APPLICATION NOTE

Atmel AT02971: Use of Ethernet on SAM4E-EK

Atmel 32-bit Microcontroller

Features

- AT91SAM4E16E Ethernet MAC (GMAC) module
  - Compatible with IEEE® 802.3 Standard
  - 10/100Mbps operation
  - MII Interface to the physical layer
  - Direct Memory Access (DMA) interface to external memory
- Ethernet network introduction
- LwIP stack and porting on SAM4E-EK
- PHY (KSZ8051MNL) implementation on SAM4E-EK
- GMAC demo
  - Web server
  - Trivial File Transfer Protocol (TFTP) server

Introduction

This application note helps users to get familiar with the GMAC interfaces and software stack on SAM4E. Brief introduction and feature highlights of GMAC are given. Some backgrounds of Ethernet network standards are discussed before we go deep into PHY implementation on SAM4E-EK and the lwIP stack. A GMAC demo with features of web and TFTP server, which are implemented under Atmel® Software Framework (ASF), is described, by running the demo application; users could quickly understand Ethernet related applications and GMAC usage on SAM4E.

Table of Contents

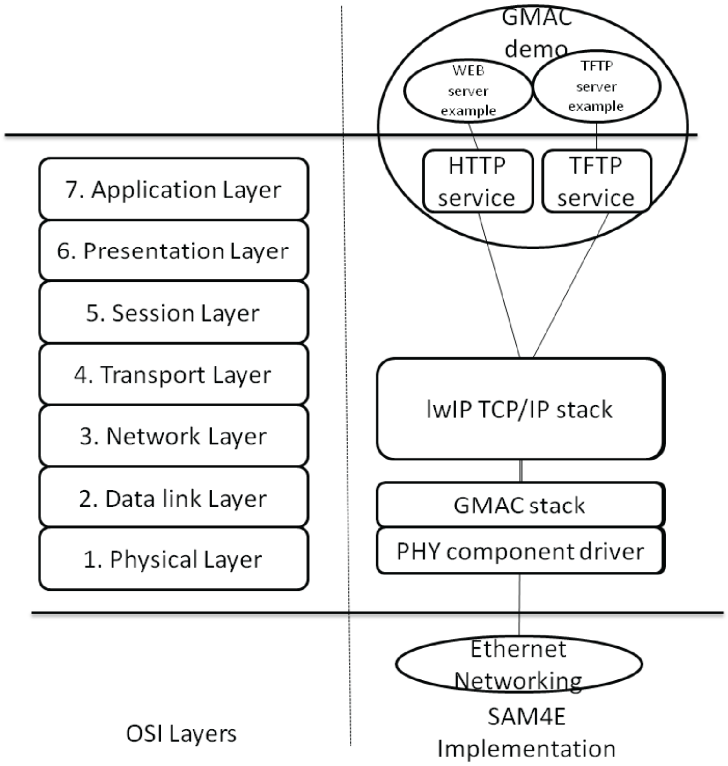
1. Introduction

This is an introduction to the Ethernet protocols, LwIP (Lightweight IP) TCP/IP stack, and lwIP on SAM4E, PHY component of KSZ8051MNL and the GMAC demo on SAM4E-EK.

The Atmel SAM4E serial of flash microcontrollers features a 10/100Mbps Ethernet MAC compatible with the Institute of Electrical and Electronics Engineers (IEEE) 802.3 standard. The GMAC stack provided in the Atmel Software Framework (ASF) simplifies the usage of the GMAC module. The lwIP TCP/IP stack on top of the GMAC stack, which is used for communicating over the Ethernet, is also provided in the ASF to accelerate the Ethernet application development on SAM4E.

Figure 1-1 shows the GMAC implementation on SAM4E compared with Open Systems Interconnection (OSI) layers, which is a widely used networking technology introduced in Ethernet Networking Layers.

Figure 1-1. GMAC Implementation on SAM4E.



1.1 SAM4E Device Overview

The Atmel SAM4E series of flash microcontrollers is based on the high performance ARM® Cortex™-M4 processor up to a maximum speed of 120MHz with Floating Point Unit (FPU). SAM4E offers a rich set of advanced peripherals including GMAC, dual CAN, USB, and HSMCI which make the SAM4E an ideal solution for wide range of industrial applications.

Key features:

- **High performance**
  ARM Cortex-M4 core-based MCU running at 120MHz with integrated FPU and 2KB of cache memory.
- **Connectivity**
  10/100Mbps Ethernet MAC supporting IEEE 1588, dual CAN, full-speed USB device and a full range of high-speed serial peripherals for fast data transfer.

- **Advanced analog**
  Dual 1Msps, 16-bit ADCs of up to 24 channels with analog front end, offering offset error correction and gain control. Also includes 2-channel, 1Msps, 12-bit DAC.
- **Design support**
  Reduce development time and cost with Atmel Studio integrated development platform, which includes ASF, a complete library of source code, project examples, drivers and stacks.

1.2 GMAC on SAM4E

The GMAC module implemented on SAM4E includes the following characteristics:

- Compatible with IEEE 802.3 Standard
- 10/100Mbps operation
- Full and half duplex operation at all three speeds of operation
- Statistics Counter Registers for RMON/MIB
- MII interface to the physical layer
- Integrated physical coding
- Direct Memory Access (DMA) interface to external memory
- Programmable burst length and endianism for DMA
- Interrupt generation to signal receive and transmit completion, or errors
- 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 Virtual Local Area Network (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

The GMAC includes the following signal interfaces:

- Media Independent Interface (MII) to an external PHY
- MDIO interface for external PHY management
- Slave APB interface for accessing GMAC registers
- Master AHB interface for memory access

2. Ethernet Networking Layers

Ethernet is a widely used computer networking technology for local area networks (LANs). The IEEE introduced the Ethernet standard as IEEE 802.3.

