other command groups.

### 5.3.6.1 IRO

Turn off the IR command processor. After issued uNetSerial will act as a serial pass through device. To turn the IR command processor back on, uNetSerial needs to be reset or the DTR pin needs to be toggled.

Usage:

**IRO**

Returns:

OK [CR/LF]

- Always returns OK

### 5.3.6.2 IRM

This command toggles the modem port's DTR pin <this can force modem into command mode and drop modem connection if configured on modem>.

Usage:

**IRM**

Returns:

OK[CR/LF]

- Always Returns OK

### 5.3.7 EEPROM COMMANDS

This section describes the IR commands that control the loading and saving of the configuration status into the EEPROM and other EEPROM access commands.

The baseline usage is:

IRE<I,S,L or location>[=value]

Specific usage is show below.

#### 5.3.7.1 IRES

This command saves the current running configuration state into non-volatile EEPROM memory and marks it as a valid configuration. If there is a valid configuration state currently in EEPROM memory, it is over written.

Usage:

**IREs**

Returns:

OK [CR/LF] - Always returns OK

#### 5.3.7.2 IREL

This command loads a previously saved valid configuration into the current running configuration. If the EEPROM does not contain a valid configuration the uNetSerial default configuration is loaded.

Usage:

**IREL**

Returns:

OK [CR/LF] - Always returns OK

#### 5.3.7.3 IREI

This command invalidates the EEPROM configuration valid byte. This can be issued before an 'IREL' to force uNetSerial to its default configuration. This is the same as the IRE0=0 command.

Usage:

**IREI**

Returns:

OK [CR/LF] - Always returns OK

#### 5.3.7.4 IRE<address>

This command can read or set any location in the uNetSerial's EEPROM memory. If only a address is specified this command returns, in hex, the data bytes stored in the EEPROM at the specified address. If an address and an equal value is specified, the specified value is stored in the EEPROM at the specified address.

Usage:

**IRE<location>[=value]**

Returns:

<value>[CR/LF]

-

#### 5.3.7.5 EEPROM Memory Map

The following is the EEPROM memory map used by uNetSerial

Address	Contents
0x0	This byte is 0xa5 if EEPROM configuration is valid.
0x1-0x9	Reserved
0xA-0x50	Non-volatile configuration storage that mirrors S-Registers 0-0x29
0x60- EEMAX	Reserved for command processor scripts

*Table 2) EEPROM Memory Map*

### 5.3.8 S-REGISTER COMMANDS

This section describes the IR S-Register command that access the current running configuration and state.

Usage:

IRS<register>[=value]    - view or set S-Register

Returns:

<value>                      - value of S-Register

The next section will describe the S-Register map and it usage.

#### 5.3.8.1 S-REGISTER Map

(subject to change, Serial Control not usable on windows emulation yet)

The following is the S-REGISTER map used by uNetSerial. It is broken up into 2 sections, a lower S-Register section and an Upper S-Register section. The lower S-Register section are special function S-Registers, the operation of each Lower S-Register is unique to itself, whereas the upper S-Register section registers all operate in the same manor.

Below are the Lower and Upper S-Register map tables:

Address	Register Name	Register Definition
0x0	TCP0_STATUS	TCP socket 0 status
0x1	TCP1_STATUS	TCP socket 1 status
0x2	UDP_STATUS	UDP status
0x3	FIRMWARE_VER	Firmware Version
0x4	BOOT_VER	Bootloader Version
0x5	OUR_IPADDR	IP Address
0x6	PRI_DNS	Primary DNS server address
0x7	SEC_DNS	Secondary DNS server address
0x8	MODEM_BAUD	Modem Baud Rate
0x9	CONSOLE_BAUD	Console Baud Rate
0xa	GPIO_DIR	GPIO Direction Byte
0xb	GPIO_STATE	GPIO State Byte

*Table 3) S-Register Lower Registers*

Address	Register Name	Register Definition
0x20	UNET_CONFIG	uNetSerial Config Byte
0x21	ESC_CHAR	Escape Char
0x22	ESC_TIMEOUT	Escape Timeout (Guard Time)
0x23	TCP_STREAM_TICK	TCPStreamTickTime
0x24	UDP_STREAM_TICK	UDPStreamTickTime
0x25	DNS_TIMEOUT	DNS Timeout
0x26	SERIAL_CONFIG	Serial_Config_Byte
0x27	MODEM_BAUD_HEX	ModemBaud, similar to S-Register 0x8.
0x28	CONSOLE_BAUD_HEX	ConsoleBaud similar to S-Register 0x9.
0x29	DIALUP_TIMEOUT	TBD
0x2a	PPP_TIMEOUT	PPP_Connect_Timeout
0x2b- 0x23	PPP_ACCM	ACCM to negotiate for the receive direction
0x2f	UDP_FLAGS	UDP_FLAGS
0x30	TCP_CONNECT_TIME	tcp_connect_timeout
0x31- 0x32	TCP_RETRANS_TIME	tcp_retransmit_timeout
0x33	IP_TTL	IP_TTL
0x34	IP_TOS	IP_TOS
0x35- 0x38	OUR_IPADDR_HEX	OUR_IP_ADDR
0x39- 0x3c	PEER_IPADDR_HEX	PEER_IP_ADDR
0x3d- 0x40	PRI_DNS_HEX	PRI_DNS_ADDR
0x41- 0x44	SEC_DNS_HEX	SEC_DNS_ADDR
0x45- 0x6C	PPP_USER_NAME	PPP username (zero terminated)
0x6D- 0x94	PPP_USER_PASS	PPP password (zero terminated)
0x95	IRCMD_STATE	IRCMD_STATE
0x96- 0x97	PPP_FLAGS	PPP_FLAGS
0x98	LCP_STATE	LCP_STATE
0x99	PAP_STATE	PAP_STATE
0x9A- 0x9B	PPP_TX_MRU	PPP_TX_MRU
0x9C- 0x9F	PPP_TX_ACCM	PPP_TX_ACCM

**Table 4) S-Register Upper Registers****5.3.8.2 Lower S-Registers**

The first eleven S-Registers are special S-Registers, their format is different than the Upper Range S-Registers. This section describes the lower S-Registers and there operation.

**5.3.8.2.1 TCP0\_STATUS**

This S-Register returns the status of the TCP 0 socket. This S-Register is read only.

Usage:

**IRS0**

Returns:

AA BBBB      - AA = Socket State      BBBB= Source Port of socket

State	Meaning
0x0	Socket is closed
0x1	Socket is in Listen State
0x2	Socket is in SYN Sent State
0x3	Socket is in SYN Received State
0x4	Socket is in Established State
0x5	Socket is in Closed Wait State
0x6	Socket is in Last Ack State
0x7	Socket is in FIN wait 1 state
0x8	Socket is in FIN wait 2 state
0x9	Socket is in CLOSING State
0xA	Socket is in TIME_WAIT State

***Table 4) TCP Socket State (see RFC793)***

**5.3.8.2.2 TCP1\_STATUS**

This S-Register returns the status of the TCP 1 socket. This S-Register is read only.

Usage:

**IRS1**

Returns:

AA BBBB      - AA = Socket State      BBBB= Source Port of socket

**5.3.8.2.3 UDP\_STATUS**

This S-Register returns the status of the UDP socket. This S-Register is read only. Returns length of waiting packet (0-0xff), source IP of waiting packet and source port of waiting packet.

