Application Report
SPNA239 – September 2019

HALCoGen Ethernet Driver With lwIP Integration Demo and Active Web Server Demo

Eric Ding

ABSTRACT

This application report illustrates the use of the Ethernet driver with two demonstrations, both with lightweight IP (lwIP) integrated, which is a widely used open-source TCP/IP stack designed for embedded systems. The demos run on TI Hercules™ Arm® Cortex®-R MCUs for functional safety. The driver code generation with the TI HALCoGen tool, the port of lwIP and the design of lwIP integration are explained. Finally, the test setup, build process and results are provided in the document.

Contents

List of Figures

List of Tables

Trademarks

Hercules, Code Composer Studio are trademarks of Texas Instruments.
Arm, Cortex are registered trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere.
All other trademarks are the property of their respective owners.

1 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 8KB internal RAM to hold EMAC buffer descriptors, which is also known as Communications Port Programming Interface (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 EMAC 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.

2 Supported Features

- Integration of the CPDMA based EMAC driver for RM46/RM48/RM57/TMS570LS/TMS570LC devices
- lwIP (lightweight IP) 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 and Static IP addressing.
- Integration with HALCoGen v04.07.01 release
- The integrated application demonstrates web server applications on RM46/RM48/RM57/TMS570LS/TMS570LC devices.
- By default, the software is configured for executing from flash.
- Diagnostic and debug messages are printed on the JTAG SCI Port. The 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 of the devices programmed with these binaries and connected to the same DHCP server will be assigned the same IP address. The default MAC address is 00:08:EE:03:A6:6C.

3 Get the Software

The latest lwIP Demo software version 04.00.00 can be downloaded from: http://git.ti.com/hercules_examples/hercules_examples/trees/master/Application/LwIP. The previous version 00.03.00 can be downloaded from: http://software-dl.ti.com/hercules/hercules_public_sw/HALCoGen_EMAC_lwIP-00.03.00-installer.exe.

Compared to 00.03.00 version, the new version has the following updates:

- Code generated with HALCoGen release 04.07
- Added LAUNCHXL2 570LC43x and LAUNCHXL2 RM57x support

The latest active web server demo software version 1.1.0 can be downloaded from: http://git.ti.com/hercules_examples/hercules_examples/trees/master/Application/ActiveWebserver. Compared to the previous 1.0.0 version, the new one updated with HALCoGen release 04.07.01 supports LAUNCHXL2 570LC43x and LAUNCHXL2 RM57x.

HALCoGen can be downloaded from: http://www.ti.com/tool/HALCOGEN.

TI ARM compiler 18.2.2 LTS is used to build the code that can be downloaded from: http://www.ti.com/tool/download/ARM-CGT-18/18.12.2.LTS.

TI Code Composer Studio™ (CCS) 8.3 is used to import, build and run the project, which can be downloaded from: http://software-dl.ti.com/ccs/esd/documents/ccs_downloads.html.

4 Configuring EMAC and MDIO Using HALCoGen GUI for the lwIP Demo

NOTE: The lwIP demo software package includes all the CCS projects and source code for various Hercules platforms. See the installation for more details. Those projects can be imported into CCS and built to run as it is.

The following sequence explains how to get the working driver code for the EMAC and MDIO modules using HALCoGen for the lwIP demo, which is a static web server demo. This is for informative purposes only.

4.1 RM46x, RM48x and TMS570LSx HDK

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 control CARDs are designed for RMII), MDIO(G3) and 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.

4.2 TMS570LC43x and RM57x HDK

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.

4.3 RM57x Launchpad (LAUNCHXL2 RM57x)

1. Navigate to the folder '<install_dir>'. Copy the RM57x folder to a folder named '<install_dir>/LAUNCHXL2-RM57'.
2. Start a HALCoGen session.
3. From HALCoGen, open the project '<install_dir>/LAUNCHXL2-RM57/HALCoGen-RM57x/HALCoGen-RM57x.hcg'.
4. Make the following changes to the HALCoGen Project:
    a. RM57L843ZWT/Driver Enable Tab. Enable the GIO Driver:
        i. Check "Enable GIO Driver".
        ii. Confirm GIO, SCI1, and EMAC drivers are enabled.
    b. RM57L843ZWT/ECLK Tab. Provide a 25 MHz Clock to the PHY:
        i. In the ECLK Pin Mode Group, change the ECLK pin Mode to ECLK.
        ii. In the ECLK pin Group, make sure DIR is checked.
        iii. In the ECLK Functional Configuration group, change Divider to 3 so that ECPCLK is 25 MHz.
        iv. Also check the "Continue on suspend" button.
    c. PINMUX/Pin Muxing Tab. Change the MII and MDIO interface pins to their default locations:
        i. Make sure MII is checked in the Enable/Disable Peripherals Group at the top of the PINMUX. This is necessary to put the MAC in MII mode, even though it will also move the MII pins to their alternate location, which have to be undone manually.
        ii. Uncheck the MII and MDIO signals on Balls A14, B4, B11, D19, E18, F3, G3, G19, H18, H19, J18, J19, K19, N19, P1, R2 and V5. These rows should now be blank, although you can select non-Ethernet functions, if desired.
        iii. Change the selection on balls T4, U7 to the default functions (from MII_RX_AVCLK4 to MII_RXCLK, and from MII_TX_AVCLK4 to MII_TX_CLK). The PHY will provide these clocks to the MAC on the launchpad.