The Ethernet standards comprise several wiring and signaling variants of the OSI physical layer in use with Ethernet. This chapter will give a brief introduction to the OSI mode for the Ethernet networking layers.

2.1 OSI Model

The OSI model (ISO/IEC 7498-1) is a product of the Open Systems Interconnection effort at the International Organization for Standardization. Seven layers are defined to characterize and standardize the functions of a communications system in terms of abstraction layer.

Table 2-1 lists the functions of the seven layers.

Table 2-1. OSI Layers.

Layer	Function
Application	Network interfaces to application
Presentation	Data representation, encryption and decryption, convert machine dependent data to machine independent data
Session	Start, managing and stop sessions between applications
Transport	Connection between two network users
Network	Logical Address and path determination
Data link	Physical addressing and Ethernet controller interfaces
Physical	Signal and binary transmission

The following sections give a brief introduction to these layers. As presentation and session layers are not involved in the GMAC stack, these two layers are not discussed.

2.2 Layer 1: Physical Layer

The physical layer defines electrical and physical specifications for devices. The major functions and services performed by the physical layer are:

- Establish data channels between communication mediums
- Transmit bit stream data

2.3 Layer 2: Data Link Layer

The data link layer provides the data link between network entities. It will detect and possibly correct errors that may occur in the physical layer.

Following are the functions of data link layer:

- Framing
- Physical Addressing
- Flow Control
- Error Control
- Access Control
- Media Access Control (MAC)

2.4 Layer 3: Network Layer

The network layer defines interconnected network functions. The network layer is the lowest one in the OSI mode that is concerned with actually getting data from one network endpoints to another even if it is on a remote network.

Following are the major functions of network layer:

- Logical addressing
- Routing
- Datagram encapsulation

2.5 Layer 4: Transport Layer

The transport layer provides transparent transfer of data based on the lower layers, providing reliable data transfer services to the upper layers. It is designed to provide the necessary functions to enable communication between software application processes on different network users.

Following are the major functions of network layer:

- Keep track of data coming from the applications
- Combine application data into a flow of data to send to the low layers
- Lost transmission detection and handling
- Manage the data transmit rate to avoid overwhelmed

2.6 Layer 7: Application Layer

The application layer is the actual layer that is used by the network applications. It provides services for user application to employ.

Following are the major functions of network layer:

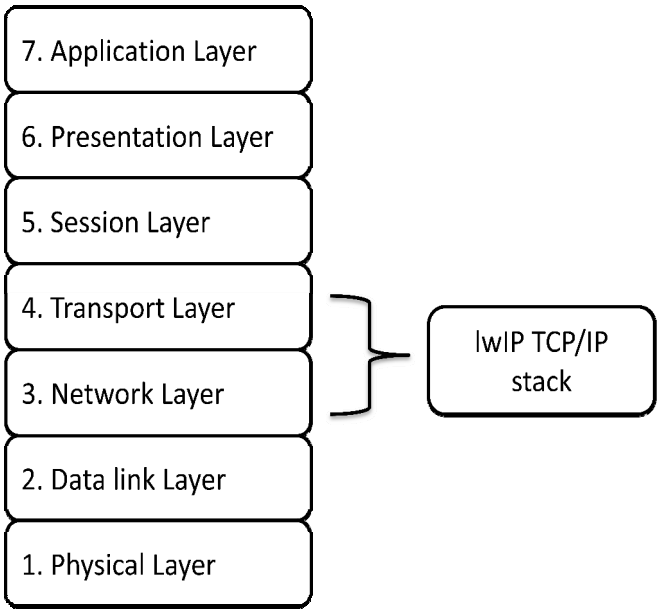
- Issue the appropriate commands to make use of the services provided by the lower layers
- Implement the functions needed by the network user

3. LwIP Stack Overview

LwIP is a free TCP/IP stack licensed under a modified Berkeley Software Distribution (BSD) license. The focus of the lwIP TCP/IP implementation is to reduce resource usage while still having a full scale TCP/IP stack. This makes lwIP widely used in embedded systems.

Figure 3-1 shows the lwIP stack implementation to the OSI layers.

Figure 3-1. LwIP TCP/IP Stack Overview.



3.1 Protocols

LwIP TCP/IP stack includes the following protocols:

- IP (Internet Protocol) including packet forwarding over multiple network interfaces
- ICMP (Internet Control Message Protocol) for network maintenance and debugging
- UDP (User Datagram Protocol) including experimental UDP-lite extensions
- TCP (Transmission Control Protocol) with congestion control, RTT (Round-Trip Time) estimation and fast recovery/fast retransmit
- DHCP (Dynamic Host Configuration Protocol)
- PPP (Point-to-Point Protocol)
- ARP (Address Resolution Protocol) for Ethernet Link and Network Protocols

3.2 Application API Layers

Several ways of using the TCP/IP services are provided by the lwIP TCP/IP stack. Netconn API and BSD socket API will be described in the next sections. Netconn API is used in the web server example and BSD socket API is used in the TFTP example.

3.2.1 Netconn API

The netconn API is high-level sequential API which has a model of execution based on the blocking open-read-write-close paradigm. An operating system is required as this API requires the use of threads. All Ethernet packets processing in the stack are done inside a dedicated thread while application runs in another or other threads.

Table 3-1 provides a summary of the netconn API functions.

Table 3-1. Netconn API Functions.

API Function	Description
netconn_new	Creates a new connection
netconn_delete	Deletes an existing connection
netconn_bind	Binds a connection to a local IP address and port
netconn_connect	Connects to a remote IP address and port
netconn_send	Sends data to the currently connected remote IP/port (not applicable for TCP connections)
netconn_recv	Receives data from a netconn
netconn_listen	Sets a TCP connection into a listening mode
netconn_accept	Accepts an incoming connection on a listening TCP connection
netconn_write	Sends data on a connected TCP netconn
netconn_close	Closes a TCP connection without deleting it

3.2.2 Socket API

LwIP offers the standard BSD socket API. This is a sequential API which is internally built on top of the netconn.

