SMART ARM-based Microcontroller

AT16827: TCP/IP Server-Client with CycloneTCP

APPLICATION NOTE

Introduction

In today's world, computer networking has become an integral part of life. There are many different networks available to share information between groups of devices through a shared communication medium. They are mainly differentiated by the physical medium and protocol standards.

Ethernet is a prime wired networking standard which is an an obvious choice for many network applications due to reliability, efficiency, and speed. Ethernet standard is used in various application segments.

Nowadays, microcontrollers integrate peripherals to support Ethernet. Atmel® | SMART SAM E70 and SAM V71 series devices contain an inbuilt peripheral for 10/100 Mbps Ethernet MAC, compatible with IEEE 802.3 standard.

This application note address the use of Ethernet MAC (GMAC) peripheral on SAM V70/E71 devices in network applications. When discussing network applications, the basic understanding of TCP/IP protocol layers is necessary. As an introduction, this application note explains the basic concepts of TCP/IP software stacks.

The TCP/IP stack used in this application note is *CycloneTCP* from *Oryx Embedded*. The CycloneTCP is a dual IPv4/IPv6 stack dedicated to embedded applications. This application note describe following topics:

- TCP/IP Protocol Model
- Ethernet Peripheral overview
- CycloneTCP overview
- HTTP Server Implementation using CycloneTCP
- HTTP Client using CycloneTCP
- Test setup and procedure

Table of Contents

Introduction....1

1. Abbreviations....4

2. TCP/IP Protocol Model....5
   - 2.1. Layered Architecture....5
     - 2.1.1. Application Layer....5
     - 2.1.2. Transport Layer....6
     - 2.1.3. Internet Layer....6
     - 2.1.4. Network Access Layer....7
   - 2.2. Encapsulation....7
   - 2.3. Basics of TCP Client – Server communication....8
   - 2.4. TCP/IP Application Programming Interface (API)....9
     - 2.4.1. The Socket Interface....9
   - 2.5. Messages on Network Interface....12
     - 2.5.1. TCP/IP Socket Opening Sequence....12
     - 2.5.2. TCP/IP Socket Connection Sequence....12
     - 2.5.3. TCP/IP Socket Data Transfer Sequence....13
     - 2.5.4. TCP/IP Socket Closing Sequence....14
   - 2.6. Summary of the section....15

3. Ethernet Peripheral Overview....16
   - 3.1. Ethernet Frame....16
   - 3.2. Ethernet MAC (GMAC)....17
     - 3.2.1. GMAC Features....17
     - 3.2.2. GMAC Memory....19
     - 3.2.3. GMAC - Transmit Block....20
     - 3.2.4. GMAC - Receive Block....21
     - 3.2.5. GMAC- Frame Filtering....22
     - 3.2.6. GMAC- PHY interfacing....22
     - 3.2.7. GMAC - Other functions....22
   - 3.3. Ethernet PHY....23
   - 3.4. Summary of the section....25

4. CycloneTCP Stack Overview....26
   - 4.1. CycloneTCP Features....26
   - 4.2. Supported Operating Systems....28
   - 4.3. Source Files....28

5. Demo Applications with CycloneTCP....29
   - 5.1. HTTP Web-Server Demo....29
   - 5.2. HTTP Client Demo....32
   - 5.3. Demo Test Procedure....35
     - 5.3.1. HTTP Web-server Demo....35
     - 5.3.2. HTTP Client on SAM V71....38

6. Conclusion....41

7. Suggested reading....42

7.1. Atmel SMART SAM E70 and V71 Datasheet....42
7.2. ARM Documentation on Cortex-M7 core....42
7.3. CycloneTCP Documentation....42
7.4. FreeRTOS documentation....42
7.5. Ethernet PHY....42
7.6. Networking Reference Standards....42

8. Revision history....44

1. Abbreviations

ACK	Acknowledgment; refer to ACK bit in TCP header
ARP	Address Resolution Protocol
CRC	Cyclic Redundancy Check
CSMA/CD	Carrier sense multiple access / collision detection
DMA	Direct Memory Access
FIN	Finish; refer to FIN bit in TCP header
HTTP	HyperText Transfer Protocol
HTTPS	HyperText Transfer Protocol Secure
ICMP	Internet Control Message Protocol
IP	Internet Protocol
LAN	Local area network
MAC	Media access control
Mbps	Megabits per second
MDI	Medium Dependent Interface or Management Data Input
MDIO	Management Data Input/Output
MDO	Management Data Output
MII	Media Independent Interface
NAT	Network Address Translation
OSI	Open System Interconnect (joint ISO and ITU standard)
PHY	Ethernet physical layer
RMII	Reduced Media Independent Interface: A 2-bit version of the MII
SYN	Synchronize; refer to SYN bit in TCP header
TCP	Transmission Control Protocol
TCP/IP	Transmission Control Protocol/Internet Protocol
UDP	User Datagram Protocol
UTP	Unshielded Twisted Pair cable

2. TCP/IP Protocol Model

The TCP/IP protocol handles information transfer between various nodes of a network. The layered architecture of TCP/IP provides multiple options for the physical medium to transfer data.

Ethernet is one such option that constitute the lower layers of TCP/IP protocol. Ethernet standard describes how networked devices can format data for transmission to other devices on the same network.

Before understanding more about Ethernet MAC peripheral and using a TCP/IP protocol in applications, this document introduces the layered architecture of TCP/IP protocol and functionality of each layer and how protocol layers interact.

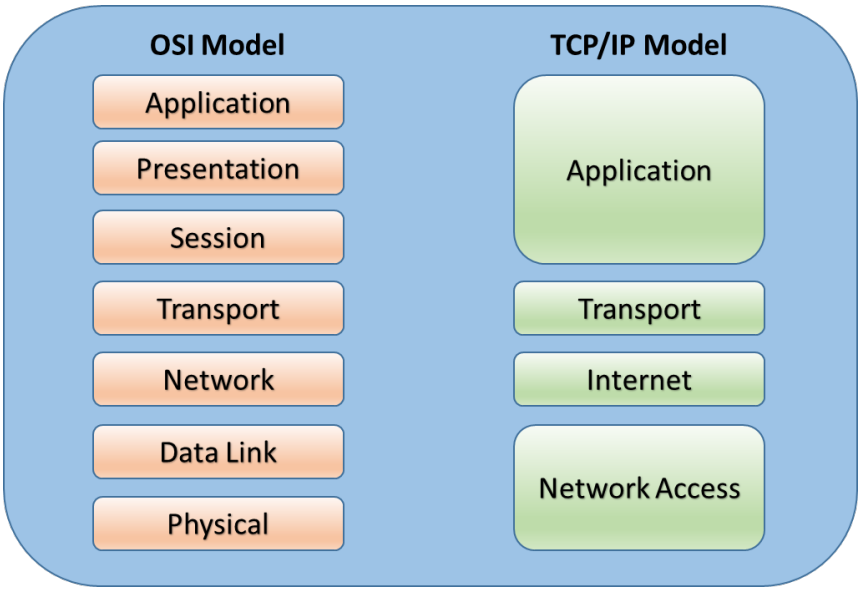
2.1. Layered Architecture

All network communication protocols follow OSI reference model from ISO (International organization of Standardization). But for internet, the preferred choice is TCP/IP model.

The TCP/IP model with four layers maps to the seven protocol layers of OSI reference model.

The following diagram provides a comparison between the OSI layers against different layers of TCP/IP stack.

Figure 2-1. OSI Model vs TCP/IP Model



2.1.1. Application Layer

The application layer is the topmost layer in TCP/IP model. This layer process the data, receive and transmit to the lower layers of TCP/IP model. It also implements the application level protocol for exchanging data with other network nodes.

The most widely known application layer protocols are:

- The Hypertext Transfer Protocol (HTTP) - used to transfer files that make up the web pages of the World Wide Web.
- The File Transfer Protocol (FTP) - used for interactive file transfer.
- The Simple Mail Transfer Protocol (SMTP) - used for the transfer of mail messages and attachments.

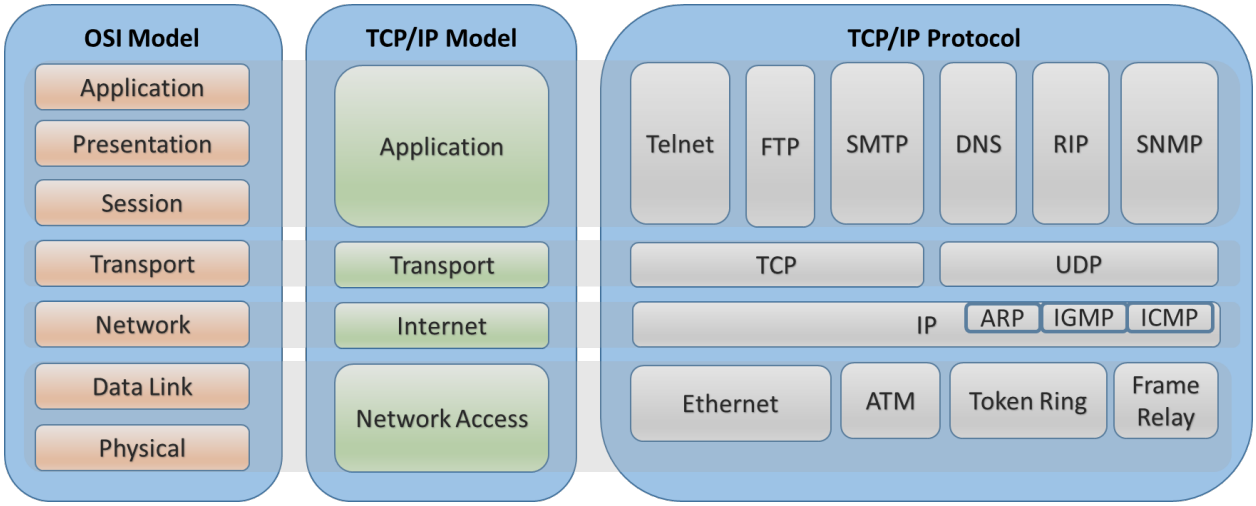
- Telnet - a terminal emulation protocol used for remotely logging on to network hosts.

Additionally, the following application layer protocols help facilitate the use and management of TCP/IP networks:

- The Domain Name System (DNS) is used to resolve a host name to an IP address.
- The Routing Information Protocol (RIP) is a routing protocol that routers use to exchange routing information on an IP network.
- The Simple Network Management Protocol (SNMP) is used between a network management console and network devices (routers, bridges, intelligent hubs) to collect and exchange network management information.

The protocols in each layers of TCP/IP stack is shown in the following illustration.

Figure 2-2. TCP/IP Protocol Stack



2.1.2. Transport Layer

The transport layer of the TCP/IP model maps fairly closely to the transport layer of the OSI model. The core protocols of the transport layer are Transmission Control Protocol (TCP) and the User Datagram Protocol (UDP).

- TCP provides a one-to-one, connection-oriented, reliable communications service. TCP is responsible for the establishment of a TCP connection, the sequencing and acknowledgment of packets sent and the recovery of packets lost during transmission.

- UDP provides a one-to-one or one-to-many, connectionless, unreliable communication service. UDP is used when the amount of data to be transferred is small (such as the data that would fit into a single packet), when the overhead of establishing a TCP connection is not desired or when the applications or upper layer protocols provide reliable delivery.

The transport layer of TCP/IP model handles the responsibilities defined for transport layer and some of the responsibilities of the session layer of OSI model.

2.1.3. Internet Layer

The internet layer of the TCP/IP model maps to the network layer of the OSI model. Consequently, the internet layer is sometimes referred to as the network layer.

The internet layer is responsible for addressing, packaging, and routing functions. The main protocols of the internet layer are IP, ARP, ICMP, and IGMP.

- The Internet Protocol (IP) is a routable protocol responsible for IP addressing, routing, and the fragmentation and reassembly of packets.
- The Address Resolution Protocol (ARP) is responsible for the conversion of the Internet layer address to the hardware address like mac address.
- The Internet Control Message Protocol (ICMP) is responsible for providing diagnostic functions and reporting errors due to the unsuccessful delivery of IP packets.
- The Internet Group Management Protocol (IGMP) is responsible for the management of IP multicast groups.

2.1.4. Network Access Layer

The lowest layer of the TCP/IP protocol stack is the network access layer. The network access layer contains two sublayers, the media access control (MAC) sublayer and the physical sublayer. The MAC sublayer is equivalent to the data link layer of the OSI model and physical sublayer aligns with the physical layer of the OSI model.

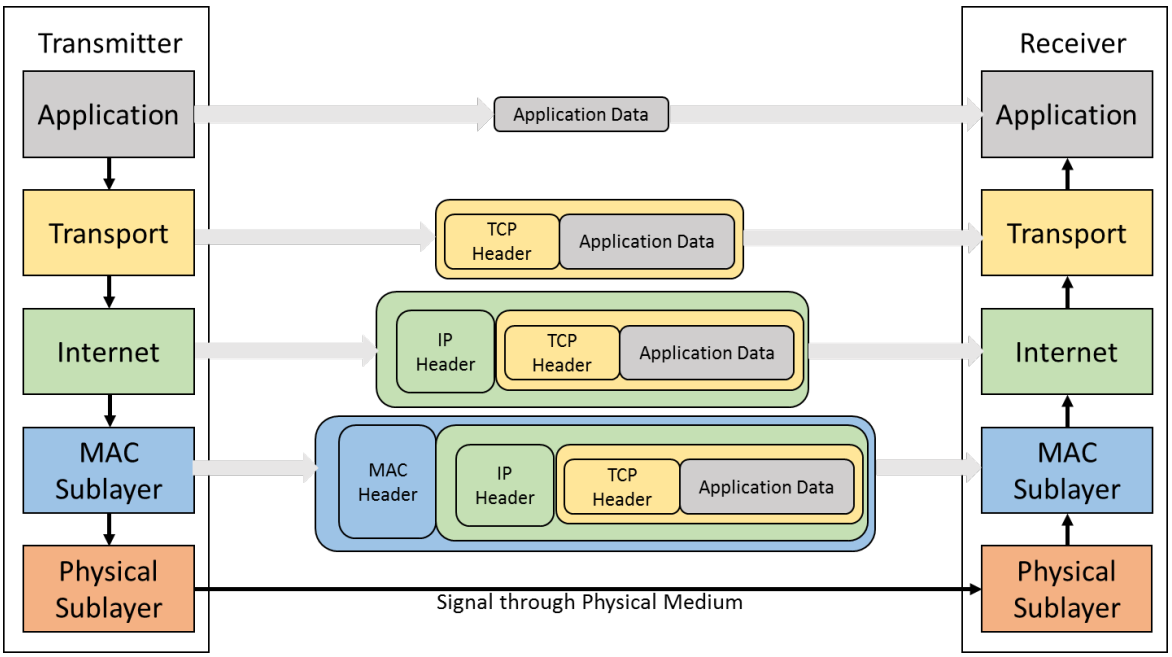
Network Access layer is responsible for placing TCP/IP packets on the network medium and receiving TCP/IP packets from the network medium.

The layered architecture of TCP/IP protocol helps in supporting different physical medium, frame formats and network access methods. Few examples are Ethernet, Wi-Fi, ATM etc.

2.2. Encapsulation

During transmission, the data from higher TCP/IP layer is wrapped inside the header of the layer below. Similarly during reception, the higher layer data is extracted from the lower layer data by removing the wrappers. This is represented in following image.

Figure 2-3. Encapsulation



Each layer of the protocol stack is responsible for a particular level of functionality. For example, the physical layer takes care of electrical transmission of bits across a medium. Each higher layer in the stack utilizes the underlying layers. There is no overlap in functions between the layers.

For example, consider the case of a web browser session. The HTTP request from application layer will be passed down to TCP layer. It will construct a TCP packet consisting of a TCP header and TCP data.

