mxchipWNet Library

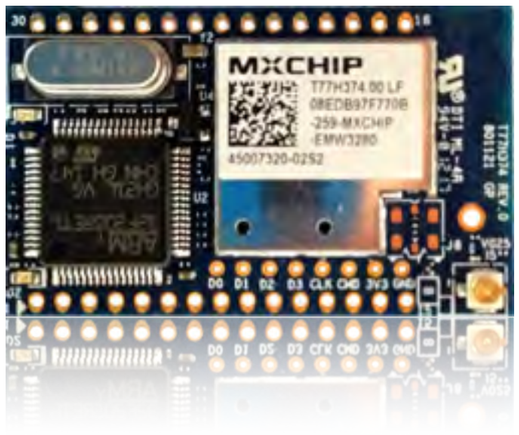
Development guide for the basic version

Agenda

- Introduction for the mxchip**WNet**™ library basic version
- Allocation of the hardware resources
- Development tools and environment
- Guide of the API functions
- Demo applications

mxchipWNet Authorized Platform

EMW3280



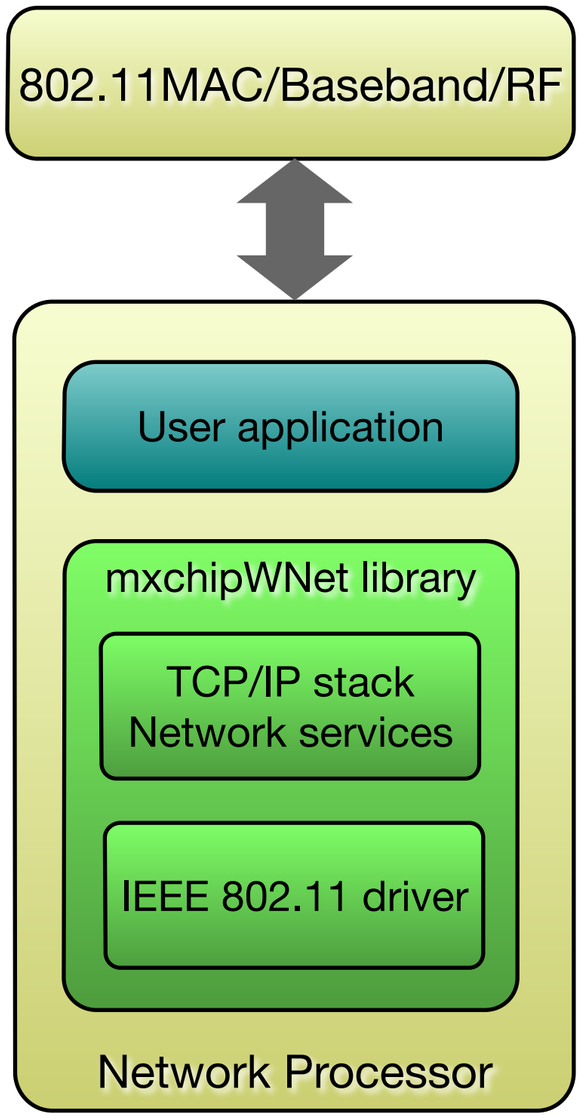
EMW3161

EMW3162

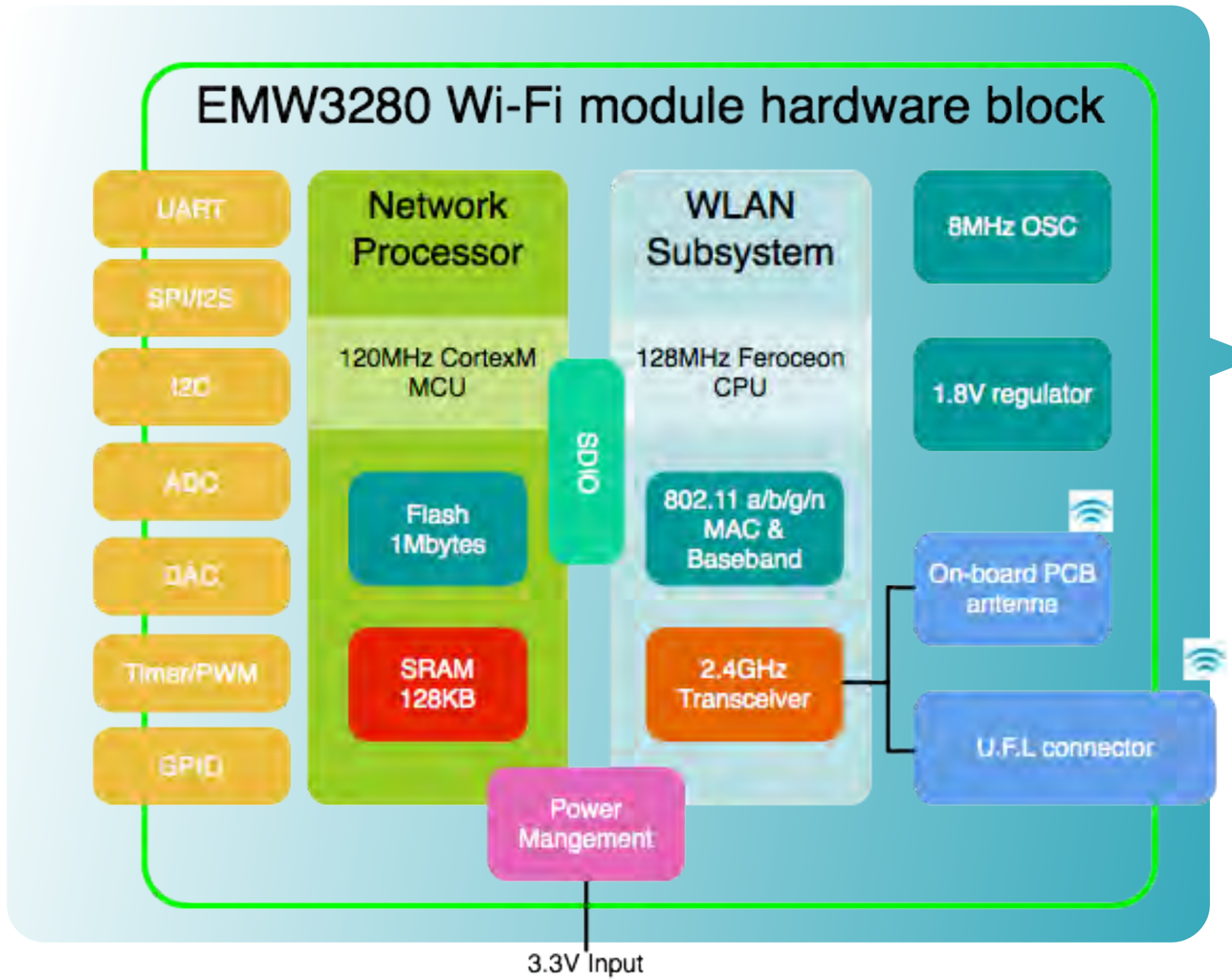
More in future…

mxchipWNet library can only run on a mxchipWNet authorized platform.

mxchipWNet Architecture



MXCHIP Wi-Fi module

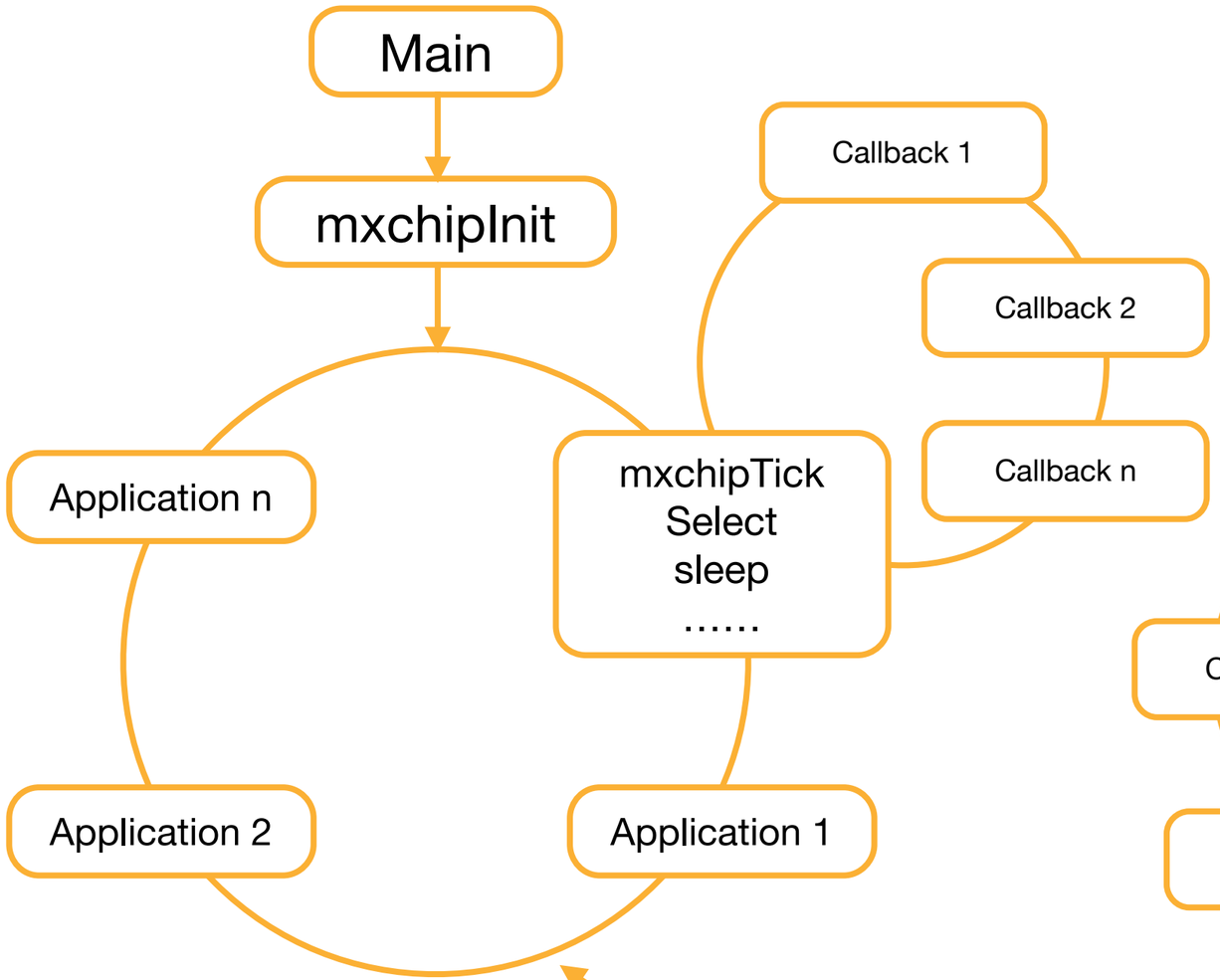


mxchipWNet Functions

- Wi-Fi Features:
  - IEEE 802.11 b/g/n driver
  - Station mode, soft AP mode and coexistent
  - WEP, WPA/WPA2 encryption
  - WPS, Easylink configuration
  - IEEE 802.11 roaming, power save mode
- TCP/IP stack Features:
  - TCP client/server, UDP unicast/multicast/broadcast
  - Block/Unblock mode
  - DHCP server/client
  - DNS, mDNS
  - TLS/SSL data encryption
  - Flexible memory allocation
- Miscellaneous:
  - A build-in RTOS (profession version only)
  - System timer
  - MD5, AES, SHA algorithm
  - Demo applications
  - ……