Usage:

**IRS2**

Returns:

AA B.B.B.B CC - AA = length of waiting packet (0-255) B= source IP of waiting packet CC= source port of waiting packet.

**5.3.8.2.4 FIRMWARE\_VER**

This S-Register returns the uNetSerial's firmware version. This S-Register is read only.

Usage:

**IRS3**

Returns:

JANUARY 2006

Preliminary – Subject to change

22

uNETSERIAL

001-0003

<firmware version>[CR/LF]

#### 5.3.8.2.5 BOOT\_VER

This S-Register returns the uNetSerial's Bootloader version. This S-Register is read only. Boot Version is returned as a 4 digit hex number, for example 0104 would be boot version 1.4.

Usage:

**IRS4**

Returns:

<Bootloader version>[CR/LF]

#### 5.3.8.2.6 OUR\_IPADDR

This S-Register sets or returns the uNetSerial's IP Address.

Usage:

**IRS5**

- To return the current IP address.

-or-

**IRS5=xxx.xxx.xxx.xxx**

- To set the current IP address.

Returns:

<IP address>[CR/LF]

- current IP address Returned.

#### 5.3.8.2.7 PRI\_DNS

This S-Register sets or returns the uNetSerial's primary DNS server IP Address.

Usage:

**IRS6**

- To return the current primary DNS server IP address.

-or-

**IRS6=xxx.xxx.xxx.xxx**

- To set the current primary DNS server IP address.

Returns:

<IP address>[CR/LF]

- current primary DNS server IP address Returned.

#### 5.3.8.2.8 SEC\_DNS

This S-Register sets or returns the uNetSerial's Secondary DNS server IP Address.

Usage:

**IRS7**

- To return the current secondary DNS server IP address.

-or-

**IRS7=xxx.xxx.xxx.xxx**

- To set the current secondary DNS server IP address.

Returns:

<IP address>[CR/LF]

- current secondary DNS server IP address Returned.

#### 5.3.8.2.9 MODEM\_BAUD

This S-Register sets or returns the Modem Port Baud Rate Divider. If the command is a 'set' the Modem Ports Baud Rate is changed immediately. See Table 5 for baud rate divider values and corresponding serial port speeds.

Usage:

**IRS8**

- To return the current Modem Port Baud Rate Divider.

-or-

**IRS8=xxx**

- To set the current Modem Port Baud Rate Divider.

Returns:

<Divider Value>[CR/LF]

- current Modem Port Baud Rate Divider.

Serial Data Rate	S-Register 0x8, 0x9 Dec Value	S-Register 0x11, 0x12 Hex Value
2400bps	191	0xBF
4800bps	95	0x5F
9600bps	47	0x2F
14400bps	31	0x1F
19200bps	23	0x17
28800bps	15	0x0F
38400bps	11	0x0B
57600bps	7	0x07
76800bps	5	0x05
115200bps	3	0x03
230500bps	1	0x01

**Table 5) BAUD rate table for AVR for sreg 0x8,0x9 and sreg 0x11,0x12**

#### 5.3.8.2.10 CONSOLE\_BAUD

This S-Register sets or returns the Console Port Baud Rate Divider. If the command is a 'set' the Console Ports Baud Rate is changed immediately. . See Table 5 for baud rate divider values and corresponding serial port speeds.

Usage:

**IRS9**

- To return the current Console Port Baud Rate Divider.

-or-

**IRS9=xxx**

- To set the current Console Port Baud Rate Divider.

Returns:

<Divider Value>[CR/LF]

- current Console Port Baud Rate Divider.

#### 5.3.8.2.11 GPIO\_DIR

This S-Register sets or returns the direction setting of the GPIO Pins (Pins 54-61). GPIO\_DIR is an 8-bit hex value where each bit in the returned or set value corresponds to an individual GPIO pin. The lowest significant bit corresponds to the GPIO0 pin (Pin 61) and the most significant bit corresponds to the GPIO7 pin (Pin 54). Writing a '1' to a pins location will turn the pin into an output, driving the pin high or low depending on the setting of the GPIO\_STATE S-Register register. Writing a '0' to a pins location sets the pin up as a High-Z input who's state (high or low) can be read by the GPIO\_STATE S-Register. This register cannot be saved into the EEPROM and must be initialized if intended use differs from its default values.

Usage:

**IRSA**

- To return the current GPIO\_DIR S-Register Value

JANUARY 2006

Preliminary – Subject to change



-or-

**IRSA=xxx**

- To set the GPIO\_DIR S-Register Value

Returns:

&lt;GPIO\_DIR Value&gt;[CR/LF]

- current GPIO\_DIR S-Register Value

Default Value:

0x00

- Register always defaults to 0x00 upon power cycle.

**5.3.8.2.12 GPIO\_STATE**

This S-Register sets or returns the current value of the GPIO Pins (Pins 54-61). GPIO\_STATE is an 8-bit hex value where each bit in the returned or set value corresponds to an individual GPIO pin. The lowest significant bit corresponds to the GPIO0 pin (Pin 61) and the most significant bit corresponds to the GPIO7 pin (Pin 54). Writing a '1' to a pins location will either 1) turn on a weak pull-up of the corresponding GPIO\_DIR pin is set to input or 2) drive the pin high. Writing a '0' to a pins location drives the pin low. This register cannot be saved into the EEPROM and must be initialized if intended use differs from its default values.

Usage:

**IRSB**

- To return the current GPIO Pin state (Pins 54-61)

-or-

**IRSB=xxx**

- To set the output characteristics of the GPIO Pins

Returns:

&lt;GPIO\_STATE Value&gt;[CR/LF]

- current GPIO pin state

Default Value:

0x00

- Register always defaults to 0x00 upon power cycle.

**5.3.8.3 Upper S-Registers**

The remaining S-Registers are all read/write values that take and return hex byte values. Be careful to not try setting read only state variables without understanding the consequences.

The usage for these S-Register values are as follows:

**IRS<register>[=value]**

Register is required, equals value is optional and will set the register to the passed value. If no equals value is passed the current registers contents are returned. The value parameter is a 2-byte hex value from 00 to ff corresponding to a decimal value between 0 to 255.

**5.3.8.3.1 UNET\_CONFIG**

This S-Register sets or returns the uNetSerials configuration byte. This configuration byte controls the following features and each feature is controlled by one bit in the value set. Multiple configuration options to be set must be set by combining their corresponding bit values. The bit controls shown in Table 6.

Bit	Configuration	Action If Set
0x1	ECHO_CMD	Turns on echoing of command port input in command mode.
0x2	ECHO_STREAM	Turns on echoing of command port input in streaming socket mode.
0x4	RESULT	Turns on numeric response codes.

0x8	INBAND_ESCAPE	Turns on inband escape (+++)
0x10	DATA_MODE_ESCAPE	Turns on inband escape (+++) for modem mode.
0x20	BOOT_BANNER	Setting this bit turns on the boot banner after reset.
0x40	DEBUG_MODE	Turns on debug messages.
0x80	RESULT_OFF	No Messages will be sent by the device. Flying Blind Mode.

**Table 6) Configuration Bits for UNET\_CONFIG**

Usage:

**IRS20**

**-or-**

**IRS20=value**

Returns:

<Divider Value>[CR/LF]

- current UNET\_CONFIG value

Default Value:

27

- Register defaults to 27 (**BOOT\_BANNER** | **ECHO\_CMD** | **ECHO\_STREAM** | **RESULT**) when default configuration is loaded

#### 5.3.8.3.2 ESC\_CHAR

This S-Register sets or returns the ESC\_CHAR configuration byte. This byte defaults to '+'. Changing this value can change the inband escape character.