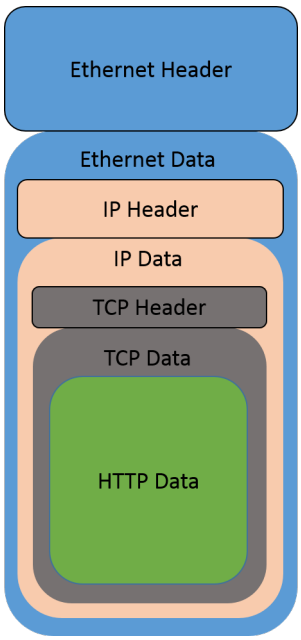
The TCP header contains information particular to the TCP protocol, such as packet sequencing information, checksum information and the source and destination port number.

This packet will be sent down to IP protocol level. An IP datagram is constructed and it contains an IP header and IP data. The IP header contains information such as the type of service, checksum information, protocol type and the source and destination IP addresses. The data field of the IP datagram contains the complete TCP packet to be transmitted.

At the data link/physical layer, the IP datagram is transported across the network using the IEEE 802.3 protocol. A MAC (IEEE 802.3) frame consists of a MAC header and a MAC payload (data). The MAC header contains information about the MAC frame, such as the source MAC address, the destination MAC address and the length of the frame. The payload field contains the complete IP datagram to be transported.

The concept of encapsulation is explained in following figure.

Figure 2-4. Application Data Encapsulation



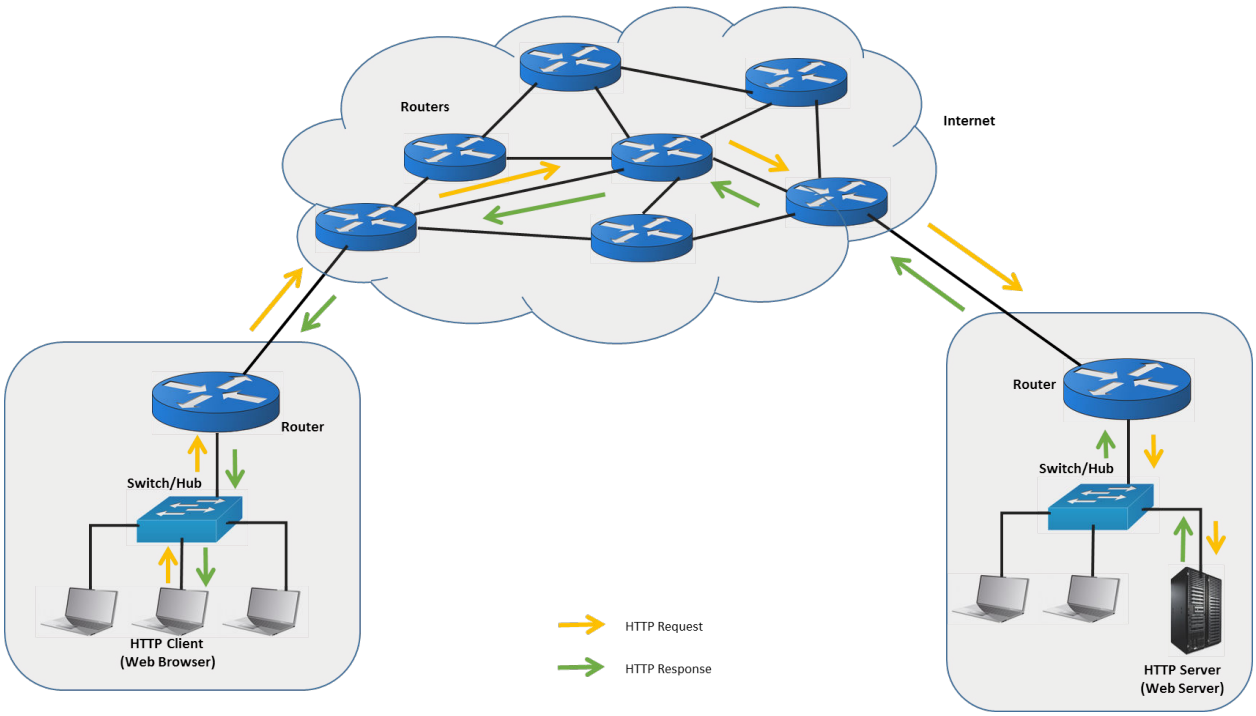
Similarly, when ethernet packets are received, it will pass through each of the above TCP/IP protocol layers. The different layers remove its wrappers and finally presents the application data to application layer.

2.3. Basics of TCP Client – Server communication

TCP is a peer-to-peer, connection-oriented protocol. There are no master/subordinate relationships. However, the applications typically use a client/server model for communication.

A *server* is an application that offers a service to internet users. A *client* is a requester of a service. An application consists of both a server and a client part, which can run on the same or on different systems. Users usually invoke the client part of the application, which builds a *request* for a particular service and sends it to the server part of the application using TCP/IP.

Figure 2-5. Ethernet Network



The server is a program that receives a request, performs the required service and sends back the results in a reply. A server can usually deal with multiple requests from same or different clients at the same time.

In real network scenario, the client and server may exist in different networks as shown above. These networks will be interconnected using many routers. The request from clients has to pass through many networks to reach the destined server. The routers with routing tables helps in finding the right direction. The routers will not check the entire message to direct the data to next router which is closer to the destination. Routing is done with with the destination IP address extracted from the received packet.

2.4. TCP/IP Application Programming Interface (API)

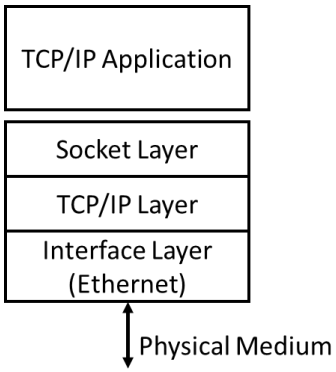
Different tasks in network application is implemented with standard application program interfaces (API). There are different types APIs used for network programming. The socket is one such application programming interface, usually provided by operating system, for programming networks at the transport layer.

Do not confuse socket as any physical entity; it is a software handle to establish connection between two network nodes.

2.4.1. The Socket Interface

Socket is standard set of function calls (API) used at application level for Interprocess Communications (IPC). It is protocol and language independent.

Figure 2-6. Socket Layer



Sockets generally relies upon client/server architecture. For TCP communications, one host listens for incoming connection requests. When a request arrives, the server host accepts it and data is transferred between the hosts.

The socket API makes use of two mechanisms to deliver data to the application level: ports and sockets.

All TCP/IP stacks have 65,536 ports for both TCP and UDP. A port is not a physical interface. It is a concept that simplifies the concept of internet communication.

Applications can create sockets, which allow them to attach to a port. When an application has created a socket and bind it to a port, data destined to that port will be delivered to the application. If there is no application listening on that port, the packet is discarded and an error may be returned to the sender.

Most commonly used socket function calls are:

`socket()`: Socket is a data structure used by the socket API. When the user calls this function, it creates a socket and returns reference a number for the socket. That reference number will be used in future calls.

`bind()`: This call allows a user to associate a socket with a particular local port and IP address. Typically this call is used on server side. When called in server, it allows the user to specify which port and IP address incoming connections must be addressed to. For outgoing connection requests, it allows the user to specify which port the connection will come from when viewed by the other host.

`listen()`: This function prepares the given socket to accept incoming TCP requests. It must be called before `accept()`.

`accept()`: This function detects incoming connection requests on the listening socket. This call will cause a task to wait until a connection request is received.

`connect()`: When a user issues a connect command, the stack creates a connection with another host. Before connect can instruct the stack to establish a connection, the user must pass a socket and a `sockaddr_in` structure containing the destination IP address and port. In TCP, the handshaking packets will be exchanged. However, in UDP no packets are exchanged.

`send()`: This call allows a user to send data over a connected socket. Because the socket is already connected, it is not necessary to specify the destination address (the destination address was set in `accept()` or `connect()`). The `send()` can be used for either UDP or TCP data.

`sendto()`: Unlike `send()`, `sendto()` requires users to specify the destination port and address. This is useful for UDP communications only, as TCP requires a pre-existing connection. The `sendto()` may be used on either connected or unconnected UDP sockets. In case that a UDP socket is already connected,

the destination address provided to `sendto()` will override the default established on the socket with `connect()`.

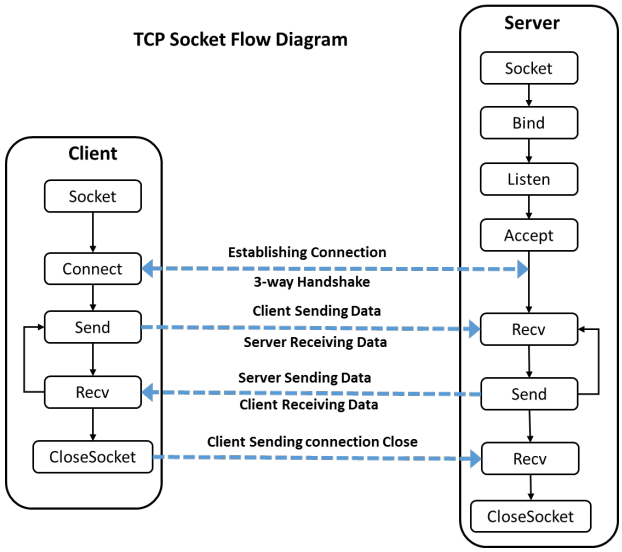
`recv()`: This function allows the user to receive data on the connected socket. The `recv()` can be used for either TCP or UDP.

`recvfrom()`: This function allows the user to receive data from a specified UDP socket (whether or not it is connected). It may not be used for TCP sockets, as they require a connection.

`close()`: This function closes (deletes) a socket that has been allocated with the socket call. If the socket is connected, it closes the connection before deleting it.

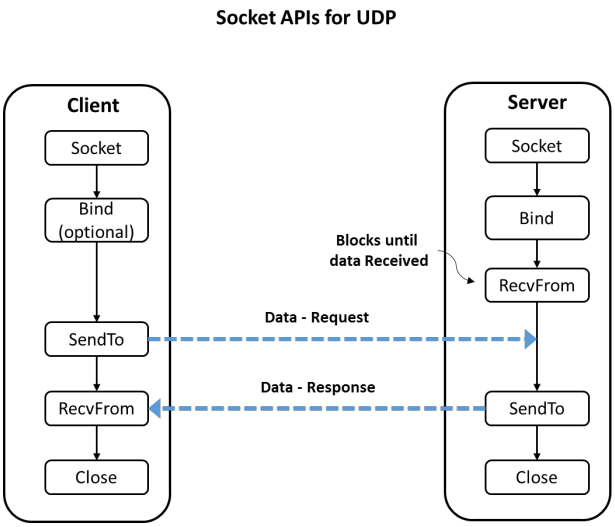
The following figure illustrates the example of server-client relationship of the socket APIs for connection oriented protocol (TCP).

Figure 2-7. Socket based TCP Server-Client



The following figure illustrates the example of server-client relationship of the socket APIs for a connectionless protocol (UDP).

Figure 2-8. Socket based UDP Server-Client



2.5. Messages on Network Interface

Previous section discussed the flow of a TCP/IP application based on socket APIs.

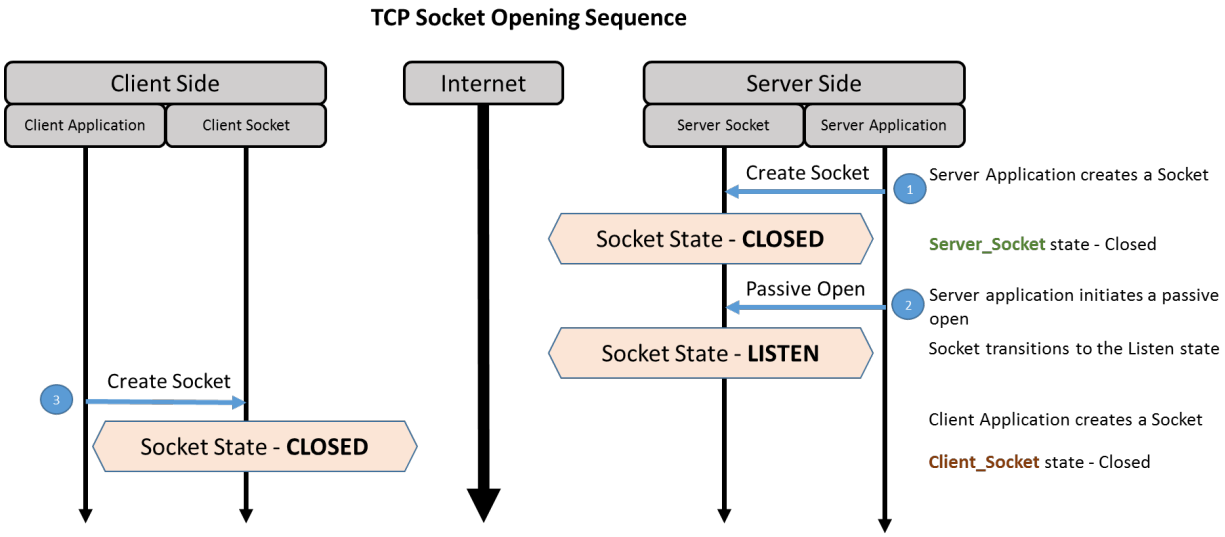
This section provide details of the messages/signals passed between two network nodes during socket function calls and the state transition of sockets on client side and server side.

In the following sequence diagrams, client and server implementation is represented as a combination of application and socket layers. The application layer implements the application protocol for network communication. The socket layer handles the signaling and message transfer of TCP/IP protocol using socket APIs.

2.5.1. TCP/IP Socket Opening Sequence

The starting point of socket based network application is opening socket handlers on client and server side.

Figure 2-9. Socket Opening

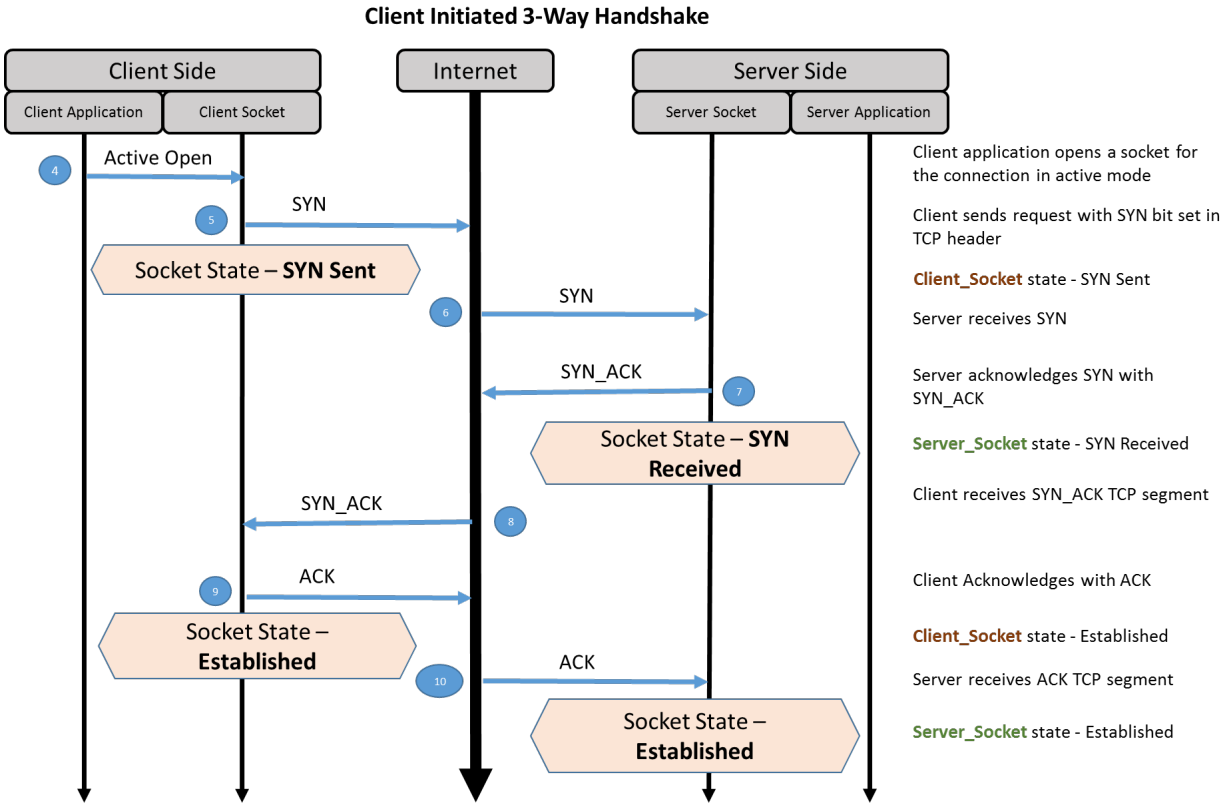


The initial state of socket after creation is ‘Closed’ state. Then at server side, the socket will bind to a port and an IP address. After binding the socket, the socket state is transitioned to ‘Listen’.