Table 3-2 provides a summary of the main socket API functions.

Table 3-2. Socket API Functions.

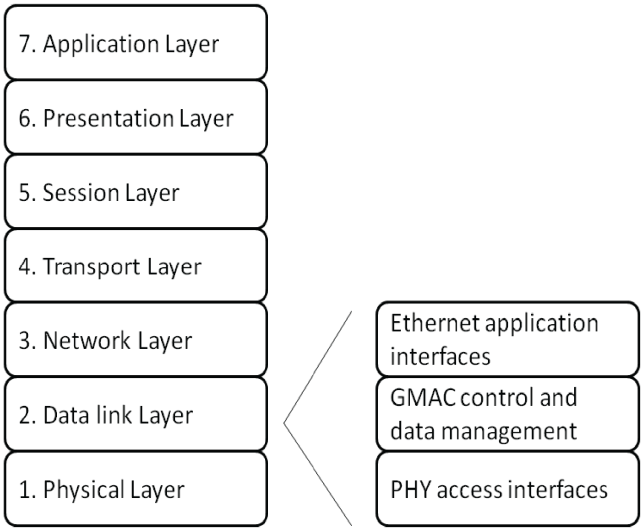
API Function	Description
socket	Creates a new socket
bind	Binds a socket to an IP address and port
listen	Listens for socket connections
connect	Connects a socket to a remote host IP address and port
accept	Accepts a new connection on a socket
read	Reads data from a socket
write	Writes data on a socket
close	Closes a socket

4. GMAC Stack Implementation on SAM4E

The GMAC stack implemented on SAM4E includes PHY access interfaces and Ethernet application interfaces. The GMAC file organization and data structure will be introduced first and then the application interfaces will be described.

Figure 4-1 shows the GMAC stack implementation on SAM4E in the OSI layers.

Figure 4-1. GMAC Stack Implementation on SAM4E.



4.1 GMAC Stack File Organization

Table 4-1 presents the modules of GMAC Stack on the SAM4E-EK.

Table 4-1. GMAC Stack on SAM4E-EK.

Stack Layer	File	Description
GMAC stack	sam\drivers\gmac\gmac.c	GMAC stack interfaces
	sam\drivers\gmac\gmac.h	GMAC stack definitions, function prototypes
PHY component	sam\components\ethernet_phy\ksz8051mnl\ethernet_phy.c	PHY interfaces implementation
	sam\components\ethernet_phy\ksz8051mnl\ethernet_phy.h	PHY interface function prototypes

4.2 GMAC Stack Data Structures

This section gives a brief introduction to the data structures used in the GMAC stack.

4.2.1 gmac_rx_descriptor

The **gmac_rx_descriptor** used for receiving data is defined in the gmac.h file:

```
typedef struct gmac_rx_descriptor {
    union gmac_rx_addr {
        uint32_t val;
        struct gmac_rx_addr_bm {
            uint32_t b_ownership:1,
            b_wrap:1,
```

```
            addr_dw:30;
        } bm;
    } addr;
    union gmac_rx_status {
        uint32_t val;
        struct gmac_rx_status_bm {
            uint32_t len:13,
            b_fcs:1,
            b_sof:1,
            b_eof:1,
            b_cfi:1,
            vlan_priority:3,
            b_priority_detected:1,
            b_vlan_detected:1,
            b_type_id_match:2,
            b_checksumoffload:1,
            b_addrmatch:2,
            b_ext_addr_match:1,
            reserved:1,
            b_uni_hash_match:1,
            b_multi_hash_match:1,
            b_boardcast_detect:1;
        } bm;
    } status;
} gmac_rx_descriptor_t;
```

Structure items:

- b_ownership: Ownership - needs to be zero for the GMAC to write data to the receive buffer. The GMAC sets this to 1 once it has successfully written a frame to memory. Software has to clear this bit before the buffer can be used again
- b_warp: Wrap - marks last descriptor in receive buffer descriptor list
- addr_dw: Address of beginning of buffer
- len: These bits represent the length of the received frame which may or may not include FCS depending on whether FCS discard mode is enabled or not
- b_fcs: This bit has a different meaning depending on whether jumbo frames and ignore FCS modes are enabled or not
- b_sof: Start of frame - when set, the buffer contains the start of a frame. If both bits 15 and 14 are set, the buffer contains a whole frame
- b_eof: End of frame - when set, the buffer contains the end of a frame. If end of frame is not set, the only valid status bit is start of frame (bit 14)
- b_cfi: Canonical Format Indicator (CFI) bit (only valid if bit 21 is set)

- vlan_priority: VLAN priority - only valid if bit 21 is set
- b_priority_detected: Priority tag detected - type ID of 0x8100 and null VLAN identifier. For packets incorporating the stacked VLAN processing feature, this bit will be set if the second VLAN tag has a type ID of 0x8100 and a null VLAN identifier
- b_vlan_detected: VLAN tag detected - type ID of 0x8100. For packets incorporating the stacked VLAN processing feature, this bit will be set if the second VLAN tag has a type ID of 0x8100
- b_type_id_match: This bit has a different meaning depending on whether RX checksum offloading is enabled or not
- b_checksumoffload: This bit has a different meaning depending on whether RX checksum offloading is enabled or not. More information could be referred in the datasheet
- b_addrmatch: Specific Address Register match
- b_ext_addr_match: Specific Address Register match found, bit 25 and bit 26 indicate which Specific Address Register causes the match
- b_uni_hash_match: Unicast hash match
- b_multi_hash_match: Multicast hash match
- b_boardcast_detect: Global all ones broadcast address detected

