Tiny Gecko

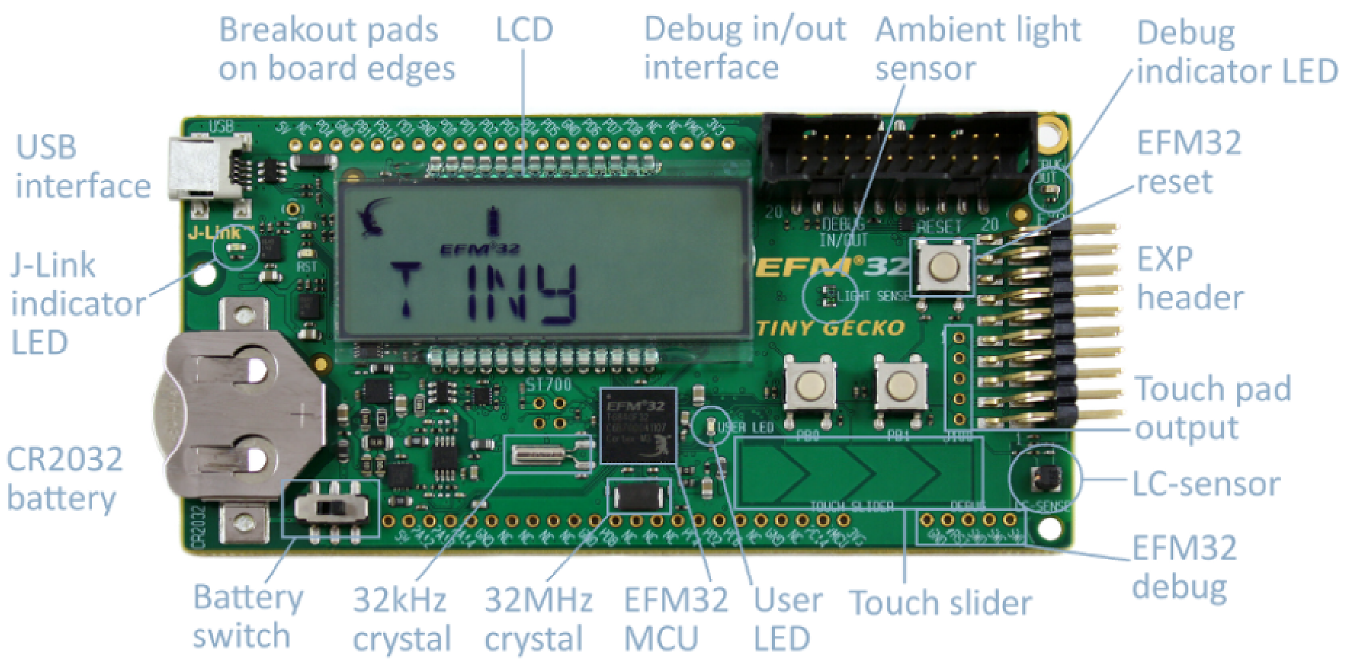
Tiny GECKO Starter Kit STK3300 is a prototyping and application development tool for the EFM32TG MCU family with the ARM Cortex-M3 CPU core. We use EMF32TG840F32 in EFM32TG family. Its SPEC is as follows.

EFM32TG Part #	Flash	RAM	GPIO(pins)	LCD	USART	LEUART	I^2C	Timer(PWM)	LETIMER	RTC	PCNT	Watchdog	ADC(pins)	DAC(pins)	ACMP(pins)	AES	EBI	LESENSE	Op-Amps	Package
840F32	32	4	56	Y	2	1	1	2 (6)	1	1	1	1	1 (8)	2 (2)	2 (8)	Y	-	Y	3	QFN64

Features

- Advanced Energy Monitoring system for precise current tracking.
- Special hardware configuration for isolation of the MCU power domain.
- Full feature USB debugger with debug out functionality.
- 160 segment Energy Micro LCD.
- **20 pin expansion header.** ← *To be used for interfacing with W5500(WIZ550io)*
- Breakout pads for easy access to I/O pins.
- Powered by USB or CR2032 battery.
- 2 user buttons, 1 user LED and a touch slider.
- Ambient Light sensor and inductive-capacitive metal sensor.
- EFM32 Op-amp footprint.
- 32MHz and 32.768kHz crystal oscillators.