2.5.2. TCP/IP Socket Connection Sequence

The client side socket will send out active open signal when socket *connect()* function is called. This will trigger transmission of SYN signal from client to server through physical network connection.

Figure 2-10. Socket - Establishing Connection



The SYN will be received and acknowledged by server. Then server will send its SYN signal, together with acknowledgment to the client.

On receipt of acknowledgment from Server, the socket at client side will transition to ‘Established’ state. The client will acknowledge the SYN message from server. On receipt of acknowledgment from Client, the socket at server side will transition to ‘Established’ state.

2.5.3. TCP/IP Socket Data Transfer Sequence

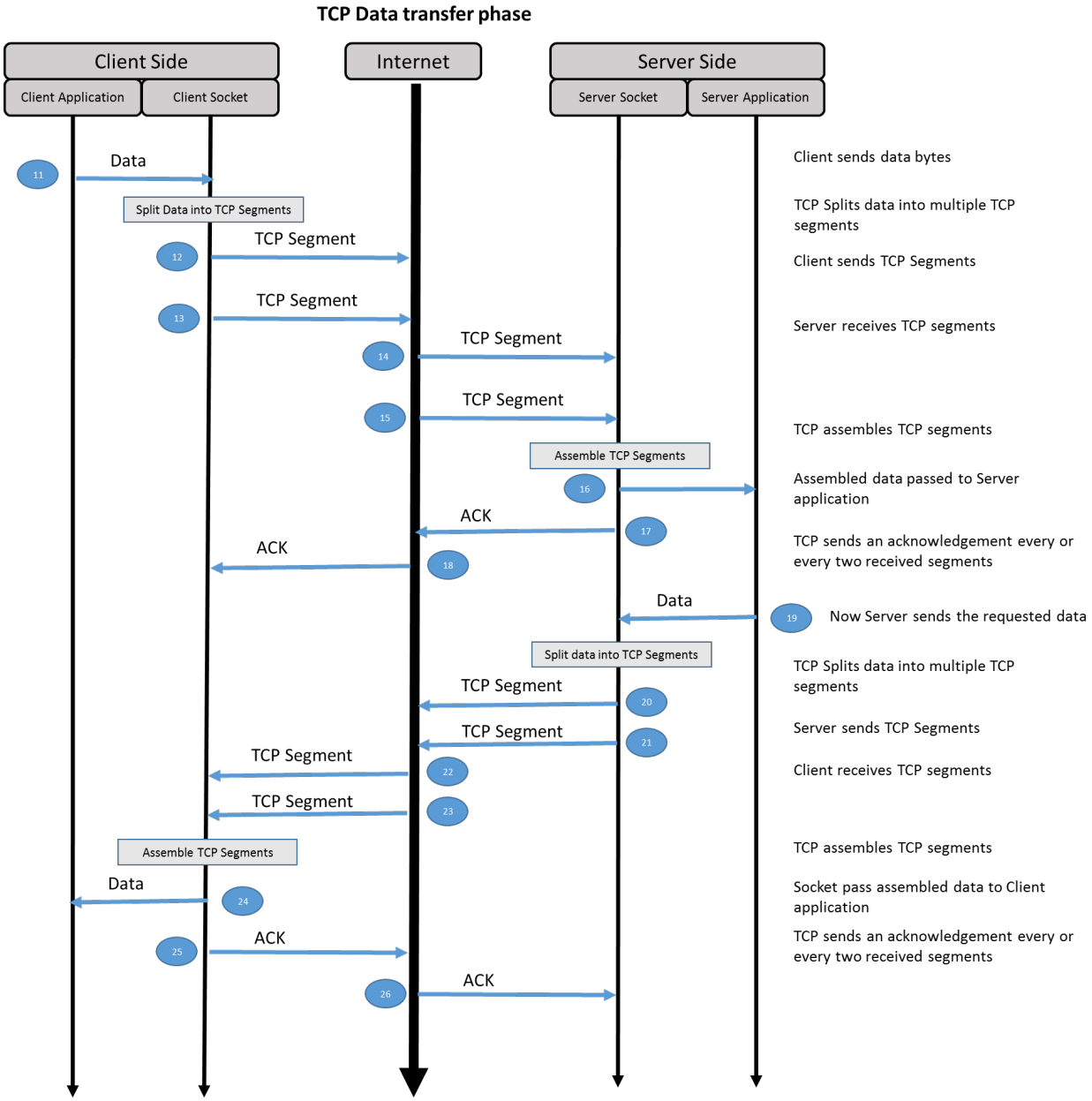
When the connection is established, the data can be transferred between server and client. See the following diagram for the typical data transfer between a server and client.

In this example, the client application send a data packet to server. If the size of packet is higher than the maximum size limit of TCP segment, the TCP layer at client will split packet into multiple TCP segments. At server side on receipt of TCP segments, the server socket acknowledges the receipt of data from client. In this example, server is acknowledging after receipt of two TCP segments. The number of data bytes to be received before the acknowledgment is configurable (refer to Delayed Ack feature in TCP). The received TCP segments will be reassembled and passed to server application.

Similarly server can also send data to client and confirm the transfer by checking acknowledgment from client.

In any of these cases, if the data transfer is not acknowledged, the message will be re transmitted.

Figure 2-11. Socket - Transferring Data



2.5.4. TCP/IP Socket Closing Sequence

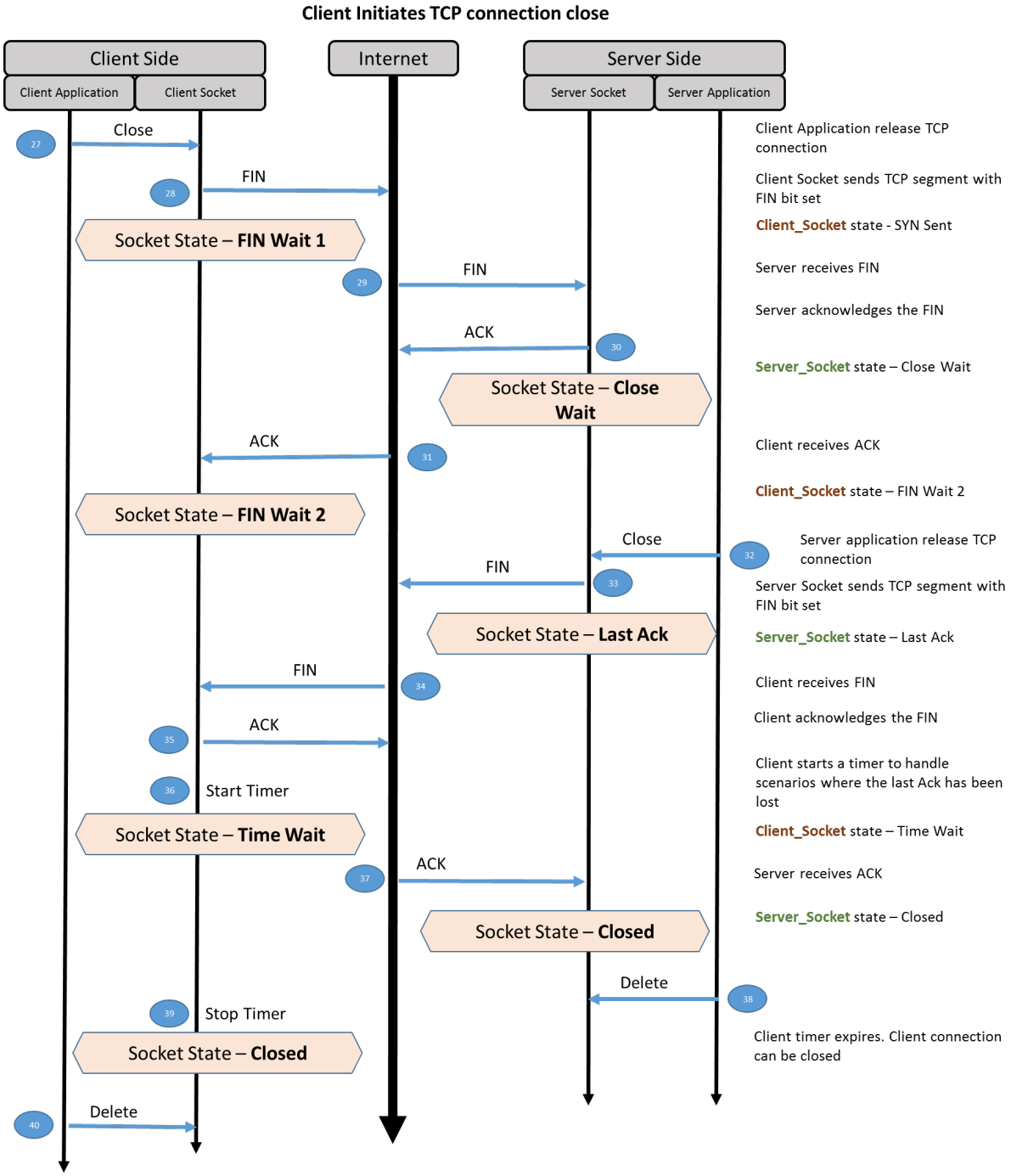
When the data transfer between server and client is complete, the communication sockets will be closed.

On calling socket *close()* function by client application, client will send FIN message to server. On receipt of FIN message by server socket, it acknowledges and transition the server socket to ‘close wait’ state.

When client receives the acknowledgment, it changes the state of client socket to ‘FIN Wait 2’.

When server completes the data transfer, it calls socket *close()*. Then FIN message will be send to client. On receipt of FIN message by client socket, it acknowledges and transition the client socket to ‘Time wait’ state. A timer will be started at client side. When timer expires, the client socket state will change to ‘Closed’ and releases the resources for client socket. When server receive the acknowledgment, it changes the state of client to ‘Closed’ and releases the resources used by server socket.

Figure 2-12. Socket - Closing Connection



2.6. Summary of the section

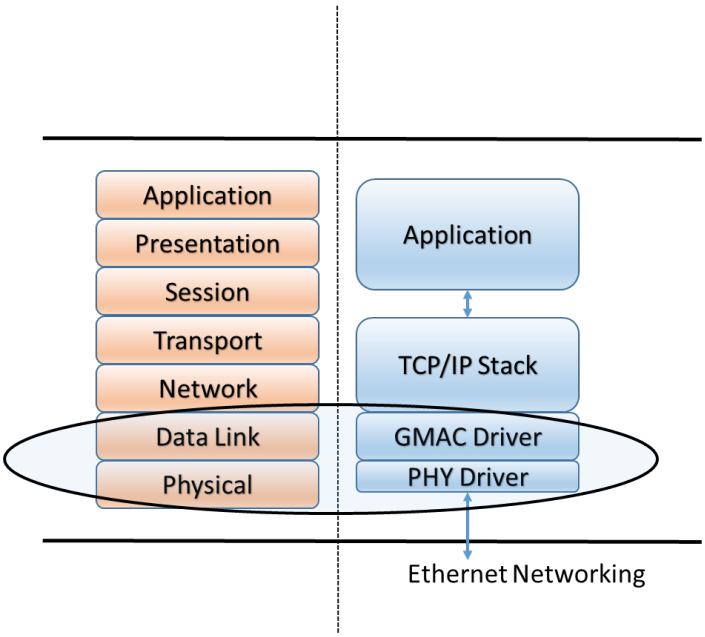
This chapter gives an overview of layered architecture of TCP/IP protocol stack. It also discuss the socket APIs and different messages transferred between network nodes.

The basic concepts of TCP/IP protocol helps in understanding how and where the Ethernet MAC peripheral on SAM E70/ SAM V71 devices fit in a network application.

3. Ethernet Peripheral Overview

Ethernet combines data link and physical layer defined by the IEEE 802.3 specification. The network access layer functions as per the TCP/IP protocol model is implemented with Ethernet MAC (GMAC) peripheral and PHY device as shown in the following diagram.

Figure 3-1. Ethernet - Datalink layer and Physical layer



Ethernet is defined by the supported maximum bit rate, mode of transmission and physical transmission medium. The following are the commonly available options.

- Supported Bit Rate (Mbits/s): 10, 100, 1000, etc.
- Transmission Medium: Coax, Fiber, UTP, etc.
- Transmission mode: Broadband, Base-band.

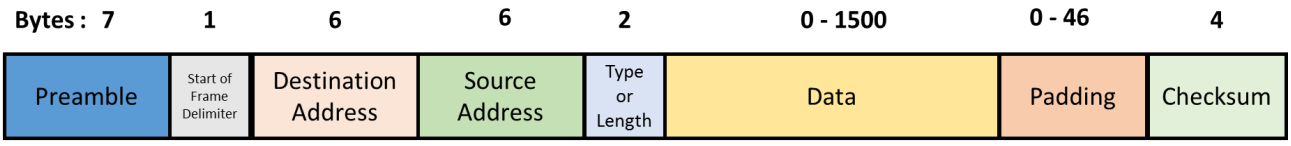
Ethernet supports Carrier Sense Multiple Access with Collision Detect (CSMA/CD) protocol and a payload size of 46 bytes to 1500 bytes.

The data link layer takes care of transmission of frames (blocks) in an error-free manner, including frame sequencing, frame flow control, etc. The physical layer does the transmission of bit streams across physical connections, including encoding, multiplexing, synchronization, clock recovery, serialization, etc.

3.1. Ethernet Frame

Ethernet frames consist of 3 portions: header (preamble to type), payload (data), and trailer (padding and checksum).

Figure 3-2. IEEE 802.3 Ethernet Frame



• Preamble—7 byte sequence (0x55) used for synchronizing receiver to transmitter. (10Mb/s Ethernet).

• Start Frame Delimiter—8-bit sequence (10101011).

• Destination Address— 6 byte Destination MAC address.

• Source Address—6 byte Source MAC address.

• Length/Type: If the value in this 2-byte field is less than or equal to 1500, this represents the payload size. If the value is greater than or equal to 1536, it represents the Ether Type (payload type).

The following are the most common EtherType values:

. IPv4 = 0x0800

. IPv6 = 0x86DD

. ARP = 0x0806

. RARP = 0x8035

• Data—Payload contained in a field between 46 bytes to just over 1500 bytes in length.

• Pad—0x00 data bytes added to the data field if there are fewer than 46 bytes of data in that field.

• Frame Check Sequence (FCS)—Detects transmission errors and provides quality of service at receiving end.

The datalink layer protocol works with the above frame format. The digital data from Ethernet MAC will be transmitted to external network after converting as electrical signals. Usually the physical layer in ethernet will be implemented on PHY devices. The standard interfaces between Ethernet MAC peripheral and PHY devices are MII/RMII for data transfer and MDIO interface for configuration of PHY devices.

3.2. Ethernet MAC (GMAC)

The Ethernet MAC peripheral in SAM E70/SAM V71 devices handle the Data Link layer functions in OSI model. It is a 10/100 Mbits/s Ethernet MAC, compatible with the IEEE 802.3 standard. It has an address checker, statistics and control registers, receive and transmit blocks, and a DMA interface.

The address checker recognizes four specific 48-bit addresses and contains a 64-bit hash register for matching multicast and unicast addresses. It can recognize the broadcast address of all ones, copy all frames, and act on an external address match signal.

