iSX Web Server Implementation With Ethernet As The Physical Layer

SCENIX

Application Note 37

Wing Poon

July 2000

1.0 Introduction

This Application Note describes the implementation of a TCP/IP networking stack on the SX48/52BD communications controller. Scenix has created an evaluation kit for demonstrating Scenix Internet connectivity networking stack. The kit contains a demonstration/evaluation board, the SX source code, and documentation on how to use/customize the stack using the built-in Application Programming Interface (API).

For information on setting up and running the SX Ethernet demo board, please refer to the iSX Ethernet Board User's Guide.

2.0 Quick Tutorial on Internet Networking

2.1 TCP/IP

The term "TCP/IP" is commonly used whenever the topic of Internet discussed. Indeed, "TCP/IP" here refers to a whole suite of networking protocols, with the core building blocks being the IP and TCP protocols. But the point to keep in mind here, is that TCP/IP can be used to imply either of two things – the first, and more common interpretation, is that it is a collection of all standard networking protocols used for communicating on the Internet, and the second, and less common interpretation, that of the TCP protocol and IP protocol exclusively.

2.2 Packet-Based vs. Stream-Based

Fundamentally, at its core, the Internet is a packet-based network. Thus everything that flows through it has to ultimately be split into discrete packets, of data and headers, which may vary in size. Often, application programmers prefer to deal with stream-based data transfer mechanisms, which is an essentially open-ended form of communication in that the amount of data that can be transmitted is non-finite. Stream-based mechanisms often have a push-mechanism too, which is a way to 'hurry' some section of data along to its destination.

The UDP protocol is an example of a packet-based protocol. TCP is an example of a stream-based protocol. Knowing this will help you determine which type of network transport layer is suitable for your application

2.3 Ethernet

Ethernet is a shared-bus multiple-access with collision-detection communication scheme. Ethernet is defined broadly enough that it supports several physical media types. The media type used in this implementation is 10Base-T, commonly known as twisted-pair.

With Ethernet, it is important to understand the difference between a logical node address and a physical node address. A physical node address in Ethernet is a guaranteed unique 48-bit number that is assigned to every Ethernet terminal interface manufactured. A logical node address is the address that networking protocols use when directing packets. It allows a many-to-one mapping of physical addresses to a logical address. The Internet world uses "IP Address" for its logical addressing. This is a 32-bit number (commonly expressed as "w.x.y.z").

Ethernet is a best-attempt delivery network. In other words, the network will try its best to deliver a packet, once sent, to its destination. It is not a guaranteed delivery network, which means that additional software is required to ensure a reliable packet delivery or stream transport service. Furthermore, Ethernet does not guarantee in-order delivery of packets, once sent.

Ethernet is a packet-switched network. The term 'frame' is used to refer to a packet of data in Ethernet. An Ethernet frame must be at least 64 bytes in length, and no more than 1518 bytes.

To ensure data integrity (i.e. to provide error-checking, but not error-correction), Ethernet frames are constructed with a 32-bit CRC. Ethernet interfaces, when receiving Ethernet frames, are required to check the CRC field and discard (usually) the frame should the CRC not match.

2.4 iSX Ethernet Stack

One can think of TCP/IP software as a being built up of four levels of abstraction.

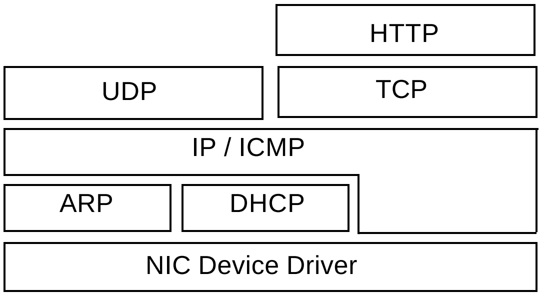


Figure2-1. iSX Ethernet Stack

At the bottom, the *Physical* layer, is the software that is specific to the physical media being used to transport the IP packets.

Above this is the *Internet* layer, which implements the protocols to enable packets to be routed from one node to another on the Internet, as well as communications test and diagnostic services.

The next layer is the *Transport* protocols layer. This layer is responsible for end-to-end communication between application programs. The protocols implemented in this layer allow multiple connections to be established by multiple application programs. These protocols can regulate the flow of data, ensure data integrity, establish and tear-down connections.