NOTE: List Conflicts show total conflicts 2 for Ball T4 and U7 - just ignore this.

    d. PINMUX/Input Pin Muxing Tab. Change all of the input MII and MDIO signals to the Default (left Column) states:
        i. MDIO=F4, MII_COL=W4, MII_CRS=V4, MII_RX_DV=U6, MII_RX_ER=U5, MII_RXCLK=T4, MII_RXD[0]=U4, MII_RXD[1]=T3, MII_RXD[2]=U3, MII_RXD[3]=V3, MII_TX_CLK=U7
    e. GIO/Port A Tab. GIOA[3] and GIOA[4] need to be driven high, to release the PHY from reset and power down.
        i. Check the "DIR" box for Bit 3 and Bit 4.
        ii. Change DOUT to '1' for Bit 3 and Bit 4.
    f. Save your modified HALCoGen Project.
    g. Press "F5" or Choose File → Generate Code from The HALCoGen menu.
    h. Exit HALCoGen.
5. Launch Code Composer Studio.
6. Import the project you just modified:
    a. From the Project Menu, choose "Import CCS Projects".
    b. Select 'search-directory'.
    c. Browse to the folder '<install_dir>'.
    d. From the list of projects available to import - select "Build-RM57x".

7. Make a copy of the original project, naming it "Build-LAUNCHXL2-RM57x":
   a. In CCS's Project Explorer, select the project "Build-RM57x".
   b. Use CTRL-C, CTRL-V or the context menu to copy/paste the project.
   c. For the new project, use the name "Build-LAUNCHXL2-RM57x". You can create a new folder '<install_dir>/LAUNCHXL2-RM57/Build-LAUNCHXL2-RM57x' for this.
8. Change the HALCoGen Project included by your new project:
   a. Select your new project, 'Build-LAUNCHXL2-RM57x', in CCS's Project Explorer.
   b. If not expanded, expand the tree under this project.
   c. Select the folder "HALCoGen-RM57x". The folder icon should indicate that it is a linked resource.
   d. From the context-menu, select "Properties" for the folder "HALCoGen-RM57x".
   e. With "Resource" selected, press "Edit" and change the location to: PROJECT_LOC\..\HALCoGen-RM57x.
   f. Make sure the Resolved Location displayed matches the HALCoGen project that you edited in step 7.
   g. You can do the same steps to resolve the locations for the "example" and "lwIP-1.4.1" folders if your CCS project was created in a different location.
9. Change the PHY Id to match the DP83630 Precision PHYTER on the Launchpad:
   a. From your "Build-LAUNCHXL2-RM57L" CCS project, navigate to and open for editing "HALCoGen-RM57x\include\HL_phy_dp83640.h".
   b. Change the last "USER CODE" block in the header file, so that it reads:

```
/* USER CODE BEGIN (2) */
/* @todo @fixme:  This is a dirty hack, but it minimizes changes for now */
#undef DP83640_PHY_ID
#define DP83640_PHY_ID  (0x20005CE1u)
/* USER CODE END */
```

10. Add Code to initialize GIOA[3] and GIOA[4] (To release the PHY):
    a. From your "Build-LAUNCHXL2-RM57L" CCS project, navigate to and open for editing "HALCoGen-RM57x\source\HL_sys_main.c".
    b. Add an include directive for "HL_gio.h", so that the first USER CODE block reads:

```
/* USER CODE BEGIN (1) */
#include "HL_gio.h"
extern void EMAC_lwIP_Main (uint8_t * emacAddress);
/* USER CODE END */
```

    c. Add a call to gioInit() from main() so that the third user block reads:

```
/* USER CODE BEGIN (3) */
gioInit();
EMAC_lwIP_Main(emacAddress);
/* USER CODE END */
```

11. Update PINMUX file:
    a. From your "Build-LAUNCHXL2-RM57L" CCS project, navigate to and open for editing "HALCoGen-RM57x\source\HL_pinmux.c".
    b. Change PINMUX_BALL_R4_ to PINMUX_BALL_R4_GIOB_3

```
pinMuxReg->PINMUX[9] = PINMUX_BALL_R4_GIOB_3 |
PINMUX_BALL_N17_EMIF_nCS_0 | PINMUX_BALL_L17_EMIF_nCS_2;
```

12. Rebuild your project.
13. Load your project onto the Launchpad and test it out.

4.4 TMS570LC43 Launchpad (LAUNCHXL2 570LC43x)

The instructions discussed in the previous RM57x Launchpad section apply to port lwIP demo on TMS570LC43, except copying '<install_dir>/ TMS570LC43x' as the starting point.

