Wireless Connectivity for MCU-based IoT Designs

Class 4: Wi-Fi

10/02/2017

Warren Miller

Presented by:

This Week's Agenda

10/30/17	Wireless Connectivity for IoT Designs
10/31/17	The Renesas Synergy Platform
11/01/17	BlueTooth
11/02/17	Wi-Fi
11/03/17	Cellular and More

Course Description

- This course will focus on three important wireless IoT connectivity methods- BlueTooth LE, Wi-Fi and Cellular.
- A short description of each technology will be provided, along with hands-on example implementations.
- The Renesas Synergy Platform will be used as the target for the hands-on implementations and interested students can optionally download the free software, which includes the popular ThreadX RTOS and associated networking stacks.
- Additionally, students can optionally purchase a Synergy hardware kit to test out the hands-on designs used in the course.

Today's Topics

- Wi-Fi Application Project
- Hardware
- Wi-Fi architecture
- Software flow
- Implementation example
- Resources

Wi-Fi Application Project

Required Resources

- Renesas Synergy starter kit SK-S7G2 rev 2.0 and above
- Longsys GT202 Wi-Fi module based on Qualcomm (QCA 4002 chipset) with a PMOD plug-in
- iOS or Android-based Smart Phone or Tablet (optional)
- Wi-Fi Access Point or Wi-Fi Router.
- Windows PC
- E2 studio ISDE or IAR EW for Synergy
- Synergy Software Package (SSP) or Synergy Stand Alone Configurator (SSC)
- Wi-Fi add on from Synergy Gallery

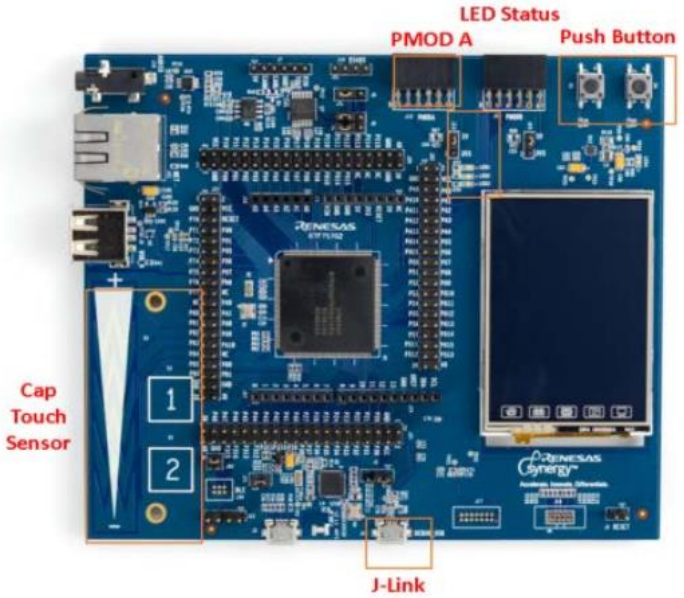
Application Project

- Typical Wi-Fi IoT application using a thermostat example
- Control thermostat operation using the smart phone via an HTTP server

Hardware

Renesas Synergy SK-S7G2 v 2.0 and above

- S7G2 microprocessor with 176 LQFP package
- Four connectors- access to all S7G2 microprocessor signals
- Low cost QVGA TFT touch screen
- Three user LEDs
- Arduino Shield Uno compatible socket
- Two mechanical switches connected directly to microprocessor interrupt pins
- Two capacitive touch-buttons connected to pins that can generate interrupts
- One capacitive slider
- Audio output
- QSPI memory (8MB)
- SPI, IIC, CAN, and SCI interface