Hardware Layout



Preparation for adding W5500

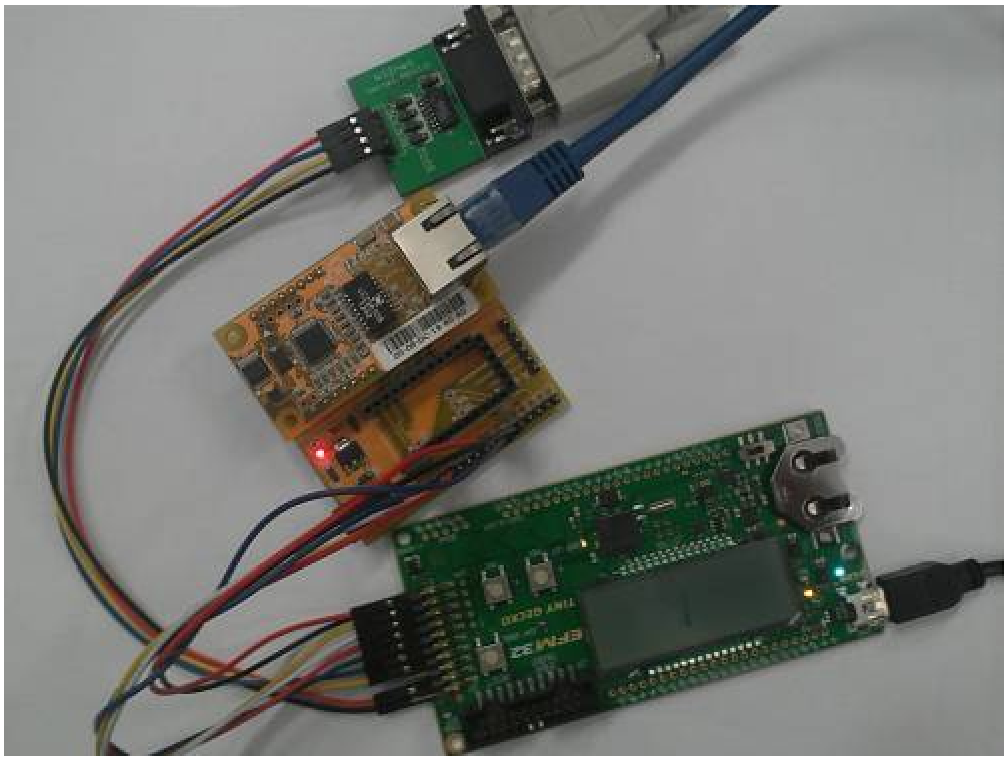
- WIZ550io or ioShileds
- Tiny Gecko Starter Kit

- IAR Embedded Workbench for ARM
- Simplicity Studio
  - Energy Micro Library : emlib
  - Application notes & example
  - Documents

- Download the latest ioLibrary BSD
  - Ethernet
  - Internet

How to connect W5500(WIZ550io)

	TinyGecko 20Pins EXP Header (TG840F032)	W5500 (WIZ550io)
MOSI	#4 (PD0)	#34 (J1:3)
MISO	#6 (PD1)	#35 (J1:4)
SCLK	#8 (PD2)	#33 (J1:5)
SCS	#10 (PD3)	#32 (J1:6)
UART TX	#12 (PD4)	Yours
UART RX	#14 (PD5)	Yours
Power	3.3V/VMCU/5V & GND	VDD & GND(J1:7-8/J2:1 & J1:1-2/J2:6)

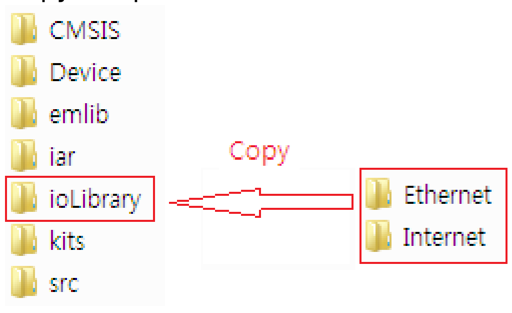


Quick Start

Download IAR project for example (Loopback/DHCP/DNS)

- efm32tg840f32_w5500_20131220.zip
- Extract the downloaded zip file on your PC.

