**Table 1: Cross Reference of Applicable Products**

| PRODUCT NAME | MANUFACTURER PART NUMBER | SMD # | DEVICE TYPE | INTERNAL PIC NUMBER |
|---|---|---|---|---|
| UT700 LEON | UT700 | 5962-13238 | Ethernet MAC Controller Module | WQ03 |

## 1.0 Overview

The UT700 LEON 3FT SPARC Processor provides one Ethernet Media Access Controller (MAC) that interfaces between the UT700 LEON's AMBA AHB and an external Ethernet PHY to an Ethernet network. This Ethernet MAC supports 10/100 Mbits speed in both full- and half-duplex modes. DMA engines are part of the Ethernet MAC design that speeds up the data transaction from the main memory to and from the transmitter and receiver FIFOs'.

Although the Ethernet MAC also provides an additional feature such as the Ethernet Debug Communication Link (EDCL), this feature will not be discussed in this application note. Instead, this application note will focus on how to enable the Ethernet MAC module to communicate with the Ethernet network.

In addition to the Ethernet MAC, this application note will briefly discuss the Ethernet PHY loopback feature for its use in data transaction testing from main memory to the transmitter, through the loopback path to the receiver, and back to the main memory.

This is software centric Application Note, the discussion of hardware in this application note will be restricted to connectivity verification and testing.

**NOTE:** Ethernet Interface operation is intended for terrestrial use only, and not guaranteed in radiation environments.

The description in this application note describes how to directly use the memory mapped interface of a specific hardware peripheral. If you are using an operating system such as RTEMS, Linux, and VxWorks or an environment such as BCC then it is recommended to use the infrastructure provided by those environments instead of accessing the peripheral directly as described in this application note.

## 1.1 Application Note Layout

This application note (AN) starts by providing a brief description of the Ethernet Media-Independent Interface (MII) and its functional purposes. The description of the MII falls under the Ethernet Hardware sections.

After the Ethernet Hardware sections, this AN provides high-level flow diagrams to depict the correct sequential steps to initialize the MAC's transmitter and receiver for data transaction with the Ethernet network. The next section provides the significant order of the enabling of the internal Ethernet blocks. This flow diagram and its elaborations fall under the Ethernet Enable Flow Diagrams.

Finally, we can apply this knowledge using C programming code to enable the UT700 LEON to communicate with the Ethernet network. The programming of C code falls under the Ethernet Programming sections.

These sub-sections are described in detail below:

- Ethernet Hardware
- Ethernet Initialization Flow Diagram
- Ethernet Programming

# 2.0 Ethernet Hardware

Since this is a software centric AN, the discussion of hardware will be restricted to connectivity verification and testing.

## 2.1 Ethernet Media-Independent Interface (MII)

The UT700 LEON Ethernet MAC provides an MII interface to connect to external Ethernet PHYs. This MII interface consists of a Management Data Input/Output (MDIO) interface, transmission, reception, clock and control signals.

The MDIO interface provides a conduit to initialize and program the external Ethernet PHY. The Ethernet PHY provides the output clock signals (TX_CLK and RX_CLK) to synchronize the data flow between the Ethernet MAC and the PHY.

The input clock frequency to the Ethernet PHY's depends on its mode of operation. When the Ethernet PHY is operating in MII or RMII mode, the input clocks frequency to the PHY must be set to 25MHz and 50MHz respectively. Since the UT700 LEON Ethernet MAC only supports MII mode operation, a 25MHz input clock frequency will suffice.

The UT700 LEON Functional Manual, **Chapter 14**, provides more information about the Ethernet MAC. Also, see **Figure 14.1** of the UT700 LEON Functional Manual that depicts the Ethernet MAC block diagram and its internal structure.

## 2.2 Ethernet PHY Loopback Feature

Most Ethernet PHY provides a hardware loopback path that allows data testing without being physically connected to a network. On the other hand, the UT700 LEON Ethernet MAC does not provide any loopback path for testing; if this is a method required for your testing, then software is needed to move data from the receiver's FIFO to the transmitter's FIFO.

## 3.0 Ethernet Initialization Flow Diagram

In this section, we will provide a flow diagram that depicts the proper sequential steps to initialize the Ethernet MAC as shown in **Figure 1**.