Hardware used with application project

•Longsys GT202 Wi-Fi module

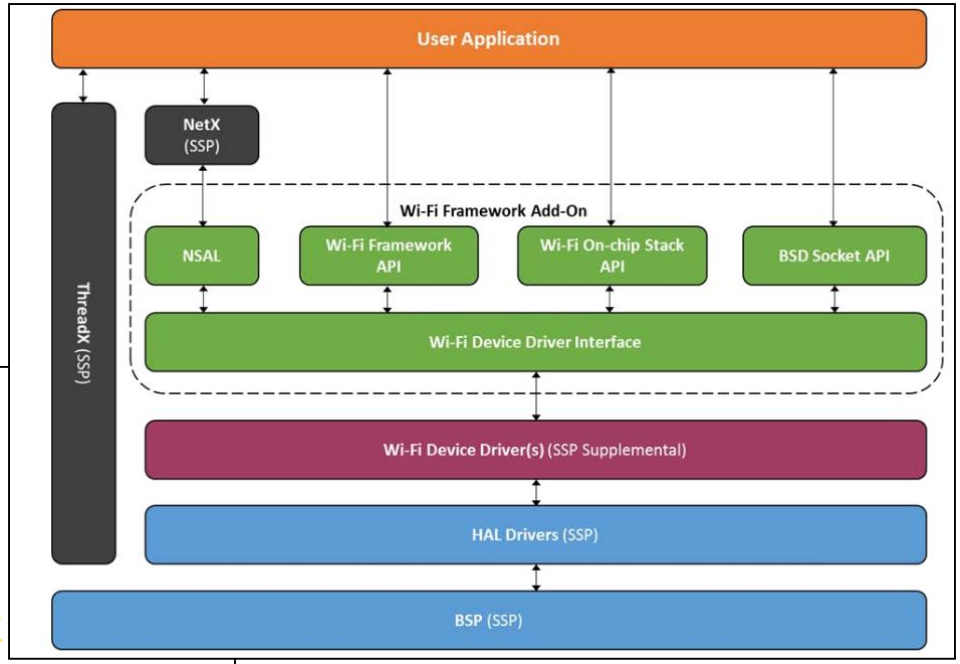
Resources

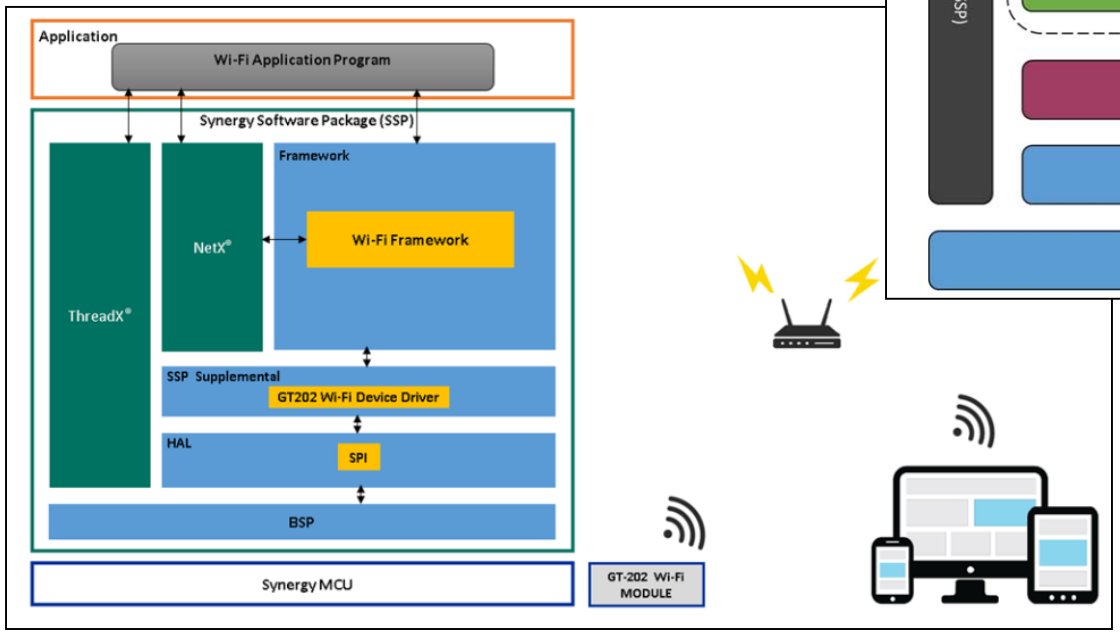
https://www.renesas.com/en-us/doc/products/renesas-synergy/doc/r12um0004eu0100_synergy_sk_s7g2.pdf

Wi-Fi Project Architecture

Wi-Fi Project Architecture

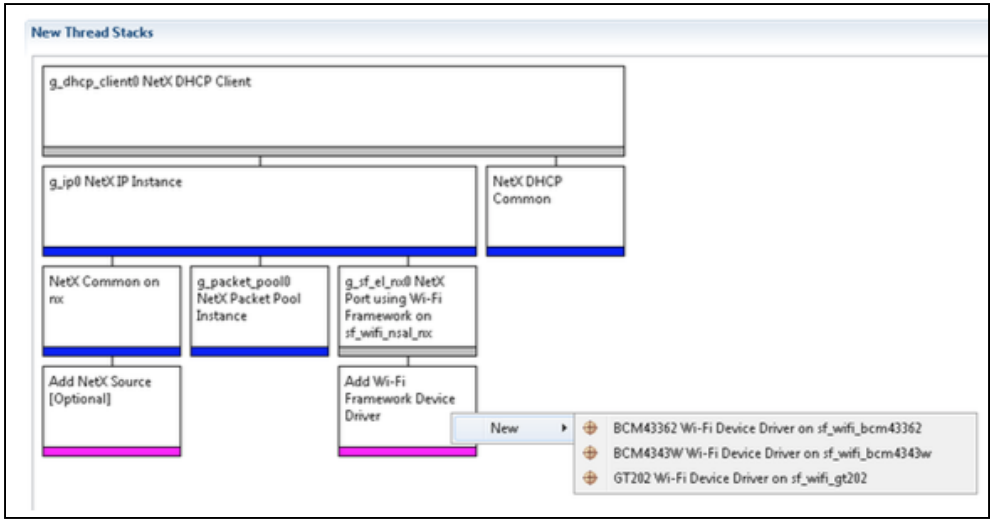
- Frameworks via Wi-Fi add-on
- ThreadX
- NetX
- Drivers
- BSP





Wi-Fi Project Design Process

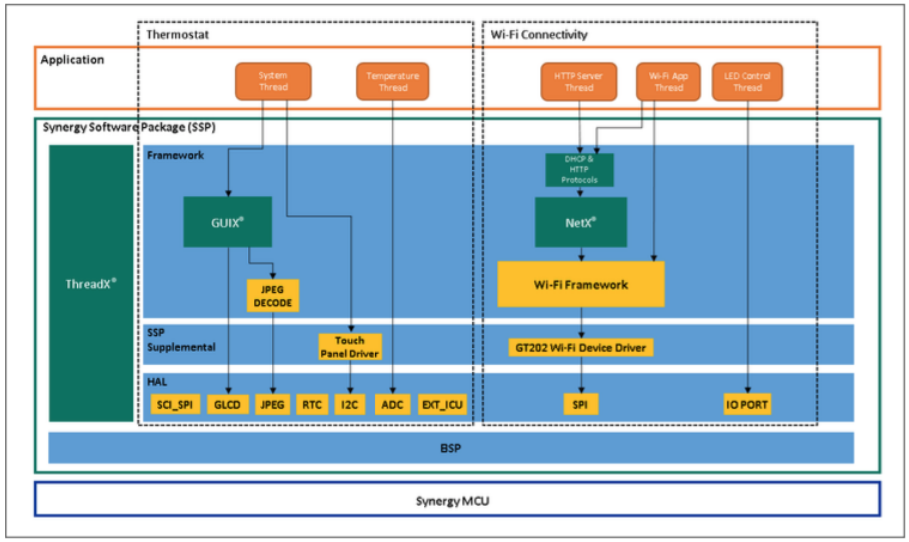
- Add DHCP Client, select lower level options
- Configure modules
- Generate code
- Add application code using APIs