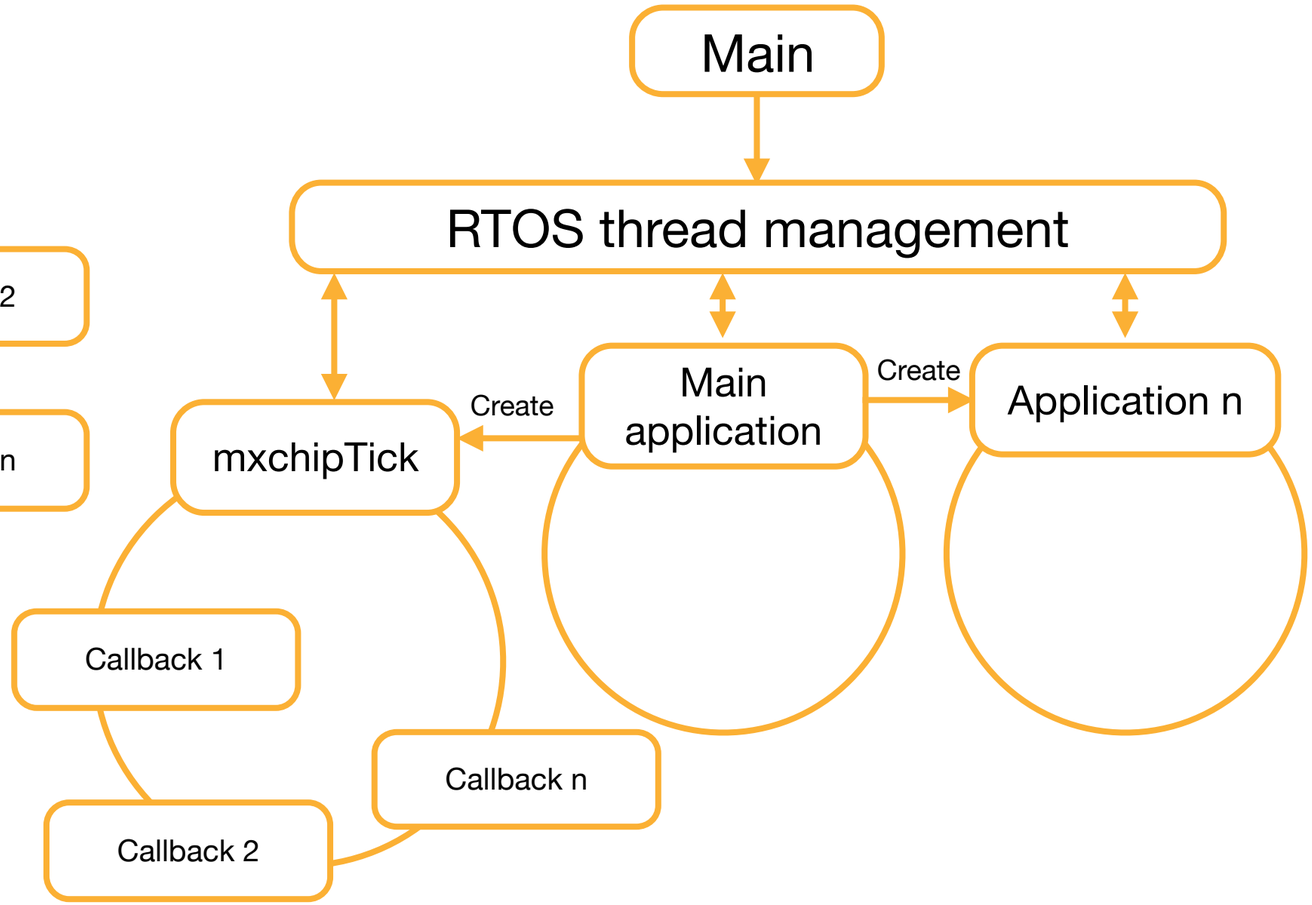
mxchipWNet Application Running Mode

Basic version



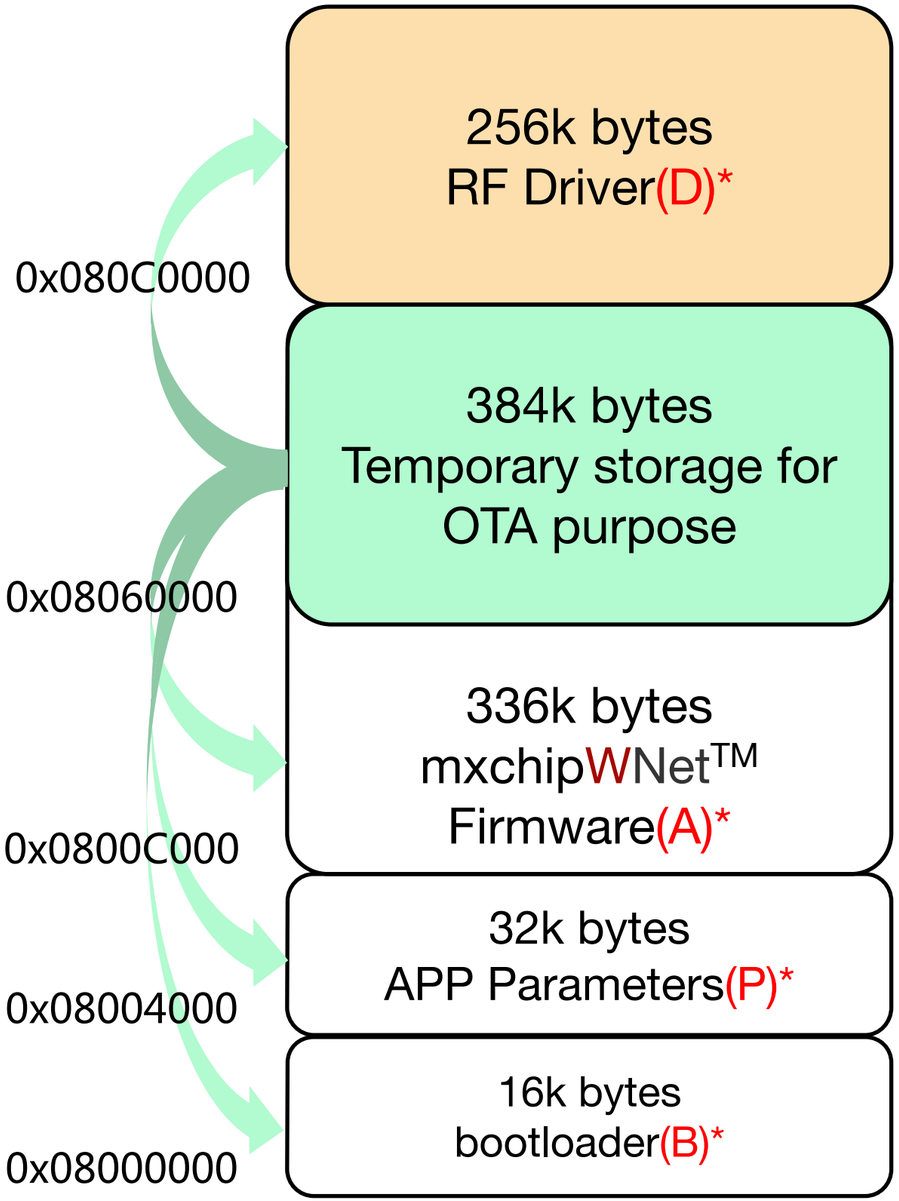
! The main loop should less than 500ms

Professional version (RTOS)



Do not execute any library function in a callback

mxchipWNet Flash Memory Map



* Content Type

Contents should not be changed

OTA function can be disabled in bootloader

Start	End	Type	Size (bytes)	Content
0x08000000	0x08003FFF	B	16k	Bootloader
0x08004000	0x0800BFFF	P	32k	OTA info，user参数
0x0800C000	0x08060000	A	336k	User application
0x08060000	0x080C0000	–	384k	OTA storage
0x080C0000	0x080FFFFF	D	256k	RF Driver

OTA Procedure

OTA steps

1. Download update data to OTA storage (User)
2. Write OTA info to 0x08004000 (User)
3. Reboot (User)
4. Bootloader update the target flash memory using update data (Bootloader)
5. Bootloader clear the update data and OTA info (Bootloader)
6. Start the application (Bootloader)