Install the latest ioLibrary BSD

- Project's ioLibrary is **empty**
- Install ioLibrary
  - Copy the previous downloaded ioLibrary_bsd [Ethernet] into [ioLibrary]
  - Copy the previous downloaded ioLibrary_bsd [Internet] into [ioLibrary]
  - 

Open Project & Make image

- Using IAR Embedded Workbench for ARM
  - Open **LoopbackTest.eww(DHCPTest.eww,DNSTest.eww)** in the project's [iar] directory.
  - Make the executable image (.bin or .out)
- *For more detail, refer to Tools manual provided IAR.*

> If You want to monitor the progressing or debugging message of the project on your Terminal program, Download the image in [iar»Debug»Exe].

Program(download) the executable image thru USB J-Link

- Using IAR Embedded Workbench for ARM
- Using AwareEnergy Commander of Simplicity Stdio
- *For more detail, Refer to Tools manual provided IAR or SiLab.*

How to Test

Tiny Gecko Config

- HFCORECLOCK : 32MHz
- HFPERCLCOK : 32MHz
- LEUART0 : 32.768KHz
  - Baud rate : 14400 (8-N-1)
- USART0 : 8MHz (*Doesn't work well on 16MHz. Maybe connected by noisy wire-jumper-line.*)
  - For SPI interface with W5500(WIZ550io)

Monitor & Debug

Define **DEBUG_MODE** with one among *DEBUG_SWO* and *DEBUG_LEUART* in [src»bsp_tg840.h]

bsp_tg840.h

```
   /* DEBUG MODE */
   #define DEBUG_NO        0
   #define DEBUG_SWO       1
   #define DEBUG_LEUART    2

   #if defined(NDEBUG)    //<- NDEBUG is defined by IAR tool in making
release image.
      #define DEBUG_MODE      DEBUG_NO
   #else
      #include <stdio.h>
      #define DEBUG_MODE      DEBUG_SWO    //<- For using IAR's
Terminal IO window.
    //#define DEBUG_MODE      DEBUG_LEUART //<- For Your serail
Terminal program
   #endif
```

Allocate Memory for Internet Data

You can adjust the memory size for Internet data in [src»bsp_tg840.h].

bsp_tg840.h

```
/* MAX DATA BUFFER SIZE */
#define DATA_BUF_SIZE   2048    // It is depend on Target System Memory
```

Loopback

The loopback example runs with a TCP session and a UDP session.

First, Board and your PC should have the network config with the same network range.
If you want to modify board-side, edit the following code in [src»LB_main.c] with the same range which your PC has. If you want to modify your PC-side, refer to IP configuration.

LB_main.c

```
wiz_NetInfo gWIZNETINFO = { .mac = {0x00, 0x08, 0xdc,0x00, 0xab, 0xcd},
//<-mac should be unique.
                            .ip = {192, 168, 1, 123},
                            .sn = {255,255,255,0},
                            .gw = {192, 168, 1, 1},
                            .dns = {0,0,0,0},
                            .dhcp = NETINFO_STATIC };
```

TCP

1. Connect to Board
   - Using Hercules test program or others, try to connect to board with xxx.xxx.xxx.xxx listen port 5000.
2. When connected
   - send data to board
   - check whether the loopback data is same to what it sent before.
3. When failed to connect
   - Check link status.
   - Check ping test
   - Check network config.
   - Check the security program as virus vaccines and fire-wall on your PC.

UDP

1. Send data
   - Using Hercules test program or others, send to board's xxx.xxx.xxx.xxx port 3000.
2. When success
   - check to the loopback data
3. When fail
   - Check link status.
   - Check ping test
   - Check network config.
   - Check the security program as virus vaccines and fire-wall on your PC.

DHCP

DHCP example runs with one UDP socket. If you want the Dynamic IP instead of static, Apply it.

To test the example,

- First ready to DHCP server to run on the LAN belongs to the board.
- SHOULD BE set the **unique** MAC address to W5500's SHAR register before *DHCP_init()* call.