The initialization process involves both allocating resources from the main memory and the settings of the MAC internal registers. In the following sections, we will provide a basic understanding of the initialization of the individual block as shown in **Figure 1.**

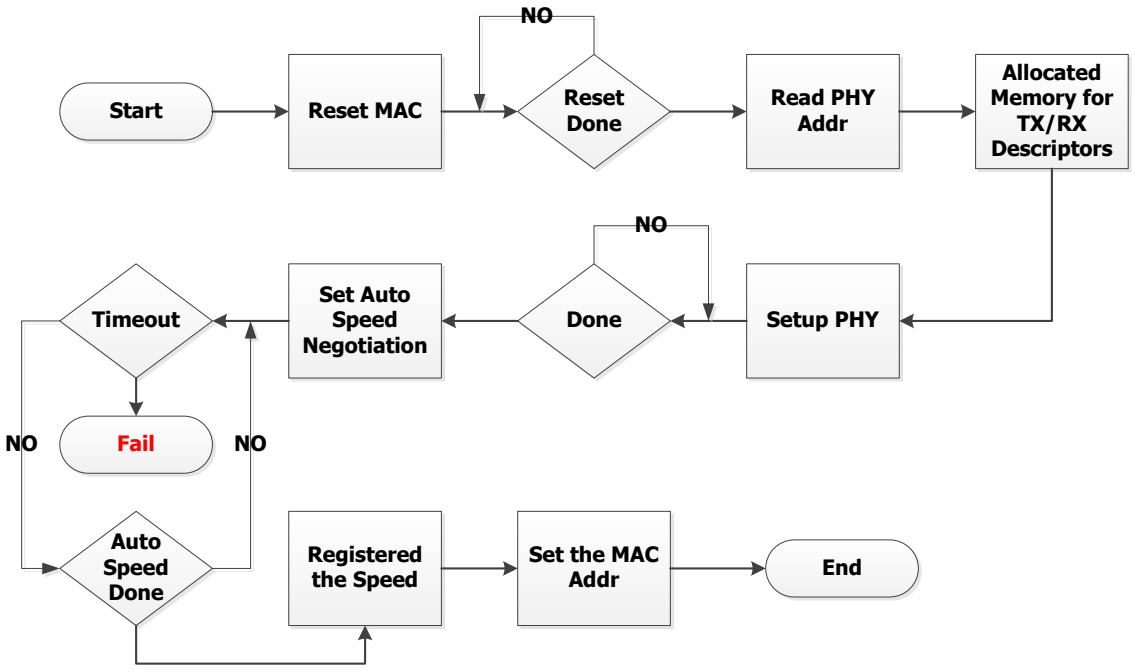


**Figure 1: Ethernet Initialization Flow Diagram**

## 3.1 Reset MAC

The reset step allows the MAC's registers to restore to its factory reset state. In this state, it assures the MAC is disabled. Once in this state, we can proceed to initialize the MAC.

## 3.2 Read PHY Address

The UT700 LEON Ethernet MAC supports multiple Ethernet PHYs, but this example we are only using one PHY. The "Read PHY address" step allows us to send the PHY configuration to the correct PHY.

## 3.3 Allocate Memory for Descriptors

The UT700 LEON Ethernet MAC includes a DMA engine that supports data packets transfer from the main memory to and from the transmitter and receiver FIFOs. This method of transferring packets improves the throughput of the application by eliminating the software overhead and spinwait time lost for waiting to read or write the FIFOs.

In order to support this feature, additional data structures and data packets buffers are created in the main memory. The data structures are the Ethernet transmitter and receiver descriptors that provide information to the DMA engine on how to move data packets to and from main memory and the FIFOs.

The MAC also provides a simple round robin method to track both the transmitted and received packets using the transmitter and receiver descriptor pointer registers. These registers will increase by one if their respective operation is executed (transmit or receive a packet). These registers are 7-bits wide and upon reaching 0x7F, a subsequent DMA operation will set these registers to 0x00.

These 7-bits registers format can pose an issue on memory scarcity system. If every data packet buffer size is set to the MTU (maximum transfer unit) size, then the memory requirement for this scheme is as follows:

Destination MAC address = 6 bytes
Source MAC address = 6 bytes
Data Size = 2 bytes
Descriptor (word 0/1) = 4 bytes each

Memory Size = (MTU + Descriptors + 6 + 6 + 2) * 2 * 128
Memory Size = (1500 + 8 +14) * 2 * 128 = 389,632 bytes

Although this simple round robin method does pose an issue of consuming a lot of memory, we can use software to mitigate the memory consumption by using the descriptor WR (Wrap enable) bit to reduce the memory usage.

In this example, we are limiting our memory usage to four set of buffers. We can redirect the descriptor pointer to wrap back to the first buffer when it reaches the fourth buffer using the following method. When a transaction happens on the third transmission, write descriptor with the WR bit set (see section **3.12**). This method will set the descriptor pointer to zero after the fourth buffer is transmitted.

## 3.4 Speed Negotiation

Many PHYs provide an auto-negotiation mode feature. The PHY on our board has this feature; therefore, we are using this feature to negotiate a connection on our behalf as either, a 10Mbits or 100Mbits link.