5 Additional Changes for Active Web Server Demo

Modifications have been done to the lwIP demo to support an active web server, which can run on LAUNCHXL2 570LC43x and LAUNCHXL2 RM57x. This active web server demo tries to showcase running on the LAUNCHXL2 570LC43x and LAUNCHXL2 RM57x Launchpad. The web server implementation supports SSI and CGI calls. Using this demo, you can toggle the LED and send text to a serial terminal connected to the com port on the LaunchPad.

NOTE: The active web server demo software package includes all the CCS projects and source code for two Hercules platforms. For details, check the installation folder. Those projects can be imported into CCS and built to run as it is. This demo currently supports only LAUNCHXL2 570LC43 and LAUNCHXL2 RM57x. The port to the other platforms should be straight forward and mainly needs changes in the HALCoGen configuration in similar lines to the HALCoGen configurations created for LAUNCHXL2 570LC43, LAUNCHXL2 RM57x in this active web server demo. The following sequence explains how to get working driver code for EMAC and MDIO modules using HALCoGen configuration and the lwIP port. This is for informative purpose only.

5.1 HALCoGen Configuration Changes

The sections below describe the additional configuration changes made to the HALCoGen in addition to changes mentioned in the lwIP demo.

- Driver Enable tab changes: Enable GIO and Enable RTI driver
- Pinmux changes: Enable GIO
- Interrupts: Enable interrupts for the RTI compare 0 and RTI compare 2
- GIO changes: Enable output direction for GIOB pin 6 and 7.THE GIO is used for the user LED 2 and LED 3.
- RTI changes: Set RTI compare 0 for 10 ms and RTI compare 2 for 500 ms. drive both compares by counter 0.The RTI compare 0 is used for servicing the lwIP timers and the RTI Compare 2 is used for the blinking animation LED LED3.

5.2 lwIP Port Changes

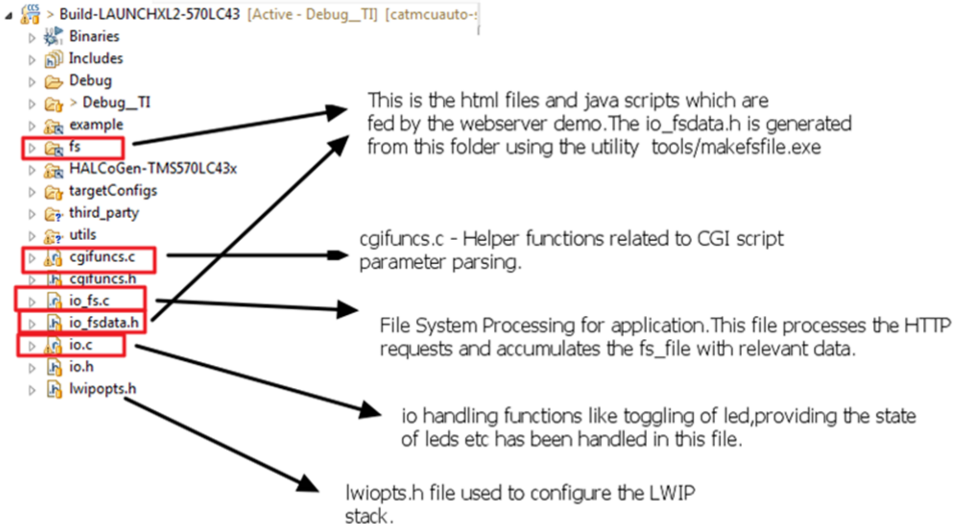
The port now services the lwIP timers needed (the demo should work even without this). A bug in the transmission of the packets smaller than 60 bytes has been fixed in the port. Some additional guard code has been added to port. The changes are done mainly in the files listed in Table 1.

Table 1. lwIP Port Changes

File	Change
\third_party\lwip-1.4.1\ports\hdk\netif\hdkif.c	Added guard conditions, fixed a bug in the hdkif_output()
\utils\lwiplib.c	Added servicing of lwIP timers
\TMS570LC43x\Build-LAUNCHXL2-570LC43\lwipopts.h	Configuration changes for the lwIP stack
\RM57x\Build-LAUNCHXL2-RM57x\lwipopts.h	Configuration changes for the lwIP stack
\example\hdk\src\lwip_main.c	Initialization of application and control path for the LwIP and EMAC driver. Added API to handle CGI and SSI calls.

5.3 CCS Project Structure

Figure 1 shows the CCS project structure and explains about the important files.



This is the html files and java scripts which are fed by the webserver demo.The io_fsdata.h is generated from this folder using the utility tools/makefsfile.exe

cgifuncs.c - Helper functions related to CGI script parameter parsing.