4.2.2 gmac_tx_descriptor

The gmac_tx_descriptor used for transmitting data is defined in the gmac.h file:

```
typedef struct gmac_tx_descriptor {
    uint32_t addr;
    union gmac_tx_status {
        uint32_t val;
        struct gmac_tx_status_bm {
            uint32_t len:14,
            reserved:1,
            b_last_buffer:1,
            b_no_crc:1,
            reserved1:3,
            b_checksumoffload:3,
            reserved2:3,
            b_lco:1,
            b_exhausted:1,
            b_underrun:1,
            b_error:1,
            b_wrap:1,
            b_used:1;
        } bm;
    } status;
} gmac_tx_descriptor_t;
```

Structure items:

- Len: Length of buffer
- b_last_buffer: Last buffer, when set this bit will indicate that the last buffer in the current frame has been reached
- b_no_crc: No CRC to be appended by MAC
- b_checksumoffload: Transmit IP/TCP/UDP checksum generation offload errors
- b_lco: Late collision, transmit error detected
- b_exhausted: Transmit frame corruption due to AHB error
- b_underrun: Transmit underrun
- b_error: Retry limit exceeded, transmit error detected
- b_wrap: Wrap - marks last descriptor in transmit buffer descriptor list
- b_used: Used - must be zero for the GMAC to read data to the transmit buffer
- addr: Byte address of buffer

4.2.3 gmac_options

The **gmac_options** used for GMAC stack configuration is defined in the gmac.h file:

```
typedef struct gmac_options {
    uint8_t uc_copy_all_frame;
    uint8_t uc_no_boardcast;
    uint8_t uc_mac_addr[GMAC_ADDR_LENGTH];
} gmac_options_t;
```

Structure items:

- uc_copy_all_frame: Enable/disable CopyAllFrame
- uc_no_boardcast: Enable/disable NoBroadCast
- uc_mac_addr: MAC address

4.2.4 gmac_device

The **gmac_device** used a generic GMAC stack device is defined in the gmac.h file:

```
typedef struct gmac_device {
    Gmac *p_hw;
    uint8_t *p_tx_buffer;
    uint8_t *p_rx_buffer;
    gmac_rx_descriptor_t *p_rx_dscr;
    gmac_tx_descriptor_t *p_tx_dscr;
    gmac_dev_tx_cb_t func_rx_cb;
    gmac_dev_wakeup_cb_t func_wakeup_cb;
    gmac_dev_tx_cb_t *func_tx_cb_list;
    uint16_t us_rx_list_size;
    uint16_t us_rx_idx;
    uint16_t us_tx_list_size;
    uint16_t us_tx_head;
    uint16_t us_tx_tail;
```

```
    uint8_t uc_wakeup_threshold;
} gmac_device_t;
```

Structure items:

- p_hw: Pointer to HW register base
- p_tx_buffer: Pointer to allocated TX buffer
- p_rx_buffer: Pointer to allocated RX buffer
- p_rx_dscr: Pointer to Rx TDs (must be 8-byte aligned)
- p_tx_dscr: Pointer to Tx TDs (must be 8-byte aligned)
- func_rx_cb: Optional callback to be invoked once a frame has been received
- func_wakeup_cb: Optional callback to be invoked once several TDs have been released
- func_tx_cb_list: Optional callback list to be invoked once TD has been processed
- us_rx_list_size: RX TD list size
- us_rx_idx: RX index for current processing TD
- us_tx_list_size: TX TD list size
- us_tx_head: Circular buffer head pointer by upper layer (buffer to be sent)
- us_tx_tail: Circular buffer tail pointer incremented by handlers (buffer sent)
- uc_wakeup_threshold: Number of free TD before wakeup callback is invoked

4.3 PHY Access Interfaces

There are two interfaces provided in the GMAC stack to access the PHY components, which are listed in Table 4-2.

Table 4-2. GMAC PHY Access Interfaces.

Function Name	Description
gmac_phy_read	Read the PHY register
gmac_phy_write	Write the value to the PHY register

4.4 Ethernet Application Interfaces

Table 4-3 lists the functions of the Ethernet application interfaces. Through these application interfaces the user can use the GMAC features without implementing the detail of the GMAC stack.

Table 4-3. Ethernet Application Interfaces.

Function Name	Description
gmac_dev_init	Initialize the GMAC driver
gmac_dev_read	Frames can be read from the GMAC in multiple sections
gmac_dev_write	Send bytes from TX buffers
gmac_dev_get_tx_load	Get current load of transmit
gmac_dev_set_rx_callback	Register/Clear RX callback. Callback will be invoked after the next received frame
gmac_dev_set_tx_wakeup_callback	Register/Clear TX wakeup callback
gmac_dev_reset	Reset TX & RX queue & statistics
gmac_handler	GMAC interrupt handler

5. PHY Implementation

The PHY component of KSZ8051MNL is used in the SAM4E-EK. This chapter gives a brief introduction to the KSZ8051 implementations.

5.1 KSZ8051MNL Introduction

There is a standard PHY access interfaces defined in the ASF, Table 5-1 lists these interfaces.

Table 5-1. Ethernet Application Interfaces.

Function Name	Description
ethernet_phy_init	Perform a HW initialization to the PHY and set up clocks
ethernet_phy_set_link	Get the Link & speed settings, and automatically set up the GMAC with the settings
ethernet_phy_auto_negotiate	Issue an auto negotiation of the PHY
ethernet_phy_reset	Issue a SW reset to reset all registers of the PHY

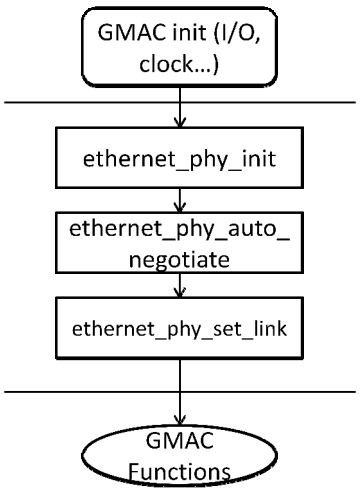
For SAM4E-EK the PHY component interfaces are implemented in the file:
sam\components\ethernet_phy\ksz8051mnl\ethernet_phy.c.