## 3.5 Set MAC Address

Ethernet connections communicate with one another using the MAC address; without it, the Ethernet is inoperable. A wrong perception to the MAC address is the use of an Internet Protocol (IP) address to supersede the MAC address. An IP address is a data link layer address whereas the Ethernet MAC address is a Physical layer address. With this clarification and the setting of the MAC address, we complete the initialization of the Ethernet MAC.

## 3.6 Ethernet Programming

We learned from the Ethernet Hardware section how the PHY and the MAC are connected and what roles the signals connecting the PHY and the MAC play. We also learned from the Ethernet Initialization section how to initialize the MAC and PHY.

In the following sections, we will provide programming examples to enable the Ethernet module in the UT700 LEON processor. The programming examples are as follows:

- Reset the Ethernet MAC
- Read PHY Address
- Allocate Memory for Descriptors
- Setup PHY, Auto Speed Negotiation
- Setup MAC address
- Transmit and receive data

**Appendix A** shows all the Ethernet MAC Controller registers used in this AN.

## 3.7 Reset the Ethernet MAC

Resetting of the Ethernet MAC is done by asserting the RS bit in the Ethernet Control Register; see **Figure 14.6** of the UT700 Functional Manual. When the MAC reset is completed, RS is de-asserted by the MAC controller.

This step is essential because it disables the MAC and set all the MAC's registers to the factory default.

```
GRETH.ETHCTR.B.RS = 1;                    // reset the Ethernet MAC
```

## 3.8 Read PHY Address

Since the UT700 does not have a build-in PHY, users are advised to refer to the PHY reference manual for correct programming.

After the MAC Controller is reset, read the ETHMDC, Ethernet MDIO Control and Status Register for the PHY address; see **Figure 14.10** of the UT700 Functional Manual.

```
phyAddr = GRETH.ETHMDC.PHY_ADDR;        // get the PHY address
```

## 3.9 Allocate Memory for Descriptors

The GRETH's MAC supports DMA's transactions, which provides fast data transfer between main memory and FIFOs'. The byproduct of this DMA scheme is that all the allocated buffers must be word-aligned.

For this example, we used the "almalloc" function to allocate word-aligned buffers. Also, refer to sections **14.2.5** and **14.2.10** of the UT700 Functional Manual for the correct descriptor structure.

```
struct descriptor
{
        volatile uint32_t ctrl;                 // word 0
        volatile uint32_t addr;                 // word 1
};

Txd = (struct descriptor*) almalloc (1024);    // 128 descriptors for TX
Rxd = (struct descriptor*) almalloc (1024);    // 128 descriptors for RX
```

## 3.10 Setup PHY, Auto Speed Negotiation

The PHY we used in this experiment provides an easy way to set the PHY into auto-speed negotiation by resetting the PHY. Users of this AN are advised to refer to the PHY's reference manual for the correct PHY programming sequence.

```
do {
     tmp = GRETH.ETHMDC.R;          // make sure the PHY is not busy
} while (tmp & GRETH_MII_BUSY);

tmp = ((data&0xFFFF) << 16) | (phyAddr << 11) | ((addr&0x1F) << 6) | 1;
GRETH.ETHMDC.R = tmp;

do {
```

```
    tmp = GRETH.ETHMDC.R;          // make sure the PHY is not busy
} while (tmp & GRETH_MII_BUSY);
```

## 3.11 Setup MAC Address

The MAC's address is a six bytes long address. Appended code shows how to program the MAC's address in the Ethernet MAC's Address MSB and LSB registers.

```
GRETH.MACMSB.R = 0x00001122;          // the upper 16 bits are not used
GRETH.MACLSB.R  = 0x33445566;
```

## 3.12 Transmit and Receive Data

Appended are software examples how to transmit and receive an Ethernet data packet from and to the main memory and MAC's FIFOs respectively.

```
#define GRETH_BD_EN          0x800
#define GRETH_BD_WR          0x1000
#define GRETH_TXEN           0x1
#define GRETH_RXEN           0x2

// Transmit
Txd.addr = (uint32_t) &tbuf;                  // TX buffer
Txd.ctrl  = GRETH_BD_EN | size;               // set descriptor enable and buf size
GRETH. ETHCTR.R = GRETH. ETHCTR.R | GRETH_TXEN;

// Receive
Rxd.addr = (uint32_t) &rbuf;                  // RX buffer
Rxd.ctrl  = GRETH_BD_EN | size;               // set descriptor enable and buf size
GRETH. ETHCTR.R = GRETH. ETHCTR.R | GRETH_RXEN;
```

**NOTE**: Transfer buffer **THEN** reset the descriptor pointer to zero

```
    Txd.ctrl  = GRETH_BD_EN | GRETH_BD_WR | size;
```

The device MAC address and the Ethernet frame are required for Ethernet communication. The Ethernet frame is beyond the scope of this AN. Nonetheless, to make this experiment successful, the first 12 bytes of the buffer must contain the destination and source address and the subsequent two bytes contain the size of the data.

