.

HALCoGen Ethernet Driver and lwIP Integration Demonstration

Contents

Introduction

The Ethernet Media Access Controller (EMAC) and Management Data Input/Output (MDIO) peripherals on the Hercules line of devices provide a full-featured Ethernet interface. The EMAC peripheral conforms to the IEEE 802.3 standard, describing the Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer specifications. The EMAC module provides an efficient interface between the processor and a local network. The EMAC supports 10Base-T (10 Mbps) and 100BaseTX (100 Mbps), in half-duplex and full-duplex modes. The EMAC control module provides an interface from the CPU to the EMAC and MDIO modules. The EMAC control module controls device interrupts and incorporates an 8k-byte internal RAM to hold EMAC buffer descriptors (also known as CPPI RAM). The MDIO module implements the 802.3 serial management interface to interrogate and control up to 32 Ethernet PHYs connected to the device by using a shared two-wire bus. Applications can use the MDIO module to configure the auto negotiation parameters of each PHY attached to the EMAC, retrieve the negotiation results, and configure required parameters in the EMAC module for correct operation.

The driver code is part of the HALCoGen release and APIs for EMAC and MDIO are listed in *include/emac.h* and *include/mdio.h*, respectively.

Supported Features

- Integration of CPDMA based EMAC driver for RM48xx/RM46xx/TMS570xxx/RM57xx/TMS570LCxx
- lwIP 1.4.1 TCP/IP stack ported for the above devices' EMAC driver
- By default the DHCP support is enabled. However the integration supports both DHCP & Static IP addressing.
- Integration with HALCoGen v4.00.00 release
- By default the software is configured for executing from flash.
- The integrated application demonstrates a webserver application on TMS570xx/TMS570LCxx/RM48xx/RM57xx device.
- Diagnostic & Debug messages are printed on JTAG SCI Port. Following are the settings for the console:

  Baud Rate: 9600

  Data: 8 bit

  Parity: None

  Stop bit: 2

  Flow Control: None

Note: Since the MAC Address is part of the binary image, all devices programmed with these binaries & connected to same DHCP server will be assigned same IP address. The default MAC address is 00:08:EE:03:A6:6C

Configuring EMAC and MDIO using HALCoGen GUI

· The following sequence is to be followed to get working driver code for EMAC and MDIO modules using HALCoGen. A working implementation of this with lwIP is available at the end of this page.

1. Under the 'Driver Enable' tab, enable EMAC Driver and SCI2 Driver.
2. Under 'VIM RAM' add the names of the ISRs for EMAC Transmit and Receive Interrupts (Channels 77 and 79 respectively).
3. Enable these interrupts under the 'VIM Channel 64-95' tab.
4. Under the 'PLL' tab, change the multiplier for both PLLs to a value of 120, such that the output frequency in both cases is 160.00 MHz.
5. Under the 'GCM' tab, change the value of the VCLKA4 Divider to 2, such that the output of VCLKA4(or VCLKA4_DIVR_EMAC in case of RM46x/TMS570LS12x Devices) is 40.00 MHz.
6. Under the 'PINMUX' tab, enable RMII or MII(depending on the board that you are using - Most HDKs support MII by default, while controlCARDs are designed for RMII), MDIO(G3) & MDCLK(V5).
7. Under the 'EMAC' tab, change the EMAC address to 00:08:EE:03:A6:6C, which is the default MAC Address. Change the physical address to 1.
8. Generate the system initialization and HAL Code.