File System Processing for application.This file processes the HTTP requests and accumulates the fs_file with relevant data.

io handling functions like toggling of led,providing the state of leds etc has been handled in this file.

lwiopts.h file used to configure the LWIP stack.

Figure 1. Active Web Server Demo CCS Project Structure

5.4 Changing the Web Pages Rendered by Web Server

As indicated in the previous section folder, /fs (file system that is rendered) contains the present html files and java scripts that are being rendered by the active web server. The contents of the folder can be converted to an .h file containing various data structures as in io_fsdata.h using the utility in tools/makefsfile.exe as shown below. (There is also a perl script available at location third_party\lwip-1.4.1\apps\httpserver_raw\makefsdata. You can read through the script to get a better understanding of the utility).

makefsfile.exe -i fs -o io_fsdata.h -r -h -q, where:

- fs is the folder containing the file system (html files, java scripts, images, and so forth) that is being rendered by the web server.
- io_fsdata.h is the .h file generated.

6 Programming Sequence Using HALCoGen Generated Drivers

Using 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 is 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.

7 Design of lwIP Integration

The integrated software deliverable consists of four main layers:

- Hardware Abstraction Layer (HAL): EMAC and MDIO HALs are part of the HALCoGen release and the PHY HAL is part of the application.
- lwIP Network Interface Layer – Hercules NetIF port for lwIP
- lwIP 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.
- System Application Layer – This includes the system initialization and is generated based on the HALCoGen GUI. This layer configures the PLLs and PINMUX.

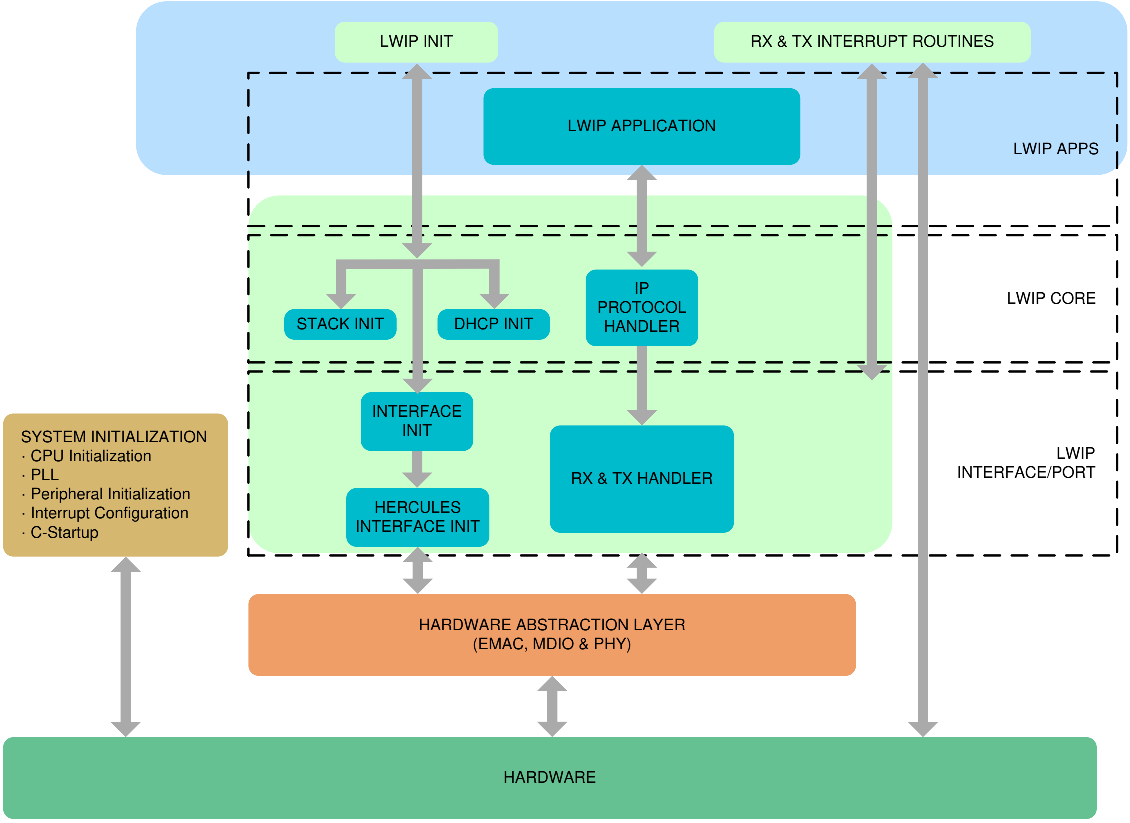


Figure 2. Software Layers With lwIP Integration

7.1 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.

7.2 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-1.4.1\ports\hdk\lwIPlib.c\lwIPlib.c. For more information about the lwIP stack implementation, see the lwIP documentation under lwip-1.4.1\doc.

7.3 Hercules Development Network Interface Layer

The main tasks of the HDK network interface layer are discussed in the following sections.

