EK-TM4C1294XL-BOOSTXL-SENSHUB Firmware Development Package

USER'S GUIDE

SW-EK-TM4C1294XL-BOOSTXL-SENSHUB-UG-2.2.0.295

Copyright

Texas Instruments
108 Wild Basin, Suite 350
Austin, TX 78746
www.ti.com/tiva-c

Revision Information

This is version 2.2.0.295 of this document, last updated on April 02, 2020.

Table of Contents

1 Introduction

The Texas Instruments® Tiva™ EK-TM4C1294XL-BOOSTXL-SENSHUB evaluation board (Tiva C Series TM4C1294 Connected LaunchPad) is a low cost platform that can be used for software development and prototyping a hardware design. A variety of BoosterPacks are available to quickly extend the LaunchPad's features.

The EK-TM4C1294XL includes a Tiva ARM® Cortex™-M4-based microcontroller and the following features:

- Tiva™ TM4C1294NCPDT microcontroller
- Ethernet connector
- USB OTG connector
- 2 user buttons
- 4 User LEDs
- 2 BoosterPack XL sites
- On-board In-Circuit Debug Interface (ICDI)
- Power supply option from USB ICDI connection, USB OTG connection or external power connection
- Shunt jumper for microcontroller current consumption measurement

This document describes the example applications that are provided for the EK-TM4C1294XL when paired with the BOOSTXL-SENSHUB BoosterPack. This BoosterPack provides a variety of motion and environmental sensors. It also provides an EM expansion option for attachement of additional peripherals such as the CC2533EMK or CC4000EMK. These examples utilize the TivaWare™ for C Series Sensor Library to extract and process information from the BOOSTXL-SENSHUB.

2 Example Applications

The example applications show how to utilize features of this evaluation board. Examples are included to show how to use many of the general features of the Tiva microcontroller, as well as the feature that are unique to this evaluation board.

A number of drivers are provided to make it easier to use the features of this board. These drivers also contain low-level code that make use of the TivaWare peripheral driver library and utilities.

There is an IAR workspace file (`ek-tm4c1294xl-boostxl-senshub.eww`) that contains the peripheral driver library project, along with all of the board example projects, in a single, easy-to-use workspace for use with Embedded Workbench

There is a Keil multi-project workspace file (`ek-tm4c1294xl-boostxl-senshub.mpw`) that contains the peripheral driver library project, along with all of the board example projects, in a single, easy-to-use workspace for use with uVision.

All of these examples reside in the `examples/boards/ek-tm4c1294xl-boostxl-senshub` subdirectory of the firmware development package source distribution.

2.1 Nine Axis Sensor Fusion with the MPU9150 and Complimentary-Filtered DCM (compdcm_mpu9150)

This example demonstrates the basic use of the Sensor Library, TM4C1294 LaunchPad and SensHub BoosterPack to obtain nine axis motion measurements from the MPU9150. The example fuses the nine axis measurements into a set of Euler angles: roll, pitch and yaw. It also produces the rotation quaternions. The fusion mechanism demonstrated is a complimentary-filtered direct cosine matrix (DCM) algorithm. The algorithm is provided as part of the Sensor Library.

This example requires that the BOOSTXL-SENSHUB be installed on BoosterPack 1 interface headers. See code comments for instructions on how to use BoosterPack 2 interface.

Connect a serial terminal program to the LaunchPad's ICDI virtual serial port at 115,200 baud. Use eight bits per byte, no parity and one stop bit. The raw sensor measurements, Euler angles and quaternions are printed to the terminal. An LED begins to blink at 1Hz after initialization is completed and the example application is running.

2.2 Humidity Measurement with the SHT21 (humidity_sht21)

This example demonstrates the basic use of the Sensoror Library, TM4C1294XL LaunchPad and SensHub BoosterPack to obtain temperature and relative humidity of the environment using the Sensirion SHT21 sensor.