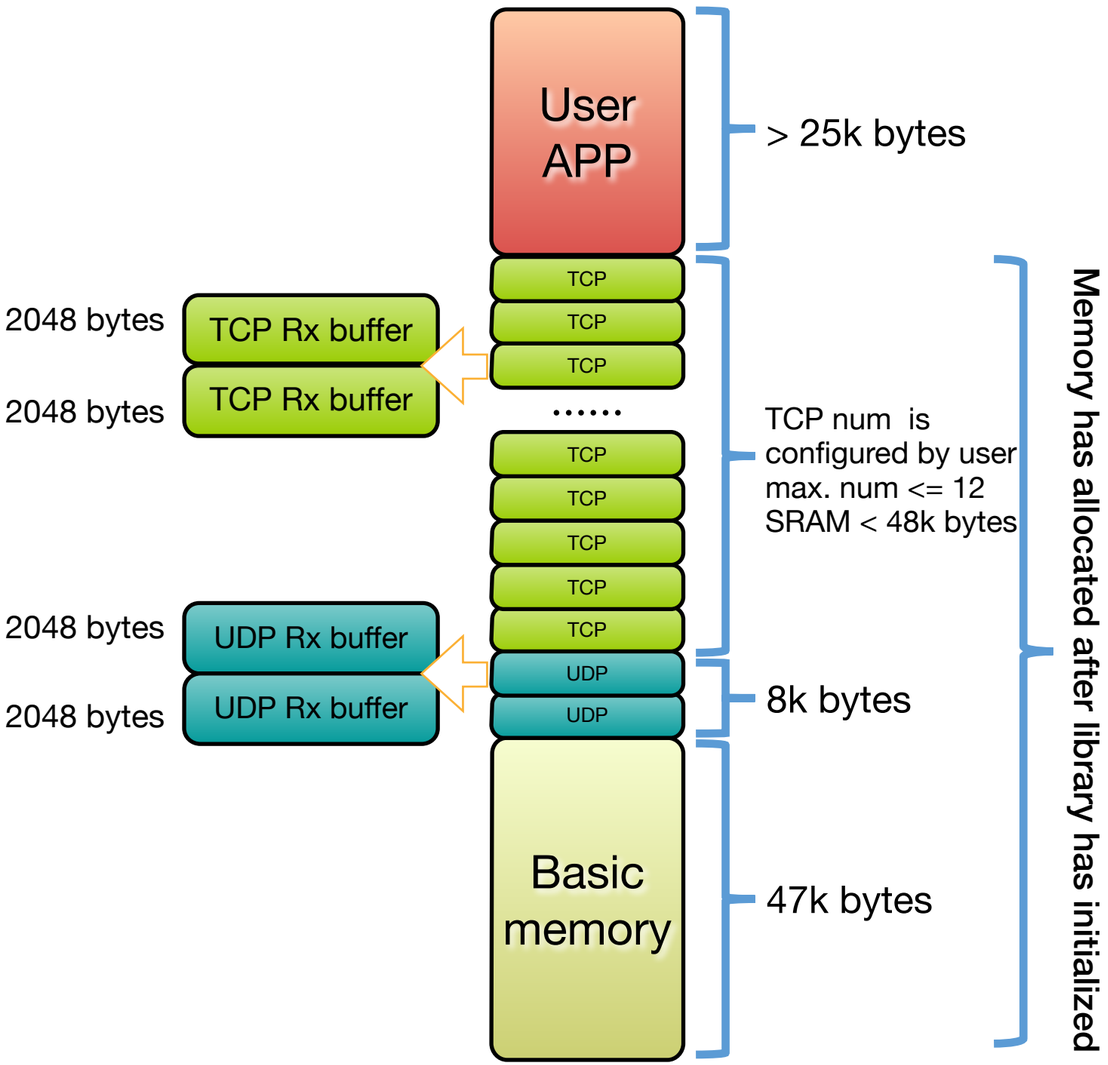
OTA info @ 0x08004000

Name	Data Type	Data Length	Content
START ADDRESS	Word	1	OTA data storage address (should be 0x08060000 only now)
LENGTH	Word	1	OTA data length
VERSION	Byte	8	Version（Not used）
TYPE	Byte	1	Target content type (’B’,’P’,’A’,’D’)
UPDATE	Byte	1	Update tag(‘U’)
REVERSED	Byte	6	Reserved

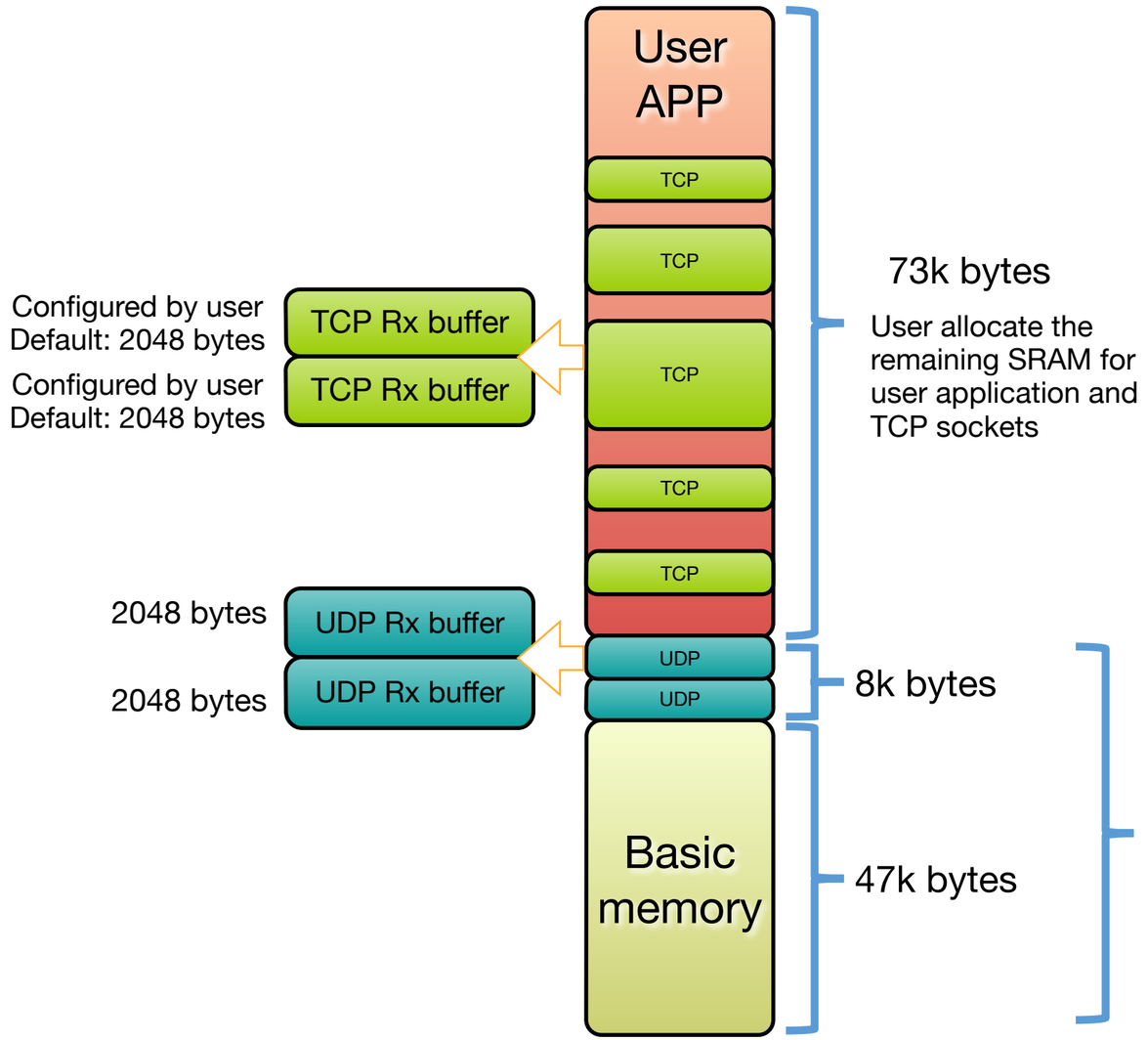
Target content type: Which data block should be updated by the new data in OTA data storage

mxchipWNet SRAM Allocation

Static allocation



Dynamic allocation



Used EMW316x Peripherals by Library

After mxchipWNet library is initialized

Peripheral	Function	Module	Note
SystemTick and interruption	Generate the time base for the library, one tick: 1ms	All	
SDIO	Communicate with RF chip	All	
DMA2 Stream 3	Communicate with RF chip	All	
EXTI Line13	Communicate with RF chip	EMW3162	
PC2	Watch dog signal output	EMW3162	Enabled by user
EXTI Line0	Communicate with RF chip	EMW3161	
PB14	Watch dog signal output	EMW3161	Enabled by user

Use UART functions

Peripheral	Module
UART1	EMW3162
PA8, PA9	EMW3162
DMA2 Stream 2	EMW3162
UART2	EMW3161
PA0, PA1	EMW3162
DMA1 Stream 5	EMW3161

Use NFC functions

Peripheral	Module
IIC1	All
PB6, PB7, PB14	EMW3162
EXTI Line14	EMW3162
PB6, PB7, PB15	EMW3161
EXTI Line15	EMW3161

Development Environment

Hardware Connection

Only RVMDK is supported now

EMW3162

EMB-380-S2

BOOT(SW1)	STATUS(SW2)	Operation mode
L	L	Factory mode
L	H	Firmware update mode
H	L/H	Working mode

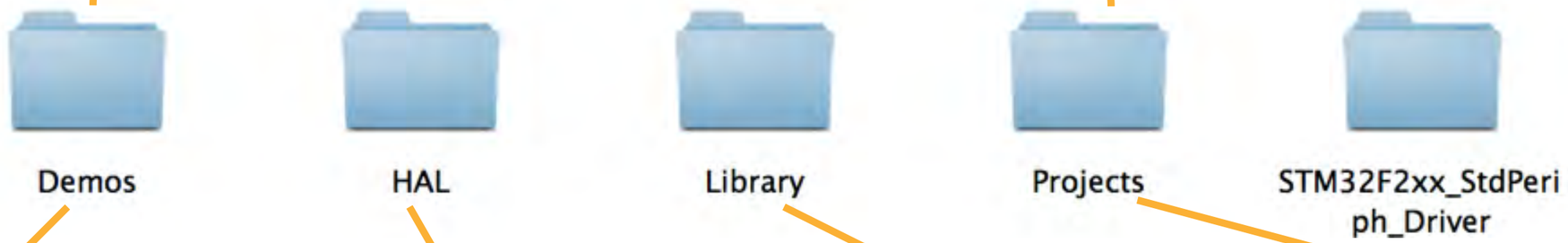
Prepare Develop Software

- USB driver
  - VCP driver: Virtual com port over USB
    - http://www.ftdichip.com/Drivers/VCP.htm
  - JTAG debugger driver: Jlink or Ulink
- IDE
  - Library and demos are tested under Realview MDK 4.7 5.0
- COM terminal
  - SecureCRT or HyperTerminal
- Network analyzer
  - Wireshark

mxchipWNet library package content

From MXCHIP

Standard peripheral library from MCU vender(ST…)



- Demo1_WiFi_Link
  - main.c
  - main.h
  - stm32f2xx_conf.h
  - stm32f2xx_it.c
  - stm32f2xx_it.h
- Demo2_TCP_UDP_ECHO
- Demo3_WPS_EasyLink
- Demo4_Webserver_OTA
- Demo5_mDNS_bonjour
- Demo6_SSL_https
- Demox_mxchipWNet_HA
- Demox_WirelessCamera