Usage:

**IRS21**

**-or-**

**IRS21=value**

Returns:

<Value>[CR/LF]

- current ESC\_CHAR value

Default Value:

2B

- Register defaults to 2B ('+' ASCII) when default configuration is loaded.

#### 5.3.8.3.3 ESC\_TIMEOUT

This S-Register sets or returns the ESC\_TIMEOUT configuration byte. This byte describes the inband escape timeout (or guard time) in 10's of MS. This byte defaults to 200, which is equivalent to 2 seconds. Changing this value can change the ESC\_TIMEOUT from zero to 255 (or zero to 2.55 seconds)

Usage:

**IRS22**

**-or-**

**IRS22=value**

Returns:

<Value>[CR/LF]

- current ESC\_TIMEOUT value

Default Value:

C8 - Register defaults to C8 (200 or 2 seconds) when default configuration is loaded.

#### 5.3.8.3.4 TCP\_STREAM\_TICK

This S-Register sets or returns the TCP\_STREAM\_TICK byte. This byte describes the TCP stream tick time. The TCP stream tick time is settable from 0 to 255 in 10's of MS. This byte defaults to 200, which is equivalent to 2 seconds. This value determines how long to wait for more input when in streaming socket mode. If this timeout occurs the current pending input data is sent in a TCP segment.

Usage:

**IRS23**

**-or-**

**IRS23=value**

Returns:

<Value>[CR/LF] - current TCP\_STREAM\_TICK value

Default Value:

C8 - Register defaults to C8 (200 or 2 seconds) when default configuration is loaded.

#### 5.3.8.3.5 UDP\_STREAM\_TICK

This S-Register sets or returns the UDP\_STREAM\_TICK byte. This byte describes the UDP stream tick time. The UDP stream tick time is settable from 0 to 255 in 10's of MS. This byte defaults to C8 (200 decimal), which is equivalent to 2 seconds. This value determines how long to wait for more input when in UDP send mode. If this timeout occurs the current pending input data is sent in a UDP packet.

Usage:

**IRS24**

**-or-**

**IRS24=value**

Returns:

<Value>[CR/LF] - current UDP\_STREAM\_TICK value

Default Value:

C8 - Register defaults to C8 (200 or 2 seconds) when default configuration is loaded.

#### 5.3.8.3.6 DNS\_TIMEOUT

This S-Register sets or returns the DNS\_TIMEOUT. This byte describes the DNS\_TIMEOUT in seconds. The DNS\_TIMEOUT byte is settable from 0 to 255 seconds. This byte defaults to 07 (decimal 7), which is 7 seconds. This value determines how long to wait for a DNS query response from a DNS server.

Usage:

**IRS25**

**-or-**

**IRS25=value**

Returns:

<Value>[CR/LF] - current DNS\_TIMEOUT value

Default Value:

JANUARY 2006

Preliminary – Subject to change

07

- defaults to 7 seconds when default configuration is loaded.

### 5.3.8.3.7 SERIAL\_CONFIG

This S-Register sets or returns the SERIAL\_CONFIG configuration byte. This configuration byte controls the various features of the modem and console serial ports. Table 7 lists the serial configuration features and bit value that controls each feature option.. Multiple configuration options to be set must be set by combining their corresponding bit values.

Bit	Configuration	Action If Set
0x1	MODEM_RTS_CTS	Turns on the use of RTS_CTS hardware flow-control on the modem serial port.
0x4	CONSOLE_RTS_CTS	Turns on the use of RTS_CTS hardware flow-control on the console serial port.
0x10	RI_PASSTHROUGH	Pass through the ring signal from the modem port to the console port.
0x20	CD_PASSTHROUGH	Pass through the carrier detect signal from the modem port to the console port.
0x80	AUTOBAUD	Sets auto BAUD rate detection on the console port.

**Table 7) Configuration Bits for SERIAL\_CONFIG**

Usage:

**IRS26**

**-or-**

**IRS26=value**

Returns:

<Value>[CR/LF]

- current SERIAL\_CONFIG value

Default Value:

0

- Register defaults to 0 when default configuration is loaded.

### 5.3.8.3.8 MODEM\_BAUD\_HEX

This S-Register sets or returns the MODEM\_BAUD\_HEX configuration byte. This register is similar to S-Register 0x08 but value written and returned are in hex. Also setting this register does not take effect at write time, it only sets the configuration byte. This configuration may be saved into the EEPROM and the changes will take effect after the next system reset or power-cycle.

Usage:

**IRS27**

**-or-**

**IRS27=value**

Returns:

<Value>[CR/LF]

- current MODEM\_BAUD rate setting in hex

#### 5.3.8.3.9 CONSOLE\_BAUD\_HEX

This S-Register sets or returns the CONSOLE\_BAUD\_HEX configuration byte. This register is similar to S-Register 0x09 but value written and returned are in hex. Also setting this register does not take effect at write time, it only sets the configuration byte. This configuration may be saved into the EEPROM and the changes will take effect after the next system reset or power-cycle.

Usage:

**IRS28**

**-or-**

**IRS28=value**

Returns:

<Value>[CR/LF] - current CONSOLE\_BAUD rate setting in hex

#### 5.3.8.3.10 PPP\_TIMEOUT

This S-Register sets or returns the PPP\_TIMEOUT configuration byte. This byte describes the PPP\_TIMEOUT in seconds. The PPP\_TIMEOUT byte is settable from 0 to 255 seconds. This byte defaults to 0F (decimal 15), which is 15 seconds. This value determines how long to try to negotiate a PPP connection with the PPP server.

Usage:

**IRS29**

**-or-**

**IRS29=value**

Returns:

<Value>[CR/LF] - current PPP\_TIMEOUT rate setting in hex

Default Value:

0F - Register defaults to 0F when default configuration is loaded.

#### 5.3.8.3.11 PPP\_RX\_ACCM

This is a set of S-Registers that contain the ACCM value that the PPP engine will try to negotiate for receive. There are four consecutive registers from 0x2a to 0x2d that represent the PPP\_RX\_ACCM. The first register 0x2a represents the lower significant bits of the PPP\_RX\_ACCM (which in turn represents the ASCII hex values of 0x0-0x07). The last register in the series represents the most significant bits of the PPP\_RX\_ACCM (which in turn represents the ASCII hex values of 0x18-0x1f).

Usage:

**IRS2a**

**-or-**

**IRS2a=value**

Returns:

<Value>[CR/LF] - current PPP\_RX\_ACCM lowest significant byte.

Default Value:

00 - All Registers of the series default to 0 (don't escape anything).

#### 5.3.8.3.12 UDP\_FLAGS

This S-Register sets or returns the UDP\_FLAGS configuration byte. This configuration byte controls the various features of the UDP stack. Table 8 lists the serial configuration features and bit value that controls each feature option. Multiple configuration options to be set must be set by combining their corresponding bit values.

Bit	Configuration	Action If Set
TBD		

**Table 8) Configuration Bits for UDP\_FLAGS**

Usage:

**IRS2E**

**-or-**

**IRS2E=value**

Returns:

<Value>[CR/LF] - current UDP\_FLAGS value

Default Value:

00 - Register defaults to 0 when default configuration is loaded.