For TMS570LC43x and RM57x devices:

1. Under the 'Driver Enable' tab, enable EMAC Driver and SCI1 Driver.
2. Under 'VIM RAM' add the names of the ISRs for EMAC Transmit and Receive Interrupts (Channels 77 and 79 respectively).
3. Enable these interrupts under the 'VIM Channel 64-95' tab.
4. Under the 'PLL' tab, change the multiplier for both PLLs to a value of 150, such that the output frequency in both cases is 300.00 MHz.
5. Under the 'GCM' tab, change the value of the VCLK1, VCLK2 and VCLK3 Dividers to 1 and VCLKA4 Divider to 2, such that the output of VCLKA4_DIV is 37.50 MHz.
6. Under the 'PINMUX' tab, enable RMII/MII, under Pin Muxing. Under Input Muxing, enable MDIO(G3), MII_COL(F3), MII_CRS(B4), MII_RX_DV(B11), MII_RX_ER(N19), MII_RXCLK(K19), MII_RXD[0], MII_RXD[1], MII_RXD[2], MII_RXD[3], MII_TX_CLK.
7. Under the 'EMAC' tab, change the EMAC address to the correct address (the default one in the example is mentioned above). The physical address is 1 by default.
8. Generate the system initialization and HAL Code.

Programming Sequence using HALCoGen generated drivers

Usng the HALCoGen generated driver code (through the procedure above), the following sequence can be used to configure and operate the EMAC and MDIO modules.

1. Initialize the EMAC module by calling **EMACInit()**. This API resets the EMAC and EMAC Control Module Registers.
2. Initialize the MDIO Module using **MDIOInit()**. Insert a short delay after this function returns to ensure that MDIO module initialization completes successfully before using other MDIO APIs.
3. Auto negotiate with the PHY device connected to MDIO. PHY Auto negotiation APIs are provided as reference and must be adapted to the external PHY present on the hardware.
4. After completing auto negotiation, get the auto negotiation result using the respective PHY's link partner ability API and set the duplex mode of operation in the EMAC using **EMACDuplexSet()**.
5. Set the MAC Address in the EMAC hardware using **EMACMACAddrSet()**
6. Enable unicast for a specific channel using the **EMACRxUnicastSet()** API (optional).
7. Initialize the TX and RX buffer descriptors in the CPPI RAM, which is local to the EMAC.
8. Enable the TX operation in EMAC using **EMACTxEnable()**. This enables the EMAC hardware transmit operation. However, transmission will not start until a valid descriptor pointer is written using EMACTxHdrDescPtrWrite().
9. Enable the RX operation in EMAC using **EMACRXEnable()**.
10. Write the RX Header Descriptor Pointer using **EMACRxHdrDescPtrWrite()**. The EMAC hardware will start receiving data at this point. The data will be stored to the buffer pointer in this buffer descriptor. After the buffer corresponding to this descriptor is filled, the next descriptor is used by the EMAC hardware according to the buffer descriptor settings.
11. Enable MII using **EMACMIIEnable()**.
12. Enable the Transmit and Receive Pulse interrupts using **EMACTxIntPulseEnable()** and **EMACRxIntPulseEnable()**. The interrupts will be routed through the EMAC Control Core to the CPU interrupt controller. This enables the EMAC TX and RX pulse interrupts at EMAC peripheral level only. The core interrupts must be enabled separately in the VIM.

The following guidelines should be observed when writing an EMAC interrupt service routine (ISR):

- In an EMAC Transmit ISR, the interrupt must be acknowledged to the EMAC hardware using **EMACCoreIntAck()**. However, the interrupt will be cleared only if the completion pointer is written using the EMACTxCPWrite() API with the last processed TX buffer descriptor.
- In an EMAC Receive ISR, the interrupt must be acknowledged to the EMAC hardware using **EMACCoreIntAck()**. Again, the interrupt will be cleared only if the completion pointer is written using the EMACRxCPWrite() API with the last processed RX buffer descriptor.