5.2 Initializing KSZ8051MNL

Based on the GMAC stack on SAM4E, the procedures to initialize the KSZ8051 is simplified to three steps:

- Initialize the PHY
- Enabling PHY auto-negotiation mode or manually selecting the mode of operation (Full-speed/Low-speed, Half-duplex/Full-duplex)
- If PHY auto-negotiation mode is selected, the application needs to poll the PHY or use a PHY interrupt in order to obtain the result of auto-negotiation (speed, duplex mode)

Figure 5-1. KSZ8051MNL Initializing Procedures.



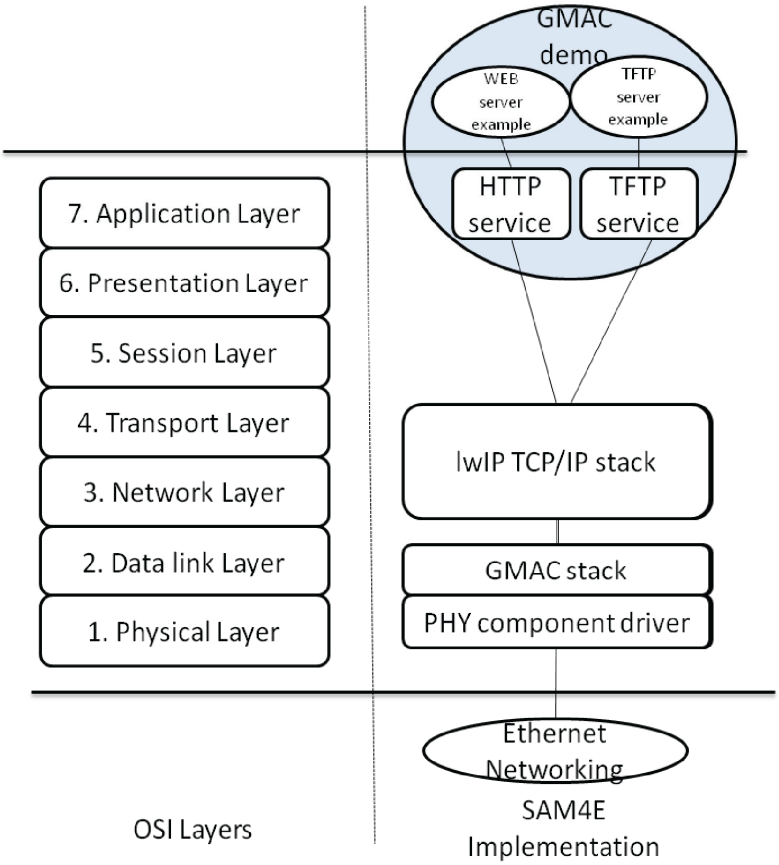
Three interfaces should be called to initialize the KSZ8051MNL component:

- ethernet_phy_init() is used to initialize the PHY, including setting the clock and the address
- ethernet_phy_auto_negotiate() is used to do the auto negotiation and get link parameters
- ethernet_phy_set_link() to set the link parameters

6. GMAC Demo on SAM4E-EK

The GMAC demo includes two applications for the usage of GMAC and lwIP stack on SAM4E, i.e., the web server example and TFTP example. Figure 6-1 shows the GMAC demo diagram.

Figure 6-1. GMAC Demo Overview.



6.1 GMAC Demo File Organization

The GMAC demo code is located in the folder: thirdparty/freertos/demo/lwip_sam_example. Table 6-1 lists the files used in the GMAC demo.

Table 6-1. GMAC Demo Used Files.

Function Name	Description
main.c	Main routine of the example
lwipopts.h	LwIP configurations
FreeRTOSConfig.h	FreeRTOS configurations
Network/ethernet.c	Ethernet management
Network/ethernet.h	Header files for Ethernet management
Partest/PartTest.c	LED management

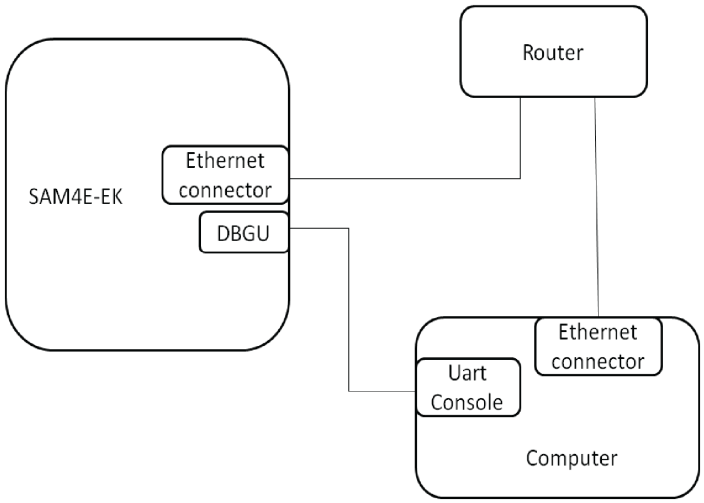
The web server and TFTP server code are located in the thirdparty/freertos/demo/lwip_avr32_uc3_example/network folder.

Function Name	Description
Basictftp/BasicTFTP.c	TFTP server implementation
Basictftp/BasicTFTP.h	TFTP server application interfaces
Basicweb/BasicWEB.c	Web server implementation
Basicweb/BasicWEB.h	Web server application interfaces

6.2 GMAC Demo Requirements

A router which provides the IP address is required to connect the SAM4E-EK and the computer. Figure 6-2 shows the hardware diagram of this example.

Figure 6-2. GMAC Demo Diagram.



For the TFTP usage, a TFTP client should be installed on the computer.