7.3.1 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. For details on pbuf handling by lwIP, see the lwIP documentation under lwip-1.4.1\doc.

7.3.2 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.

7.3.3 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.

7.4 lwIP Application Layer

This layer contains the Ethernet application (HTTP server, echo server, and so forth). This is located at lwIP-1.4.1/apps/<application>. This is the layer at which all the incoming packets terminate and all outgoing packets originate. This release contains a httpserver_raw application that runs a sample web server on the device.

7.5 System Application Layer

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

8 Release Folder Structure

The lwIP demo software supports eight different variants. The folder for each variant contains the following:

- Build folder: This contains the CCS project that can be imported into your workspace and built.
- HALCoGen folder: This folder contains the source and include folders generated using the latest version on HALCoGen.

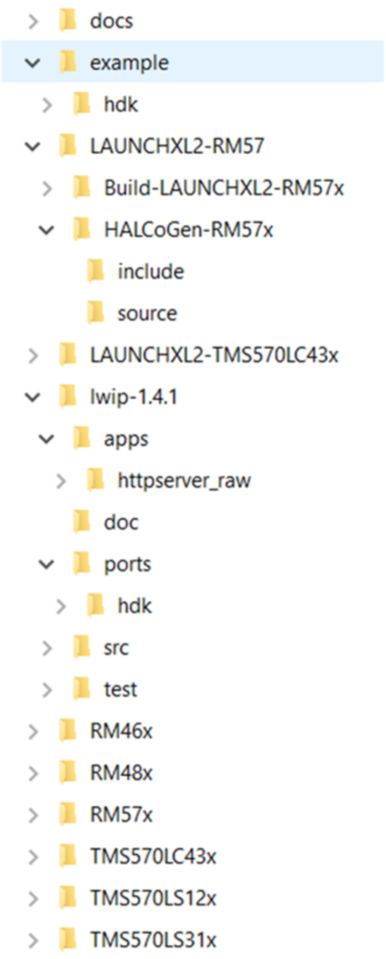


Figure 3. lwIP Demo Software Folder Structure

NOTE:

- The 'Build-xx' folder contains the CCS project for the device.
- The 'HALCoGen-xx' folder contains the HALCoGen generated driver files for the device.
- The 'doc' folder contains manifest and user's guide.
- The 'example' folder contains a port of the lwIP example.
- 'lwIP-1.4.1' contains all the lwIP library files.

The active web server demo software supports two different variants. The folder structure is similar to the lwIP demo. In addition,

- The 'fs' folder contains the html file and Java script to generate webpage file.
- The 'tools' folder contains makefsfile utility.
- The 'utils' folder contains lwIP related initialization code.