Up to this point, all the layers discussed are generally implemented by the Operating System running on the host. This is so that software developers need not reinvent the wheel when they need to communicate information across a network. Along these lines, the iSX stack provides for the same infrastructure for thin, resource-scarce, embedded devices.

Finally, capping the stack, is the Applications layer. Software at this level does not need to worry about the mechanics of how information is transported from one machine on the Internet to another. At the same time, applications are guaranteed that whatever information they want to communicate will be transported over LANs and WANs, including the globe-spanning Internet. Application layer software may implement standard services (e.g. DNS, FTP, SMTP, HTTP, etc.), or they may be proprietary customized applications whose audience is restricted to the same applications running on other hosts.

2.5 ARP

The Address Resolution Protocol (ARP) allows the dynamic mapping of logical addresses to physical addresses.

As discussed in section 2.3, Ethernet physical interface supports a unique 48-bit address. However, for efficient routing of packets, the IP networks use a 32-bit logical address. This is a hierarchical addressing scheme as it allows individual nodes to be part of a subnet, which in turn can be subsumed into a larger subnet. The Ethernet specification is independent of the logical addressing scheme used. The delivery of all Ethernet frames is specified exclusively by Ethernet physical addresses.

ARP, hence, is the bridge that maps IP addresses to physical addresses. It provides the mechanism by which a node can transmit an IP packet to a destination, whose IP address is all that is known, over an Ethernet network.

2.6 IP Datagrams

A datagram is a packet of actual data combined with a packet header. *All* Internet communication travels in the form of IP datagrams. They are the common mail pouches of the Internet world. Hence it is important to understand the role of IP, which is described next.

2.7 IP/ICMP

IP stands for Internet Protocol. It is the common denominator of the entire TCP/IP suite of protocols. The IP protocol assumes an unreliable, best-effort, connectionless packet delivery system, which is exactly what Ethernet provides.

Let us describe each of the above terms in more detail.

By *unreliable*, it is meant that a sent packet is not guaranteed to reach its destination, even if the destination is reachable. There are several reasons why a packet may not reach its destination:

1. It was corrupted en-route and discarded at some point,
2. The destination node is not connected to the network.
3. The network routing tables were incorrect thus the network does not know how to deliver the packet.
4. Network congestion has forced a router somewhere en-route to drop the packet.

Best-effort, means that the network always tries its best, given what it knows, to deliver the packet.

Connectionless, implies that there is no handshaking involved between the sender and the recipient(s) prior to the packet being sent. Senders are at liberty to send whenever they wish, subject of course to their getting time on the Ethernet, which is a shared-bus network.

P*acket*, implies that all data that traverses an IP network have to be of finite size, and accompanied by a header. IP packets are allowed to be fragmented by the network as the packet traverses the network. However, the SX-Stack does not accept fragmented packets, since it does not do packet reconstruction.

2.8 DHCP

Dynamic Host Control Protocol (DHCP) allows the use of the client-server paradigm for host machines to dynamically bootstrap, as well as configure, themselves when placed in a networked environment. Often used for its ability to dynamically assign scarce IP addresses to networked devices upon startup, DHCP is however not restricted to that, as it has the capability to configure server address, router address and any number of vendor-specific configuration parameters.

For DHCP to work, there must be both a client (such as the iSX) as well as a server(s). The client does not need to know the IP address of the server, as it will discover it for itself. The server(s) will offer an IP address to the client on request. Some servers allocate limited-time leases on IP addresses, which means the client will need to periodically renew its lease on its IP address.

2.9 UDP

The User Datagram Protocol (UDP) allows applications to send packets of data across the network to each other. It can also be used to broadcast data to multiple nodes. UDP is a *unreliable*, *connectionless* delivery service (see section 2.7).

UDP packets may be lost in the network and may arrive out of order with other sent UDP packets. However, UDP packets are checked for data integrity; so if they are received, it is safe to assume the data contained within is good.

To send a UDP packet to a destination application, the source application must know both the IP address of the destination host, as well as the port number of the destination application.

2.10 TCP

The Transmission Control Protocol (TCP) has become the protocol of choice for many applications because it allows for *connection-based*, *reliable* transport of data across an unreliable network. It does this at the cost of throughput, latency and bandwidth utilization.