ISDE Property	Setting	Description
Parameter Checking	BSP, Enabled, Disabled (Default: BSP)	Enable or disable the parameter checking.
On-Chip Stack Support	Enabled, Disabled (Default: Disabled)	On-chip stack support selection
Driver Heap Size in Bytes (Minimum 8192 bytes)	8192	Driver heap size selection
Name (Must be a valid C symbol)	g_sf_wifi0	Module name
Hardware Mode	802.11a, 802.11b, 802.11g, 802.11n (Default: 802.11n)	Hardware mode selection
Transmit (TX) Power (Valid Range 1-17)	10	Transmit power selection
Ready/Clear to Send (RTS/CTS) Flag	Enabled, Disabled (Default: Enabled)	Ready/Clear to send selection
Delivery Traffic Indication Message (DTIM) Interval (Valid Range: 1-255)	3	Delivery traffic indication message interval selection
Broadcast SSID (AP mode only)	Enabled, Disabled (Default: Enabled)	Broadcast SSID selection
Beacon Interval in Microseconds (AP mode only and must be greater than 1023)	1024	Beacon interval in microseconds selection

Function Name	Example API Call and Description
.open	g_sf_wifi0.p_api->open (g_sf_wifi0.p_ctrl, g_sf_wifi0.p_cfg); This API initializes and enables the WIFI module. The open function returns the WiFi control structure, uniquely identifying the instance of the WiFi framework.
.close	g_sf_wifi0.p_api->close(g_sf_wifi0.p_ctrl); This API un-initializes the WiFi module and powers it off.
.infoGet	g_sf_wifi0.p_api->infoGet (g_sf_wifi0.p_ctrl, wifi_info); this API takes the WiFi control structure as an argument and returns the following information obtained from the WiFi module: • Chipset/driver information string • RSSI value (unsigned integer 16 bits) • Noise level (unsigned integer 16 bits) • Link Quality (unsigned integer 16 bits)
.statisticsGet	g_sf_wifi0.p_api->statisticsGet (g_sf_wifi0.p_ctrl, p_stats); This API gets the data statistics from the WiFi module. It takes the WiFi control structure as an argument and returns the following statistics: • Received packets (unsigned integer 32 bits) • Transmitted packets (unsigned integer 32 bits) • Transmit packet errors (unsigned integer 32 bits)
.transmit	g_sf_wifi0.p_api->transmit (g_sf_wifi0.p_ctrl, p_buffer, length); This API sends the data/packet out. This function takes the WiFi control structure as an argument. It takes the network packet buffer, and the network packet buffer length as arguments. The WiFi framework transmit function passes the packet buffer to the WiFi driver for transmission.
.receiveCallback	This is a callback API, which gets called when the data/packet is received by the WiFi module. This function receives packet buffer and packet length as arguments.
.provisioningGet	g_sf_wifi0.p_api->provisioningGet (g_sf_wifi0.p_ctrl, &sf_wifi_provisioninfo); This API takes the WiFi control structure as an argument and returns the following parameters: • Mode (enumeration, that is, AP or client) • Channel (unsigned integer 8 bits) • SSID (string)

App Thread



Code is in src/wifi_app_thread_entry.c

- Brings in the DHCP client with TCP/IP core
- Brings in the Wi-Fi Framework, NSAL and

Wi-Fi driver and initializes them

1) Make sure IP link is enabled

(nx_ip_interface_status_check (&g_ip0, 0, NX_IP_LINK_ENABLED, &ip_status, NX_WAIT_FOREVER)) ;

2) Provision Wi-Fi

g_sf_wifi0.p_api->provisioningSet (g_sf_wifi0.p_ctrl, &g_provision_info);

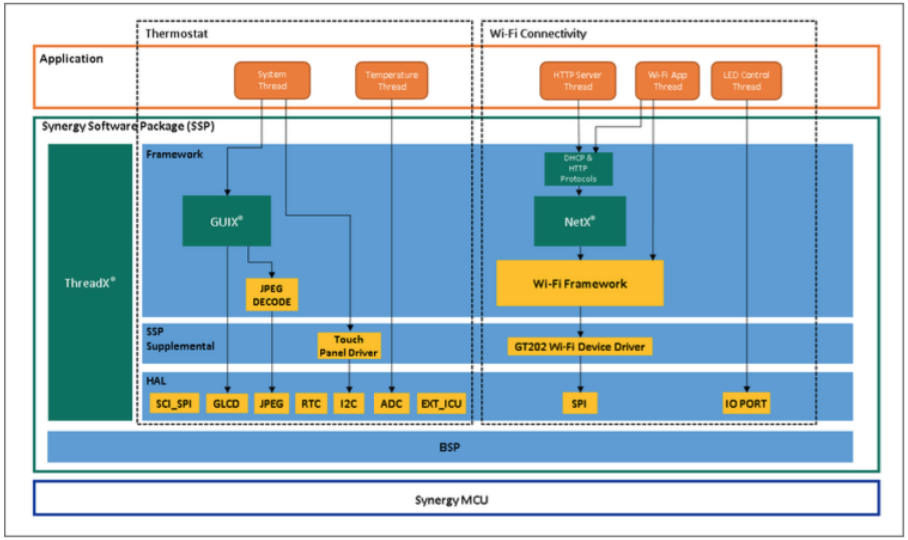
3) Start DHCP

nx_dhcp_start (&g_dhcp_client0);