#### 5.3.8.3.13 TCP\_CONNECT\_TIME

This S-Register sets or returns the TCP\_CONNECT\_TIME register. This byte describes the TCP connect timeout in seconds. The TCP\_CONNECT\_TIME byte is settable from 0 to 255 seconds. This byte defaults to 0a (decimal 10), which is 10 seconds. This value determines how long to wait for a TCP connection to establish before giving up.

Usage:

**IRS2F**

**-or-**

**IRS2F=value**

Returns:

<Value>[CR/LF] - current TCP\_CONNECT\_TIME value

Default Value:

0a - Register defaults to 10 seconds when default configuration is loaded.

#### 5.3.8.3.14 TCP\_RETRANS\_TIME

This S-Register sets or returns the TCP\_RETRANS\_TIME register. This byte describes the TCP retransmission timeout in. The TCP\_CONNECT\_TIME byte is settable from 0 to 255 in 10's of MS. This byte defaults to FA (decimal 250), which is 2.5 seconds. This value determines how long to wait for an ACK before re-transmitting as segment.

Usage:

**IRS30**

**-or-**

JANUARY 2006

Preliminary - Subject to change

**IRS30=value**

Returns:

&lt;Value&gt;[CR/LF] - current TCP\_RETRANS\_TIME value

Default Value:

FA - Register defaults to 2.5 seconds when default configuration is loaded.

**5.3.8.3.15 IP\_TTL**

This S-Register sets or returns the IP\_TTL register. This byte describes the IP time to live value transmitted in all IP packets. IP TTL defaults to 64 (decimal 100). This value determines how many hops an IP packet generated from this device can take before being discarded

Usage:

**IRS31****-or-****IRS31=value**

Returns:

&lt;Value&gt;[CR/LF] - current IP\_TTL value

Default Value:

64 - Register defaults to 100 hops

**5.3.8.3.16 IP\_TOS**

This S-Register sets or returns the IP\_TOS register. This byte describes the IP type of service value transmitted in all IP packets. IP TOS defaults to 0.

Usage:

**IRS32****-or-****IRS32=value**

Returns:

&lt;Value&gt;[CR/LF] - current IP\_TOS value

Default Value:

0 - Regular type of service

**5.3.8.3.17 OUR\_IP\_ADDR\_HEX**

This is a set of S-Registers that contains the device IP address, this can be a user set or PPP negotiated value. The PPP engine will try to negotiate this value during PPP bring up. There are four consecutive registers from 0x33 to 0x36 that represent the OUR\_IP\_ADDR. The first register 0x33 represents the lower significant byte of OUR\_IP\_ADDR (i.e. In bold 192.168.11.**150**). The last register in the series (0x36) represents the most significant byte of OUR\_IP\_ADDR (i.e. In bold **192**.168.11.150). The IP address bytes represented in this series of registers are HEX values. To access the devices IP address in decimal values please see S-Register 0x05.



Usage:

**IRS33**

**-or-**

**IRS33=value**

Returns:

<Value>[CR/LF] - current Lowest Significant byte of IP address.

Default Value:

0 - All IP address Bytes default to 0

#### 5.3.8.3.18 PEER\_IP\_ADDR\_HEX

This is a set of S-Registers that contains the PPP links peer IP address, this can be a user set or PPP negotiated value. The PPP engine will try to negotiate this value during PPP bring up. There are four consecutive registers from 0x37 to 0x3a that represent the PEER\_IP\_ADDR\_HEX. The first register 0x37 represents the lower significant byte of OUR\_IP\_ADDR (i.e. in bold 192.168.11.**150**). The last register in the series (0x3a) represents the most significant byte of OUR\_IP\_ADDR (i.e. in bold **192**.168.11.150). The IP address bytes represented in this series of registers are HEX values.

Usage:

**IRS37**

**-or-**

**IRS37=value**

Returns:

<Value>[CR/LF] - current Lowest Significant byte of the PPP peer's IP address.

Default Value:

0 - All peer's IP address Bytes default to 0

#### 5.3.8.3.19 PRI\_DNS\_ADDR\_HEX

This is a set of S-Registers that contains the primary DNS server IP address, this can be a user set or PPP negotiated value. The PPP engine will try to negotiate this value during PPP bring up. There are four consecutive registers from 0x3b to 0x3f that represent the PRI\_DNS\_ADDR\_HEX. The first register 0x3b represents the lower significant byte of primary DNS server address (i.e. in bold 192.168.11.**150**). The last register in the series (0x3f) represents the most significant byte of primary DNS server address (i.e. in bold **192**.168.11.150). The IP address bytes represented in this series of registers are HEX values. To access the primary DNS server address in dotted decimal format, see S-Register 0x6.

Usage:

**IRS3b**

**-or-**

**IRS3b=value**

Returns:

<Value>[CR/LF] - Lowest Significant byte of the primary DNS servers IP address.

Default Value:

0 - All secondary DNS server IP address Bytes default to 0

### 5.3.8.3.20 SEC\_DNS\_ADDR\_HEX

This is a set of S-Registers that contains the secondary DNS server IP address, this can be a user set or PPP negotiated value. The PPP engine will try to negotiate this value during PPP bring up. There are four consecutive registers from 0x40 to 0x43 that represent the SEC\_DNS\_ADDR\_HEX. The first register 0x40 represents the lower significant byte of primary DNS server address (i.e. in bold 192.168.11.**150**). The last register in the series (0x43) represents the most significant byte of primary DNS server address (i.e. in bold **192**.168.11.150). The IP address bytes represented in this series of registers are HEX values. To access the secondary DNS server address in dotted decimal format, see S-Register 0x7.

Usage:

**IRS40**

**-or-**

**IRS40=value**

Returns:

<Value>[CR/LF] - Lowest Significant byte of the secondary DNS servers IP address.

Default Value:

0 - All secondary DNS server IP address Bytes default to 0

### 5.3.8.3.21 IRCMD\_STATE

This is a read only S-Registers, writing to it can cause unpredictable results. This register returns the state of the IR command processor. The states are shown below in table 9. These states are also described in section 5.0 Internet Command Processor and shown in Figure 4 of that section.

Value	State
0x1	Modem Command State
0x3	Modem Data State
0x4	Internet Command State
0x6	Streaming Socket State
0x9	Modem Pass Through State

*Table 9) IR command processor state table*

Usage:

**IRS44**

**-or-**

**IRS44=value**

Returns:

<Value>[CR/LF] - Current IR command processor state.

Default Value:

01 - On Reset or power up uNetSerial defaults to Modem Command State.

**5.3.8.3.21 PPP\_FLAGS**

This is a read only S-Registers, writing to it can cause unpredictable results. This register returns the state information of the PPP Engine. The states are shown below in table 10.

Bit	Configuration	Action If Set
0x1	PPP_RX_READY	PPP is activated.
0x8	PPP_UP	PPP connection is up and connected.

**Table 10) PPP FLAGS bit table**

Usage:

**IRS45**

**-or-**

**IRS45=value**

Returns:

<Value>[CR/LF]

- Current IR command processor state

**5.3.8.3.22 LCP\_STATE**

This is a read only S-Registers, writing to it can cause unpredictable results. This register returns the state information of the LCP state machine in the PPP Engine. The states are shown below in table 11.