Design of LwIP Integration
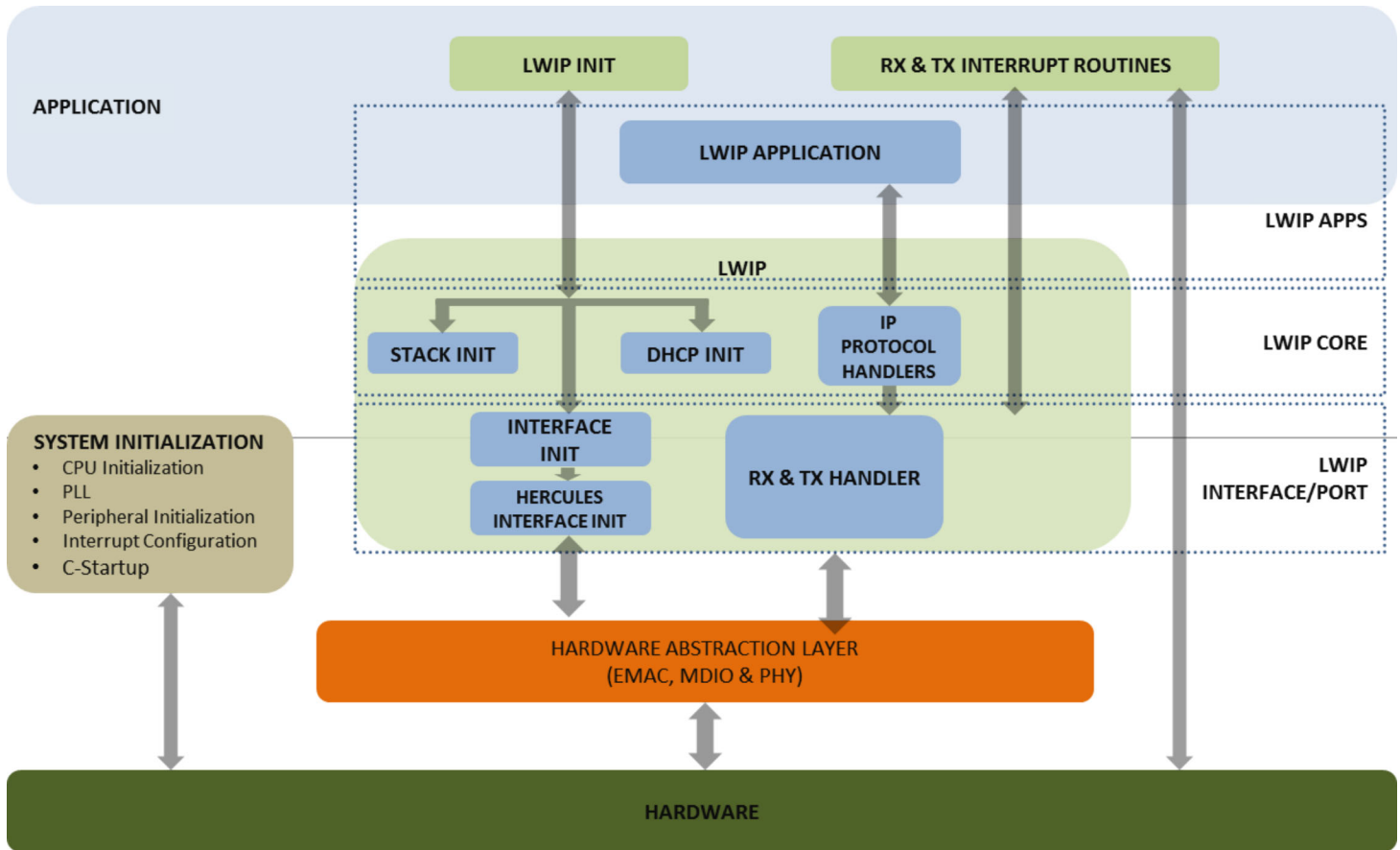
The integrated software deliverable consists of four main layers:

1. Hardware Abstraction Layer (HAL): EMAC & MDIO HALs are part of the HALCoGen release and the PHY HAL is part of the application.
2. lwIP Network Interface Layer – Hercules NetIF port for LwIP
3. IWIP Application Layer – An IP stack application based on LwIP. Examples are provided for HTTP server, UDP based client and echo server. The packets start and end at this layer.
4. System Application Layer – This includes the system initialization and is generated based on the HALCoGen GUI. This layer configures the PLLs & PINMUX.

Hardware Abstraction Layer

This layer implements the lowers level hardware abstraction APIs that can be used for control and configuration of the EMAC device.