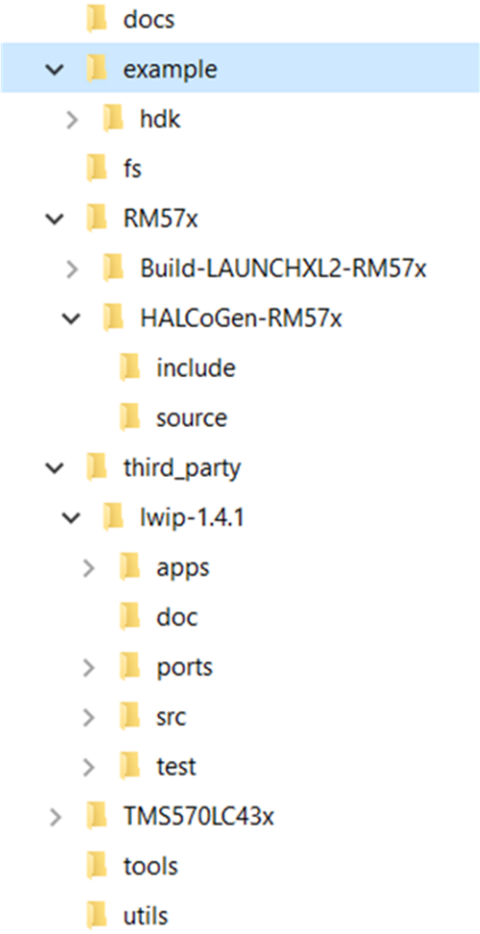


Figure 4. Active Web Server Demo Software Folder Structure

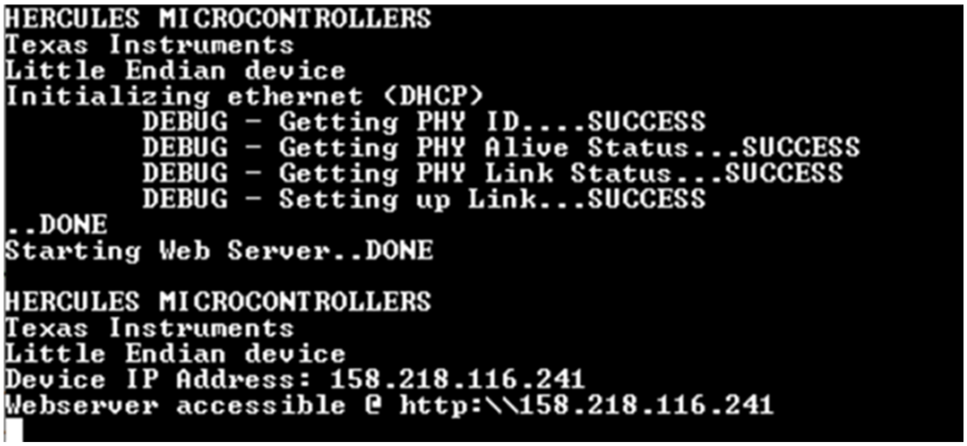
9 Run the Test

9.1 Hardware Setup

1. Connect Ethernet port on HDK or Launchpad to network with DHCP server.
2. For HDK, enable the Ethernet on EVM by setting switch S2 bit4 to “ON” position.
3. Connect the JTAG USB port to PC (used for console output also).
4. Set the console output with 9600, 8-N-2, no flow control.

9.2 Building and Executing the lwIP Demo

1. Import the project into CCS from <install_folder>/<device>/Build_xx folder, based on the board.
2. Build the project. Default target configuration is supplied with this software.
3. Download and execute the binary (disconnect the device from debugger and reset). Figure 5 shows a screen capture of the console output.

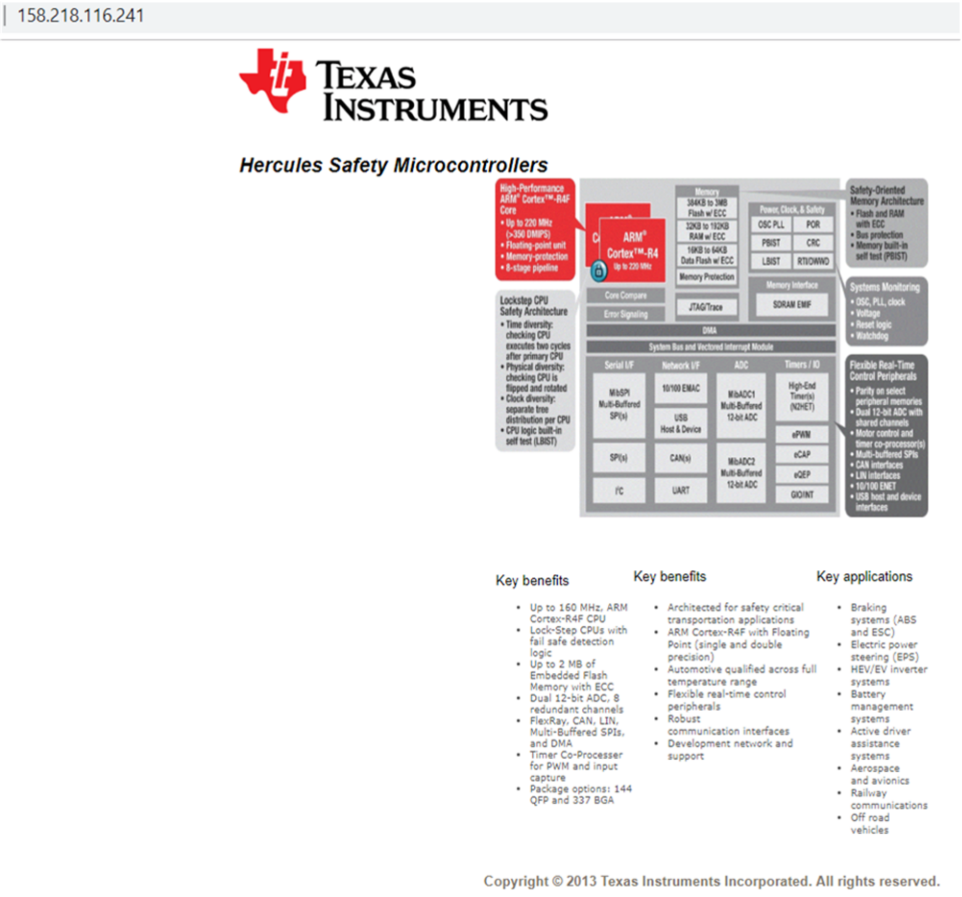


```
HERCULES MICROCONTROLLERS
Texas Instruments
Little Endian device
Initializing ethernet (DHCP)
        DEBUG - Getting PHY ID....SUCCESS
        DEBUG - Getting PHY Alive Status...SUCCESS
        DEBUG - Getting PHY Link Status...SUCCESS
        DEBUG - Setting up Link...SUCCESS
..DONE
Starting Web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

HERCULES MICROCONTROLLERS
Texas Instruments
Little Endian device
Device IP Address: 158.218.116.241
Webserver accessible @ http:\\158.218.116.241
```