If you want to monitor or display message of DHCP processing, Define DEBUG_MODE in bsp_tg840.h (refer to the above description *“How to test»Monitor & Debug”*.) and Define **_DHCP_DEBUG_** in [ioLibrary»Internet»DHCP»dhcp.h].

dhcp.h

```
/*
 * @brief
 * @details If you want to display debug & processing message, Define
_DHCP_DEBUG_
 * @note    If defined, it dependents on <stdio.h>
 */
//#define _DHCP_DEBUG_
```

When DHCP run in DEBUG mode, Force to be the leased time 10 seconds for quickly testing to maintain the leased IP. The below codes are in *parseDHCPMSG()* of “dhcp.c”.

dhcp.c

```
   case dhcpIPaddrLeaseTime :
      p++;
      opt_len = *p++;
      dhcp_lease_time  = *p++;
      dhcp_lease_time  = (dhcp_lease_time << 8) + *p++;
      dhcp_lease_time  = (dhcp_lease_time << 8) + *p++;
      dhcp_lease_time  = (dhcp_lease_time << 8) + *p++;
   #ifdef _DHCP_DEBUG_
      dhcp_lease_time = 10;    // <- Force to be leased time 10 secs.
   #endif
```

1. Reset the board for starting DHCP processing.
2. DHCP running
   - Send DISCOVER message
   - Receive OFFER message
   - Send REQUEST message
   - Receive ACK message (when success). But Receive NACK message or Timeout occurred (when fail)
3. When success
   - If first leased IP, Call the default function or yours when callback is registered by reg_dhcp_cbfunc().
   - For maintaining the leased IP, Send DISCOVER message every the half of leased time.
     - During this phase, If the leased IP is changed, Call the default function or yours when callback is registered by reg_dhcp_cbfunc().
   - Every the leased IP, Send ARP the leased IP for checking IP Conflict.

   - When IP conflict, call the default function or yours when callback is registered by reg_dhcp_cbfunc().
4. When failed
   - When receiving NAK message or timeout occurred after the max time (**DHCP_WAIT_TIME** X **DHCP_RETRY_CNT**) to wait receive ACK or NACK message.
   - The default static IP configuration applied by *gWIZNETINFO* in DHCP_main.c
   - Check link status of board.
   - Check the MAC address to be UNIQUE
   - Check the security program as virus vaccines and fire-wall o your PC.
   - Check the DHCP server on the LAN belongs to the board.

DHCP_WAIT_TIME & DHCP_RETRY_CNT And DHCP_time_handler()

For checking timeout to DHCP processing, Define *DHCP_WAIT_TIME* & *DHCP_RETRY_CNT* in dhcp.h and SHOULD BE register *DHCP_time_handler()* in your 1s timer handler.

The below code is the max timeout 30s (10 sec * 3 retry count).

dhcp.h

```
   /* Retry to processing DHCP */
   #define     MAX_DHCP_RETRY          2        ///< Maximum retry count
   #define     DHCP_WAIT_TIME          10       ///< Wait Time 10s
```

The below is example to register DHCP_time_handler() in DHCP_main.c

DHCP_main.c

```
/***********************************************************************
****//**
 * @brief SysTick_Handler
 * Interrupt Service Routine for system tick counter

***********************************************************************
******/
void SysTick_Handler(void)
{
  msTicks++;       /* increment counter necessary in Delay()*/

  ///////////////////////////////////////////////////////
  // SHOULD BE Added DHCP Timer Handler your 1s tick timer
  if(msTicks % 1000 == 0)  DHCP_time_handler();
  ///////////////////////////////////////////////////////
}
```

Register your callback function

DHCP_main.c

```
/*******************************************************
 * @ brief Call back for ip assign & ip update from DHCP
 *******************************************************/
void my_ip_assign(void)
{
   getIPfromDHCP(gWIZNETINFO.ip);
   getGWfromDHCP(gWIZNETINFO.gw);
   getSNfromDHCP(gWIZNETINFO.sn);
   getDNSfromDHCP(gWIZNETINFO.dns);
   gWIZNETINFO.dhcp = NETINFO_DHCP;
   /* Network initialization */
   network_init();      // apply from dhcp
#if DEBUG_MODE != DEBUG_NO
   printf("DHCP LEASED TIME : %d Sec.\r\n", getDHCPLeasetime());
#endif
}

/************************************
 * @ brief Call back for ip Conflict
 ************************************/
void my_ip_conflict(void)
{
#if DEBUG_MODE != DEBUG_NO
   printf("CONFLICT IP from DHCP\r\n");
#endif
   //halt or reset or any...
   while(1); // this example is halt.
}

int main(void)
{
  //
  // Initialize the board
  //

  // SHOULD BE set the UNIQUE mac before DHCP started.
  setSHAR(gWIZNETINFO.mac);

  DHCP_init(SOCK_DHCP, gDATABUF);

  // if you want different action instead default ip assign, update,
conflict,
  // if cbfunc == 0, act as default.
  reg_dhcp_cbfunc(my_ip_assign,my_ip_assign,my_ip_conflict);

  while(1)
  {
     /// your main loop
  }
}
```

If any call back function is null, Call the default function in dhcp.c

- When *ip_assign* is null, call the *default_ip_assign()*.
- When *ip_update* is null, call the *default_ip_update()*.
- When *ip_conflict* is null, call the *default_ip_conflict()*.