Like UDP, TCP achieves multiplexing through the use of source and destination ports. A TCP ‘connection’ is defined to be a communication channel established between two ‘end-points’. An ‘end-point’ is defined to be a unique combination of both an IP address and a port number.

Before a single byte can be transmitted using TCP, a connection has to be set up between end-points. This is typically done using a three-way handshake involving the transmission of three TCP packets. The TCP protocol takes care of data sequencing, re-transmissions and data-checking.

It is important to understand that TCP is based on a client-server paradigm. This does not mean that one transmits exclusively while the other receives only. The ‘client’ is the application which *initiates* the connection with the ‘server’. To achieve that, the TCP protocol allows server applications to do a *passive-open*, which means listening on that port and accepting connections to it, while allowing client applications to do an *active-open*, which initiates the connection. To be both a ‘server’ and a ‘client’, an application generally requires the TCP/IP stack to support multiple TCP connections.

3.0 Application Programming Interface (API)

3.1 UDP Functions

3.1.1 UDPAppInit()

Application UDP Initialization code. This function is called automatically once by the stack during startup.

INPUT: none

OUTPUT: none

3.1.2 UDPAppProcPktIn()

Application Incoming UDP packet handler. This function is called whenever an application (matches udpRxDestPortxSB) packet is received. The application can call NICReadAgain() to extract sequentially extract each byte of the <data> field in the UDP packet.

INPUT: {udpRxDataLenMSB,udpRxDataLenLSB} = number of bytes in UDP <data>
{udpRxSrcPortMSB,udpRxSrcPortLSB} = UDP <source_port>

OUTPUT: none

3.1.3 UDPStartPktOut()

Starts an outgoing UDP packet by constructing an IP and UDP packet header.

INPUT: {remoteIP0-3} = destination IP addr for UDP pkt
{udpTxSrcPortMSB,udpTxSrcPortLSB} = UDP Source Port
{udpTxDestPortMSB,udpTxDestPortLSB} = UDP Destination Port
{udpTxDataLenMSB,udpTxDataLenLSB} = UDP Data Length (just data)

OUTPUT: none

3.1.4 UDPEndPktOut()

Wraps up and transmits the UDP packet.

INPUT: none

OUTPUT: none

3.1.5 NICReadAgain()

Call this function to extract, one byte at a time, the data encapsulated in the UDP packet This function should be called within UDPAppProcPktIn().

INPUT: none

OUTPUT: w = byte read

3.1.6 NICWriteAgain()

Call this function to write, on byte at a time, the data to be sent in a UDP packet.This function should be called after calling UDPStartPktOut(), and before calling UDPEndPktOut().

INPUT: w = byte to be written

OUTPUT: none

3.2 UDP Variables

3.2.1 remoteIP[3:0]

Destination IP address of outgoing packet, as well as, Source IP address of incoming packet.

3.2.2 myIP[3:0]

Source IP address of outgoing packet, as well as, filter for Destination IP address of incoming packets. This is usually set to the IP address assigned to the SX.

3.2.3 UDPRxSrcPortMSB, UDPRxSrcPortLSB

Source UDP Port number of incoming packet.

3.2.4 UDPRxDestPortMSB, UDPRxDestPortLSB

Filter for Destination Port number of incoming UDP packets.

3.2.5 UDPRxDataLenMSB, UDPRxDataLenLSB

Length, in bytes, of the data field of incoming UDP packet.

3.2.6 UDPTxSrcPortMSB, UDPTxSrcPortLSB

Source UDP Port number of outgoing packet.

3.2.7 UDPTxDestPortMSB, UDPTxDestPortLSB

Destination UDP Port number of outgoing packet.

3.2.8 UDPTxDataLenMSB, UDPTxDataLenLSB

Length, in bytes, of the data field of incoming UDP packet.

3.3 TCP Functions

3.3.1 TCPAppInit()

Application TCP Initialization code. Called repeatedly as long as TCP connection state is closed.

INPUT: none

OUTPUT: none

3.3.2 TCPAppTxBytes()

Called before transmitting a TCP packet to see if the application has any data it wishes to send. The application cannot send more than TCP_SEG_SIZE (1400) bytes at one go.