Demo source codes

- flash_if.c
- flash_if.h
- HAL_EMW3161.h
- HAL_EMW3162.h
- hal_misc.c
- platform.c
- platform.h
- stm32f2xx_conf.h
- system_stm32f2xx.c
- uart.c

Hardware layer source codes

- Flash operation
- UART total solution
- ……

- mxchipWNET.h
- mxchipWNet_3161.lib
- mxchipWNet_3162.lib
- mxchipWNet_TypeDef.h

mxchipWNet™ library

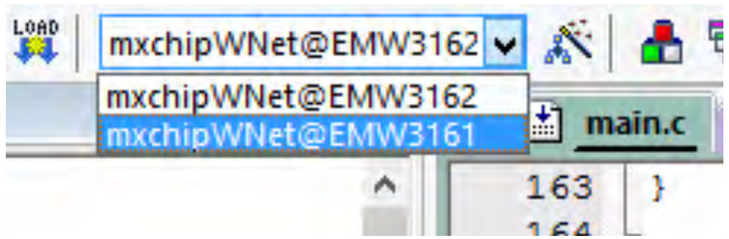
Open all demo projects from here

- Demo1_WiFi_Link
  - EWARM
  - Output
  - RVMDK
- Demo2_TCP_UDP_ECHO
- Demo3_WPS_EasyLink
- Demo4_Webserver_OTA
- Demo5_mDNS_bonjour
- Demo6_SSL_https
- Demox_mxchipWNet_HA
- Demox_WirelessCamera
- Demos.eww
- Demos.uvmpw

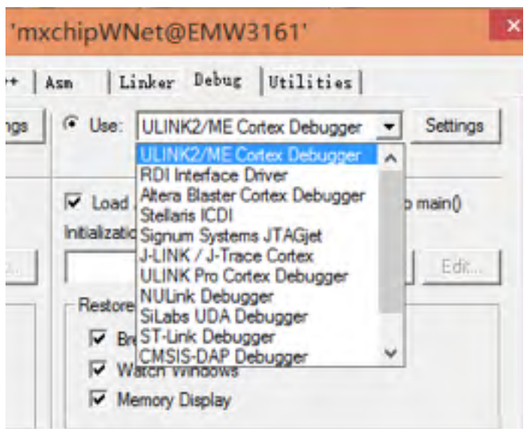
Demo project for different IDE

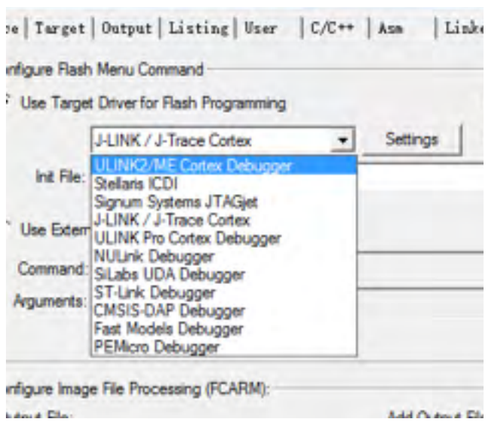
Development Environment - RVMDK Settings

1. Select a correct build target

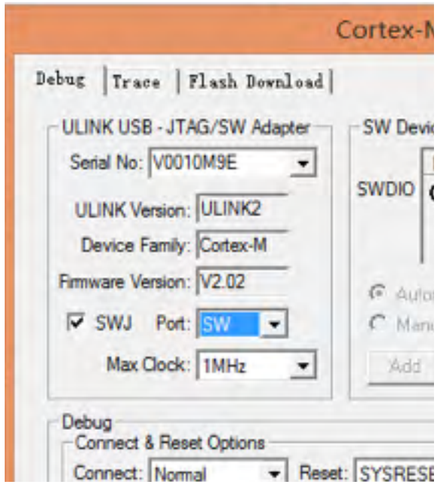


2. Select a correct debugger

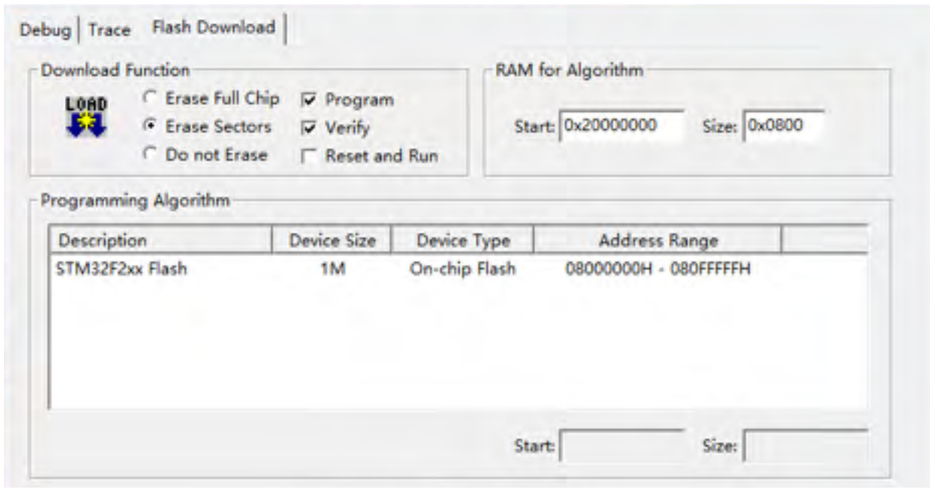




3. Select a correct debug protocol (SW)

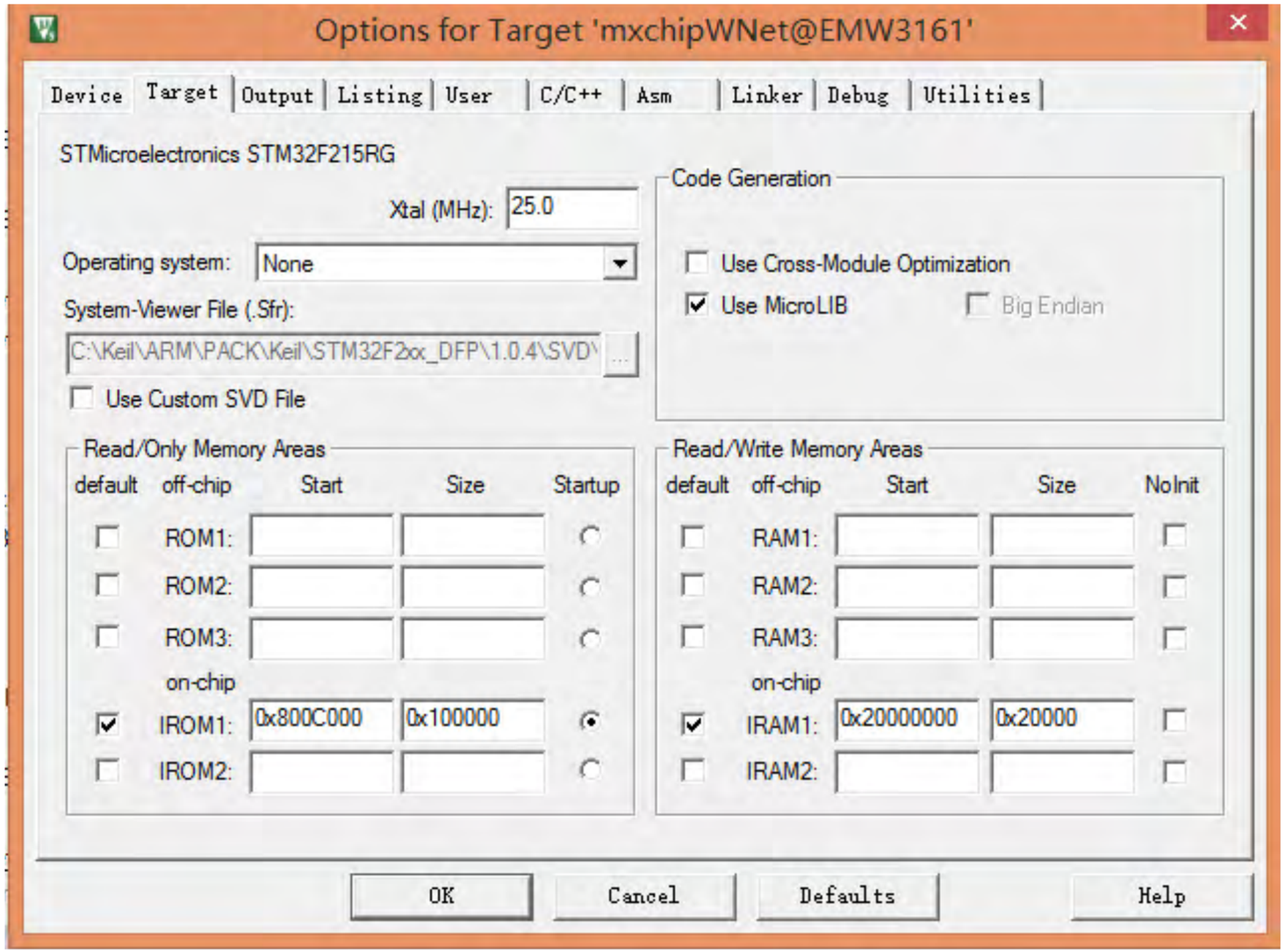


4. Select a correct flash algorithm



Development Environment - RVMDK Settings

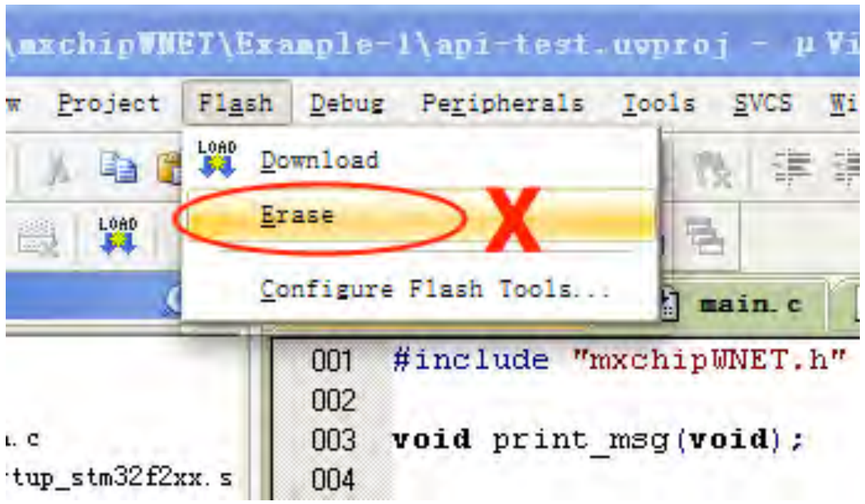
5. Change the ROM base address to 0x0800C000:

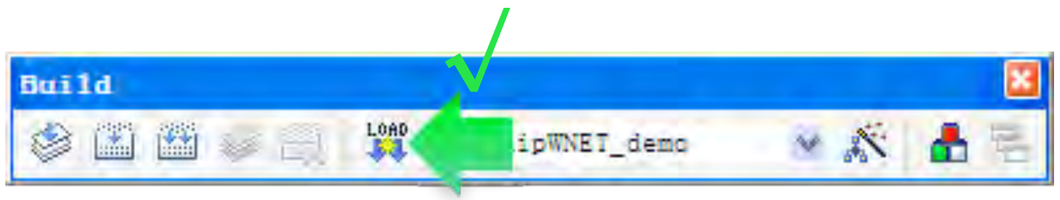


Caution When Download your APP

No not execute any instructions which may erase the whole flash memory!