This example requires that the SensHub BoosterPack is installed on BoosterPack 1 interface headers on the LaunchPad. See the code comments for information on porting this to use BoosterPack 2.

Connect a serial terminal program to the LaunchPad's ICDI virtual serial port at 115,200 baud. Use eight bits per byte, no parity and one stop bit. The humidity and temperature as measured by the SHT21 is printed to the terminal. An LED will blink to indicate the application is running.

2.3 Humidity Measurement with the SHT21 (humidity_sht21_simple)

This example demonstrates the usage of the I2C interface to obtain the temperature and relative humidity of the environment using the Sensirion SHT21 sensor.

The I2C7 on the EK-TM4C1294XL launchPad is used to interface with the SHT21 sensor. The SHT21 sensor is on the BOOSTXL_SENSHUB boosterPack expansion board that can be directly plugged into the Booster pack 1 connector of the EK-TM4C1294XL launchPad board. Please make sure proper pull-up resistors are on the I2C SCL and SDA buses.

This example uses the following peripherals and I/O signals. You must review these and change as needed for your own board:

- I2C7 peripheral
- GPIO Port D peripheral
- I2C7_SCL - PD0
- I2C7_SDA - PD1

UART0, connected to the Virtual Serial Port and running at 115,200, 8-N-1, is used to display messages from this application.

2.4 Light Measurement with the ISL29023 (light_isl29023)

This example demonstrates the basic use of the Sensor Library, TM4C1294 Connected LaunchPad and the SensHub BoosterPack to obtain ambient and infrared light measurements with the ISL29023 sensor.

The SensHub BoosterPack must be installed on BoosterPack 1 interface. See code comments for changes needed to use BoosterPack 2 interface.

Connect a serial terminal program to the LaunchPad's ICDI virtual serial port at 115,200 baud. Use eight bits per byte, no parity and one stop bit. The raw sensor measurements are printed to the terminal. An LED blinks at 1Hz once the initialization is complete and the example is running.

The code automatically adjusts the dynamic range of the sensor when the intensity reaches a min or max threshold within the current range setting.

2.5 Pressure Measurement with the BMP180 (pressure_bmp180)

This example demonstrates the basic use of the Sensor Library, the EK-TM4C1294XL LaunchPad, and the SensHub BoosterPack to obtain air pressure and temperature measurements with the

BMP180 sensor.

SensHub BoosterPack (BOOSTXL-SENSHUB) must be installed on BoosterPack 1 interface headers.

Instructions for use of SensorHub on BoosterPack 2 headers are in the code comments.

Connect a serial terminal program to the LaunchPad's ICDI virtual serial port at 115,200 baud. Use eight bits per byte, no parity and one stop bit. The raw sensor measurements are printed to the terminal. The LED blinks at 1 Hz once the initialization is complete and the example is running.

2.6 Temperature Measurement with the TMP006 (temperature_tmp006)

This example demonstrates the basic use of the Sensor Library, TM4C1294 Connected LaunchPad and the SensHub BoosterPack to obtain ambient and object temperature measurements with the Texas Instruments TMP006 sensor.

SensHub BoosterPack (BOOSTXL-SENSHUB) Must be installed on BoosterPack 1 interface headers.

Connect a serial terminal program to the LaunchPad's ICDI virtual serial port at 115,200 baud. Use eight bits per byte, no parity and one stop bit. The raw sensor measurements are printed to the terminal. An LED blinks at 1Hz once the initialization is complete and the example is running.

3 Buttons Driver

3.1 Introduction

The buttons driver provides functions to make it easy to use the push buttons on this evaluation board. The driver provides a function to initialize all the hardware required for the buttons, and features for debouncing and querying the button state.

This driver is located in `examples/boards/ek-tm4c1294xl-boostxl-senshub/drivers`, with `buttons.c` containing the source code and `buttons.h` containing the API declarations for use by applications.