6.3 LwIP Porting on SAM4E

Two parts are involved in the porting of lwIP on SAM4E, the lwIP low level drivers and lwIP configuration.

6.3.1 Initializing LwIP

The ethernet_if.c file in the thirdparty/lwip/lwip-port-1.4.0/sam/netif folder is used to link the lwIP stack to the GMAC Ethernet network interface. Table 6-2 lists the interfaces used for this link.

Table 6-2. LwIP TCP/IP Stack Low Level Drivers.

Function Name	Description
low_level_init	Calls the Ethernet driver functions to initialize the GMAC Ethernet peripheral
low_level_output	Calls the Ethernet driver functions to send an Ethernet packet
low_level_input	Should allocate a buffer and transfer the bytes of the incoming packet from the interface into the buffer
ethernetif_init	Calls low_level_init() to set up the network interface
ethernet_input	Calls low_level_input to receive a packet and provide it to the lwIP stack

In the case of the RTOS implementation, an additional file is required; sys_arch.c. This file implements an emulation layer for the RTOS services. The implementation can be found in thirdparty/lwip/lwip-port-1.4.0/sam/sys_arch.c.

6.3.2 Configuring LwIP

The lwipopt.h file in the thirdparty/freertos/demo/lwip_sam_example folder is used to configure the lwIP TCP/IP stack. Table 6-3 lists the major configuration items.

Table 6-3. LwIP Configurations.

Function Name	Description
LWIP_DHCP	Enable/disable DHCP
MEM_SIZE	LwIP heap memory size: used for all lwIP dynamic memory allocations
MEMP_NUM_PBUF	Total number of MEM_REF and MEM_ROM pbufs
MEMP_NUM_TCP_PCB	Total number of TCP PCB structures
MEMP_NUM_TCP_PCB_LISTEN	Total number of listening TCP PCBs
MEMP_NUM_TCP_SEG	The maximum number of simultaneously queued TCP segments
PBUF_POOL_SIZE	The total number of pbufs of type PBUF_POOL
PBUF_POOL_BUFSIZE	Size of a pbuf of type PBUF_POOL
TCP_MSS	TCP maximum segment size
TCP_SND_BUF	TCP send buffer space for a connection
TCP_SND_QUEUELEN	Maximum number of pbufs in the TCP send queue
TCP_WND	Advertised TCP receive window size

6.4 FreeRTOS Introduction

As mentioned in the netconn API, an Operation System (OS) is required to provide the thread services. The FreeRTOS is used in the GMAC demo.

FreeRTOS is a scaleable realtime kernel designed for embedded systems, with small memory cost and full function of thread support. For more information about FreeRTOS, see www.freertos.org.

In the GMAC demo the TCP/IP task is created by lwIP stack. It's an internal task and all the TCP/IP stack functions are processed through this task.

Two application tasks are created for the Ethernet usage: web server and TFTP tasks. The implementations of these two tasks will be discussed in the next sections.

```
/* Create the WEB server task. This uses the lwIP RTOS abstraction layer. */
sys_thread_new("WEB", vBasicWEBServer, (void *)NULL,
        lwipBASIC_WEB_SERVER_STACK_SIZE,
        lwipBASIC_WEB_SERVER_PRIORITY);


/* Create the TFTP server task. This uses the lwIP RTOS abstraction layer. */
sys_thread_new("TFTP", vBasicTFTPServer, (void *)NULL,
        lwipBASIC_TFTP_SERVER_STACK_SIZE,
        lwipBASIC_TFTP_SERVER_PRIORITY);
```

6.5 Web Server Implementation

The following code is the main routine for the web server example.

```
{
struct netconn *pxHTTPListener, *pxNewConnection;

    /* Create a new tcp connection handle */
    pxHTTPListener = netconn_new(NETCONN_TCP);
    netconn_bind(pxHTTPListener, NULL, webHTTP_PORT);
    netconn_listen(pxHTTPListener);

    /* Loop forever */
    for( ;; )
    {
     while(netconn_accept(pxHTTPListener, &pxNewConnection) != ERR_OK)
     {
        vTaskDelay(webSHORT_DELAY);
         }
         vParTestSetLED(webCONN_LED, pdTRUE);

         if(pxNewConnection != NULL)
         {
              prvweb_ParseHTMLRequest(pxNewConnection);
         }/* end if new connection */

         vParTestSetLED(webCONN_LED, pdFALSE);

    } /* end infinite loop */
}
```

Example description:

- Function netconn_new(NETCONN_TCP) is called to create a new networking connection. Hypertext transfer protocol (HTTP) protocol is based on TCP, so TCP protocol, defined as NETCONN_TCP, is used
- Bind the net connection to the IP address and port. It is used for web server purpose, the address is left empty and the port number should be 80.
  netconn_bind(pxHTTPListener, NULL, webHTTP_PORT))
- Listen for the web client request
  netconn_listen(pxHTTPListener)
  Then the web server is initialized to waiting for web client request.
- netconn_accept() to accept a connection
- prvweb_ParseHTMLRequest() is used to handle the http request