Bit	FLAG	State if set
0x1	LCP_TX_UP	LCP transmit direction is up
0x2	LCP_RX_UP	LCP receive direction is up
0x4	LCP_RX_AUTH	LCP requested authentication
0x8	LCP_TERM_PEER	LCP peer terminated LCP
0x10	LCP_OPT_MAGIC	LCP MAGIC number flag
0x20	LCP_OPT_PFC	LCP PFC flag
0x40	LCP_OPT_ACFC	LCP ACFC flag

**Table 11) Configuration Bits for LCP\_STATE**

Usage:

**IRS46**

**-or-**

**IRS46=value**

Returns:

<Value>[CR/LF]

- Current IR command processor state

**5.3.8.3.23 PAP\_STATE**

This is a read only S-Registers, writing to it can cause unpredictable results. This register returns the state information of the PAP state machine in the PPP Engine. The states are shown below in table 12.

Bit	FLAG	State if set
0x1	PAP_TX_UP	LCP transmit direction is up
0x20	PAP_TX_AUTH_FAIL	PAP authentication failed if set.

***Table 11) Configuration Bits for PAP\_STATE***

Usage:

**IRS47**

Returns:

<Value>[CR/LF] - Current IR command processor state

**5.3.8.3.24 PPP\_TX\_MRU**

This is a set of two read only S-Registers, writing to it can cause unpredictable results. These registers returns the negotiated maximum receive unit offered by the peer. Register 0x48 returns the lowest significant byte of the 16 bit value, and register 0x49 returns the most significant byte of the 16 bit MRU value.

Usage:

**IRS48**

Returns:

<Value>[CR/LF] - Current IR command processor state

## 6.0 BOOTLOADER OPERATION

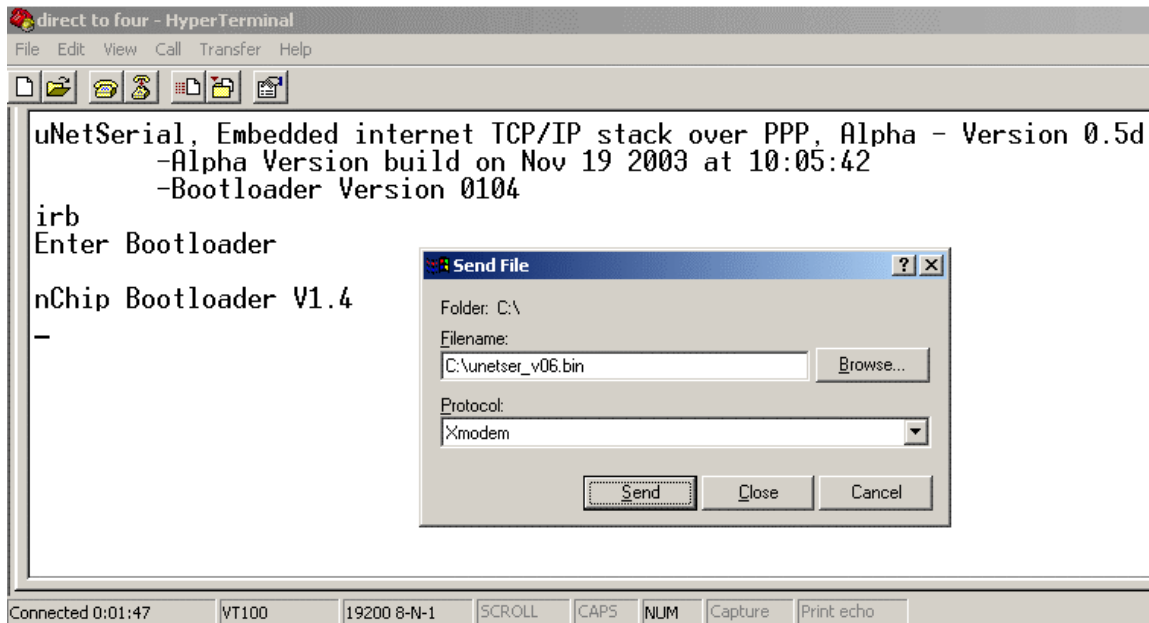
(Note: the bootloader is not available for the FREE version of the uNetSerial device)

All uNetSerial chips contain bootloaders. A bootloader allows the uNetSerial chip to be upgraded by sending the chip new code via the serial port. This allows enhancements, bug fixes and special versions of the uNetSerial firmware to be loaded at any time. There are 2 methods of entering into bootloader mode, one by issuing the 'IRB' command, and two by pulling SW5 (pin 29 on module, 42 on chip) low and holding it there during a power cycle or reset. The bootloader accepts x-modem upload of new firmware. If the bootloader is entered via reset and holding SW5 low the console serial port will be running at 19200bps. If the bootloader is entered from the uNetSerial firmware via the 'IRB' command the serial port will be running at the same rate as the uNetSerial firmware was running at.

When entering the boot loader the console port will output:

nChip Bootloader V1.4

This signifies that the bootloader is active. The following screenshot shows entering the bootloader and getting ready to send a new image via x-modem using the windows program HyperTerminal.



**Figure 6) Bootloader Operation**

Once the x-modem transfer of the new boot image has completed the new version of the firmware should startup and its version number should be printed to the screen.

## ***7.0 CORRUPT EEPROM RECOVERY***

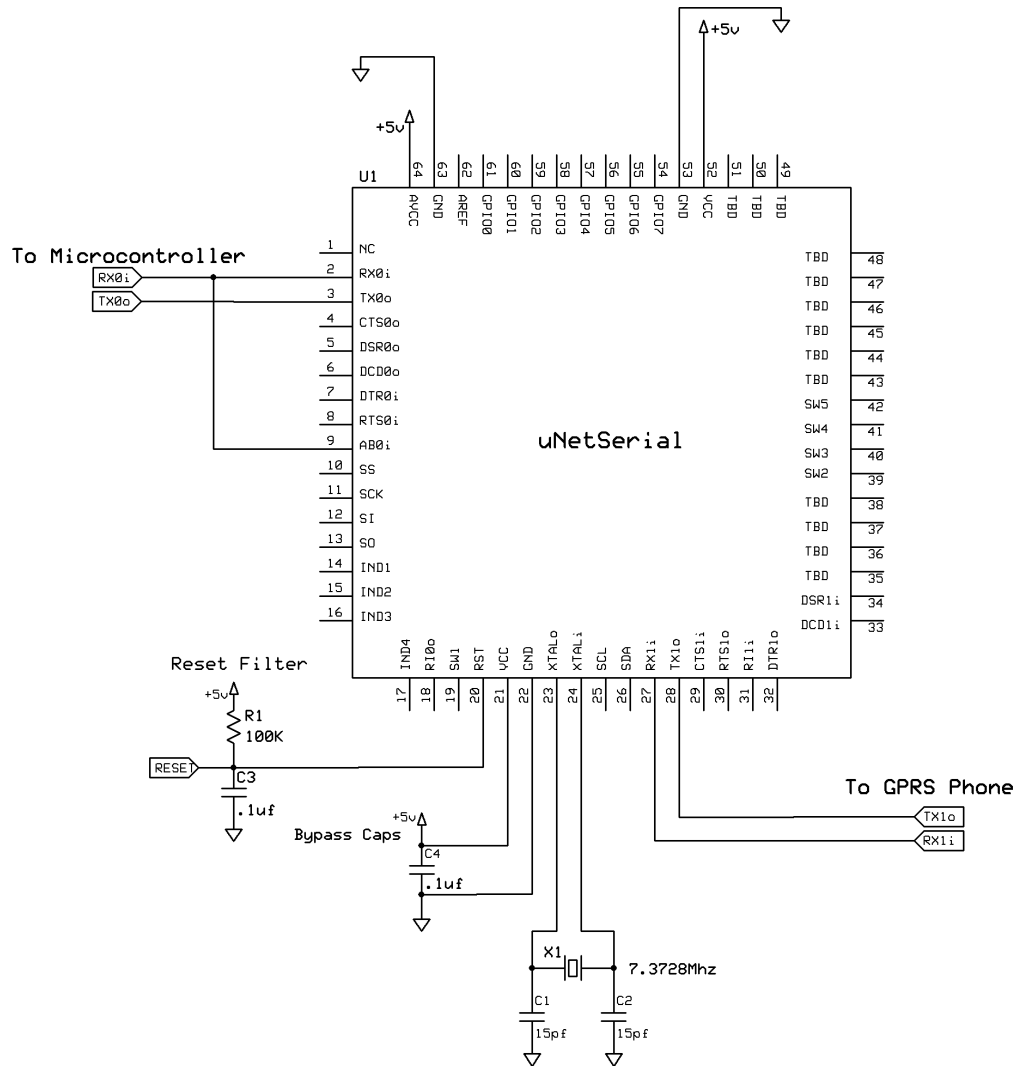
It is possible that a user could write a wrong value into a serial baud rate S-Register or other such miss-configuration that renders the uNetSerial Chip inaccessible. In these rare cases it is possible to reset the EEPROM to factory defaults by pressing and holding SW5 for 15 seconds.