3.2 API Functions

Functions

- void ButtonsInit (void)
- uint8_t ButtonsPoll (uint8_t ∗pui8Delta, uint8_t ∗pui8RawState)

3.2.1 Function Documentation

3.2.1.1 ButtonsInit

Initializes the GPIO pins used by the board pushbuttons.

Prototype:
```
void
ButtonsInit(void)
```

Description:
This function must be called during application initialization to configure the GPIO pins to which the pushbuttons are attached. It enables the port used by the buttons and configures each button GPIO as an input with a weak pull-up.

Returns:
None.

3.2.1.2 ButtonsPoll

Polls the current state of the buttons and determines which have changed.

Prototype:

```
uint8_t
ButtonsPoll(uint8_t *pui8Delta,
            uint8_t *pui8RawState)
```

Parameters:

pui8Delta points to a character that will be written to indicate which button states changed since the last time this function was called. This value is derived from the debounced state of the buttons.

pui8RawState points to a location where the raw button state will be stored.

Description:

This function should be called periodically by the application to poll the pushbuttons. It determines both the current debounced state of the buttons and also which buttons have changed state since the last time the function was called.

In order for button debouncing to work properly, this function should be called at a regular interval, even if the state of the buttons is not needed that often.

If button debouncing is not required, the the caller can pass a pointer for the *pui8RawState* parameter in order to get the raw state of the buttons. The value returned in *pui8RawState* will be a bit mask where a 1 indicates the buttons is pressed.

Returns:

Returns the current debounced state of the buttons where a 1 in the button ID's position indicates that the button is pressed and a 0 indicates that it is released.

3.3 Programming Example

The following example shows how to use the buttons driver to initialize the buttons, debounce and read the buttons state.

```
//
// Map Left button to the GPIO Pin 0 of the button port.
//
#define LEFT_BUTTON             GPIO_PIN_0

//
// The button example
//
void
ButtonExample(void)
{
    unsigned char ucDelta, ucState;

    //
    // Initialize the buttons.
    //
    ButtonsInit();

    //
    // From timed processing loop (for example every 10 ms)
    //
    {
        //
        // Poll the buttons.  When called periodically this function will
        // run the button debouncing algorithm.
        //
        ucState = ButtonsPoll(&ucDelta, 0);

        //
        // Test to see if the SELECT button was pressed and do something
        //
        if(BUTTON_PRESSED(LEFT_BUTTON, ucState, ucDelta))
        {
            //
            // TODO: SELECT button action code
            //
        }
    }
}
```

4 Pinout Module

4.1 Introduction

The pinout module is a common function for configuring the device pins for use by example applications. The pins are configured into the most common usage; it is possible that some of the pins might need to be reconfigured in order to support more specialized usage.

This driver is located in `examples/boards/ek-tm4c1294xl-boostxl-senshub/drivers`, with `pinout.c` containing the source code and `pinout.h` containing the API declarations for use by applications.

4.2 API Functions

Functions

- void LEDRead (uint32_t *pui32LEDValue)
- void LEDWrite (uint32_t ui32LEDMask, uint32_t ui32LEDValue)
- void PinoutSet (bool bEthernet, bool bUSB)

4.2.1 Function Documentation

4.2.1.1 LEDRead

This function reads the state to the LED bank.

Prototype:

```
void
LEDRead(uint32_t *pui32LEDValue)
```

Parameters:

pui32LEDValue is a pointer to where the LED value will be stored.

Description:

This function reads the state of the CLP LEDs and stores that state information into the variable pointed to by pui32LEDValue.

Returns:

None.

4.2.1.2 LEDWrite

This function writes a state to the LED bank.

Prototype:

```
void
LEDWrite(uint32_t ui32LEDMask,
         uint32_t ui32LEDValue)
```

Parameters:

ui32LEDMask is a bit mask for which GPIO should be changed by this call.

ui32LEDValue is the new value to be applied to the LEDs after the ui32LEDMask is applied.