INPUT: none

OUTPUT: {tcpUnAckMSB,tcpUnAckLSB} = number of bytes to transmit

3.3.3 TCPAppRxBytes()

Indicator to the application that a packet has been received and that TCPAppRxByte is about to be called as many times as they are bytes of data.

INPUT: {tcpAppRxBytesMSB,tcpAppRxBytesLSB} = number of received data bytes

OUTPUT: none

3.3.4 TCPAppTxData()

This routine is called once for each byte the application has says it wishes to transmit.

INPUT: none

OUTPUT: w = data byte to transmit

3.3.5 TCPAppRxData()

Called once for each byte received in a packet.

INPUT: w = received data byte

OUTPUT: none

3.3.6 TCPAppTxDone()

This is called following the last call to TCPAppTxData(). It signifies the transmitted data has successfully reached the remote host.

INPUT: none

OUTPUT: none

3.3.7 TCPAppRxDone()

This is called following the last call to TCPAppRxData(). It signifies the end of the received packet.

INPUT: none

OUTPUT: none

3.3.8 TCPAppPassiveOpen()

Do a passive open. For example, listen for connections on a given port.

INPUT: {tcbLocalPortMSB, tcbLocalPortLSB} = TCP port to listen on

OUTPUT: none

3.3.9 TCPAppActiveOpen()

For example, initiate a connect to a remote TCP.

INPUT: {remoteIP0-3} = destination IP address
{tcbLocalPortMSB,tcbLocalPortLSB} = local TCP port
{{tcpRemotePortMSB,tcbRemotePortLSB} = remote TCP port

OUTPUT: none

3.3.10 TCPAppClose()

Force the current connection to close.

INPUT: none

OUTPUT: none

3.4 Variables

3.4.1 remoteIP[3:0]

IP address of remote end-point for which a TCP connection has been, or is to be, established.

3.4.2 myIP[3:0]

IP address of local end-point. This is usually set to the IP address assigned to the SX.

3.4.3 TCPLocalPortPortMSB, TCPLocalPortLSB

Local TCP port number. A TCP connection ‘end-point’ is specified by the unique pair comprising of the node’s IP address, as well as a socket, or ‘port’ number.

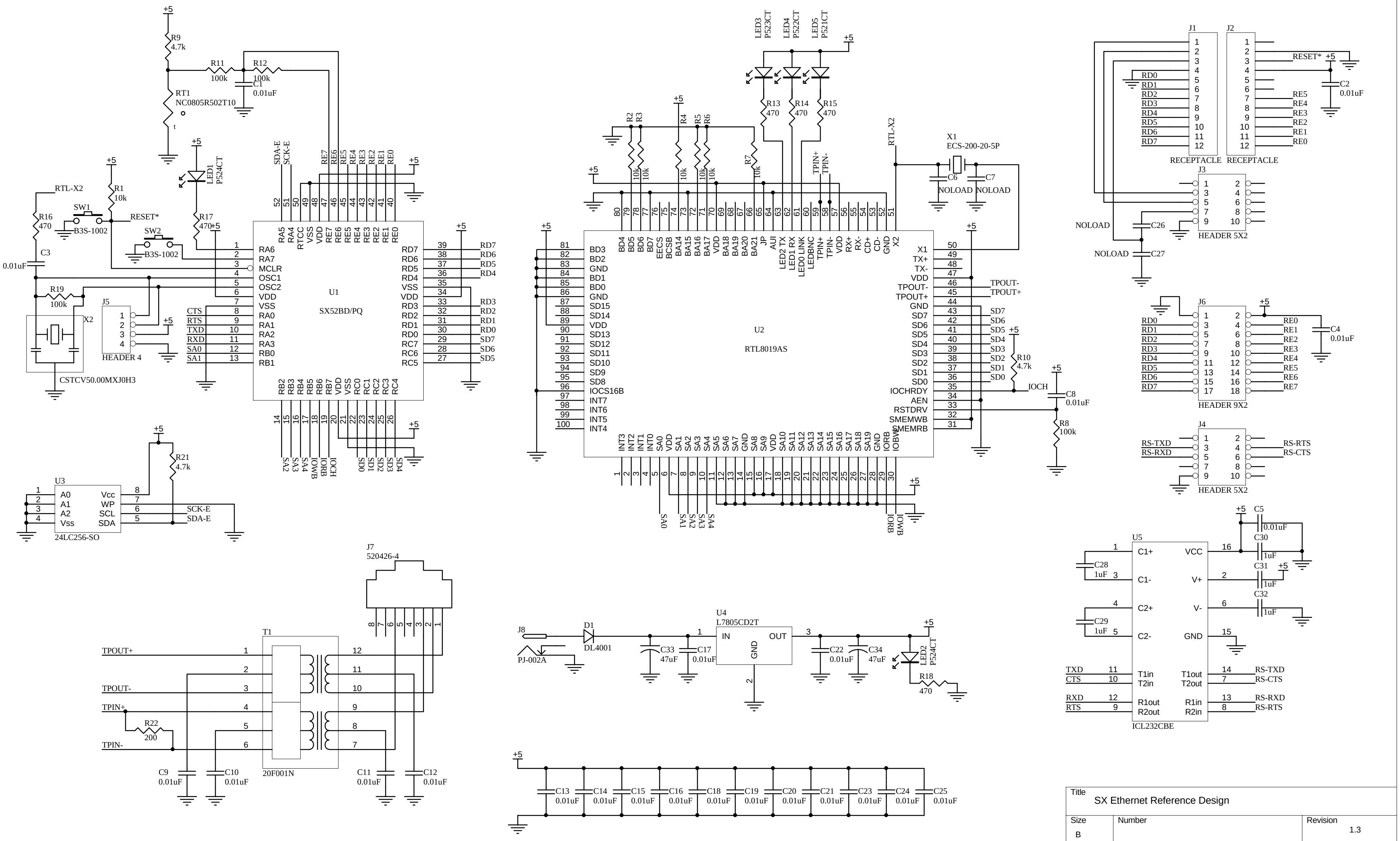
3.4.4 TCPRemotePortMSB, TCPRemotePortLSB