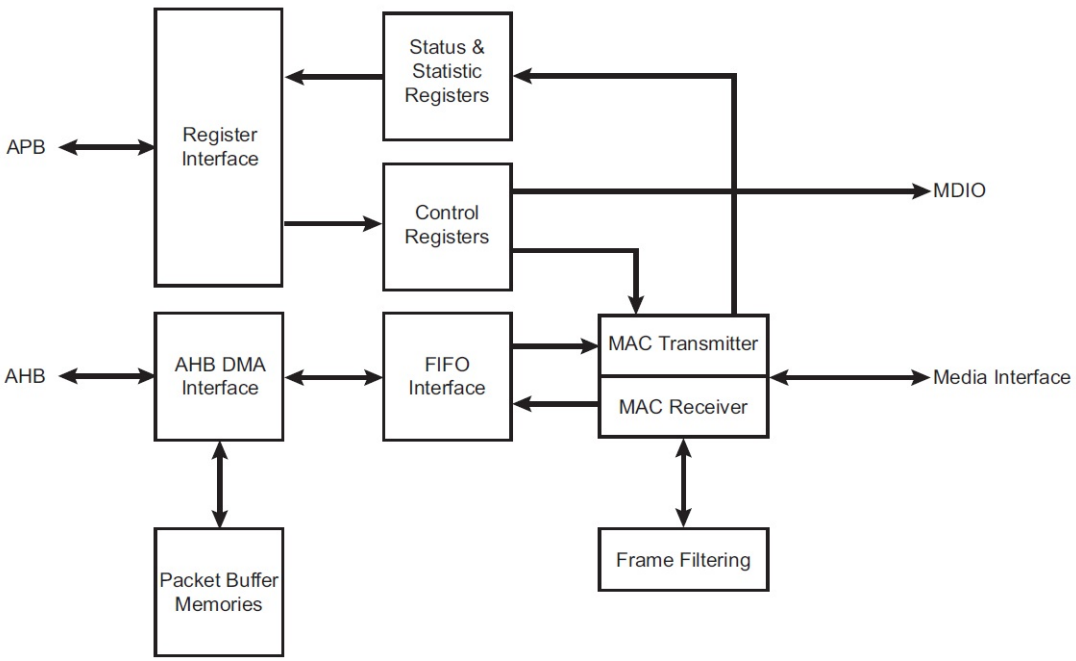
3.2.1. GMAC Features

- Compatible with IEEE Standard 802.3
- 10, 100 Mbps Operation
- Full and Half Duplex Operation at all Supported Speeds of Operation
- Statistics Counter Registers for RMON/MIB
- MII/RMII Interface to the Physical Layer
- Integrated Physical Coding
- Direct Memory Access (DMA) Interface to External Memory
- Support for 3 Priority Queues in DMA
- 8 Kbytes Transmit RAM (2 KB for Queue 0, 2 KB for Queue 1, 4 KB for Queue 2) and 4 Kbytes Receive RAM
- Programmable Burst Length and Endianism for DMA
- Interrupt Generation to Signal Receive and Transmit Completion, Errors or Other Events
- Automatic Pad and Cyclic Redundancy Check (CRC) Generation on Transmitted Frames

- Automatic Discard of Frames Received with Errors
- Receive and Transmit IP, TCP and UDP Checksum Offload. Both IPv4 and IPv6 Packet Types Supported
- Address Checking Logic for Four Specific 48-bit Addresses, Four Type IDs, Promiscuous Mode, Hash Matching of Unicast and Multicast Destination Addresses and Wake-on-LAN
- Management Data Input/Output (MDIO) Interface for Physical Layer Management
- Support for Jumbo Frames up to 10240 Bytes
- Full Duplex Flow Control with Recognition of Incoming Pause Frames and Hardware Generation of Transmitted Pause Frames
- Half Duplex Flow Control by Forcing Collisions on Incoming Frames
- Support for 802.1Q VLAN Tagging with Recognition of Incoming VLAN and Priority Tagged Frames
- Support for 802.1Qbb Priority-based Flow Control
- Programmable Inter Packet Gap (IPG) Stretch
- Recognition of IEEE 1588 PTP Frames
- IEEE 1588 Time Stamp Unit (TSU)
- Support for 802.1AS Timing and Synchronization
- Supports 802.1Qav Traffic Shaping on Two Highest Priority Queues

The block diagram of GMAC peripheral in SAM E70/ SAM V71 is as follows.

Figure 3-3. GMAC Block Diagram



The GMAC includes the following blocks and interfaces:

- MII, RMII to an external PHY for data transfer
- MDIO interface for external PHY management
- MAC Transmitter block
- MAC Receiver block
- FIFO
- Packet buffer

- AHB DMA interface
- Mac frame filtering

3.2.2. GMAC Memory

3.2.2.1. FIFO

The MAC transmitter receives data for transmission from a small Transmit FIFO. The transmit FIFO is filled from transmit packet buffer.

Similarly, there is a receive FIFO. The receive FIFO is filled with the received frame by MAC receive block. The received frame will be moved from FIFO to receive packet buffer in case address match.

3.2.2.2. Packet Buffer

The Ethernet MAC does packet buffering in a dedicated internal dual-port SRAM memory.

The transmitter block fetch packets for transmission from system memory through AHB interface and store in packet buffer. The MAC transmitter will fetch frame data from packet buffer through FIFO interface.

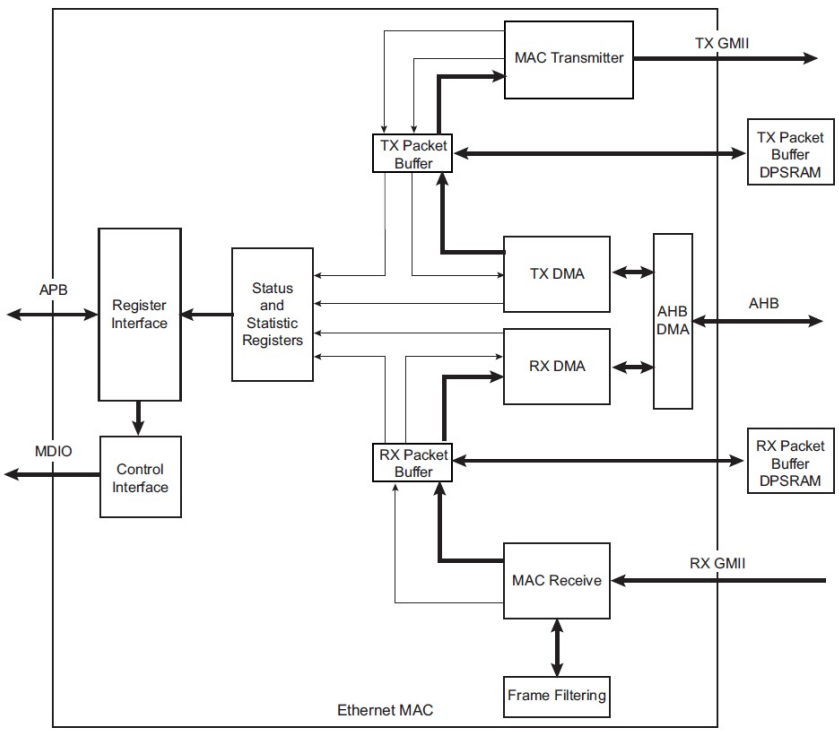
Similarly the receive packet buffer stores frames from the MAC receiver, through receive FIFO, along with their status and statistics.

The packet buffer support two programmable modes; 1) Full store and forward 2) Partial store and forward.

In case of transmit side, for full store and forward mode, the entire transmit packet will be stored in packet buffer before MAC transmitter starts transmitting. In partial store and forward mode, the mac transmitter will start transmitting when programmable number of packet data is available in packet buffer.

Receive side also support full store and forward mode and partial store and forward mode.

Figure 3-4. GMAC - Memory Interface



FIFO interface is not shown in the above block diagram.

On the receive side, there is 4 Kbytes Dual Port SRAM for packet buffer. So, a full packet (say 1500 bytes) can be sent by DMA from packet buffer to AHB memory whilst another is being received.

On transmit side, different queues have different sizes of DP-SRAM. i.e. Queue 2 has 2Kbytes, Queue 1 and Queue 0 have 1Kbyte each.

3.2.2.3. AHB Memory Buffers

The AHB memory buffers are part of data memory of the device.The start location for each receive AHB buffer is stored in memory in a list of receive buffer descriptors at an address location pointed to by the receive buffer queue pointer. Received frames are written to receive buffers in memory interfaced via AHB interface. The receive buffer depth is programmable in the range of 64 bytes to 16 Kbytes.

Frames to transmit are stored in one or more transmit AHB buffers. Transmit frames can be between 1 and 16384 bytes long. The maximum number of buffers permitted for each transmit frame is 128. The start location for each transmit AHB buffer is stored in memory in a list of transmit buffer descriptors at allocation pointed to by the transmit buffer queue pointer.

3.2.3. GMAC - Transmit Block

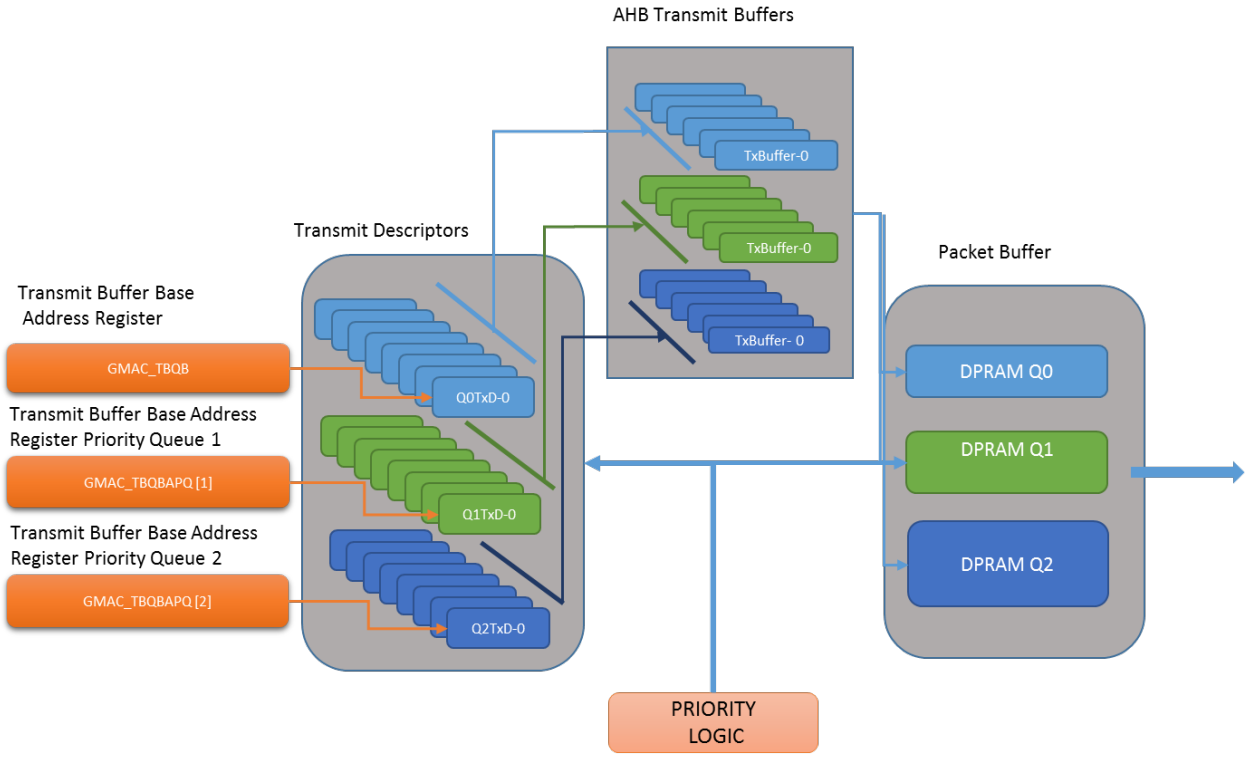
The MAC transmitter can operate in either half duplex or full duplex mode and transmits frames in accordance with the Ethernet IEEE 802.3 standard. In half duplex mode, the CSMA/CD protocol of the IEEE 802.3 specification is followed.

Transmitter block receives data through the FIFO interface. Frame assembly starts by adding preamble and the start frame delimiter. Data is taken from the transmit FIFO interface a word at a time. If necessary, padding is added to take the frame length to 60 bytes. CRC is calculated using an order 32-bit polynomial. This is inverted and appended to the end of the frame taking the frame length to a minimum of 64 bytes.

Transmit data will be send to PHY using the MII/RMII interface.

The following diagram shows the memory interface on transmit side of GMAC

Figure 3-5. GMAC - Transmit Memory Interface



The data frames to be transmitted is stored in one or multiple AHB Transmit buffers. These transmit buffers are listed in a data structure called Transmit descriptor. The Transmit Buffer Queue Pointer register points to start address of this descriptor. The above mechanism is available for all the 3 priority queues.

When the data to be transmitted is written to the buffer and the descriptors are updated, the transmit circuits can then be enabled by writing to the Network Control register.

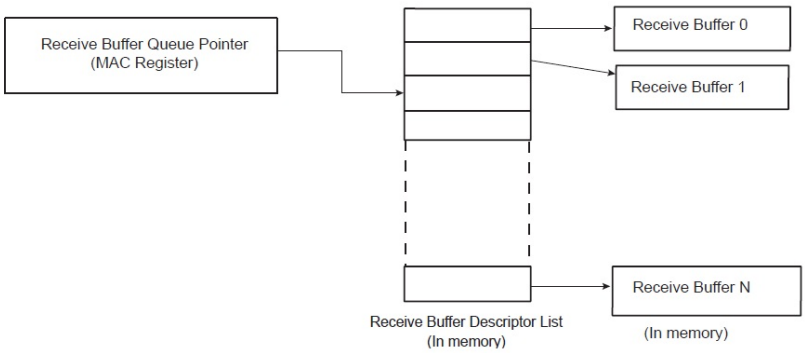
During transmission the transmit frames stored in Transmit buffers will be transferred to packet buffers by DMA and then to transmitter block via FIFO.

3.2.4. GMAC - Receive Block

The MAC receive block checks for valid preamble, FCS, alignment and length. It presents received frames to the FIFO interface and stores the frame destination address for use by the address checking block.

At end of frame reception the receive block give signal internally to DMA block whether the frame is good or bad. The DMA block will recover the current receive buffer if the frame was bad.

Figure 3-6. GMAC - Receive memory interface



Receive data is written to areas of Receive buffers in system memory. Receive buffers are listed in a data structure called receive descriptor. The Receive Buffer Queue Pointer register points to the start address of receive descriptor.

Refer the 'Programming Interface' in GMAC section of device datasheet for the initialization of GMAC peripheral. After the receive buffers and descriptors are updated, the receive circuits can be enabled by writing to the address recognition registers and the Network Control register.

3.2.5. GMAC- Frame Filtering

During reception, the filter block determines which frames should be written to the FIFO interface and on to the DMA.

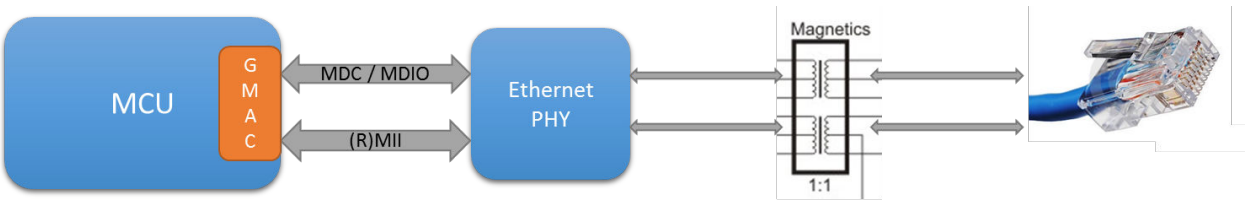
The destination address of received frames is compared against the data stored in the Specific Address registers when they have been activated. The addresses are deactivated at reset or when their corresponding Specific Address register Bottom is written. They are activated when Specific Address register Top is written.