Once the reset of the EEPROM has take place, Indicator 2 will light up.

After a successful reset of the EEPROM, simply reset or power cycle the uNetSerial device and it will operate in its default configuration.

## 8.0 HARDWARE INTEGRATION

Shown below is a very simple circuit that allows the uNetSerial Chip to be used. Very few components and hookups are needed. Hardware flow control can be added by connecting the hardware flow control lines. Indicators can be hooked to LED's to show console activity.



**Figure 7) Simple uNetSerial usage Schematic**

In section 9 a complete system example schematic is shown with full serial port flow control, RS232 level converters, power supply and onboard socket modem.

## 9.0 SAMPLE IR COMMAND SEQUENCES

The following is a set of sample IR command sequences for common Internet services. If you are evaluating with a terminal emulator note that internet commands need to be terminated with both a return and a linefeed, whereas IR and modem AT commands only need a return.

### 9.1 SAMPLE HTTP CLIENT

This communications protocol provides a straightforward way to transmit and receive data with an HTTP server. This is not limited to HTML content because any ASCII data can be transmitted via HTTP. This is an effective way of transmitting any ASCII data. The use of CGI can make this a powerful tool.

A simple CGI program can accept an HTTP GET with data parameters, and process the data, possibly checking for checksums included in the data, possibly parsing the data for specific information. A response can be generated on the fly and returned in the same HTTP GET transaction, possibly containing status, or configuration information. The data can then be stored in a database or simple file. This also provides a capability for dynamic formatting a presentation of the data in HTML format for a web browser. This can be done through various means, depending on the data storage and the web tools available.

For example:

1. The client device connects to the HTTP server through the uNetSerial device.
2. The client device performs a GET to a CGI program running on the server.
3. The client device sends parameter data to the CGI program in the GET URL.
4. The CGI program parses the parameter data and saves it to the server.
5. The CGI returns status to the client device in the GET return body.
6. The client device disconnects.
7. The data can be viewed through any PC web browser through another CGI program running on the server.

In the example above, the communications would look something like this:

```

→      IR
          Check to see if there is an uNetSerial device present.
← OK
→      IRC username,password
          Initiate PPP connection (assumes modem or phone is connected to service
          provider)
← CONNECT
          The PPP connection was successfully negotiated. uNetSerial is now in
          connected and command modes.
→      IRT0 www.mycal.net:80
          Connect to an TCP server at www.mycal.net and port 80 (HTTP).
← CONNECT
          The TCP connection was made for socket 0. uNetSerial is now in connected and
          online modes. (Note GET must be all caps).
→      GET /vending-demo/?id=09&slota=1&slotb=2
          This is data being sent to a HTTP Server. It is a HTTP 0.9 GET transaction to a
          CGI script that can capture the argument data (id # and slota and slotb values).
```



The script logs the data and return status in the HTTP 0.9 GET Body. Make sure both return and line feed are sent on this command. Alternatively you can request via HTTP 1.0 with 'get http://www.mycal.net/vending-demo/?id=1 HTTP/1.0' but you will get a more complex return.

← 01 10 10;

This is data being returned from the HTTP Server. It is a HTTP 0.9 GET Body. This is the data output by the CGI script (01=ok 10=new slota value 10=new slotb value ;=end data.)

→ +++

Escape uNetSerial from online mode to command mode. The uNetSerial is now in connected and command modes.

← OK

→ IRX0

Release the socket 0 connection.

← OK

The TCP socket was closed and the socket 0 was released.

→ IRD

Terminate the PPP connection.

← OK

Done.

## 9.2 SAMPLE SMTP CLIENT

This communications protocol provides a straightforward way to transmit data with an SMTP server. This is generally used to send data to a specific email address. This is an effective way of transmitting any ASCII data.

For example:

1. The client device connects to the SMTP server through the uNetSerial device.
2. The client device enters the return address of the device.
3. The client device enters one or more target addresses.
4. The client device sends data to the server.
5. The client device disconnects.
6. The data can be retrieved from the mailbox of the target address.

In the example above, the communications would look something like this:

```

→      AT
        Check to see if there is an uNetSerial device present.
←      OK
→      IRC username,password
        Connect to the POP and negotiate the PPP connection.
←      CONNECT
        The PPP connection was successfully negotiated. The uNetSerial is now in
        connected and command modes.
→      IRT0 mycal.net:25
        Connect to an TCP server at mycal.net and port 25 (SMTP).
←      CONNECT
        The TCP connection was made for socket 0. uNetSerial is now in connected and
        online modes.
←      220 mycal.com ESMTP Sendmail 8.9.3/8.9.3; Tue, 10 Dec 2000 15:27:02 -0800
        This is data being sent from the SMTP server upon receiving a TCP connection.
→      HELO remotedevice.nchip.com
        This is the client checking for an active SMTP Server session and "logging in".
        Make sure both return and line feed are sent on this command and all other
        commands sent to the server.
←      250 OK
        This is SMTP Server's reply.
→      MAIL FROM:<remotedevice@nchip.com>
        This is the client setting the return address of the message.
←      250 OK - mail from <remotedevice@nchip.com>
        This is SMTP Server's reply.
→      RCPT TO:<devicelogger@nchip.com>
        This is the client setting a recipient of the message. This command can be called
        multiple times for a single message to set multiple recipients of the message.
←      250 OK - Recipient <deviceserver@nchip.com>
        This is SMTP Server's reply.
```

- DATA
- This is the client telling the server that it is going to begin streaming the message body.
- ← 354 Send data. End with CRLF.CRLF
- This is SMTP Server's reply. The message body is terminated by a "CRLF.CRLF" sequence.
- Subject: this is a test, this is only a test
- if this were real data, it would have meaning
- .
- This is the message body. Notice it was terminated with a "CRLF.CRLF". This sequence is not included in the message data that the server receives.
- ← 250 OK
- This is SMTP Server's reply. It has sent the message at this point.
- QUIT
- This is the client terminating the SMTP Server session.
- ← 221 closing connection
- This is SMTP Server's reply. The Server then terminates the socket connection.
- ← NO CARRIER
- There is no need to escape uNetSerial from online mode to command mode because the server closed the socket. The uNetSerial is now in connected and command modes.
- IRX0
- Release the active socket connection.
- ← OK
- The TCP socket was closed and the socket 0 was released.
- IRD
- Terminate the PPP connection.
- ← OK
- Done.