The Default Static Config When Failed

In this example, When DCHP is failed **MY_MAX_DHCP_RETRY** times, Apply the default static configuration as below.

DHCP_main.c

```
#define MY_MAX_DHCP_RETRY  5      //<- DHCP process is five times
attempted.


/**********************************************************************
****//**
 * @brief Default Network Information
 
**********************************************************************
******/
wiz_NetInfo gWIZNETINFO =
        {
            .mac = {0x00, 0x08, 0xdc,0x00, 0xab, 0xcd},
            .ip = {192, 168, 1, 123},
            .sn = {255,255,255,0},
            .gw = {192, 168, 1, 1},
            .dns = {0,0,0,0},
            .dhcp = NETINFO_DHCP
        };
```

DHCP_init() / DHCP_run() / DHCP_stop()

DHCP_init can be called for initializing DHCP and restarting DHCP.
DHCP_run() SHOULD BE periodically called by your main task for the leased IP to maintain or check timeout.
DHCP_stop() can be called when DHCP doesn't process any more.
The below code shows how to use DHCP APIs.

DHCP_main.c

```
  /*******************************/
  /* WIZnet W5500 Code Examples  */
  /* DHCP test                   */
  /*******************************/
  /* Main loop */

  // must be set the default mac before DHCP started.
  setSHAR(gWIZNETINFO.mac);

  DHCP_init(SOCK_DHCP, gDATABUF);
  // if you want different action instead default ip assign, update,
conflict,
  // if cbfunc == 0, act as default.
  reg_dhcp_cbfunc(my_ip_assign,my_ip_assign,my_ip_conflict);

  prevTick = msTicks;

  while(1)
  {
      switch(DHCP_run())
      {
      case DHCP_IP_ASSIGN:
      case DHCP_IP_CHANGED:
         /* If this block empty, act with default_ip_assign &
default_ip_update  */
         //
         // This example calls the registered 'my_ip_assign' in the two
case.
         //
         // Add to ...
         //
         break;
      case DHCP_IP_LEASED:
         //
         // TO DO YOUR NETWORK APPs.
         //
         break;
      case DHCP_FAILED:
         /* ===== Example pseudo code =====  */
         // The below code can be replaced your code or omitted.
         // if omitted, retry to process DHCP
         my_dhcp_retry++;
         if(my_dhcp_retry > MY_MAX_DHCP_RETRY)
         {
         #if DEBUG_MODE != DEBUG_NO
            printf(">> DHCP %d Failed\r\n",my_dhcp_retry);
         #endif
            my_dhcp_retry = 0;
            DHCP_stop();      // if restart, recall DHCP_init()
            network_init();   // apply the default static network
         }
```

```
            break;
        default:
            break;
        }
    }
```

DNS

DNS example runs with one UDP socket. When you want to translate the domain name to IP address, it is useful.

To test the example,
Most of all, your board IP address SHOULD BE the public IP address to be accessible on Internet.

If you want to monitor or display message of DNS processing, Define DEBUG_MODE in bsp_tg840.h (refer to the above description *“How to test»Monitor & Debug”.*) and Define _DNS_DEBUG_ in [ioLibrary»Internet»DNS»dns.h].

dns.h

```
/*
 * @brief Define it for Debug & Monitor DNS processing.
 * @note If defined, it dependents on <stdio.h>
 */
//#define _DNS_DEBUG_
```

1. Reset the board for starting DNS processing
2. DNS running
   - Send DNS query to DNS server
   - Receive DNS response from DNS server
3. When success
   - Available the translated IP address on your network application.
4. When fail
   - No DNS response from DNS server during the max wait time as **MAX_DNS_RETRY** X **DNS_WAIT_TIME** seconds.
   - When DNS failed, you can retry to another DNS server.
     - Typically, 1st DNS failed, Retry to 2nd DNS.
   - No response from all DNS server
     - Check link status of board.
     - Check the network configuration to access on Internet.
     - Check the DNS IP address to valid.