If a receive frame address matches an active address, the frame is written to the FIFO interface and on to DMA memory.

3.2.6. GMAC- PHY interfacing

For the physical layer, external PHY devices are connected to GMAC peripheral.

Figure 3-7. GMAC - PHY Interfacing



GMAC has following interfaces:

- MII, RMII to an external PHY - The interface between the PHY and the MAC for transferring Ethernet data is Media Independent Interface (MII) or Reduced Media Independent Interface (RMII).
- MDIO interface for external PHY management - A management interface used to read and write PHY register called MDIO.

3.2.7. GMAC - Other functions

The other main functions supported by Ethernet MAC peripheral on SAM E70/SAM V71 devices are,

- Hardware offload features

- Receive and transmit IP, TCP and UDP checksum offload.
- Automatic pad and cyclic redundancy check (CRC) generation on transmitted frames.
- Time-stamping of packets.

- Support for higher layer applications
  - IEEE 1588 features.
  - Routing of data to separate queues.
  - Traffic shaping.

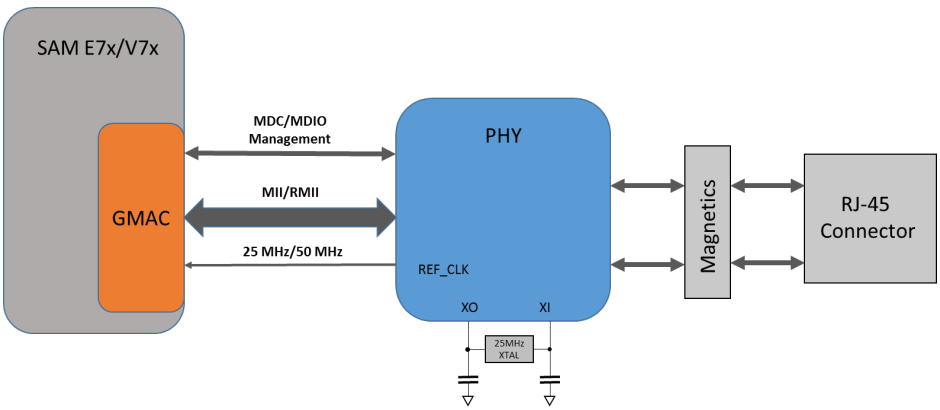
Refer to the GMAC section of device datasheet for more details.

3.3. Ethernet PHY

PHY is an abbreviation for the physical layer of the OSI model and refers to the circuitry required to implement physical layer functions.

The PHY connects a link layer device (often called MAC as an abbreviation for Media Access Control) to a physical medium such as an optical fiber or copper cable.

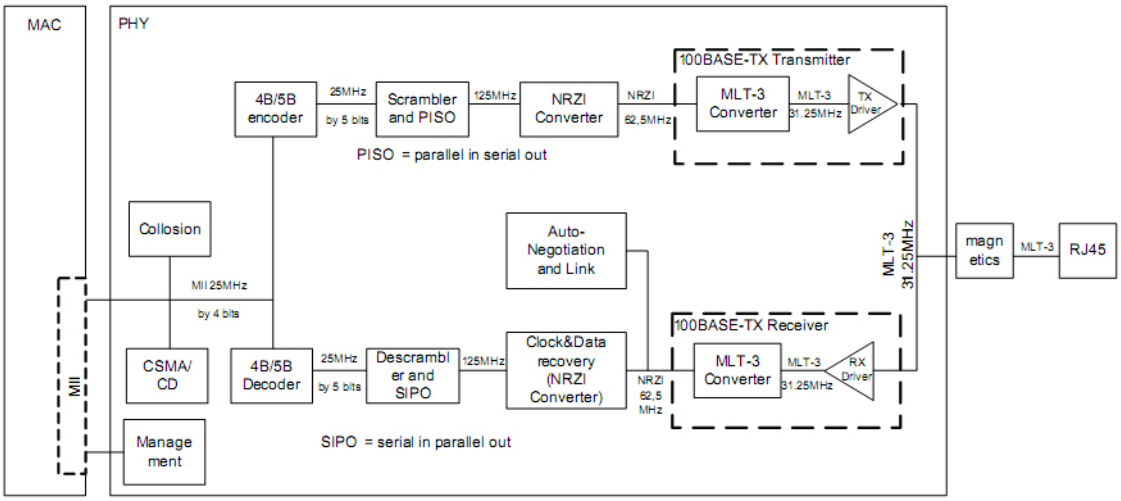
Figure 3-8. Ethernet PHY



The Ethernet PHY transmit and receive Ethernet frames in electrical domain. It converts the digital transmit data from link-layer packet protocol to analog domain by modulating electrical signals according to the transmit data. On receive side, PHY demodulates the electrical signals and present the digital receive data to higher layers.

A generic block representation of a 100Mbits/s PHY is as shown below.

Figure 3-9. PHY Block Diagram



PHY supports half duplex and full duplex communication. In half duplex communication, multiple simultaneous transmissions in a shared communication medium happens. It results in interference, which requires resolution by the CSMA/CD access control protocol.

- CS = Carrier Sense
- MA = Multiple Access
- CD = Collision Detection

Carrier sense controls the transmission line if there is traffic. If CS detects the transmission line free, data transfer is started.

The Ethernet network is designed for multiple accesses by different network nodes. When collision is detected, the command to restart the transmission will be given after waiting for a random time.

On the otherside, Full Duplex operation allows simultaneous communication between a pair of stations. In this case, physical layer is capable of supporting simultaneous transmission and reception without interference. So no need for CSMA/CD on full duplex operation.

The standard connection between the Ethernet MAC and PHY is Media Independent Interface (MII).

The encode and decode mechanism varies with type of network. For 10Mbit/s networks, the PHY works with the Manchester encode and decode mechanism. In 100-Mbit/s networks, the functions are 4B/5B encode and decode, scrambling, non-return-to zero/inverted (NRZI) coding, and MLT-3 conversion.

The relation between standards and coding mechanisms are as shown.

10Mbit/s = Manchester coding

100Mbit/s = 4B/5B coding

Another feature supported by PHY is Auto-Negotiation.

The Auto-Negotiation function automatically configures the PHY to optimal link parameters based on the capabilities of its link partners. The link partners can use Auto-Negotiation to figure out the highest speed that each link partner support as well as automatically setting full-duplex operation if both ends support that mode.

3.4. Summary of the section

This section provides an overview of datalink layer and physical layer on an Ethernet based network. It also provide the basics of Ethernet MAC peripheral on SAM E70/ SAM V71 devices and externally interfaced PHY.

4. CycloneTCP Stack Overview

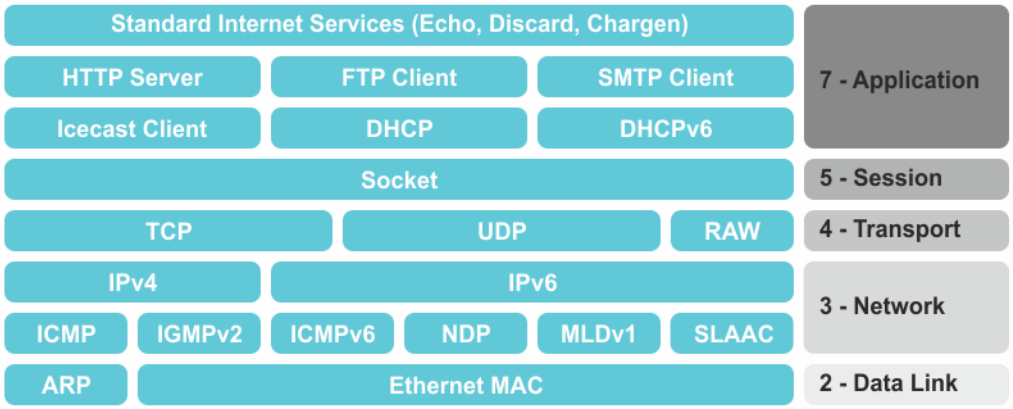
CycloneTCP is a dual IPv4/IPv6 stack dedicated to embedded applications. CycloneTCP conforms to RFC standards and offers seamless interoperability with existing TCP/IP systems. By supporting IPv6, CycloneTCP eases deployment of next-generation Internet. The stack is distributed as a full ANSI C and highly maintainable source code.

4.1. CycloneTCP Features

- Built-in support for multiple network interfaces
- BSD style socket API
- Blocking/non-blocking socket operation and event-driven functions (select and poll)
- Efficient data transfer through zero copy
- Well-crafted TCP module with selective ACK and congestion control
- Raw socket interface
- Multicast support (IGMPv2 and MLDv1)
- IP fragmentation and reassembly support
- Flexible memory footprint
- High throughput
- Dialog-based configuration wizard
- Portable architecture (no processor dependencies)
- Straightforward port to any RTOS
- Debugging and trace functionality to ease development and integration
- Host name resolution (DNS, mDNS and NetBIOS Name Service)
- mDNS and NetBIOS responder
- DNS-based service discovery (DNS-SD)
- Auto-IP (dynamic configuration of IPv4 link-local addresses)
- DHCP client
- SLAAC (IPv6 stateless address autoconfiguration)
- DHCPv6 client and relay agent
- FTP server and client
- Web server with SSI and CGI scripting
- Secured Web server (HTTPS)
- SMTP client
- Network time synchronization (SNTP client)
- SNMPv1 and SNMPv2c agent
- PPP (Point-to-Point Protocol)

The following block diagram provides architecture of CycloneTCP stack.

Figure 4-1. CycloneTCP Block Diagram



The TCP/IP stack is available either as open source (CycloneTCP Open) or under a commercial license (CycloneTCP Lite, Pro or Ultimate) for proprietary developments in a commercial context:

- The open source version CycloneTCP Open is available for free. Developers must freely distribute the complete source code of their application, making it available for end users.
- The commercial version CycloneTCP Lite includes all the core features of the IPv4 stack but does not provide any application protocols. This version targets developers who only need access to sockets.
- The commercial version CycloneTCP Pro is a full-featured IPv4 stack. This version includes application protocols such as FTP, HTTP, SMTP, SNMP and SNTP.
- The commercial version CycloneTCP Ultimate is a full-featured dual stack (IPv4 and/or IPv6). This version includes application protocols such as FTP, HTTP, SMTP SNMP and SNTP.

	CycloneTCP Open	CycloneTCP Lite	CycloneTCP Pro	CycloneTCP Ultimate
License	Open source	Commercial	Commercial	Commercial
Source code	✓	✓	✓	✓
Royalty free	✓	✓	✓	✓
Doxygen documentation	✓	✓	✓	✓
PDF user's manual	✗	✓	✓	✓
Support and maintenance	optional	✓	✓	✓

The features supported by different versions of CycloneTCP stack is as given in the table below.

	CycloneTCP Open	CycloneTCP Lite	CycloneTCP Pro	CycloneTCP Ultimate
ARP	✓	✓	✓	✓
IPv4	✓	✓	✓	✓
ICMP	✓	✓	✓	✓
IGMPv2	✓	✓	✓	✓
IP fragmentation and reassembly	✓	✓	✓	✓
UDP	✓	✓	✓	✓
TCP	✓	✓	✓	✓

BSD sockets	✓	✓	✓	✓
Auto-IP	✓	✓	✓	✓
DHCP client	✓	✓	✓	✓
DNS client	✓	✓	✓	✓
NetBIOS responder	✓	✓	✓	✓
mDNS	✓	✓	✓	✓
DNS-SD	✓	✗	✓	✓
FTP client/server	✓	✗	✓	✓
HTTP server with CGI scripting	✓	✗	✓	✓
SMTP client	✓	✗	✓	✓
SNTP client	✓	✗	✓	✓
SNMPv1 and SNMPv2c agent	✓	✗	✓	✓
Icecast/SHOUTcast client	✓	✗	✓	✓
IPv6	✓	✗	✗	✓
NDP	✓	✗	✗	✓
ICMPv6	✓	✗	✗	✓
MLDv1	✓	✗	✗	✓
SLAAC	✓	✗	✗	✓
DHCPv6 client	✓	✗	✗	✓
DHCPv6 relay agent	✓	✗	✗	✓
PPP (Point-to-Point Protocol)	✓	optional	optional	optional

4.2. Supported Operating Systems

CycloneTCP supports major open source and commercial RTOS:

- FreeRTOS
- ChibiOS/RT
- CMSIS-RTOS
- Keil RTX
- Micrium µC/OS-II
- Micrium µC/OS-III
- Segger embOS
- SYS/BIOS (TI-RTOS)

4.3. Source Files

Download the source files at http://www.oryx-embedded.com/download.html.

You can browse through the source code at http://www.oryx-embedded.com/doc/files.html.

5. Demo Applications with CycloneTCP

In this section, a web-server and web-client is implemented on SAM V71 Xplained Ultra board with CycloneTCP stack and GMAC peripheral. The TCP/IP stack and network application is running on FreeRTOS platform.

The features supported by CycloneTCP stack is provided in section 3. The basic features demonstrated in this example projects are

- IPV4
- TCP
- DHCP Client
- ARP
- HTTP Server with CGI Scripting

5.1. HTTP Web-Server Demo

The web-server demo application handles HTTP requests from web client and responds to the requests. The demo is implemented with multiple FreeRTOS tasks. This server is capable of handling 8 connections simultaneously.

You can download the demo source files from the link given at Source Files section. The web-server demo project for SAM V71 Xplained Ultra is available at `\demo\atmel\samv71_xplained_ultra\web_server_demo\as6`. Open the project (`web_server_demo.atsln`) with Atmel Studio 6.2 for browsing the source code and debugging on hardware. (Currently, the project available ay Oryx repository is generated with Atmel Studio 6.2. This can be ported to newer Atmel Studio versions)

The demo is mostly implemented around main function for initialization, netTask for TCP/IP stack, `httpListenerTask` and `httpConnectionTask`.

The source code consists

1. `main()` function – for core, clock initialization, RTOS initialization and TCP/IP initialization
2. `netTask()` function – rtos task handling TCP and IP processing
3. `httpListenerTask()` function – rtos task that listens to socket
4. `httpConnectionTask()` function – rtos task for sending and receiving TCP/IP messages

The following diagram provides the basic flow of Web-server implementation with code snippets.