Description:

The first parameter acts as a mask. Only bits in the mask that are set will correspond to LEDs that may change. LEDs with a mask that is not set will not change. This works the same as GPIOPinWrite. After applying the mask the setting for each unmasked LED is written to the corresponding LED port pin via GPIOPinWrite.

Returns:

None.

4.2.1.3 PinoutSet

Configures the device pins for the standard usages on the EK-TM4C1294XL.

Prototype:

```
void
PinoutSet(bool bEthernet,
          bool bUSB)
```

Parameters:

bEthernet is a boolean used to determine function of Ethernet pins. If true Ethernet pins are configured as Ethernet LEDs. If false GPIO are available for application use.

bUSB is a boolean used to determine function of USB pins. If true USB pins are configured for USB use. If false then USB pins are available for application use as GPIO.

Description:

This function enables the GPIO modules and configures the device pins for the default, standard usages on the EK-TM4C1294XL. Applications that require alternate configurations of the device pins can either not call this function and take full responsibility for configuring all the device pins, or can reconfigure the required device pins after calling this function.

Returns:

None.

4.3 Programming Example

The following example shows how to configure the device pins.

```
//
// The pinout example.
//
void
PinoutExample(void)
{
    //
    // Configure the device pins.
    // First argument determines whether the Ethernet pins will be configured
    // in networking mode for this application.
    // Second argument determines whether the USB pins will be configured for
    // USB mode for this application.
    //
    PinoutSet(true, false);
}
```

5 HTTP Driver

5.1 Introduction

The http.c file provides functions to facilatate the creation and management of HTTP application. These functions are used by the Exosite Abstraction Layer `exosite_hal_lwip.c` to send HTTP requests to the Exosite server for IoT applications.

This driver is located in `examples/boards/ek-tm4c1294xl-boostxl-senshub/drivers`, with `http.c` containing the source code and `http.h` containing the API declarations for use by applications.

5.2 API Functions

Functions

- void HTTPMessageBodyAdd (char ∗pcDest, char ∗pcBodyData)
- void HTTPMessageHeaderAdd (char ∗pcDest, char ∗pcHeaderName, char ∗pcHeaderValue)
- void HTTPMessageTypeSet (char ∗pcDest, uint8_t ui8Type, char ∗pcResource)
- void HTTPResponseBodyExtract (char ∗pcData, char ∗pcDest)
- void HTTPResponseHeaderExtract (char ∗pcData, uint32_t ui32HeaderIdx, char ∗pcHeaderName, char ∗pcHeaderValue)
- uint32_t HTTPResponseParse (char ∗pcData, char ∗pcResponseText, uint32_t ∗pui32NumHeaders)

5.2.1 Function Documentation

5.2.1.1 HTTPMessageBodyAdd

Add body data to to a HTTP request.

Prototype:

```
void
HTTPMessageBodyAdd(char *pcDest,
                   char *pcBodyData)
```

Parameters:

pcDest is a pointer to the destination/output string.

pcBodyData is a pointer to a string containing the body data. This can be anything from HTML to encoded data (such as JSON).

Description:
Note that this function must be called after HTTPMessageTypeSet() and HTTPMessageHeaderAdd() as it simply appends the body data to an existing string/buffer.

Returns:
None.

5.2.1.2 HTTPMessageHeaderAdd

Add a header to a HTTP request.

Prototype:
```
void
HTTPMessageHeaderAdd(char *pcDest,
                     char *pcHeaderName,
                     char *pcHeaderValue)
```

Parameters:
pcDest is a pointer to the destination/output string.
pcHeaderName is a pointer to a string containing the header name.
pcHeaderValue is a pointer to a string containing the header data.

Description:
Note that this function must be called after HTTPMessageTypeSet() as it simply appends a header to an existing string/buffer.

Returns:
None.

5.2.1.3 HTTPMessageTypeSet

Set the HTTP message type.