Figure 5. Demo UART Console Output

4. The web server is accessible at the IP address displayed in the console.



158.218.116.241

Texas Instruments

Hercules Safety Microcontrollers

High-Performance ARM® Cortex™-R4F Core
• Up to 220 MHz (>350 DMIPS)
• Floating-point unit
• Memory-protection
• 8-stage pipeline
ARM® Cortex™-R4
Up to 220 MHz
Core Compare
Error Signaling
Lockstep CPU Safety Architecture
• Time diversity: checking CPU executes two cycles after primary CPU
• Physical diversity: checking CPU is flipped and rotated
• Clock diversity: separate tree distribution per CPU
• CPU logic built-in self test (LBIST)
Memory
384KB to 3MB Flash w/ ECC
32KB to 192KB RAM w/ ECC
16KB to 64KB Data Flash w/ ECC
Memory Protection
JTAG/Trace
Power, Clock, & Safety
OSC PLL
POR
PBIST
CRC
LBIST
RTI/DWWD
Memory Interface
SDRAM EMIF
DMA
System Bus and Vectored Interrupt Module
Serial I/F
MibSPI Multi-Buffered SPI(s)
SPI(s)
I²C
Network I/F
10/100 EMAC
USB Host & Device
CAN(s)
UART
ADC
MibADC1 Multi-Buffered 12-bit ADC
MibADC2 Multi-Buffered 12-bit ADC
Timers / IO
High-End Timer(s) (N2HET)
ePWM
eCAP
eQEP
GIOINT
Safety-Oriented Memory Architecture
• Flash and RAM with ECC
• Bus protection
• Memory built-in self test (PBIST)
Systems Monitoring
• OSC, PLL, clock
• Voltage
• Reset logic
• Watchdog
Flexible Real-Time Control Peripherals
• Parity on select peripheral memories
• Dual 12-bit ADC with shared channels
• Motor control and timer co-processor(s)
• Multi-buffered SPIs
• CAN interfaces
• LIN interfaces
• 10/100 ENET
• USB host and device interfaces

Key benefits

- Up to 160 MHz, ARM Cortex-R4F CPU
- Lock-Step CPUs with fail safe detection logic
- Up to 2 MB of Embedded Flash Memory with ECC
- Dual 12-bit ADC, 8 redundant channels
- FlexRay, CAN, LIN, Multi-Buffered SPIs, and DMA
- Timer Co-Processer for PWM and input capture
- Package options: 144 QFP and 337 BGA

Key benefits

- Architected for safety critical transportation applications
- ARM Cortex-R4F with Floating Point (single and double precision)
- Automotive qualified across full temperature range
- Flexible real-time control peripherals
- Robust communication interfaces
- Development network and support

Key applications

- Braking systems (ABS and ESC)
- Electric power steering (EPS)
- HEV/EV inverter systems
- Battery management systems
- Active driver assistance systems
- Aerospace and avionics
- Railway communications
- Off road vehicles

Copyright © 2013 Texas Instruments Incorporated. All rights reserved.

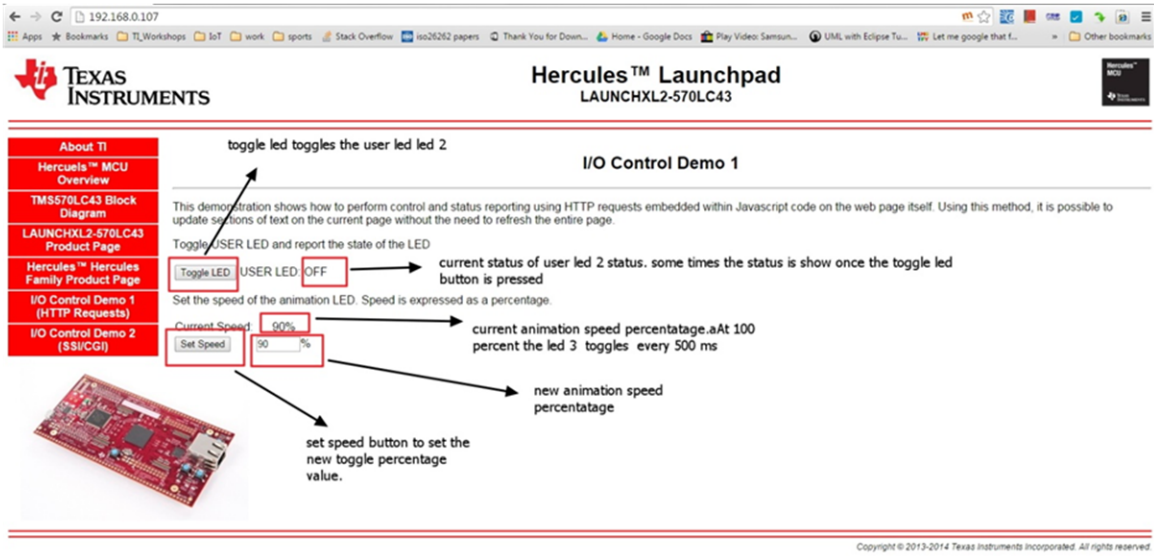
Figure 6. Web Server From lwIP Demo

9.3 Building and Executing the Active Web Server Demo

The active web server demo serves a web page that can be navigated, it is not a static web page and it also allows controls to control some LEDs on the Launchpad. There are two demos bundled.