6.6 TFTP Server Implementation

The following code is the main routine for the TFTP server example.

```
{
  // Create socket
    lSocket = socket(AF_INET, SOCK_DGRAM, 0);
    if (lSocket < 0) {
      return;
    }
    memset((char *)&sLocalAddr, 0, sizeof(sLocalAddr));
    sLocalAddr.sin_family = AF_INET;
    sLocalAddr.sin_len = sizeof(sLocalAddr);
    sLocalAddr.sin_addr.s_addr = htonl(INADDR_ANY);
    sLocalAddr.sin_port = htons(TFTP_PORT);

    if (bind(lSocket, (struct sockaddr *)&sLocalAddr, sizeof(sLocalAddr)) < 0) {
      // Problem setting up my end
      close(lSocket);
      return;
    }
    lRecvLen = sizeof(cData);
    lFromLen = sizeof(sFromAddr);
    lDataLen = recvfrom(lSocket, sHdr, lRecvLen, 0,
                  (struct sockaddr *)&sFromAddr, &lFromLen);
    vParTestSetLED( TFTP_LED , pdTRUE );
    close(lSocket); // so that other servers can bind to the TFTP socket

  switch (ntohs(sHdr->th_opcode)) {
      case WRQ:
        tftpd_write_file(sHdr, &sFromAddr, lFromLen);
        vParTestSetLED( TFTP_LED , pdFALSE );
        break;
      case RRQ:
        tftpd_read_file(sHdr, &sFromAddr, lFromLen);
        vParTestSetLED( TFTP_LED , pdFALSE );
        break;
      case ACK:
      case DATA:
```

```
            case ERROR:
              vParTestSetLED( TFTP_LED , pdFALSE );
              // Ignore
              break;
    }
```

Example description:

- Function socket (AF_INET, SOCK_DGRAM, 0) is called to create a new socket. The socket is using the type SOCK_DGRAM and protocol AF_INET. In the Internet domain, the SOCK_DGRAM socket type is implemented on the User Datagram Protocol/Internet Protocol (UDP/IP) protocol. AF_INET means the socket is IP address and port number.
- Set the IP address properties by the code:
  *memset((char *)&sLocalAddr, 0, sizeof(sLocalAddr));*
  sLocalAddr.sin_family = AF_INET;
  sLocalAddr.sin_len = sizeof(sLocalAddr);
  sLocalAddr.sin_addr.s_addr = htonl(INADDR_ANY);
  sLocalAddr.sin_port = htons(TFTP_PORT);
- Bind the socket to the IP address and port:
  *bind(lSocket, (struct sockaddr *)&sLocalAddr, sizeof(sLocalAddr))*
- Get the TFTP request from remote:
  *recvfrom(lSocket, sHdr, lRecvLen, 0, (struct sockaddr *)&sFromAddr, &lFromLen);*
- Close the socket so that other servers can bind to the TFTP socket:
  close(lSocket);
- Handle TFTP requests according to the opcode: switch (ntohs(sHdr->th_opcode))
- Call tftpd_write_file(sHdr, &sFromAddr, lFromLen) to handle write in case of write request
- Call tftpd_read_file(sHdr, &sFromAddr, lFromLen) to handle read in case of read request

6.7 Getting Started Using GMAC Demo

6.7.1 GMAC Demo Kit

The kit used for GMAC demo is the SAM4E-EK. The SAM4E-EK is an evaluation kit featuring the SAM4E16 device BGA144 package with optional socket footprint, on board 12MHz and 32.768kHz crystal, a 2.8” TFT color LCD display with touch panel and backlight, one Ethernet physical transceiver layer with RJ45 connector, CAN port with driver, mono/stereo headphone jack output, QTouch® interfaces, full speed USB device port, Serial Flash memory, NAND Flash memory, SD/MMC interface, LEDs, push buttons, BNC connector for ADC input and DAC output, JTAG/ICE port, UART port with RS232 driver, USART port with RS232 driver multiplexed with RS485 function with driver. For more information about SAM4E-EK, see www.atmel.com/tools/SAM4E-EK.asfpx.

6.7.2 Tools for GMAC Demo Software

The GMAC demo is developed based on the Atmel Studio 6. The following tools are required for the startup of the GMAC demo:

- Atmel Studio 6.1 (or above) installed
- Segger J-Link (the latest version) installed

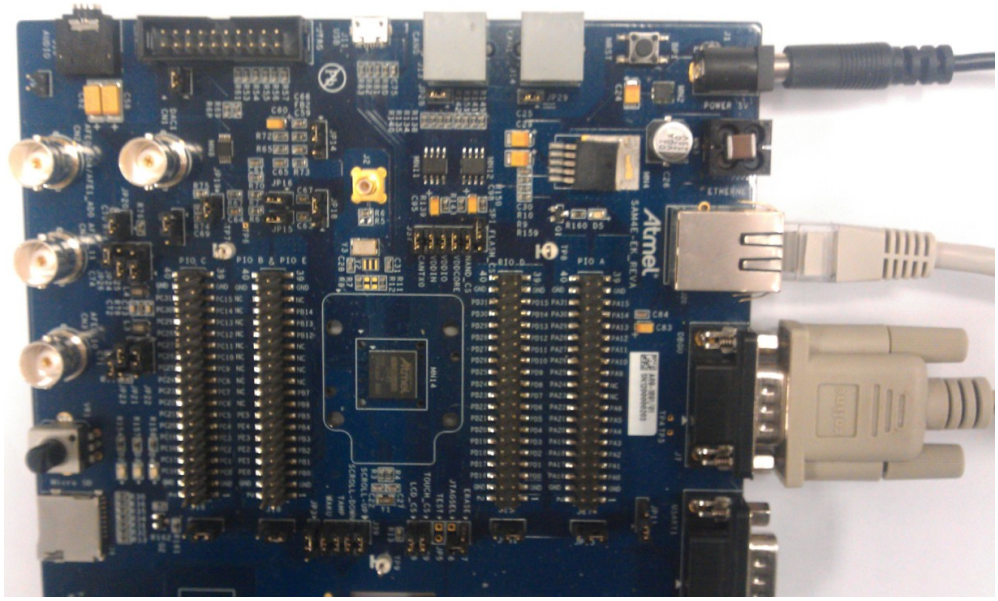
A SAM-ICE™ is required for programming and debugging purpose. If programming the GMAC demo only, the Atmel SAM-BA® software can be used instead of the SAM-ICE. For more information about SAM-BA usage, see www.atmel.com/tools/ATMELSAM-BAIN-SYSTEMPROGRAMMER.aspx.