Prototype:
```
void
HTTPMessageTypeSet(char *pcDest,
                   uint8_t ui8Type,
                   char *pcResource)
```

Parameters:
pcDest is a pointer to the destination/output string.
ui8Type is the HTTP request type. Macros such as HTTP_MESSAGE_GET are defined in http.h.
pcResource is a pointer to a string containing the resource portion of the HTTP message. The resource goes in between the type (ex: GET) and HTTP suffix on the first line of a HTTP request. An example would be index.html.

Description:
This function should be called to start off a new HTTP request.

Returns:
None.

5.2.1.4 HTTPResponseBodyExtract

Extract the body from a HTTP response string/buffer.

Prototype:

```
void
HTTPResponseBodyExtract(char *pcData,
                        char *pcDest)
```

Parameters:

pcData is a pointer to the source string/buffer.

pcDest is a pointer to a string that will receive the body data.

Returns:

None.

5.2.1.5 void HTTPResponseHeaderExtract (char ∗ *pcData*, uint32_t *ui32HeaderIdx*, char ∗ *pcHeaderName*, char ∗ *pcHeaderValue*)

Extract a specified header from a HTTP response string/buffer.

Parameters:

pcData is a pointer to the source string/buffer.

ui32HeaderIdx specifies the index of the header to extract.

pcHeaderName is a pointer to a string that will receive the name of the header specified by ui32HeaderIdx.

pcHeaderValue is a pointer to a string that will receive the value of the header specified by ui32HeaderIdx.

Description:

Note that this function should be used in conjunction with HTTPResponseParse() since it notifies the application of the number of headers in a string/buffer.

Returns:

None.

5.2.1.6 HTTPResponseParse

Parse a HTTP response.

Prototype:

```
uint32_t
HTTPResponseParse(char *pcData,
                  char *pcResponseText,
                  uint32_t *pui32NumHeaders)
```

Parameters:

pcData is a pointer to the source string/buffer.

pcResponseText is a pointer to a string that will receive the response text from the first line of the HTTP response.

pui32NumHeaders is a pointer to a variable that will receive the number of headers detected in pcData.

Description:

Note that this function must be called after HTTPMessageTypeSet() as it simply appends a header to an existing string/buffer.

Returns:

Returns the HTTP response code. If parsing error occurs, returns 0.

6 Ethernet Client Driver

6.1 Introduction

The eth_client_lwip.c file provides Ethernet client functions to interface with the server. These functions are used by the Exosite Abstraction Layer `exosite_hal_lwip.c` to manage connection with the Exosite server in IoT applications.

This driver is located in `examples/boards/ek-tm4c1294xl-boostxl-senshub/drivers`, with `eth_client_lwip.c` containing the source code and `eth_client_lwip.h` containing the API declarations for use by applications.

6.2 API Functions

Functions

- uint32_t EthClientAddrGet (void)
- err_t EthClientDHCPConnect (void)
- int32_t EthClientDNSResolve (void)
- void EthClientHostSet (const char ∗pcHostName, uint16_t ui16Port)
- void EthClientInit (uint32_t ui32SysClock, tEventFunction pfnEvent)
- void EthClientMACAddrGet (uint8_t ∗pui8MACAddr)
- void EthClientProxySet (const char ∗pcProxyName, uint16_t ui16Port)
- int32_t EthClientSend (int8_t ∗pi8Request, uint32_t ui32Size)
- uint32_t EthClientServerAddrGet (void)
- int32_t EthClientTCPConnect (void)
- void EthClientTCPDisconnect (void)
- void lwIPHostTimerHandler (void)
- err_t TCPConnected (void ∗pvArg, struct tcp_pcb ∗psPcb, err_t iErr)
- void TCPError (void ∗vPArg, err_t iErr)
- err_t TCPReceived (void ∗pvArg, struct tcp_pcb ∗psPcb, struct pbuf ∗psBuf, err_t iErr)
- err_t TCPSent (void ∗pvArg, struct tcp_pcb ∗psPcb, u16_t ui16Len)

6.2.1 Function Documentation

6.2.1.1 EthClientAddrGet

Returns the IP address for this interface.