Erase the full chip may also erase the bootloader and RF driver





Safe download:
Only use download function to download your application

mxchipWNet™ Library Basic Version

- mxchipWNet library (basic version) is written using standard C constructs and compiled with the ARM RealView Compiler and IAR Compiler.

- mxchipWNet library (basic version) designed as a stand alone TCP/IP stack and Wi-Fi driver and does not require an RTOS kernel to run.

- The mxchipWNet (basic version) functions do not require RTOS to run.

- All of mxchipWNet library functions are not reentrant.

mxchipWNet Library API Catalog

mxchipWNet library runtime

- library global definition
- Library initialization
- Library maintenance in runtime

Wi-Fi control and Network services

- Scan available hotspots
- Connect/disconnect Wi-Fi network
- Wi-Fi power management
- WPS, Easylink
- DNS, DHCP

BSD socket APIs

- TCP/UDP communication
- UDP multicast
- TLS/SSL encryption

Tools

- External interruption configuration
- Data format conversion
- Memory debugger
- MD5, AES…

mxchipWNet Library Runtime

mxchipWNet™ Configuration

Define the library's running parameters

void lib_config(lib_config_t* conf)

- input: RAM allocation mode, watchdog output, max num of TCP sockets……

mxchipWNet™ Initialization

Initialize the **mxchipWNet™** library. This should be execute at the beginning.

MxchipStatus mxchipInit(void);

- Return: Success or not

mxchipWNet™ function block

Processes **TCP/IP stack and Wi-Fi driver** operations. The mxchipTick functions are the core of the protocol stack and generate the corresponding callbacks.

Some other library functions can also execute the same function block besides their main functions like select, sleep, msleep, send, recv……

void mxchipTick(void)

- Input: None
- Return: None

Scan Available Hotspots

Scan nearby Wi-Fi Networks

MxchipStatus mxchipStartScan(void)

Module start scan all nearby APs in all supported channels

Return scan results (callback)

void mxchipScanCompleteHandler(UwtPara_str *pApList)

This is a callback function. Return the scan results

- Input: Address to store **UwtPara_str**

- Return : none

```
typedef  struct  _UwtPara_str
{
  char ApNum;
  ApList_str * ApList;
} UwtPara_str;
```

```
typedef  struct  _ApList_str
{
    char ssid[32];
    u8 signal;  // 0-100
}ApList_str;
```

WLAN Connection

Start a Wi-Fi network connection

Start a network connection according a given network profile, library will try to connect to this network periodically before connect is established.

User can call this function twice to enter coexistence mode: station mode + Soft AP mode

int StartNetwork(network_InitTypeDef_st* pNetworkInitPara)

- input: pNetworkInitPara - network profile

Parameter Name	Note	Parameter Name	Note
Wi-Fi mode	Soft AP or Station	**Gateway IP address**	Use in static address
SSID	Name of Wi-Fi network	**DNS server IP address**	Use in static address
KEY	Encryption type is self-adaption	**DHCP mode**	DHCP server/client
Local IP address	Use in static address mode	**Address pool start**	
Net mask	Use in static address mode	**Address pool end**	
		Retry interval	Default: 100ms

Return Wi-Fi status (callback)

This function will be called by library once the Wi-Fi status is changed.

void WifiStatusHandler(int status)

- Input: the new Wi-Fi status— Station up/down Soft AP up/down

- Return : none

Wi-Fi Configuration

Start a Wi-Fi configuration procedure

Start WPS, Easylink, NFC, BT procedure for Wi-Fi configuration.

int OpenConfigmodeWPS(int timeout)

int OpenEasylink(int timeout)

- input: configuration timeout

Return Wi-Fi parameters (callback)

This function will be called by library once the configuration procedure is success or failed.

void RptConfigmodeRslt(network_InitTypeDef_st *nwkpara);

- Input: point to Configuration result, =0 if failed

- Return : none

Parameter Name	Note
SSID	Name of Wi-Fi network
KEY	Encryption key

Check Current Wi-Fi Link Status and Others

Check Wi-Fi link status

We are planing to return more information about the current connected AP.

int CheckNetLink(sta_ap_state_t *ap_state);

- input: point to the structure of the current AP status

Parameter Name	Note	Parameter Name	Note
Connected	Connected to a AP or not	**SSID**	Current connected AP's ssid
Signal strength	Current signal strength, 0-100	**BSSID**	Current connected AP's bssid

IEEE power save mode

void ps_enable(void);

void ps_disable(void);

Disconnect wlan

int wlan_disconnect(void); **int sta_disconnect(void);**

int uap_stop(void);

RF power on/off

int wifi_power_down(void);

int wifi_power_up(void);

DNS Service

Block mode

Start DNS resolving and read its result

The function will return if DNS is success or time out(5 seconds).

int gethostbyname(const u8 * hostname, u8 * ip_addr, u8 ipLength)

- Input: domain name—“www.google.com”

ip_addr, ipLength—the memory to store the returned address

- Return: Success or not

Unblock mode

Start DNS resolving procedure

The function will return immediately.

u32 dns_request(char *hostname);

- Input: domain name—“www.google.com”

- Return: -1: Start DNS service failed

0: Start success, waiting for callback

>0: The IP address (The host name is already a ip address or the result is stored in the library form a previous DNS procedure.)

Return DNS result (callback)

This function will be called by library once the DNS procedure is success or timeout (5 seconds).

void dns_ip_set(u8 *hostname, u32 ip);

- Input: A given domain name—“www.google.com”

ip : The ip address of the given domain name, if ip = -1, means the DNS is failed

DHCP Results and Current IP

Return DHCP result (callback)

This function will be called by library once the DHCP is successful.

void NetCallback(net_para_st *pnet)

- Input: pnet—point to the result of the DHCP

Parameter Name	Note	Parameter Name	Note
DHCP mode	DHCP server/client	**Gateway IP address**	
Local IP address		**DNS server IP address**	
Net mask		**MAC address**	
		Broadcast address	No function, reserved

Read current IP status

You can call this function anytime to get the current IP parameters.

int getNetPara(net_para_st * pnetpara， WiFi_Interface iface);

- input: pnetparapo – int to the memory that store the IP parameters

iface – Wi-Fi network interface: station mode or soft AP mode

mxchipWNet™ API overview: BSD Socket APIs

Use these functions to manage all data transmission and reception

Refer http://en.wikipedia.org/wiki/Berkeley_sockets for details

```
int socket(int domain, int type, int protocol);
int setsockopt(int sockfd, int level, int optname,const void *optval, socklen_t optlen);
int bind(int sockfd, const struct sockaddr_t *addr, socklen_t addrlen);
int connect(int sockfd, const struct sockaddr_t *addr, socklen_t addrlen);
int listen(int sockfd, int backlog);
int accept(int sockfd, struct sockaddr_t *addr, socklen_t *addrlen);
int select(int nfds, fd_set *readfds, fd_set *writefds, fd_set *exceptfds, struct timeval_t *timeout);
ssize_t send(int sockfd, const void *buf, size_t len, int flags);
ssize_t sendto(int sockfd, const void *buf, size_t len, int flags,const struct sockaddr_t *dest_addr,
socklen_t addrlen);
ssize_t recv(int sockfd, void *buf, size_t len, int flags);
ssize_t recvfrom(int sockfd, void *buf, size_t len, int flags,struct sockaddr_t *src_addr, socklen_t *addrlen);
int read(int sockfd, void *buf, size_t len);
int write(int sockfd, void *buf, size_t len);
int close(int fd);
```

Socket Parameters

Parameter Name	Note
SO_BLOCKMODE	Block mode or unblock mode (Connect, Recv, Send)
SO_CONTIMEO	Connect timeout in block mode (Default: 10 seconds)
SO_SNDTIMEO	Send data timeout in block mode （Default: infinity）
SO_RCVTIMEO	Receive data timeout in block mode （Default: infinity）
SO_RDBUFLEN	Receive data buffer length in TCP socket (check mxchipWNet SRAM allocation)
SO_WRBUFLEN	Send data buffer length in TCP socket (check mxchipWNet SRAM allocation)
IP_ADD_MEMBERSHIP	Add the socket to a multicast group
IP_DROP_MEMBERSHIP	Remove the socket from a multicast group

Block/Unblock Mode in Connect

Connect

int connect(int sockfd, const struct sockaddr_t *addr, socklen_t addrlen)

Block mode

The function will return if connection is successful or time out.

Unblock mode

The function will return immediately. Library generate a callback function : “socket_connected”
If the socket connect failed, recv function based on this function will return -1.

socket connected (callback)

void socket_connected(int fd);

Return the File descriptor of the successful connected socket

Block/Unblock Mode in Send/Recv

Send data

ssize_t send(int sockfd, const void *buf, size_t len, int flags)

int write(int sockfd, void *buf, size_t len)

ssize_t sendto(int sockfd, const void *buf, size_t len, int flags,const struct sockaddr_t *dest_addr, socklen_t addrlen)

Block mode

The function will return if data are stored in socket's send buffer or timeout. Return the actual data sent to buffer.

Note: If you don't want any delay when send buffer is full, you can use int tx_buf_size(int sockfd) function to read the remaining buffer size first.

Unblock mode

The function will return immediately. Return the actual data sent to buffer.