6.7.3 GMAC Demo Board Connections

The GMAC demo uses DHCP by default, so it is required to connect the SAM4E-EK to the debug console. Figure 6-3 shows the typical connection of SAM4E-EK.

- Build and program the demo code into the SAM4E flash memory
- Be sure the debug port of SAM4E-EK has been connected to the computer

Figure 6-3. SAM4E-EK Board Connections.



6.7.4 Load GMAC Demo

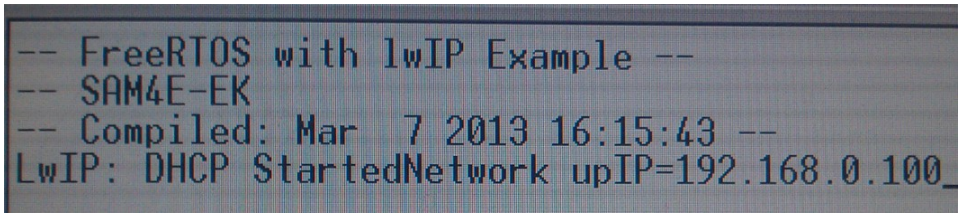
- Download the zipped file of GMAC demo code at the same web page of this application note
- Unzip the file and double click the GMAC_Demo.cproj. Then the Atmel Studio will open the demo
- Build the project: Build → Build Solution
- Load the code in SAM4E and start debugging: Debug → Start Debugging and Break

Now the GMAC demo has been programmed and the debugger stops at the beginning of main(). To execute it, click on Debug → Continue.

6.7.5 GMAC Demo Execution

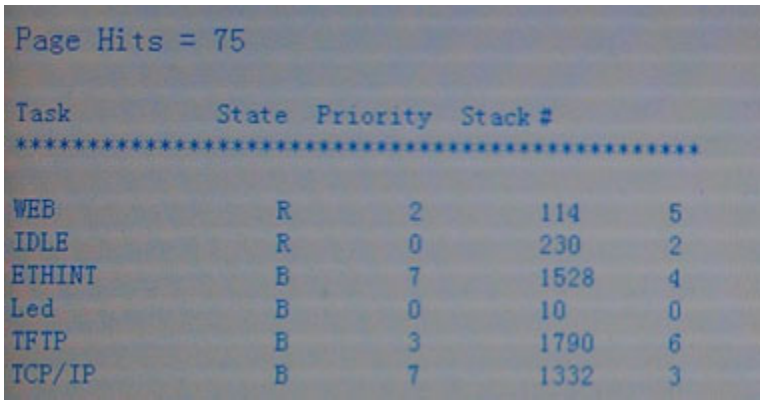
- Get the IP address assigned by DHCP. If failed, the address will not appear in the console. Figure 6-4 shows the sample screen of DHCP assignment, 192.168.0.100 is assigned to the board for example.

Figure 6-4. DHCP Address Assignment.



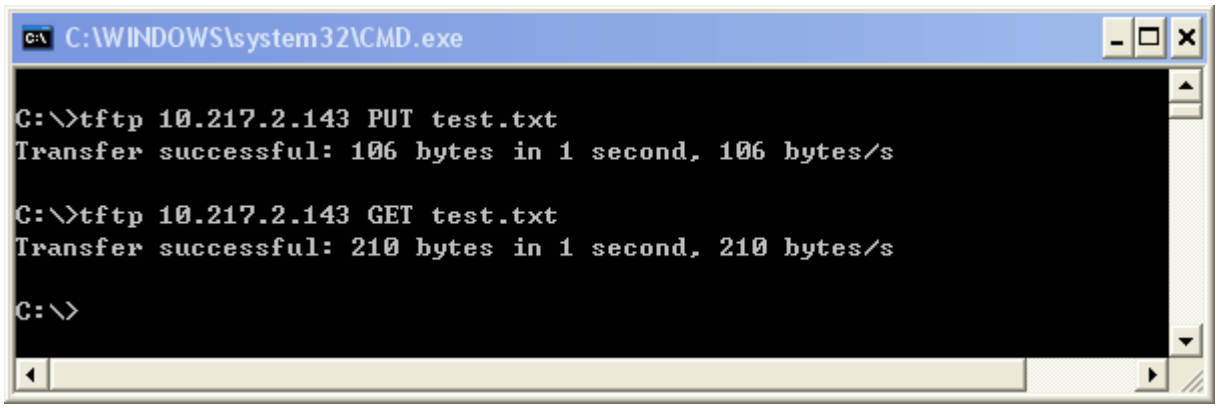
- Open a web client, such as Internet Explorer, and type the board's IP address in the web browser; the statistics of the board will be shown in the browser, as shown in Figure 6-5

Figure 6-5. SAM4E-EK Statics in the Web Browser.



- Open the TFTP client and type the TFTP commands to exchange. Figure 6-6 shows a sample file transfer of the TFTP server

Figure 6-6. TFTP File Transfer Example.



7. Revision History

Doc. Rev.	Date	Comments
42134A	05/2013	Initial document release

Atmel | Enabling Unlimited Possibilities®

Atmel Corporation
1600 Technology Drive
San Jose, CA 95110
USA
Tel: (+1)(408) 441-0311
Fax: (+1)(408) 487-2600
www.atmel.com

Atmel Asia Limited
Unit 01-5 & 16, 19F
BEA Tower, Millennium City 5
418 Kwun Tong Road
Kwun Tong, Kowloon
HONG KONG
Tel: (+852) 2245-6100
Fax: (+852) 2722-1369

Atmel Munich GmbH
Business Campus
Parkring 4
D-85748 Garching b. Munich
GERMANY
Tel: (+49) 89-31970-0
Fax: (+49) 89-3194621

Atmel Japan G.K.
16F Shin-Osaki Kangyo Building
1-6-4 Osaki, Shinagawa-ku
Tokyo 141-0032
JAPAN
Tel: (+81)(3) 6417-0300
Fax: (+81)(3) 6417-0370