4) Check that IP address is resolved and then get leased address

nx_ip_status_check(&g_ip0, NX_IP_ADDRESS_RESOLVED, (ULONG *) &status, 10);
nx_ip_interface_address_get (&g_ip0, 0, &ip0_ip_address, &ip0_mask);

HTTP Server Thread



- Code is in src/http_server_thread.c
  - Brings in the HTTP Server application

- This thread creates the HTTP server, and starts the server where the user created page will be hosted from the board.
- It also gives the option to Get/Set the user data from/to the page.
- It also sets the event flag to indicate to the LED control thread the user desired operation (ON/OFF/BLINK) for the LEDs on the board.

1) Create server

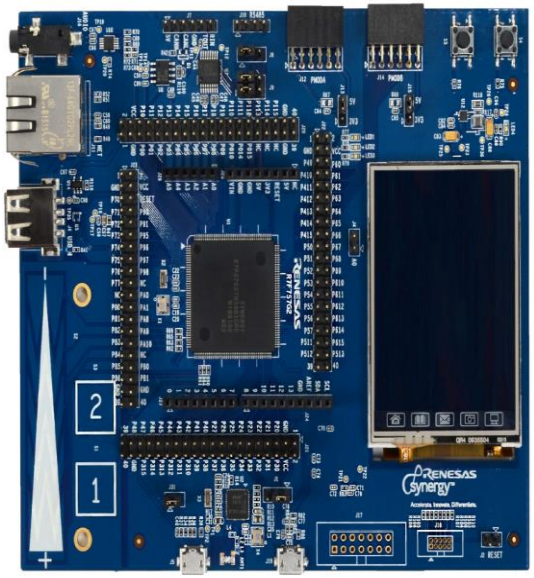
```
status = nx_http_server_create(&my_server, (CHAR*)"My HTTP Server", &g_ip0,
&ram_disk_media, &NETX_SVR, 4096, &g_packet_pool0, authentication_check, get_notify);
```

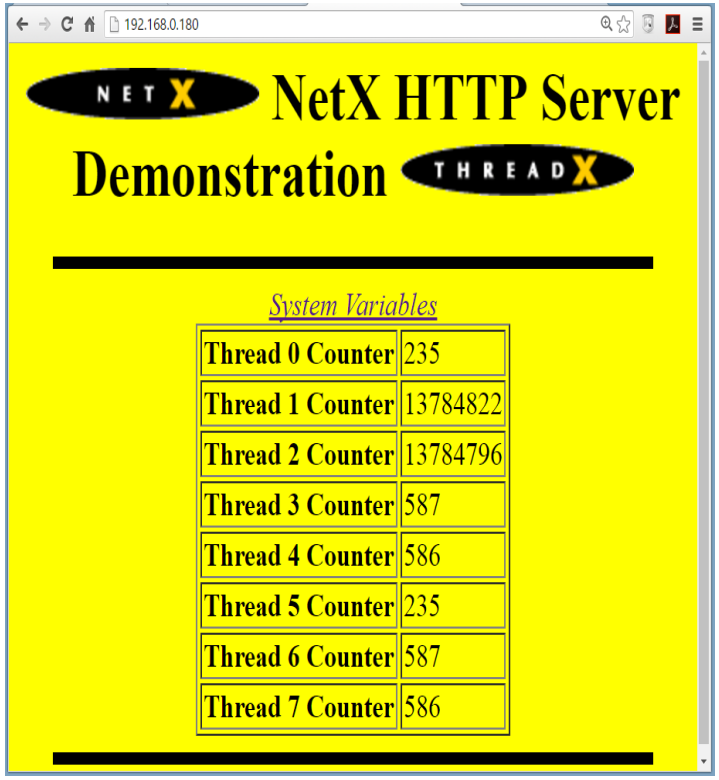
2) Start server

```
status = nx_http_server_start (&my_server);
```