Get useable send buffer size

int tx_buf_size(int sockfd)

Block mode

The function will return if enough data is read from socket's receive buffer or timeout, return -1 if current socket has error, and need to be closed.

Note: Use select function first to check if the socket has received any data, this may avoid the delay if no data is received.

Unblock mode

The function will return immediately.

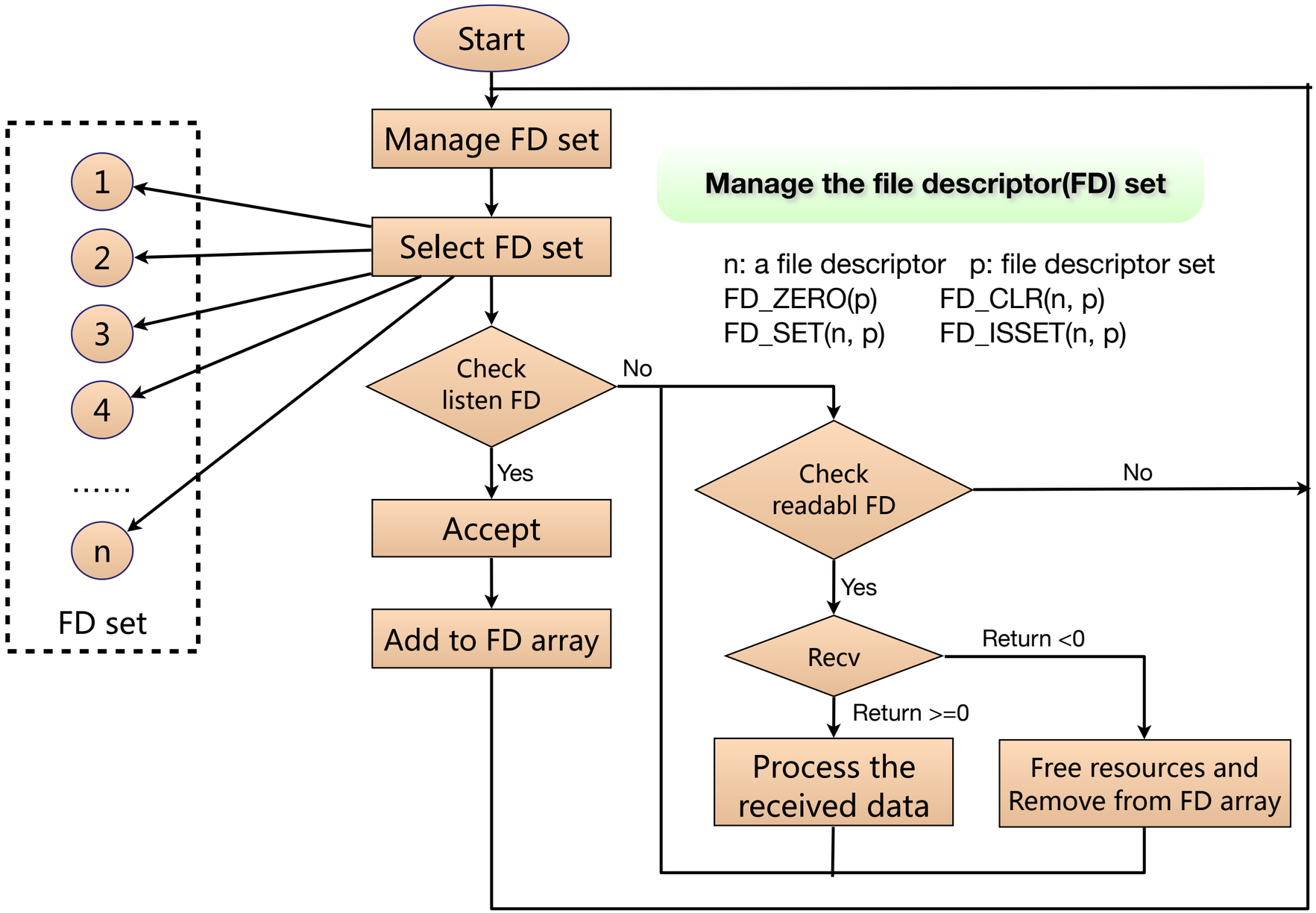
Receive data

ssize_t recv(int sockfd, void *buf, size_t len, int flags)

int read(int sockfd, void *buf, size_t len)

ssize_t recvfrom(int sockfd, void *buf, size_t len, int flags, struct sockaddr_t *src_addr, socklen_t *addrlen)

“Select” Mechanism



UDP Code Example

UDP socket creation

```
fd_udp = socket(AF_INET, SOCK_DGRM, IPPROTO_UDP);
addr.s_port = 8090;
bind(fd_udp, &addr, sizeof(addr));
```

Select and receive on the UDP socket

```
FD_ZERO(&readfds);
FD_SET(fd_udp, &readfds);

select(1, &readfds, NULL, NULL, &t);

if (FD_ISSET(fd_udp, &readfds)) {
    con = recvfrom(fd_udp, buf, 3*1024, 0, &addr, &addrLen);
    sendto(fd_udp, buf, con, 0, &addr, sizeof(struct sockaddr_t));
 }
```

TCP Client Code Example

TCP client socket creation

```
fd_client = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
addr.s_ip = inet_addr(ipstr);
addr.s_port = 80;
if (connect(fd_client, &addr, sizeof(addr))!=0) {
    close(fd_client);
    fd_client = -1;
}
```

Select and receive on the TCP client socket

```
FD_ZERO(&readfds);
if(fd_client != -1){
    FD_SET(fd_client, &readfds);
    select(1, &readfds, NULL, NULL, &t);
    if(FD_ISSET(fd_client, &readfds)){
        con = recv(fd_client, buf, 2*1024, 0);
        if(con > 0)
            printf("Get %s data successful! data length: %d bytes\r\n", WEB_SERVER, con);
        else{
            close(fd_client);
            fd_client = -1;
} } }
```

TCP Server Code Example

TCP server socket creation

```
fd_listen = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
addr.s_port = 8080;
bind(fd_listen, &addr, sizeof(addr));
listen(fd_listen, 0);
```

Select and accept on the TCP server socket

```
FD_ZERO(&readfds);
FD_SET(fd_listen, &readfds);
select(1, &readfds, NULL, NULL, &t);
if(FD_ISSET(fd_listen, &readfds)){
    j = accept(fd_listen, &addr, &len);
    if (j > 0) {
        inet_ntoa(ip_address, addr.s_ip );
        for(i=0;i<MAX_TCP_CLIENTS;i++) {
            if (clientfd[i] == -1) {
                clientfd[i] = j;
                break;
            }
        }
    }
}
```

TCP Server Code Example

Select and receive on the TCP server socket

```
FD_ZERO(&readfds);
for(i=0;i<MAX_TCP_CLIENTS;i++) {
    if (clientfd[i] != -1)
    FD_SET(clientfd[i], &readfds);
}
select(1, &readfds, NULL, NULL, &t);

for(i=0;i<MAX_TCP_CLIENTS;i++) {
    if (clientfd[i] != -1) {
        if (FD_ISSET(clientfd[i], &readfds)) {
            con = recv(clientfd[i], buf, 1*1024, 0);
        if (con >= 0)
            send(clientfd[i], buf, con, 0);
        else {
            close(clientfd[i]);
            clientfd[i] = -1;
        }
    }
}
```

TCP Keep-alive and TLS/SSL

TCP keep-alive

Send keep-alive package to maintain the TCP link if module is working as a TCP client. If it exceed num*seconds, a error will be set on the socket(RECV return a “-1”). The socket resource should be released.

void set_tcp_keepalive(int num, int seconds)

void get_tcp_keepalive(int *num, int *seconds)

TLS/SSL memory size definition

The memory size should be larger than the possible TLS/SSL encrypted data package size.

int setSslMaxlen(int len);

int getSslMaxlen(void);

Enable TLS/SSL encryption on a socket

The memory size should be larger than the largest TLS/SSL encrypted data package size.

int setSSLmode(int enable, int sockfd)

Input: A specified socket

Return: -1: Failed 0: Success

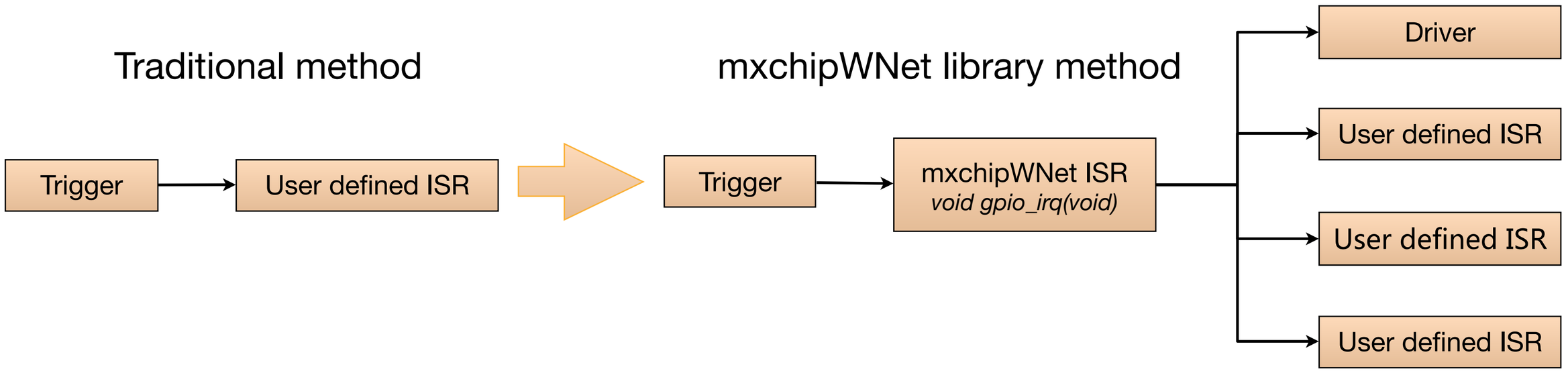
STM32 EXTI Management

New EXTI configuration function

int gpio_irq_enable(GPIO_TypeDef* gpio_port, uint8_t gpio_pin_number, gpio_irq_trigger_t trigger, gpio_irq_handler_t handler, void* arg)

int gpio_irq_disable(GPIO_TypeDef* gpio_port, uint8_t gpio_pin_number)

GPIO configuration function is not changed.