- **emac.c** – Ethernet MAC and Control module. This file is generated using HALCoGen. In this release the file is located at *<Device>\HALCoGen-xx\source\emac.c*, where xx is the device variant.
- **mdio.c** – MDIO interface between the PHY and MAC. This file is generated using HALCoGen. In this release the file is located at *<Device>\HALCoGen-xx\source\emac.c*
- **phy_dp83640.c** – Example PHY Device HAL for DP83640 PHY on HDK. This file should be written for the external Ethernet PHY on the hardware.

lwIP Interface Layer

To interface with the rest of the network, the device abstraction layer needs to be glued with a network stack that can form and interpret network packets. The device abstraction hooks into the interface layer of lwIP. This is also referred to as the device-specific "port" or the hdk-interface for lwIP. It defines standard interface entry points and state variables. A network device is represented by struct netif, generically referred to as netif. The netif contains all the information about the interface, including the IP/MAC address(s), TCP/IP options, protocol handlers, link information, and (most importantly) the network device driver entry point callbacks. Every network interface must implement the linkoutput and init callbacks, and all state information is maintained in this structure. The interface layer also implements the core interrupt handling and DMA handling. All the required function calls for initializing the lwIP stack and registering the network interface are performed in *lwip-1.3.2\ports\hdk\lwiplib.c\lwiplib.c*. Refer to the lwIP documentation for more information about the lwIP stack implementation.

Hercules Development Network Interface Layer

· The main tasks of the HDK network interface layer are:

Network device initialization

The first step towards bringing up the interface is done as part of the **hdkif_init()**. This function is called when the network device is registered with the lwIP stack using netif_add. As part of the initialization, the netif output callbacks are registered and hardware initialization, including PHY and DMA initialization, is performed. DMA buffer descriptor (BD) pools are maintained in the CPPI RAM for both TX and RX channels. The descriptor chains are maintained by the "free_head", which points to the next unused/free descriptor in the BD pool, and "active_tail", which points to the last BD in the active queue that has been en-queued to the hardware. The packet buffers (pbuf) are pre-allocated for maximum length and queued in the receive buffer descriptors before the reception begins. Please refer to the lwIP documentation for details on pbuf handling by lwIP.

Packet data transmission

Packet data transmission takes place inside the linkoutput callback registered with the lwIP stack. This callback is invoked whenever the lwIP stack receives a packet for transmission from the application layer. The pbuf can contain a chain of packet buffers and hence the DMA descriptors are properly updated (chained if necessary), with SOP, EOP and length fields. The first DMA descriptor is marked with the EOP and OWNER flags, while only the last is set with the EOP flag. After filling the BD's with the pbuf information, the BD, which corresponds to the SOP is written to the HEAD descriptor pointer register to start the transmission. Once a packet is transmitted, the EMAC Control Core generates a transmit interrupt. This interrupt is cleared only if the completion pointer is written with the last BD processed. In the interrupt handler, the next BD to process is taken and traversed to reach the BD that corresponds to the end of the packet. This BD, which corresponds to the end of the packet, is written to the completion pointer. After this, the pbuf that corresponds to this packet is freed. Thus it is made sure that the freeing of pbuf is done only after the packet transmission is complete.

Packet data reception

Packet reception takes place in the context of the interrupt handler for receive. As described earlier, the receive buffer descriptors are en-queued to the DMA before the reception can actually begin. The pbuf allocated for maximum length, may actually contain a chain of packet buffers. The descriptors are updated for OWNER flag only. The EOP, SOP and EOQ are updated by the DMA upon completion of the reception. One important point to note is that, the actual data received may be less/more than the max length allocated. Hence the pbuf chain needs to be adjusted as detailed here. First, the active_head (which is the first BD en-queued), is checked for OWNER bit having been cleared by the DMA. Then the BD list is traversed, starting at the active_head, to find the EOP BD, which is the last BD of the current packet. While doing so if the EOP is not found on the current BD, then the pbuf of the current BD is chained to the pbuf of the next BD, since the current packet has spilled over to the next BD. Once, the EOP is found the last pbuf is updated as the terminator (pbuf->next = NULL). Thus, the entire packet is collected and passed to the upper layer for processing. Since, the current BD has been done with, it is put back at the free_head, by allocating a new pbuf.