Simple Web Server Design

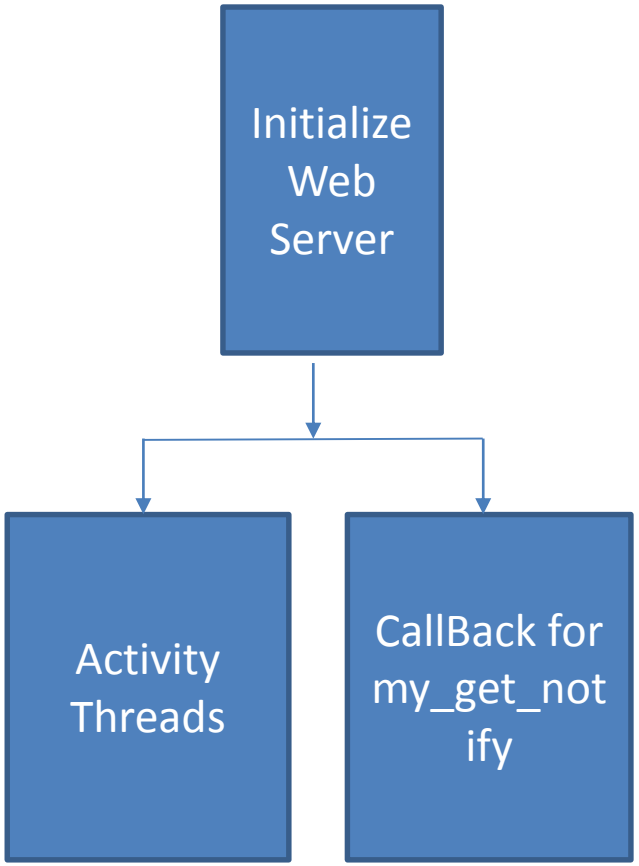
- A simple web server is a common embedded design that illustrates how easy networking designs can be with SSP
- Webserver provides view of embedded design activity
- Demonstration design uses the SK-S7G2 kit





Web Server Description

- Initialize the webserver
- Create activity threads to generate data for web page reporting
- Create a callback function (my_get_notify) to respond to resource requests
- Client requests a resource
    - Usually an html file or graphics file
    - Data can be generated on the fly or read from a file system
    - In this simple example graphics files are stored as an array of bytes and html code generated by the program



Implementation

Initialize Web Server

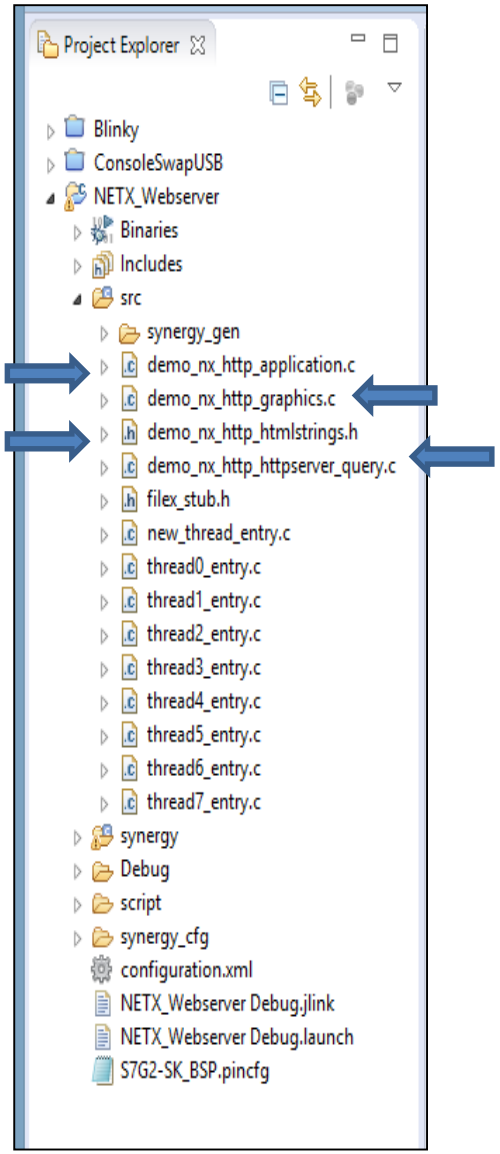
application.c creates the webserver:

- nx_system_initialize: Initialize NetX TCP/IP stack
- nx_packet_pool_create: Create packet pool
- nx_ip_create: Create IP instance
- nx_ip_fragment_enable: Enable fragmentation
- nx_arp_enable: Enable SRP
- nx_icmp_enable: Enable 'ping'
- nx_http_server_create: Create http server instance
- nx_http_server_start: Starts the server instance

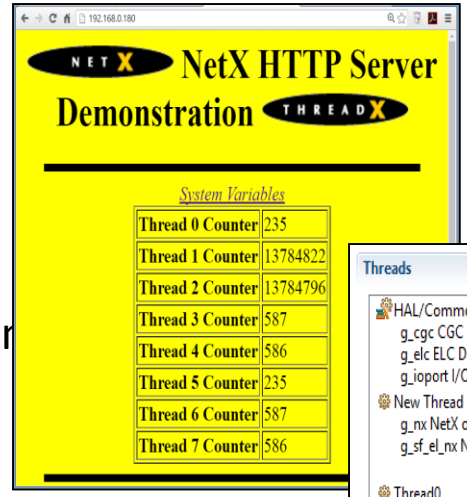
graphics.c holds the logo hex files

htmlstrings.h holds the html string defines

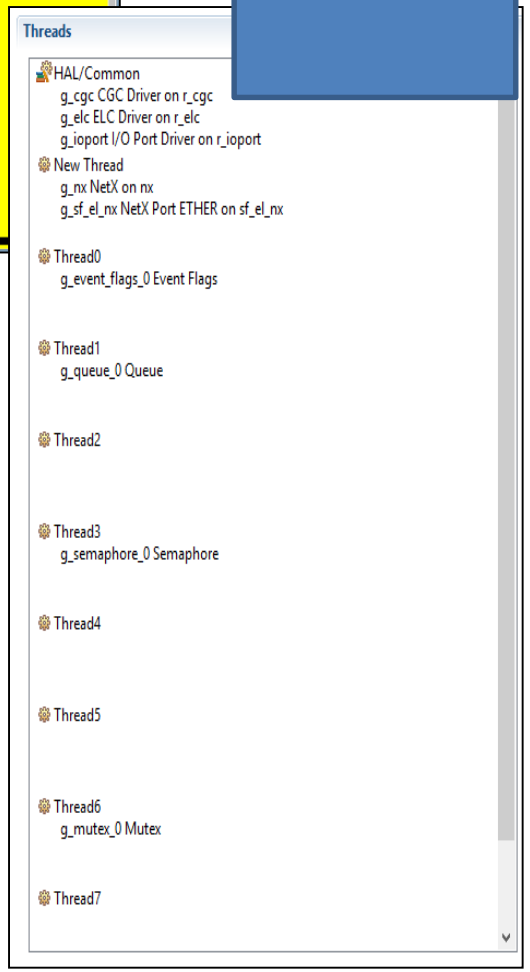
httpserver_query.c holds code to handle URL requests