Prototype:

```
uint32_t
EthClientAddrGet(void)
```

Description:

This function will read and return the currently assigned IP address for the Tiva Ethernet interface.

Returns:

Returns the assigned IP address for this interface.

6.2.1.2 EthClientDHCPConnect

DHCP connect

Prototype:

```
err_t
EthClientDHCPConnect(void)
```

Description:

This function obtains the MAC address from the User registers, starts the DHCP timer and blocks until an IP address is obtained.

Returns:

This function always returns ERR_OK.

6.2.1.3 EthClientDNSResolve

Handler function when the DNS server gets a response or times out.

Prototype:

```
int32_t
EthClientDNSResolve(void)
```

Description:

This function is called when the DNS server resolves an IP or times out. If the DNS server returns an IP structure that is not NULL, add the IP to to the g_sEnet.sResolvedIP IP structure.

Returns:

This function will return an lwIP defined error code.

6.2.1.4 EthClientHostSet

Set the host string for the Ethernet connection.

Prototype:

```
void
EthClientHostSet(const char *pcHostName,
                 uint16_t ui16Port)
```

Parameters:

pcHostName is the string used as the host server name.

ui16Port is the string used as the host port.

Description:

This function sets the current host used by the Ethernet connection. The *pcHostName* value can be 0 to indicate that no host is in use or it can be a pointer to a string that holds the name of the host server to use. The content of the pointer passed to *pcHostName* should not be changed after this call as this function only stores the pointer and does not copy the data from this pointer.

Returns:

None.

6.2.1.5 EthClientInit

Initialize the Ethernet client

Prototype:

```
void
EthClientInit(uint32_t ui32SysClock,
              tEventFunction pfnEvent)
```

Parameters:

ui32SysClock is the input for the system clock setting of the TM4C

pfnEvent is the network event handler.

Description:

This function initializes all the Ethernet components to not configured. This tells the SysTick interrupt which timer modules to call.

Returns:

None.

6.2.1.6 EthClientMACAddrGet

Returns the MAC address for the Tiva Ethernet controller.

Prototype:

```
void
EthClientMACAddrGet(uint8_t *pui8MACAddr)
```

Parameters:

pui8MACAddr is the 6 byte MAC address assigned to the Ethernet controller.

Description:

This function will read and return the MAC address for the Ethernet controller.

Returns:

Returns the weather server IP address for this interface.

6.2.1.7 EthClientProxySet

Set the proxy string for the Ethernet connection.

Prototype:

```
void
EthClientProxySet(const char *pcProxyName,
                  uint16_t ui16Port)
```

Parameters:

pcProxyName is the string used as the proxy server name.

ui16Port is the string used as the proxy port.

Description:

This function sets the current proxy used by the Ethernet connection. The *pcProxyName* value can be 0 to indicate that no proxy is in use or it can be a pointer to a string that holds the name of the proxy server to use. The content of the pointer passed to *pcProxyName* should not be changed after this call as this function only stores the pointer and does not copy the data from this pointer.

Returns:

None.

6.2.1.8 EthClientSend

Send a request to the server

Prototype:

```
int32_t
EthClientSend(int8_t *pi8Request,
              uint32_t ui32Size)
```

Parameters:

pi8Request request to be sent

ui32Size length of the request to be sent. This is usually the size of the request minus the termination character.

Description:

This function will send the request to the connected server

Returns:

This function will return an lwIP defined error code.

6.2.1.9 EthClientServerAddrGet

Returns the weather server IP address for this interface.

Prototype:

```
uint32_t
EthClientServerAddrGet(void)
```

Description:
This function will read and return the server IP address that is currently in use. This could be the proxy server if the Internet proxy is enabled.