## 4.0 Summary and Conclusion

After going through this AN, the reader should know how to enable the Ethernet MAC Controller and how the descriptors are setup to support DMA transactions between the main memory and the MAC's FIFOs. The reader should also take note that this Ethernet MAC Controller does not support any CRC generation required by any data link layer transport protocol.

For more information about our UT700 LEON 3FT/SPARC™ V8 Microprocessor and other products please visit our website, www.cobham.com/HiRel or email us at info-ams@cobham.com.

# Appendix A

The header files are designed for this application note purpose only.

```
/*****************************************************************************\
* MODULE: Ethernet Media Access Controller (MAC)                              *
\*****************************************************************************/

#define GRETH_ADDR          0x80000E00
struct   GRETH _TAG {
                union {
                        vuint32_t R; // Figure 14.6: Ethernet Control Register
                        struct {
                                vuint32_t ED            :1;         // EDCL Available
                                vuint32_t BS            :3;         // EDCL Buffer Size
                                vuint32_t RES27to08     :20;        //
                                vuint32_t SP            :1;         // Speed
                                vuint32_t RS            :1;         // Reset
                                vuint32_t PM            :1;         // Open Packet Mode
                                vuint32_t FD            :1;         // Full Duplex
                                vuint32_t RI            :1;         // Enable Receiver Interrupts
                                vuint32_t TI            :1;         // Enable Transmitter Interrupts
                                vuint32_t RE            :1;         // Receive Enable
                                vuint32_t TE            :1;         // Transmit Enable
                        } B;
                } ETHCTR;
                union {
                        vuint32_t R; // Figure 14.7: GRETH Status and Interrupt Source Register
                        struct {
                                vuint32_t RES31to08     :24;        //
                                vuint32_t IA            :1;         // Invalid Address
                                vuint32_t TS            :1;         // Too Small
                                vuint32_t TA            :1;         // Transmitter AHB Error
                                vuint32_t RA            :1;         // Receiver AHB Error
                                vuint32_t TI            :1;         // Transmitter Interrupt
                                vuint32_t RI            :1;         // Receiver Interrupt
                                vuint32_t TE            :1;         // Transmitter Error
                                vuint32_t RE            :1;         // Receiver Error
                        } B;
                } ETHSIS;
                union {
                        vuint32_t R; // Figure 14.8: Ethernet MAC Address MSB
                        struct {
                                vuint32_t RES31to16     :16;        //
                                vuint32_t ADDR_MSB      :16;        // MAC MSB address
                        } B;
                } MACMSB;
                union {
                        vuint32_t R; // Figure 14.9: Ethernet MAC Address LSB
                        struct {
                                vuint32_t ADDR_LSB      :32;        // MAC LSB address
                        } B;
                } MACLSB;
                union {
                        vuint32_t R; // Figure 14.10: Ethernet MDIO Control and Status Register
```

```
		struct {
			vuint32_t DATA		:24;	// Rx or Tx Data
			vuint32_t PHY_ADDR	:5;		// address of the PHY
			vuint32_t REG_ADDR	:1;		// address of the register
			vuint32_t RES5		:1;		//
			vuint32_t NV		:1;		// Not Valid
			vuint32_t BU		:1;		// Busy
			vuint32_t LF		:1;		// Link Fail
			vuint32_t RD		:1;		// Read
			vuint32_t WR		:1;		// Write

		} B;
	} ETHMDC;
	union {
		vuint32_t R; // Figure 14.11: Ethernet Transmitter Descriptor Pointer Register
		struct {
			vuint32_t TXDTRA	:22;		// Transmitter Descriptor Table addr
			vuint32_t TX_DESC	:7;		// descriptor incremental field
			vuint32_t RES2to0	:3;		//
		} B;
	} ETHTDP;
	union {
		vuint32_t R; // Figure 14.12: Ethernet Receiver Descriptor Pointer Register
		struct {
			vuint32_t RXDTRA	:22;		// Receiver Descriptor Table addr
			vuint32_t R`X_DESC	:7;		// descriptor incremental field
			vuint32_t RES2to0	:3;		//
		} B;
	} ETHRDP;
};
```

## REVISION HISTORY

| Date | Rev. # | Author | Change Description |
|---|---|---|---|
| 11/07/2016 | 1.0.0 | MTS | Initial Release |
| 1/12/2017 | 1.0.1 | MTS | Added a note in page 1 |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

*Cobham Semiconductor Solutions*

This product is controlled for export under the U.S. Department of Commerce (DoC). A license may be required prior to the export of this product from the United States.

Cobham Semiconductor Solutions
4350 Centennial Blvd
Colorado Springs, CO 80907

E: info-ams@aeroflex.com
T: 800 645 8862