Metrics



Activity Threads

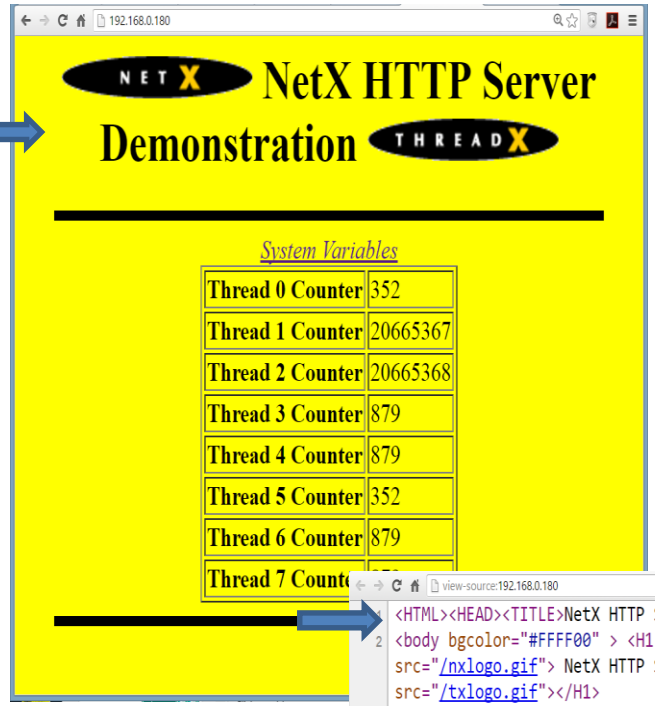


The simple webserver reports counter values fro several activity threads

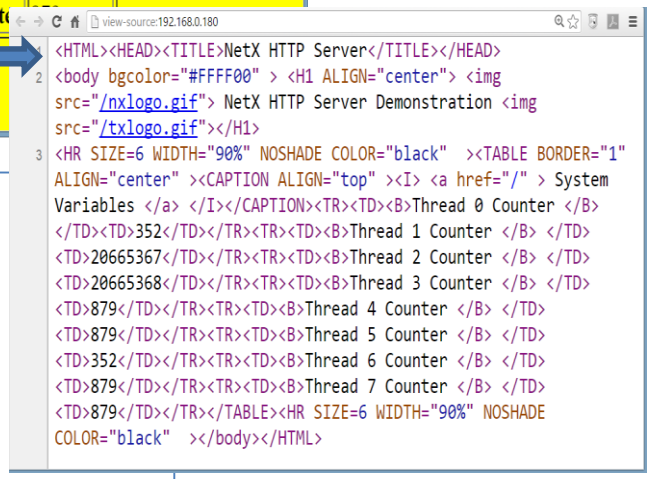
- Thread 0: Status flag to wake Thread 5, Sleep 10
- Thread 1: Sends message to queue 0
- Thread 2: Receive message from queue 0
- Thread 3: Get semaphore, Sleep 2, Release
- Thread 4: Get semaphore, Sleep 2, Release
- Thread 5: Wait for event flag
- Thread 6: Get mutex, Sleep 2, Release
- Thread 7: Get mutex, Sleep 2, Release

Reporting

- httpserver_query.c serves up resource requests via my_get_notify
- Creates packet via packet_allocate()
- Write html into the packet via htmlwrite()
- Send packet via tcp_socket_send