MAX_DNS_RETRY & DNS_WIAT_TIME and DNS_time_handler()

For checking timeout to DNS processing, Define *MAX_DNS_RETYR* & *DNS_WAIT_TIME* in dns.h and SHOULD BE register *DNS_time_handler()* in your 1s timer handler.

The below code is the max timeout 6s (2 retry count * 3 seconds)

dns.h

```
#define   MAX_DNS_RETRY     2        ///< DNS query retry Count
#define   DNS_WAIT_TIME     3        ///< Wait response time. unit 1s.
```

The below code is example to register DNS_time_handler() in DNS_main.c

DNS_main.c

```
/***********************************************************************
****//**
 * @brief SysTick_Handler
 * Interrupt Service Routine for system tick counter

************************************************************************
******/
void SysTick_Handler(void)
{
  msTicks++;       /* increment counter necessary in Delay()*/

  ////////////////////////////////////////////////////////
  // SHOULD BE Added DNS Timer Handler your 1s tick timer
  if(msTicks % 1000 == 0)  DNS_time_handler();
  ////////////////////////////////////////////////////////
}
```

DNS Servers & Domain Name

In this example, 1st & 2nd DNS server and Domain name are as below. *gWIZNETIFNO* SHOULD BE the public IP address to access on Internet. So this example maybe failed until gWIZNETINFO be modified with IP address to be accessible on Internet.

DNS_main.c

```
/***********************************************************************
****//**
 * @brief Default Network Information

************************************************************************
******/
/* SHOULD BE available Network information for communicating to a DNS
server */
wiz_NetInfo gWIZNETINFO = { .mac = {0x00, 0x08, 0xdc,0x00, 0xab, 0xcd},
                            .ip = {192, 168, 1, 123},
                            .sn = {255,255,255,0},
                            .gw = {192, 168, 1, 1},
                            .dns = {8,8,8,8},            // Google
public DNS Server
                            .dhcp = NETINFO_STATIC };

uint8_t DNS_2nd[4]    = {168,126,63,1};      //secondary DNS server IP
uint8_t Domain_name[] = "www.google.com";    // for example domain name
```

The maximum length of domain names to be converted SHOULD BE defined by **MAX_DOMAIN_NAME** in dns.h. 'MAX_DOAMIN_MAME define' should be included the number of 'null' character (one) for End of String.

dns.h

```
/*
 * @brief Maximum length of your queried Domain name
 * @todo SHOULD BE defined it equal as or greater than your Domain name
length + null character(1)
 * @note SHOULD BE careful to stack overflow because it is allocated
1.5 times as MAX_DOMAIN_NAME in stack.
 */
#define  MAX_DOMAIN_NAME   16       // for example "www.google.com"
```

DNS_init() / DNS_run()

DNS_init() SHOULD be called before DHCP_run().
DNS_run() can be called for translating the domain name into IP address. This is blocked until DNS process is success or failed.
The below code shows how to use DNS APIs.

DNS_main.c

```
   /* DNS client initialization */
   DNS_init(SOCK_DNS, gDATABUF);

   /* DNS processing */
   if ((ret = DNS_run(gWIZNETINFO.dns, Domain_name, Domain_IP)) > 0) //
try to 1st DNS
   {
   #if DEBUG_MODE != DEBUG_NO
      printf("> 1st DNS Response\r\n");
   #endif
   }
   else if ((ret != -1) && ((ret = DNS_run(DNS_2nd, Domain_name,
Domain_IP))>0))     // retry to 2nd DNS
   {
   #if DEBUG_MODE != DEBUG_NO
```

```
        printf("> 2nd DNS Response\r\n");
    #endif
    }
    else if(ret == -1)
    {
    #if DEBUG_MODE != DEBUG_NO
        printf("> MAX_DOMAIN_NAME is too small. Should be redefined it.\r
\n");
    #endif
        led_msTick = 100;
    }
    else
    {
    #if DEBUG_MODE != DEBUG_NO
        printf("> DNS Failed\r\n");
    #endif
        led_msTick = 100;
    }

    if(ret > 0)
    {
    #if DEBUG_MODE != DEBUG_NO
        printf("> Translated %s to %d.%d.%d.%d\r\n",Domain_name,Domain_IP
[0],Domain_IP[1],Domain_IP[2],Domain_IP[3]);
    #endif
        //
        // TO DO
        //
    }
```