9.3.1 I/O Control Demo 1

The demo provides the toggle LED button that can be used to toggle LED 2 on the Launchpad. The demo also shows the current status of the LED. It provides a text box that can be used to control the blinking speed of the animation on LED 3. This demo showcases the CGI handling capability of the web server demo. Figure 7 shows the IO control demo 1 page.



Hercules™ Launchpad
LAUNCHXL2-570LC43

I/O Control Demo 1

This demonstration shows how to perform control and status reporting using HTTP requests embedded within Javascript code on the web page itself. Using this method, it is possible to update sections of text on the current page without the need to refresh the entire page.

Toggle USER LED and report the state of the LED

USER LED: OFF

Set the speed of the animation LED. Speed is expressed as a percentage.

Current Speed: 90%

90 %

toggle led toggles the user led led 2

current status of user led 2 status. some times the status is show once the toggle led button is pressed

current animation speed percentatage.aAt 100 percent the led 3 toggles every 500 ms

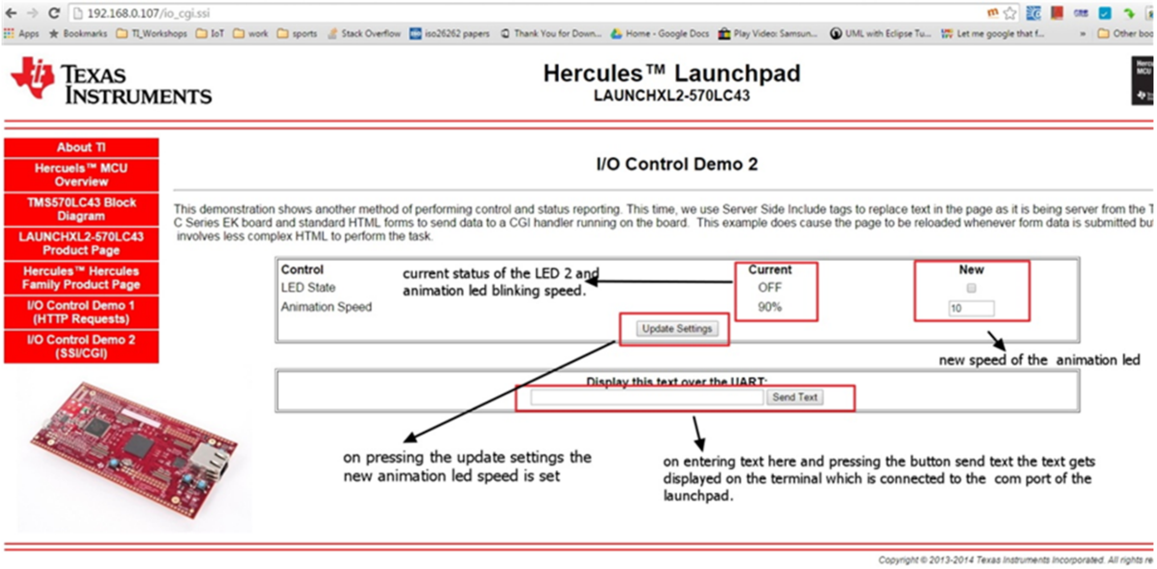
new animation speed percentatage

set speed button to set the new toggle percentage value.

Figure 7. Active Web Server I/O Control Demo 1

9.3.2 I/O Control Demo 2

The IO control demo 2 showcases the SSI capability of the web server. It shows the current status of LED 2 and the current blinking percentage of the animation on LED 3. The percentage of the blinking can be updated. It also provides a text box in which you can enter any text and observe it on the UART console connected to the com port of the launchpad.



Hercules™ Launchpad

LAUNCHXL2-570LC43

I/O Control Demo 2

This demonstration shows another method of performing control and status reporting. This time, we use Server Side Include tags to replace text in the page as it is being server from the T
C Series EK board and standard HTML forms to send data to a CGI handler running on the board. This example does cause the page to be reloaded whenever form data is submitted bu
involves less complex HTML to perform the task.

Control	Current	New
LED State	OFF	□
Animation Speed	90%	10

Display this text over the UART:

current status of the LED 2 and animation led blinking speed.

new speed of the animation led

on pressing the update settings the new animation led speed is set

on entering text here and pressing the button send text the text gets displayed on the terminal which is connected to the com port of the launchpad.

Copyright © 2013-2014 Texas Instruments Incorporated. All rights re

Figure 8. Active Web Server I/O Control Demo 2

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265