lwIP Application Layer

This layer contains the Ethernet application (HTTP server, echo server, etc.). This is located at lwip-1.4,1/apps/<application>. This is the layer at which all the imcoming packets terminate and all outgoing packets originate. This release contains a *httpserver_raw* application that runs a sample webserver on the device.
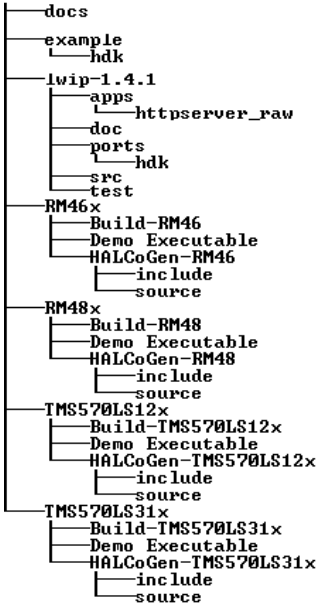
System Application Layer

This layer implements system level initialization and provides options for lwIP stack. This layer can contain any other algorithms, decoding, etc. The main IP stack based application is part of the lwip directory as mentioned above.

Release Folder Structure

There are 6 different variants supported. The folder for each variant contains the following:

1. Build Folder: This contains the Code Composer Studio project which can be imported into your workspace and built.
2. HALCoGen Folder: This folder contains the source and include folders generated using the latest version on HALCoGen.
3. Demo Executable: This folder contains a .out file which can be executed on the device HDK that is being used for development.

· <Install Directory>
```
├──docs
├──example
│   └──hdk
├──lwip-1.4.1
│   ├──apps
│   │   └──httpserver_raw
│   ├──doc
│   ├──ports
│   │   └──hdk
│   ├──src
│   └──test
├──RM46x
│   ├──Build-RM46
│   ├──Demo Executable
│   └──HALCoGen-RM46
│       ├──include
│       └──source
├──RM48x
│   ├──Build-RM48
│   ├──Demo Executable
│   └──HALCoGen-RM48
│       ├──include
│       └──source
├──TMS570LS12x
│   ├──Build-TMS570LS12x
│   ├──Demo Executable
│   └──HALCoGen-TMS570LS12x
│       ├──include
│       └──source
└──TMS570LS31x
    ├──Build-TMS570LS31x
    ├──Demo Executable
    └──HALCoGen-TMS570LS31x
        ├──include
        └──source
```

Note:

- The '**Build-xx**' folder contains the CCSv5 project for the device.
- The '**HALCoGen-xx**' folder contains the HALCoGen generated driver files for the device.
- The '**Demo Executable**' Folder contains the .out file for the sample project for the device, which can be loaded and executed.
- The '**example**' folder contains a HDK port of the lwIP example.
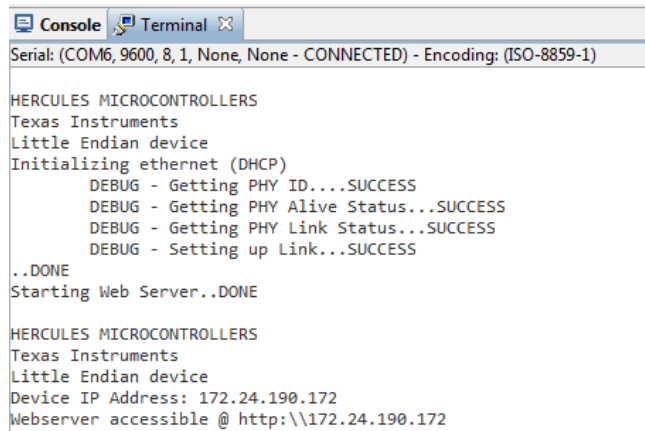- '**lwip-1.4.1**' contains all the lwIP library files.