Remote TCP port number. A TCP connection ‘end-point’ is specified by the unique pair comprising of the node’s IP address, as well as a socket, or ‘port’ number.

Appendix A: SX Ethernet Demo Board Schematic

Following is the description of the key elements of the schematic:

- The frequency of X2, the crystal/resonator, can be changed by the user, since the TCP/IP protocol stack is non-real time and operates almost exclusively in the mainline context. Of course, certain restrictions apply. For example, the receive buffer on the NIC should not be allowed to overflow, or packets will be lost.
- R16 and C3 can be loaded in lieu of X2 for cost-reduction (will reduce SX speed to 20MIPS).
- The supplied code will only work with SX48/52 Revision 2.x silicon (the production-release version).
- Whilst the supplied reference design demonstration code works with the prescribed I/O pinouts, in general, the I/O pinouts can be changed with little modification to the code.
- Some components in the reference design are only used for demonstration purposes, please see the Bill-of-Materials section for a list of the core components.
- Connectors J1, J2, J3 and J6 are for future expansion.
- U3 is a 32kB I2C Serial EEPROM, and is used for storing web data only. Thus, users who do not need HTTP need not have this component in their design.
- U5 (and associated components) and J4 are used for reprogramming the Serial EEPROM (U3).
- R9, R11, R12, RT1 and C1 implement a temperature sensor, which will provide real-time data to be displayed on a dynamic web-page for demonstration purposes.
- LED1, R17 and SW2 are for demonstration purposes only.
- The reference design hardcodes the 48-bit Ethernet physical address in the SX, which is FLASH-reprogrammable. The user can also store the physical address in a separate configuration device by attaching a 9346 Serial 1k-bit EEPROM to BD[5:7] on U2. Refer to Sect. 6.3 of the RTL8019 datasheet for more information.
- J5 is the In-System-Programming (ISP) header. The SX can be (re)programmed using just the OSC1and OSC2 pins.
- For more information on U2, visit http://www.realtek.com.tw/cn . For information on ordering or pricing on U2, visit http://www.realtek.com.tw/cn/contact/service.htm .
- For more information on T1, visit http://www.bothhandusa.com/datasheets/filters/filters.htm .
- For information on ordering or pricing on X2, visit http://www.murata.com/murata/murata.nsf/pages/sales/#salesreps .