Figure 5-1. HTTP Server Demo - `main()`

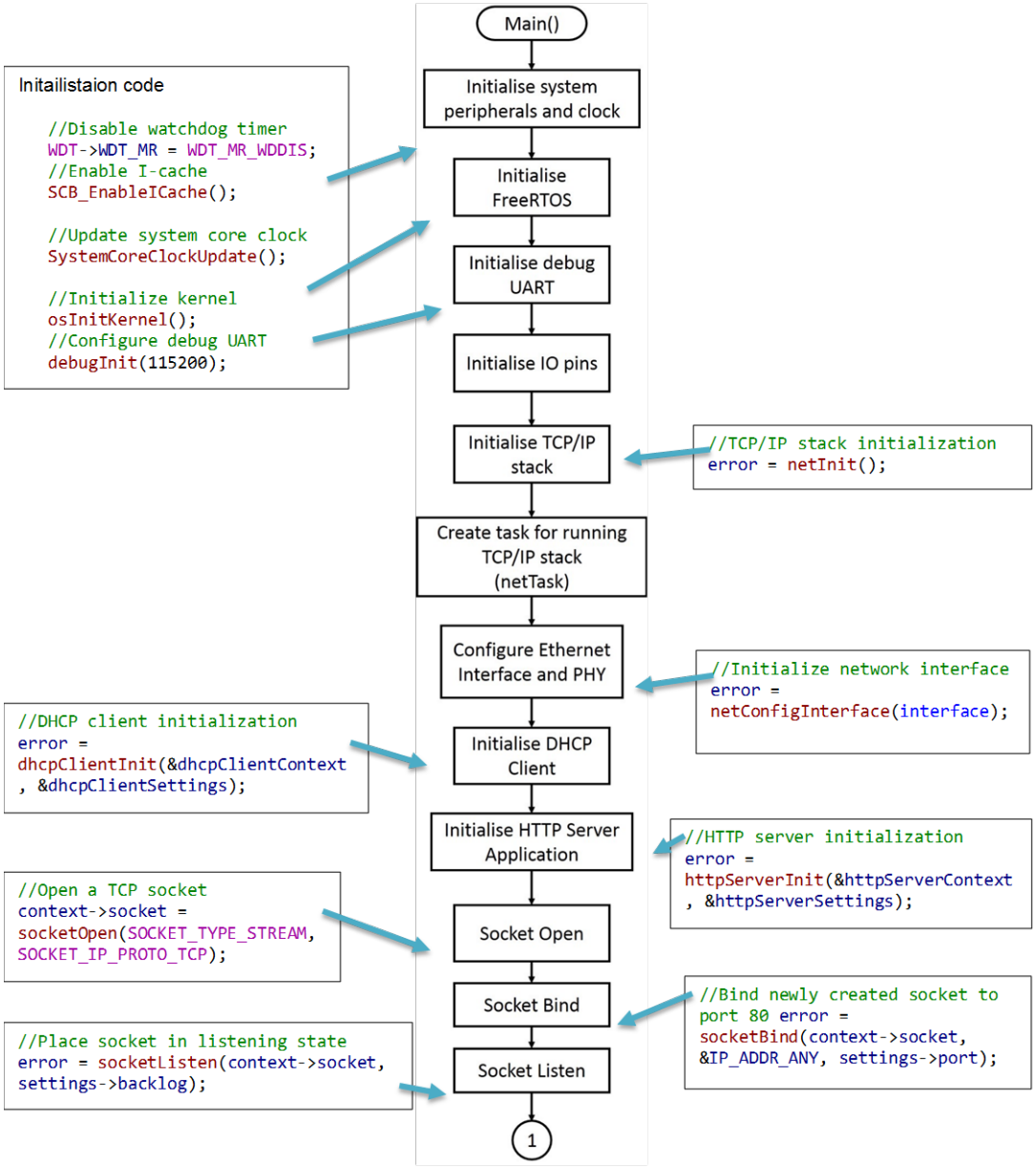


Figure 5-2. HTTP Server Demo - `main()` - continued

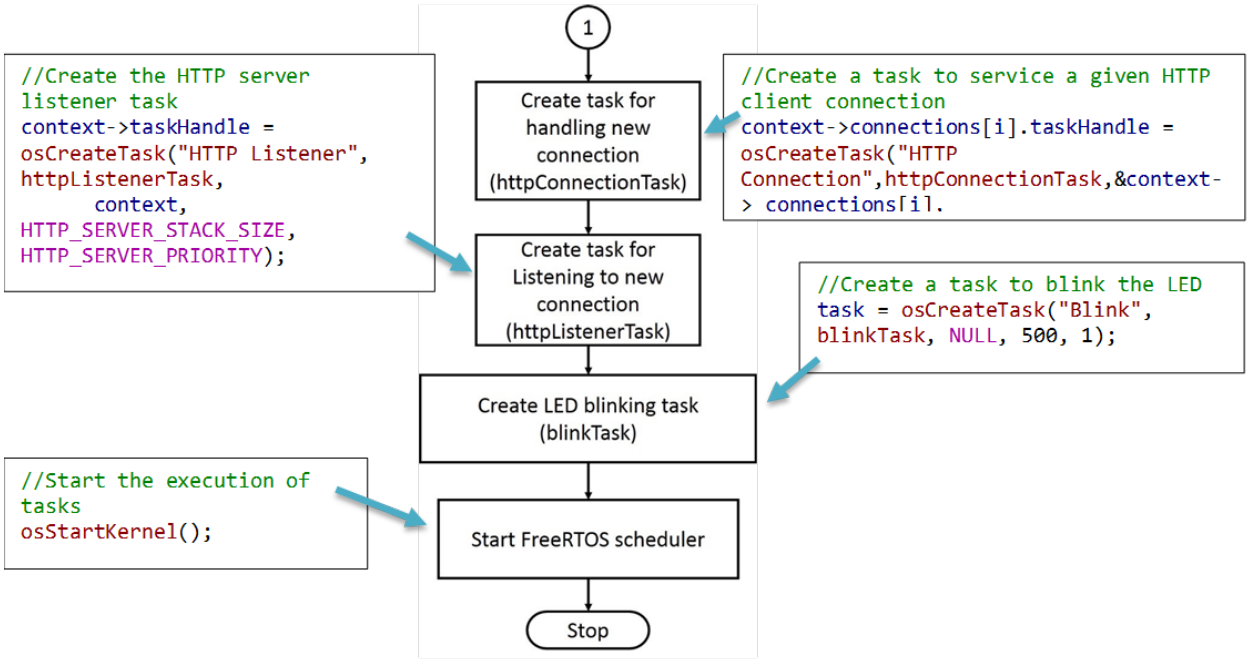


Figure 5-3. HTTP Server Demo - Listener task

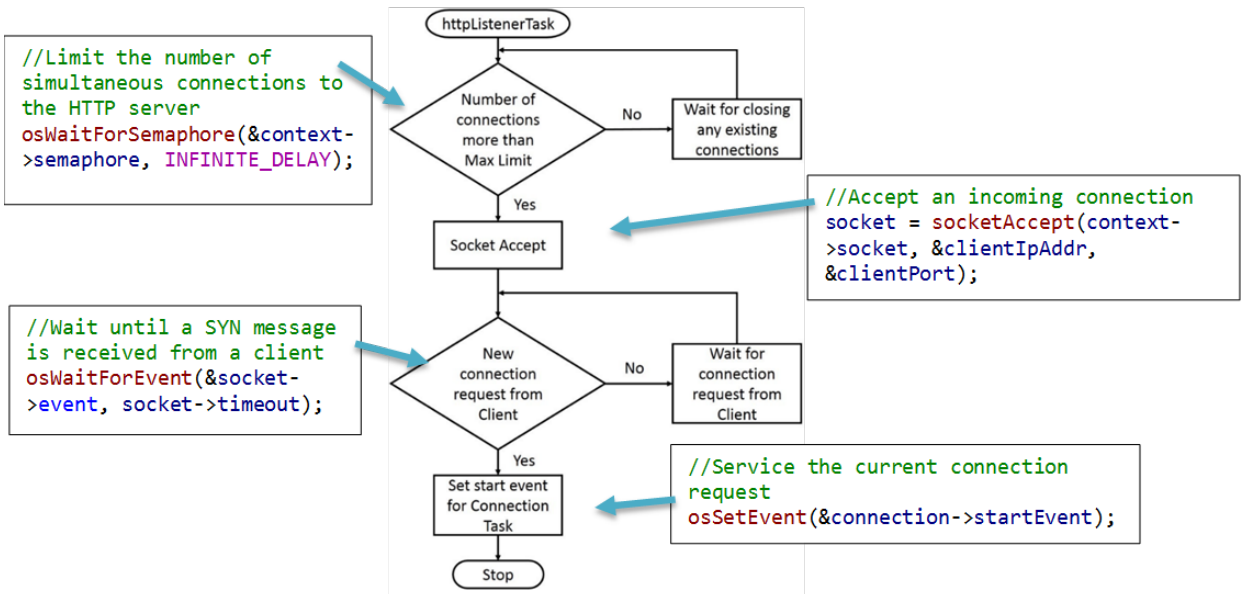


Figure 5-4. HTTP Server Demo - Connection task

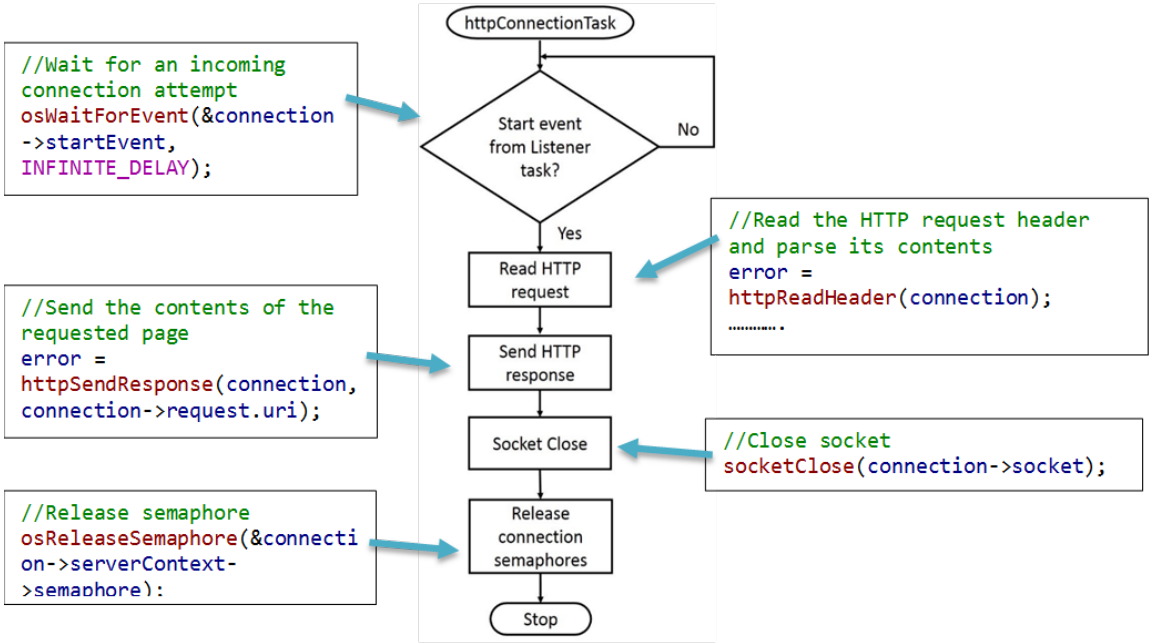
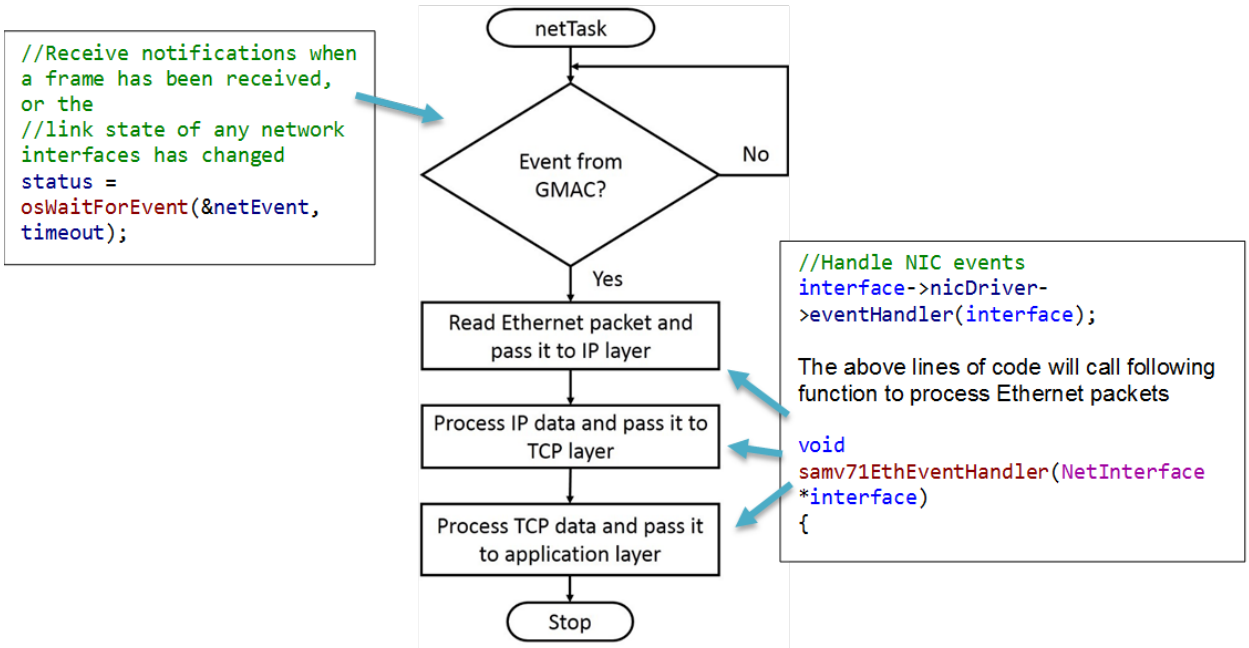


Figure 5-5. HTTP Server Demo - TCP/IP task



5.2. HTTP Client Demo

The HTTP Client demo implements a web-client on SAM V71 Xplained Ultra board. It sends HTTP request on press of SW0 push button on SAM V71 Xplained ultra board. The response received from web server is printed on the debug terminal.

Follow the steps below to open the client demo project.

1. The source project for HTTP client is available as an attachment with this application note as `http_client_demo.zip` file. Download the .zip file and unzip it.
2. Download the CycloneTCP source files from the link given at Source Files section.
3. Copy the http_client_demo folder (from unzipped folder) to the path `\demo\atmel\samv71_xplained_ultra\` of CycloneTCP source files. The web-client demo project for SAM V71 Xplained Ultra is available at `\demo\atmel\samv71_xplained_ultra\http_client_demo\as6`.
4. Open the project (`http_client_demo1.atsln`) with Atmel Studio 6.2 for browsing the source code and debugging on hardware.

The demo is mostly implemented around `main` function for initialization, `netTask` for TCP/IP stack and `userTask` for http client application.

The source code consists

1. `main()` function – for core, clock initialization, RTOS initialization and TCP/IP initialization
2. `netTask()` function – rtos task handling TCP and IP processing
3. `userTask()` function – HTTP client application

The following diagram provides the basic flow of web client implementation with code snippets.