CallBack for my_get_notify

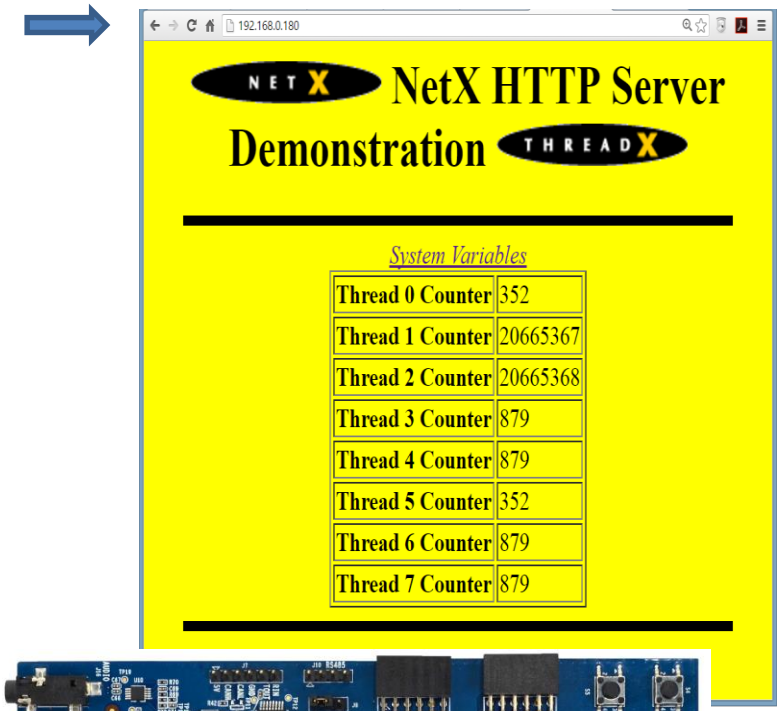


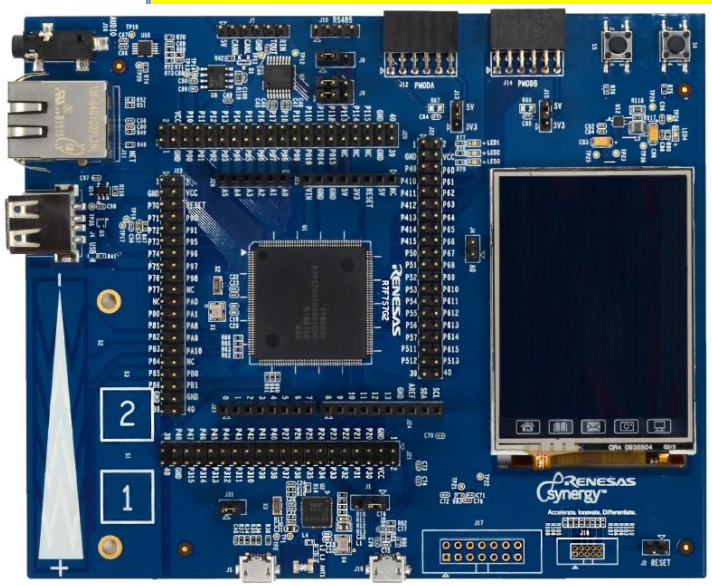

```
68          /* return requested resource or query */
69          if(strcmp((const char*)resource,(const char*)"/")==0)
70          {
71
72            /* obtain a packet for our html code to be sent to the client */
73            status += nx_packet_allocate(server_ptr -> nx_http_server_packet_pool_ptr,
74                                         &resp_packet_ptr,
75                                         NX_TCP_PACKET,
76                                         NX_WAIT_FOREVER);
77
78            /* write HTML code into the packet */
79            /* htmlwrite(p,s)  (nx_packet_data_append(p,s,strlen(s), server_ptr-> nx_http_server_packet_pool_ptr,NX_WAIT_FOREVER))
80            status += htmlwrite(resp_packet_ptr, htmlresponse);
81            status += htmlwrite(resp_packet_ptr, htmltag);
82            status += htmlwrite(resp_packet_ptr, titleline);
83            status += htmlwrite(resp_packet_ptr, bodytag);
84            status += htmlwrite(resp_packet_ptr, h1line);
85            status +=  nx_tcp_socket_send(&(server_ptr -> nx_http_server_socket), resp_packet_ptr, NX_HTTP_SERVER_TIMEOUT);
86
```

```
#define titleline "<HEAD><TITLE>NetX HTTP Server</TITLE></HEAD>\r\n"
```

Demonstration

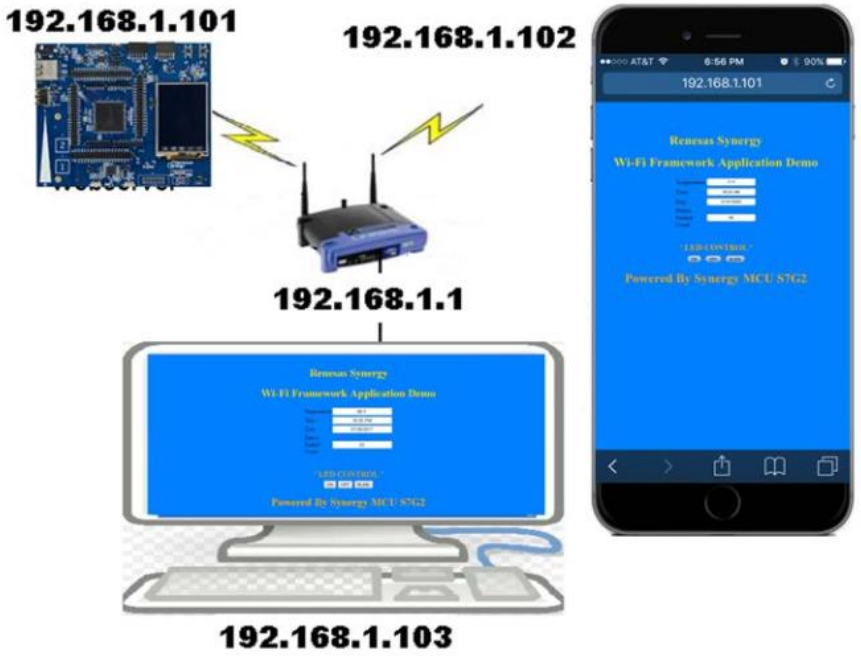
- Set-up PC host
- Connect Ethernet Cable
- Ping
- Enter server address
- Refresh web page
- See metrics change
- Compare metrics