### ***9.3 SAMPLE UDP TRANSACTION***

In next release.

## 9.4 SAMPLE FTP CLIENT

This communications protocol provides a straightforward way to transmit data with an FTP server. This is generally used to send or receive data to a specific file on a server. This requires both sockets to be used and uses that PASV protocol version of FTP. See **RFC 959** for more information on the FTP protocol

A FTP session would look something like this:

```

→      AT
        Check to see if there is an uNetSerial device present.
      ← OK
→      IRC username,password
        Connect to the POP and negotiate the PPP connection.
      ← CONNECT
        The PPP connection was successfully negotiated. The uNetSerial is now in
        connected and command modes.
→      IRT 0 ftp.hosts.co.uk:21
        Connect to an TCP server at ftp.hosts.co.uk and port 21 (FTP command port).
      ← CONNECT
        The TCP connection was made for socket 0. uNetSerial is now in connected and
        in streaming socket mode..
      ← 220 ProFTPD 1.2.10 Server (NamesCo Limited) [192.168.0.7]
        This is data being sent from the FTP server upon receiving a TCP connection.
→      user bigdog
        Login with user bigdog
      ← 331 Password required for bigdog
        User bigdog needs a password
→      pass nchip
        Login with user bigdog
      ← 230 User bigdog logged in.
        User bigdog is logged in, ready for ftp protocol.

→      TYPE I
        Login
      ← 200 Type set to I
        User \

→      PASV
        Set PASV mode
      ← 227 Entering Passive Mode (212,84,175,70,152,37).
        Server set for PASV mode on 212.84.175.70 port 38949 for data port
→      [guard time]+++[guard time]
        Escape to command mode, can also be done by toggling DTR. Guard time is ~1
        second or what is set in S-Register
      ← OK
  
```

UNetSerial is now in command mode.

- IRT 1 212.84.175.70:38949  
Connect seconds socket to data FTP port.  
← OK  
Socket connected
- [guard time]+++[guard time]  
Escape to command mode, can also be done by toggling DTR  
← OK  
UNetSerial is now in command mode.
- IRR0  
Return to socket 0 (command FTP socket)  
← OK  
Socket connected
- STOR /testfile.txt  
Send a data file called testfile.txt  
← 150 Opening BINARY mode data connection for /eaglepics.txt  
FTP server now ready to receive file.
- [guard time]+++[guard time]  
Escape to command mode, can also be done by toggling DTR  
← OK  
UNetSerial is now in command mode.
- IRR1  
Return to socket 1 (data FTP socket)  
← OK  
Socket connected
- EaglePICS provides a range of products and services supporting internet connectivity over GPRS networks.  
Send data to socket.
- [guard time]+++[guard time]  
Escape to command mode, can also be done by toggling DTR  
← OK  
UNetSerial is now in command mode.
- IRX1  
Close socket 1, which completes data transfer  
← OK  
Socket now closed
- IRR0  
Connect back to socket 0, command socket.  
← OK  
Connected back to socket 0.  
← 226 Transfer complete.  
Transfer complete message returned from server.
- [guard time]+++[guard time]  
Escape to command mode, can also be done by toggling DTR  
← OK

→ UNetSerial is now in command mode.  
IRX0  
Close socket 0, can also use 'quit' command  
← OK  
Socket now closed

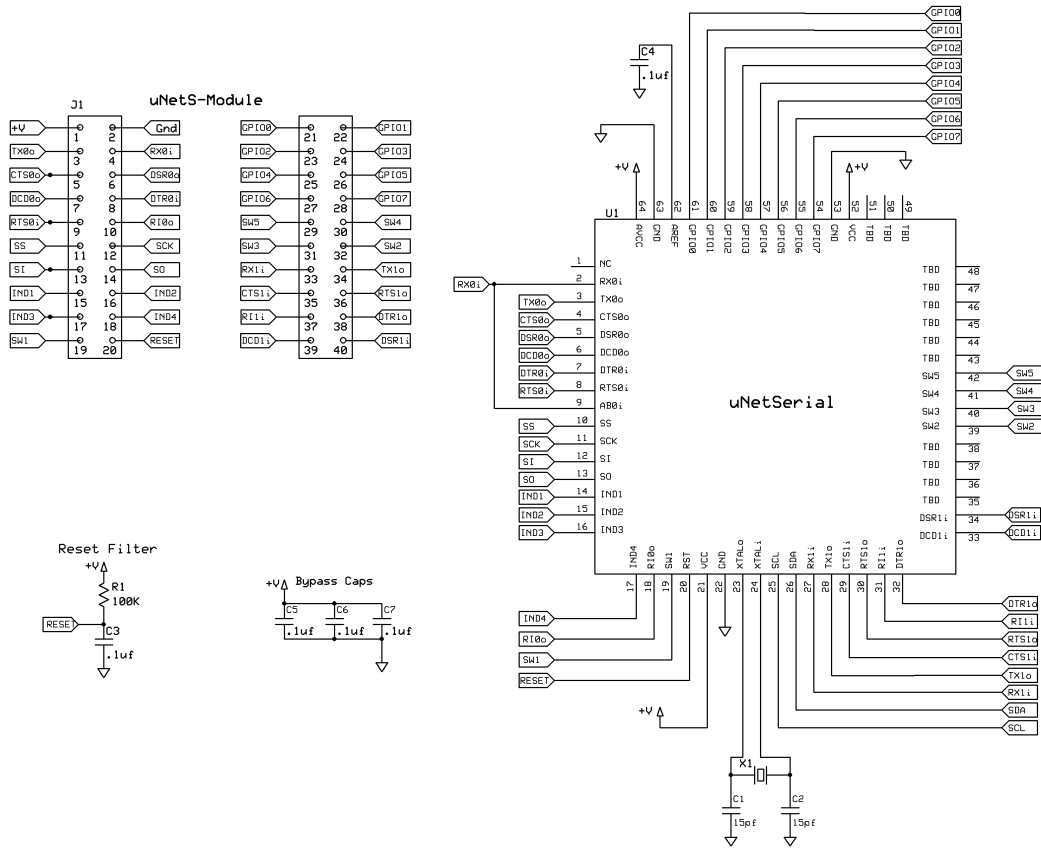
## **10.0 SCHEMATICS**

The following page contains an example system that incorporates an uNetSerial Internet command processor.