Figure 5-6. HTTP Client Demo - `main()`

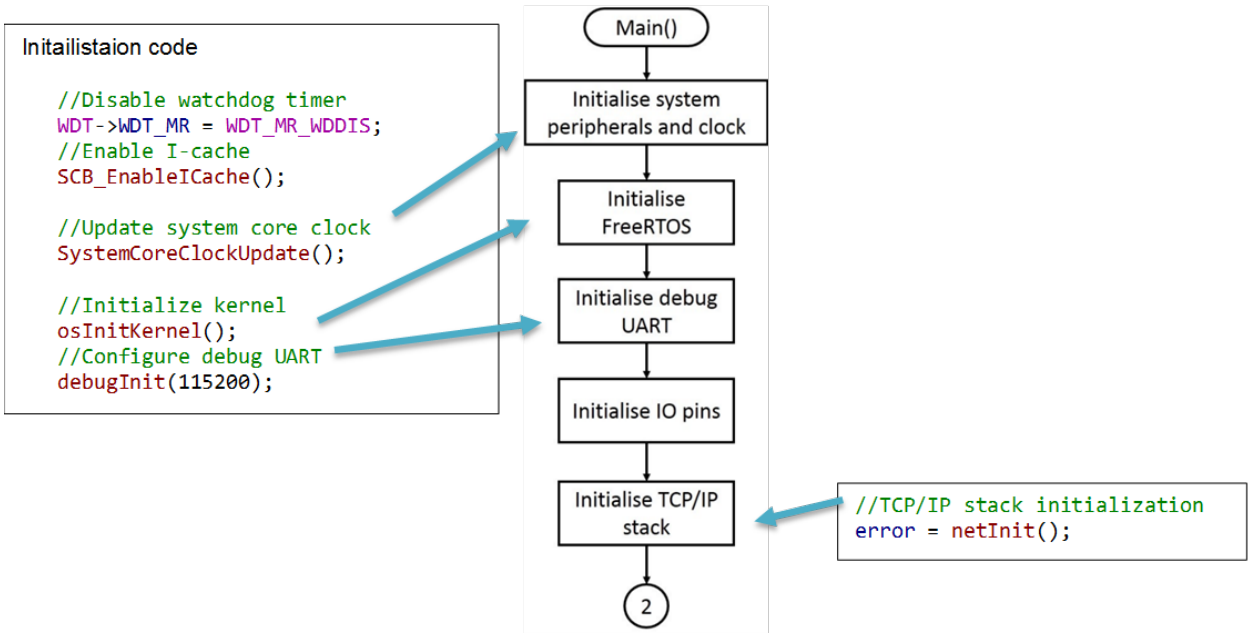


Figure 5-7. HTTP Client Demo - `main()` - continued

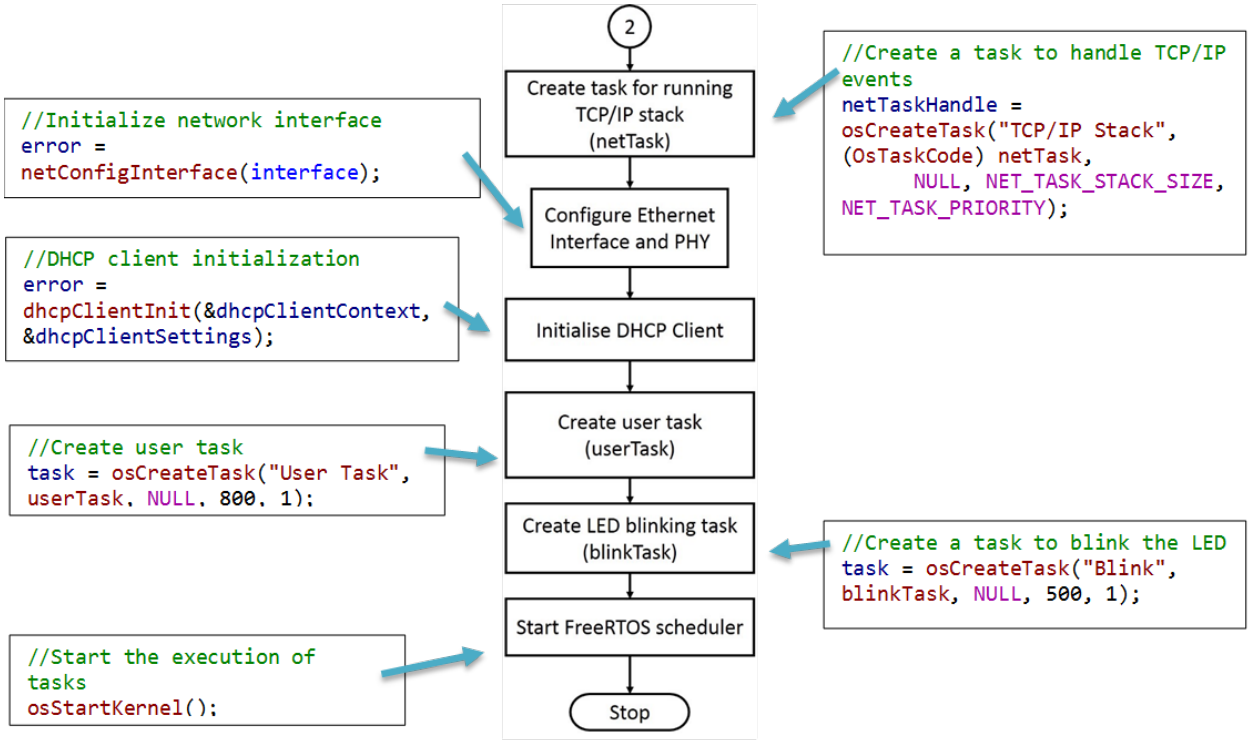


Figure 5-8. HTTP Client Demo - TCP/IP Task

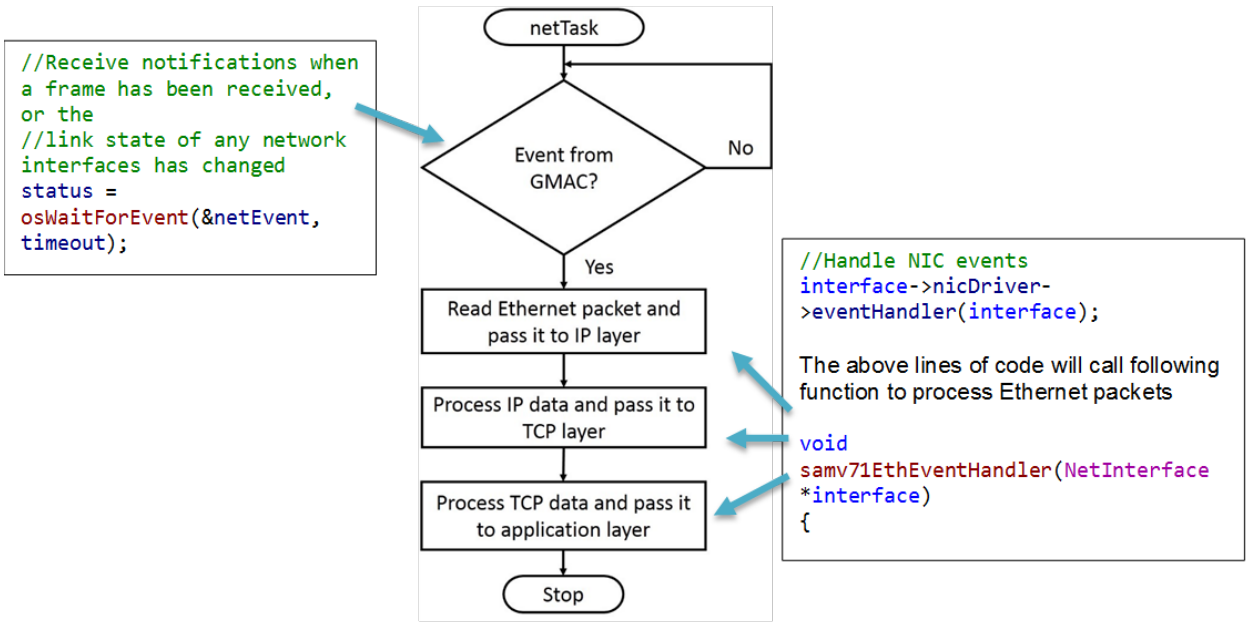
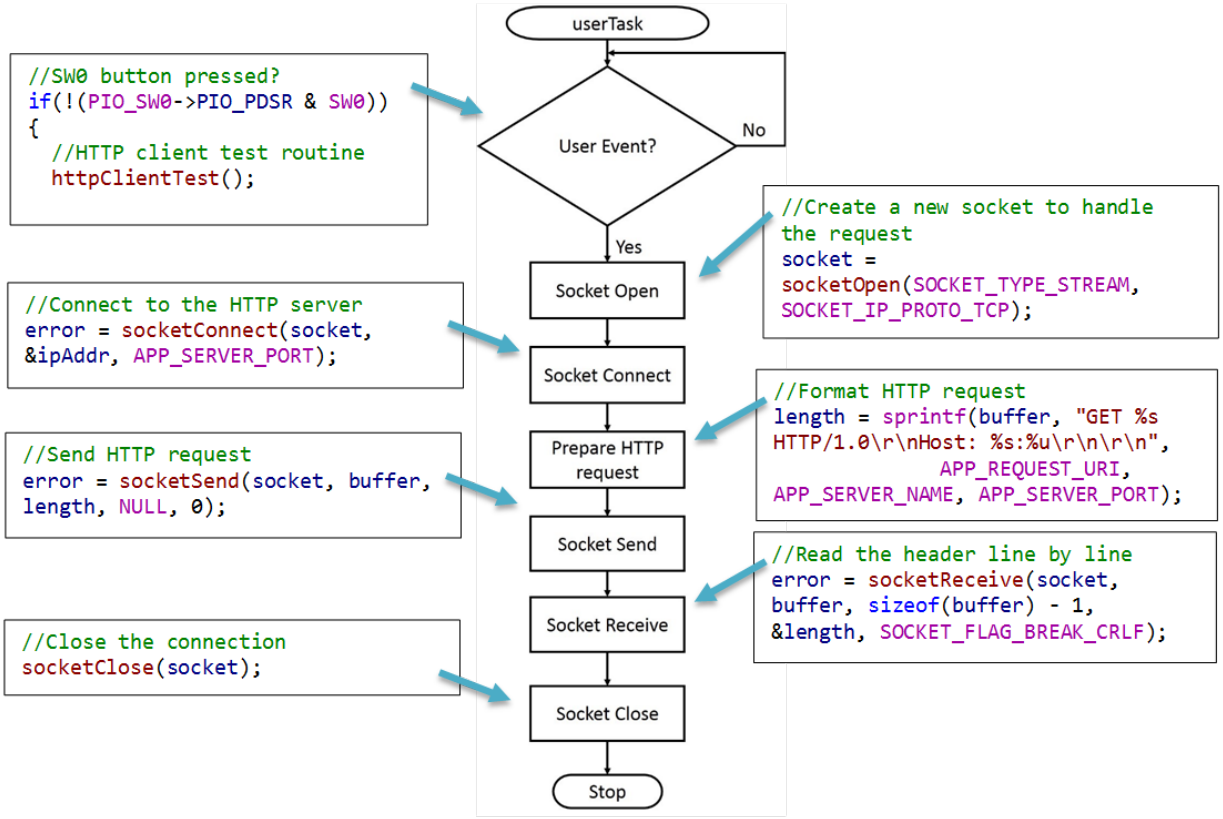


Figure 5-9. HTTP Client Demo - Application Task



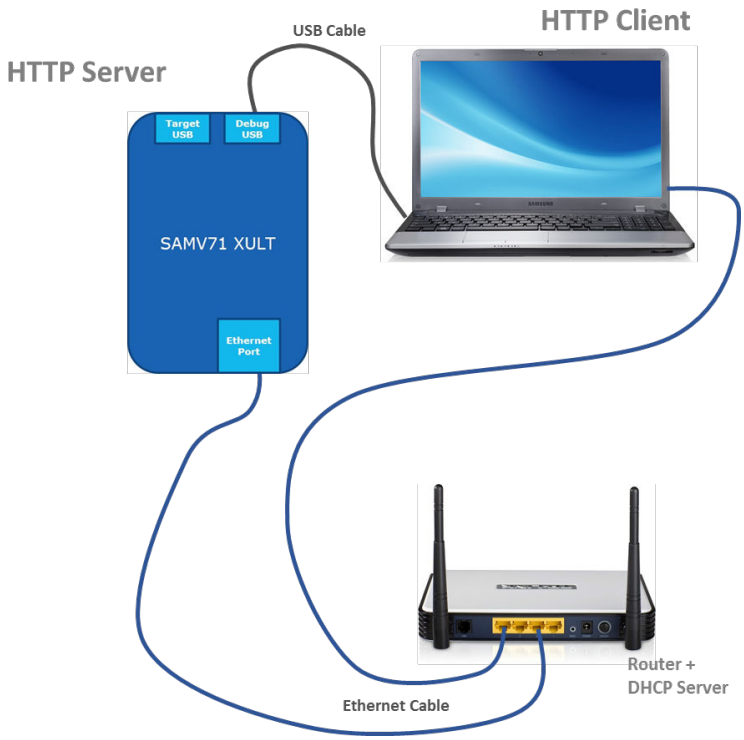
The above flow diagram with relevant code snippets will help users to understand the source code for the demo projects better. The contact information for further queries on CycloneTCP is available at http://www.oryx-embedded.com/index.html#&panel1-5.

5.3. Demo Test Procedure

5.3.1. HTTP Web-server Demo

The hardware setup for the web-server demo is as given in the figure. The web-server will be running on SAM V71 Xplained Ultra board. The client can be a web browser on computer in the network. The IP address for server will be assigned by a DHCP server running on router hardware.

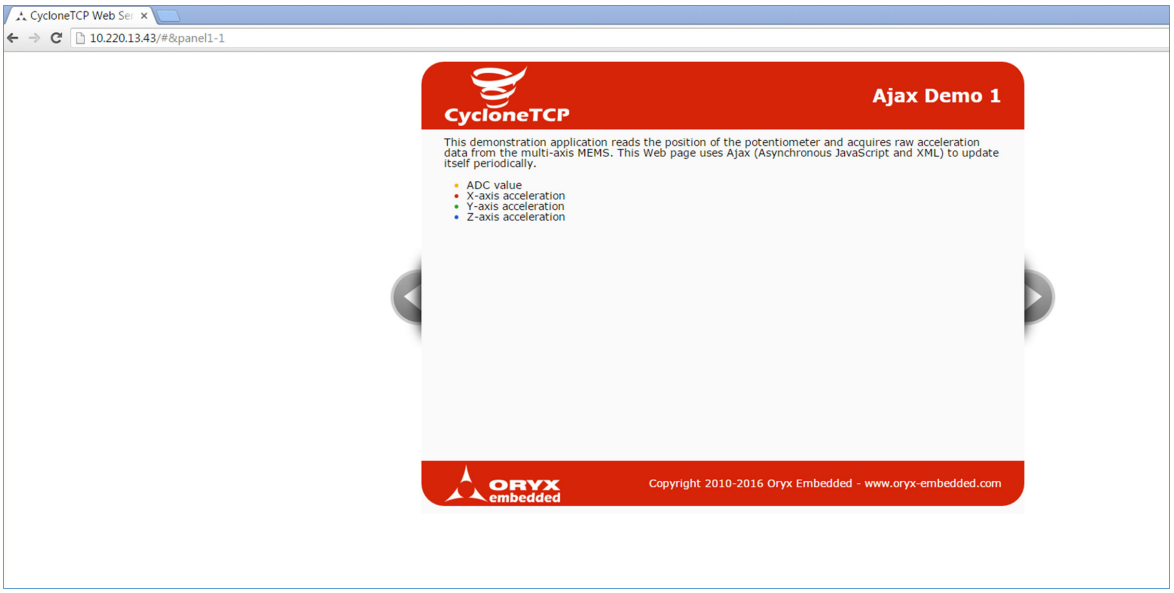
Figure 5-10. HTTP Server Demo - Test Setup



After programming the SAM V71 board with web-server demo, follow the steps given below.

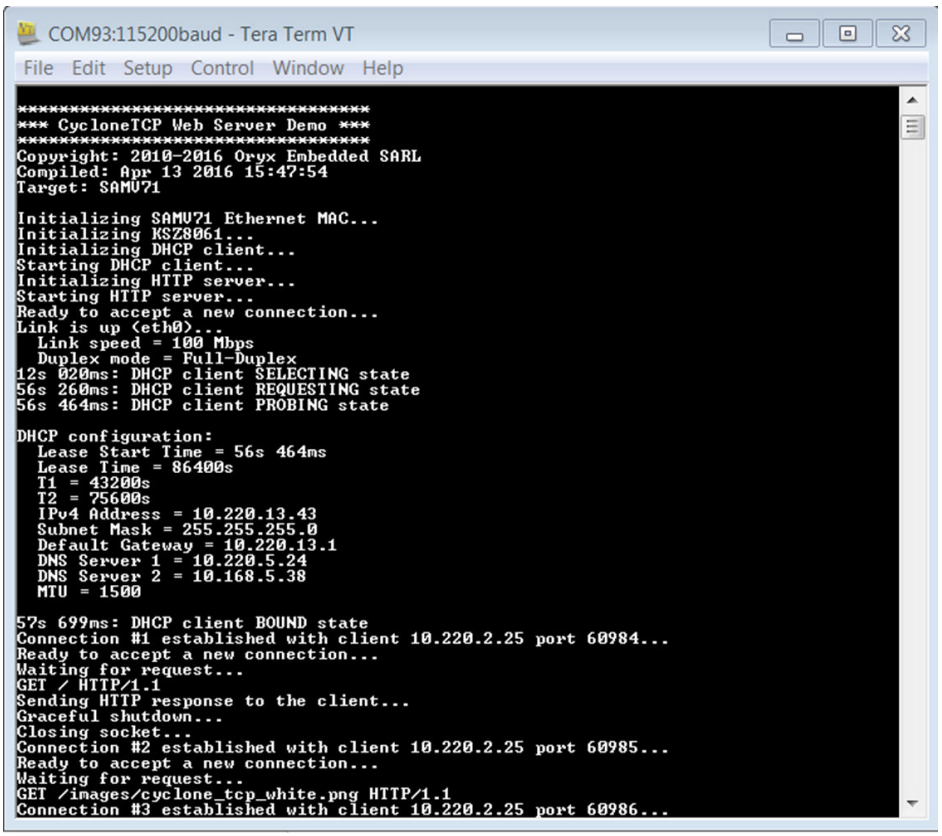
1. Connect the USB cable from computer to ‘Debug USB’ port of SAM V71 Xplained Ultra to power the board.
2. Open the terminal window on computer for the EDBG Virtual COM port enumerated for SAM V71 Xplained Ultra. Set baud rate to 115200. This terminal will show the debug messages.
3. Plug Ethernet cable to SAM V71 Xplained Ultra from a router or from the LAN.
4. Reset SAM V71 Xplained Ultra using reset switch. The terminal will print debug messages.
5. Wait for few seconds to see the DHCP server assigning IP address (xxx.xxx.xxx.xxx) to server running on SAM V71 Xplained Ultra.
6. Open a web browser on a computer which is connected to the same router or to the same LAN.
7. Type the server IP address (xxx.xxx.xxx.xxx) on the address bar of web browser window and press ‘Enter’.
8. The Cyclone TCP Demo page will be displayed on web browser as shown below.