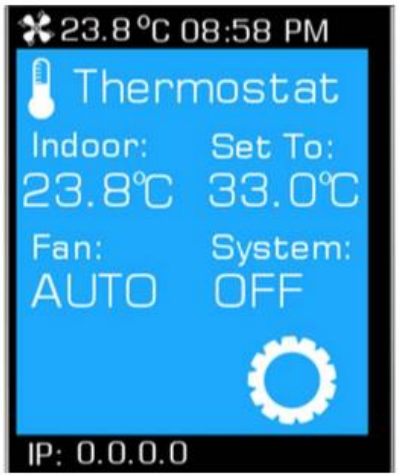


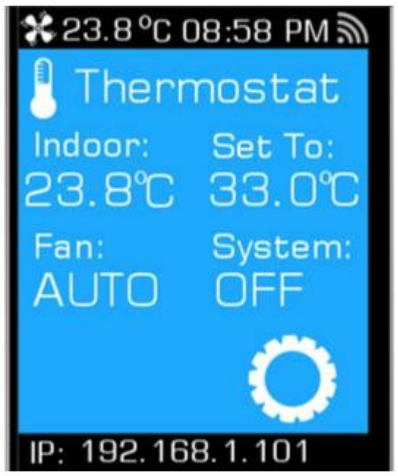


Wi-Fi Application Set-up

- Connects to Smartphone via Router/Access Point
- Web server running on the board can be accessed by browser
- Need to modify User Credentials

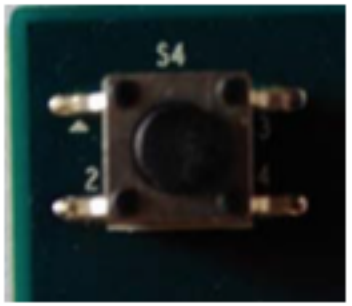




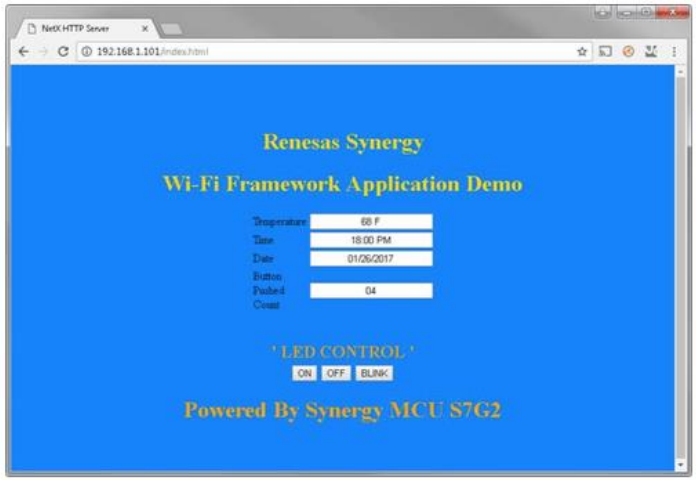


Wi-Fi Application Running

- Can access web page
- Can control LEDs
- Can press button and the web page reports the number of button presses

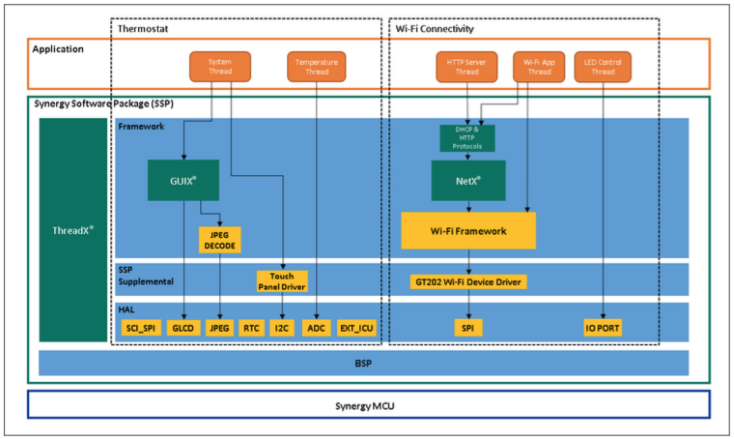


Class Resources



- Synergy Wi-Fi Project

https://www.renesas.com/en-us/doc/products/renesas-synergy/apn/r11an0082eu0101-synergy-wifi-sk-s7g2.pdf



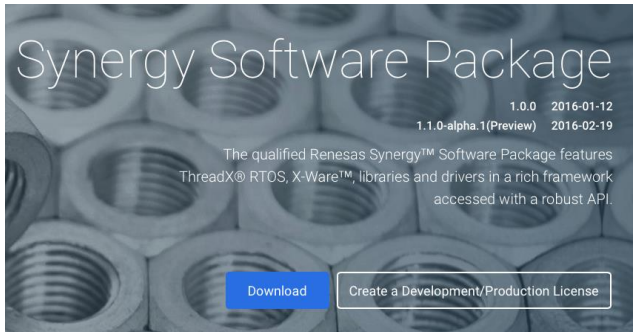
- Other CEC Courses

https://www.designnews.com/continuing-education-center/may-16-day-1-introduction-rtos-concepts

https://www.designnews.com/continuing-education-center/july-26-day-2-implementing-networking-connectivity-using-rtos-and-apis-part-1

Course Resources

- Express Logic Web Site: www.rtos.com
- Express Logic Articles and White papers Here
- Express Logic RTOS Book (Amazon) Here



- Renesas Synergy Platform Kits at Digi-Key Here
- Course Kit Resources Here
- Renesas Synergy Platform Here
- Renesas Synergy Gallery (https://synergygallery.renesas.com/auth/login)

This Week's Agenda

10/30/17	Wireless Connectivity for IoT Designs
10/31/17	The Renesas Synergy Platform
11/01/17	BlueTooth
11/02/17	Wi-Fi
11/03/17	Cellular and More