System Timer Management

A grobal variable

A system timer begins to count when library is initialized. Time unit: millisecond.

extern int MS_TIMER

Delay

These functions perform a delay, and library can still be executed during this time.

int sleep(int seconds);

int msleep(int mseconds);

Execute a function after some time

The specified function can be called by library as a callback once the predefined time has reached, recall this function in the callback can make the function execute periodically

int SetTimer(unsigned long ms, void (*psysTimerHandler)(void))

Watch dog tick (callback)

Library generate a callback once the library reach the time that need to reset the watch dog.

void WatchDog(void)

Demo 1: Wi-Fi Link

- Demonstrated library functions:
  - Scan Wi-Fi hotspots
  - Connect to a wireless router as station
  - Display the DHCP results
  - Setup a soft AP
  - coexistence mode: station mode + Soft AP mode
  - STM32 external interrupt configuration

- User operation
  - Enter a correct SSID and key in the source code
  - Build and run the demo application
  - Read the result from serial terminal
  - Press the WPS button to start a Wi-Fi scanning

UART output:*

```
Start scan
connect to MXCHIP_RD.....
Find 7 APs:
    SSID: TP-LINK_8C7150, Signal: 100%
    SSID: MXCHIP_RD, Signal: 100%
    SSID: MXCHIP_OTA_SSID, Signal: 100%
    SSID: MXCHIP-test, Signal: 67%
    SSID: DING9F, Signal: 67%
    SSID: MXCHIP_Guest, Signal: 60%
    SSID: MXCHIP_PD, Signal: 52%
IP address: 192.168.2.184
NetMask address: 255.255.255.0
Gateway address: 192.168.2.1
DNS server address: 192.168.2.1
MAC address: c893460001cd
Wi-Fi up
Setup soft AP: uAP
Start scanning by user...
Find 8 APs:
    SSID: TP-LINK_8C7150, Signal: 100%
    SSID: MXCHIP_RD, Signal: 100%
    SSID: MXCHIP_OTA_SSID, Signal: 100%
    SSID: MXCHIP-test, Signal: 75%
    SSID: MXCHIP_PD, Signal: 70%
    SSID: DING9F, Signal: 65%
    SSID: MXCHIP_Guest, Signal: 35%
    SSID: QX-3F, Signal: 17%
```

* Output maybe vary depends on actual operation

Demo 2: TCP UDP Echo

- Demonstrated library functions:
  - Connect to a wireless router as station
  - Work as a http client, and read content of a web page
  - Work as a TCP server to accept client connection
  - Open a UDP port
  - Perform echo function on every socket connections

- User operation
  - Enter a correct SSID and key in the source code
  - Build and run the demo application
  - Read the result from serial terminal
  - Setup a TCP client to connect module on port 8080
  - Setup a UDP client to connect module on port 8090

UART output*:

```
Connect to MXCHIP_RD.....
IP address: 192.168.2.184
NetMask address: 255.255.255.0
Gateway address: 192.168.2.1
DNS server address: 192.168.2.1
MAC address: c893460001cd
Station up
DNS test: www.baidu.com address is 61.135.169.125
Connect to web server success! Reading web pages...
Get www.baidu.com data successful! data length: 1616 bytes
Get www.baidu.com data successful! data length: 1024 bytes
Get www.baidu.com data successful! data length: 1024 bytes
Get www.baidu.com data successful! data length: 1024 bytes
Get www.baidu.com data successful! data length: 2048 bytes
Get www.baidu.com data successful! data length: 2048 bytes
Get www.baidu.com data successful! data length: 1024 bytes
Get www.baidu.com data successful! data length: 1024 bytes
Get www.baidu.com data successful! data length: 1024 bytes
Get www.baidu.com data successful! data length: 1024 bytes
Get www.baidu.com data successful! data length: 1024 bytes
Get www.baidu.com data successful! data length: 1024 bytes
Get www.baidu.com data successful! data length: 1082 bytes
Client 192.168.2.172:49483 connected
Web connection closed.
```

* Output maybe vary depends on actual operation

Demo 3: WPS EasyLink

- Demonstrated library functions:
  - Connect to a wireless router as station
  - WPS configuration
  - Easylink configuration

- User operation
  - Build and run the demo application
  - Select WPS or EasyLink from on serial terminal
  - When using WPS, press WPS button on the router
  - When using EasyLink, use the Easylink APP on iOS/ Android

UART output*:

```
+***************(C) COPYRIGHT 2013 MXCHIP corporation************+
|         EMW316x WPS and EasyLink configuration demo            |
+ command ----------------+ function ----_-----------------------+
| 1:WPS                  | WiFi Protected Setup                  |
| 2:EasyLink             | One step configuration from MXCHIP    |
| 3:REBOOT               | Reboot                                |
| ?:HELP                 | displays this help                    |
+------------------------+---------------------------------------+
|                        By William Xu from MXCHIP M2M Team      |
+----------------------------------------------------------------+

MXCHIP> 1

WPS started....., press WPS button on your AP
Configuration is successful, SSID:MXCHIP_OTA_SSID, Key:
connect to MXCHIP_OTA_SSID.....

MXCHIP> IP address: 192.168.1.101
NetMask address: 255.255.255.0
Gateway address: 192.168.1.1
DNS server address: 192.168.1.1
MAC address: c893460001cd
Station up

MXCHIP>
```

* Output maybe vary depends on actual operation

Demo 4: Web Server OTA

- Demonstrated library functions:
  - Setup a soft AP
  - Build-in web server
  - Update the firmware using OTA on the web page
  - Read and write the temporary parameters

- User operation
  - Build and run the demo application
  - Connect to the soft AP established by module
  - Open the web page on 192.168.1.1 using a browser

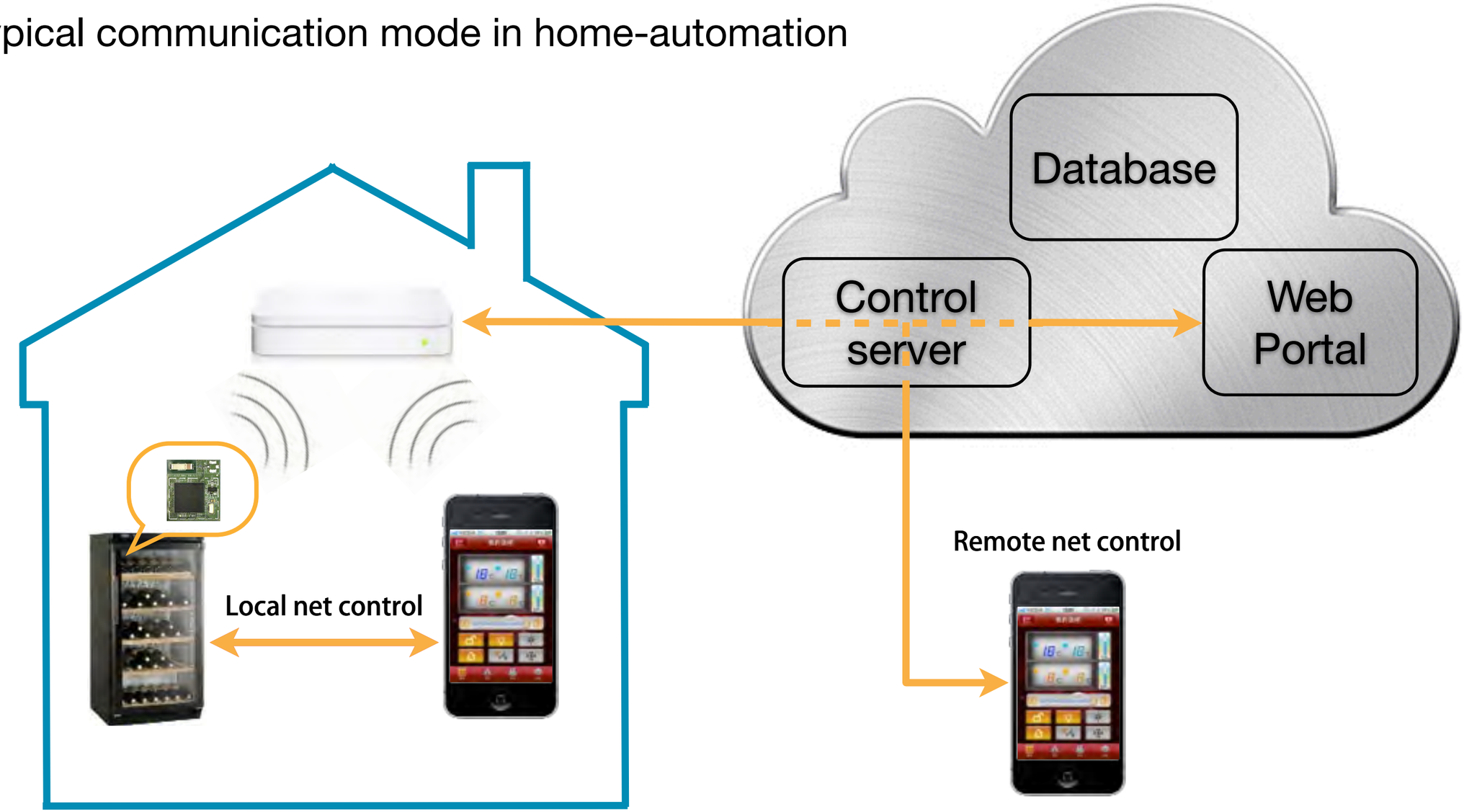
UART output:

```
MXCHIP> Establish soft AP: MXCHIP_0001cd.....
```