Returns:
Returns the weather server IP address for this interface.

6.2.1.10 EthClientTCPConnect

TCP connect

Prototype:
```
int32_t
EthClientTCPConnect(void)
```

Description:
This function attempts to connect to a TCP endpoint.

Returns:
This functions returns a '0' if successful, or a '1' if an error an error has occurred.

6.2.1.11 EthClientTCPDisconnect

TCP discconnect

Prototype:
```
void
EthClientTCPDisconnect(void)
```

Description:
This function attempts to disconnect a TCP endpoint.

Returns:
None.

6.2.1.12 lwIPHostTimerHandler

Periodic Tick for the Ethernet client

Prototype:
```
void
lwIPHostTimerHandler(void)
```

Description:
This function is the needed periodic tick for the Ethernet client. It needs to be called periodically through the use of a timer or systick.

Returns:
None.

6.2.1.13 TCPConnected

Finalizes the TCP connection in client mode.

Prototype:
```
err_t
TCPConnected(void *pvArg,
             struct tcp_pcb *psPcb,
             err_t iErr)
```

Parameters:
pvArg is the state data for this connection.
psPcb is the pointer to the TCP control structure.
iErr is not used in this implementation.

Description:
This function is called when the lwIP TCP/IP stack has completed a TCP connection.

Returns:
This function will return an lwIP defined error code.

6.2.1.14 TCPError

Handles lwIP TCP/IP errors.

Prototype:
```
void
TCPError(void *vPArg,
         err_t iErr)
```

Parameters:
vPArg is the state data for this connection.
iErr is the error that was detected.

Description:
This function is called when the lwIP TCP/IP stack has detected an error. The connection is no longer valid.

Returns:
None.

6.2.1.15 TCPReceived

Finalizes the TCP connection in client mode.

Prototype:
```
err_t
TCPReceived(void *pvArg,
            struct tcp_pcb *psPcb,
            struct pbuf *psBuf,
            err_t iErr)
```

Parameters:

pvArg is the state data for this connection.

psPcb is the pointer to the TCP control structure.

psBuf

iErr is not used in this implementation.

Description:

This function is called when the lwIP TCP/IP stack has completed a TCP connection.

Returns:

This function will return an lwIP defined error code.

6.2.1.16 TCPSent

Handles acknowledgment of data transmitted via Ethernet.

Prototype:

```
err_t
TCPSent(void *pvArg,
        struct tcp_pcb *psPcb,
        u16_t ui16Len)
```

Parameters:

pvArg is the state data for this connection.

psPcb is the pointer to the TCP control structure.

ui16Len is the length of the data transmitted.

Description:

This function is called when the lwIP TCP/IP stack has received an acknowledgment for data that has been transmitted.

Returns:

This function will return an lwIP defined error code.

Products		**Applications**	
Audio	www.ti.com/audio	Automotive and Transportation	www.ti.com/automotive
Amplifiers	amplifier.ti.com	Communications and Telecom	www.ti.com/communications
Data Converters	dataconverter.ti.com	Computers and Peripherals	www.ti.com/computers
DLP® Products	www.dlp.com	Consumer Electronics	www.ti.com/consumer-apps
DSP	dsp.ti.com	Energy and Lighting	www.ti.com/energy
Clocks and Timers	www.ti.com/clocks	Industrial	www.ti.com/industrial
Interface	interface.ti.com	Medical	www.ti.com/medical
Logic	logic.ti.com	Security	www.ti.com/security
Power Mgmt	power.ti.com	Space, Avionics and Defense	www.ti.com/space-avionics-defense
Microcontrollers	microcontroller.ti.com	Video and Imaging	www.ti.com/video
RFID	www.ti-rfid.com		
OMAP Applications Processors	www.ti.com/omap	**TI E2E Community**	e2e.ti.com
Wireless Connectivity	www.ti.com/wirelessconnectivity		