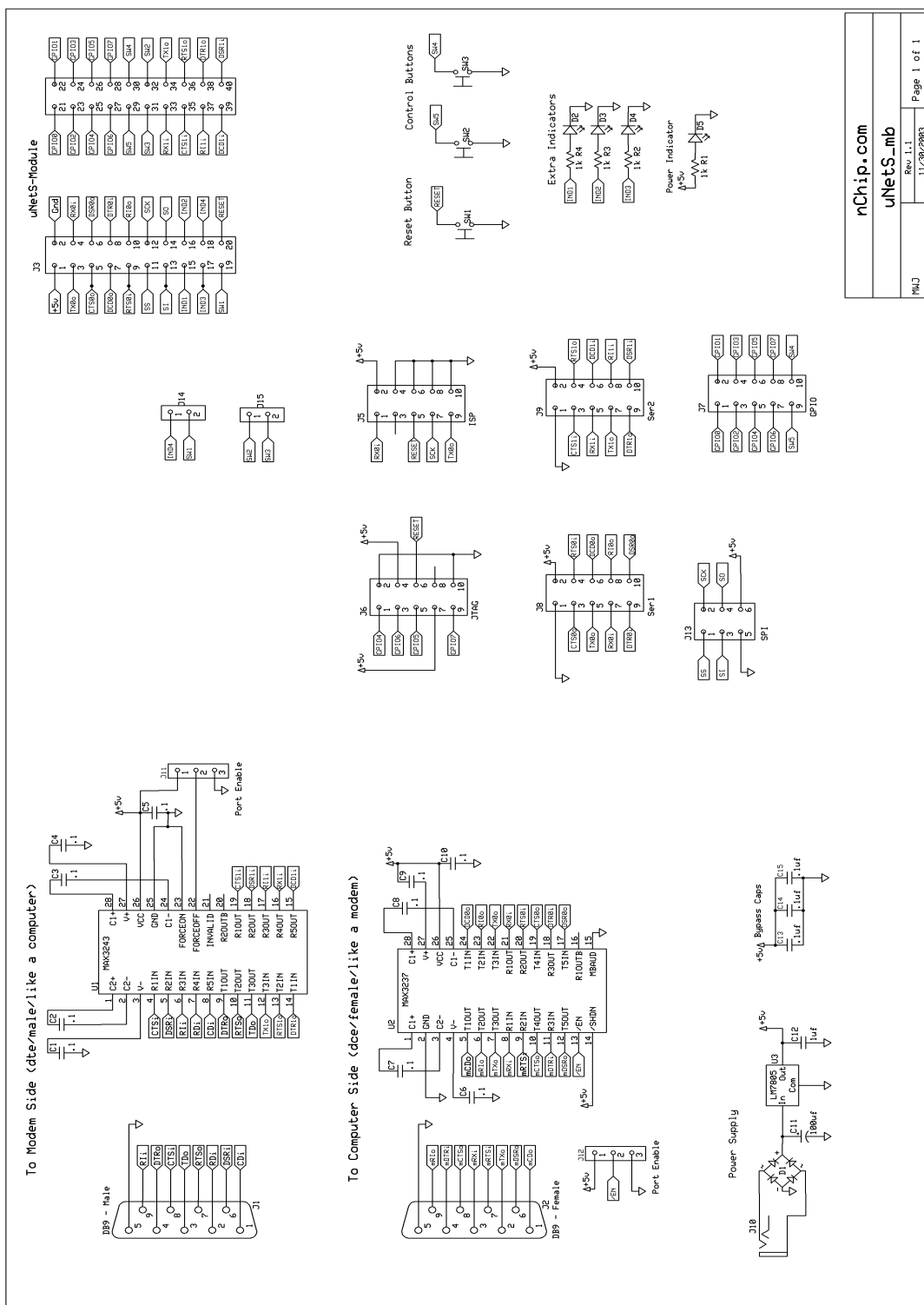
This first schematic is that of the uNetSerial Module. The second schematic is the uNetSerial development board that accepts an uNetSerial Module. The third is an example of a dialup Internet application using a wintec 2400bps modem.



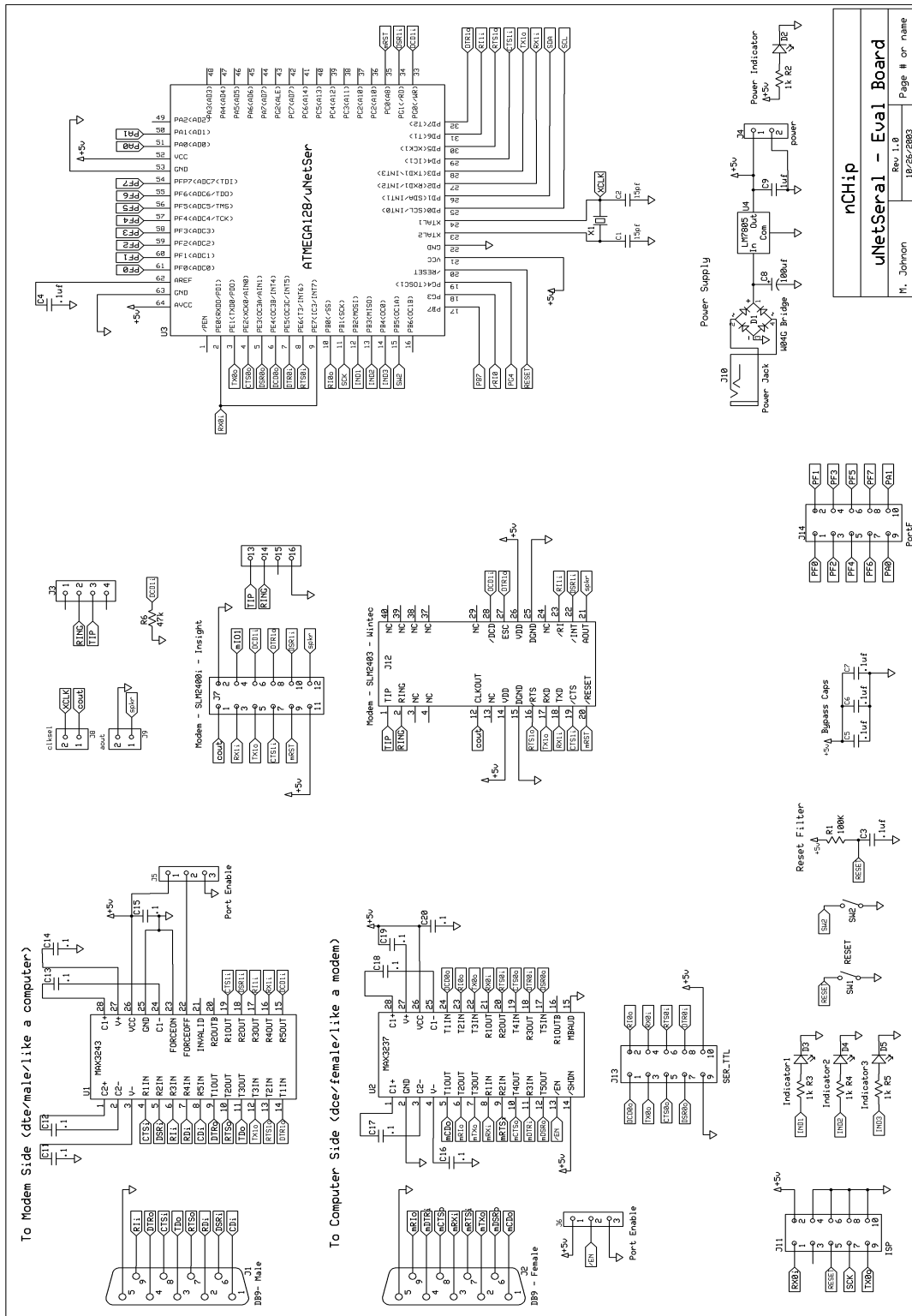
## 10.1 uNETSERIAL MODULE SCHEMATIC



## 10.2 UNETSERIAL DEVELOPMENT BOARD SCHEMATIC



### 10.3 UNETSERIAL MODEM EVALUATION SCHEMATIC



page left blank