Hardware Setup

- Connect ethernet port on HDK to network with DHCP Server
- Enable the Ethernet on EVM by setting switch S2 bit4 to On
- Connect the JTAG USB port to PC (used for console output also)

Building & Executing

- Release folders contains CCS Projects for RM48xx and RM46xx(Little Endian) & TMS570xx Devices (Big Endian).
- Based on the device import the project into CCS from *<install_folder>\<device line>*
- Build the project
- Target configuration, is supplied with this software. Set the correct target configuration as the Default & Active configuration
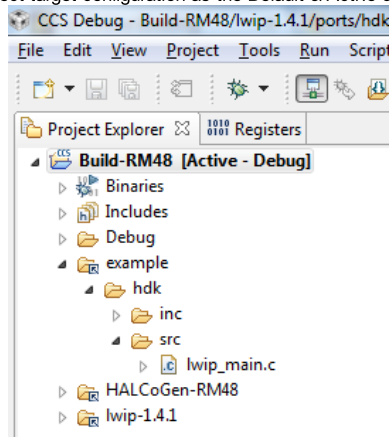
- Download and execute the binary (Disconnect the device from debugger and reset). Below is an screen capture of console output:

```
Console  Terminal  ⊠
Serial: (COM6, 9600, 8, 1, None, None - CONNECTED) - Encoding: (ISO-8859-1)


HERCULES MICROCONTROLLERS
Texas Instruments
Little Endian device
Initializing ethernet (DHCP)
          DEBUG - Getting PHY ID....SUCCESS
          DEBUG - Getting PHY Alive Status...SUCCESS
          DEBUG - Getting PHY Link Status...SUCCESS
          DEBUG - Setting up Link...SUCCESS
..DONE
Starting Web Server..DONE

HERCULES MICROCONTROLLERS
Texas Instruments
Little Endian device
Device IP Address: 172.24.190.172
Webserver accessible @ http:\\172.24.190.172
```

The webserver is accessible at the IP address displayed in the console.

Note:

The RM46xx project enables RMII by default, which works for the RM46 controlCARDs. If you're using an RM46 HDK, however, you will need to change this setting to MII(in the PINMUX tab), and then rebuild the project.

Version 00.03.00 (supports TMS570LC43xx and RM57x devices):

lwIP_Demo_v03
http://software-dl.ti.com/hercules/hercules_public_sw/HALCoGen_EMAC_LWIP-00.03.00-installer.exe

Known Issues
For version 0.0.3, problems have been reported when using RMII mode (especially on the CNCD boards). The issue is that the RMIISPEED bit of the MACCONTROL register is left at the default 10MBPs while most times the phy will autonegotiate to 100MBPs. When the phy and mac mismatch, it appears the the MAC is transmitting (but not receiving) however no packets will be observed coming from the phy.

Patch here: File:Lwip 0 0 3 patch rm46.zip

To use the patch, you need GNU 'patch' - http://sourceforge.net/projects/mingw/files/Installer/ was used to install MSYS diff/patch. To actually apply the patch:

Open an MSYS shell
Change directories to /C/TI/Hercules (default install) or where ever the parent folder of "HALCoGen EMAC Driver with lwIP Demonstration/v00.03.00/" is on your machine.
patch -p3 -i <pathtopatchfile>/lwip_0_0_3_patch_rm46.patch
You should see: patching file 'HALCoGen EMAC Driver with lwIP Demonstration/v00.03.00/RM46x/HALCoGen-RM46/include/phy_dp83640.h' patching file 'HALCoGen EMAC Driver with lwIP Demonstration/v00.03.00/RM46x/HALCoGen-RM46/source/phy_dp83640.c' patching file 'HALCoGen EMAC Driver with lwIP Demonstration/v00.03.00/lwip-1.4.1/ports/hdk/netif/hdkif.c'

if not try again..

The same patches to phy_dp83640.h and phy_83640.c need to be applied to other subdirectores (e.x. RM48x) when RMII is needed. But all projects use the common 'hdkif.c' file in the lwip-1.4.1/ ports/ folder so this file only needs to be patched a single time.