Title		
SX Ethernet Reference Design		
Size	Number	Revision
B		1.3
Date: 25-Jul-2000	Sheet of	
File: C:\Scenix\EthernetRefDesign\SXEthernetRefDesign.DDB	Drawn By:	
]

Appendix B: Bill of Material

The reference designators highlighted in bold are core components of the design, all other components are either used only for demonstration purposes, or whose use is optional.

Used	Part Type	Designator	Footprint	Part Field 1
19	0.01uF	C1 C2 C3 C4 C5 **C8** **C13 C14** **C15 C16** **C17 C18** **C19 C20** **C21 C22** **C23 C24** **C25**	0603	*
4	0.01uF	**C9 C10 C11** **C12**	1206	*
5	1uF	C28 C29 C30 C31 C32	0805	*
2	47uF	C33 C34	PANASONIC_ALUMSM-C	PANASONIC_ECE-V0JA470SR
4	NOLOAD	C6 C7 C26 C27	0603	*
1	DL4001	D1	DL-41	VISHAY_DL4001
2	RECEPTACLE	J1 J2	HDR12X1-100	MILLMAX_310-93-164-41-001
2	HEADER 5X2	J3 J4	HDR5X2-100	3M_2380-6121TN
1	HEADER 4	J5	HDR4X1-100	3M_2340-6111TN
1	HEADER 9X2	J6	HDR9X2-100	3M_2380-6121TN
1	520426-4	**J7**	520426-4	AMP_520426-4
1	PJ-002A	J8	PJ-002A	CUISTACK_PJ-002A
2	P524CT	LED1 LED2	0603	PANASONIC_P524CT
1	P523CT	LED3	0603	PANASONIC_P523CT
1	P522CT	LED4	0603	PANASONIC_P522CT
1	P521CT	LED5	0603	PANASONIC_P521CT
1	200	**R22**	0603	*
6	470	R13 R14 R15 R16 R17 R18	0603	*
4	4.7k	R9 **R10 R20** **R21**	0603	*
8	10k	R1 **R2 R3** **R4 R5 R6** **R7** R11	0603	*
3	100k	**R8** R12 **R19**	0603	*
1	NC0805R502T10	RT1	0805	THERMOMETRICS_NC0805R502T10
2	B3S-1002	SW1 SW2	BS3S-1002	OMRON_BS3S-1002
1	FB2022	**T1**	FB2022	BOTHHAND_FB2022
1	SX52BD/PQ	**U1**	QFP52-10X10	SCENIX_SX52BD/PQ
1	RTL8019AS	**U2**	QFP100-14X20	REALTEK_RTL8019AS
1	24LC256-SO	**U3**	SOIC8-200	MICROCHIP_24LC256-SM
1	L7805CD2T	U4	TO-263	STM_L7805CD2T
1	ICL232CBE	U5	SOIC16-300	INTERSIL_ICL232CBE
1	ECS-200-20-5P	**X1**	HC49-SM	ECS_ECS-200-20-5P
1	CSTCV50.00MXJ0H3	**X2**	MURATA_CSTCV	MURATA_CSTCV50.00MXJ0H3

Appendix C: References

1. AN23: PPP/UDP Virtual Peripheral Implementation [Scenix]
2. AN27: TCP Virtual Peripheral Implementation [Scenix]
3. AN25: HTTP Virtual Peripheral Implementation [Scenix]
4. RTL8019 Datasheet [Realtek]
5. RTL8019AS Datasheet [Realtek]
6. DP83905 Datasheet [National Semiconductor]
7. Internetworking with TCP/IP Volume I, 3rd Edition [Douglas E. Comer]
8. RFC1533: DHCP Options
9. RFC1541: DHCP
10. RFC791: IP
11. RFC792: ICMP
12. RFC793: TCP
13. RFC951: BOOTP
14. RFC768: UDP

Lit #: SXL-AN37-02

Sales and Tech Support Contact Information

For the latest contact and support information on SX devices, please visit the Scenix Semiconductor website at www.scenix.com. The site contains technical literature, local sales contacts, tech support and many other features.

SCENIX

1330 Charleston Road
Mountain View, CA 94043
Contact: Sales@scenix.com
http://www.scenix.com
Tel.: (650) 210-1500
Fax: (650) 210-8715