Figure 5-11. HTTP Server Demo - Web Page



9. You also see the terminal window for the debug log. For more detailed analysis, capture the communication between server and client using Wireshark (https://www.wireshark.org/).

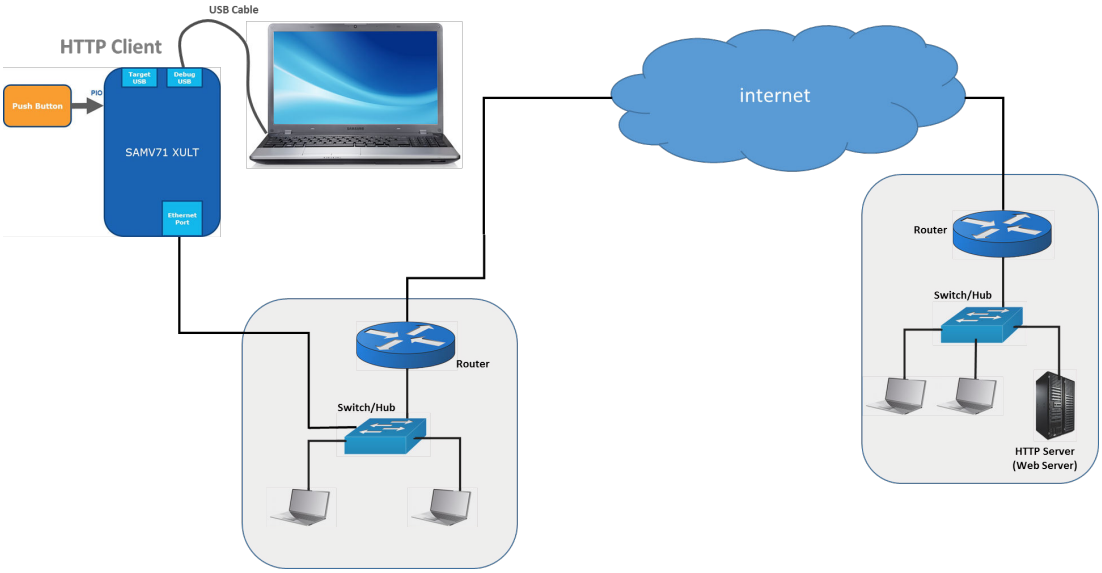
Figure 5-12. HTTP Server Demo - Debug Messages



5.3.2. HTTP Client on SAM V71

The hardware setup for the HTTP Client Demo is as given in the figure below. The client application will be running on SAM V71 Xplained Ultra board. The client interacts with a web server. The IP address for client will be assigned by a DHCP server running on router hardware.

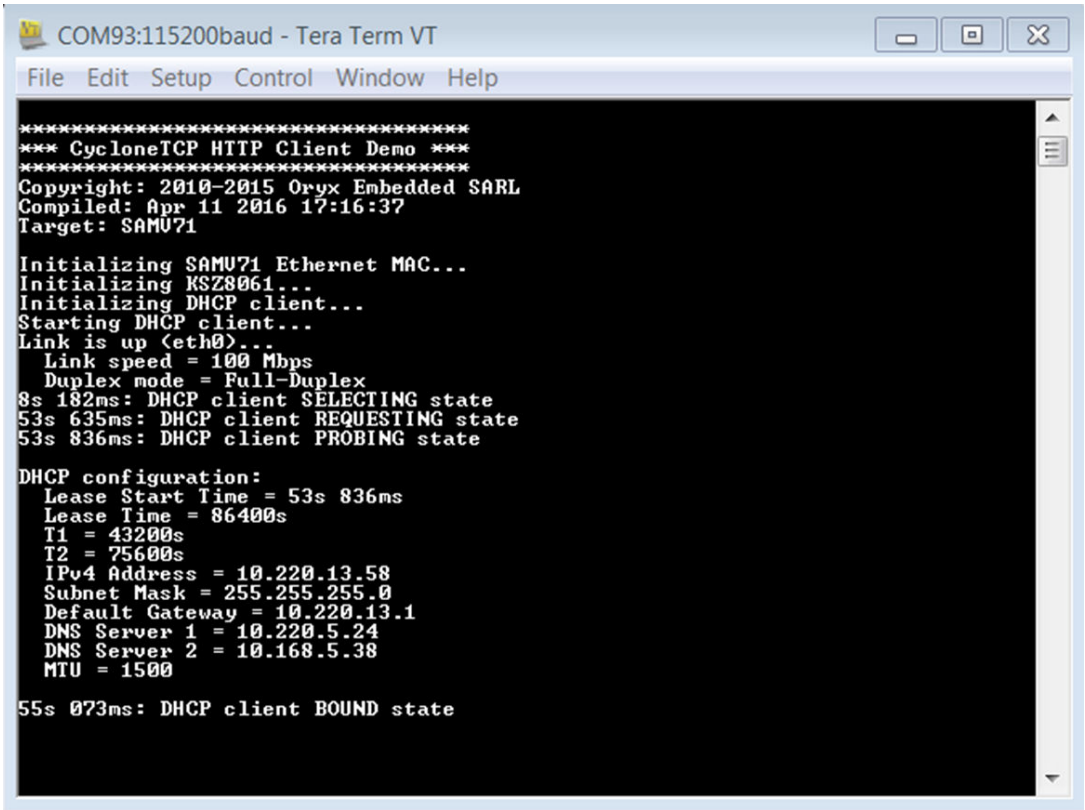
Figure 5-13. HTTP Client Demo - Test setup



After programming the SAM V71 board with HTTP Client Demo, follow the steps given below.

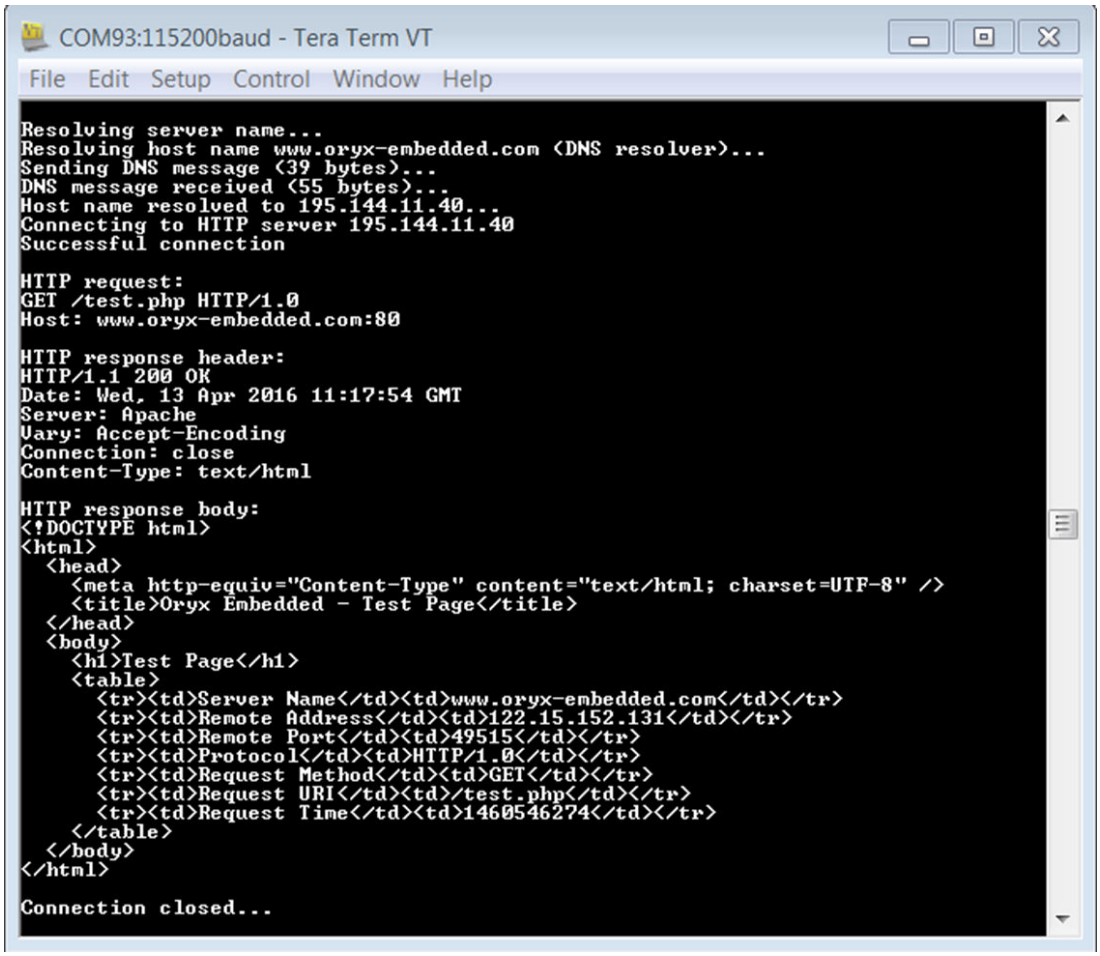
1. Connect the USB cable from computer to ‘Debug USB’ port of SAM V71 Xplained Ultra to power the board.
2. Open the terminal window on computer for the EDBG Virtual COM port enumerated for SAM V71 Xplained Ultra. Set baud rate to 115200. This terminal will show the debug messages.
3. Plug Ethernet cable to SAM V71 Xplained Ultra from a router or from the LAN.
4. Reset SAM V71 Xplained Ultra with reset switch on board. The terminal will print debug messages.
5. Wait for few seconds to see the DHCP server assigning IP address (xxx.xxx.xxx.xxx) to the client running on SAM V71 Xplained Ultra.

Figure 5-14. HTTP Client Demo - DHCP server assigning IP address



6. Press SW0 button on SAM V71 Xplained Ultra. Then client application will send request to ‘www.oryx-embedded.com’
7. The response from ‘www.oryx-embedded.com’ will be printed on debug terminal as follow.

Figure 5-15. HTTP Client Demo - Server response in debug terminal



8. For more detailed analysis, capture the communication between server and client using Wireshark (https://www.wireshark.org/).

6. Conclusion

This application note discuss the basics of Ethernet based network communication. It gives overview of different layers network model. The implementation of network applications over TCP/IP stacks using APIs similar to BSD sockets is described. The different messages passing between two network nodes is also explained.

With higher processing power and Ethernet GMAC peripheral, SAM E70 and SAM V71 devices are good choice for networking application segment.The Ethernet GMAC peripheral implements the lower layers of networking applications.

This document also introduces CycloneTCP TCP/IP stack from Oryx and its supported features. The demo applications are made above CycloneTCP stack running with FreeRTOS.

The concepts discussed in this document is applicable to any available TCP/IP stack, including LwIP stack available with Atmel Software Framework.

7. Suggested reading

7.1. Atmel SMART SAM E70 and V71 Datasheet

The device datasheet contains block diagrams of the peripherals and details about implementing firmware for the device. It also contains the electrical specifications and expected characteristics of the device.

The datasheet is available on http://www.atmel.com in the Datasheets section of the product page.

7.2. ARM Documentation on Cortex-M7 core

- Cortex®-M7 Devices Generic User Guide (ARM DUI 0646A)
- Cortex-M7 Technical Reference Manual (revision r0p2)
- ARM® Coresight ETM-M7 (revision r0p1)

These documents are available at ARM infocenter section.

7.3. CycloneTCP Documentation

Documentation for CycloneTCP stack at http://www.oryx-embedded.com/cyclone_tcp.html.

7.4. FreeRTOS documentation

The FreeRTOS documentation is available at http://www.freertos.org.

7.5. Ethernet PHY

The PHY solutions available at http://www.micrel.com/index.php/products/lan-solutions/phys.html.

7.6. Networking Reference Standards

- **Network Layer (IPv4)**
  - RFC 791: Internet Protocol Specification
  - RFC 792: Internet Control Message Protocol Specification
  - RFC 815: IP Datagram Reassembly Algorithms
  - RFC 826: Ethernet Address Resolution Protocol
  - RFC 1112: Host Extensions for IP Multicasting
  - RFC 1122: Requirements for Internet Hosts - Communication Layers
  - RFC 2113: IP Router Alert Option
  - RFC 2236: Internet Group Management Protocol, Version 2
- **Network Layer (IPv6)**
  - RFC 2460: Internet Protocol, Version 6 (IPv6) Specification
  - RFC 2464: Transmission of IPv6 Packets over Ethernet Networks
  - RFC 2710: Multicast Listener Discovery (MLD) for IPv6
  - RFC 3484: Default Address Selection for Internet Protocol version 6 (IPv6)

  - RFC 3493: Basic Socket Interface Extensions for IPv6
  - RFC 4291: IP Version 6 Addressing Architecture
  - RFC 4294: IPv6 Node Requirements
  - RFC 4443: Internet Control Message Protocol Version 6 (ICMPv6) Specification
  - RFC 4861: Neighbor Discovery for IP version 6 (IPv6)
  - RFC 4862: IPv6 Stateless Address Autoconfiguration
- **Transport Layer**
  - RFC 768: User Datagram Protocol
  - RFC 793: Transmission Control Protocol
  - RFC 2018: TCP Selective Acknowledgment Options
  - RFC 5681: TCP Congestion Control
  - RFC 6298: Computing TCP's Retransmission Timer
- **Application Layer**
  - RFC 959: File Transfer Protocol (FTP)
  - RFC 1035: Domain Names – Implementation and Specification
  - RFC 1945: Hypert ext Transfer Protocol - HTTP/1.0
  - RFC 2131: Dynamic Host Configuration Protocol
  - RFC 2132: DHCP Options and BOOTP Vendor Extensions
  - RFC 2616: Hypertext Transfer Protocol - HTTP/1.1
  - RFC 2617: HTTP Authentication: Basic and Digest Access Authentication
  - RFC 2818: HTTP Over TLS
  - RFC 3207: SMTP Service Extension for Secure SMTP over Transport Layer Security
  - RFC 3315: Dynamic Host Configuration Protocol for IPv6 (DHCPv6)
  - RFC 3646: DNS Configuration options for DHCPv6
  - RFC 4954: SMTP Service Extension for Authentication
  - RFC 5321: Simple Mail Transfer Protocol
  - RFC 6762: Multicast DNS

.

8. Revision history

Doc Rev.	Date	Comments
42738A	06/2016	Initial document release

Atmel | Enabling Unlimited Possibilities®

Atmel Corporation 1600 Technology Drive, San Jose, CA 95110 USA **T:** (+1)(408) 441.0311 **F:** (+1)(408) 436.4200 | **www.atmel.com**