Build-in web page

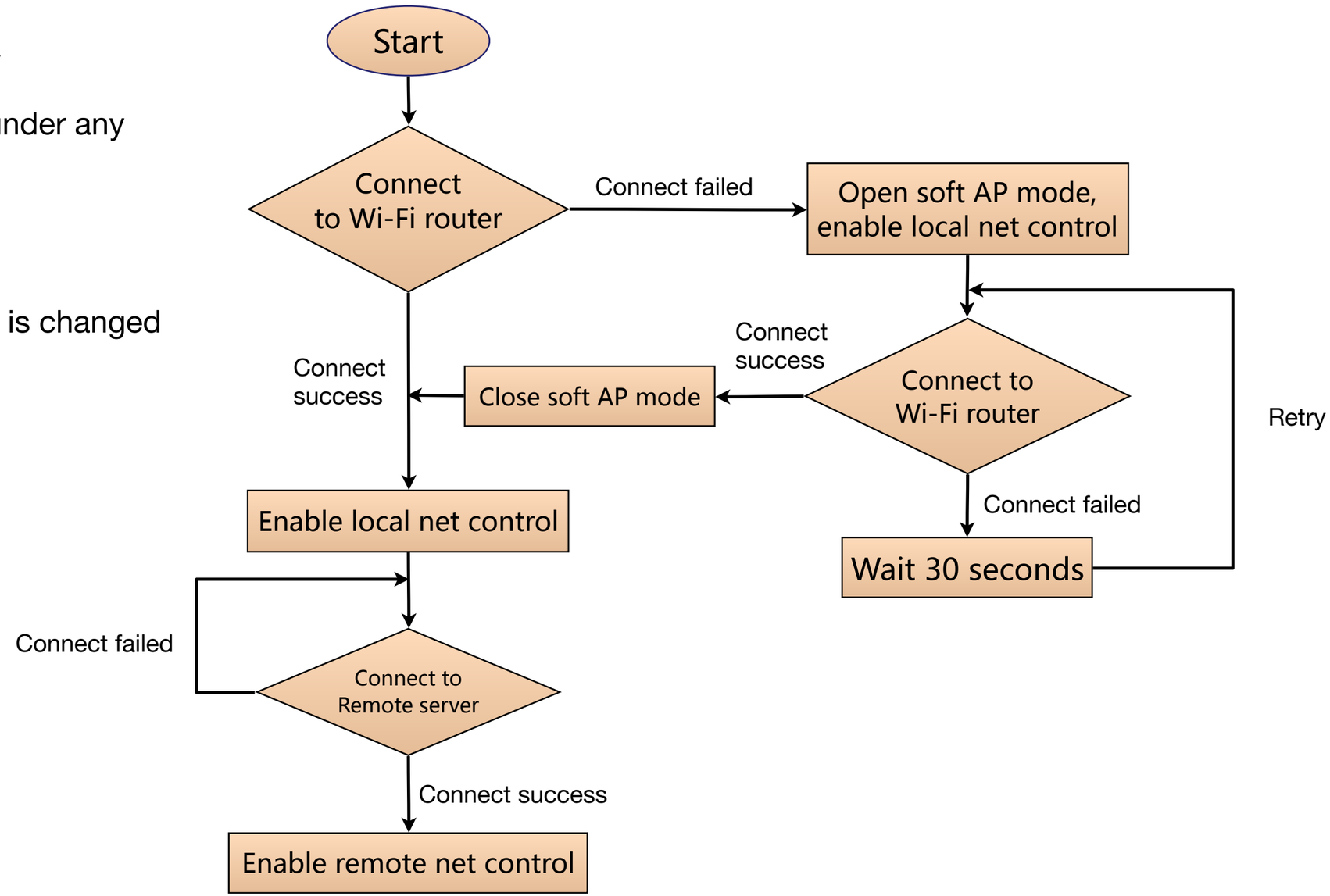
Application Demo: mxchipWNet-HA

- Typical communication mode in home-automation



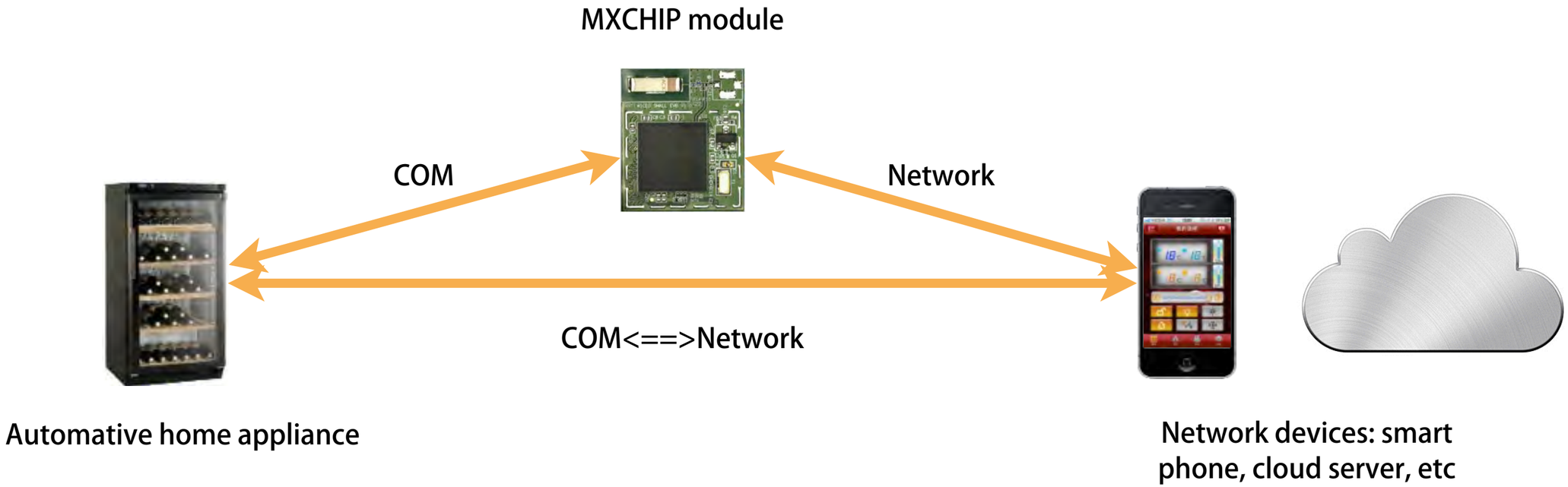
Application Demo: mxchipWNet-HA

- Auto Wireless connectivity
  - Wireless connectivity is available under any circumstance
  - Automatic switching
  - Low power under station mode
  - Report to host device when status is changed



Application Demo: mxchipWNet-HA

- Data directions
  - COM: Read Wi-Fi status, open/close Wi-Fi, start WPS configuration…
  - Network: OTA, configure module from APP, cloud service login …
  - COM<==>Network: Remote control…



Application Demo: mxchipWNet-HA

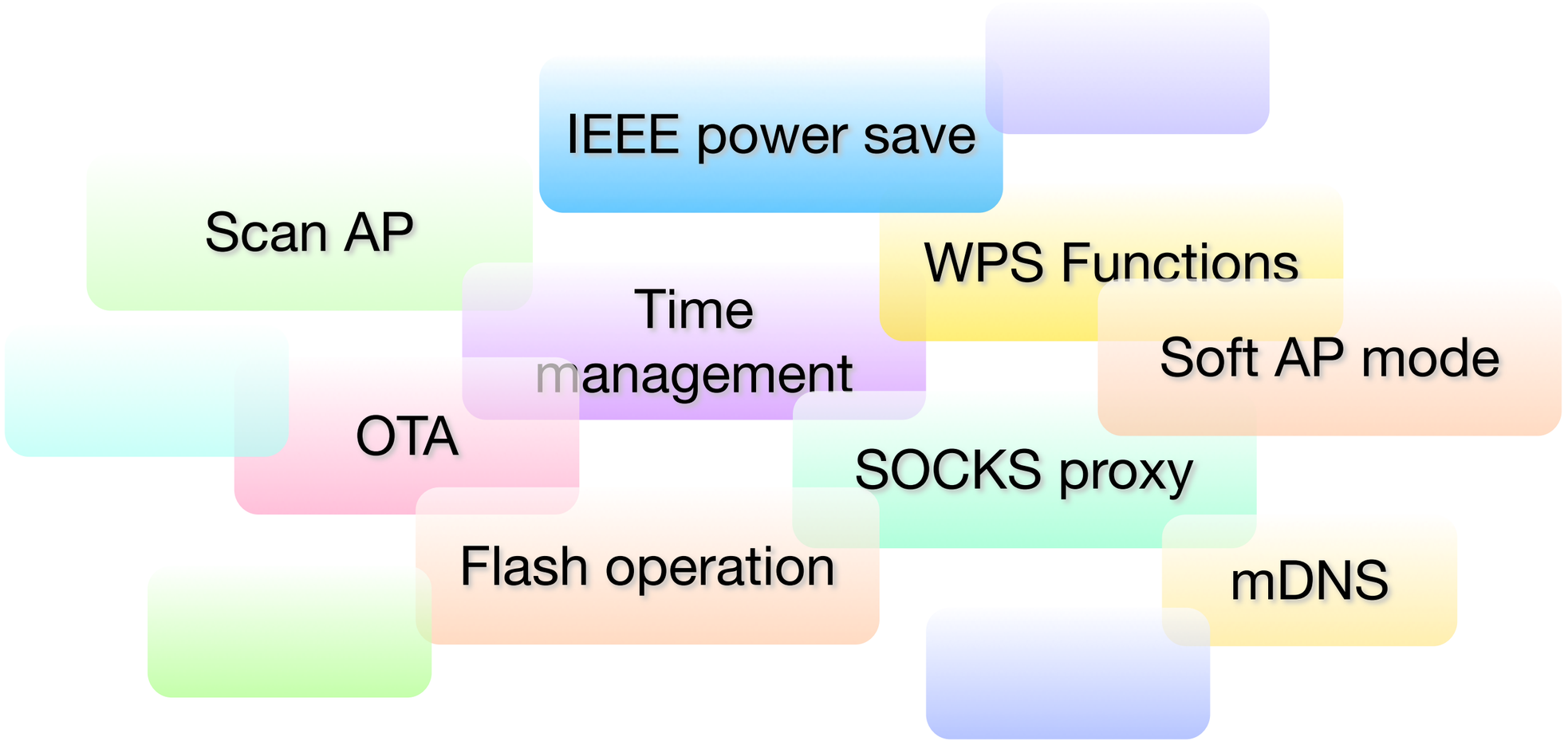
- Command formats

Heade	Command code	Return	Data length	Data	Checksum
BB 00	U16	U16	U16	U8[length]	U16

- Typical commands

Name	Code	Data direction	Function
CMD_READ_VERSION	1	Network	Read firmware version
CMD_READ_CONFIG	2	Network	Read firmware configuration
CMD_WRITE_CONFIG	3	Network	Write firmware configuration
CMD_SCAN	4	Network	Scan available hotspots
CMD_OTA	5	Network	OTA
CMD_NET2COM	6	Netowrk->COM	Send COM data to network
CMD_COM2NET	7	COM->Network	Send network data to COM
CMD_GET_STATUS	8	UART	Read module’s current status
CMD_CONTROL	9	UART	Module control
CMD_SEARCH	10	Network（UDP）	Module discovery

mxchipWNet™ API overview



THE END

Make wireless connections
Simple

Thank you!